Számítástudomány jegyzet

Készítették Grolmusz Vince előadásai alapján a 2021. évi hallgatók

(Nem hivatalos, lektorálatlan verzió)

2021. tavasz

Tartalomjegyzék

Vizsgatematika

1. 32 bites szám kitalálása barkochbával
2. A számítógépek egy absztrakt modellje: A Turing-gép. Nyelvek, abc. Példák Turing-gépekre. A palindrómák felismerése egy- és kétszalagos Turing-géppel.
3. Az univerzális Turing-gép definíciója és létezése.
4. Church tézis.
5. A k-szalagos Turing gép szimulálható 1 szalagossal $O(N^2 2)$ időben.
6. A RAM-gép.
7. A RAM-gép és a Turing-gép ekvivalenciája: a Turing-gép szimulálása RAM-gépen. A RAM-gép szimulálása Turing gépen.
8. Rekurzív és rekurzíve felsorolható nyelvek, ezek alapvető tulajdonságai. Majdnem minden nyelv nemrekurzív, Minden véges nyelv rekurzív.
9. Majdnem minden nyelv nem rekurzíve felsorolható.

10. A megállási probléma.
11. A Dominó-probléma. L_{NEMRAK} rekurzíve felsorolható,
12. Algoritmikusan eldönthetetlen, hogy egy leírásával adott Turing-gép az üres inputon leáll-e.
13. L_{KIRAK} nem rekurzíve felsorolható.
14. Idő-és tárkorlátos nyelvosztályok. DSPACE(f(n)), DTIME(f(n)), P, PSPACE.
15. Egyszerű relációk a tárigény és az időigény között.
16. Példák polinomiális idejű algoritmusokra: euklideszi algoritmus; $a^b \mod m$.
17. Lineáris gyorsítási tétel.
18. Minden rekurzív f függvényhez létezik olyan rekurzív nyelv, amely $\notin DTIME(f(n))$ (azaz tetszőlegesen nehéz rekurzív nyelv van). A nem-determinisztikus Turing-gép. Nem-determinisztikus nyelvosztályok.
19. NP, co-NP definíciói.
20. Tanú fogalma. Példák polinomiális tanúra. Az NP nyelvosztály jellemzése tanúkkal.
21. Pratt tétele. Számok faktorizációja, A FACTOR nyelv. Nyelvek polinomiális visszavezetése, tulajdonságai. NP-teljesség definíciója.
22. Boole függvények, Boole polinomok, CNF formulák, kielégíthetőség. A SAT nyelv.
23. Cook tétele: A SAT NP-teljes.
24. Halmazrendszer kétszínezhetősége NP-teljes. A gráfok háromszínezhetõsége NP-teljes. Minden háromnál nagyobb k-ra a gráfok k-színezhetősége NP-teljes. Az INDEPENDENT nyelv NP-teljessége.
25. Az $INDEPENDENT_k$ nyelv P-ben van, minden k-ra.
26. A SAT3, a LEFOG, a LEFED NP-teljes.
27. A K-PART, PART NP-teljes. A $SUBSET-SUM$ és a HÁTIZSÁK feladat NP-teljessége.
28. Egy algoritmus a hátizsák feladat megoldására. Az algoritmus lépésszámbecslése. Ibarra és Kim skálázási eljárása közelítő optimum keresésére.
29. Gráf kromatikus számának közelítése.
30. Véletlen algoritmusok, polinomazonosság ellenőrzése, Schwartz lemma.
31. Alkalmazás páros gráfban teljes párosítás létének eldöntésére.
32. Prímtesztelő eljárások: egy egyszerű módszer, amely nem mindig működik, és a Rabin-Miller teszt.
33. Kriptográfia: titkos- és nyilvános kulcs. OTP : One Time Pad.
34. Titkos kulcs-csere protokoll.
35. Az RSA titkosírás, digitális aláírás.
36. Titokmegosztás: egy optikai és egy algebrai megoldás.
37. Kommunikációs játékok. A Mehlhorn-Schmidt tétel.
38. A nem-determinisztikus kommunikációs komplexitás jellemzése fedő téglalapokkal.
39. A Simon-Rabin protokoll az ID függvény gyors, randomizált kiszámítására.
40. Példák nemtriviális protokollokra: részfa-metszet, teljes-üres részgráf metszet. Az Aho-Ullman-Yannakakis tétel; a P, NP, co-NP analógiái kommunikációs bonyolultságra, ezek viszonya.

1. Alapvető fogalmak

1.1. Bevezetés

1.1.1. *példa* (32 bites barkochba). Adott egy $x \in \{0,1\}^{32}$ bitsorozat, szeretnénk minél kevesebb kérdésből kitalálni.

1.1.2. *megjegyzés.* A 32 bites barkochba megoldható 32 kérdésből: minden kérdésben megkérdezzük, hogy x adott bitje 0-e.

1.1.3. állítás. *Nincsen olyan eljárás, amely a 32 bites barkochbát megoldja legfeljebb 31 kérdéssel.*

Bizonyítás. Tegyük fel, hogy van egy ilyen eljárás. Ekkor minden x-hez kapunk egy 31 hosszú igen–nem sorozatot. Egy ilyen válaszsorozatból egyértelműen ki kell, hogy tudjuk találni x-et, de ekkor legfeljebb 2^{31} lehetséges értékét tudjuk lefedni az x-nek.

1.1.4. definíció (Turing-gép)**.** Egy $T = (k, \Sigma, \Gamma, \alpha, \beta, \gamma)$ hatos, ahol

- $k \in \mathbb{N}$,
- $3 \leq |\Sigma| < \infty$, $* \in \Sigma$,
- $|\Gamma| < \infty$, $\mathrm{START}, \mathrm{STOP} \in \Gamma$,
- $\alpha : \Sigma^k \times \Gamma \to \Gamma$,
- $\beta : \Sigma^k \times \Gamma \to \Sigma^k$,
- $\gamma : \Sigma^k \times \Gamma \to (-1, 0, 1)^k$.

Kezdetben T a START állapotban van, az első kivételével minden szalagon minden mezőben $*$ áll, az első szalagon a fej alatt kezdődik az input és attól jobbra folytatódik. (Az inputnak az első csillagnál van vége.) Tetszőleges $h \in \Sigma^k$ esetén $\alpha(h, \mathrm{STOP}) = \mathrm{STOP}, \beta(h, \mathrm{STOP}) = h, \gamma(h, \mathrm{STOP}) = (0, 0, .., 0)$. Az output az utolsó szalag tartalma megálláskor.

1.1.5. *megjegyzés.* k a **szalagok száma**, Σ egy **ábécé**, $*$ az üres karakter, Γ az **állapothalmaz**, α, β és γ **átmenetfüggvények**.

Az átmenetfüggvények azt mondják meg, hogy ha adott állapotban olvasunk k karaktert, akkor mi történik. α megmondja, hogy mi lesz az új állapot, β azt, hogy mit kell írni, és γ azt, merre mozognak a fejek.

1.1.6. definíció. Σ^* a Σ betűiből képzett véges sorozatok halmaza, $\Sigma_0 = \Sigma \setminus \{*\}$

1.1.7. definíció (Σ feletti nyelv)**.** Egy $L \subseteq \Sigma_0^*$ halmaz.

1.1.8. definíció (sztenderd probléma)**.** Adott egy Σ ábécé, $L \subseteq \Sigma_0^*$ nyelv és egy $w \in \Sigma_0^*$ szó. A sztenderd probléma az a kérdés, hogy w eleme-e L-nek.

1.1.9. definíció. T **felismeri** az $L \subseteq \Sigma_0^*$ nyelvet, ha T a Σ_0^* minden elemén megáll és $w \in L$ esetén 1-et, $w \in \Sigma_0^* \setminus L$ esetén 0-t ad.

1.1.10. *példa.* Legyen $L = \{\{01\}^\ell : \ell \in \mathbb{N}\}$ (itt $\min \mathbb{N} = 1$). Legyen $k = 2$, $\Sigma = \{*, 0, 1\}$, $\Gamma = \{\text{START}, \text{STOP}, \text{N}, \text{NE}\}$.

- Ha $u \neq \text{STOP}$, $\gamma(h, u) = (1, 0)$,
- $\alpha(0, \cdot, \text{START}) = \text{N}, \alpha(1/*, \cdot, \text{START}) = \text{STOP}$,
- $\alpha(0/*, \cdot, \text{N}) = \text{STOP}$, $\alpha(1, \cdot, \text{N}) = \text{NE}$,
- $\alpha(0, \cdot, \text{NE}) = \text{N}, \alpha(1/*, \cdot, \text{NE}) = \text{STOP}$.
- $\beta(j, *, \text{NE}) = (j, 1)$, β minden más értéke $(j, 0)$.

Ekkor T felismeri L-t.

1.1.11. definíció (palindromnyelv)**.** Azon szavak halmaza, amelyek balról jobbra és jobbról balra olvasva ugyanazt adják.

1.1.12. *példa.* Vegyük a következő kétszalagos gépet: amikor az első fej talál egy szót, akkor elkezd jobbra mozogni, egyenként beolvassa a betűket, a második fej pedig lemásolja a második szalagra. Majd a második fejet visszavisszük a szó elejére (az első csillagnál megáll), majd az első fej balra, a második fej jobbra megy, és akkor áll meg a gép 1-gyel, ha nem állt meg még 0-val, és egyszerre olvas mindkét fej $*$-ot, és 0-val, ha különböző betűt olvas a két fej. Ez pontosan a palindromszavakon áll meg.

1.1.13. *megjegyzés.* A fenti gép egy n hosszú szót $O(n)$ lépésből ismer fel. (**Lineáris** algoritmus.)

1.1.14. *példa.* Mi van, ha egyszalagos géppel akarjuk felismerni a palindromszavakat? Beolvassa a gép az első karaktert, megjegyzi (olyan állapotot vesz fel, amiből vissza lehet következtetni, hogy mit olvasott), majd kitörli. Aztán elmegy az input végéig, összehasonlítja az utolsó karaktert a megjegyzettel, és megáll 0-val, ha nem ugyanaz. Ha ugyanaz, akkor eggyel balra megy, megjegyzi, hogy mit olvasott, kitörli, elmegy a másik végére, stb. Ha mindenhol $*$ van, akkor 1-el megáll.

1.1.15. *megjegyzés.* Ez egy $O(n^2)$ algoritmus.

1.2. Turing-gép

1.2.1. definíció. T $k+1$ szalagos Turing-gép a Σ ábécé felett **szimulálja** a Σ feletti S k szalagos Turing-gépet a $p \in \Sigma_0^*$ programmal, ha a T $k+1$. szalagjára p-t írva tetszőleges $w \in \Sigma_0^*$ inputra T és S ugyanakkor áll meg vagy nem áll meg w-n, és megálláskor a T első k szalagján ugyanaz van, mint S-en.

1.2.2. definíció ($k+1$ szalagos univerzális Turing-gép Σ felett)**.** Olyan T, amelyre minden S k szalagos Σ feletti Turing-géphez létezik $p \in \Sigma_0^*$ program, hogy T szimulálja S-t p-vel.

1.2.3. tétel. *Minden $k \in \mathbb{N}$ és minden Σ ábécéhez létezik Σ feletti $k+1$ szalagos univerzális Turing gép.*

Bizonyítás. Először: $k+2$ szalagra.

Első k szalag: a szimulálandó S Turing gép k szalagjának a tartalma lesz minden szimulálandó lépés után. $S = (k, \Sigma, \Gamma_s, \alpha_s, \beta_s, \gamma_s)$.

$k+1$. szalag: p program. $p = \mathrm{CONCAT} ug\alpha_s(u,g), \beta_s(u,g), \gamma_s(u,g)$, ahol $u \in \Sigma^k, g \in \Gamma_s$ (konkatenáció: egymás után fűzés). p sok mezőt használ. Előre meghatározott módon elkódoljuk a Σ elemeivel az állapotokat, illetve a mozgásokat.

$k+2$. szalag: $g \in \Gamma_s$

Működés:

- 1 lépésben elolvassa a fejek alatti mezők tartalmát.
- Szimulációs lépés: az olvasott betűk és g függvényében megnézi, hogy mi $\alpha_s, \beta_s, \gamma_s$ értéke.
- Megnézés: p-n elkezd menni a fej, amíg olyan u-val kezdődő szót talál, amiben pontosan ugyanazok a betűk vannak mint az első k szalagon "fentről lefelé" ezután megnézi, hogy a $k+2$. szalagon lévő g érték megfelel-e az u után konkatenált g-nek, ha nem, megy tovább, ha megfelel, akkor a $k+1$. szalagon az ug után konkatenálva lévő $\alpha_s(u,g)$-t átmásolja a $k+2$. szalagra, majd betűnként a $\beta_s(u,g)$-t az első k szalagra, majd a választott kódolás szerint $\gamma_s(u,g)$-t olvasva mozgatja az első k fejet jobbra/balra tehát minden lépésben legfeljebb egyszer olvassuk a $k+1$. szalagon lévő p szót.

Ezzel a $k+2$ szalagos verzióval kész vagyunk.

Gond: Pazarlás, hogy a g elfoglal egy egész szalagot, de mivel nem tudjuk, hány állapota van S-nek (ez véges sok, de nem korlátos sok), így nem tudjuk megcsinálni, hogy pl. kijelölünk 10 pozíciót neki a $k+1$. szalagon.

$k+1$ szalagra: úgy csinálunk, mintha egy szalagon két fej lenne, ekkor ugyanis középen egy elválasztó egyik irányában lehetne a p egy fejjel, a másik irányban pedig g egy másik fejjel. Egy fejjel fogunk szimulálni kettőt. Minden páratlanadik mező megfelel az eredetinek, párosadikban pedig vagy * vagy 1 szerepeljen. *, ha nem a tőle jobbra eső mezőben van a "második" szimulált fej, 1 ha ott van épp. Így szimuláltunk két fejet egy szalagon, ezzel befejezve a tétel bizonyítását. □

1.2.4. tétel. *Minden k szalagos S Turing géphez létezik egy 1 szalagos T Turing gép, amely az S-et szimulálja az alábbi értelemben:*

- $\forall w \in \Sigma_0^*$ *inputra T leáll* $\iff$ *S leáll w-n*
- *leálláskor az S k. szalagján ugyanaz van, mint T szalagján*
- *ha S leállásig t lépést tesz meg, akkor T leállásig $O(t^2)$ lépést tesz meg*

1.2.5. *megjegyzés.* Ez nem univerzális, azaz csak egy konkrét S-re garantálja, hogy működik, nem az összes k szalagosra.

Bizonyítás. Ötlet: Ha mindkét irányban végtelen az az egy szalag, akkor felhullámoztatjuk a szalagot (csak a felfele részt nézzük, hogy mod k nézhessük), k emelet magas hullámokra, ahol a különböző emeletek felelnek meg S különböző szalagjainak.

Legyen inkább $2k$ magas a hullám, T tud számolni modulo $2k$. Ekkor S első szalagja megfelel az első emeletnek, a harmadik emelet a második szalagnak, az ötödik emelet a harmadik szalagnak, stb. A páros emeletek segédfunkciót fognak ellátni. Ha lenne k fej, akkor könnyű lenne (csak pl 1 jobbra lépés helyett $2k$-t kellene lépni), de nincs, így az előző bizonyításhoz hasonló módszert alkalmazunk. A segédmezőkön $*$ vagy 1-es legyen: 1-es, ha fölötte állna az egyik fej, amennyiben k lenne belőle. T tudja mod $2k$, hogy hány lépést tett meg, azaz melyik emeleten van (azt nem tudja számolni a belső állapotaiban, hogy hány lépést tett meg, mert véges sok állapota van).

Szimuláció: Valamilyen jelet tesz az eddig használt szalagrész elejére és végére (amit később odébb tehet).

- Olvasás: Elindul a szalag egyik végéről (ahol ő már járt), és addig megy, míg meg nem találja a másik jelet, és közben a páros mezőkön keresi az 1-eseket, és ha talál, akkor megnézi, hogy fölötte mi van, így mire végigért minden emeletről pontosan egy betűt fog elolvasni.
- Írás: Tudja, hogy hol állna mind a k szimulált fej, magában tudja, hogy milyen állapota van a g_s-nek, ez alapján beírja a megfelelő betűket a megfelelő helyekre úgy, hogy megint végigmegy az eddig használt megjelölt szalagrészen.
- Elmozdulás/Fejmozgatás: Tudja (olvasás, g_s alapján), hogy merre kell elmozdulni minden szalagon, a régit kicsillagozza, az új helyet megkeresi, beírja az 1-est.

Gusztustalan rész: hogy kerül a helyére az input és az output. Motiváció: S-ben input az 1. szalagon, T-ben definíció szerint a szalag elején, vagyis ezt át kell vinni az első emeletre megfelelő sorrendben. Helyrerakás: az 1. helyen jó van. A 2. helytől a végéig tartó szakaszt odébb másolja $2k-1$-gyel, majd mindig a következő betűtől csinálja ugyanezt, míg a végére nem ér (mindig a végén kezdi, hogy ne vesszen el adat). Ha u hosszú az input, akkor ez $O(u^2)$ lépés alatt kész van. Az output az S utolsó szalagján van, hasonlóan rendezhető át T-n.

Szimuláció ideje: S t lépést tett meg, akkor legfeljebb $k \cdot t$ mezőt látogattak meg a fejek, így T is $O(t)$ mezőt használ, $O(t^2)$ lépéssel tudjuk szimulálni az S-et. □

1.2.6. *megjegyzés.* Church-tézis: minden, ami kiszámolható, az Turing géppel is kiszámolható.

1.2.7. definíció (RAM: Random Access Machine)**.** A RAM-gép áll egy végtelen memóriából, amelynek celláit $X[i]$-vel jelöljük, $i \in \mathbb{Z}$. $\forall i$-n $X[i]$-be tetszőleges $\mathbb{Z}$-beli szám írható. A RAM-gép ezen kívül programtárat tartalmaz, amelybe programsorok írhatók be. A következő programsorok megengedettek:

- $X[i] := 0$;
- $X[i] := 1$;
- $X[i] := X[i] \pm X[j]$;
- $X[i] := X[i] \pm 1$;
- $X[i] := X[j]$
- $X[i] := X[X[j]]$
- $X[X[i]] := X[j]$
- IF $X[i] \leq 0$ THEN GOTO [programsor címe]

A RAM-gép a végrehajtást az első programsorral kezdi. A programsorok számozva vannak 1-től $r \in \mathbb{N}$-ig. Ha egy programsor nem ugrást ír elő, akkor a végrehajtást a rákövetkező programsoron kell folytatni. A RAM-gép leáll, ha a végrehajtás üres programsorra érkezik. A RAM-gép outputja a véges memória tartalma, ha a gép leáll. Kezdetben $\forall i : X[i] = 0$.

1.2.8. *megjegyzés.* Szorzás, osztás, sinus, cosinus, stb nincs, a RAM-gép csak aritmetikát tud csinálni, abból is csak primitívet.

1.2.9. *példa* (Element Distinctness). $ED(X_1, X_2, \ldots, X_n)$
$ED(X) = 1$, ha $\forall i \neq j : x_i \neq x_j$, egyébként $ED(X) = 0$.
Ha egyenként hasonlítom össze az elemeket: $\binom{n}{2}$, azaz $O(n^2)$.
Ha sorbarendezem az elemeket: $O(n \cdot \ln n)$.
RAM-géppel:

- FOR $i = 1$ TO n LET $X[X[i]] := i$ NEXT i
- ellenőrizzük, hogy $\forall i : X[X[i]] = i$: minden esetben igaz $\iff ED(X) = 1$

Most megmutatjuk a két múlt óráról megmaradt tételünket, a RAM, és Turing gép ekvivalenciáját.

1.2.10. tétel. *Minden $\Sigma = \{0, 1, 2\}$ fölötti egy szalagos $(\Sigma, \Gamma, \alpha, \beta, \gamma)$ Turing-géphez van olyan program a RAM-gépen, hogy minden $w \in \Sigma_0^*$ inputra a Turing-gép megáll akkor és csak akkor ha a RAM-gép megáll, megálláskor az output ugyanaz, és ha a Turing-gép megállásig t lépést tesz meg, akkor a RAM-gép $\mathcal{O}(t)$ programsort hajt végre megállásig.*

1.2.11. *megjegyzés.* A fenti állításban a Turing gép * karakterét és a 0-t azonosítsuk. Turing-gépben minden cella *-al van inicializálva, RAM-gépben pedig 0-val, ezért ez egy természetes feltétel. Továbbá feleltessük meg a Turing-gép állapotait az $\{1, 2, 3, \ldots, r\}$ halmaznak, ahol az 1 a START-nak, az r pedig a STOP-nak felel meg.

Bizonyítás. Programblokkokat fogunk írni, először a $P_i \quad (i \in \Gamma)$ blokk azt fogja szimulálni, hogy a turing gép az i állapotban olvas. A páratlan indexű RAM-gép cellákat megtartjuk adminisztrációra, a Turing-gép u. mezője meg lesz feleltetve az $X[2u]$ memóriahelynek a RAM-gépben. $X[1]$ fogja tárolni a Turing-gép fejének a pozícióját, $X[3]$ segédmező. Többszalagos gép esetében annyi segédcella kell ahány fej van, és akkor $X[uk]$-ban lesz az u. szalagpozíció eltárolva. A P_i programszegmens a következő legyen:

$$X[3] := X[X[1]]$$

$$IF \quad X[3] \leq 0 \quad THEN \quad GOTO \quad [Q_{i0}cime]$$

$$X[3] := X[3] - 1$$

$$IF \quad X[3] \leq 0 \quad THEN \quad GOTO \quad [Q_{i1}cime]$$

$$X[3] := X[3] - 1$$

$$IF \quad X[3] \leq 0 \quad THEN \quad GOTO \quad [Q_{i3}cime]$$

Idáig tart a P_i programszegmens megadása. Szavakkal: megnézzük a fej jelenlegi pozícióját, ami az 1-es regiszterben van eltárolva, azt átmásoljuk az adminisztratív cellánkba, majd esetszét-választunk aszerint, hogy mennyit olvasott a fej. A Q_{ij} programszegmenseket most megadjuk, ezekkel kódoljuk el az írást, és a fejek mozgását. Először az írás, $Q_{ij} := \{$

$$X[3] := 0$$

$$X[3] := X[3] + \beta(i, j)$$

A fenti sorban kétszer adunk hozzá egyet, ha β értéke kettő.

$$X[X[1]] := X[3]$$

Ezzel oda is írtuk a fej pozíciójának megfelelő helyre, most a fej mozgatását szimuláljuk,

$$X[1] := X[1] + \gamma(i, j)$$

$$X[1] := X[1] + \gamma(i, j)$$

Kétszer írtuk le, mert csak a párosadik helyeken tárolunk szalagértékeket. Végül az állapotváltozást szimuláljuk.

$$X[3] := 0$$

$$IF \quad X[3] \leq 0 \quad THEN \quad GOTO \quad [P_{\alpha(i,j)}cime]\}$$

Ezt azért csináljuk csak így, mert nincs feltétel nélküli ugróutasítás, de ez a sor mindig lefut hiszen a feltételt előtte igazzá tettük. Ezzel Q_{ij} leírását, és a bizonyítást is befejeztük. □

1.2.12. *megjegyzés.* Megszámolva a sorokat egy Turing-gép lépést legfeljebb 14 sor RAM-gép kóddal tudunk szimulálni, a konstrukció nyilvánvalóan általánosodik tetszőleges sok fejre, és ábécé méretre.

A következőkben belátjuk a másik irányt, mindent amit RAM-géppel számolhatunk, Turing-géppel is lehet. A bizonyítás vázlatos lesz, hiszen problémás Turing-gépet *korrektül* megadni. Akadályt jelent, hogy a RAM-gép tetszőlegesen sokat léphet a memóriájában egy lépés alatt, míg a Turing-gép csak "egyesével" tud mozogni.

1.2.13. definíció. A RAM-gép egy programjának futási ideje (*logaritmikus költsége*) $u \in \mathbb{N}$, ahol u-t úgy kapjuk, hogy minden végrehajtott programsorra összegezzük a benne szereplő számok bináris hosszát.

1.2.14. *példa.* Az $X[4] := X[4] + 1$ kódsornak $2\lceil \log_2(X[4]+1) \rceil + 7$ a költsége, kétszer szerepel benne $X[4]$, amit $2\lceil \log_2(X[4]+1) \rceil$ darab bittel lehet leírni, továbbá mivel $4_{10} = 100_2$ ezért még $3 + 3 + 1$ a regiszterek indexének a hossza.

1.2.15. tétel. *Minden RAM-gép programhoz van egy olyan T Turing-gép amely ugyanazt számolja ki mint a RAM-gép (ugyanaz az output, és ugyanakkor áll meg), és a RAM-gép működésének logaritmikus költsége u, akkor leállásig T $\mathcal{O}(u^2)$ lépést tesz meg.*

Mért van szükség a logaritmikus költségre? Vegyük észre, hogy RAM-gép egy utasítással tud másolni bármilyen nagy értéket ($X[1] := X[3]$), nem lenne fair Turing-géppel szemben, melynek el kell mozgatni a fejet, és mindez lépésekbe kerül. Még így is meglepő a tétel, hiszen nem lineáris, hanem logaritmikus költséggel a lépéseken megtehető ez.

Bizonyítás vázlat. A RAM-gépnek van valahány programsora. Készítünk annyi Turing-gépet, ahány programsor van, és ezeket összeragasztgatjuk. Mivel az instrukciók nagyon egyszerűek, majdnem triviálisan átírhatjuk őket Turing-gépekké, a nehéz kérdés a cellák közötti áttérés. Evégett fenntartunk egy extra szalagot "adattárnak". Egy $X[i] := j$ utasításra ráírjuk a szalagra hogy $*j**i$, ha utána van egy sor $X[s] := t$, akkor utánafűzzük $*j**i*t**s$, majd ha $X[i] := z$ következik akkor $*j**i*t**s*z**i$ tovább konkatenálunk. Ezen a szalagon sosem törlünk, ha egyszer egy már meglévő cella tartalmára van szükségünk, pl $X[l] := X[i]$, akkor elindulunk jobbról ezen az adatszallagon, és amint először $*i**z$-t olvasunk (jobbról!) akkor megkaptuk a legfrissebb értékét az i. cellának. Mivel binárisan kódoljuk az értékeket, ez a szallag legfeljebb $\mathcal{O}(u)$ hosszú, legfeljebb 1-szer kell minden utasításban végigolvasni, szintén $\mathcal{O}(u)$ darab utasítás van, az elemi utasítások lineáris időben szimulálhatóak, tehát összességében $\mathcal{O}(u^2)$ hosszú a futásidő. □

Ezzel befejeztük a RAM-gépek tárgyalását. Mostmár mindenki elhiszi, hogy a Turing-gép az egy jó modell a számítógépek absztrachálására:).

1.2.16. *megjegyzés.* Polinomiális Church-tézis Minden értelmes számítási modell ugyanazt tudja kiszámolni, mint a Turing gép, és a Turing gép futásidejében polinomiális sok lépésben.

2. Kiszámíthatóságelmélet

Alapvető kérdés: Milyen nyelveket tudunk felismerni Turing-gépekkel? Emlékeztetőül:
1.1.9. definíció. Azt mondjuk, hogy a T Turing-gép felismeri az $L \subset \Sigma_0^*$ nyelvet, ha T $\forall w \in \Sigma_0^*$ inputra megáll, és L elemein 1-et, a komplementumán 0-t ad outputként.
2.0.2. definíció. Ha $L \subset \Sigma_0^*$, χ_L jelöli az L karakterisztikus függvényét.

2.0.3. definíció. $f : \Sigma_0^* \to \Sigma_0^*$ függvény rekurzív, ha $\exists T$ turing gép, amely $\forall w \in \Sigma_0^*$ inputon leáll, és leálláskor outputja $f(w)$.

Tehát egy függvény rekurzív, ha van olyan Turing-gép ami őt kiszámolja.

2.0.4. definíció. Az L nyelvet rekurzívnak nevezzük, ha χ_L rekurzív.

Tehát a Turing-géppel fölismerhető nyelvek pontosan a rekurzív nyelvek. Ezek azok a nyelvek amikhez "valamilyen közünk lehet" abban az érdelemben, hogy egy Turing gép véges sok lépésben kiszámolja nekünk hogy egy adott szó benne van-e.

2.0.5. állítás. *Minden véges nyelv rekurzív.*

Bizonyítás. Véges nyelvben van leghosszabb szó, ennek a hossza (legyen n) határozza meg a nyelv szófájának a mélységét. Kiindulva az üres szóból felépítjük a szófát, $n+1$ lépésig építjük mindig a jelenlegi csúcsot reprezentáló szó végéhez konkatenálva a következő betűt (vagy *-ot). A fa leveleire írunk továbbá egy 0 v 1 címkét aszerint, hogy része-e a nyelvnek, vagy sem. Ebből közvetlenül tudunk Turing-gépet csinálni, minden csúcs egy állapot, attól függően, hogy mit olvasunk, lépünk. □

Tehát a véges rekurzív nyelvek érdektelenek.

2.0.6. *példa.* Végtelen rekurzív nyelv az első órai $L = \{(01)^n : n \in \mathbb{N}\}$.

Igaz azonban a következő

2.0.7. tétel. *Majdnem minden nyelv nem rekurzív.*

Bizonyítás. Világos, hogy megszámlálható sok Turing-gép van, és egy gép legfeljebb egy nyelvet ismer fel, tehát legfeljebb megszámlálható sok rekurzív nyelv létezik. Az összes nyelvek halmaza viszont nem megszámlálható, hiszen $|\mathcal{P}(\Sigma_0^*)| = c$. □

Ez persze nem teszi értelmetlenné amit csinálunk, rengeteg érdektelen nyelvet nem tudunk felismerni, és nem is várható el, hogy bármit fel lehesesn ismerni Turing-géppel, de azért marad elég érdekes nyelv is, amit fel tudunk ismerni. Jó lenne, ha konkrét nyelvekről el tudnánk dönteni, hogy rekurzívak-e vagy sem.

2.0.8. definíció. Az L nyelv rekurzíven fölsorolható ha $\exists f : \Sigma_0^* \to \Sigma_0^*$ rekurzív függvény, hogy $L = im(f)$, vagy **ha üres**.

Mért hívjuk ezt rekurzíven fölsorolhatónak? Mi lenne ha csinálnánk egy $\hat{T}$ turing-gépet, ami azt csinálja, hogy a Σ_0^* elemeit egymás után fölírja egy szalagra például lexikografikus sorrendben. Ha ennek a gépnek az outputját rákötjük egy f-et kiszámoló Turing-gép inputjára, akkor az outputszalagon felsoroljuk L összes elemét.

2.0.9. tétel. *Az L nyelv rekurzíven fölsorolható akkor és csak akkor, ha $\exists T$ turing gép, amely Σ_0^* elemei közül pontosan L-en áll meg.*

Bizonyítás. $\Rightarrow$: Legyen a T Turing-gép a következő: soroljuk föl Σ_0^* elemeit, egy w inputra a gép ellenőrizze le, hogy $w = f(w_i)$, ha igen STOP, ha nem, w_{i+1}-re ugyanezt. Ha esetleg üres lenne L, csinálunk egy gépet ami semmilyen inputra nem áll meg.
$\Leftarrow$: Megint fölsoroljuk Σ_0^*-elemeit $(w_i)_1^\infty$. T-t lefuttatjuk w_i-n, ha leáll, legyen az output w_i. Futtassuk T-t $w_{i-\lfloor\sqrt{i}\rfloor^2}$-en i lépésen keresztül. Ha ennyi lépés alatt leáll, akkor output $w_{i-\lfloor\sqrt{i}\rfloor^2}$. Ha nem áll le, akkor adjon vissza egy fix $a \in L$-t. Az indexben lévő függvény minden j értéket végtelen sokszor veszi fel, továbbá korlátlanul sok lépésen keresztül. Tehát ha T megáll az adott indexű szón, egyszer ezt érzékeljük, és kiírjuk, ha pedig nem áll le rajta, akkor egy fix L beli elemet írunk ki, mivel az a szó nem eleme a nyelvnek, így nem is soroljuk föl. □

Be akarjuk látni, hogy majdnem minden nyelv nem rekurzíve felsorolható.

2.0.10. *megjegyzés.* Ez azért érdekes, mert ha L rekurzív akkor rekurzíve felsorolható.

Bizonyítás. Az L nyelv rekurzívsége miatt van olyan Turing-gép, mely 1-et ad $w \in L$ esetén, 0-t ad $w \notin L$ esetén. Ez a gép módosítható úgy, hogy $w \notin L$ esetén végtelen ciklusba kerüljön. Ekkor a 2.0.8 definíció után definiált Turing-gép éppen az L nyelv elemein áll le, azaz L rekurzíve felsorolható. □

2.0.11. állítás. *Majdnem minden nyelv nem rekurzíve felsorolható.*

Bizonyítás. Ugyanolyan számosságos érvelés, mint rekurzív nyelvekre: $\aleph_0$ Turing-gép van, és $2^{\aleph_0} = c$ lehetséges nyelv. □

2.0.12. állítás. *Ha A, B rekurzív, akkor $A \cup B$, $A \cap B$, $A \setminus B$ szintén.*

Bizonyítás. Triviális. □

2.0.13. állítás. *Ha A, B rekurzíve felsorolható, akkor $A \cup B$, $A \cap B$ szintén.*

Bizonyítás. Szintén triviális. □

Azonban az nem mindig igaz, hogy A, B rekurzíve felsorolható nyelvekre $A \setminus B$ is rekurzíve felsorolható. Erre mindjárt adunk példát. Arra is keresünk példát, hogy egy nyelv rekurzíve felsorolható, de nem rekurzív, illetve hogy nem is rekurzíve felsorolható.

2.1. Eldönthetetlen feladatok

2.1.1. definíció (Megállási probléma)**.** Tudunk-e olyan számítógépet csinálni, mely kap egy inputot, amely tartalmazza egy másik számítógép programját és egy inputot ahhoz a géphez. A gépünknek meg kell mondania, hogy az elkódolt gép a megadott inputra leáll, vagy nem áll le. Belátjuk, hogy ilyen számítógépet nem lehet csinálni.

2.1.2. tétel. *Legyen T egy 2-szalagos univerzális Turing-gép a Σ ábécé felett. Jelölje L_T azon $w \in \Sigma_0^*$ szavak halmazát, melyekre T leáll ha mindkét szalagjára w-t írunk. Ekkor L_T rekurzíve fölsorolható, de nem rekurzív.*

A rekurzív felsorolhatóság elég trivi, a másik oldal bizonyításához néhány lemmára van szükség.

2.1.3. lemma. *L rekurzív $\Leftrightarrow$ L és $\Sigma_0^* - L$ is rekurzíve felsorolható.*

Bizonyítás. $\Rightarrow$: Volt, hogy ha L rekurzív, akkor rekurzíve felsorolható is. Ha L rekurzív, akkor nyilván $\Sigma_0^* - L$ is, így az is rekurzíve felsorolható.

$\Leftarrow$: A 2.0.9 tétel szerint egy nyelv pontosan akkor rekurzíve felsorolható, ha létezik Turing-gép, mely pontosan az elemein áll le. Legyen tehát T_1 olyan gép, mely L elemein áll meg, T_0 pedig olyan, mely $\Sigma_0^* - L$ elemein áll meg. Konstruálhatunk egy $\hat{T}$ gépet, mely adott w inputra "odaadja" ezt az inputot a T_0, T_1 gépeknek és figyeli hogy melyik áll le (az egyik nyilván le fog). Ha T_i áll le, akkor $\hat{T}$ kiírja az i outputot. Ekkor $\hat{T}$ felismeri az L nyelvet. □

Tétel bizonyítása. Ahhoz, hogy belássuk, hogy L_T nem rekurzív, az előző lemma értelmében azt kell belátni, hogy a komplementere nem rekurzíve felsorolható. Indirekte bizonyítunk. Ha $\Sigma_0^* - L_T$ rekurzíve felsorolható lenne, akkor létezne olyan $\hat{T}$ gép, mely pontosan $\Sigma_0^* - L_T$ elemein áll le, ebből pedig konstruálható olyan $\hat{\hat{T}}$ egyszalagos gép is, amely pontosan $\Sigma_0^* - L_T$ elemein áll le (k-szalagos gép szimulálható egyszalagossal). Ekkor van egy $\hat{\hat{T}}$-t szimuláló program T-n (hisz az utóbbi egy univerzális Turing-gép). Legyen ez a program p.

Eleme-e p az L_T nyelvnek? Ha $p \in L_T$, az annyit jelent, hogy T megáll a (p, p) bemenetre (mikor mindkét szalagjára p-t írunk). Ez ekvivalens azzal, hogy $\hat{\hat{T}}$ megáll p-n. Ez meg ekvivalens azzal hogy $\hat{T}$ megáll p-n. Végül ez ekvivalens azzal hogy $p \in \Sigma_0^* - L_T$. De ez az ekvivalencialánc láthatóan ellentmondásra vezet (hogy $p \in L_T \Leftrightarrow p \in \Sigma_0^* - L_T$). Ezzel tehát indirekte igazoltuk a tétel állítását. □

2.1.4. következmény. *A (T, w) párokat tartalmazó nyelv, ahol a T Turing-gép megáll w-re, nem rekurzív.*

Bizonyítás. Ha ez rekurzív, akkor amennyiben T egy kétszalagos univerzális Turing-gép, eldönthető, hogy T megáll-e a (w, w) inputon, azaz $w \in L_T$ teljesül-e. L_T azonban nem rekurzív, így ez ellentmondásra vezetne.

Turing-géppel eldönthetetlen nyelvre példa:

2.1.5. *példa* (Dominóprobléma). A dominók négyzetek. A négyzetek oldalaira számok vannak írva. Két négyzet bizonyos oldalaik mentén egymás mellé tehető, ha azonos szám szerepel a két oldalon. A dominókat elforgatni, tükrözni tilos a megadott helyzethez képest. Dominóprobléma inputja: dominókészlet. Azaz meg van adva véges sok típusú dominó, mindegyik típusból végtelen sok áll rendelkezésünkre. Kérdés: parkettázható-e az egész sík ilyen dominókészlettel. A könnyebbség kedvéért azt a kikötést is hozzátesszük a feladathoz, hogy az 1. dominótípust a felsorolásból kötelező legyen használni. Az inputot ilyen formában kapjuk: (a,b,c,d) alakú blokkok sorozata (azaz négy számból álló blokkok zárójelekkel elválasztva). Ezzel a kódolással egy számnégyes a $\overline{\left|\begin{smallmatrix} & a & \\ d & & b \\ & c & \end{smallmatrix}\right|}$ címkézésű dominónak felel meg.

2.1.6. definíció. A dominónyelv azon nyelv, melyben egy szó pontosan akkor van, ha a fenti konvenció szerint kódol egy dominókészletet.

2.1.7. definíció. L_{KIRAK} a dominónyelv azon szavait tartalmazza, amelyek olyan dominókészletet kódolnak, amivel a sík parkettázható.

2.1.8. definíció. L_{NEMRAK} a dominónyelv azon szavait tartalmazza, amelyek olyan dominókészletet kódolnak, amivel a sík nem parkettázható.

2.1.9. *példa.* (1,1,1,1) benne van az L_{KIRAK}-ban.

2.1.10. *megjegyzés.* Trivi, hogy a dominónyelv rekurzív.

2.1.11. tétel. *Az L_{NEMRAK} rekurzíve felsorolható.*

A bizonyításhoz előbb egy lemma:

2.1.12. lemma. *$w \in L_{KIRAK} \Leftrightarrow$ w-vel kirakható minden N természetes számra a $(2N+1) \times (2N+1)$ négyzet.*

Bizonyítás. $\Rightarrow$: Trivi.

$\Leftarrow$: Olyan parkettázásait fogjuk keresni a $(2N+1)\times(2N+1)$ négyzetek sorozatának, amelyben bármely két négyzet közül a nagyobbik parkettázása kiterjesztése a kisebbik parkettázásának. Hogy ilyen létezik, az a Kőnig-lemmához hasonlóan látható be. Az 1×1 nagyságú négyzetet az 1. dominótípussal fedjük. Most a 3×3-ra keresünk jó fedést. Minden $(2N+1)\times(2N+1)$-re van jó fedés, ezek megszoríthatók a 3×3-as négyzetre is. Persze a 3×3-as négyzetnek véges sok fedése adható a dominókészlettel. Ebből adódik, hogy valamelyik, a 3×3-ason vett fedés végtelenszer fordul elő. Válasszuk azt, majd ismételjük ugyanezt a gondolatmenetet a $5\times5, 7\times7, \ldots$ négyzetekre. Ezáltal minden $(2N+1)\times(2N+1)$ méretű négyzetet tudunk úgy prakettázni, hogy a kisebben vett parkettázás mindig a nagyobb parkettázások megszorítása legyen. Ezekből már könnyedén összeállítható a teljes síkon vett parkettázás. □

Tétel bizonyítása. Legyen a T Turing-gép olyan, hogy sorban megpróbálja kirakni a $(2N+1)\times(2N+1)$ négyzeteket $N=0,1,\ldots$ esetekre a megadott dominókkal. Ha leáll, akkor talált egy nem kirakható négyzetet, ami a Lemma értelmében megmutatja hogy $w \in L_{NEMRAK}$. Különben nem áll le. □

Célunk: L_{KIRAK} nem rekurzíve felsorolható.

2.1.13. lemma. *Legyen V rekurzív és $L \subset V$. Ekkor L rekurzív pontosan akkor ha L és $V-L$ rekurzíve felsorolható.*

Bizonyítás. Ha L rekurzív, akkor L és $V-L$ nyilván rekurzíve felsorolható. Ha pedig L és $V-L$ is fekurzíve felsorolható, akkor 2.1.3 mintájára egy Turing géppel leellenőrizhetjük, hogy $w \in V$ (hiszen rekurzív), ha nem, akkor adjon 0-t, ha igen kezdje el szimulálni egy T_0, és T_1 gépeket, melyek pontosan L és $V-L$ elemein állnak le, a rekurzíve fölsorolhatóság miatt persze ilyenek létznek. Világos, hogy pontosan az egyik fog leállni, ennek függvényében, ha T_0 áll le, adjon vissza 1-et, ha T_1, akkor 0-t. □

Másik irányhoz:

2.1.14. lemma. *Turing-géppel nem eldönthető, hogy egy másik, programjával megadott Turing-gép az üres inputra megáll-e.*

Bizonyítás. Legyen T Turing-gép, $u \in \Sigma_0^*$. Rendeljük a (T,u) párhoz a T_u gépet, amely azt teszi, hogy letörli a szalagját, ráírja u-t, majd úgy fut mint T. A T_u speciálisan az üres inputra is úgy fut, mint T az u inputra. Ha mármost lenne olyan gépünk, mely megállapítja hogy üres inputra megáll-e egy gép, akkor ezzel a trükkel a megállási problémára is kapnánk egy megoldást, ami ellentmondás lenne. □

2.1.15. tétel. *L_{KIRAK} nem rekurzíve fölsorolható.*

Bizonyítás. A 2.1.13 lemmát szeretnénk $V = L_{DOMINO}$, és $L = L_{KIRAK}$ szereposztással. Belátjuk, hogy L_{KIRAK} nem rekurzív, ebből következik, hogy nem is rekurzíve fölsorolható (hiszen a komplementere $\in RF$). A problémát redukáljuk a megállási problémára üres inputon. Legyen T tetszőleges Turing-gép, megadunk egy $T \mapsto U$ leképezést, amely dominókészletet rendel egy Turing-géphez, amellyel pontosan akkor áll le az üres inputra, ha U-val nem rakható ki a sík. T-ről föltehető, hogy egyszalagos, ha $T=(1,\Sigma,\Gamma,\alpha,\beta,\gamma)$ megadjuk U-t:

$$(*,N,*,N);(*,p,*,p);((*,\mathrm{START}),p,(\mathrm{START},*),N)$$

ez utóbbi lesz a kezdődominónk. Továbbá $\forall h \in \Sigma : (h,\emptyset,h,\emptyset)$. Ha $\alpha(h,g)=g'$ (de nem STOP!, ez a kontrollkarakter nem kerül dominóra, mert különben a leálló dominó is lefedné a síkot) , $\beta(h,g)=h', \gamma(h,g)=0 : \forall g \in \Gamma : ((h',g'),\emptyset,(h,g),\emptyset)$. Ha $\gamma(h,g)=-1 : (h',\emptyset,(h,g),(g',1))$, ha pedig $\gamma(h,g)=1 : (h',(g',0),(h,g),\emptyset)$ alakú dominó legyen az előző sorok kvantorait figyelembe véve. Lesznek továbbá átmeneti dominók, $((h,g),\emptyset,h,(g,0));((h,g),(g,1),h,\emptyset)$. Az U

teljes készlet ezen dominókból, és a kezdődominót kivéve mindegyiknek a középpontos tükörképét tartalmazza. Először állítjuk, hogyha T nem áll le az üres inputon, akkor ez a készlet lefedi a síkot. Koordinátázzuk a síkot, fedjük az origót a kezdődominóval, a bal, és jobb félegyenest fedje az $N-N$, és a $p-p$ dominók. Lefedjük a felső félsíkot, és tükrösen az alsót. Ha nem áll le, akkor minden esetre tesz egy lépést, tehát a $(*, \text{START})$ oldalához kapcsolódik valamely átmeneti dominó, és ezt folytathatjuk vég nélkül. Ha pedig U fedi a síkot, akkor ismét csak az $N-N$, és $p-p$ dominók fedhetik az x-tengelyt, tehát a felső szintet csak úgy fedhetjük le, hogy a gép állapotaival kapcsolódunk, és mivel fedjük a síkot, nem álltunk le semmilyen véges lépésben. Ezzel tételünket beláttuk. □

Ezzel van mégegy példánk eldönthetetlen nyelvre, és példánk RF^C nyelvre.

3. Tár és idő

3.0.1. *megjegyzés.* 1900-ban egy konferencián David Hilbert német matematikus felsorolt olyan feladatokat, melyek a legfontosabb, legfontosabb matematikai problémák a 20. században. A 10. ezen a listán a Diofantikus egyenletek megoldhatósága, tehát létezik-e olyan algoritmus, hogy tetszőleges $\forall n \forall P \in \mathbb{Z}[X_1, .., X_n]$-ről eldönti, hogy létezik-e megoldása $\mathbb{Z}^n$-felett. 1970-ben Yuri Matiyasevich orosz matematikus oldotta meg végül. A 20. század első felében volt kedvelt kutatási téma az eldönthetőség, utána az erőforrás optimalizálás került a figyelem középpontjába (amikor már kiforrottabb lett a számítástechnológia, konkrét feladatokat kellett kiszámítani konkrét számítógépekkel), mit lehet viszonylag gyorsan, kis erőforrásfelhasználással kiszámítani? Ez vezet át a következő nagy témánkhoz.

3.0.2. definíció. Legyen T egy TG, amely minden inputra megáll. $time_T(n)$ jelöli a legfeljebb n hosszú inputokon használt lépésszámok maximumát. $space_T(n)$ pedig a T által legfeljebb n hosszú inputokon használt mezők száma (azon mezők száma, melyeket meglátogat a fej megállásig). A hosszúságot persze betűben mérjük.

A továbbiakban mindig föltesszük, hogy T minden inputra megáll, és $f : \mathbb{N} \to \mathbb{N}$.

3.0.3. definíció. Ha $L \subset \Sigma_0^*$ egy nyelv, és $w \in \Sigma_0^*$ egy szó, a $w \in L$ kérdés kiszámításán azt a feladatot értjük, hogy egy T Turing gép 1-et ad, ha $w \in L$, és 0-t, ha $w \in \overline{L}$

3.0.4. definíció. $DTIME(f(n))$ jelöli azon L nyelvek osztályát, ahol minden L-hez létezik T mely felismeri L-et, és a $w \in L$ feladatot $f(|w|)$ lépésben megválaszolja.

3.0.5. definíció. $DSPACE(f(n))$ a fentivel analóg módon azon nyelvek osztálya, melyeket valamely T Turing-gép felismer, és a $w \in L$ kérdést $f(|w|)$ tár(=mező) felhasználásával dönti el.

3.0.6. definíció.

$$P := \bigcup_{i=1}^{\infty} DTIME(n^i)$$

a polinomiális időben, és

$$PSPACE = \bigcup_{i=1}^{\infty} DSPACE(n^i)$$

a polinomiális tárban felismerhető nyelvek osztálya.

3.0.7. *megjegyzés.* Mért pont a polinomiális futásidőt/tárat szeretjük? Például a négyzetképzésre zárt, tehát a szalagok számától függetlenül vannak benne ebben az osztályban. A gyakorlatban a kivető általában 10 alatt van, a valóságban egy 100-as kitevő nagyjából ugyanannyira használhatatlan, mint ha exponenciális időben futna.

3.0.8. *példa.* Tegyük föl, hogy adott három algoritmus A_i. Az első lineáris, a második köbös, a harmadik expoenciális futási idővel, ezer időegység alatt szeretnénk választ kapni, mekkorára növelhetjük az input méretét. A_1-nél persze 1000-ig, A_2, A_3-nál nagyjából 10-ig. Mennyivel nagyobb inputokat tudunk kiszámolni tízszer ennyi idő alatt? Lineárisnál persze 10-szer akkorát, A_2-re nagyjából 2-3-szor akkorát, de mivel a harmadik exponenciális, ott csak kb 3-mal nagyobb inputot tudunk kiszámolni. Itt a minőségbeli különbség, polinomiális esetben multiplikatíve nő a kiszámítható input mérete, míg exponenciálisnál additíven. A valóságban persze nem igazán lehet csak úgy megtízszerezni a számítás kapacitásunkat.

3.1. Tár és idő viszonya

Ha T egy k szalagos Turing gép, akkor $k \cdot time_T(n) \geq space_T(n)$ teljesül. Következésképp $DTIME(f(n)) \subset \bigcup_{k=1}^{\infty} DSPACE(k \cdot f(n))$ teljesül.

3.1.1. definíció. Egy T TG konfigurációja leírható a

$$(g, p_1, .., p_k, h_1, .., h_k)$$

sorozattal, ahol $g \in \Gamma, p_i \in \mathbb{N}, h_i \in \Sigma_0^*$ a p_i-k a fejek pozícióját kódolják el a h_i-k elejétől számolva, a h_i-k pedig a mezők azon helyeinek tartalma, ahol a fej járt már, illetve az input, ha esetleg nem olvasta még mind be (az u. lépésben mindegyik legföljebb $u + c$ hosszú).

Ebből a konfigurációból "hibernálhatjuk" a TG-et, ez teljes információt ad a jelenlegi számítási helyzetről, innen később folytatható.

3.1.2. állítás. *Ha egy TG minden inputra leáll, akkor nem kerülhet kétszer ugyanabba a konfigurációba.*

Bizonyítás. Képzeljük el a konfigurációk által alkotott irányított gráfot. Ha egy konfigurációt kétszer vesz föl, akkor kör van ebben a gráfban, és a gép ciklusba kerül. □

3.1.3. állítás. $DSPACE(f(n)) \subset \bigcup_{c=1}^{\infty} DTIME(c^{f(n)})$

Bizonyítás. Föltettük, hogy T mindig megáll, és $L \in DSPACE(f(n))$-et felismeri $f(n)$ tárban. Hány konfiguráció lehet, ha $f(n)$ tárat használ? $c \cdot f(n)^k \cdot |\Sigma|^{f(n)}$ látványosan fölülről becsüli ezt az értéket, ezzel a keresett állítást beláttuk. □

3.1.4. *példa* (Az euklideszi algoritmus.). Input $b \leq a \in \mathbb{N}$, az output pedig $(a, b) = LNKO(a, b)$. Az algoritmus maga ismételt maradékos osztásokból áll:

$$\begin{aligned} a &= c_1 b + r_1 \\ b &= c_2 r_1 + r_2 \\ r_1 &= c_3 r_2 + r_3 \\ &\vdots \end{aligned}$$

Egy idő után 0 maradékot kapunk, az ezt megelőző maradék lesz az output. Az input mérete $log(a)+log(b)$, legfeljebb $b-1$ lépésben ér véget az algoritmus, ez így nem polinomiális! Az input méretében tehát exponenciális. Mivel azonban maradékos osztást végzünk, az r_i-k nem csak hogy szigorúan monoton csökkennek, de minden lépésben legalább a felére csökkennek, tehát valóban logaritmikus sok lépésben véget ér az algoritmus (feltéve, hogy tudunk polinomiális időben osztani).

3.1.5. *példa* (a^b mod m). Input $a, b, m \in \mathbb{N}$. Az egyszerű hatványozás persze nem lehet polinomiális, hiszen már az input hosszának kiírása is több lépésbe kerül aszimptotikusan. *1. eset.* Ha $b = 2^l$, akkor ismételten négyzetre emelünk, és redukálunk mod m, legfeljebb l lépésben meghatároztuk a keresett maradékosztályt.
2. eset. Ha b nem kettő hatvány, akkor felírjuk kettes számrendszerben, és a fenti eljárással elvégezzük minden kettes számrendszerbeli 1 helyiértékére az $a^{2^{l_i}}$-t, és összeszorozzuk, ez még mindig logaritmikus sok lépés. b kettes számrendszer beli alakja is gyorsan meghatározható ismételt osztással.

3.1.6. *példa.* $\binom{u}{v}$ mod m, és $u!$ mod m-re nem ismert polinomiális algoritmus (de elvben létezhet??).

Mért vagyunk ilyen nagyvonalúak a konstans szorzókkal?

3.1.7. tétel (Lineáris gyorsítási tétel)**.** *Legyen $f : \mathbb{N} \to \mathbb{N}$ függvény, $f(n) \geq n$. Tegyük fel, hogy a T TG az L nyelvet felismeri $f(n)$ időben, azaz $\forall w \in \Sigma_0^*$-ra T leáll legföljebb $f(|w|)$ lépésben, és kiszámolja L karakterisztikus függvényét. Ekkor $\forall c > 0 : \exists \hat{T}$ TG, ami ugyanezen L-et ismeri fel, és $time_{\hat{T}}(n) \leq cf(n) + 2n$*

Bizonyítás. Időgépet készítünk. Feltesszük, hogy T egyszalagos, $\hat{T}$ pedig $2p-1$ szalagos lesz, ahol $\frac{1}{2p} \leq c$. A sokszalagos gép egyszalagoson való szimulálását csináljuk visszafelé, $2p$ magasra felhullámoztatjuk T szallagját, sok fejjel egy lépésben a gép p lépését tudjuk szimulálni, egy

adott szalagpozíció "p sugarú környezetét" lefedjük a sok fejjel, és elkódoljuk az állapotokba. A $+2n$ az input átmásolására szolgál az új szalagokra, az első fejjel olvasunk egyet, az i. ráírja a szalagjára, és így tovább. □

3.1.8. tétel. *Legyen f egy rekurzív függvény. $\exists L$ rekurzív nyelv, amire $L \notin DTIME(f(n))$.*

Bizonyítás. Föltehető, hogy $f(n) \geq n$, legyen T kétszalagos univerzális TG. Legyen továbbá $L := \{w | T\ w, w$ inputra leáll legfeljebb $f(|w|)^3$ lépésben$\}$. Állítjuk, hogy ezen L megfelelő. Egy TG-el kiszámolhatjuk $f(|w|)^3$-öt, majd futtatjuk T-t a w, w inputon ennyi lépésig, ha leáll, akkor elfogadjuk, ha nem, akkor nem. Ez bizonyítja hogy L rekurzív. Indirekt tegyük fel, hogy $L \in DTIME(f(n))$, ekkor $\exists \hat{T}$ mely felismeri L-et $f(n)$ lépésben. Ekkor létezik olyan egyszalagos $\hat{\hat{T}}$ TG, ami L-et felismeri $cf(|w|)^2$ lépésben, tehát kellően hosszú inputokra legfeljebb $f(|w|)^3$-ben. Ezen $\hat{\hat{T}}$ géphez rendeljünk egy egyszalagos T_0-t, amely ciklusba kerül ha $\hat{\hat{T}}$ elfogad, és megáll ha $\hat{\hat{T}}$ elutasít. Legyen p T_0 programja T-n. $p \in L$?. Ha igen, akkor p, p inputra T leáll $f(|w|)^3$ lépésben, tehát T_0 a pn leáll, tehát $\hat{\hat{T}}$ elutasította p-t, tehát nincs benne L-ben ellentmondás. Ha $p \notin L$, akkor T_0 leáll $f(p)^3$ lépésben, vagyis T leáll $f(|p|)^3$ lépésben p, p inputtal, $p \in L$, ellentmondás. □

3.2. A nemdeterminisztikus Turing-gépek

Egy determinisztikus TG-et teljesen leírja a konfigurációs gráfja, adott inputra mindig ugyanúgy fut. Ezzel szemben egy nemdeterminisztikus TG lefutása nem egy irányított út, hanem egy bonyolultabb gráf, minden konfigurációból több konfigurációba lép*het*. Nem véletlenszerűen lép, nem eloszlásokkal fogunk babrálni.

3.2.1. definíció (NDTG)**.** $(k, \Sigma, \Gamma, \Phi)$ nemdeterminisztikus TG, az első három komponens u.a. mint a sima TG-nél, és $\Phi \subset (\Sigma^k \times \Gamma) \times (\Sigma^k \times \Gamma \times \{-1, 0, +1\}^k)$ az úgynevezett átmenetreláció. Ha T a g állapotban a $b = (b_1, .., b_k)$ betűket olvassa, ezután a g' állapotba kerül, a b' betűket írja, és a fejek $\epsilon \in \{-1, 0, +1\}^k$ irányokba mozognak, akkor megengedett (legális) egy lépés, ha $((b, g), (b', g', \epsilon)) \in \Phi$. Azt mondjuk, hogy a T nemdeterminisztikus TG t lépésben elfogad egy inputot, ha létezik megengedett lépéseknek olyan t hosszú sorozata, amely végén megáll, és kiírja, hogy "1".

3.2.2. *megjegyzés.* Ha egy NDTG t lépésben elfogadja a w szót, akkor lehet olyan csupa legális lépésből álló működése is, amelyben

- nem áll meg
- megáll, és kiírja, hogy „koronavírus”.

Ha mondjuk minden állapotból 3 különböző másikba mehet, t lépés után 3^t különböző konfigurációban lehetne, ha csak egy ilyen sorozatban fogad el ez elhanyagolhatóan kicsi valószínűségű lesz, de minket nem a valószínűség érdekel.

3.2.3. *példa.* $G3C := \{G : G$ gráf, és $\chi(G) \leq 3\}$, mutatunk egy NDTG-t, amely lineáris időben eldönti, hogy egy gráf benne van-e ebben a nyelvben. Rendeljünk minden csúcshoz egy színt (a lehetséges háromból) nemdeterminisztikusan (tehát minden lehetséges háromszinezést, akár jó akár nem, megengedettnek tekintünk). Majd leellenőrizzük, hogy jól szineztünk-e azzal, hogy megnézzük minden él végpontjai különböző színűek-e. Ha igen, kész vagyunk, ha nem, elutasítunk. Pontosan akkor van elfogadó lefutása ennek a gépnek, ha a gráf háromszinezhető.

3.2.4. definíció. $NTIME(f(n))$ jelöli azon L nyelvek összességét, melyekhez létezik egy T NDTG, amely L-et felismeri(/elfogadja) $f(|w|)$ lépésben, minden mást elutasít.

$NSPACE(f(n))$ hasonlóképpen azon nyelvek osztálya, melyekhez létezik olyan NDTG, amely a nyelvet felismeri $f(|w|)$ tárban, minden mást elutasít.

$$NP := \bigcup_{0}^{\infty} NTIME(n^i)$$

$$NPSPACE := \bigcup_{0}^{\infty} NSPACE(n^i)$$

3.2.5. *megjegyzés.* $PSPACE = NPSPACE$, ahogy majd később látni fogjuk. $P = NP$-t senki nem tudja, pl. $G3C \in NP$, és nem tudjuk, hogy P-ben van-e.

Az NP osztállyal analóg módon

3.2.6. definíció.

$$co - NP = \{L : \Sigma_0^* - L \in NP\}$$

vagyis az NP-beli nyelvek komplementereinek osztálya.

3.2.7. *megjegyzés.* P szimmetrikus nyelvosztály, vagyis $P = co - P$, míg NP nem szimmetrikus, hiszen csak azt kötöttük ki, hogy egy NDTG nem fogadja el a nyelv komplementerének szavait. Ezért érdekes $co - NP$, itt egy NDTG megmondja nekünk, hogy egy adott szó nincs benne L-ben. Mivel minden TG egyben NDTG is $P \subset NP \cap co - NP$.

Adunk egy ekvivalens definíciót NP-re.

3.2.8. definíció. Azt mondjuk, hogy a $w \in \Sigma_0^*$-nek az L beli tagságára az y szó *polinomiális tanú*, ha L-hez van olyan T (determinisztikus) TG, hogy $w \in L \Leftrightarrow T$ elfogadja a (w, y) párt $|w|$-ben polinomiális időben.

3.2.9. *megjegyzés.* Tehát minden esetre $|y| \leq |w|^c$

3.2.10. tétel. *$L \in NP \Leftrightarrow \exists T$ (determinisztikus) TG és $c > 0$ konstans, hogy $w \in L \Rightarrow \exists y \in \Sigma_0^*$, $|y| \leq |w|^c$ és T elfogadja a (w, y) párt $|w|^c$ időben, vagyis $w \in L$-nek van polinomiális tanúja.*

3.2.11. *példa.* Mért "tanú"? Gondoljunk a Hamilton-kör keresési problémára, egy adott kört könnyű leellenőrizni, végighúzzuk az ujjunkat, mint később látni fogjuk a $HAMILTON := \{G : G\text{-ben van Hamilton-kör}\}$, NP-beli. A nemlétezést ellenben nem tudjuk (mai tudásunk szerint) ilyen könnyen megindokolni, nincs polinomiális tanú H-kör nemlétezésére, avagy $HAMILTON \not\in^{?} co-NP$.

Bizonyítás. ($\Rightarrow$): Egy adott $w \in L$ esetén y legyen a $\hat{T}$ NDTG elfogadó lépéseinek sorozata (amelyek megengedettek). Ha most megadunk egy w inputot, a $\hat{T}$ leírását és az y-t, akkor az tudja ellenőrizni, hogy az a w input esetén a $\hat{T}$ NDTG-nek az elfogadó lépéseinek a sorozata-e.

($\Leftarrow$): Csinálunk egy $\hat{T}$ NDTG-t. Nemdeterminisztikus fázisban fölír egy $y_1y_2..y_t$ szót (ahol $y_i \in \Sigma$), majd ellenőrzi, hogy T elfogadja-e a $(w, y_1y_2..y_n)$ párt. □

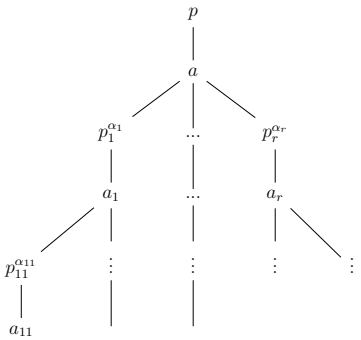
3.2.12. *példa* (Példák NP-beli nyelvekre).

- Az előbb említett $HAMILTON$ nyelv a fenti ekvivalencia miatt.
- A korábban látott háromszínezhető gráfok $G3C$ nyelve.
- $INDEPENDENT := \{(G, k) : G \text{ gráf}, k \in \mathbb{N}, \text{ és } \alpha(G) \geq k\}$, vagyis azon gráfok nyelve, hogy van G-ben legalább k elemű független csúcshalmaz.
- $PRIM := \{p \in \mathbb{N} : p \text{ prím}\}$, ez a nyelv triviálisan $co-NP$-beli, hiszen adunk egy osztót, azt polinom időben le lehet ellenőrizni.

3.2.13. tétel (Pratt)**.** $PRIM \in NP$

Bizonyítás. Ehhez szükségünk lesz arra a tényre, hogy egy n szám prím volta ekvivalens azzal, hogy $\exists a : (a, p) = 1$, és $\langle a \rangle = (\mathbb{Z}/n\mathbb{Z})^{\times}$ ($\Rightarrow$ létezik primitív gyök mod p, $\Leftarrow$ a valamelyik hatványa 1, tehát relatív prím p-hez, és így minden hatványa is, vagyis az $1, 2, .., p-1$ reprezentánsok mind relatív prímek p-vel, vagyis p prím). $o(a)|p-1$, legyen $p-1 = \prod p_i^{\alpha_i}$, és $r_i = \frac{p-1}{p_i}$. Ha minden i-re $a^{r_i} \not\equiv 1$ mod p, akkor ő biztosan primitív gyök.
A tanúnk tehát az a, és $p-1$ prímfelbontása, ezekből leellenőrizzük, hogy $a^{p-1} \equiv 1$ mod p, és $a^{r_i} \not\equiv 1$ mod p. Ezeket mint a múltkor láttuk ki lehet számolni polinom időben, és $O(\log(p))$ prímtényezője van $p-1$-nek.

De nem hisszük el, hogy a prímfelbontás helyes, tehát még további információra is van szükségünk, minden p_i-hez kérünk egy saját a_i primitív gyököt bizonyítékként, majd ezekhez is, és így tovább... Milyen nagy lesz a fa amit így kapunk? Mivel minden szinten a prímek osztói az előző szintbelieknek, legfeljebb feleakkorák, vagyis a fa szintjeinek száma $\leq 2\log(p)$, és minden szint szélessége $\leq \log(p)$, hiszen egy szint elemeinek a szorzata az előző szint elemei-1-eknek a szorzata. Következésképpen legfeljebb $2\log^2(p)$ csúcsa lesz a fának, ami polinomiális $\log(p)$-ben, vagyis



az input p hosszában. □

Vagyis a prímnyelv $NP \cap co-NP$-beli, és ma már azt is tudjuk, hogy P-beli. Általában az szokott történni, hogy egy nyelvről belátják, hogy metszetbeli, majd találnak rá egy polinomiális algoritmust, ez meglepő lenne azonban a kövekező esetben:

3.2.14. *példa.* $FACTOR := \{(n,k) : n,k \in \mathbb{N} : \exists d | n : 1 < d \leq k\}$. Ismerjük persze az egészek faktorizációs problémáját, input egy természetes szám, output vagy egy $u, v : 1 < u, v < n$ és $n = uv$ felbontás, vagy az hogy n prím. Ezt ismételve u, v-re faktorizálhatjuk n-et polinomidőben, ha a fenti problémát meg tudjuk oldani polinomidőben. Ha tudunk faktorizálni, akkor az $(n,k) \in^? FACTOR$ eldönthető, megnézzük a legkisebb prímosztóját. Fordítva is igaz, ha a faktornyelvbeli tagságot meg tudjuk mondani, akkor tudunk lineáris kereséssel faktorizálni is. Az triviális, hogy $FACTOR \in NP$, egy $d \leq k$ tanúskodik erről. Azonban benne van $co-NP$-ben is, erről tanúskodik n prímtényezős felbontása. A prímséget az előbbi tétel szerint tudjuk ellenőrizni, azt is, hogy a szorzatuk valóban n, végül a legkisebb prímosztó tanúskodik arról, hogy nincs k-nál kisebb osztója n-nek.

Nem tudjuk, hogy P-beli lenne a $FACTOR$ nyelv, ha igaz lenne, akkor az RSA, és más publikus kulcsú titkosítási algoritmusok haszontalanná válnának. Igaz továbbá, hogy a $FACTOR$ nyelv kvantumszámítógépen polinomiális időben elvégezhető (Schur algoritmus), de ezeknek tárgyalása nem része a kurzusnak (más igazán híres problémát nem lőnek le kvantumgépek, pl. az NP-teljes nyelveket nem).

3.3. $P = NP$?

3.3.1. definíció. $L, K \subset \Sigma_0^*$ nyelvek, L polinomiálisan visszavezethető K-ra, ha van egy olyan $f : \Sigma_0^* \to \Sigma_0^*$, amely polinomiális időben kiszámolható, és minden $w \in \Sigma_0^*$-ra $w \in L \Leftrightarrow f(w) \in K$. Ezt $L \propto K$-val jelöljük.

3.3.2. *megjegyzés.* f-ről nincs feltéve, hogy injektív/szürjektív, stb, csak az, hogy polinomiális időben kiszámítható.

3.3.3. definíció. Az L nyelv NP-teljes, ha $L \in NP$, és minden $K \in NP$-re igaz, hogy polinomiálisan visszavezethető L-re, vagyis $\forall K \in NP : K \propto L$.

3.3.4. *megjegyzés.* Tehát egy NP-teljes nyelv nem túl nehéz abban az értelemben, hogy NP-ben van, de elég nehéz abban az értelemben, hogy legalább olyan nehéz, mint bármely más NP-beli nyelv.

3.3.5. *megjegyzés.* Legyen L NP-teljes, és tegyük föl, hogy $L \in P$. Ebből következik, hogy $P = NP$, hiszen minden $K \in NP$-re $K \propto L \in P$. Így úgy tudjuk a $w \in K$? problémát eldönteni, hogy kiszámoljuk $f(w)$-t polinom időben, majd erről eldöntjük, hogy L-ben van-e, ami megtehető feltevés szerint polinom időben. Vegyük észre, hogy ha $P = NP$, akkor $P = co-NP$ is, hiszen minden P-beli nyelv komplementuma is P-beli.

3.3.6. definíció. $x_i \in \{0,1\}$ logikai változók, ezekre alkalmazhatjuk a logikai műveleteket: *és* $(x_1 \wedge x_2)$, *vagy* $(x_1 \vee x_2)$, valamint *nem* $(\bar{x}_1)$. *Boole formulának* nevezünk egy ezen műveletekből álló kifejezést. *Boole függvény* egy $f : \{0,1\}^n \to \{0,1\}$ függvény. Persze minden Boole formulához hozzárendelhetünk egy ilyen függvényt a formula kiértékelése által.

3.3.7. *megjegyzés.* Ez fordítva is igaz, $\alpha \in f^{-1}(1)$-re $\bigvee_{\alpha \in f^{-1}(1)}(x_{i_1} \wedge .. \wedge x_{i_n}) \wedge (\bar{x}_{j_1} \wedge .. \wedge \bar{x}_{j_k})$, ahol az x_{i_*}-ok az α 1-es koordinátáinak megfelelő változók, a többi a 0-s változók. Világos, hogy nem egyértelmű ez a hozzárendelés, bármely azonosan igaz formulát hozzávagyolhatnánk, tehát értelmes kérdés az, hogy egy Boole-formula mely Boole-függvényt írja le?

3.3.8. definíció. *Konjunktív normálformában* (vagy röviden CNF-ben) van egy Boole formula, ha $(* \vee \bar{*} \vee .. \vee *) \wedge () \wedge ... \wedge ()$ alakú, tehát esetleg tagadott változók vagyolásainak ésekkel való összekötése.

3.3.9. definíció. $SAT := \{\varphi : \varphi$ CNF-ben van, és nem azonosan hamis (*kielégíthető*)$\}$.

3.3.10. tétel (Cook)**.** *A SAT nyelv NP-teljes.*

3.3.11. *megjegyzés.* Mit is kell bizonyítani?

- $SAT \in NP$, ez könnyű, egy kielégítő helyettesítés polinomiális tanú, a kiértékelés lineáris idejű, minden zárójel belsejét kell külön külön megnézni.
- $\forall L \in NP : L \propto SAT$, ez a nehezebb. $w \in \Sigma_0^*$ $w \mapsto \varphi_w$ CNF, hogy $w \in L \iff \varphi_w \in SAT$

Bizonyítás. Legyen $T = (1, \Sigma, \Gamma, \Phi)$ egy NDTG, létezik egy $c > 0$ hogy $w \in L$-et T eldönti $|w|^c$ lépésben. A c, w, T adatokhoz szeretnénk polinomiális időben egy φ_w CNF-át konstruálni. Definiáljuk az $x[n,g], y[n,p], z[n,p,h]$ Boole változókat, ahol $0 \leq n \leq \lceil |w|^c \rceil = N$, $g \in \Gamma$, $-N \leq p \leq N$, és $h \in \Sigma$. Nézzük T egy megengedett lefutását. $x[n,g] = 1$ pontosan akkor, ha T az $n.$ lépésben a g állapotban van. $y[n,p] = 1$ ha T feje az $n.$ lépésben a $p.$ pozicióban van. $z[n,p,h]$ pedig akkor legyen igaz, ha T szalagjának a $p.$ pozíciójában a h betű van az $n.$ lépésben. Ez megmondja a gép lefutását teljesen. Olyan formulát szeretnénk, amelyet az x, y, z változók pontosan akkor elégítenek ki, ha T egy megengedett működését írják le a w inputon, melynek végén a gép elfogad legfeljebb N lépésben.

A konstrukció több segédkifejezésből fog előállni $\varphi_w := \bigwedge \varphi_i$ alakban.

$$\varphi_1 := \bigwedge_{n=0}^{N} \bigvee_{g \in \Gamma} x[n,g]$$

az első segédformula, ez azt követeli meg, hogy minden lépésben valamilyen állapotban legyen a gép.

$$\varphi_2 := \bigwedge_{n=0}^{N} \bigwedge_{g \neq g' \in \Gamma} (\overline{x[n,g]} \vee \overline{x[n,g']})$$

a gép egyszerre csak egy állapotban lehet. Analóg módon $\varphi_3 := \bigwedge_{n=0}^{N} \bigvee_{p=-N}^{N} y[n,p]$, mert valahol mindig lennie kell a fejnek. $\varphi_4 := \bigwedge_{n=0}^{N} \bigwedge_{-N \le p \neq p' \le N} (\overline{y[n,p]} \vee \overline{y[n,p']})$, a fej egyszerre csak egy helyen lehet. Hasonlóképpen a szalagon lévő betűkre is, mindig lennie kell egynek és csak egynek minden cellában. $\varphi_5 := \bigwedge_n \bigwedge_p \bigvee_h z[n,p,h]$, és $\varphi_6 := \bigwedge_n \bigwedge_p \bigwedge_{h \neq h'} \overline{z[n,p,h]} \vee \overline{z[n,p,h']}$, mostmár megköveteltük a szalag tartalmát is.

$$\varphi_7 := x[0, START] \wedge y[0,0] \wedge x[N, STOP] \wedge z[N,0,1]$$

a start, stop állapotok megkövetelése, és hogy a 0. mezőben legyen 1 mert elfogadunk.

$$\varphi_8 := \bigwedge_{p=-N}^{-1} z[0,p,*] \wedge \bigwedge_{p=0}^{|w|-1} z[0,p,w_{p+1}] \wedge \bigwedge_{p=|w|}^{N} z[0,p,*]$$

itt az inputot követeltük meg. Most pedig az átmeneteket:

$$\varphi_9 := \bigwedge_n \bigwedge_p \bigwedge_{(h,g,h',g',\epsilon) \notin \Phi} (\overline{x[n,g]} \vee \overline{y[n,p]} \vee \overline{z[n,p,h]} \vee \overline{x[n+1,g']} \vee \overline{y[n+1,p+\epsilon]} \vee \overline{z[n+1,p,h']})$$

Ez azt mondja, hogy ha van egy nem legális átmenetünk, a benti formula pontosan akkor lesz hamis, ha egy nem megengedett átmenetet kódolna el a bemenet.

$$\varphi_{10} := \bigwedge_n \bigwedge_p \bigwedge_h (y[n,p] \vee \overline{z[n,p,h]} \vee z[n+1,p,h])$$

tehát ha nincs valahol a fej, akkor nem változhat a szalag tartalma. Trivi, hogy az x,y,z változók kielégítik a konstruált kifejezést, visszafelé szintén a konstrukcióból világos, hogy ha egy változóhalmaz kielégíti ezt, akkor az a T egy futását írja le. Nem bizonyítjuk külön, de ezen formulákat egy TG polinomiális időben tudja generálni, ezzel a bizonyítás készen van. □

3.3.12. definíció. *Hipergráf* egy (V, H) pár, V egy véges halmaz, $H = \{H_i : \forall i : h_i \subset V\}$. Egy hipergráf *két színnel szinezhető*, hogyha a V-t ki tudjuk szinezni két színnel úgy, hogy minden (hiper)él belemetsz mindkét színosztályba.

3.3.13. tétel. *A $H2C \in NPC$, tehát a kétszínezhető gráfok nyelve NP-teljes.*

Bizonyítás. $H2C \in NP$ triviális, egy jó szinezés polinomiális tanú. Mért? Legyen a hipergráf incidenciamátrixszal megadva, a sorok a csúcsok, az oszlopok az élek, ezen a mátrixon lineáris időben végigiterálunk. Az állítás másik feléhez elegendő $SAT \propto H2C$-t megmutatni, mert előbbi NP-teljes, így két lépésben minden nyelvet visszavezethetünk $H2C$-re a SAT-on keresztül (a polinomiális visszavezetés tranzitív!).

Legyen tehát φ konjunktív normálforma, kellene egy $\varphi \mapsto \mathcal{H}$ lekpezés. Ezen $\mathcal{H}$ alaphalmaza a φ változói, ezek negáltjai (literálok) és még egy x szimbólum. Legyen minden $\{x_i, \bar{x}_i\}$ pár egy él, és φ minden zárójelének tartalma $\cup\{x\}$ is legyen él (pl. $(x_1 \vee \bar{x}_2 \vee x_3) \mapsto \{x_1, \bar{x}_2, x_3, x\}$).

Ha $\varphi \in SAT$, akkor van kielégítő helyettesítése, tehát a literálokat ki lehet szinezni igazra és hamisra a hozzárendeléstől függően, legyen továbbá x hamis. Állítjuk, hogy ez egy jó

két-szinezés. A tagadáspárokból álló élek megfelelőek, hiszen mindegyiket vagy igaznak, vagy hamisnak választottuk a kiértékelésben, a párja a másik színt kapja. A többi élben legalább egy igaz lesz, mert ez egy megfelelő kiértékelés, és vagyok sorozata pontosan akkor igaz, ha legalább az egyik tagja igaz. Így az él jól van szinezve, mert mindegyiknek tagja x aki hamis volt.

Ha a konstruált halmazrendszer kétszinezhtő, akkor ez egy értékhozzárendelést ad, mert minden változóra ő és a tagadása különböző színű. Melyik szín az igaz? Nézzük meg az x pont színét, mindig az ő színe legyen a hamis, ez jó lesz, mert minden zárójelben lesz legalább egy másik színű literál, tehát minden zárójelben van legalább egy "igaz" literál, vagyis a kapott helyettesítés kielégíti a formulát. □

3.3.14. *megjegyzés.* Csinálhattuk volna úgy is, hogy megvizsgáljuk hogy φ SAT-ban van-e vagy sem, majd hozzárendelni egy triviálisan kétszínezhető, vagy nem színezhető gráfot, de nem ezt tettük, mert nem tudunk erre jó tesztet. A tartalmazástól függetlenül csináltuk meg a konstrukciót.

3.3.15. tétel. $G3C \in NPC$

Bizonyítás. Láttuk már hogy NP-ben van, tanú a szinezés. Belátjuk, hogy $H2C \propto G3C$. $\mathcal{H} = (V, H)$ egy hipergráf, a konstruálandó ("sima") gráf ugyanezen a V halmazon lesz megadva, még egy extra x csúcs hozzávételével, aki mindenkivel össze van kötve. A hipergráf minden H_i éléhez hozzárendelünk továbbá egy $2\lceil(|H_i|+1)/2\rceil - 1$ csúcsú ($= |H_i|$, ha páratlan, $|H_i|+1$ ha páros) körgráfot az eddigi csúcsoktól diszjunkt pontokon. Ciklikus sorrendben kössük össze a hipergráf egy élében lévő csúcsokat a megfelelő kör pontjaival. Ha páros sok csúcsa van, akkor a kör egyik pontjával kétszer.

Ha egy kétszinezhető hipergráfra a konstrukciót alkalmazzuk, legyen x zöld, a hipergráf csúcsai pedig pirosak és kékek a kétszinezésből megfelelő módon. Hogyan vannak a páratlan hosszú körök kiszinezve? A páratlan hosszú kör minden csúcsára megmondhatjuk, hogy a hipergráf éleinek szinezése milyen követelményeket ad meg rá, ha a szomszédja piros, akkor nem lehet a kör megfelelő pontja (esetleg pontjai) pirosak, végezzük ezt el minden csúcsra. Nevezzük blokknak a körben a szomszédos "nem lehet piros" cimkéjű csúcsok kiterjeszthetetlen halmazait. Mivel páratlan hosszú a kör, biztosan van páratlan elemszámú blokk, ennek mindkét végét szinezzük a megengedett színre (pl ha nem piros van ráírva, akkor kékre), a blokk (mondjuk) jobb oldali szomszédos mezőjén legfeljebb két szín van tiltva az eddigi szinezés szerint (a kék, és a cimkéje), tehát azt tudjuk valahogyan szinezni, ezt folytathatjuk amíg körbe nem érünk. Az eljárás akkor akadhat el amikor elérünk a blokk másik széléhez, de ekkor is tudunk jól szinezni, mert itt is legfeljebb két szín van tiltva, és mivel ugyan az a szín a blokk másik végén, így az nem jelent egy harmadik tiltást. A blokk, mivel páratlan hosszú a peremfeltétel mellett felváltva kiszinezhető.

A másik irányhoz, tegyük fel, hogy a kapott gráf 3-szinezhető, legyen x színe zöld, ekkor a hipergráf csúcsaira a kék-piros szineket használhatjuk. Mit jelentene, ha lenne egyszínű él a hipergráfban? Ez pont azt jelentené, hogy a hozzá tartozó páratlan kör két színnel van kiszinez-

ve, ez pedig látványosan lehetetlen. Tehát a piros és kék színek jól szinezik a hipergráfot. Így polinomiálisan visszavezettük a $H2C$ nyelvet $G3C$-re, mivel előbbi NP-teljes, utóbbi is az. □

3.3.16. következmény. *Minden $k \geq 3$-ra a GkC (legfeljebb) k-színnel szinezhető gráfok nyelve NP-teljes.*

Bizonyítás. Világos, hogy $GkC \in NP$, tanú a szinezés. Másrészt $G3C \propto GkC$ teljesül. Minden G gráfhoz egy $\hat{G}$ gráfot rendeülnk. Álljon az új gráf a régi gráfból, és még egy K_{k-3}-ból, amit az eredeti gráf minden csúcsával összekötünk. Világos, hogy ha az eredeti gráf 3-szinezhető, akkor az új 3+k-3 színnel szinezhető, hiszen egy teljes gráf kromatikus száma a csúcsszámával egyenlő, illetve ha az új gráf k-szinezhető, akkor a teljes gráf $k-3$ színét nem használhatjuk az eredeti gráf szinezésére mert mindenkivel össze vannak kötve, ergo a fennmaradó 3 színnel van szinezve. □

3.3.17. *megjegyzés.* $k = 2$-re a feladat polinomiális, mint azt végmaton láthattuk - t.i. szélességi bejárással, ez ugyanis vagy kiszinezi, vagy ad egy páratlan kört.

3.3.18. *megjegyzés.* Az eddigiekből következik, hogy ha polinomiális időben ki tudnánk számolni tetszőleges gráf kromatikus számát, akkor el tudnánk dönteni a GkC nyelvbeli tartalmazást is, és $P = NP$ lenne. Ezt úgy mondjuk, hogy a kromatikus szám kiszámolása *NP-nehéz* (ha lenne rá polinomiális algoritmus, akkor egy NPC-beli nyelv felismerhető lenne).

3.3.19. tétel. *Egy gráf függetlenségi számának kiszámolása NP-nehéz.*

3.3.20. definíció. $INDEPENDENT := \{(G,k) : G$ gráf amiben van legalább k elemű független csúcshalmaz$\}$

3.3.21. *megjegyzés.* Tartalmazásra nézve maximálisat triviális találni mohó algoritmussal, de ez nem lesz maximális általában.

3.3.22. tétel. $INDEPENDENT \in NPC$

Bizonyítás. A nyelv NP-beliségére tanú egy legalább k elemszámú független csúcshalmaz. Továbbá $G3C \propto INDEPENDENT$ teljesül. Konstruálnunk kell egy (G', k) párt, ami pontosan akkor van benne a nyelvben, ha a kiindulási G gráf háromszínezhető. Vegyük G három diszjunkt példányát. A másolásnál azonos ősű csúcsok példányait kössük össze. Legyen $k := |V(G)|$, állítjuk, hogy ez a konstrukció jó.

Ha G háromszinezhető, akkor partícionálhatjuk a csúcsait háromfelé, hogy egy osztályon belül nem mennek élek. Tekintsük az első másolatban az első színhez, a másodikban a másodikhoz, harmadikban a harmadikhoz tartozó csúcsokat, ezek együtt egy megfelelő méretű csúcsosztályt alkotnak.

Tegyük fel, hogy a konstruált gráfban van egy k elemű független csúcshalmaz. Ez a csúcshalmaz belemetsz mindhárom másolatba valahogyan (esetleg üresen). Ezeknek a metszeteknek az őse egy-egy színosztályt definiál a kiindulási gráfon a függetlenség miatt, tehát az eredeti gráf 3-szinezhető. □

3.3.23. definíció. $INDEPENDENT_k = \{G : \alpha(G) \geq k\}$

3.3.24. *megjegyzés.* Minden k-ra az $INDEPENDENT_k$ nyelv P-ben van, mert polinomiálisan sok k csúcsú részhalmaza van a csúcsoknak, ezeken végigiterálva leellenőrizzük, hogy van-e köztük független.

3.3.25. *megjegyzés.* Az $INDEPENDENT$ nyelv egy optimalizálási feladat gráfokon, ha ki tudjuk számolni a függetlenségi számot polinomiálisan, akkor a tagságot is, és fordítva.

3.3.26. definíció. $SAT - 3 := \{\varphi \in SAT : \varphi$ minden változója legfeljebb három klózban szerepel$\}$.

3.3.27. tétel. $SAT - 3 \in NPC$

Bizonyítás. Egy kielégítés ismét csak megfelelő polinomiális tanú. Megmutajuk, hogy $SAT \propto SAT - 3$. Ismét egy megfelelő transzformációt kell csinálnunk, amely tetszőleges kielégíthető CNF-hez egy olyat rendel aminek minden változója legfeljebb 3 klózban szerepel. $\varphi = (x_1 \vee ...) \wedge ()...$ legyen, a j. klózban az x_i változót kicseréljük egy új x_i^j változóra, így minden változó pontosan egy klózban fog szerepelni, el szeretnénk érni, hogy fix i-re az x_i^j-k mind ugyan azt a logikai értéket vegyék fel. Hogy fejezzük ki azt hogy $x_1^1 = x_1^2 = ...$? Az implikációt szeretnénk használni, az x_1^j-k implikálják körbe egymást $x_1^1 \to x_1^2 \to ... \to x_1^1$, ez pontosan azt éri el, hogy ezek a változók mind ugyanazt az értéket vehessék csak fel. Elemi logikából tudjuk, hogy $a \to b = \bar{a} \vee b$, tehát $x_1^1 \vee \bar{x_1^2}$, stb alakokat toldunk a formula végére. Világos, hogy ezekkel a toldásokkal pontosan háromszor fog szerepelni minden változó, továbbá az is, hogy ekvivalens a konstrukció. □

3.3.28. definíció. $LEFOG := \{(H, k) : H$ hipergráf, melynek élei lefoghatók k csúccsal$\}$
$LEFED := \{(H, k) : H$ csúcsai lefedhetők legfeljebb k hiperéllel$\}$

3.3.29. tétel. $LEFOG \in NPC$

Bizonyítás. Megmutatjuk, hogy $SAT - 3 \propto LEFOG$, egy k elemű lefogó csúcshalmaz (tehát amely minden hiperélbe belemetsz) nyilván polinomiális tanú, pl. az incidenciamátrixszal.

Hogy rendeljünk egy megfelelő CNF-hez hipergráfot? Csúcsok a literálok, és tagadásaik, élek legyenek az $\{x, \bar{x}\}$ halamzok, továbá a klózok, ez lesz H, k legyen a változók száma a formulánkban.

Kellene, hogy a formula pontosan akkor elégíthető ko, ha ez a hipergráf lefogható k csúccsal. Ha van egy kielégítésünk, akkor véve egy jó értékhozzárendelést mint kétszinezés a csúcsokon, az igaz szín pont lefedi az összes hiperélt (persze pont annyi elemű mint szeretnénk).

Ha van egy lefogó csúcshlamazunk, nevezzük az elemeit igaznak, pont egy megfelelő érték-hozzárendelést kapunk, amely kielégíti a formulát. □

3.3.30. tétel. *$LEFOG \in NPC$ akkor is, ha minden csúcs legfeljebb 4 halmazban van benne.*

Bizonyítás. Az előbbi konstrukciót nézzük. Mivel $SAT-3$-at vettük a konstrukcióban igazából olyan hipergráfot kaptunk, amelyben minden csúcs legfeljebb 4 élben van benne. □

3.3.31. *megjegyzés.* A $LEFOG$ nyelv duálisa a $LEFED$. Minden élnek és csúcsnak veszünk egy pontot, ehhez csinálunk egy páros gráfot úgy, hogy egy élt pontosan azokkal a pontokkal kötünk össze, amik benne vannak. A $LEFOG$ nyelvben a csúcsok között van k db olyan, hogy azoknak az "él" szomszédai már a teljes színosztály a páros gráfban. Ezzel szemben a $LEFED$ nyelvben az "él" osztályból keresünk k olyan csúcsot, amiknek a szomszédai a másik ("csúcs") osztályban már mindenki. Ezzel $LEFED \propto LEFOG$, és persze NP-beli is, vagyis $LEFED \in NPC$. (A hipergráf páros gráf reprezentációját "fordítva" nézzük).

3.3.32. tétel. *$LEFED \in NPC$ marad akkor is, ha minden él legfeljebb 4 csúcsból áll.* □

3.3.33. definíció. $K-PART := \{(H,k) : H$ hipergráf, $k \in \mathbb{N}$, és H éleiből kiválasztható legfeljebb k db úgy, hogy az alaphalmaz partícióját adják$\}$

3.3.34. tétel. $K-PART \in NPC$

Bizonyítás. Megmutatjuk, hogy $LEFED \propto K-PART$, és hogy NP-beli. Polinomiális tanú a partícionáló élek halmaza. Szokásos visszavezetési érvelés $(H,k) \mapsto (\hat{H},k)$ a megfelelő tulajdonságokkal. $\hat{H} = \{h : \exists h' \in H : h \subset h'\}$, ez általában túl sok lenne (nem lenne polinomiális ezt a hipergráfot megkonstruálni), de feltehető az eddigiek szerint, hogy minden él legfeljebb 4 csúcsú (így minden élhez legfeljebb 16 másik élt rendelünk).

Vegyünk legfeljebb k élt, melyek H-t fedik, ebből szeretnénk egy ugyanennyi elemű partíciót találni. Vegyük az első fedő hiperélt, majd a második ettől diszjunkt részét, a harmadik ezen előzőektől diszjunkt részét és így tovább, ezek mind élei lesznek $\hat{H}$-nak, találtunk egy kellő partícionáló élhalmazt.

Ha van egy partícionáló élhalmazunk $\hat{H}$-ban, akkor tekintsük az őket adó éleket H-ban, ezek persze lefedik, és szintén legfeljebb k db van belőlük. □

3.3.35. definíció. $PART := \{H : H$ hipergráf, melynek van olyan élekből álló halmaza, amely partícionálja az alaphalmazt$\}$

3.3.36. *megjegyzés.* $K-PART$-hoz képest most nem párok a nyelv elemei, a priori nem tudjuk hogy hány éllel lehet lefedni. Világos, hogy NP-beli persze, ismét egy fedő élhalmaz tanú.

3.3.37. tétel. $PART \in NPC$

Bizonyítás. $(H,k) \mapsto H'$, hogyan? Vegyünk a hipergráf mellé még k db pontot, ez lesz H' alaphalmaza, az élei pedig az eredeti H élei uniója az extra k db pont egyelemű részhalmazaival (ergo minden élnek lesz k db másolata egy-egy extra csúccsal).

Ha H partícionálható **pontosan** k éllel, az első élhez vegyük hozzá az extra pontokból az elsőt, a másodikhoz a másodikat stb, ezzel partícionáltuk a H'-t. Mért tehettük fel, hogy nem legfeljebb hanem pontosan k éllel partícionáltuk? Ha kevesebb is partícionálja, akkor hozzávehetjük az üres halmazt kellően sokszor a megfelelő partícióhoz.

Ha a nagy gráfot tudjuk partícionálni, akkor a k db új pontnak mind külön-külön partícionáló élben kell lennie, ezeknek a H-ba eső része partícionálja azt, persze legfeljebb k db van belőlük és konstrukció miatt élek is. □

3.3.38. definíció ($SUBSET - SUM$)**.** A *részhalmazösszeg feladat* és *nyelv*. Az input egy $(a_1, .., a_n, b) \in \mathbb{N}^{n+1}$, kérdés hogy van-e olyan $I \subset \{1, 2, .., n\}$, hogy $\sum_{i \in I} a_i = b$ teljesül? Ebből a részösszegnyelv:

$$SUBSET - SUM := \{(a_1, .., a_n, b) \in \mathbb{N}^{n+1} : \exists I \subset \{1, 2, .., n\} : \sum_{i \in I} a_i = b\}$$

3.3.39. tétel. $SUBSET - SUM \in NPC$

Bizonyítás. NP beliség szokás szerint triviális, tanú I. Belátjuk, hogy $PART \propto SUBSET - SUM$. Egy H hipergráfhoz rendelünk egy természetes szám $n+1$-est (n a hipergráf csúcs-, m pedig az élszámát jelöli). A hipergráf alaphalmazának minden elemére felírjuk az $r = m + 1$ szám hatványait 0-tól $n-1$-ig. Ha A_i hiperél H-ban, akkor a_i a csúcsaihoz rendelt r-hatványok összege ($=\sum_{v_i \in A_j} r^i$). Legyen $b := \frac{r^n - 1}{r - 1}$.

Ha H-nak van partícionáló élhlamaza, akkor persze az ezekhez tartozó a_i-ket választva megkapjuk b-t.

Ha előállítottuk b-t az a_i-k egy részhalmazának összegeként, kellene, hogy ez csak úgy lehet, hogy a megfelelő élek partícionálnak. Ez azért lesz igaz, mert b-t felírva az r alapú számrendszerben, $r = m + 1$ miatt ha az a_i-kből felírtuk valahogy, akkor az r hatványok együtthatói azonosan legfeljebb m-el egyenlőek, tehát ez is a valódi felírása b-nek ebben a számrendszerben, ezt akartuk belátni. □

Most a hátizsákfeladatot fogjuk nézni (lol). Az input egy $v_1, .., v_m, s_1, .., s_m, b \in \mathbb{N}$ szám $2m + 1$-es. Kalózok az andokba rejtették a kincseiket (valami vízközelebbi hely helyett), a v_i az értéke, az s_i pedig a súlya az adott tárgyaknak, célunk maximalizálni a bepakolt értéket, feltéve, hogy legfeljebb b tömegű dolgokat vihetünk el ($\sum s_i \leq b$).

3.3.40. definíció.

$$HA'TIZSA'K := \{(s_1, .., s_m, v_1, .., v_m, b, k) \in \mathbb{N}^{2m+2} : \exists I \subset \{1, 2, .., m\}, \sum_{i \in I} s_i \leq b, \sum_{i \in I} v_i \geq k\}$$

3.3.41. tétel. *Ez a nyelv is NP teljes.*

Bizonyítás. Tanú I. $SUBSET - SUM \propto HA'TIZSA'K$-ot fogjuk kihozni.

$$(a_1, .., a_n, b) \mapsto (a_1, .., a_n, a_1, .., a_n, b, b)$$

lesz a visszavezetésünk. Az egyik irány triviális, mivel az összeg egyenlő b-vel, kisebb és nagyobb-egyenlő is. A másik irányban ugyanígy, ha a hátizsáknyelvben van amit kapunk a konstrukcióból, akkor a kétoldali egyenlőtlenség miatt a részösszegnyelvben is benne lesz. □

Most megoldjuk a hátizsákfeladatot polinomidőben:

3.3.42. *megjegyzés.* Ha $v_i = v$, akkor mohón a legkönnyebbeket bepakolhatjuk, trivi. Ha a súlyok egyenlőek, hasonlóképpen a legértékesebbeket pakoljuk be, amíg befér ($\lfloor b/s \rfloor$).

Egy dinamikus programozási algoritmust adunk a hátizsákfeladatra: csinálunk egy mátrixot s_{ij} := azon tárgyrészhalmazok súlyainak a minimuma, amelyeket az első i tárgyból választunk ki, és az értékük legalább j, ha van ilyen, $+\infty$, ha nincs. Így egy m sorú és $1 + \sum v_i$ oszlopú mátrixot kapunk, az m. legalsó sorban látunk valami érdekeset, ha elindulunk ebben a sorban a legszélső elemtől visszafelé amíg b-alá nem érünk megkapjuk a maximális értéket, amire a súlyösszeg megfelelő. Az első sort könnyen kitölthetjük, $0, s_1, .., s_1, \infty, ..$ amint elérünk a $v_1 + 1$. oszlophoz. A többi sorhoz legyen $s_{ij} := \min(s_{i-1,j}, s_{i-1,j-v_i} + s_i)$ (az egy indexű s_i-k a súlyok továbbra is, a kétindexűek a mátrix elemei), ezzel a rekurzióval már kitölthetjük a mátrixot. Hány lépésben fut ez le? $O(m(\sum v_i))$ lépésben, ezzel szemben az input mivel számjegyekkel van megadva $O(\sum \log(s_i) + \sum \log(v_i) + \log(b))$.

3.3.43. definíció. v_i *apró*, ha létezik olyan $c > 0$, hogy $v_i \leq m^c$ minden i-re.

Ekkor a fenti algoritmus polinomiálisan is lefut $O(m^{2+c})$ időben. Az órai felvétel nem volt érthető, itt le van írva az algoritmus, a pdf az előadás oldaláról származik.

Ibarra-Kim approximációs algoritmusa

Legyen adva n darab tárgy pozitív egész w_i súlyokkal és e_i értékekkel, valamint egy w súlykorlát, továbbá egy $\varepsilon > 0$ pontossági korlát. Tegyük fel, hogy minden i-re $w_i \leq w$, hiszen a súlykorlátnál nehezebb tárgyakat úgysem vihetnénk magunkkal. Legyen

$$\mathrm{OPT} = \max \left\{ \sum_{i \in I} e_i \quad \middle| \quad I \subseteq \{1, 2, \ldots, n\} \quad \text{és} \quad \sum_{i \in I} w_i \leq w \right\},$$

valamint legyen I^* egy olyan részhalmaz, amire a maximum éppen eléretik.

Legyen most M egy később alkalmasan választandó szám, ezzel készítsük el a $w'_i = w_i$, $e'_i = \lfloor e_i / M \rfloor$ súlyokat és értékeket, erre a dinamikus programozási algoritmussal keressük meg a w méretű hátizsákban elvihető legnagyobb értéket, legyen ez OPT'. Ezen optimum vétessen fel valamely J részhalmazon, ezt egyébként a dinamikus programozási algoritmussal visszalépéssel (minden minimumszámításnál annak megjegyzésével, hogy a két tag közül melyik eredményezi a minimumot) ki lehet számítani.

Először is vegyük észre, hogy

$$\frac{e_i}{M} - 1 \leq \left\lfloor \frac{e_i}{M} \right\rfloor \leq \frac{e_i}{M},$$

valamint hogy I^* optimális válsztása miatt

$$\mathrm{OPT}' = \sum_{i \in J} \left\lfloor \frac{e_i}{M} \right\rfloor \leq \frac{1}{M} \sum_{i \in J} e_i \leq \frac{1}{M} \mathrm{OPT}.$$

Másfelől a J optimális volta miatt

$$\mathrm{OPT}' = \sum_{i \in J} e'_i \geq \sum_{i \in I^*} e'_i \geq \sum_{i \in I^*} \frac{e_i}{M} - 1 \geq \frac{1}{M} \sum_{i \in I^*} e_i - n = \frac{1}{M} \mathrm{OPT} - n.$$

Ezek alapján tehát látjuk, hogy

$$\mathrm{OPT} - nM \leq M \cdot \mathrm{OPT}' \leq \mathrm{OPT},$$

most még megmutatjuk, hogy $nM \leq \varepsilon \mathrm{OPT}$, ezzel azt fogjuk látni, hogy

$$(1 - \varepsilon)\mathrm{OPT} \leq M \cdot \mathrm{OPT}' \leq \mathrm{OPT},$$

azaz az algoritmusból adódó $M \cdot \mathrm{OPT}'$ a valódi optimumnak legfeljebb ε hibájú becslése.

Megadjuk végül M értékét. Legyen $E = \sum_{i=1}^{n} e_i$, ezzel legyen

$$M = \frac{\varepsilon E}{n^2},$$

ekkor

$$nM = \varepsilon \frac{E}{n} \leq \varepsilon \max_{1 \leq i \leq n} e_i \leq \varepsilon \mathrm{OPT},$$

hiszen az átlag nem nagyobb a maximumnál valamint bármelyik konkrét egyetlen tárgyat önmagában el tudjuk vinni.

Már csak az algoritmus lépésszáma van hátra, ez

$$O\left(n \sum_{i=1}^{n} e_i'\right) \leq O\left(n \sum_{i=1}^{n} \frac{e_i}{M}\right) \leq O\left(\frac{nE}{M}\right) \leq O\left(\frac{n^3}{\varepsilon}\right),$$

ami valóban polinomiális approximációs algoritmust jelent.

3.3.44. tétel. *Hasonló polinomiális approximációs tétel nem létezhet egy gráf kromatikus számának közelítésére.*

Bizonyítás. Mi lenne, ha ki tudnánk számolni egy $f(G)$-t, amire $1 \leq f/\chi < 4/3$, azaz $\chi(G) \leq f(g) < \frac{4}{3}\chi(G)$? Indirekt tegyük fel, hogy ki tudnánk-e ezt számolni, ekkor el tudnánk dönteni, hogy egy gráf háromszinezhető-e, hiszen $f(G)$-nek pontosan 3-nak kell lennie. $4/3$ helyett 2-vel sem lehet igaz a tétel, sőt hasonló feltevés mellett $n^{1-\epsilon}$-al sem. □

4. Randomizált algoritmusok

A véletlent, mint erőforrást fogjuk használni.

4.0.1. *példa* (Polinomazonosság probléma). f, g két n-változós egész együtthatós polinom. Kérdés, hogy $f \equiv g$, ekvivalensen eldöntendő, hogy egy polinom azonosan 0-e (az $f - g$ jelen esetben). A komplikációt az adja, hogy nem feltétlenül monomok összegeként van adva a polinom, egy n tagú, tagonként 2 elemű szorzat kibontásában 2^n tag lesz, vagyis simán kibontani, és megnézni nem opció. Behelyettesíthetünk különböző a_i vektorokat, ha valamelyik nem nullát ad, nyertünk, de ha mind 0, attól nem feltétlenül lesz azonosan 0. Kérdés, hogy mennyire lehetünk biztosak a dolgunkban l behelyettesítés után, ha mind nulla.

4.0.2. lemma (Schwartz-lemma)**.** *Ha $h \neq 0$ n-változós polinom, és minden változójában legfeljebb d-ed fokú, ha a $\xi \in_R \{1, .., N\}^n$ vektort koordinátánként függetlenül egyenletes eloszlásból sorsoljuk, akkor $\mathbb{P}(h(\xi) = 0) \leq nd/N$*

Egyváltozós nemnulla polinomnak csak fokszámnyi gyöke lehet, de többváltozósnak akár kontinuum sok is (pl.: $x - y$). Egy kockában becsüljük meg, hogy mennyire valószínű, hogy az l próbánk mind hamis pozitívat ad. Ha $N = 2 * n * d$, akkor legfeljebb $1/2$ valószínűséggel találunk el egy gyököt, csináljuk meg ezt a próbát 100-szor, vagy kapunk egy nemnulla helyettesítést, vagy pedig azt mondjuk, hogy $h \equiv 0$, a hibánk valószínűsége legfeljebb $1/2^{100}$.

Bizonyítás. $n =$ 1-re az algebra alaptétele adja az indukció kezdetét. n-re tudjuk, $n + 1$? $h(x_1, .., x_n, x_{n+1})$ polinom, írjuk fel x_1 hatványai szerint $h = \sum x_1^i h_i$, ahol a h_i-k n-változósak. Legyen t a legnagyobb index, amire $h_t \not\equiv 0$, ha t=0, akkor az indukciós feltevés szerint készen vagyunk.

$$\mathbb{P}(h(\xi) = 0) = \mathbb{P}(h(\xi) = 0 | h_t(\xi) \neq 0)\mathbb{P}(h_t(\xi) \neq 0) + \mathbb{P}(h(\xi) = 0 | h_t(\xi) = 0)\mathbb{P}(h_t(\xi) = 0)$$

a teljes valószínűség tételéből, becsüljük a két tagot. A másodikra alkalmazhatjuk az indukciós feltevést: $\leq 1 \cdot nd/N$. Nyilván $\mathbb{P}(h_t(\xi) \neq 0) \leq 1$, a szorzótényezőjéhez $h_t(\xi) \neq 0$ miatt egy 1-változós t-edfokú polinom nullhelyét keressük, indukcióból így

$$\mathbb{P}(h(\xi) = 0 | h_t(\xi) \neq 0) \leq t/N \leq d/N$$

összeadva $(n + 1)d/N$. □

4.0.3. *megjegyzés.* Lényeges feltétel, hogy h nem azonosan nulla, hiszen különben azonosan 1 a keresett valószínűség!

Ha ki tudnánk számolni egy csupa transzcendens számokból álló vektort, egy számolással tudnánk dönteni, de persze ilyen számokkal csak közelítőleg tudunk számolni. Véletlenített algoritmusból egyre csökkentve a szükséges véletlen bitek számát, ha mondjuk $O(\log n)$ véletlen bitre lenne szükség, akkor egyszerűen megnézzük mind az $O(n)$ lehetséges helyettesítést, nyerve egy determiniszikus algoritmust, így látták be például, hogy $PRIM \in P$ (lényegében).

4.1. Prímtesztek

Az input n, az output 1 ha prím, 0 ha nem. Az osztókat végignézve $n-1$-ig, vagy akár csak $\sqrt{n}$-ig exponenciális algoritmus lesz, Eratosztenész szitája is stb. Random választva számokat, megnézve, hogy osztói-e n-nek szintén nem jó, mi van ha $n = pq$, két nagy prímre, $1/\sqrt{n}$ valószínűséggel találjuk el random választva ismét. Az ötlet a kis-Fermat tételből jön. Legyen $\mathbb{Z}_m^*$ a redukált maradékosztályok multiplikatív csoportja mod m, $H_m := \{a \in \mathbb{Z}_m^* : a^{m-1} \equiv 1 \mod m\}$ nyilvánvalóan egy részcsoport. Ekkor vagy egyenlő vele, vagy pedig az elemszáma osztja $|\mathbb{Z}_m^*|$-ot, tehát legfeljebb fele akkora. Ha m prím, akkor persze egyenlőség teljesül, ezt mondja ki a tétel, tehát ha nincs egyenlőség, akkor m nem prím, továbbá ekkor az $\{1, 2, .., m-1\}$ halmaz elemeinek legfeljebb felére teljesülhet a kis-Fermat tétel állítása, vagyis legalább $1/2$ valószínűséggel találunk olyan a-t, ami ellentmond a feltételnek. Így ha 100-szor választunk, legfeljebb $1/2^{100}$ valószínűséggel jelezzük hibásan, hogy prím. Léteznek azonban a Carmichael-számok, más néven pszeudoprímek, melyekre $H_m = \mathbb{Z}_m^*$, de nem prím.

A Rabin-Miller teszt ad erre megoldást. Tegyük fel, hogy m páratlan, $m-1 = 2^M q$, ahol q is páratlan.

$$a^{m-1} - 1 = a^{2^M q} - 1 = (a^q - 1)(a^q + 1)(a^{2q} + 1)(a^{2^2 q} + 1) \dots (a^{2^{M-1} q} + 1)$$

A teszt a következőből áll: tekintsünk egy véletlen a számot $\{1, .., m-1\}$-ből, kiszámoljuk a fenti kifejezés jobb oldalán lévő szorzat tagjait, és ellenőrizzük, hogy m ezek közül legalább az egyiket osztja. Ha igen, akkor elfogadjuk prímnek, ha nem, elutasítjuk. Ha m nem prím, akkor legalább $1/2$ valószínűséggel "lebukik" a Rabin-Miller teszten, prímek átmennek (Lovász 5.2.2. Lemma). Ezt elvégezve mondjuk 100-szor megfelelően kicsiny hibavalószínűséget kapunk, és a számolások polinomidőben elvégezhetőek, és polinomiálisan sokat kell elvégezni belőlük.

A polinomekvivalencia problémát alkalmazzuk most egy gráfelméleti feladatra. Egy páros G gráfban teljes párosítást szeretnénk keresni. Készítsük el az A mátrixot, melynek sorai feleljenek meg a gráf egyik, oszlopai pedig a másik színosztálybeli csúcsainak, és álljon 1 a megfelelő helyen, ha a csúcsok között megy él, 0 különben. Egy teljes párosítás egy bástyaelrendezésnek felel meg nyilván. Ha $\det A \neq 0$, akkor a hozzá tartozó gráfban van teljes párosítás, hiszen a kifejtésben találunk legalább egy nemnulla tagot, ez pedig pont egy jó elrendezésnek felel meg.

Visszafelé nem következtethetünk, lehet, hogy van sok jó elrendezés, de pont kiejtik egymást, és a determináns nulla lesz. Írjunk most az u_iv_j él helyére most nem egyest, hanem egy x_{ij} változót, ezzel kapjuk, hogy ennek az A' mátrixnak a determinánsa pontosan akkor nem azonosan nulla mint polinom, ha létezik teljes párosítás. A visszafelé következtetés azért működik, mert minden monom legfeljebb egyszer szerepel a determináns kifejtésében. $\det A'$ egy legfeljebb n^2 változós, változónként legfeljebb elsőfokú, tehát $\xi \in \{1, 2, .., 2n^2\}^n$-beli random választással elvégezhetjük a polinomekvivalencia randomizált tesztjét (mondjuk százszor). Azért érdekel minket ez, mert a determinánsszámítást tudjuk párhuzamosítani, tehát gyorsabban is ki tudjuk számolni modern eszközökön, plusz könnyebben programozható ez, mint a Kőnig-tétel. Ugyanez véghezvihető általános gráfra is.

5. Kriptográfia

5.1. Titkosítási eljárások

Alíz, és **B**ob üzeneteket akar váltani, **É**va tudta nélkül, aki lehallgatja a kommunikációs csatornájukat, és ismeri a titkosítási módszerüket (E=eavsdropper). Tiktokos/nyilvános kulcsú titkosítás a két fő paradigma, előbbivel kezdjük. A titkos kulcs hátránya, hogy a kommunikáció kezdete előtt megbízható módon ki kell cserélnie A-nak B-vel a titkos kulcsaikat.

5.1.1. definíció (One Time Pad titkosítás)**.** Legyen $r \in \{0,1\}^N$ véletlen vektor, $x \in \{0,1\}^N$ az üzenet, A elküldi $x \oplus r$-et ($\oplus$ a bitenként mod 2 összeadás) B-nek, ehhez mégegyszer hozzáadva r-et B visszakapja az üzenetet. Az $x_i + r_i$ koordináták $1/2$ valószínűséggel 1 vagy 0, tehát a kódolt üzenet homogénnek néz ki E számára.

Abszolút biztonságos, hátránya hogy a kódcserét meg kell ejteni korábban.

5.1.2. *megjegyzés.* A gyakorlatban A-nak és B-nek is van egy-egy külön fájlja a titkos r kulcsokkal (mindketten ismerik mindkét fájlt, de az egyikük csak az egyiket, a másikuk csak a másikat használja enkódolásra), és ebből mindig megfelelő bitnyit használnak fel.

5.1.3. definíció (diszkrétlogaritmus-probléma)**.** Input: $a, a^b \pmod p, p$ prím, output: b.

A diszkrétlogaritmus-problémáról úgy gondoljuk, hogy nehéz megoldani.

5.1.4. definíció (titkoskulcs-csere protokoll)**.** A üzenet x, ezt akarja elküldeni. Generál egy véletlen $p \gg x$ prímet és egy $u : (p, u) = 1$ számot, és elküldi B-nek az $(x, x^u \pmod p), p)$ hármast. B generál egy $v : (p, v) = 1$ számot, és visszaküldi x^v-t (mod p). Ekkor a titkos kulcs $x^{uv} \pmod p$ lesz, amelyet mindketten ki tudnak számolni.

Ekkor E-nek $(x, x^u \pmod p), p)$ alapján kellene kiszámolnia u-t, amely pont a diszkrét logaritmus-probléma, amelyről úgy gondoljuk, hogy nehéz, továbbá azt is úgy gondoljuk, hogy a diszkrét logaritmus-probléma nélkül nem határozható meg x^{uv}.

Ezek titkos kulcsot használó titkosítások voltak. Ezzel szemben a nyilvános kulcsú titkosítások a következő sémát használják: van egy $M(\cdot,\cdot)$ gyors algortimus (például $\mathbb{N}\times\mathbb{N}$-en), és A-nak, B-nek van e_A illetve e_B nyilvános, és f_A, f_B titkos kulcsaik.

Ha A el akarja küldeni x-et, akkor elküldi B-nek $M(x,e_B)$-t. Teljesülnie kell f_B-re és M-re, hogy $M\big(M(x,e_B),f_B\big)=x$. Hasonlóan $M\big(M(y,e_A),f_A\big)=y$ kell. Ekkor e_B, e_A közzétehető nyilvánosan, f_A-t, f_B-t pedig egyedül A-nak, illetve B-nek kell ismernie. Ekkor ha $M(x,e_B)$-t nem lehet könnyen visszafejteni e_B ismeretében sem, de gyorsan kiszámolható, akkor ez jó titkosítás. Kérdés, hogy van-e ilyen.

5.1.5. *megjegyzés* (Üzenet aláírása). Általában magától igaz, hogy $M\big(M(x,f_B),e_B\big)=x$ is teljesül (hasonlóan e_A-ra és f_A-ra). Ekkor A tudja hitelesíteni az üzenetét: elküldi $\big(x,M(x,f_A)\big)$-t, és ekkor B tudja $M(x,f_A)$-ra alkalmazni e_A-t, és ha megkapja az x-et, akkor meggyőződhet róla, hogy valóban A küldte az üzenetet.

5.1.6. definíció (RSA)**.** Legyen $p,q\gg 0$ különböző prímszámok, $m=pq$. (Ezeket megtalálhatjuk úgy, hogy generálunk nagy számokat, ellenőrizzük, hogy prímek-e, és, mivel $\frac{\pi(x)}{x}\sim\frac{1}{\ln x}$, viszonylag rohadt lassan tudunk is két prímet találni.)

Ekkor $\varphi(m)=(p-1)(q-1)=pq-(p+q)+1$, így m ismeretében $\varphi(m)$ és a (p,q) pár kiszámítása ekvivalens.

A választ f_A-t, hogy $(f_A,\varphi(m))=1$ és keres e_A-t, hogy $e_Af_A\equiv 1\pmod{\varphi(m)}$ (Euler-Fermat miatt ilyen persze létezik).

Legyen $M(x,f_A)=x^{f_A}\pmod m$. Ekkor B ezt felemeli az e_A. hatványra, ekkor $x^{e_Af_A}\equiv x$, így ez egy jó nyilvános kulcs–titkos kulcs pár A-nak. (Feltéve, hogy sem x nem számolható ki x^{f_A} ismeretében, és p,q sem m ismeretében.)

5.1.7. *megjegyzés.* Ha e_A,f_A kongruens mod m kis számokkal, akkor nem olyan nehezen feltörhető az RSA.

5.2. Titokmegosztás

Cél: van egy titok (egy kód), és van n játékosunk, $2\le k\le n$, és az a cél, hogy bármely k játékos együtt megtudja a titkot, de semelyik $\le k-1$ ne tudhasson meg semmit.

5.2.1. *példa* (algebrai megoldás). Legyen $a_i\in_R\{1,2\dots N\}$ $(1\le i\le k-1)$, és legyen $f(x)=a_{k-1}x^{k-1}+a_{k-2}x^{k-2}\dots+\dots a_1x+a_0$, ahol $a_0=f(0)$ a titok.

A játékosok legyenek $P_1,P_2\dots P_n$, és P_i-nek odaadjuk $f(i)$-t. Ekkor bármelyik k játékos ismer egy $k-1$ fokú polinomot k helyen, így polinominterpolációval meghatározható f, így $f(0)$ is. Ha $\le k-1$ ember áll össze, akkor ha ehhez hozzáveszem $f(0)=a$-t tetszőleges a-ra, akkor az egyértelműen meghatároz egy polinomot, így a $\le k-1$ érték alapján $f(0)$ nem meghatározható.

5.2.2. *példa* (optikai megoldás). Két játékos van, mindkettőnél egy-egy kép, ha ezeket egymásra helyezzük, akkor rajzolódjon ki a titok, egyébként semmit ne lehessen tudni. A titokról feltesszük, hogy egy fekete pixelekből álló kép.

Osszuk fel az első fóliát jó sok 2×1-es téglalapra, és minden ilyen téglalapot úgy színezünk ki, hogy az egyik fele fekete, a másik átlátszó, és $\frac{1}{2}$-$\frac{1}{2}$ eséllyel a bal, illetve a jobb oldalát sötétítjük be. A másik fóliát is felosztjuk ilyen téglalapokra, és ha a T téglalap a titokban fekete, akkor azt az oldalát színezzük ki, amelyik az első fólián átlátszó, ha pedig a titokban világos, akkor azt, amelyik az első fólián fekete. Ekkor a két fóliát egymásra rakva szürke háttér előtt kirajzolódik a titok.

Ekkor az első fólia nyilván semmit nem mond a titokról, mert nem is használtuk fel, hogy az hogyan néz ki. A második fóliában minden pixel 1 vagy 0 valószínűséggel fekete, feltéve a titkot, de a feltétel nélkül ott is $\frac{1}{2}$-$\frac{1}{2}$ valószínűséggel fekete illetve átlátszó egy pixel.

6. Kommunikációs játékok

6.1. Alapfogalmak

Cél: van két játékos, akik bármit ki tudnak számolni gyorsan, de egymás között nehezen kommunikálnak.

6.1.1. definíció. Adott $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ és $x, y \in \{0,1\}^n$. A ismeri x-et, de y-t nem, B ismeri y-t, de x-et nem. Ki akarják számolni $f(x,y)$-t. A költség az A és B között (bármely irányban) kommunikált bitek száma.

Akkor tekintjük $f(x,y)$-t kiszámoltnak, ha az egyik játékos ismeri $f(x,y)$-t, és a másik játékos tudja, hogy az egyik tudja.

6.1.2. definíció. A P protokoll mellett f költsége a legrosszabb (x,y) input páron $\kappa_P(f)$.

6.1.3. *megjegyzés.* Megkövetelhetnénk, hogy mindketten tudják $f(x,y)$-t, ez 1 bit különbséget jelentene csak legfeljebb.

6.1.4. definíció. A közös számolási módszer szabályait, hogy mikor ki, és milyen bitet küld *protokollnak* nevezzük. (Ez az algoritmus megfelelője több játékos esetén.)

6.1.5. *példa.* Legyen f tetszőleges, ekkor A elküldheti x-et B-nek, aki „ingyen” kiszámolja $f(x,y)$-t. Ennek a költsége n.

6.1.6. *példa* (ID-függvény). Legyen

$$\mathrm{ID}(x,y) = \begin{cases} 1, & \text{ha } x = y \\ 0, & \text{ha } x \neq y \end{cases}$$

Ekkor a fenti P protokollal $\kappa_P(\mathrm{ID}) = n$ teljesül.

6.1.7. definíció (kommunikációs bonyolultság)**.** $\kappa(f)$ a $\kappa_P(f)$-ek minimuma az összes f-et kiszámoló P protokollon.

6.1.8. tétel. $\kappa(\mathrm{ID}) = n$.

Ennek a bizonyításához kell a következő definíció és tétel.

6.1.9. definíció. f kommunikációs mátrixa az az $M_f \in \{0,1\}^{2^n \times 2^n}$, amelynek sorai x-szel, oszlopai y-nal vannak indexelve, és az x-hez tartozó sor y-hoz tartozó oszlopában $f(x,y)$ szerepel.

6.1.10. tétel (Mehlhorn–Schmidt)**.** $\kappa(f) \geq \log r(M_f)$*, ahol* $r(M_f)$ *az* M_f *mátrix rangját jelöli.*

6.1.11. *megjegyzés.* A fenti tételben, és a továbbiakban log mindig a 2 alapú logaritmus lesz.

Bizonyítás. Legyen P egy adott protokoll. Tegyük fel, hogy A kezd. Ekkor A kommunikál egy bitet. Ez rögzített P protokoll mellett bizonyos x-ekre 0, bizonyos x-ekre 1. Ezzel az M_f mátrixot két részre bontja: az egyik részben azon sorok vannak, amelyekre 0-t mond, a másikban azok, amelyekre 1-et. Ezek közül az egyik sorrangja $\geq \frac{1}{2} r(M_f)$.

Ezt ismételjük addig, amíg A lép. Amikor B lép, akkor ugyanez elismételhető oszloprangra, de egy mátrix sor- és oszloprangja megegyezik. Ha x és y olyan, hogy minden lépésnél a nagyobb rangú részmátrixot adják meg, akkor k lépés után a részmátrix rangja $\geq 2^{-k} r(M_f)$.

Tegyük fel, hogy a k. lépésben vége van a játéknak. Ekkor szimmetriaokokból feltehető, hogy A tudja $f(x,y)$-t, és B tudja, hogy A tudja. Mivel A tudja $f(x,y)$-t, az így kapott részmátrix minden sora homogén, azaz vagy csupa 0-t, vagy csupa 1-et tartalmaz. Ha pedig egy sor nem homogén, akkor A nem tudhatja biztosan $f(x,y)$-t. Hasonlóan, az, hogy B tudja biztosan $f(x,y)$-t, az azzal ekvivalens, hogy a kapott részmátrix minden oszlopa homogén.

Mivel homogén részmátrix rangja 1, az előbbi egyenlőtlenség szerint $1 \geq 2^{-k} r(M_f)$, azaz $2^k \geq r(M_f)$ fog teljesülni minden olyan (x,y) párra, amelyeket P k lépésben számol ki. □

6.1.12. következmény. *Innen könnyen kijön, hogy* $\kappa(\mathrm{ID}) = n$*, ugyanis* $M_{ID} = I_{2^n}$*, és* $r(I_{2^n}) = 2^n$*, tehát* $n \leq \kappa(\mathrm{ID})$ *a Mehlhorn–Schmidt-tétel miatt. Másrészt láttuk, hogy* $\kappa(f) \leq n$ *minden* f*-re, így* $\kappa(\mathrm{ID}) = n$.

6.1.13. *megjegyzés.* Felső becslés nem ismeretes $\kappa(f)$-re. Lovász és Suchs nevéhez fűződő sejtés szerint $\exists c > 0 : \kappa(f) \leq \log^c(r(M_f))$. Tudjuk, hogy $c > 2$ kell hogy teljesüljön. Ismert továbbá, hogy $\kappa(f) \leq r(M_f)$.

6.1.14. következmény. $DISJ(x,y) = \chi_{x \cdot y = 0}$*, a halmazdiszjunktsági feladat. Akkor erre is* $\kappa(DISJ) = n$.

Bizonyítás. Elemszám szerint rendezve az n elemű halmaz részhalmazait a sorokban, és a komplementereiket az oszlopokban $M_{DISJ} = \begin{bmatrix} 1 & * & * & \dots \\ 0 & 1 & * & \dots \\ 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$ felsőháromszög alakú, vagyis $\kappa(DISJ) = n$ □

6.1.15. definíció (f nemdeterminisztikus kommunikációs bonyolultsága)**.** Alíz ismeri x-et, Bob ismeri y-t, E.T. ismeri mindkettőt, és f-et is. Utóbbi meg akarja győzni a játékosokat, hogy tudja. Ezt egy bizonyítással teszi, amit függetlenül A-nak, és B-nek is el kell fogadnia. Egy fix E.T. által az (x, y) párra adott bizonyítás hossza, amikor azt akarja bizonyítani, hogy $f(x, y) = 1$ legyen $\kappa_1^{E.T.}(f(x, y))$. Legyen továbbá $\kappa_1^{E.T.}(f) := \max_{x,y:f(x,y)=1} \kappa_1^{E.T.}(f(x, y))$, végül $\kappa_1(f) = \min_{E.T.} \kappa_1^{E.T.}(f)$ a legjobb E.T. által a legrosszabb esetben adott bizonyítás hossza. Hasonlóan definiáljuk a $\kappa_0(f)$-et is.

6.1.16. *megjegyzés.* $\max \kappa_0(f), \kappa_1(f) \leq \kappa(f)$ teljesül, hiszen reprodukálhatja az adott esetben a protokoll által megszabott kommunikációját

6.1.17. *példa.* Ha $x \neq y$, akkor az $(i, x_i = 0)$ pár (ahol $y_i = 1$) megadása $\log(n) + 1$ bit hosszú, és bizonyítja, hogy az ID feladat nem teljesül. Egyenlőségre nem látszik kapásból hasonló jó bizonyítás.

6.1.18. tétel (A nemdeterminisztikus kommunikációs bonyolultság jellemzése fedő téglalapokkal)**.** *$\kappa_1(f)$ az a legkisebb t szám, hogy M_f egyesei lefedhetők 2^t darab csupa 1-es részmátrixxal*

6.1.19. *megjegyzés.* M_f-et már ismerjük, a kommunikációs mátrix. A tételben részmátrix alatt az oszlopok, és sorok egy-egy részhalmazait kiválasztva, a metszetekből álló részt értjük. Figyelem, ez nem feltétlenül egy összefüggő téglalap! $\left[\begin{smallmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{smallmatrix}\right]$-ben az első és utolsó sor, és oszlopok által meghatározott rész is egy ilyen csupa egyes részmátrix.

6.1.20. következmény. *Láttuk, hogy $M_{ID} = I_{2^n}$, ezt pedig csak úgy fedhetjük le csupa 1-es téglalapokkal, ha külön-külön kiválasztjuk az átlóelemeket. Következik, hogy $\kappa_1(ID) = n$.*

Bizonyítás. ($\kappa_1(f) \leq t$) Tekintsük a fedő téglalapokat. Alíznak van egy sora, Bobnak egy oszlopa. A protokollban megállapodnak a 2^t darab fedőmátrix egy sorrendjében. E.T. bizonyítása az lesz, hogy hanyadik részmátrixban van az (x, y) metszet, ez t bittel kódolható, leellenőrzik, hogy benne van-e az adatuk, és mivel ez csupa egyesből áll, így szükségszerűen $f(x, y) = 1$.
($\kappa_1(f) \geq t$) Legyen $H_\alpha = \{(x, y) :$ A-nál x, B-nél y van, és α üzenetet hallják, akkor elfogadják a bizonyítást$\}$. Ha $(x_1, y_1), (x_2, y_2) \in H_\alpha$, akkor $(x_1, y_2), (x_2, y_1) \in H_\alpha$, hiszen az α bizonyítást Alíz elfogadta (x_1, y_1)-re, az ő nézőpontjából semmi nem különbözteti meg a szituációt attól, mintha (x_1, y_2) lenne a felállás, ezt pedig Bob is elfogadja, hiszen számára (x_1, y_2), és (x_2, y_2) ugyanolyan, és ez utóbbit elfogadta α-ra. Következik, hogy minden α-ra H_α megfelel egy részmátrixnak. Ha E.T. legfeljebb t bitből bizonyítani tudja, hogy $f(x, y) = 1$, ez szolgáltat lazannyát és 2^t darab csupa egyes részmátrixot. □

Randomizálva azonban gyorsan is lehet a következő Simon és Rabin nevéhez fűződő protokollal. A generál egy véletlen p prímet $\in \{1, .., n^2\}$ (ahol $\log x, \log y \leq n$), és elküldi az ($x \bmod p, p$) üzenetet, B pedig leellenőrzi, hogy $x \equiv y \mod p$ teljesül-e, és ezt mondjuk százszor megismétlik.

Ha egyszer is az teljesül, hogy inkongruensek, akkor az eredeti számok sem lehettek egyenlőek, ha mindig kongruensek, és mégsem egyenlőek, akkor százszor teljesült az, hogy $p|x-y \neq 0$. $\leq 2^n$ számoknak legfeljebb n darab prímosztója lehet, és n^2-ig nagyjából $\pi(n^2) \sim \frac{n^2}{2\log(n)}$ darab prím van. Annak a valószínűsége, hogy egyszer teljesül a kongruencia

$$\mathbb{P}(p|x-y) \leq \frac{n}{\frac{n^2}{2\log(n)}} = \frac{2\log n}{n} \to 0$$

Egy kommunikáció $4\log n$ bitet küld, ergo összesen $400\log n$ bitnyi kommunikáció történik.

Nem (teljesen) triviális protokollok:

6.1.21. *példa.* Tekintsünk egy fagráfot, aminek van két részfája. Kérdés, hogy az n csúcsú T fa T_1, T_2 részfáinak van-e közös csúcsa. Alíz kapja T_1-et, Bob T_2-t értelemszerűen, és mindketten ismerik T-t. Ez eldönthető lenne a $DISJ$ játék speciális eseteként, de adunk egy okosabb protokollt. Alíz megmondja T_1 egy tetszőleges v csúcsát (ez ugye $\log n$ bit kommunikáció). Majd Bob kiszámolja T_2-ben a v-hez legközelebbi w csúcsot, mivel fában egyértelmű út van két csúcs között, ez értelmes. Ezt visszaküldi Alíznak, ellenőrzi, hogy $w \in T_1$, ha igen, ez metszetbeli, és készen vagyunk, ha nem, akkor azt mondja, hogy a két fa diszjunkt. Ugyanis, ha a legközelebbi w pont nem része a fának, de egy további u pont része lenne T_1-nek, az uw szakasz T_2-ben van, az uv szakasz pedig T_1-ben, vagyis u közelebb van v-hez, mint w.

6.1.22. *példa.* Most Alíz és Bob két részgráfot kap egy G gráfból úgy, hogy G_A független csúcsokból áll, G_B pedig egy teljes részgráf. Kérdés, hogy van-e metszet? Világos, hogy ha van, legfeljebb 1 pontból állhat.

- Alíz megnézi, hogy van-e legalább $n/2$ fokú v csúcs a gráfjában, ha igen, akkor $(1, v)$-t küldi el, ha nem, 0-t.
- Bob megnézi, hogy van-e $< n/2$ fokú w csúcs G_B-ben, ha igen, $(1, w)$-t küld, ha nincs, 0-t.

Ezek után Bob tudja, hogy G_A v-ből, és a nem-szomszédaiból áll, ez legfeljebb $n/2$ csúcsból áll, és iteratíven folytathatjuk ezt az eljárást amíg lehet. Ha Bob talál egy kis fokszámú w csúcsot, akkor az ő gráfjának a többi csúcsa ennek a szomszédai közül kerül ki, és ismét rekurzíven folytatható az eljárás. Mi történik, ha mindketten 0-t küldenek? Alíz gráfjában minden csúcs kisebb mint $n/2$ fokú, G_B-ben pedig minden csúcs legalább $n/2$ fokú, ez a két feltétel kizárja egymást, így a két gráf diszjunkt. Addig ismételgetik a fenti lépést, amíg nem mondanak mindketten nullát. Egy lépés $\log n + 1$ bit, és $\log n$ lépésben persze kimerítik a gráfot, vagyis $O(\log^2 n)$ bitre van összesen szükség.

6.1.23. tétel (Aho-Ullman-Yanakakis)**.** *Minden f-re $\kappa(f) \leq (2 + \kappa_0(f))(2 + \kappa_1(f))$*

6.1.24. lemma. *Ha M egy $0-1$ mátrix, H egy azonosan nulla részmátrixa, H sorai alkossák az A, oszlopai a B mátrixot, ekkor $\rho(A) + \rho(B) \leq \rho(M)$, ahol $\rho(M)$ a sor/oszloppermutációval képezhető legnagyobb négyzetes felsőháromszög részmátrix méretét jelölli, aminek a főátlója csupa 1-ből áll.*

Bizonyítás. A lemma azon múlik, hogy A és B-t külön-külön mozgathatjuk, a csupa nulla metszet nem fog változni, és a másik mátrixhoz nem nyúltunk hozzá, diszjunkt sorokból/oszlopokból áll. Egy permutációval megfelelő helyre visszük A-ban a maximális U_A felsőháromszög mátrixot, ezt B-ben is elvégezve (U_B) kapunk egy $\left[\begin{smallmatrix} U_B & \\ 0 & U_A \end{smallmatrix}\right]$ felsőháromszöget M-ben.

$$\begin{bmatrix} & B_1 & \\ A_1 & 0 & A_2 \\ & B_2 & \end{bmatrix} \to \begin{bmatrix} & B_1 & B_1 & & \\ & B_1 & U_B & & \\ A_1 & 0 & 0 & U_A & A_2 \\ A_1 & 0 & 0 & A_2 & A_2 \\ & B_2 & B_2 & & \end{bmatrix}$$

□

A-U-Y tétel bizonyítása. Világos, hogy $\rho(M_f) \leq r(M_f)$, és $\log \rho(M_f) \leq \kappa_1(f)$ teljesülnek, mert egy csupa 1 főátlójú felsőháromszög mátrix teljes rangú, illetve 6.1.18 miatt. Indukcióval belátjuk, hogy $\kappa(f) \leq (2 + \log \rho(M_f))(2 + \kappa_0(f))$. Ha $\rho(M_f) = 1$, akkor nem is kell kommunikálni, mert egy ilen mátrixban vagy csak egyesek állnak, vagy pontosan egy sorában vagy oszlopában vannak egyesek. Az általános lépésben tekintsük a kommunikációs mátrix nullásainak a fedését $2^{\kappa_0(f)}$ darab csupa nulla részmátrixszal. Alíz megnézi, hogy fedi-e az ő x inputjának egy részét olyan csupa 0 részmátrix, hogy a hozzá tartozó sorokból alkotott A mátrixra $\rho(A) \leq \rho(M_f)/2$, ha igen, akkor elküldi az (1, a csupa nulla részmátrix sorszáma) üzenetet, ez legfeljebb $1+\kappa_0(f)$ bit kommunikáció, ha nincs ilyen részmátrix, akkor 0-t küld. Bob hasonlóan megnézi, hogy van-e az y-jához olyan fedő csupa 0 mátrix, amely oszlopaihoz tartozó B mátrixra $\rho(B) \leq \rho(M_f)/2$, ha igen (1,a fedő mátrix sorszáma), ha nincs ilyen, akkor pedig 0-t küld. Mi történik, ha mindketten 0-t küldenek? Akkor $f(x, y) = 1$, hiszen ha 0 lenne, akkor a metszetüket lefedné egy csupa 0 részmátrix, de az eddigi kommunikáció szerint az ezen fedőmátrixhoz tartozó sorok, és oszlopok ρ értékei összesen többet adnak, mint $\rho(M_f)$, ellentmondásban a lemmánkkal. □

6.1.25. definíció. $f \in P^{CC}$, ha $\exists c > 0 : \kappa(f) \leq \log^c n$. $f \in NP^{CC}$, ha $\exists c > 0 : \kappa_1(f) \leq \log^c n$, és $f \in co - NP^{CC}$, ha $\exists c > 0 : \kappa_0(f) \leq \log^c n$

A fenti tétel következményeként adódik, hogy $P^{cc} = NP^{CC} \cap co - NP^{CC}$. Láttuk továbbá, hogy $NP^{CC} \neq co - NP^{CC}$, mert ID benne van a jobb oldalban, de a balban nincs. $P^{CC} \neq co - NP^{CC}$ szintén az ID miatt (így $P^{CC} \neq NP^{CC}$ is teljesül).