1. tétel

1. *32 bites szám kitalálása barkochbával, ennek optimalitása. N bites szám kitalálása barkochbával, ennek optimalitása. (Illusztráció: játék a géppel, Alexander Bogomolny.) Hazug barkochba. MAX, MIN, (MAX, MIN) megkeresése páronkénti összehasonlításokkal. A legjobb és a legrosszabb játékos kitalálásának optimalitása. A kavicsos konstrukció. Rendezés kitalálása barkochbával.*
   A legjobb és a második legjobb teniszjátékos kiválasztása.
   Ordó-jelölések. Rendezéshez legrosszabb esetben $\log n!$ *összehasonlítás kell.* $\log n!$ *becslései. Beszúrás, egy elem beszúrásához n elemű listába* $\lceil \log(n+1) \rceil$ *összehasonlítás kell, ennyi azonban elég is.*
   Rendezés beszúrásokkal $0\left(n \cdot \log n\right)$ *összehasonlítással. Összefésüléses rendezés. Két n hosszú sorozat 2n-1 összehasonlítással összefésülhető, és ez optimális is. Rendezés összefésüléssel.*

32 bites szám kitalálása barkochbával, ennek optimalitása

Tegyük fel, hogy minden kérdésért fizetni kell, ezért cél, hogy a legkevesebb kérdéssel kitaláljuk a számot. Legjobb esetben elég 1 kérdés.
Módszer legyen, hogy azt kérdezzük: „igaz-e, hogy az *i*. szám nem nulla?”. Ezt 32-szer megkérdezve biztosan kitaláljuk a számot.
Állítás: egy 32 bites szám 32 kérdéssel kitalálható, és ez optimális is a legrosszabb esetben.
Bizonyítás: a kitalálhatóság világos. Optimális, mert tegyük fel, mindig pont 31-et kérdezünk, azaz adott a 31 hosszú igen-nem-ek sorozatainak halmaza. Ennek elemszáma 2^{31}. A számok viszont 32 bitesek, ezért belőlük 2^{32} féle van. Ezek szerint nem lehet bijekciót képezni a két halmaz között, azaz 31 kérdésből nem található ki a legrosszabb esetben a szám.

N bites szám kitalálása barkochbával, ennek optimalitása

Tétel: N bites szám kitalálásához N kérdés elég, és ennyi kell is.
A bizonyítás analóg az előbbi esettel.
Tétel: egy N elemű H halmaz egy elemét $\lceil \log n \rceil$ kérdéssel tudom kitalálni, és ez optimális is.
Input: n szám
Modell: barkochba
Output: max(n szám)
Ilyen kérdéseket tehetek fel: „igaz-e, hogy $a_i<a_j$?”
Eszerint n-1 összehasonlítás kell és elég is.

Hazug barkochba

Ha a barátunk hazudik, akkor legfeljebb egyszer hazudik. Meg kell találni a helyes sorozatot (32 bites számot). Módszer lehet: mindent kétszer kérdezünk meg. Az egyik bitnél kétféle értéket fogunk kapni, de vajon melyik lehet a jó? Jobb módszer: végigkérdezzük a 32 kérdést, majd feltesszük neki azt, hogy „hazudtál az első 32-ben?”. Ha a válasz igen, és ez igaz, akkor nem tudjuk, hogy hol hazudott, tehát ez néha nem ad jobb megoldást az előbbinél.
Jobb módszer: megkérdezzük a 32 kérdés után, hgoy hazudott-e. Ezt kétszer kell megkérdezni, mert lehet, hogy az utolsó kérdésnél hazudik. Ha az derül ki, hogy eddig még nem hazudott (ellentmondó válaszokat ad az utolsó kérdésre), akkor készen vagyunk, és megvan a szám. Ha az derül ki, hogy már korábban hazudott, akkor logaritmikus kereséssel meg lehet keresni, hogy melyik kérdésnél, és lehet javítani a számjegyet. A logaritmikus

keresés 32 bit esetén 5 lépésben megadja a keresett helyet, ezért 32 bites szám esetén 39 kérdést kell feltenni.

MAX, MIN, (MAX, MIN) megkeresése páronkénti összehasonlításokkal

Bemenet n db egész szám, a kimenet a MAX. A maximum megtalálásához elég n-1 összehasonlítás, de ennyi kell is. Bizonyítás: teniszjátékosok közül a legjobb kiválasztásához legalább n-1 mérkőzés kell. A mérkőzések legvégén van egy legjobb, és van n-1, akik nem a legjobbak, tehát egyszer legalább vesztettek. Eszerint van legalább n-1 mérkőzés, ahol ez az n-1 játékos veszíthetett. A MIN megkeresése összehasonlításokkal ugyanennyi lépést igényel.
Bemenet n db egész szám, a kimenet a (MAX, MIN) pár. Ötletek: a maximum megtalálásához elég n-1 összehasonlítás. Ha viszont megvan a maximum, a minimum megtalálásához elég n-2 összehasonlítás. Együtt ez 2n-3 összehasonlítást jelent. Állítás: van jobb: $\frac{3}{2}n-2$.

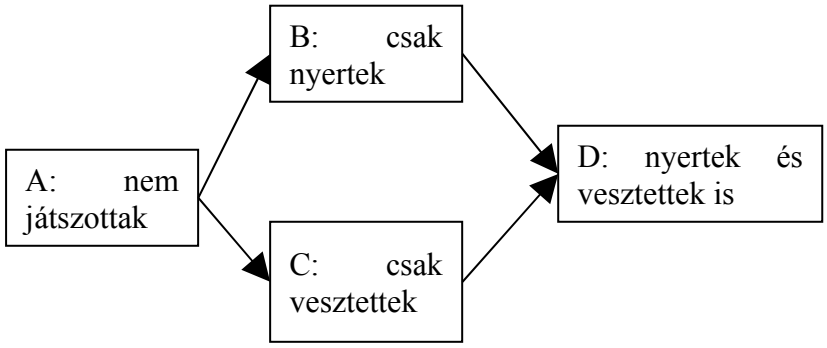
Ha páronként összehasonlítok számokat, akkor a számok két osztályba kerülnek, lesz a nagyobbak osztálya és a kisebbeké. Ha páratlan sok szám van, akkor az egyik szám mindkét osztályban benne lesz. Ekkor a nagyobbak osztályában megkeresem a maximumot, ami $\left\lceil \frac{n}{2} \right\rceil - 1$ összehasonlítással megvan, ugyanígy a kisebbek osztályában a minimumot is megkeresem. Az összes összehasonlítás páros sok szám esetén $n+2\cdot\left(\frac{n}{2}-1\right)$, páratlan sok esetén pedig $\left\lfloor \frac{n}{2} \right\rfloor + 2\cdot\left(\left\lceil \frac{n}{2} \right\rceil - 1\right)$. Vagyis $\frac{3}{2}n-2$ összehasonlítással megtalálom a minimumot és a maximumot is egyszerre.

A legjobb és a legrosszabb játékos kitalálásának optimalitása

A kavicsos konstrukció

Tétel: $\frac{3}{2}n-2$ összehasonlítás kell a (MAX, MIN) pár kiszámításához.

Bizonyítás:



Kezdetben mindenki az A-ban van, a játék végekor pedig senki nincs A-ban, B-ben és C-ben 1 van, D-ben n-2.
Az elején mindenki kap három kavicsot, amikor valaki elhagyja A-t, befizet 1-et, ha elhagyja B-t vagy C-t, akkor 2-t. Így a játékban összesen 4 kavics marad (mert 1-1 ember van a B-ben

és a C-ben), a többi a bírónál (3n-4). A bíró minden mérkőzésen legalább két kavicsot kap, ezért legalább $\frac{3n-4}{2}$ mérkőzés kell.

Rendezés kitalálása barkochbával

Elég olyan kérdéseket feltenni, hogy az i-edik elem kisebb-e, mint a j-edik elem. Beszúrásos rendezés esetén elég megkeresni az ő helyét.
0 hosszú rendezett sorozatban 1 helyre lehet tenni az elemet, és ezt log1 kérdéssel meg lehet tudni..
1 hosszú rendezett sorozatban 2 helyre lehet tenni az elemet, és ezt log2 kérdéssel meg lehet tudni.
2 hosszú rendezett sorozatban 3 helyre lehet tenni az elemet és ezt log3 kérdéssel meg lehet tudni.
…
n-1 hosszú rendezett sorozatban n helyre lehet tenni az elemet és ezt log(n) kérdéssel meg lehet tudni.

Összesen tehát kell $\sum_{i=1}^{n} \lceil \log i \rceil = \log n!$ kérdés.

A legjobb és a második legjobb teniszjátékos kiválasztása

Input: n db természetes szám
Output: az első két legnagyobb szám
Maximumkereséssel a legnagyobb n-1 összehasonlítással, a második legnagyobb n-2 összehasonlítással megadható. Ez 2n-3 összehasonlítás. Van jobb.
Jobb módszer a tenisz-torna rendszere, ahol első körben két-két szomszédos játékos játszik. Második körben azok játszanak, akik győztek már, stb. A torna végén kialakuló háromszög csúcsában van a legjobb játékos. Ő $\lceil \log n \rceil$ másik játékost vert meg, ezért $\lceil \log n \rceil - 1$ összehasonlítással megadható a második legjobb is. Összesen ezért $(n-1) + (\lceil \log n \rceil - 1) = n + \lceil \log n \rceil - 2$ összehasonlítás kell.

Ordó-jelölések

Θ-jelölés: aszimptotikusan éles korlát. Informálisan: el kell dobni az alacsonyabb rendű tagokat, és figyelmen kívül kell hagyni a legmagasabb rendű tag főegyütthatóját.
O-jelölés: ha csak egy aszimptotikus felső korlát van, akkor használjuk ezt. Egy állandó tényezőtől eltekintve felső korlátot ad. Minden bemenetre felső korlátot jelent az algoritmusnak.

$f, g : N \to R^+$

$f = O(g) \Leftrightarrow \exists c > 0 : \forall n : cg(n) \geq f(n)$

$f = \Omega(g) \Leftrightarrow g = O(f) \Leftrightarrow g(n) \leq cf(n) \Leftrightarrow \frac{g(n)}{c} \leq f(n)$

Ω-jelölés: aszimptotikus alsó korlát.

Rendezéshez legrosszabb esetben $\log n!$ összehasonlítás kell. $\log n!$ becslései

Input: n szám
Output: az n szám növekvően rendezve.

Ekkor $\lceil \log n! \rceil$ kérdés elég és kell. Ugyanis

$\log n! \leq n \cdot \log n$

$\log n! = \log 1 + \log 2 + \ldots + \log n \leq \log n + \log n + \ldots + \log n$

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \ldots \frac{x^n}{n!} + \ldots$$

$$e^n = \frac{n^0}{0!} + \frac{n^1}{1!} + \frac{n^2}{2!} + \ldots \frac{n^n}{n!} + \ldots$$

$$e^n > \frac{n^n}{n!}$$

$$n! > \left(\frac{n}{e}\right)^n$$

$n \cdot \log n - n \cdot \log e \leq \log n! \leq n \cdot \log n$

Tehát tényleg $\lceil \log n! \rceil$ kérdés kell. Ha valaki megsúgná a rendezést, akkor n-1 kérdés elég volna.

Vödör-rendezés

Input. Egész számok, például 1 és 20n közötti méretkorláttal. Az i értékűt az i-edik vödörbe rakom.

1 Ft-ot fizetünk 1 szám olvasásáért és írásáért is, valamint két szám összehasonlításáért. A vödrökben keresés logaritmikusan mehet.

Beszúrás, egy elem beszúrásához n elemű listába $\lceil \log(n+1) \rceil$ összehasonlítás kell, ennyi azonban elég is

Egy szigorú monoton növekvő sorozatba kell beszúrni egy olyan elemet, ami még nem szerepel benne. m elem esetén m+1 helyre tehetjük az újat, ezért $\lceil \log(m+1) \rceil$ kérdésből kitalálhatjuk a helyét, ha bármit kérdezhetünk.

A kezdetben üres sorozatba n elemet beszúrunk. Ez $\lceil \log 1 \rceil + \ldots \lceil \log n \rceil = \lceil \log n! \rceil \geq \log n!$ összehasonlítást jelent.

Rendezés beszúrásokkal $0(n \cdot \log n)$ összehasonlítással

Az üres sorozat n páronként különböző elemmel történő feltöltése $\lceil \log 1 \rceil + \ldots \lceil \log n \rceil \leq n \cdot \log n$ összehasonlítással megoldható.

Összefésüléses rendezés

Alaplépés: összefésülés. 2 rendezett sorozatot kell összefésülni. Az eredménysorozat hossza a bemenő két sorozat hosszának összege, legyen m+t. Ekkor összesen m+t-1 összehasonlítás biztosan elég, és nem lehet kevesebbel.

Az összefésülés lépése: nyilvántartom a két sorozatban az aktuális elemet. Kiválasztom a két aktuális közül a kisebbiket, és átírom az eredménysorozatba, majd léptetem az átírt elemet tartalmazó sorozat aktuális elemét. Így m+t-1 összehasonlítást kell elvégezni.

Két n hosszú sorozat 2n-1 összehasonlítással összefésülhető, és ez optimális is

Ha m=t=n, akkor világos.

Rendezés összefésüléssel

Adott n elemű sorozat. A szomszédosokat összefésülöm, majd az így kapott 2 hosszú sorozatokat vizsgálom. A szomszédos 2 hosszú sorozatokat összefésülöm, majd az így kapott 4 hosszú sorozatokat vizsgálom. És így tovább, amíg pontosan 1 rendezett sorozatom nem lesz. (Természetes, hogy a „jobb szélen” lehetnek az aktuális lépésben megadott 2-hatványtól eltérő elemszámú sorozatok is, hiszen nem biztos, hogy a kiindulási sorozat pontosan 2-hatvány elemszámú volt.) Minden lépésben legfeljebb n összehasonlítást végzünk, viszont $\log n$ összehasonlítás-sorozattal eljutunk a rendezett sorozatig, ezért az összes összehasonlítások száma $n \cdot \log n$.

3. tétel

1. *Középső elem megtalálása, a Floyd-Rivest algoritmus. Karucuba-Ofman algoritmus két nagy szám szorzatára. Strassen mátrix-szorzása.*

Középső elem megtalálása, a Floyd-Rivest algoritmus

Input: n különböző szám.

Output: k. legkisebb szám (speciális eset: $k = \left\lfloor \frac{n}{2} \right\rfloor$).

Az algoritmus lényege: szétbontjuk a bemenetet meghatározott elemszámú halmazokra, majd ezeket a halmazokat rendezzük, és a középső elemeket összehasonlítjuk. Az így kapott középső elemek *m* mediánsa szerint bontsuk kétfelé az összes számot, az ennél kisebbekre, és az ennél nagyobbakra: $S_{\le m} = \{a_i | a_i \le m\}$, $S_{> m} = \{a_j | a_j > m\}$. Ha $S_{\le m}$ elemszáma nagyobb *k*-nál, akkor a megoldás a kisebbek rendezett halmazának *k*. eleme. Ha nem éri el a *k*-t, akkor a nagyobbak között lesz, mégpedig a rendezett $S_{> m}$ halmaz $k - |S_{\le m}|$. eleme.

Például, legyen adott az n, és bontsuk fel 5-ös csoportokra. Ekkor minden csoportban 8 összehasonlítással lehet rendezni, és megtalálni a középső elemet. Az 5-ös csoportok középső elemeinek mediánsa legyen *m*, és legyen $S_{\le m} = \{a_i | a_i \le m\}$ és $S_{> m} = \{a_j | a_j > m\}$ a két halmaz (ez m-1 összehasonlítás).

Állítás: $\frac{3n}{4} \ge |S_{\le m}|, |S_{> m}| \ge \frac{n}{4}$

Bizonyítás: Mindkét halmazba az *n* elem közül legalább kétötöd fog kerülni, ha 5-ös csoportokra osztottuk. Ez viszont több, mint az *n* negyede.

Végrehajtási idő: ha $n \le 50$,

$$t(n) \le \frac{n}{5} \cdot 8 + t\left(\frac{n}{5}\right) + n - 1 + t\left(\frac{3n}{4}\right) \le \frac{8n}{5} + 4cn + n + 15cn = 19cn + \frac{8n}{5} + n \le 20cn$$

$$\frac{13}{5} \le c$$

$$t(n) \le 20 \cdot \frac{13n}{5} = 52n$$

Ahol $t\left(\frac{n}{5}\right) \le 20c\frac{n}{5} = 4cn$, és $t\left(\frac{3n}{4}\right) \le 20c\frac{3n}{4}$ és $t_k(n)$: az algoritmus lépésszáma n elemből a k. legkisebbre, valamint $t(n) := \max_{i=1}^{n} t_k(n)$. Azt akartuk megmutatni, hogy $t(n) \le 20cn$.

Ha $n \le 50$, akkor kevesebb, mint 6n összehasonlításra van szükség, ezért $c \ge \frac{6}{20}$ jó becslés c-re.

Karucuba-Ofman algoritmus két nagy szám szorzatára

Input: két tízes számrendszerbeli szám, *a* és *b*. Output: a két szám szorzata.
Modell: az összeadás ingyen van, két számjegy szorzása viszont 1Ft.
Cél, minél olcsóbban kiszámolni a szorzatot!
Legyen 2n a hosszabbik szám hossza.

$a = 10^n a' + a''$

$b = 10^n b' + b''$

$ab = (10^n a' + a'')(10^n b' + b'') = 10^{2n} a'b' + 10^n (a'b'' + a''b') + a''b''$

Ennek költsége összesen $4n^2$ Ft.

Ha kiszámoljuk $(a' - a'')(b' - b'') = a'b' + a''b'' - (a'b'' + a''b')$-t, $a'b'$-t és $a''b''$-t, akkor megkapjuk összeadásokkal $(a'b'' + a + b')$-t is, így ennek a költsége $3n^2$ Ft.

t(k):=a 2 db k hosszú szám összeszorzásának költsége.

t(1)=1, t(n)=?

$$t(2n) = 3t(n) = 3^2 t\left(\frac{n}{2}\right) = 3^3 t\left(\frac{n}{4}\right) = \ldots = 3^{s+1} t\left(\frac{n}{2^s}\right) = 3^{s+1} = \left(2^{\log_2 3}\right)^{s+1} = 2^{(s+1)\log_2 3} = (2n)^{\log_2 3}$$

$\log_2 3 \approx 1{,}58$

Így $(2n)^2$ helyett $(2n)^{1,58}$ lett.

Felhasználás: polinomok szorzására, vagy négyzetes mátrixok szorzására.

n×n-es mátrixok szorzásának ilyen költsége n^3 Ft, mert 1 elem kiszámolás n szorzásba kerül, vagyis n Ft, és van n^2 elem.

Strassen mátrix-szorzása

A mátrixszorzás visszavezethető egyszerűbb műveletekre is.

2×2-es mátrix esetén:

$$m_1 = (a_{12} - a_{22})(b_{21} + b_{22})$$
$$m_2 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$m_3 = (a_{11} - a_{21})(b_{11} + b_{12})$$
$$m_4 = (a_{11} + a_{12})b_{22}$$
$$m_5 = a_{11}(b_{12} - b_{22})$$
$$m_6 = a_{22}(b_{21} - b_{11})$$
$$m_7 = (a_{21} + a_{22})b_{11}$$

és így

$$c_{11} = m_1 + m_2 - m_4 + m_6$$
$$c_{12} = m_4 + m_5$$
$$c_{21} = m_6 + m_7$$
$$c_{22} = m_2 - m_3 + m_5 - m_7$$

.

Ezzel a felbontással 8 szorzás helyett 7 szorzást kell csak elvégezni.

2-hatvány oldalú négyzetes mátrixok szorzása esetén is elég lesz a kevesebb szorzás, mert részmátrixok szorzásával is megoldható a feladat.

Futási idő:

t(k):=a 2 db k×k-as mátrix szorzásának költsége Strassen algoritmusával.

t(1)=1

k=2^s esetén: $t(k) = 7t\left(\frac{k}{2}\right) = 7^2 t\left(\frac{k}{4}\right) = \ldots = 7^s t\left(\frac{k}{2^s}\right) = 7^s = \left(2^{\log_2 7}\right)^s = 2^{s\log 7} = k^{\log 7}$.

4. tétel

2. *Dinamikus programozás: maximális intervallum-összeg lineáris időben. Egy 0-1 mátrixban a legnagyobb egybefüggő, négyzet alakú, csupa-1-es részmátrix megtalálása lineáris időben*. Mátrix-szorzás optimális zárójelezése, a naiv algoritmus és a dinamikus programmal elérhető lépésszám összehasonlítása.

Dinamikus programozás

Valamilyen táblázatot kitöltünk, és az újabb értékeket a már kiszámoltakból és az inputból számoljuk ki.

Maximális intervallum-összeg lineáris időben

Input: n db egész szám, output: maximális intervallumösszeg ($\max_{1 \le i \le j \le n} \left(\sum_{k=i}^{j} a_k \right)$).

Állítás: Van lineáris sok összeadást használó megoldás.
b_i:=az a_i jobbvégpontú intervallumok közül a maximális összegűnek az összege.
Megjegyzés: A 0 hosszú intervallum összege 0, és jobbvégpontja bárhol van!
Miért jó a b_i? Mert az output a max(b_i).
$b_1 := \max(0, a_1)$ és tegyük fel, hogy i-1-ig ismerjük a b-ket. Ekkor $b_i := \max(b_{i-1} + a_i, 0)$, vagyis megjegyezzük, hogy melyik volt az eddigi maximum.

Egy 0-1 mátrixban a legnagyobb egybefüggő, négyzet alakú, csupa-1-es részmátrix megtalálása lineáris időben

Input: Egy 0-1 mátrix. Output: a legnagyobb egybefüggő csupa 1-es részmátrix.
Ha az input mérete n^2, akkor n^3 ellenőrizendő részmátrix van, így a futási idő n^5, ami nem lineáris idejű.
A lineáris megoldás: legyen egy $n \times n$-es mátrix, aminek a b_{ij} eleme azt mondja meg, hogy az inputmátrixban az a_{ij} jobb-alsó csúcsú négyzet alakú egybefüggő csupa 1-es részmátrixok közül a legnagyobb oldalmérete mekkora, ha a_{ij} 1es, különben 0. Viszont b_{ij} kiszámolható dinamikusan a már ismert adatokból: $b_{ij} := \begin{cases} 0 \\ 1 + \min(b_{i,j-1}, b_{i-1,j-1}, b_{i-1,j}) \end{cases}$.
Ez cn^2 lépés alatt megoldja a feladatot, ami az inputmérethez képest lineáris.

Mátrix-szorzás optimális zárójelezése, a naiv algoritmus és a dinamikus programmal elérhető lépésszám összehasonlítása

Input: $n_1, \ldots, n_{m+1}$, m db mátrix méretei, és a mátrixok ($A_i \in T^{n_i \times n_{i+1}}$)
Output: az optimális zárójelezés, a szorzás költsége. Cél, hogy a költség minimális legyen.
Mikor jó a mátrixszorzás? Akkor jó, hogyha kevés műveletet kell elvégezni. A mátrixszorzás asszociatív, ezért tetszőlegesen zárójelezhető.
Naiv algoritmus: brute force, vagyis elkészítem az összes fajta zárójelezést, és kiválasztom közülük a jót. m db mátrix szorzása esetén legalább $2^{\frac{m}{3}}$-féleképpen lehet zárójelezni. Ez exponenciális, tehát a futási idő miatt nem érdemes vele foglalkozni.
Dinamikus programmal viszont egy gyors megoldást kapunk.

$m_{i,j} := A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$ szorzat kiszámításának optimális költsége. Minket $m_{1,m}$ érdekel. $m_{i,i} = 0$, $m_{i,j} := \min_{i \le k \le j}\left(m_{i,k} + n_i \cdot n_k \cdot n_j + m_{k+1,j}\right)$.

A legkisebb indexkülönbségűeket szorzom össze először, és ezt ismétlem. A műveletek száma $\frac{m^2}{2} \cdot n \cdot m = cm^3$, azaz $0\left(n^3\right)$-ös algoritmus.

5. tétel

1. *A hátizsák-probléma, megoldása dinamikus programmal. Apró értékek esetén az algoritmus polinomiális.*

A hátizsák-probléma, megoldása dinamikus programmal

Input: $\begin{matrix} v_1,\ldots,v_m \\ s_1,\ldots,s_m \end{matrix}$ értékek és súlyok, valamint *b* a hátizsák teherbírása. Output: a hátizsákba kerülő kövek indexei, valamint az összsúly.
Speciális eset, ha minden súly egyenlő. Ekkor az érték szerint csökkenő sorrendbe kell helyezni, és a legnagyobb értékűtől kezdve addig pakolunk, amíg meg nem telik a hátizsák.
Ha minden érték egyenlő, akkor súly szerint növekvő sorrendbe rendezzük a köveket, és a legkisebb súlyútól kezdve addig pakolunk a zsákba, amíg fér.

Egyébként készítünk egy mátrixot, aminek sorai 1-től m-ig indexeltek i-vel, és 0-tól $\sum_{k=1}^{m} v_i$ -ig j-vel.
És ide még jön a táblázat elemeinek megalkotása

Apró értékek esetén az algoritmus polinomiális

Ha az s_i-hez képest a v_i kicsi, akkor ez egy jó algoritmus. Pl. $\log s_i \sim v_i$
Létezik a feladatnak javítása, ami megközelítően optimális, de sokkal gyorsabb.

6. tétel

2. Adatok tárolása: láncolt lista és tömb. *Beszúrásos rendezés láncolt listán és tömbön. A kupac fogalma; DELETEMIN, INSERT. Williams és Floyd kupac-rendezése* $0(n\log n)$ *művelettel. Gráfok ábrázolása. Gráf-algoritmusok: Szélességi gráf-bejárás, ezzel komponensek meghatározása, összefüggőség-vizsgálat, feszítő-erdő megadása.*

Adatok tárolása: láncolt lista és tömb

Beszúrásos rendezés láncolt listán és tömbön

Tömbön a beszúrás helyének keresése $\lceil \log(k+1) \rceil$ lépésben adódik, a beszúrás előtt pedig a nála nagyobb elemeket el kell tolni eggyel, ami $\log k + \frac{k}{2}$ lépést jelent. Az összese elem beszúrása $\log n! + \frac{n(n+1)}{4} = 0(n^2)$ lépésből van meg.

Láncolt lista esetén nincs szükség mozgatásra, tehát magának a beszúrásnak a költsége konstans. Viszont sokba kerül a hely megtalálása, ez $0(n^2)$ ideig tart.

A kupac fogalma

Def: Kupacnak nevezünk egy teljes bináris fát, amelyből a levelek egy részét jobb oldalról, alulról kezdve letöröltük. A fa csúcsaiban számok vannak, amelyeket néha kulcsnak is nevezünk. A kupacban levő számokra teljesül, hogy a szülő kulcsa kisebb vagy egyenlő, mint a gyerekek kulcsa. Műveletek: INSERT, DELETEMIN.

INSERT

Alsó szinten legbaloldalibb helyre berakok egy levelet (vagyis az utolsó szinten kiegészítem a sort egy újjal, így igaz marad, hogy ez majdnem teljes bináris fa).
Ráteszem az új számot.
Javítom a kupacot, vagyis addig cserélgetek a szülővel, amíg a szülő nagyobb, mint a beszúrt elem. Ezzel az új elem felbilleg a helyére.
Az insert műveletigénye a kupacmélységgel arányos, tehát O(kupacmélység).
n csúcsú kupac esetén ez O(log *n*).

DELETEMIN

Kitöröljük a gyökérből az ott lévő számot (ez a legkisebb a kupacban).
Az „utolsó” levél tartalmát átírjuk a gyökérbe, kitöröljük az „utolsó” levelet, és javítjuk a kupacot, azaz összehasonlítjuk a két gyereket, és a kisebbik gyerekkel kicseréljük. Addig cserélünk a gyerekekkel, amíg helyre nem áll a kupac-tulajdonság.
Költsége O(log *n*).

Williams és Floyd kupac-rendezése $0(n\log n)$ *művelettel*

Egy n elemű kupac INSERT-ekkel felépítése n-szer O(log *n*), vagyis $0(n\log n)$.
Egy n elemű kupacból DELETEMIN-nel lehet rendezett sorozatot adni n-szer O(log *n*) lépéssel, vagyis $0(n\log n)$.

Gráfok ábrázolása

Adjacencia-mátrix

Négyzetes táblázat, a sor és oszlop-azonosítók a gráf csúcsai. A cellákban számok vannak. Ha az (i, j)-edik helyen 1-es van, azt jelenti, az i. és a j. csúcs között van él. Ha 0 van, akkor azt jelenti, nincs él.

Incidencia-mátrix

Táblázat, melynek sor-azonosítói a csúcsok, oszlop-azonosítói pedig az élek. Egy oszlopban két 1-es van, a két végpontnál.

Él-lista

Olyan láncolt lista, amelyben egyrészt a csúcsok egymáshoz vannak láncolva „függőlegesen”, másrészt a csúccsal szomszédos többi csúcs is hozzá van láncolva egy adott csúcshoz, de „vízszintesen”. Az élek a „vízszintes” láncolásból derülnek ki.

Szélességi gráf-bejárás, ezzel komponensek meghatározása, összefüggőség-vizsgálat, feszítő-erdő megadása.

Szélességi keresés vagy bejárás esetén input: gráf, output lehet annak eldöntése, hogy a gráf összefüggő-e; mennyi a komponensek száma; melyek a komponensek; adott pontból az összes többi pont távolsága mennyi; gráf párosságának eldöntése; feszítőfa-keresés.

7. tétel

3. *Egy gráfnak exponenciálisan sok feszítőfája is lehet. Minimális költségű feszítőfa keresése. Prim algoritmusa, kupac-implementáció.* Dijkstra algoritmusa szemléletesen: golyók és fonalak. *Dijkstra algoritmusa egy pontból kiinduló minimális költségű utak kiszámítására, kupac implementáció.*

Egy gráfnak exponenciálisan sok feszítőfája is lehet

Áll: nagyon nagy lehet a feszítőfák száma. Például tekintsük azt a gráfot, ami egy abroncs-hoz hasonlít, az élek a küllők, és az abroncs maga, a csúcsok az abroncs pontjai, és a tengely. n csúcs esetén 2^{n-1}-1 db feszítőfa biztosan van, ez exponenciálisan sok.

Minimális költségű feszítőfa keresése

Input: G gráf, V csúcshalmaz, E élhalmaz, $w: E \to N$ súlyfüggvény.
Output: $\{\}$, ha G nem összefüggő, különben a minimális súlyú feszítőfa élei.

Prim algoritmusa, kupac-implementáció

Ez egy mohó algoritmus.
v_0-ból indulok ki, mint kezdőcsúcs, a T halmazban kezdetben ez az elem van csak.
T-hez hozzáveszem a T és nem T-beli csúcsokat összekötő élek közül a legolcsóbbat.
Ha van még csúcs T-n kívül, akkor folytatom az előző lépéssel.
Áll: Ezzel feszítőfát kapunk, ha a gráf összefüggő, sőt a minimális súlyú feszítőfát.
Biz: indirekt tegyük fel, hogy nem azt kaptunk. Legyen T a Prim algoritmus által talált fa. Ha van ennél kisebb súlyú, akkor van olyan is, amelynek a legkisebb a súlya. Ha több ugyanilyen legkisebb súlyú is van, akkor van egy olyan is, amelyiknek a legtöbb közös éle van T-vel. Legyen ez T', és T'<>T, ezért T'-ben van olyan él, ami T-ben nincs benne, és fordítva.
Van egy olyan $e \in T - T'$ él, amelyet az ilyen tulajdonságú élek közül először vettem hozzá T-hez. Mivel *e* nincs benne T'-ben, T'-höz hozzávéve az új T'-ben lesz kör. Ha a régi T'-ben egy élet kicserélek egy másik élre, akkor nem lesz nagyobb a költsége, viszont több közös éle lesz T-vel, ami ellentmondáshoz vezet.
Lépésszám-becslés: Jelölje D(i) a $v_i \notin T$ csúcs távolságát az épülő fától, azaz $D(i) = \min_{x \in T} w(x, v_i)$. A nem létező él súlya végtelen nagy.
n-k számból számolunk minimumot: O(n-k-1) művelet kell, ezért az összes műveletigény (k megy 1-től n-1-ig) O(n^2) a minimumok kiválasztásához. Ehhez kell még egy O(n)-es művelet, a csúcsok lekezelése, és $0(|E|)$ az élek hozzávétele miatt. Mindegyik közül az n^2 a legmagasabb fokú, ezért a teljes műveletigény O(n^2).
Ha a T-hez hozzáveszek egy új y csúcsot, akkor D(i)=min(D(i),w(y,v_i)) lesz.
Kupac
Új műveletre van szükség: DECREASE_KEY, vagyis kulcs-csökkentés. Ez egy O(log n)-es művelet. Csökkentem a kulcsot, majd rendezem a kupacot.
Alkalmazása: D(i)-ket berakom a kupacba. Kitörlöm a minimumot (ez megy be a feszítőfába), majd módosítom a kulcsokat a DECREASE_KEY-vel.
$n \cdot DELETE\_MIN + |E| \cdot DECREASE\_KEY = 0(n \log n) + |E| \cdot 0(\log n)$.
Lépésszám-becslés:
Kupac nélkül $0(n^2)$.
Kupaccal: $0(n \log n) + |E| \cdot 0(\log n)$

Dijkstra algoritmusa egy pontból kiinduló minimális költségű utak kiszámítására, kupac implementáció

Legyen $S = \{v_0\}$. Ekkor $v_i \in V - S$-re D(i) legyen a legrövidebb olyan út hossza, amely az utolsó élét kivéve teljesen az S-beli csúcsok között fut. S legyen a lebegő csúcsok hossza.
Vétessék fel a D(i)-k minimuma a j indexnél, azaz $D(j) := \min(D(i) : v_i \in V - S)$, és $S := S \cup \{v_j\}$. Továbbá $D(i) := \min(D(i), D(j) + w(v_i, v_j))$.
Áll: Az a D(i), amelyikből éppen v_i S-be vétele előtt számoltunk ki, megadja a legrövidebb v_0-v_i út hosszát.
Biz: Azt kell megmutatni, hogy tényleg a legrövidebb olyan út hosszát számolja ki, amely az utolsó élét kivéve teljesen az S-beli csúcsok között fut.
Akkor lenne hamis, ha a v_j rajta lenne a legrövidebb úton, de nem a legutolsón lenne rajta.
n-1 db DELETE_MIN + |E| db DECREASE_KEY: $0(n \log n) + |E|0(\log n)$ kupacban, $0(n^2)$ a minimumszámítás + $0(|E|)$

8. tétel

4. *Páros gráfok. Egy gráf párosságának eldöntése. Párosításkeresés páros gráfban: alternáló utak. Lépésszám-becslés. Maximális méretű párosítás, teljes párosítás.*

Páros gráfok

Def: Egy gráf akkor páros, ha csúcsai két osztályba oszthatók úgy, hogy csak az osztályok között mennek élek.

Egy gráf párosságának eldöntése

0-t és 1-et írok a csúcsokra. A 0-s csúcs szomszédjai 1-esek, az 1-es csúcs szomszédjai 0-k legyenek. Ha 1-et és 0-t is kéne írni, akkor meg kell állni, mert biztosan nem páros lesz. Következmény, hogy csak páros hosszú kör lehet egy páros gráfban.

Párosításkeresés páros gráfban: alternáló utak
Maximális méretű párosítás, teljes párosítás

Probléma: lovagok és hölgyek vannak, tudjuk, hogy melyik lovagnak ki tetszik. A lovag csak azzal táncolhat, aki tetszik neki. Megoldás: páros gráf, ha a lovagnak tetszik a hölgy, akkor van él. Kell, hogy legyen több hölgy, mint lovag, vagy legalább annyi.
Def: Egy G gráfban van **teljes párosítás**, ha van egy olyan G' részgráfja G-nek, hogy G'-ben (vagy G-ben?) minden csúcs foka 1.
Def: **Maximális méretű** egy párosítás, ha vagy teljes, vagy pedig nincs olyan párosítás a gráfban, amely több csúcsot fedne le.
Feladat: input: egy gráf, amely páros, output: maximális méretű párosítás.
Def: Legyen G egy páros gráf és M egy akármilyen párosítás G-ben. Ekkor **alternáló út**nak nevezünk egy olyan utat G-ben, amely mindkét végpontja M-mel nem lefedett csúcs és minden második éle M-ben van.
Lemma: A G gráfban az M párosítás maximális méretű, akkor és csak akkor, ha G-ben nincs alternáló út.
Biz: ➔ Tegyük fel, van egy alternáló út. Az alternáló úton cserélünk. Ami benne volt, az nem lesz benne, ami nem volt benne, az benne lesz.
← Indirekt tegyük fel, hogy M nem maximális méretű, tehát van nagyobb, amit jelöljön M'. Ezek szerint vannak élek, amelyek M-ben és M'-ben is szerepelnek (A). Vannak olyan élek, amelyek M'-ben szerepelnek, de M-ben nem (B). Vannak olyan élek is, amelyek M-ben szerepelnek, és M'-ben nem (C).
Ha a B-beli csúcsok nem lennének M-mel lefedve, akkor alternáló út adódna, ezért ezek is le vannak fedve M-beliekkel. Ugyanannyian vannak M-beliek, mint M'-beliek a B-ben. És van még néhány M-mel lefedett. Következésképp legalább annyi M-beli van, mint M'-beli. Így ellentmondásra jutottunk, mert az volt az állítás, hogy M'-ben többen vannak.
Cél: alternáló utakat keresünk a páros gráfhoz, és ha már nem találunk, akkor a maximális méretű párosításunk megvan (a lemma szerint).
Módszer: irányított gráfon szélességi keresés. A párosításban levők felfelé irányítottak, a többiek lefelé. Ha így egy fedetlenből egy másik fedetlenbe el tudok jutni, alternáló utat kapok.
2 mesterséges csúcsot kell felvenni fent és lent, ezek legyenek A és B. A-hoz és B-hez csak azokat kötöm hozzá, amik nincsenek lefedve. Ha le tudok jutni A-ból B-be, akkor találok alternáló utat.

Lépésszám-becslés

A szélességi bejárás $0\left(n^2\right)$ lépésű. Ezzel legfeljebb eggyel tudom növelni a párosítást. $\frac{n}{2}$ növeléshez legfeljebb $0\left(n^3\right)$ lépés kell.

9. tétel

5. *Stabil házasságok problémája, ennek megoldása lineáris időben. Maximális méretű független csúcsrendszer keresése gráfban. Kis javítás az exponenciális algoritmusban. Speciális eset intervallum-gráfokra: az intervallumpakolás. MIN-MAX-tétel, gyors algoritmus, lépésszám-becslés.*

Stabil házasságok problémája, ennek megoldása lineáris időben

Ugyanannyi (n) hölgy és úr van. Mindenkit megkérdezünk, hogy kivel házasodna. A válaszok például a következő táblázat szerint lehetnek:

A	B	C	D	E	1	2	3	4	5
2	1	2	1	5	E	D	A	C	D
5	2	3	3	3	A	E	D	D	B
1	3	5	2	2	D	B	B	B	C
3	4	1	4	1	B	A	C	A	E
4	5	4	5	4	C	C	E	E	A

Az urak a betűk, a hölgyek a számok.

Def: Azt mondjuk, hogy egy házassági rendszer stabil, ha nincs benne olyan X férfi és Y nő, hogy ők nem házasok, de X jobban kedveli Y-nt, mint a feleségét, és Y jobban kedveli X-et, mint a férjét.

Tétel: Tetszőleges inputra van stabil házassági rendszer, és ezt könnyű megtalálni.

Algoritmus: Az urak megkérik a listában legfölül szereplő hölgy kezét, akit még nem kérdezett előtte. A hölgy igent mond, ha még nincs vőlegénye, vagy ha a kérő jobban tetszik neki, mint az addigi kérői. (A sárga hátterű számok adják a végeredményt.) (Ez az urak szempontjából jó stabil házassági rendszer. A hölgyek szempontjából nézett házassági rendszer más eredményt adhat!)

Állítás: Az algoritmus véget ér, ha ugyanannyi férfi van, mint nő, és ha egy nőnek már volt kérője, akkor mindig lesz. Legfeljebb n^2 leánykérés lehet, ezért az algoritmus lineáris, mert $2n^2$ az inputméret.

Állítás: A házassági rendszer stabil.

Biz: A hölgyek vőlegényeinek minősége szigorú monoton javul. Az urak menyasszonyainak minősége szigorú monoton romlik. Tegyük fel, hogy X és Y instabil házasságot kötöttek. Ez lehetetlen feltevés.

Maximális méretű független csúcsrendszer keresése gráfban

Def: G(V,E) egy gráf. A $V_1 \subseteq V$ **független** csúcshalmaz $\Leftrightarrow$ V_1 semelyik két csúcsa között nincs él.

Def: Egy V_1 független csúcshalmaz akkor **maximális** V-ben, ha nincs olyan V_2 független részhalmaza V-nek, amelynek elemszáma nagyobb V_1 elemszámánál, és $V_1 \subseteq V_2$.

Def: **Maximális méretű** egy független csúcshalmaz, ha nincs nála nagyobb elemszámú független.

Maximális csúcshalmazt könnyű találni, maximális méretűt azonban nehéz. Maximális csúcshalmazt kapok, ha kiválasztok egy csúcsot, és kidobom a szomszédjait. A maradék csúcsokból megint kiválasztok egyet, majd annak is kidobom a szomszédjait, stb.

Algoritmus: Egyszerű exponenciális algoritmus **maximális méretű** független csúcshalmaz keresésére:

Megnézem V hatványhalmazának (P(V)) minden elemét, hogy független-e. Ha az, akkor a méretét összehasonlítom az eddigi maximális méretű függetlenek méretével. Ha a mostani nagyobb, cserélek. $|P(V)| = 2^n$, ezért $2^n n^2$ lépésben találja meg a megoldást.

Kis javítás az exponenciális algoritmusban

Esetszétválasztás: 1. eset, ha G-ben minden csúcs foka legfeljebb 2. 2. eset, ha van benne legalább harmadfokú csúcs.

1. eset: Az ilyen gráfok fajtái: izolált pontok gráfja, körök gráfja, utak gráfja. Az izolált csúcsoknál az összes csúcs együtt adja a maximális méretű független csúcsrendszert. A kör esetén minden második csúcsot kell kiválasztani. Az út esetén is minden második csúcsot kell választani, de az út végéről kell indulni.

2. eset: a) eset: V benne van a maximális méretű függetlenben; b) eset: nincs benne.

n csúcsú gráfra α^n lépésben meg tudom csinálni, ha $1 \le \alpha \le 2$.

$$\alpha^n = \alpha^{n-4} + \alpha^{n-1}$$

A 2a. esetben $\alpha^4 = 1 + \alpha^3$. Azért lehet n-4 és n-1, mert V-k szomszédjai 3-an nem

$$\Rightarrow \alpha = 1{,}81$$

számítanak, a többi szomszéd viszont igen. Az algoritmus így $2^n n^2$ helyett $1{,}81^n n^2$ lépésben lefut.

Bizonyos speciális esetekben a maximális méretű független csúcshalmaz meghatározása könnyű.

Nagy adathalmazra nem jó algoritmus, mert exponenciális.

Speciális eset intervallum-gráfokra: az intervallumpakolás MIN-MAX-tétel, gyors algoritmus, lépésszám-becslés

Ha G intervallumgráf, akkor gyorsan megy a legnagyobb független csúcshalmaz kiszámolása.

Def: G intervallumgráf, ha G előáll a következő módon: egy I valós intervallum bizonyos részintervallumainak felelnek meg G csúcsai. v és v' élt alkot, akkor és csak akkor, ha v és v'-nek megfelelő intervallumok metszik egymást.

Tétel: Az intervallumgráfokban az input méretében polinomiálisan sok lépésben megtalálható a legnagyobb független csúcshalmaz.

Biz: új feladat kitűzésével és megoldásával.

Új feladat: Tegyük fel, hogy egy teherautóval való fuvarozást akarunk megoldani. 1 fuvar 10000 Ft-ba kerül, a munka 8-22 óráig tarthat. Egyszerre csak egy fuvar vállalható, de minden fuvar ugyanannyit ér. Kellene, hogy a legtöbb munkát vállaljuk el.

Input: intervallumrendszer.

Output: maximális méretű független intervallumrendszer (ebben nincsenek metsző intervallumok).

Javasolt eljárás: elvállaljuk azt, ami az elvállalhatók közül a legkorábban véget ér. Ha nincs több elvállalható, akkor vége az eljárásnak.

Miért ez a legjobb?

Lemma: Ha egy pontrendszert akkor nevezünk **lefogónak**, ha az összes intervallumot metszi, akkor igaz lesz, hogy a minimális méretű lefogó pontrendszer elemszáma legalább annyi, mint a maximális méretű független intervallumrendszer elemszáma.

Biz: a lefogó pontrendszer minden pontja benne van az intervallumrendszer egy-egy intervallumában, és minden intervallumban van egy lefogó pont.

Megj: tegyük fel, hogy I lefogó pontrendszer, J egy független intervallumokból álló rendszer és az elemszámuk megegyezik. Ekkor I minimális méretű lefogó pontrendszer és J maximális

méretű független intervallumrendszer. Ugyanis ha nem volna minimális I, lenne kisebb, ami ellentmond a tételnek.
Cél: megmutatjuk, hogy a fenti javasolt független J intervallumrendszerhez van I lefogó pontrendszer, hogy az elemszámuk megegyezzen, és ezzel készen is vagyunk.
I legyen a J elvállalt munkák jobb végpontjai, J nyilván független intervallumrendszer és az elemszámuk megegyezik. Kell, hogy I lefogó pontrendszer legyen, vagyis metsszen minden intervallumot. Tegyük fel, van olyan intervallum, ami két jobb végpont között van. Ez nem lehet, mert ha lett volna ilyen, azt is elvállaltuk volna, tehát a jobb-végpontok lefogó pontrendszert alkotnak, ezért a minimális meg a maximális a tétel szerint adódik.
Ezek szerint a javasolt eljárás jó.
Kérdés: mennyire gyors?
1. rendezni kell bal- és jobb-végpontok szerint (2 rendezés, melyek azok, amiket elvállalhatunk; az elvállalhatók közül melyik az első)
$n \log n + n \log n + n \Rightarrow 0(n \log n)$ -es algoritmus

10. tétel

6. *Véletlent használó algoritmusok. Polinomazonosság-ellenőrzés, a Schwartz-lemma. Számelméleti algoritmusok: az euklideszi algoritmus, lépésszám-becsléssel. a^b mod m kiszámítása polinomiális időben. Prímtesztek: egy, amely pszeudoprímekre nem működik; egy másik (Rabin-Miller), amely mindig működik.*

Véletlent használó algoritmusok
Polinomazonosság-ellenőrzés, a Schwartz-lemma

Feladat: döntsük el, azonos-e két egész-együtthatós polinom. Átfogalmazva: a két polinom különbség-polinoma vajon az azonosan 0 polinom-e. A beszorzásos kiszámolás általában nagyon lassú. Viszont egy polinom akkor azonosan nulla, ha minden behelyettesítéssel 0-t kapunk.

Eljárás: vegyünk véletlen egész számokat, helyettesítsük be őket az x-ek helyébe (n-változós polinom esetén n véletlen szám kell). Számoljuk ki a polinom értékét. Ha egyszer az jön ki, hogy ez nem nulla, akkor a polinom biztosan nem a nulla-polinom. Ha nagyon sokszor az jön ki, hogy a polinom értéke nulla, és egyszer sem jön ki az, hogy nem null, akkor nagy valószínűséggel (de nem biztosan) mondható, hogy ez az azonosan nulla polinom.

Lemma: **Schwartz-lemma**: Legyen **h** nem azonosan nulla polinom, egész együtthatós, n változós, minden változójában legfeljebb d-edfokú. Válasszuk az x1, …, xn számokat véletlenül, egyenletes eloszlásban és függetlenül az {1, 2, … N} halmazból. Ekkor

$\Pr(h(x_1, x_2, \ldots, x_n) = 0) \le \frac{dn}{N}$. Legyen N=2dn, ekkor a valószínűség 0,5-nél kisebb. Ha 100-szor választok véletlen behelyettesítést, és mindig 0-t kapok, akkor $\frac{1}{2^{100}}$ lesz a tévedés valószínűsége.

Biz: feltételes valószínűség: $\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}$. (Tétel: teljes valószínűség tétele: $\Pr(B)\Pr(A|B) + \Pr(\overline{B})\Pr(A|\overline{B}) = \Pr(A)$.)

n=1-re: 1 változós polinom, véges sok gyöke van, legfeljebb d különböző gyöke. Annak a valószínűsége, hogy $\{1,2,\ldots,N\}$-ből választva pont a gyököt találom el: $\le \frac{d}{N}$.

Indukció: tegyük fel, n-ig tudjuk, n+1-re szeretnénk belátni.

$h(x_1, \ldots, x_n, x_{n+1}) = \sum_{i=0}^{t} h_i x^i$, ahol $1 \le t \le d$ és $h_t \neq 0$ n-változós polinom. (Az n+1 változós polinomot x_1-gyel kifejtettük, a h_i-k legfeljebb n-változósak.)

$$\Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0) = \Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0 \mid h_t(\xi_1,\ldots,\xi_{n+1}) = 0) \cdot \Pr(h_t(\xi_1,\ldots,\xi_{n+1}) = 0) +$$
$$\Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0 \mid h_t(\xi_1,\ldots,\xi_{n+1}) \neq 0) \cdot \Pr(h_t(\xi_1,\ldots,\xi_{n+1}) \neq 0) \leq$$
$$\Pr(h_t(\xi_1,\ldots,\xi_{n+1}) = 0) + \Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0 h_t(\xi_1,\ldots,\xi_{n+1}) \neq 0) \leq \frac{(n+1)d}{N}$$
$$\Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0 \mid h_t(\xi_1,\ldots,\xi_{n+1}) = 0) \leq 1$$
$$\Pr(h_t(\xi_1,\ldots,\xi_{n+1}) \neq 0) \leq 1$$
$$\Pr(h_t(\xi_1,\ldots,\xi_{n+1}) = 0) \leq \frac{nd}{N}$$
$$\Pr(h(\xi_1,\ldots,\xi_{n+1}) = 0 h_t(\xi_1,\ldots,\xi_{n+1}) \neq 0) \leq \frac{t}{n} \leq \frac{d}{n}$$

Számelméleti algoritmusok: az euklideszi algoritmus, lépésszám-becsléssel. a^b mod m kiszámítása polinomiális időben

Input: a, b természetes számok, $a \geq b$. Output: lnko(a,b)=(a,b).

$a = c_1 b + r_1$

$b = c_2 r_1 + r_2$

$r_1 = c_3 r_2 + r_3$

$\vdots$

$r_{i+1} = c_{i+3} r_{i+2} + 0$

$(a,b) = r_{i+2}$

Tegyük fel, hogy a, b 200 decimális jegyűek. Hány lépésben tudjuk (a,b)-t kiszámolni? 10^{100} lépés.

Több is mondható. Legfeljebb $2\log a$ maradékot számolunk.

$10^3 \approx 2^{10}$

$\left(10^3\right)^{66} = 2^{10 \cdot 66} = 2^{660}$

Input mérete: log *a* + log *b*, 2 log *a* új maradékot kell számolni. Polinomiális algoritmus.

Input: a, b, m természetes számok. Output: $a^b \equiv m$.

Mi a helyzet mod m nélkül? 10^b-t kellene kiszámolni, ez exponenciális volna az input méretében.

Input mérete: a betűk vagy számjegyek száma.

Nem ismert polinomiális algoritmus a következők kiszámolására: $\binom{a}{b} \bmod m$.

$n! \bmod m$

Gyors polinomiális algoritmus adódik, ha

1. eset: $b = 2^t$, mert ekkor $a, a^2 \bmod m, a^{2^2} \bmod m, a^{2^3} \bmod m, \ldots$-et kell kiszámolni, mindig az előzőt négyzetre kell emelni. $t = \log b$ négyzetre emelés kell, vagyis inputméretnyi négyzetre emelés.

2. eset: b nem 2 hatványa.

Áll: b mindig előáll különböző 2-hatványok összegeként.

Biz: 2-es számrendszer alapján tiriviális.

$b = 2^{\alpha_1} + \ldots + 2^{\alpha_z}$

$a, a^2, a^{2^2}, \ldots, a^{2^{\alpha_z}} \bmod m$, ahol α_z a legnagyobb.

$\log b$ szorzás négyzetre emelés, $\log b$ redukálás.

$a^{2^{\alpha_1}} \cdot \ldots \cdot a^{2^{\alpha_z}} \equiv a^{2^{\alpha_1}+\ldots+2^{\alpha_z}} \equiv a^b \bmod m$.

Prímtesztek: egy, amely pszeudoprímekre nem működik; egy másik (Rabin-Miller), amely mindig működik

Input: egy természetes szám (m), output: 1 ha m prím, 0, ha nem prím.

Ha minden természetes szám, ami kisebb vagy egyenlő $\sqrt{m}$ nem osztója m-nek, akkor m prím.

Tegyük fel, hogy 200 decimális jegyű prímek kellenek. Hány ilyen p prím van?

Nagy prímszámtétel: Ha $\Pi(x)$ jelöli az $\{1,2,\ldots,x\}$ halmazban lévő prímek számát, akkor $\Pi(x) \approx \frac{x}{\ln x}$.

Honnan szerezzünk prímeket?

Választunk egy véletlen számot, majd teszteljük, hogy prím-e. Véletlen algoritmusok évtizedek óra ismertek. 2002-ben Aggraval, Kayol, Saxena olyan algoritmust dolgoztak ki, ami nem használ véletlent, de polinomiális idejű.

Két változat:

Ami nem mindig működik:

Def: Azt mondjuk, hogy m pszeudoprím, ha m nem prím, de minden nála kisebb, hozzá relatív prím egészre teljesül, hogy $a^{m-1} \equiv 1 \bmod m$.

Pszeudoprímek pedig vannak, pl. a 4 és a 6 ilyenek.

1 módszer véletlen prímtesztre, amely pszeudoprímekre nem jó:

Tétel: (kis-Fermat-tétel) p prím → minden a p-hez relatív prímre igaz, hogy $a^{p-1} \equiv 1 \bmod p$.

G legyen a mod m redukált maradékosztályok halmaza $= \{a : (a,m) = 1 \wedge 1 \le a \le m-1\}$. Ekkor G csoport az uv mod m műveletre.

$H = \{u \in G : u^{m-1} \equiv 1 \bmod m\}$. Ekkor $H \subseteq G$ és H csoport.

$1^{m-1} \equiv 1 \bmod m$

$a, b \in H \Rightarrow a \cdot b \in H$

$a^{m-1} \cdot b^{m-1} \equiv (ab)^{m-1} \equiv 1 \bmod m$

$a^{m-1} \equiv 1 \bmod m \Rightarrow \left(a^{-1}\right)^{m-1} = \left(a^{m-1}\right)^{-1} \equiv 1^{-1} \equiv 1 \bmod m$

Így $H \le G \Rightarrow |H| \| |G|$.

Ha m prím: G=H (kis-Fermat tétel miatt)

Ha m pszeudoprím: G=H (a pszeudoprímség definíciója miatt)

Ha m egyik sem, akkor $H \subset G \Rightarrow |H| \le \frac{|G|}{2}$.

100-szor ismétlem:

Veszünk egy véletlen a-t, hogy $1 < a < m$. Kiszámoljuk (a,m)-et. Ha ez nem 1, akkor m nem prím, és készen vagyunk. Ha relatív prímek, akkor kiszámoljuk $a^{m-1} \bmod m$-et. Ha ez nem kongruens 1-gyel, akkor legalább 0,5 a valószínűsége, hogy m nem prím és nem pszeudoprím. Ha ez kongruens 1-gyel, akkor m prím vagy pszeudoprím.

Ha egy legalább 0,5 valószínűségű esemény előfordul legalább 100 független kísérletben, annak a valószínűsége legfeljebb $\frac{1}{2^{100}}$.

A Rabin-Miller-teszt

Lemma: Legyen m páratlan szám, $m-1=2^M\cdot N$, ahol N páratlan. m prím $\Leftrightarrow$ $a^N-1, a^N+1, a^{2N}+1, a^{2^2N}+1, \ldots, a^{2^{M-1}N}+1$ számok közül legalább egy osztható m-mel, ahol $\forall\, a\in\{1,2,\ldots,m-1\}$.

Biz: Ha m prím, akkor a fenti számok szorzata $a^{m-1}-1\equiv 0 \bmod m$.

Ha m nem volna prím, akkor lenne egy d, ami osztója m-nek, és nem 1, és a=d-vel.

Lemma: Ha m nem prím, akkor az $\{1,2,\ldots,m-1\}$ „a” elemeinek legalább a felére m nem osztója $a^N-1, a^N+1, a^{2N}+1, a^{2^2N}+1, \ldots, a^{2^{M-1}N}+1$ számok egyikének sem. (nem bizonyítjuk)

R-M teszt:

Veszünk egy véletlen a-t $\{1,2,\ldots,m-1\}$-ből. Ha (a,m)=1 nem igaz, akkor m nem prím, tehát készen vagyunk.

Ha (a,m)=1, akkor kiszámoljuk az $a^N-1, a^N+1, a^{2N}+1, a^{2^2N}+1, \ldots, a^{2^{M-1}N}+1$ számokat, és ellenőrizzük, hogy valamelyiket osztja-e m.

Ha egyiket sem osztja, akkor m nem prím, de ha valamelyiket osztja, akkor kezdjük elölről az algoritmust.

100-szor ismételve már nagy valószínűséggel azt állíthatjuk, hogy prím, ha nem derült ki róla, hogy nem prím.

100 ismétlés után $\frac{1}{2^{100}}$ a valószínűsége annak, hogy nem prím.

A tesztben kevesebb, mint 100-szor van log m szorzás és log m redukálás.

11. tétel

7. *Kriptográfia: titkos kulcsú titkosírások, a one-time pad. Titkos kulcs-csere protokoll (Diffie & Hellman). Nyilvános kulcsú titkosírások. A Rivest-Shamir-Adleman titkosírás. Digitális aláírás. Titokmegosztási eljárások: Shamir módszere és egy optikai megoldás.*

Kriptográfia: titkos kulcsú titkosírások, a one-time pad

Legyen r egy n hosszú véletlen bitsorozat. Legyen az r a kulcs, amit Alice és Bob ismernek, de Eve nem. Legyen x az üzenet.
Ekkor az x üzenetet kódoljuk az r kulcs szerint, majd dekódoljuk szintén az r kulcs szerint. A kódolás módszere a kizáró vagy. $x \oplus r \oplus r = x$. Mivel az i. bitet 50% valószínűséggel kódolja 1-gyé vagy 0-vá, Eve nem tudja kitalálni, hogy mi lehetett az eredeti üzenet.

Titkos kulcs-csere protokoll (Diffie & Hellman)

Cél egy közös r generálása. Alice vesz egy véletlen p prímet és egy q számot úgy, hogy (p,q)=1 legyen. Ehhez vesz egy olyan számot, amire igaz, hogy $1 \le a \le p-1$, a természetes, és $q^a \bmod p$ -t kiszámolja.
$\left(q, q^a \bmod p\right)$-t megkapja Bob, aki generál egy b-t, hogy $1 \le b \le p-1$ legyen, és visszaküldi Alice-nak $q^b \bmod p$ -t. (Helyesebb lenne talán, hogy q helyett p-t adja oda Alice Bobnak, mert különben Bob nem tudja visszaküldeni a megfelelő modulust.) Alice ezt a számot az a-adik hatványra emeli, így $r = q^{ab} \bmod p$ lesz, és ezt használják közösen, hiszen ezt mindketten elő tudják állítani.
Ha Eve figyel, akkor látja, hogy milyen üzenetek mennek át, de nem fogja tudni, hogy az átmenő folyamatokban az a és a b mik.
Diszkrét logaritmus probléma: adott $q, q^a \bmod p, p$, és ebből kell a-t meghatározni. Polinomiális időben nem megoldható a probléma.

Nyilvános kulcsú titkosírások

Pl. RSA, ElGamal
Talán a legelterjedtebb kódolási eljárás.
Alapja: az egyirányú függvények:
Def: $f : A \to R$ egyirányú, ha létezik inverze és x-ből f(x)-et könnyű (polinomiális) algoritmussal ki lehet számolni, viszont f(x)-ből x-et nem lehet polinomiális algoritmussal kiszámolni, a legtöbb esetben, vagy a ***legrosszabb esetben***, vagy átlagosan.
Megj: nem ismerünk egyirányú függvényeket.
Megj: széleskörű nézet, hogy $f(x,y) = x \cdot y$, vagy $f(p,q,a) = q^a \bmod p$ egyirányúak.
Hogyan használjuk az egyirányú függvényeket?
M(x,e)=elkódolt üzenet, ahol M a kódolási eljárás, x az üzenet, e a nyilvános kulcs.
Fontos, hogy M(M(x,e),f)=x, ahol f a titkos kulcs.
Feltétel: ha ismerem M-et, e-t és M(x,e)-t is, ebből ne tudjam polinomiális időben kiszámolni az x-et.
Mitől lehet ezt nyilvános kulcsú titkosításban használni?
e: egy karaktersorozat, ami a kulcs, ezt megjegyzem a hozzá tartozó névhez. Amikor az illetőnek küldök üzenetet, akkor ezzel a nyilvános kulccsal kódolom az elküldendő üzenetet. A címzett a saját titkos kulcsával dekódolja ezt.

A Rivest-Shamir-Adleman titkosírás

Hogyan generál nyilvános és titkos kulcsot? Alice készít *p* és *q* nagyméretű véletlen prímeket, és összeszorozza őket (*m=pq*). Veszünk egy *e*-t, hogy *(p-1)(q-1)* és *e* relatív prímek legyenek. Addig kísérletezek az *e*-vel, amíg nem lesz jó. Ekkor létezik olyan *f*, hogy $ef \equiv 1 \bmod (p-1)(q-1)$.

A nyilvános kulcs lesz az (m,e) pár, a titkos kulcs pedig az f.

A kódolás pedig: $M(x,e) = x^e \bmod m$. $M(y,f) \equiv y^f \bmod m$

Miért jó? $M(M(x,e),f) = x^{ef} = M(M(x,f),e)$. $ef \equiv 1 \bmod \varphi(m)$

$x^{ef} = x^{k\varphi(m)+1} = x\left(x^{\varphi(m)}\right)^k \equiv x$

Biztonságosnak látszik, mert ismerjük e-t, m-et, (p-1)(q-1)-et, akkor ismerem p-t, és q-t. Nehéz ezt a számolást elvégezni.

Vannak esetek, amikor az RSA-t könnyű feltörni. Ha e-nek vagy f-nek az abszolút értéke kicsi, akkor könnyű megtenni ezt. Nem triviális, hogy miért könnyű feltörni, de könnyű ☺. Van egy szabvány, hogy mekkora értékeket használjunk a feltörhetőség minimálisra szorításához. Nem garantál semmit, de az egyszerű támadások ellen megvéd.

Amennyiben majd tudunk faktorizálni, akkor gyorsan feltörhető a kód. De nem tudunk faktorizálni.

Digitális aláírás

Digitális aláírásra is használható a nyilvános kulcsú titkosítás, ha a titkos kulcs nem kerül nyilvánosságra. Módszer: a titkos kulccsal kódolok, ami a nyilvános kulccsal dekódolható.

Digitális aláírás módszere: Adott x levél. M(x,e) -vel küldve titkos levelet kap a titkos kulcs tulajdonosa. Ha M(x,f)-et küld a titkos kulcs tulaja, akkor digitálisan aláírta. Ha M(M(x,f),e), akkor a címzett tudja megfejteni úgy, hogy még a digitális aláírást is megkapja.

Feltétel: M(M(x,e),f)=x és M(M(x,f),e)=x.

Titokmegosztási eljárások: Shamir módszere és egy optikai megoldás

Például bank széfjének nyitása.

n db ember, mindegyik kap valamilyen titkot.

Cél az, hogy tetszőleges k ember együtt már hozzájusson a titokhoz az n-ből, de k-1 ember már legyen kevés, sőt, ők már ne tudjanak meg semmit!

Shamir titokfelosztása: legyen $a_0 \in N$ egy természetes szám, és legyen ez a titok. És legyen $a_1, a_2, ..., a_{k-1}$ véletlen számok 1-től N-ig. Ekkor $P(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1}$.

Ha ezt egy test fölött csinálom, akkor $\begin{matrix} 1. & 2. & \dots & n. \\ P(1) & P(2) & \dots & P(n) \end{matrix}$.

K ember ismeri az egész P-t (polinominterpoláció) speciálisan a_0 -t is

Ha csak k-1 jön össze, akkor semmit nem tudnak.

$P(i_1), P(i_2), \cdots, P(i_{k-1})$

$\{i_1, i_2, \cdots, i_{k-1}\} \subset \{1,2,\cdots,n\}$.

$a_0 = P(0) := U \in N$

Milyen megoldás van számítógép vagy interpoláció nélkül?

Optikai titokmegosztás

Egy képet szét lehet vágni több rétegre úgy, hogy azok részleteiben nem felismerhetők, de egymásra helyezve kialakul az ábra.
Gond: ha nem ugyanaz a nyomtató nyomtatja ki, nem lesz pontos az illesztés.
Módszer: felosztjuk pixelekre, sőt, páros pixelekre. A szétbontás során minden egyes ilyen párnál véletlenszerűen vagy az egyik felét színezem be, vagy a másikat.
Ha átlátszóra szeretném csinálni, akkor a másik ábrán ugyanott satírozom be, ha sötétet szeretnék, akkor a pixelpár másik pixelét színezem be. (Így a világos terület fele lesz sötét, a másik fele világos (50%), a sötétnél pedig 100% lesz a lefedés.)
Ami nem rögtön világos: miért nem lehet a második fóliából kideríteni, hogy mi a titok, mert Alice-nál véletlenszerűen lett kiszínezve, Bobnál viszont Alice fóliájától és a mintától függ a kitöltés. Viszont nem tudjuk, hogy melyik pixel hogyan függ egy másiktól.

12. tétel

8. Kis átmérőjű és kisfokú hálózatok: a hiperkocka, a CCC, a pillangó, a shuffle-exchange és a de Bruijn gráfok. Ezek konstrukciója, fokszáma, bejárása.

Kis átmérőjű és kisfokú hálózatok

n pont (csúcs), amelyeknek kicsi az átmérőjük és kicsi a fokszámuk
n csúcs, fokszám: n-1, átmérő: 1
Def: átmérő egy gráfban a maximális távolság két pont között. Másképp: minimális távolságok maximuma.

A hiperkocka

1 dimenziós kocka: szakasz, kétdimenziós: négyzet, háromdimenziós: kocka
Négydimenziós kockát nehéz rajzolni.
Pl. kétdimenziós kockánál a csúcsokat jelölje két bit, akkor szomszédos két csúcs, ha a bitjeik közül csak az egyik különböző.
Háromdimenziósnál három bit van, úgy jelöljük, hogy a felső lap és az alsó lap számozása az első jegytől eltekintve ugyanaz, és az első lapnak van 0 az első bit helyén, a hátsónál meg 1.
K-dimenziós hiperkocka: 2^k csúcsú gráf, a csúcsok a $\{0,1\}^k$ elemeivel vannak címkézve, két csúcs össze van kötve éllel, pontosan akkor, ha egy koordinátában különböznek.
(Pl. 8-dimenziós hiperkockánál két szomszédos csúcs: 00010110 és 10010110, valamint 00010110 és 00010010.)
A fokszám ilyen K-dimenziós hiperkockában éppen K.
Átmérő: ha minden koordinátában különböznek, akkor a távolságuk K lesz, mert éppen K lépésben lehet minden koordinátát felcserélni. Ezért az átmérő K=logn.
Egyre több csúcsnál a fokszám nőni fog, és ez nem túl jó. Az lenne jó, ha a fokszám nem változna, rögzített érték maradna.
Hiperkockánál: n csúcs, logn a fokszám, logn az átmérő.
Lehet-e olyat, hogy az átmérő ne nagyon nőjön, de a fokszám konstans legyen.
(4dimenziós kocka készítése: veszek két háromdimenziósat, és az egyikben 0-t írok a csúcsokhoz, a másikban 1-et.)

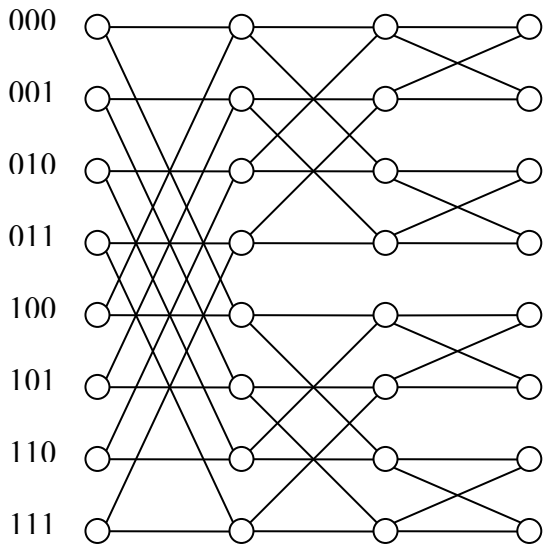
A CCC

Fokszám csökkentése hiperkockán.
Cube-connected cycles
Módszer: a kocka csúcsát levágjuk, és lesz helyette egy kör (háromD-ben háromszög) (gráfban kör). Háromdimenzióban nem sokat segít, mert a fokszám ugyanúgy 3 lesz. K-dimenziós kockán a csúcsot lecseréljük K hosszú körre. Az eddig az adott csúcsba bemenő K élet a K hosszú kör egy-egy pontjához csatlakoztatom → lesz K db 3 fokú csúcs. (Sok csúcs lesz, de a fokszám kicsi.)
Ettől az átmérő 2K-ra nőtt.
A CCC-n n*k csúcs, 3 a fokszám, az átmérő: 2logn=2K.
A CCC durva és nem túl szép megoldás, vannak szebbek is.
Megj.: ha konstans fokszámon van, akkor az átmérő nem lehet logn-nél kisebb. (sok processzor összekötésekor érdekes, ahol fontos, hogy hány másik processzorhoz kapcsolódik egy processzor).

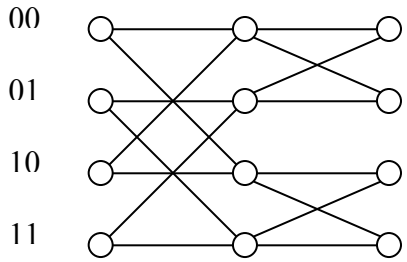
Valóban, ha van egy csúcs, és el szeretném érni az összes többi csúcsot. Ha d a fokszám, akkor egy távolságban d darabot érek el, 2 távolságban d*d, n távolságban d^n darab csúcsot érek el. Kell: $d^{átmérő} \geq n, átmérő \geq c \cdot \log n$.

A pillangó

(egy szebb és fontosabb hálózat)
A hiperkocka minden csúcsának megfeleltetek egy egyenest



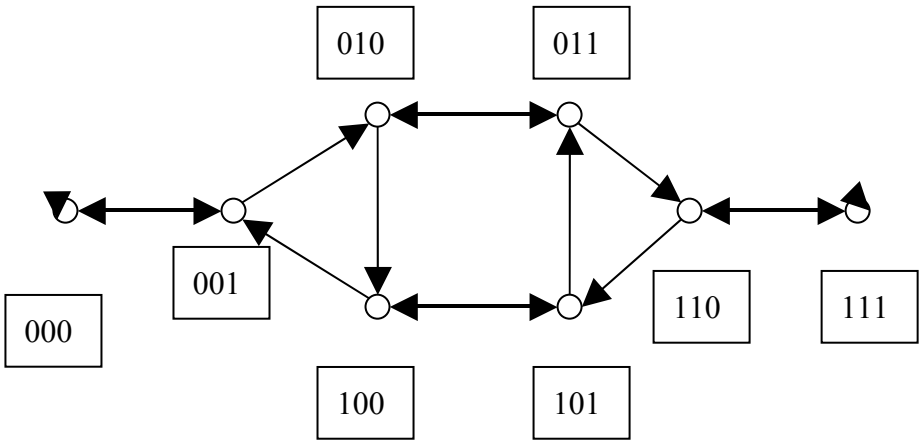
Szét vannak húzva a kockacsúcsok.
Minden kocka-csúcshoz egy vonalat rendelek, amin k+1 csúcs van.
Ez rekurzív, mert a kétdimenziós pillangó így néz ki (ez a háromdimenziós felének egy darabja):



A 3dimenziósban a fokszám legfeljebb négy.
Pillangó: n*(logn+1)=n*(k+1) csúcs, az átmérője: k+1, a fokszáma: 4.
Routing-szabály: u1, u2, … , uk-ból v1, v2, … , vk-ba szeretnék menni valahogy. Ha ui=vi, akkor vízszintesen megyek, ha ui<>vi, akkor ferdén.
Nagyon sokszor jól használható hálózat.
Vonalak közti átmérő: k. Mindenhova eljuthatok max k db ferde vonalon.

A shuffle-exchange

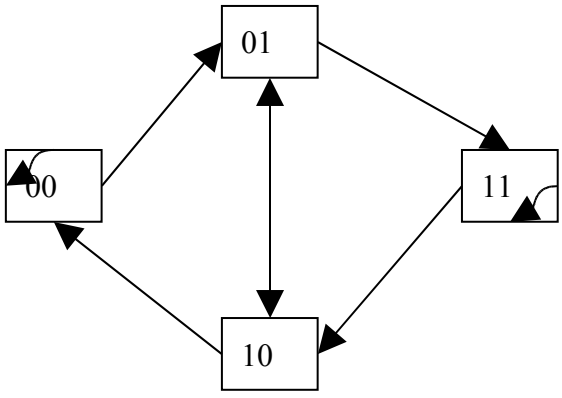
A csúcsok k hosszú 01 sorozatokkal vannak címkézve, u1, u2, …, uk össze van kötve a v1, v2, …, vk-val, ha az utolsó bitben különböznek. (Mindig két valamivel van összekötve egy csúcs.) Ekkor: u1, u2, …, uk össze van kötve u1, u2, …, u(k-1) oda-vissza nyíllal (exchange él), 1-uk –kal, és u2, …, u(k-1), uk, u1-ekkel egyirányú nyíllal (shuffle él).
Példa:



Fokszám 3, csúcsszám: n.
Az exchange élet lehet kétszeres élnek számolni (és most számoljuk így).
Módszer: u1, u2, …, uk-ból indulok. Ha uk=vk, akkor shuffle élen megyek, ha vk=1-uk, akkor exchange élen megyek. Innentől újra megyünk tovább.
Routing a shuffle-exchange gráfban:
Cél: „u1, u2, …, uk”-ból eljutni a „v1, v2, …, vk”-ba.
Ha uk=vk, akkor shuffle-ön megyek. „u2, u3, …, *vk*, u1”
Ha uk<>vk, akkor exchange-en megyek. „u1, u2, …, u(k-1), *vk*”
Az exchange-en menve már lesz egy közös az u és a v sorozatoknak. Ezután muszáj lesz egy shuffle-ön továbbmenni, ezért az „u2, u3, …, u(k-1), *vk*, u1” lesz a sorozat (így két lépésben jutottunk oda, ahova az egyenlőség esetén egy lépésben).
Ezért az átmérő: 2K.
Azért hívják kártya-keverő gráfnak, mert hasonlít ahhoz, amikor kártyát keverünk…

De Bruijn gráf

2^k csúcs, 2^{k+1} él, és az „u1, u2, …, uk” össze van kötve az „u2, u3, …, uk, 1” és az „u2, u3, …, uk, 0” sorozatokkal.
Pl.



$n=2^k$ csúcs, a fokszám 4. Az átmérő: u sorozatból v sorozatba megyünk, először oda kell menni, ami az utolsó helyre a v1-et rakja, aztán oda, ami a v2-t adja, stb. Így k lépésben eljutok u-ból v-be, ezért az átmérő k.
Azért jobb a shuffle-exchange gráffal szemben, mert itt k az átmérő.

Hálózati torlódás

Az a legjobb eljárás, hogy ha el akarok küldeni egy-egy csomagot minden csúcsból másik csúcsba valamelyik algoritmus használatával, akkor kicsi a valószínűsége a torlódásnak, ha mindig a legrövidebb úton próbálok küldeni, akkor valószínű a torlódás.