Algoritmusok bonyolultsága

3. előadás

http://www.ms.sapientia.ro/~kasa/komplex.htm

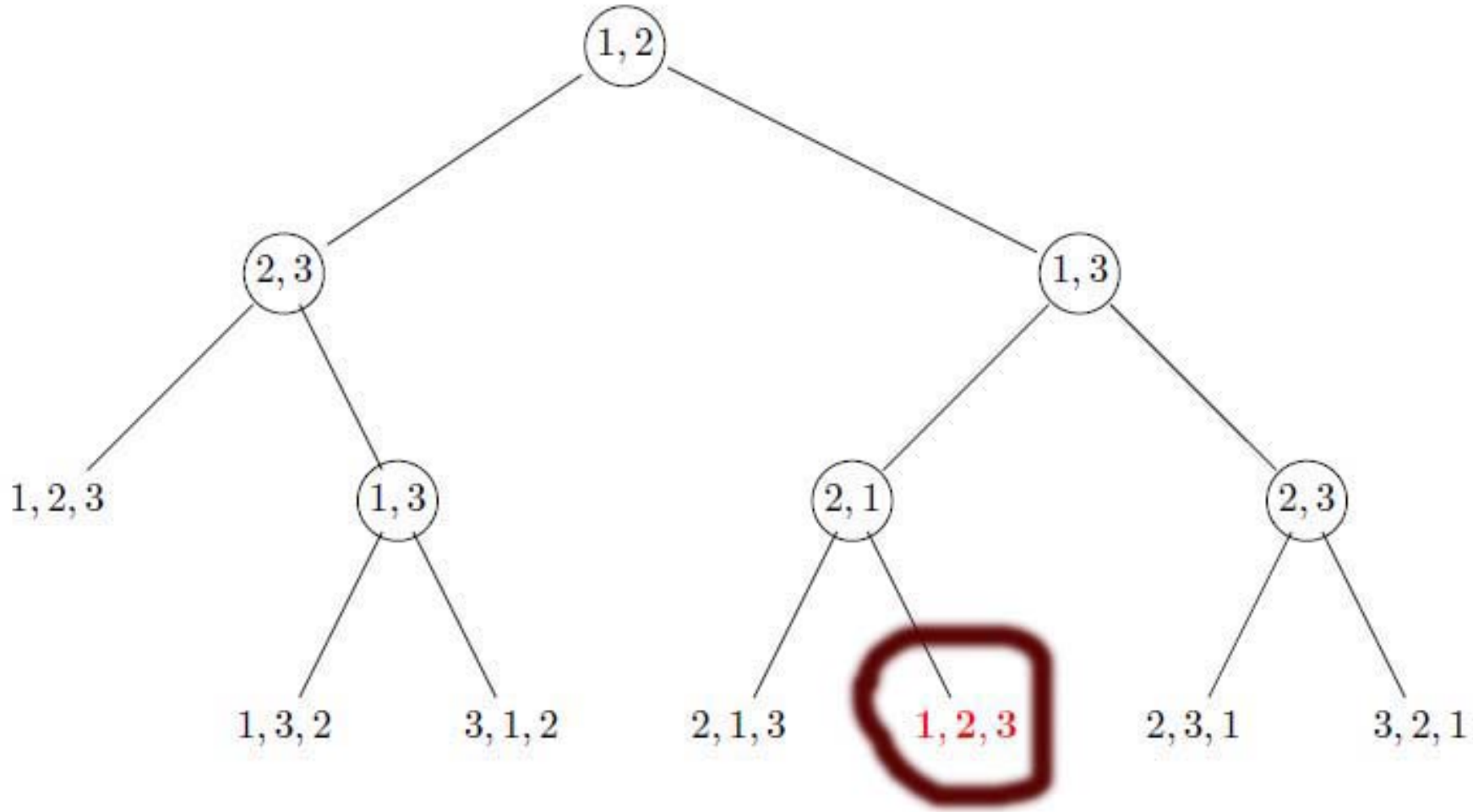
Alsó korlát rendezési algoritmusoknál

	$A(n)$	$W(n)$
Buborékrendezés	$\Theta(n^2)$	$\Theta(n^2)$
Gyorsrendezés	$n \log n$	$\Theta(n^2)$

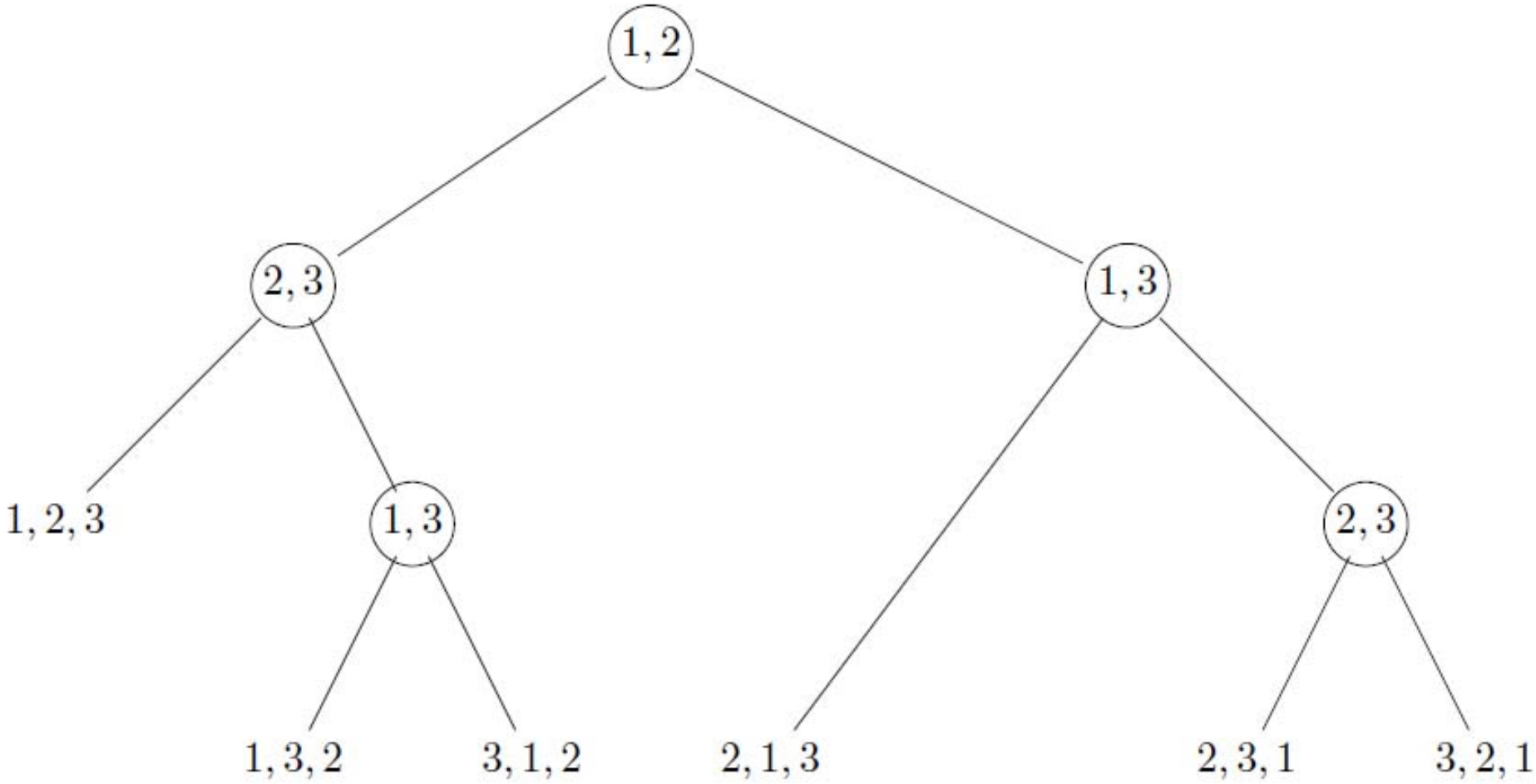
Mennyi az alsó korlát?

Tetszőleges rendezési algoritmus esetében készítünk egy döntési fát. Minden $a : b$ összehasonlításnak megfelel egy csúcs, ha $a \leq b$, akkor a bal, ha $a > b$, akkor a jobb oldali részfában folytatjuk.

A csúcsban 1,2 azt jelenti, hogy x_1-et hasonlítjuk x_2-vel, azaz 1, 2 jelentése: $x_1 : x_2$.



A buborékrendezés döntési fája, ha $n = 3$. A pirossal jelzett csúcs gyakorlatilag nem létezhet. A levelek száma legalább $n!$



A buborékrendezés döntési fája $n = 3$ esetében.

Ha egy bináris fában ℓ a levelek száma, d pedig a fa mélysége, akkor

- $\ell \leq 2^d$
- $d \geq \lceil log \ell \rceil$
- Egy rendezési algoritmus döntési fája mélysége legalább $\lceil log n! \rceil$.

Tétel

Tetszőleges rendezési algoritmus esetében az összehasonlítások száma legrosszabb esetben legalább $\lceil log n! \rceil$ (ahol n a rendezendő elemek száma).

$$n! \geq n(n-1)(n-2)\cdots\left(\left\lceil \frac{n}{2} \right\rceil\right) \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\log n! \geq \frac{n}{2} \log \frac{n}{2}, \qquad \Theta(n \log n)$$

Pontosabb közelítés:

$$\log n! = \sum_{j=1}^{n} \log j$$

$$\log n! = \sum_{j=1}^{n} \log j \geq \int_1^n \log x dx = \log e \int_1^n \ln x dx =$$

$$= \log e \left[x \ln x - x\right]_1^n = \log e(n \ln n - n + 1) = n \log n - n \log e + \log e \geq$$

$$\geq n \log n - n \log e$$

Mivel $\log e \approx 1.44$:

Tétel

n elem rendezésére legrosszabb esetben megközelítőleg $\lceil n \log n - 1.5n \rceil$ *összehasonlítás szükséges.*

Átlageset

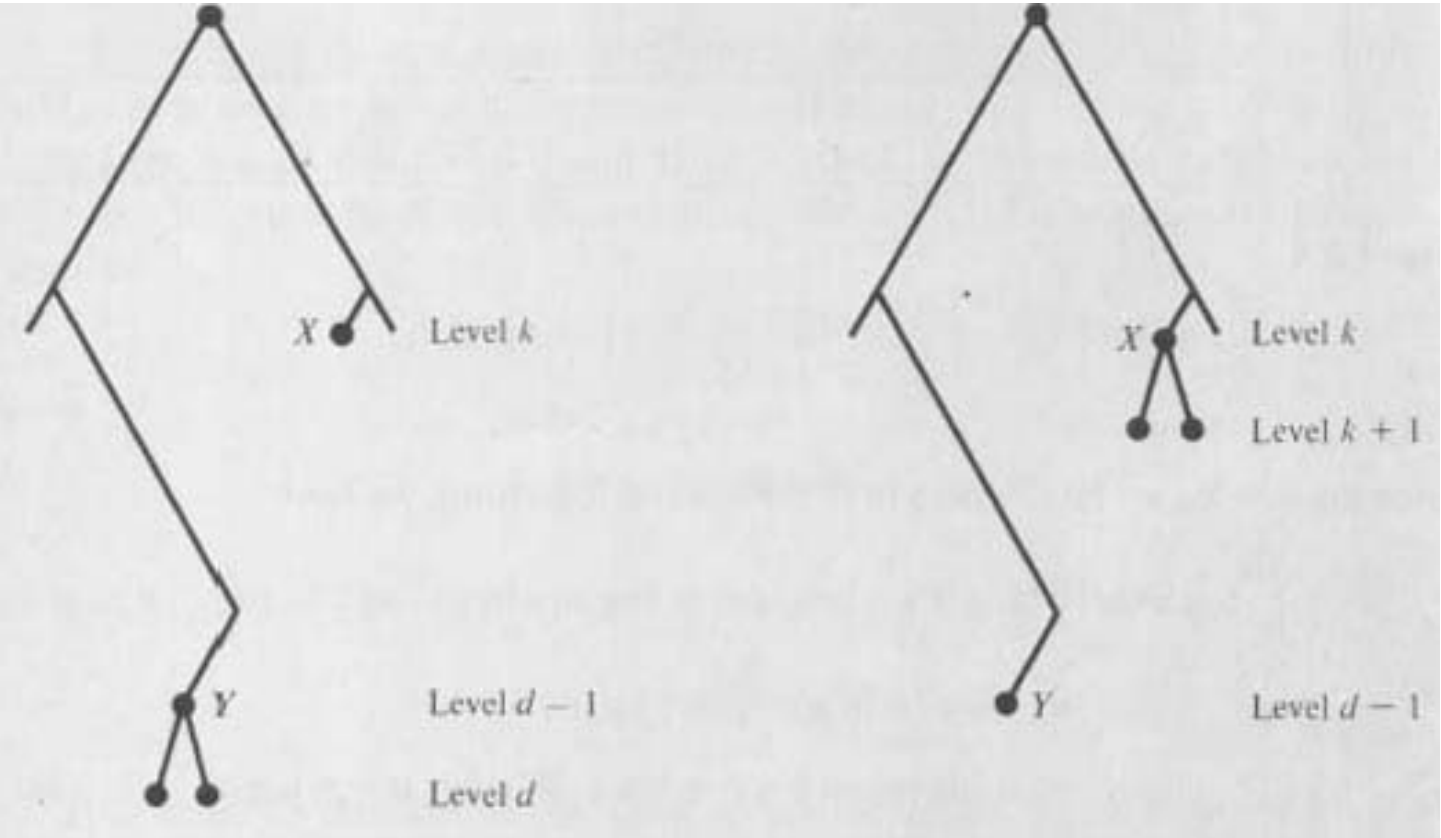
összúthossz (*external path length*): a gyökértől a levelekig futó utak hosszának az összege

b-fa: olyan bináris fa, amelyben minden csúcsnak 0 vagy 2 leszármazottja van

Lemma

Az ℓ levelű b-fákban az összúthossz akkor minimális, ha a levelek szomszédos szinteken vannak.

Legyen d a fa mélysége. Legyen X egy k szinten levő levél, és Y egy $d-1$ szinten levő csúcs, amelynek 2 levél leszármazottja van, és $k \leq d-2$. Az Y két leszármazottját vigyük át X leszármazottjává. Számítsuk ki az új összúthosszat.



S. Baase: Computer Algorithms

Ezáltal:

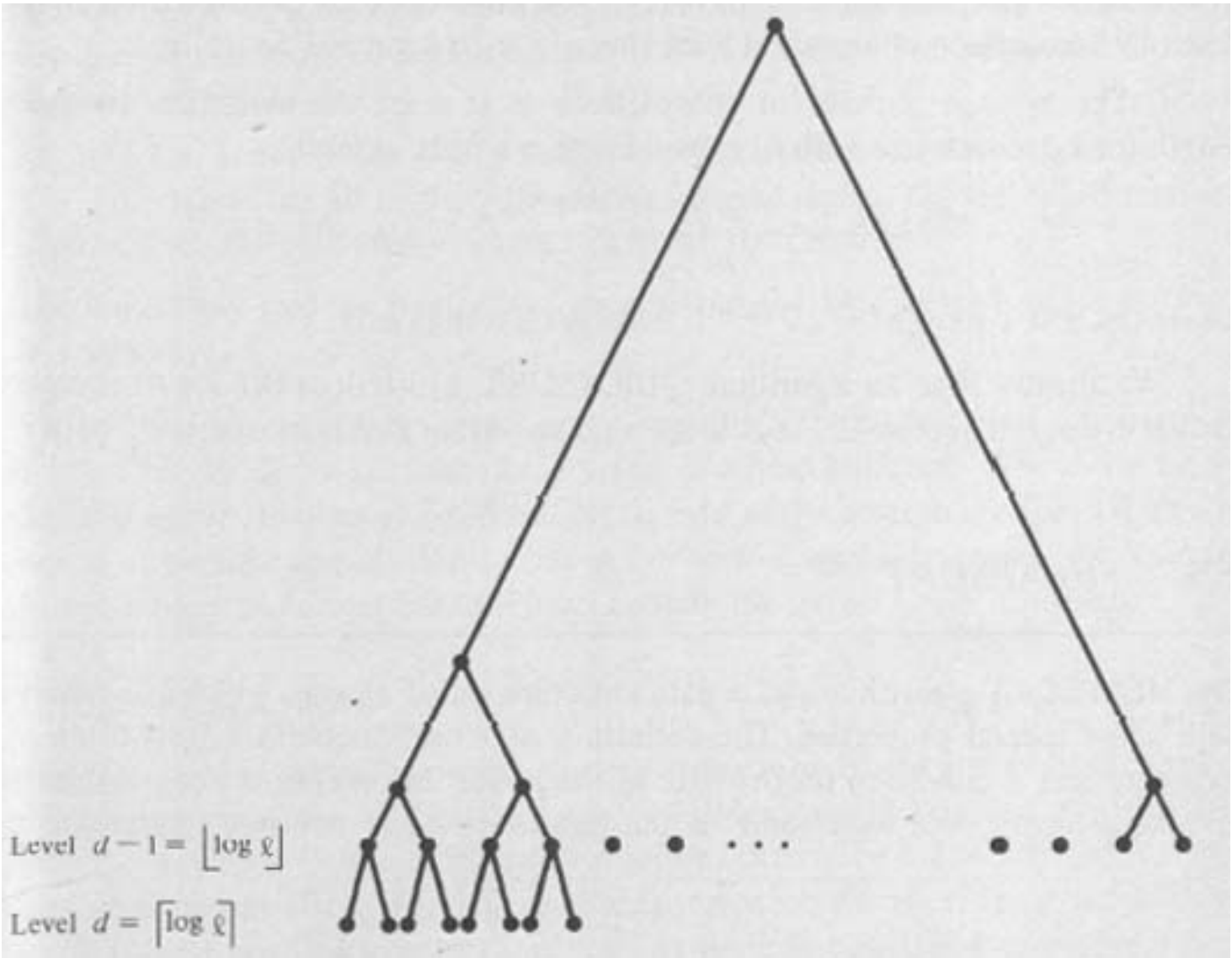
az összúthossz csökkenése: $2d + k$

az összúthossz növekedése: $2(k + 1) + d - 1 = 2k + d + 1$

az összúthossz változása: $2d + k - (2k + d + 1) = d - k - 1 > 0$
mivel $k \leq d - 2$.

Lemma

ℓ levelű b-fában a minimális összúthossz: $\ell \lfloor \log \ell \rfloor + 2\left(\ell - 2^{\lfloor \log \ell \rfloor}\right)$



S. Baase: Computer Algorithms

- ℓ 2-nek hatványa: minden levél $\log \ell$ szinten van.
  Ekkor az összúthossz $\ell \lfloor \log \ell \rfloor$.
- ℓ 2-nek nem hatványa: a fa mélysége $d = \lceil \log \ell \rceil$, és a levelek a $d-1$ és d szinten vannak.
  $d-1$ szintig az összúthossz $\ell(d-1)$, a d szinten a levelek száma: $2(\ell - 2^{d-1})$
  Az összúthossz tehát:
  $\ell(d-1) + 2(\ell - 2^{d-1}) = \ell \lfloor \log \ell \rfloor + 2\left(\ell - 2^{\lfloor \log \ell \rfloor}\right)$

Tétel

n elem rendezésére átlagesetben legalább $\lfloor \log n! \rfloor \approx \lfloor n \log n - 1.5n \rfloor$ *összehasonlítás szükséges.*

$$\frac{n!\lfloor \log n! \rfloor + 2\left(n! - 2^{\lfloor \log n! \rfloor}\right)}{n!} = \lfloor \log n! \rfloor + \varepsilon$$

ahol $0 \leq \varepsilon < 1$, mivel $n! - 2^{\lfloor \log n! \rfloor} < \frac{n!}{2}$.

Kupacrendezés

KUPACOL(A, i)

1. $b := \text{BAL}(i)$
2. $j := \text{JOBB}(i)$
3. if ($b \leq$ *kupacméret*[A]) **és** ($A_b > A_i$)
4. then $max := b$
5. else $max := i$
6. if ($j \leq$ *kupacméret*[A]) **és** ($A_j > A_{max}$)
7. then $max := j$
8. if $max \neq i$
9. then $A_i \Leftrightarrow A_{max}$
10. KUPACOL(A, max)

KUPACOT-ÉPÍT(A)

1. $kupacméret[A] := hossz[A]$
2. for $i := \dfrac{hossz[A]}{2}$ downto 1
3. do KUPACOL(A, i)

KUPACRENDEZÉS(A)

0. KUPACOT-ÉPÍT(A)
1. $kupacméret[A] := hossz[A]$ ▷ a kupacméret változik
2. $n := hossz[A]$
3. for $i := n$ downto 2
4. do $A_1 \Leftrightarrow A_i$
5. $kupacméret[A] := kupacméret[A] - 1$
6. KUPACOL(A,1)
7. return A

Kupacrendezés bonyolultsága

kupac mélysége $\leq \log n$

KUPACOL bonyolultsága $n \log n$

KUPACOT-ÉPÍT bonyolultsága $n \log n$

kupacrendezés legrosszabb esetben $n \log n$

	$A(n)$	$W(n)$	tárigény
Buborékrendezés	$\Theta(n^2)$	$\Theta(n^2)$	helyben
Gyorsrendezés	$\Theta(n \log n)$	$\Theta(n^2)$	$\log n$-nel arányos
Kupacrendezés	$\Theta(n \log n)$	$2n\lfloor \log n \rfloor$	helyben

Összefésülés

Adottak:

$$A_1 \leq A_2 \leq \cdots \leq A_n$$

$$B_1 \leq B_2 \leq \cdots \leq B_m$$

Feladat:
Rendezzük növekvő sorrendbe az

$$A_1, A_2, \cdots, A_n, B_1, B_2, \cdots, B_m$$

sorozatot!

Összefésülés (A, B)

```
 1. n := hossz[A], m := hossz[B]
 2. i := 1, j := 1, k := 0
 3. while (i ≤ n) és (j ≤ m)
 4.         do k := k + 1
 5.            if A_i < B_j
 6.               then C_k := A_i
 7.                    i := i + 1
 8.               else C_k := B_j
 9.                    j := j + 1
10. while (i ≤ n)
11.         do k := k + 1
12.            C_k := A_i
13.            i := i + 1
14. while (j ≤ m)
15.         do k := k + 1
16.            C_k := B_j
17.            j := j + 1
18. return C
```

Összefésülés – elemzés

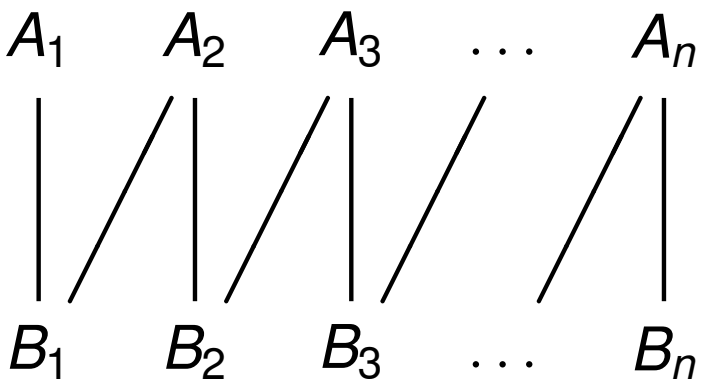
Legrosszabb esetben $n + m - 1$ összehasonlítás kell.
Pl.

$A : 1, 3, 5, 7, 9$
$B : 2, 4, 10$

Ha $n = m$, akkor az Összefésülés optimális.

Tétel

Két n elemből álló sorozat összefésülésére legalább $2n - 1$ összehasonlítás szükséges.



$A_1 < B_1 < A_2 < B_2 < \cdots < A_n < B_n$

Meg lehet választani a bemenetet, hogy összehasonlítások a köv. legyenek:

$A_i : B_i, \quad i = 1, 2, \ldots, n$

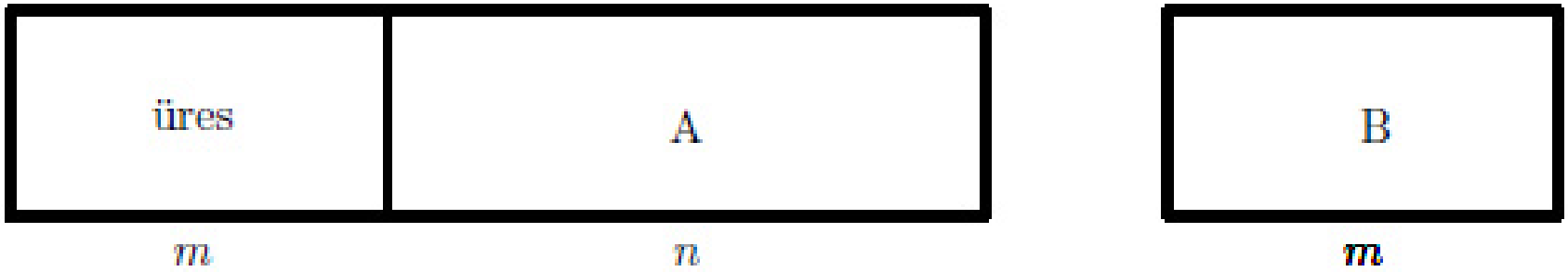
$B_i : A_{i+1}, i = 1, 2, \ldots, n-1$

Ha egy algoritmus nem hasonlítja össze pl. az A_i és B_i elemeket (egy adott i-re), akkor nem tudja helyesen összefésülni. Hasonlóképpen, ha nem hasonlítja össze B_i és A_{i+1} elemeket (egy adott i-re).

Tehát $2n-1$ összehasonlítás szükséges, kevesebbel nem lehet megoldani a feladatot minden bemenetre.

Tárhasználat optimalizálása

összefésülés előtt



összefésülés után

$2(m+n)$ helyett $2m+n$

Összefésűléses rendezés

MERGESORT(A, b, j)

1. if $b < j$ ▷ b bal index, j jobb index
2. then $k := \left\lfloor \frac{b+j}{2} \right\rfloor$
3. MERGESORT(A, b, k)
4. MERGESORT($A, k+1, j$)
5. ÖSSZEFÉSÜL(A, b, k, j)
7. return A

Összefésűléses rendezés

ÖSSZEFÉSÜL(A, b, k, j)

1. $n_1 := k - b + 1$
2. $n_2 := j - k$
3. for $p := 1$ to n_1
4. do $L_p := A_{b+p-1}$
5. for $r := 1$ to n_2
6. do $R_r := A_{k+r}$
7. $L_{n_1+1} := \infty$ ▷ strázsa
8. $R_{n_2+1} := \infty$ ▷ strázsa
9. $p := 1$
10. $r := 1$
11. for $i := b$ to j
12. do if $L_p \leq R_r$
13. then $A_i := L_p$
14. $p := p + 1$
15. else $A_i := R_r$
16. $r := r + 1$

Összefésűléses rendezés – elemzés

Legrosszabb esetben: $n \log n$

Külső rendezés (Knuth, 3. kötet)

A külső rendezés két lépésb?l áll:

- futamok el?állítása (angolul *run*)
- futamok összefésülése

két módszer:

1) többfázisú összefésülés
2) kaszkád összefésülés

Többfázisú összefésülés

3 szalagot használunk: T_1, T_2, T_3

1. Osszuk szét a kezdeti futamokat felváltva T_1-en és T_2-n.
2. A T_1 és T_2 szalagokról fésüljük össze a futamokat T_3-ra. Ha T_3 egyetlen futamot tartalmaz, álljunk meg.
3. Másoljuk a T_3-on lev? futamokat felváltva T_1-re és T_2-re, majd folytassuk a 2. lépéssel

1. fázis	1^8	1^5	–
2. fázis	1^3	–	2^5
3. fázis	–	3^3	2^2
4. fázis	5^2	3^1	–
5. fázis	5^1	–	8^1
6. fázis	–	13^1	–

k^n jelentése: n darab k hosszúságú futam (k hosszúság: az eredeti futam k-szorosa)

Elemzés 3 szalag esetében, Fibonacci elrendezésben

Ha r a futamok száma, akkor az átmásolások átlagos száma $1.04 \log r$.

Kaszkád összefésülés

	T1	T2	T3	T4	T5	T6	Feldolgozott kezdeti futamok
1. menet	1^{55}	1^{50}	1^{41}	1^{29}	1^{15}	–	190
2. menet	–	$1^{5}*$	2^{9}	3^{12}	4^{14}	5^{15}	190
3. menet	15^{5}	14^{4}	12^{3}	9^{2}	$5^{1}*$	–	190
4. menet	–	$15^{1}*$	29^{1}	41^{1}	50^{1}	55^{1}	190
5. menet	190^{1}	–	–	–	–	–	190

Knuth. 3. kötet

55 50 41 29 15
40 35 26 14
26 21 12
14 9
5

Gazdaságos faváz – Kruskal-algoritmus – Joseph Kruskal (1928–2010)

Legyen $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_1, e_2, \ldots, e_m\}$ és $G = (V, E, \mathcal{W})$ egy súlyozott egyszerű gráf úgy, hogy $\mathcal{W}(e_1) \leq \mathcal{W}(e_2) \leq \cdots \leq \mathcal{W}(e_m)$.

KRUSKAL(E)

```
1.  for j=1,2, ..., n
2.      do h_j := j
3.  i := 1
4.  while h elemei különbözőek
5.      do if (e_i végpontjai v_k, v_l) és (h_k ≠ h_l)
6.          then kiír e_i
7.               for j:=1, 2,..., n
8.                   do if h_j = h_l
9.                       then h_j := h_k
10.     i:=i+1
```

Legrosszabb esetben

Az élek rendezése: $\Theta(m \log m)$.
While ciklus $m\Theta(n^2)$. Lehet javítani:

4. sor: **while** $i \leq m$,

7–9. sor: láncolt lista: $\Theta(1)$ (konstans idő)

Ekkor ez m, tehát az algoritmus $\Theta(m \log m)$ legrosszabb esetben.

Gazdaságos faváz – Prim-algoritmus – Robert Clay Prim (1921–)

Legyen $G = (V, E, \mathcal{W})$ egy súlyozott egyszerű gráf. Az algoritmus kezdőcsúcsként az x-et használja.

PRIM(G, x)
1. $A := \{x\}$
2 $B := V \setminus A$
3. while $A \neq V$
4. do legyen $\{a, b\} \in E$, $a \in A$, $b \in B$ a legkisebb súlyú él az összes A és B közötti él közül
5. kiír $\{a, b\}$
6. $A := A \cup \{b\}$
7. $B := B \setminus \{b\}$

Legrosszabb esetben

Ez a változat így $\Theta(n^3)$.
De a 4. lépésben nem kell az összes élt megvizsgálni, elég A-ból csak a legutolsót, amely átkerült.
Így $(n-1)+(n-2)+\ldots 2 = \Theta(n^2)$.

Dijkstra algoritmusa – Edsger W. Dijkstra (1930–2002)

Legrövidebb út keresése, pozitív súlyok esetében

DIJKSTRA(G, u)

1. $S := \{u\},\ T := V \setminus S,\ l(u) := 0$
2. for minden $v \in V,\ v \neq u$
3. do $l(v) := \infty$
4. $x := u$
5. while $T \neq \emptyset$
6. do for minden $v \in N^{\text{ki}}(x) \cap T$
7. do if $l(v) > l(x) + w(x, v)$
8. then $l(v) := l(x) + w(x, v)$
9. $p(v) := x$
10. Legyen $x \in T$: $l(x) = \min_{y \in T} l(y)$
11. $S := S \cup \{x\},\ T := T \setminus \{x\}$
12. return l, p

A Dijkstra-algoritmus elemzése

Legrosszabb esetben: $\Theta(n^2)$.

Ha "minden csúcsból minden csúcsba" számítjuk, akkor $\Theta(n^3)$. Az egyik leggyorsabb algoritmus legrövidebb út keresésére.

Floyd–Warshall-algoritmus

Robert W. Floyd (1936–2001), Stephen Warshall (1935–2006)

$D_0 := (d_{ij}^{(0)})_{i,j=\overline{1,n}}$

$$\text{ahol} \quad d_{ij}^{(0)} = \begin{cases} \mathcal{W}(v_i, v_j) & \text{ha } \{v_i, v_j\} \in E \\ 0 & i = j \\ \infty & \text{ha } \{v_i, v_j\} \notin E, i \neq j \end{cases}$$

Kezdetben $p_{ij} := i$ ha $d_{ij} \neq \infty$ és $i \neq j$; más esetekben $p_{ij} := 0$.

FLOYD_WARSHALL(D_0)

1. $D := D_0$
2. for $k := 1$ to n
3. do for $i := 1$ to n
4. do for $j := 1$ to n
5. do if $d_{ij} > d_{ik} + d_{kj}$
6. then $d_{ij} := d_{ik} + d_{kj}$
7. $p_{ij} := p_{kj}$
8. return D, p

Egy v_i–v_j utat (az v_i csúcsból az v_j csúcsba) a következő algoritmussal határozzuk meg:

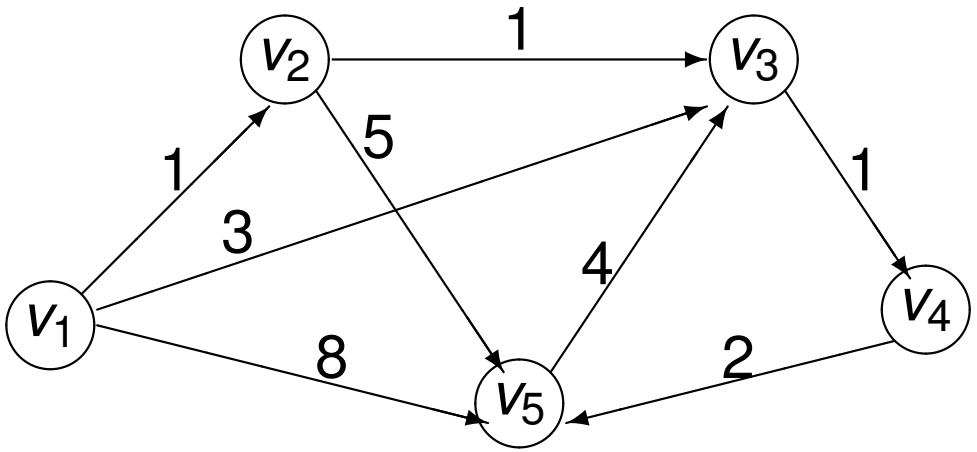
1. $k := n$
2. $u_k := j$
3. while $u_k \neq i$
4. do $u_{k-1} := p_{iu_k}$
5. $k := k - 1$

A keresett út: $v_{u_k}, v_{u_{k+1}}, \ldots, v_{u_n}$.

Az algoritmus elemzése

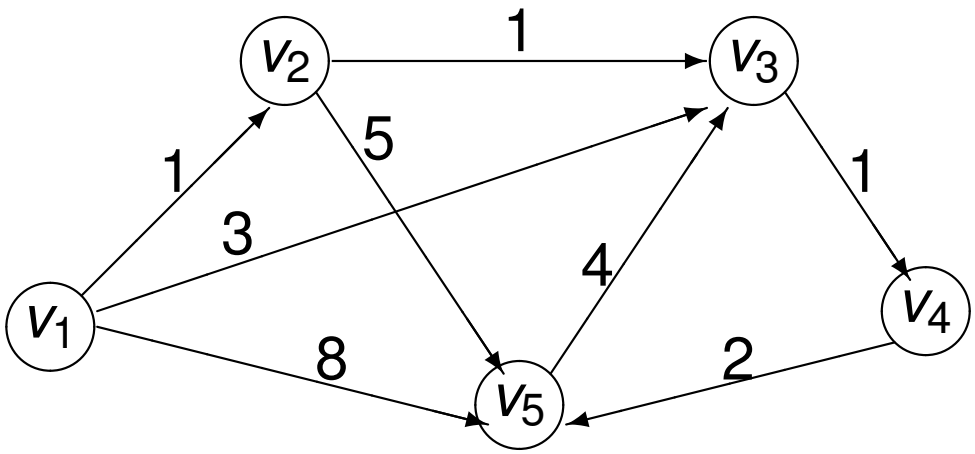
A három egymásba ágyazott ciklus miatt, a bonyolultság $\Theta(n^3)$, minden esetben.

Példa



A gráf szomszédsági mátrixa és a megfelelő P mátrix kezdeti értéke:

$$D_0 = \begin{pmatrix} 0 & 1 & 3 & \infty & 8 \\ \infty & 0 & 1 & \infty & 5 \\ \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & 4 & \infty & 0 \end{pmatrix} \qquad P_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$



Az algoritmus eredménye a D és P mátrixok:

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 5 \\ \infty & 0 & 1 & 2 & 4 \\ \infty & \infty & 0 & 1 & 3 \\ \infty & \infty & 6 & 0 & 2 \\ \infty & \infty & 4 & 5 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 2 & 3 & 4 \\ 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 5 & 0 & 4 \\ 0 & 0 & 5 & 3 & 0 \end{pmatrix}$$

Keressük a v_1–v_4 utat: $u_5 = 4$, $u_4 = 3$, $u_3 = 2$, $u_2 = 1$.
Az út: v_1, v_2, v_3, v_4.

Legrövidebb utak algoritmusainak összehasonlítása

algoritmus	bonyolultság	bonyolultság „*minden csúcsból minden csúcsba*”
Moore	$\Theta(n^2)$	$\Theta(n^3)$
Dijkstra	$\Theta(n^2)$	$\Theta(n^3)$
Ford	$\Theta(n^3)$	$\Theta(n^4)$
Bellman–Kalaba	$\Theta(n^3)$	$\Theta(n^4)$
Floyd–Warshall	$\Theta(n^3)$	$\Theta(n^3)$

Zárt Euler-vonal – Fleury-algoritmus (1883)

FLEURY(G, u)

1. válasszunk ki egy u-hoz illeszkedő e élt, amelyiknek másik végpontja v
2. $i := 1$
3. $w_i := e$
4. töröljük ki a gráfból az e élt
5. $u := v$
6. while van még él a gráfban
7. do válasszunk ki egy u-hoz illeszkedő e élt, amelyiknek másik végpontja v, és amelyik csak akkor lehet híd, ha nincs más lehetőség
8. $i := i + 1$
9. $w_i := e$
10. töröljük ki a gráfból az e élt
11. $u := v$
12. return $w_k, k = 1, 2, \ldots, i$

Fleury-algoritmus elemzése

A 6. sort m-szer (élek száma) végezzük el.

Híd ellenőrzése (pl. szélességi bejárással, komponens keresésével): $\Theta(m)$.

Így az algoritmus legrosszabb esetben $\Theta(m^2)$.

Ez, jobb hídkereső algoritmussal, valamelyest javítható.

Zárt Euler-vonal – Hierholzer-algoritmus (1873)

HIERHOLZER$_0(G, w)$ // w a kezdő csúcs

0. $u := w$
1. válasszunk ki egy u-hoz illeszkedő e élt, amelyiknek másik végpontja v
2. $i := 1$
3. $e_i := e$
4. töröljük ki a gráfból az e élt
5. $u := v$
6. while $v \neq w$
7. do válasszunk ki egy u-hoz illeszkedő e élt, amelyiknek másik végpontja v
8. $i := i + 1$; $e_i := e$
9. töröljük ki a gráfból az e élt
10. $u := v$
11. return $e_k, k = 1, 2, \ldots, i$

Hierholzer-algoritmus és elemzése

HIERHOLZER-algoritmus:

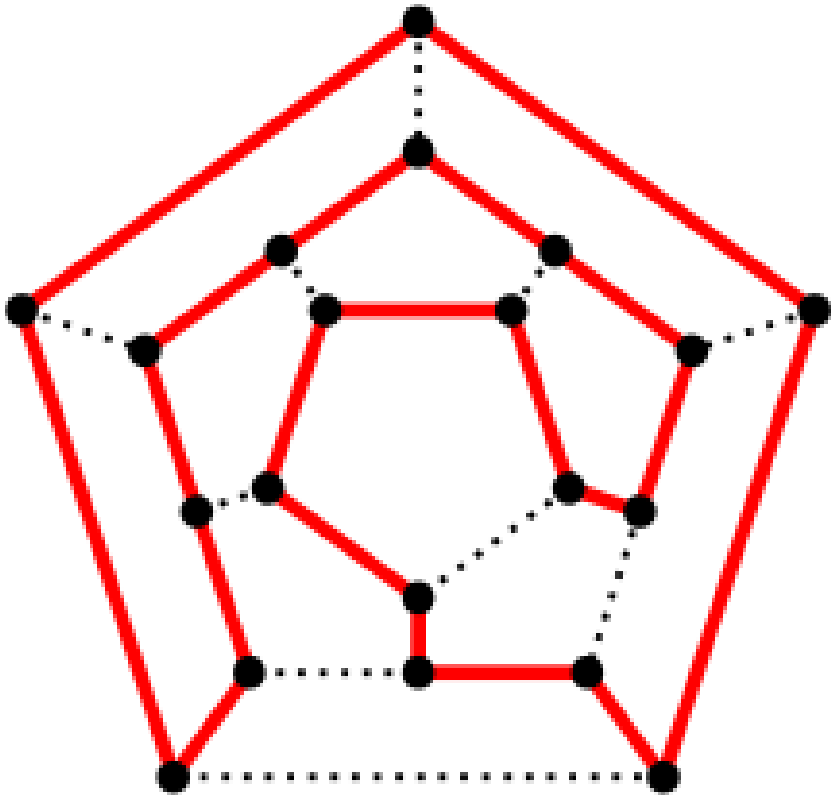
- Alkalmazzuk a HIERHOLZER$_0$ algoritmust egy tetszőleges csúcsból kiindulva.
- Ha a kapott zárt vonal nem tartalmazza a gráf minden élét, kiválasztunk a vonalon egy olyan csúcsot, amelyhez illeszkedik egy, a vonalon kívüli él is. Innen elindulva ezen az élen, újra alkalmazzuk a HIERHOLZER$_0$ algoritmust.
- Mindaddig folytatjuk, amíg az összes élt nem felhasználjuk.

Elemzés:

Ha megőrizzük (pl. listában) a vonalon kívüli, de hozzá illeszkedő éleket, akkor az algoritmus bonyolultsága $\Theta(m)$, azaz lineáris az élek számához viszonyítva.

Hamilton-kör

A Hamilton-kör keresése NP-teljes feladat.



(Wikipédia)

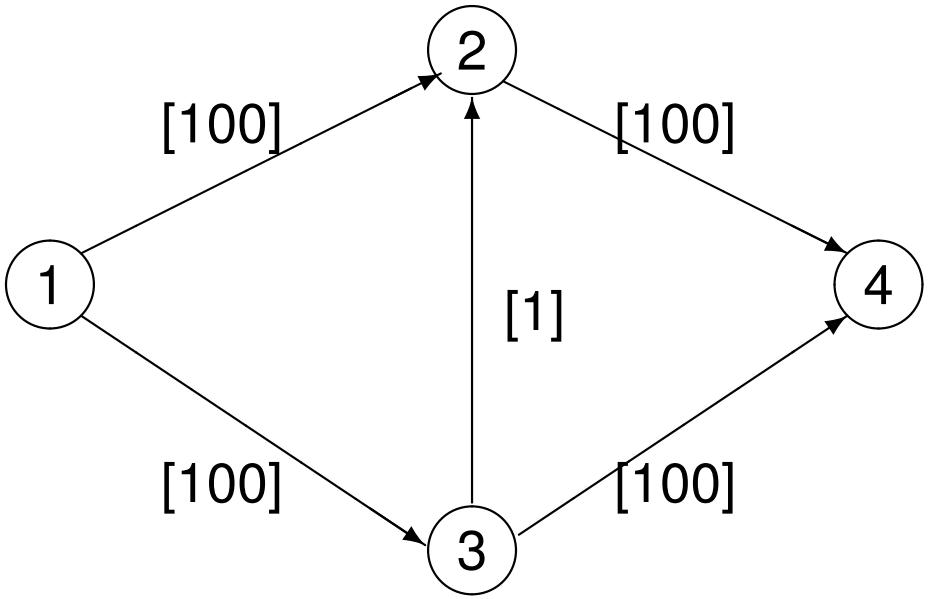
- nincs szükséges és elegséges feltétel

Dirac Gábor tétele: *Egyszerű gráfban, ha minden csúcs foka* $\geq n/2$*, akkor létezik Hamilton-út.*
Ore tétele: *Egyszerű gráfban, ha minden nem szomszédos csúcspárra* $\varphi(x) + \varphi(y) \geq n$*, akkor létezik Hamilton-út.*

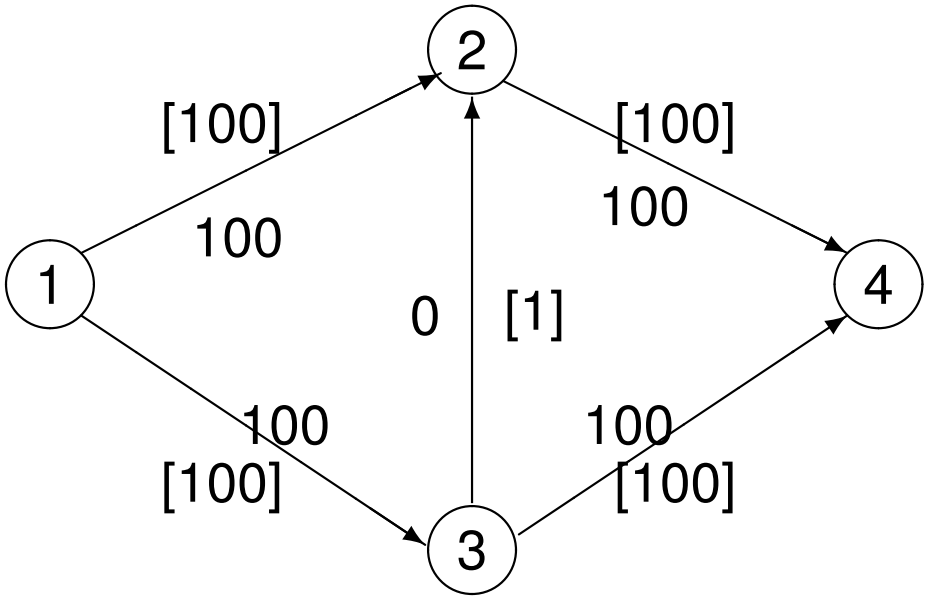
- nincs polinomiális algoritmus

A Ford–Fulkerson-algoritmus elemzése

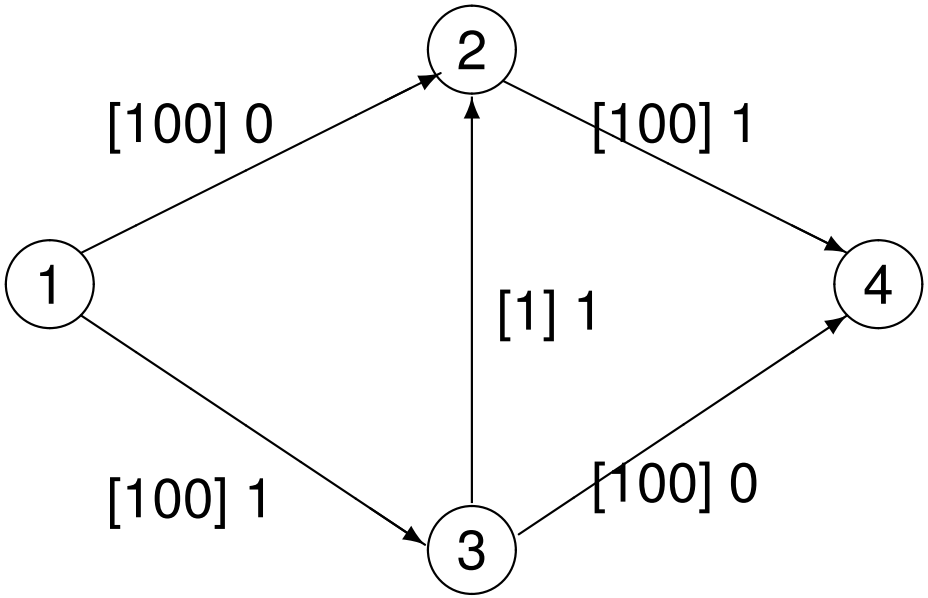
Tekintsük a következő példát:

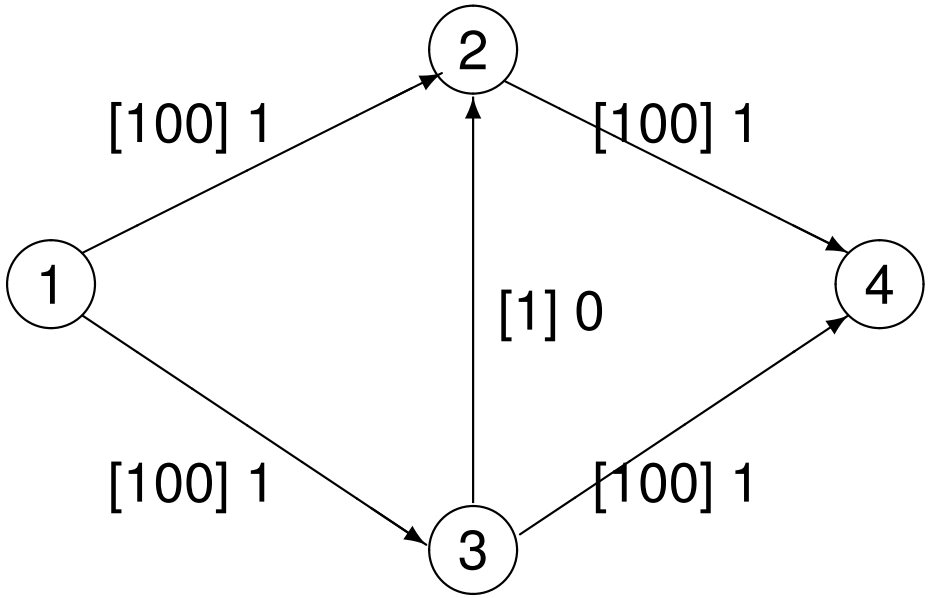


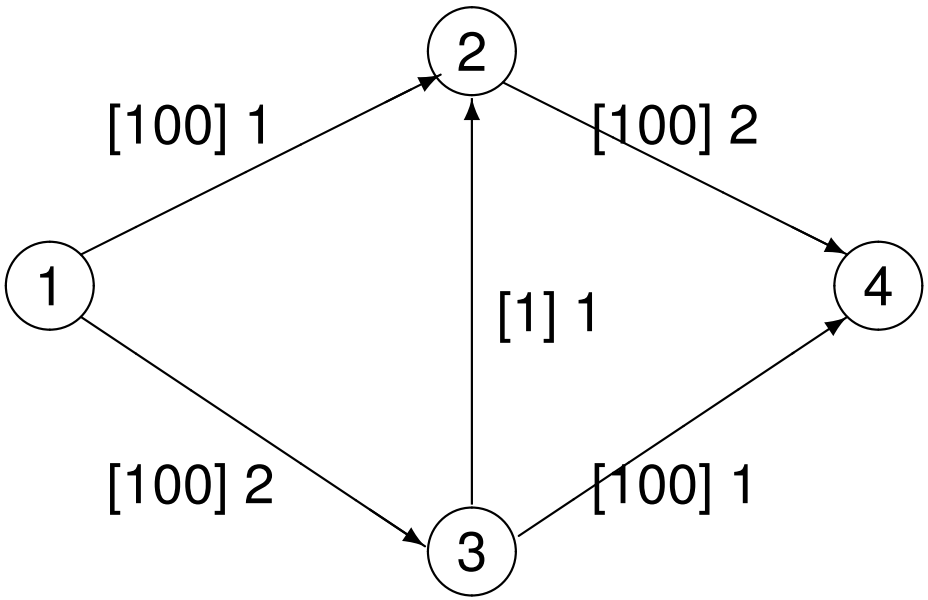
A Ford–Fulkerson-algoritmust alkalmazva a megoldást két lépésben megkaphatjuk. Nulla értékű folyammal indulva, az $1, 2, 4$ úton növelni lehet a folyamot 100-zal, aztán hasonlóképpen az $1, 3, 4$ úton is. Tehát a maximális folyam értéke 200.

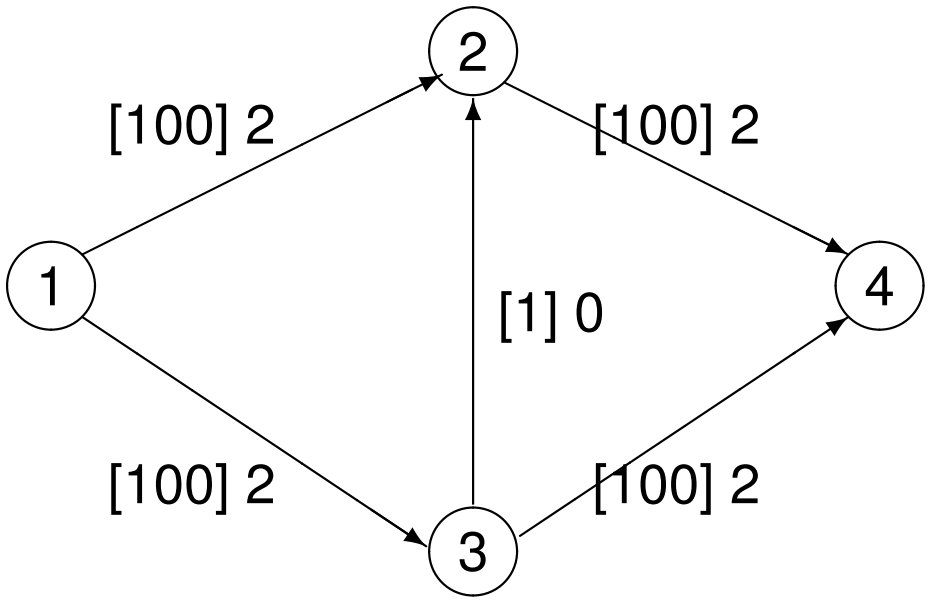


De az algoritmus ugyanúgy választhatja először az $1, 3, 2, 4$ utat, amelyen a folyam 1-gyel növelhető. Ezután, a $1, 2, 3, 4$ láncot használva, a folyamot szintén 1-gyel lehet növelni.









Így folytatva, az eredményt 200 lépésben kapjuk meg. Az algoritmus bonyolultsága ily módon függ a folyam értékétől.
Ha $m = |E(G)|$, a hálózat éleinek száma, $v(f)$ pedig a folyam értéke, akkor az algoritmus bonyolultsága $O(mv(f))$ vagy $O(n^2 v(f))$, ha n a hálózat csúcsainak a száma (pszeudopolinomiális algoritmus).

Az algoritmus javítható, ha minden alkalommal a lehetséges láncok közül a legrövidebbet (legkevesebb élből állót) választjuk. Ha a javítóláncot szélességi kereséssel határozzuk meg, akkor mindig a legrövidebbet kapjuk.

A Ford–Fulkerson-algoritmusnak ezt a változatát Edmonds–Karp-algoritmusnak nevezzük. Ennek a bonyolultsága $O(nm^2)$ vagy $O(n^5)$.

Cormen–Leiserson–Rivest–Stein: Új algoritmusok

Minimális értékű maximális párosítás

Súlyozott teljes páros gráfokat vizsgálunk.
A párosítás értéke = a párosítás élei súlyának az összege

A magyar módszer

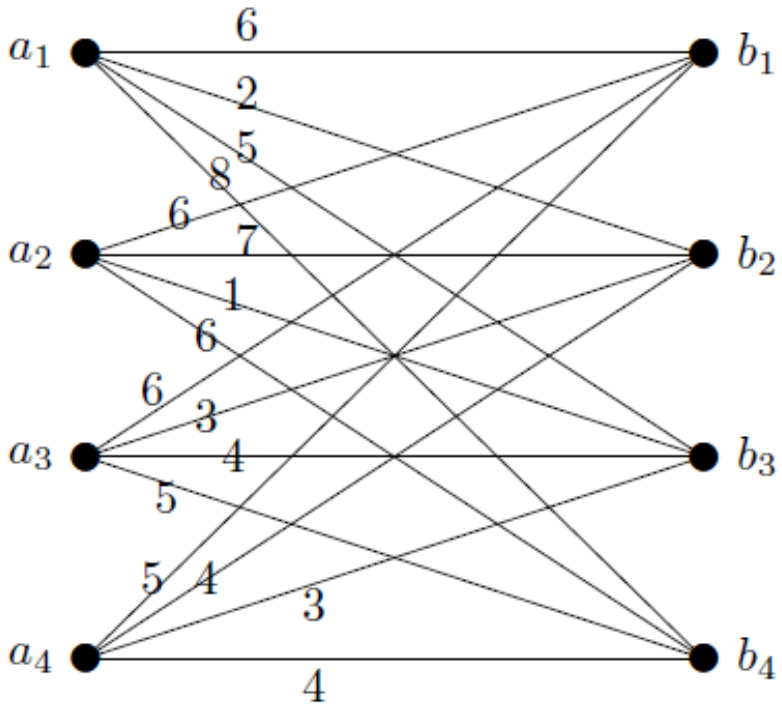
(*Harold Kuhn, 1955,* Kőnig Dénes és Egerváry Dénes tiszteletére)

teljes, súlyozott gráf esetében: $K_{n,n}$

1. Minden sorból és oszlopból kivonjuk a legkisebb értéket, hogy legyen legalább egy 0 mindegyikben.
2. Keressünk maximális számú független 0-t.
3. Amíg a független 0-k száma nem n, végezzük el:
   3.1. Minimális lefedés, módosítás.
   3.2. Keressünk maximális számú független 0-t.

Az algoritmus bonyolultsága: $O(n^3)$.

Példa.



6	2	5	8
6	7	1	6
6	3	4	5
5	4	3	4

Kivonunk 2-t az első sorból, 1-et a másodikból, 3-at a harmadik és negyedik sorból.

4	0	3	6
5	6	0	5
3	0	1	2
2	1	0	1

Kivonunk 2-t az első sorból, 1-et a 4. oszlopból:

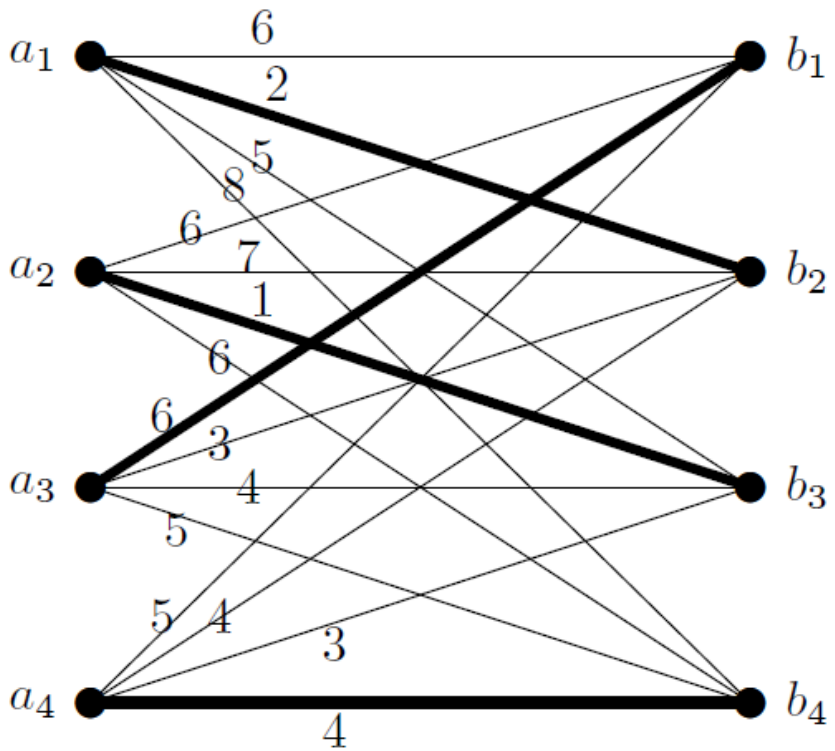
2	0	3	5
3	6	0	4
1	0	1	1
0	1	0	0

Csillaggal megjelölünk max. számú független 0-t. Három (zöld) vonallal (két függőleges, egy vízszintes) lefedjük a 0-kat.

2	0*	3	5
3	6	0*	4
1	0	1	1
0*	1	0	0

Az elemek módosítása (Kivonunk 1-et — a legkisebb fedetlen elem — az összes fedetlen elemből, nem módosítjuk az egy vonallal lefedett elemeket, hozzáadunk 1-et a kétszer lefedett elemekhez.) Csillaggal megjelölünk max. számú független 0-t.

1	0*	3	4
2	6	0*	3
0*	0	1	0
0	2	1	0*



A max. párosítás értéke: $2 + 1 + 6 + 4 = 13$.