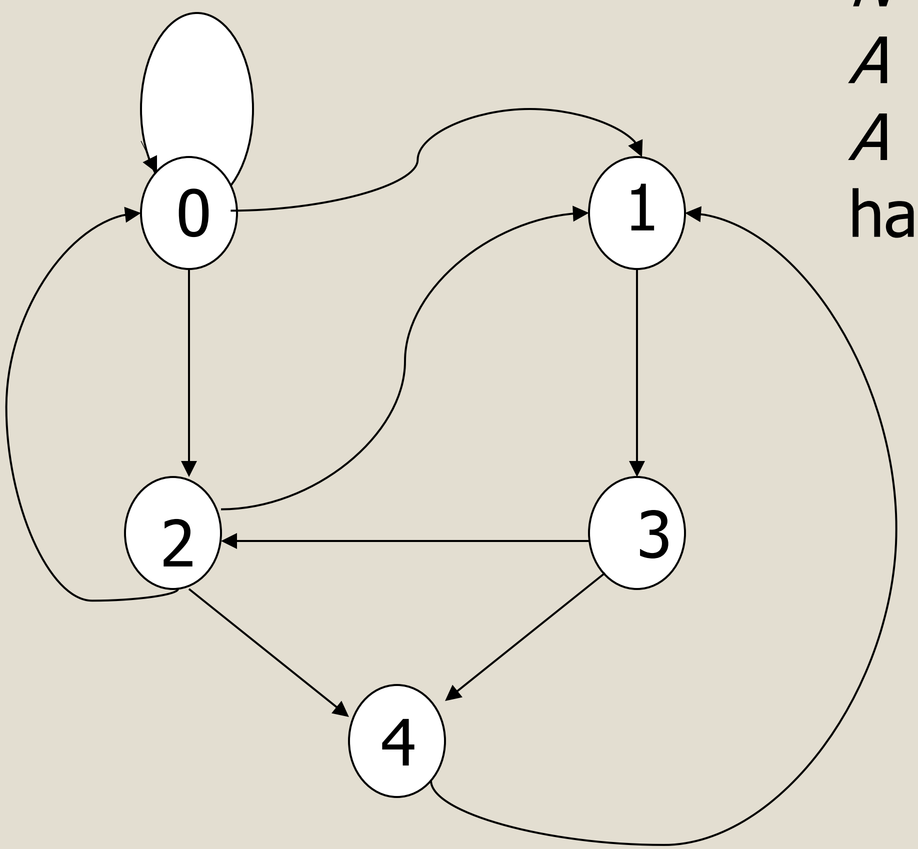
ADATSZERKEZETEK ÉS ALGORITMUSOK

Python nyelven

Gráf adatszerkezet

Irányított gráf 1.

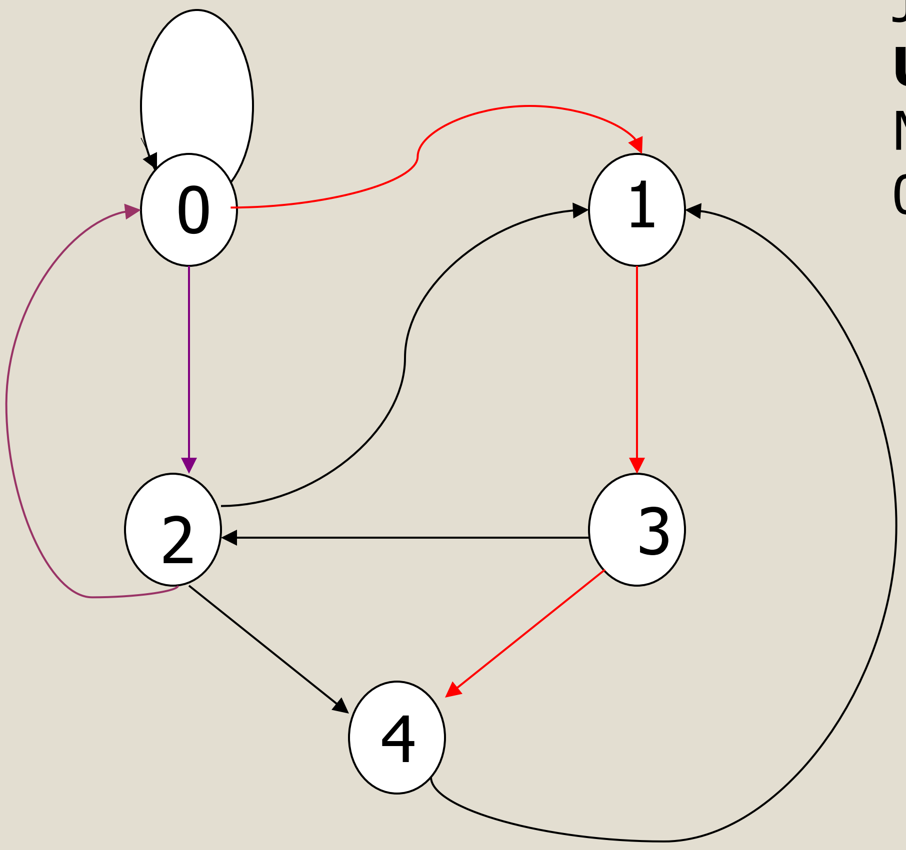


N a csúcsok halmaza
A az élek halmaza
A egy reláció *N*x*N* halmazon

$N = \{0, 1, 2, 3, 4\}$

$A = \{(0,0), (0,1), (0,2), (1,3), (2,0), (2,1), (2,4), (3,2), (3,4), (4,1)\}$

Irányított gráf 2.



Példa ciklusra: ~~(0,0)~~ vagy (0, 2, 0)

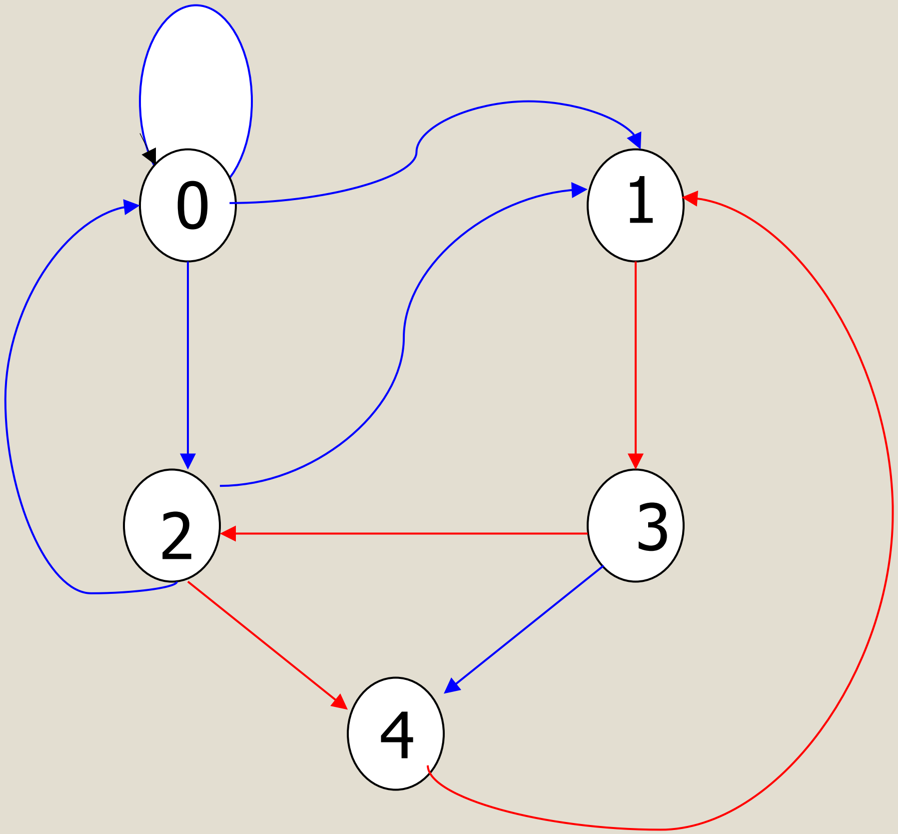
Példa útra:
{(0,1), (1,3), (3,4)}
Jelölés: (0,1,3,4)
Út hossza: 3
Minden csúcsból önmagába 0 hosszú út vezet.

Ha (i,j) él, akkor i ***szülő**je* j-nek, és j ***gyermek**e* i-nek

Más jelöléssel i $\mapsto$ j

Ciklus: olyan nem-triviális, azaz pozitív hosszú út, melynek kezdete és vége megegyezik.
Egy csúcs **fokszám**a:
A belőle kiinduló élek száma

Irányított gráf 3.



A ciklust bármelyik pontján kezdhetjük: (1,3,2,4,1) ekvivalens (2,4,1,3,2) -vel. A ciklus *egyszerű*, ha csak a kiinduló pontban van ismétlődés. Pl. a „piros" ciklus egyszerű, a (0,2,0,1,3,2) ciklus nem egyszerű.

Minden ciklus tartalmaz ugyanonnan induló egyszerű ciklust is.

Irányított gráf 4.

- Minden ciklus tartalmaz ugyanonnan induló egyszerű ciklust is.

Bizonyítás.

Legyen **(v, v_1, v_2,…, v_k, v**) egy nem egyszerű ciklus. Ekkor

1. vagy v előfordul legalább háromszor,
2. vagy létezik egy olyan **u**, melyre a ciklus a következő alakú:
   (v,…,u,…,u,…,v).

Az 1. esetben távolítsunk el mindent a ciklusból **v** utolsó előtti előfordulásáig, a fennmaradó rész egyszerű ciklust alkot.

A 2. esetben a két **u** közötti részt távolítsuk el, és helyettesítsük egyetlen **u**-val. Ekkor egy az eredetinél rövidebb ciklust kapunk.

Ha ez nem egyszerű, ismételjük az eljárást elölről. Véges sok lépés után az eljárás véget ér.

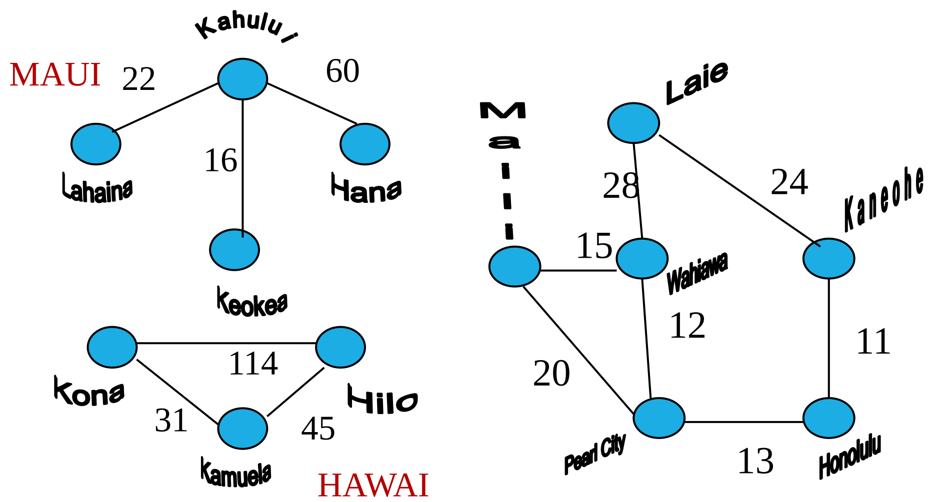
Irányított gráf 5.

Ha egy gráf tartalmaz ciklust, a gráfot *ciklikusnak* nevezzük. A fentiek szerint egy gráf pontosan akkor ciklikus, ha tartalmaz egyszerű ciklust.

Ha egy gráf nem tartalmaz ciklust, akkor *aciklikusnak* nevezzük.

Egy út aciklikus, ha nem tartalmaz ciklust. (Labirintusnál ezt egyszerű útnak hívtuk.) Ha két csúcs között van út, akkor aciklikus út is van, hiszen egy ciklust helyettesíthetünk a kezdőpontjával.

IRÁNYÍTATLAN GRÁF 1: A HAWAI SZIGETEK ÚTHÁLÓZATA



OAHU

Irányítatlan gráf 2

- az **él** nem (u,v) rendezett pár, hanem {u,v} halmaz. Ekkor **u** és **v** szomszédosak.
- Ciklus definiálása: (u,v,u) –t kihagyjuk, hiszen csak ua. élen megyünk oda-vissza
- Csak az ***egyszerű ciklust*** definiáljuk: olyan legalább **3 hosszú** út, amelynek kezdő- és végcsúcsa megegyezik, és az utolsó csúcs kivételével nincs ismétlődés
  (Maili, Wahiawa, Pearl City, Maili) vagy másképp
  (Wahiawa, Pearl City, Maili, Wahiawa) vagy másképp
  (Pearl City, Maili, Wahiawa, Pearl City)
  és még fordított körüljárással is.
-

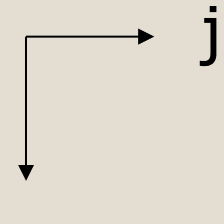
Irányított gráf ábrázolása szomszédsági mátrixszal

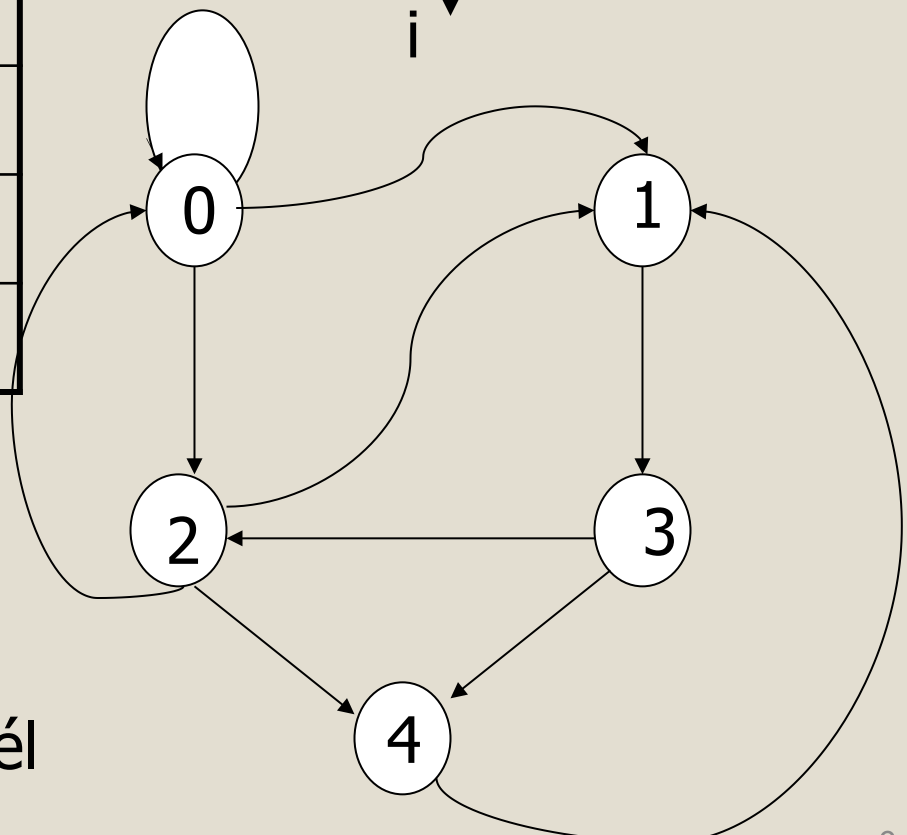
hova

honnan

	0	1	2	3	4
0	1	1	1	0	0
1	0	0	0	1	0
2	1	1	0	0	1
3	0	0	1	0	1
4	0	1	0	0	0

Jelölje E a szomszédsági mátrixot.



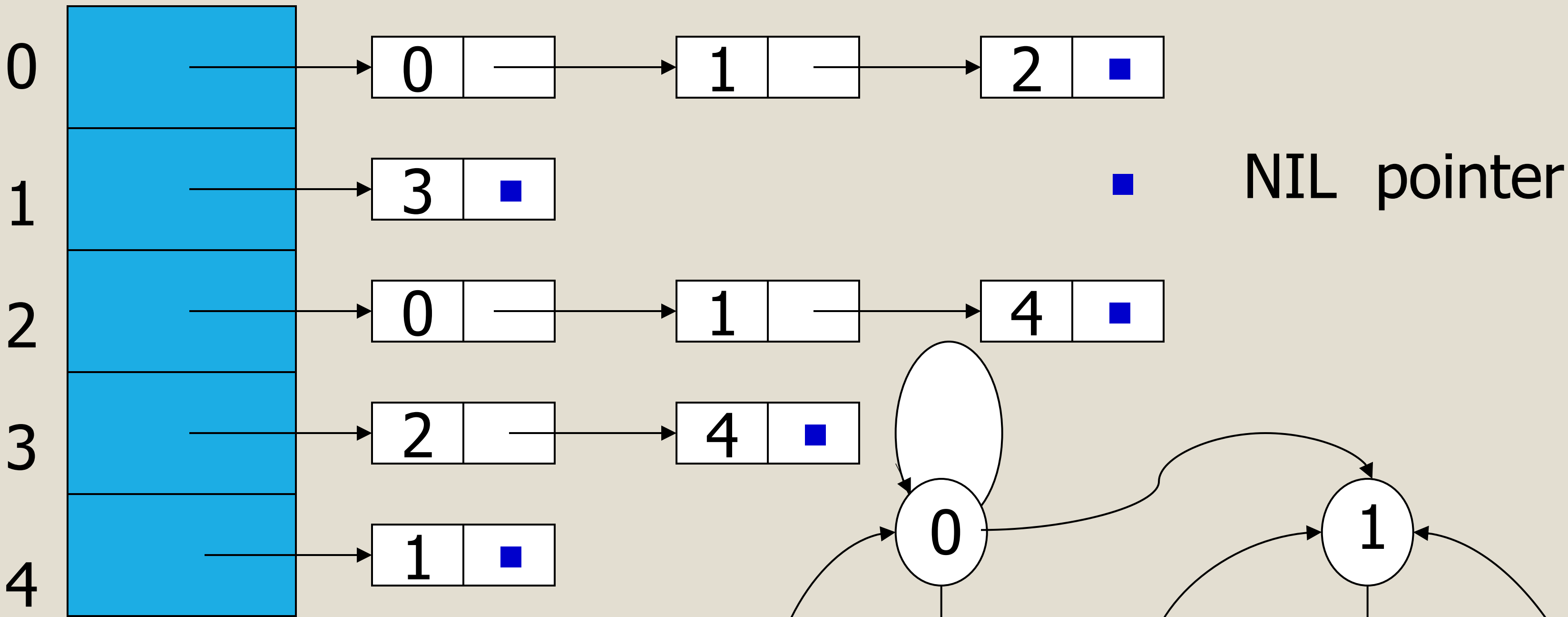


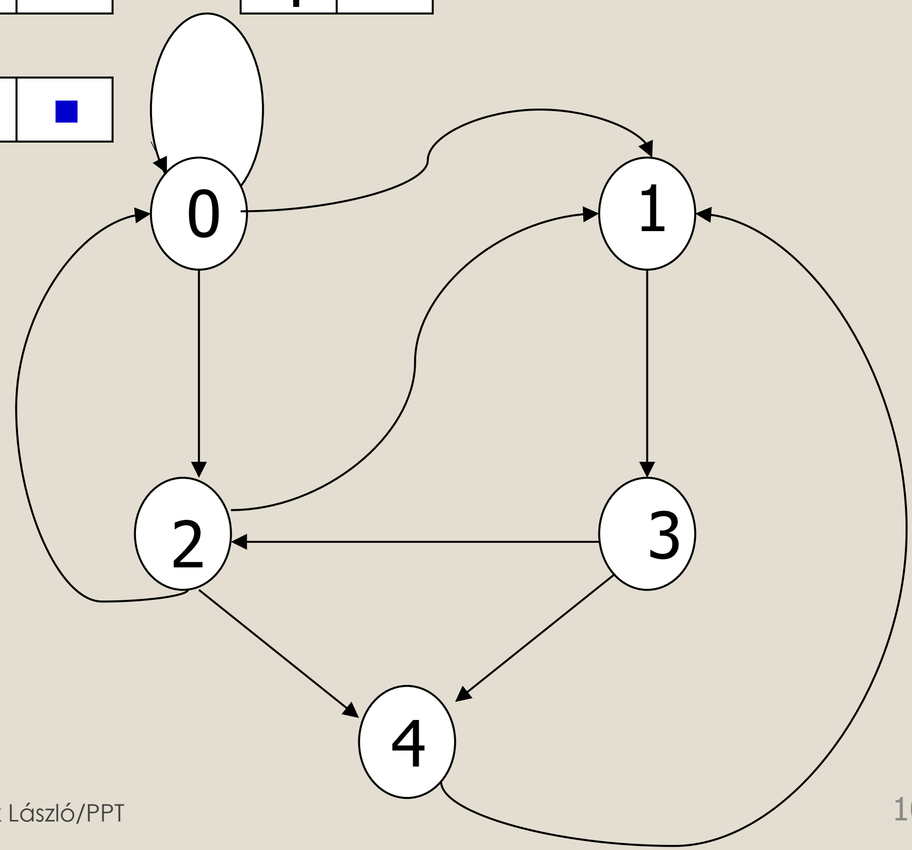
$E(i,j)=1 \Longleftrightarrow i \mapsto j$

i-dik sor: hova megy i-ből él

j-dik oszlop: honnan jön j-be él

Irányított gráf ábrázolása szomszédsági listával





Pointer-tömb tartalmazza a csúcsokat, a listákon a csatlakozó utódok vannak (a sorrend mindegy)

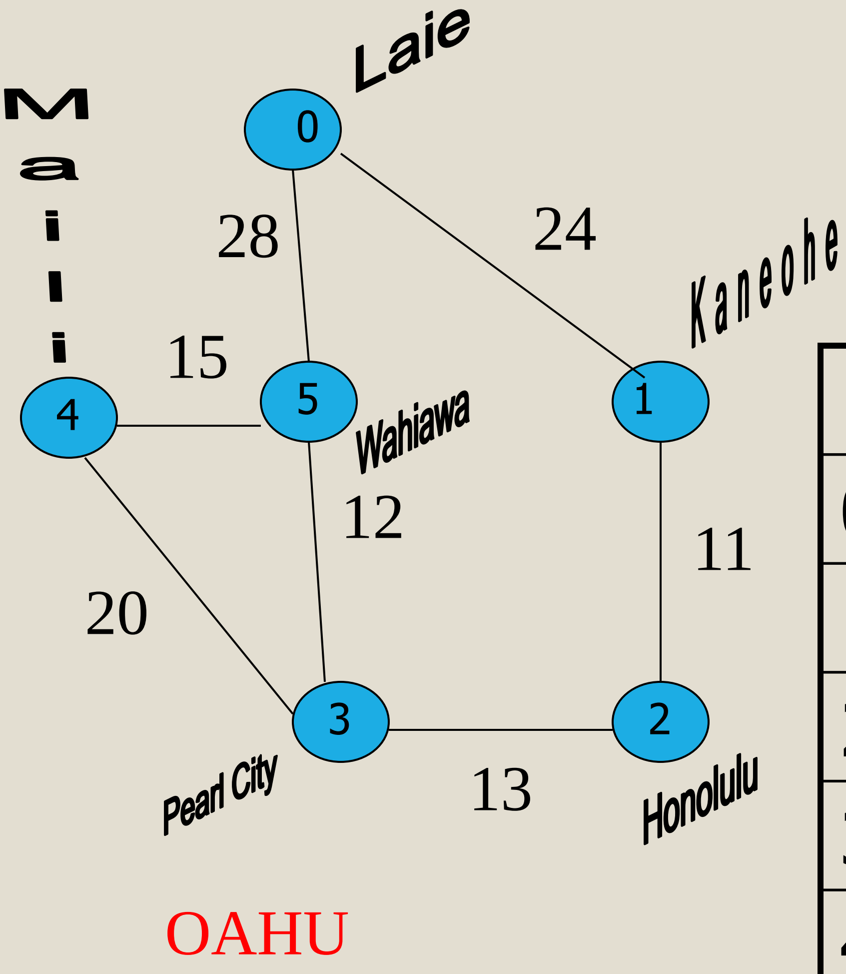
Listával vagy mátrixszal?

Ha a gráf sűrű, azaz a lehetséges élek közül sok van behúzva (max n^2 lehet), akkor a mátrixos tárolás takarékosabb. (n^2 bit szükséges),
Tegyük fel, hogy a gráf ritka, azaz viszonylag kevés él van benne. Egy listaelem tartalmaz egy *integert* és egy *pointert* (32+32=64 bit). x db él esetén ez $64x$ bit. További $32n$ bitet igényel a pointertömb, n csúcs esetén.

$$32n + 64x \langle n^2 \quad \Rightarrow \quad x \langle \frac{n^2}{64} - \frac{n}{2}$$

Nagy n esetén, ha az összes lehetséges él 64-edénél kevesebb él van, akkor a lista megéri.

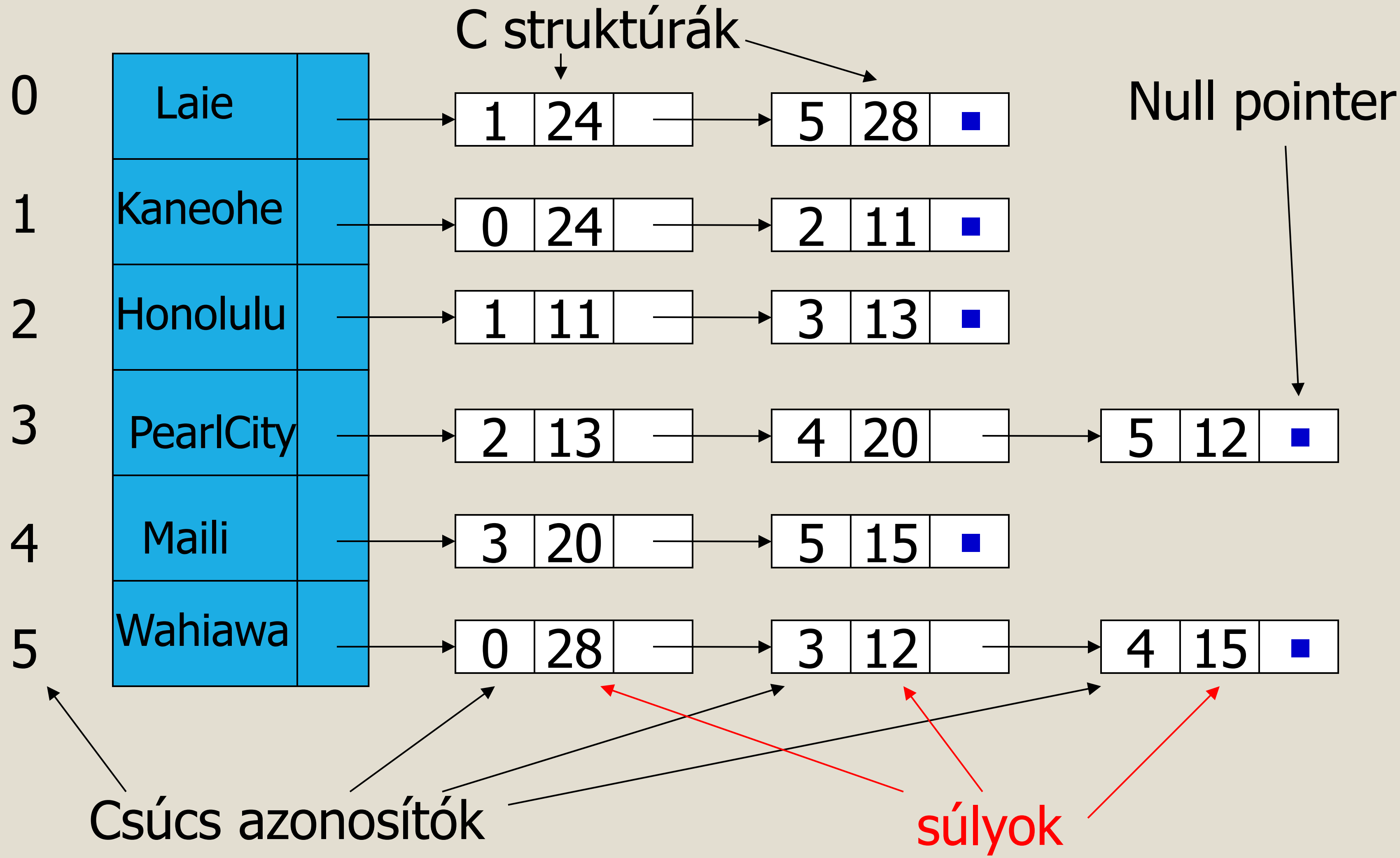
Súlyozott élű irányítatlan gráf szomszédsági mátrixa



A szomszédsági mátrixba a súlyt írjuk be. Kell egy érték (példánkban –1) amely a *nincs kapcsolatot* jelenti.

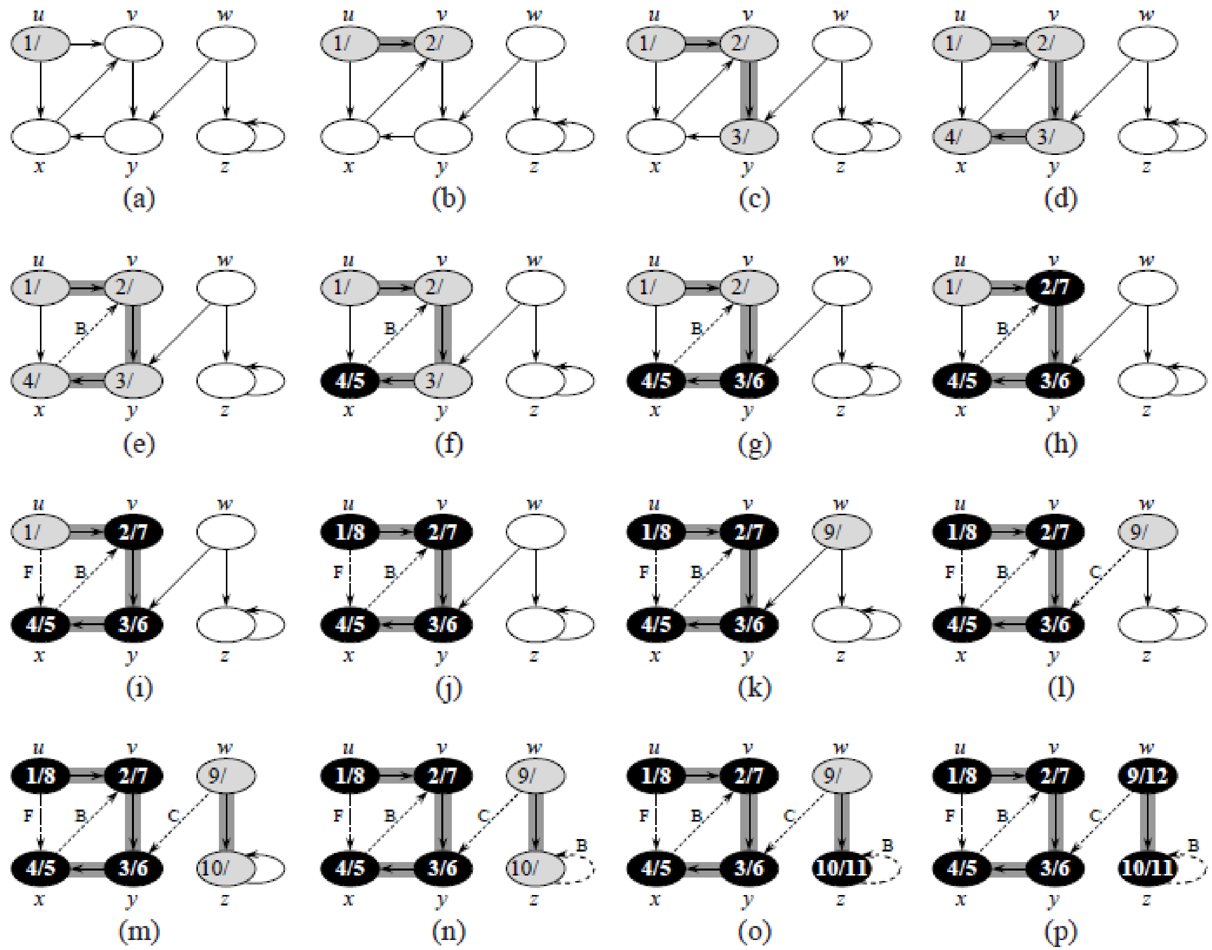
	0	1	2	3	4	5
0	-1	24	-1	-1	-1	28
1	24	-1	11	-1	-1	-1
2	-1	11	-1	13	-1	-1
3	-1	-1	13	-1	20	12
4	-1	-1	-1	20	-1	15
5	28	-1	-1	12	15	-1

Súlyozott élű irányítatlan gráf szomszédsági listája



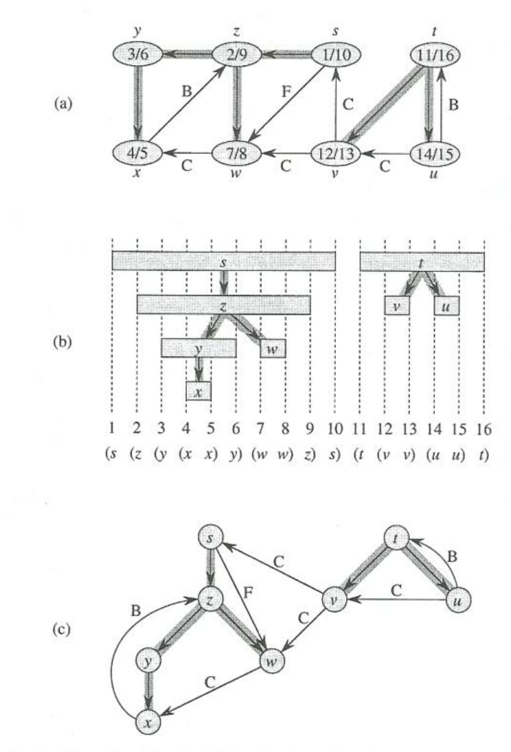
Gráf mélységi bejárása
Cél: mélységi fa kialakítása

- Ötlet:
- Csúcsok színezése és
- időadatokkal bővítése



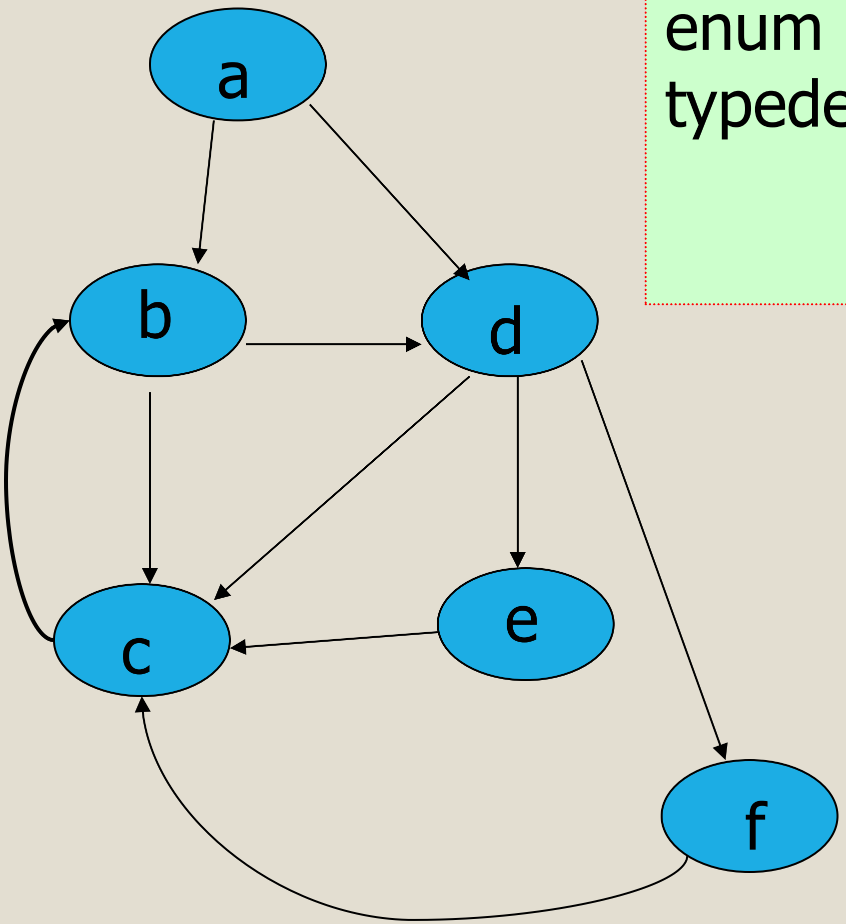
Mélységi bejárás tulajdonságai

- Élek típusai
- MK erdő



Gráf mélységi bejárása (Depth-First Search: irányított gráf bejárása)

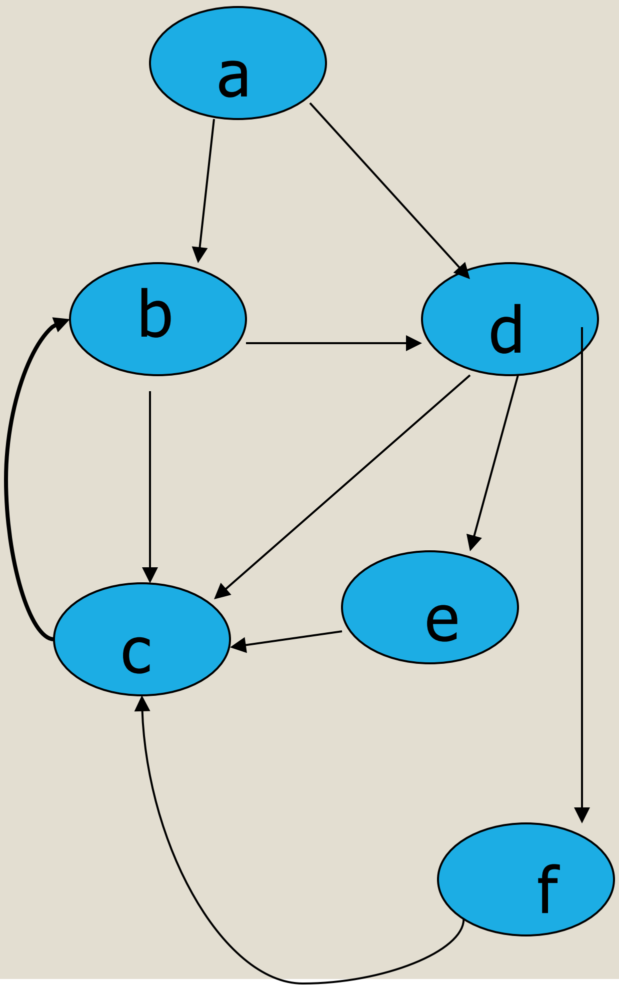
Nem Ariadné fonala, mert ott irányítatlan gráf volt, de az ötlet hasonló




```
enum MARKTYPE{VISITED,UNVISITED}
typedef struct{ enum MARKTYPE mark;
          LIST successors;
         } GRAPH[MAX];
```

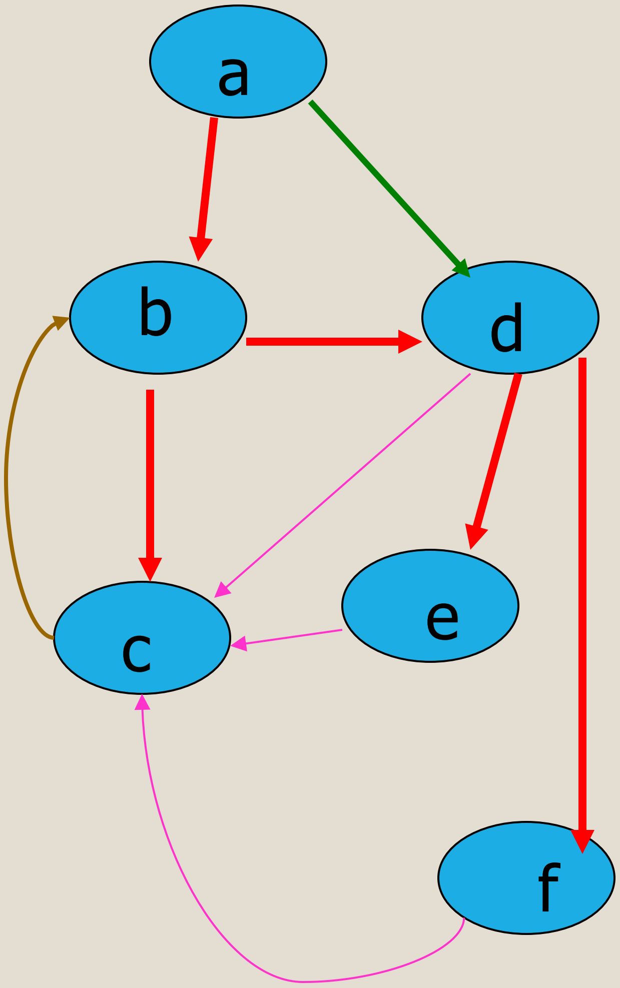
```
typedef struct CELL *LIST;
struct CELL {
     NODE nodeName;
     LIST next;
};
```

A program (rekurzív dfs)




```
void dfs(NODE u, GRAPH G){
LIST p; // u szomszédsági listáján fut
NODE v; // p által mutatott csúcs
      G[u].mark=VISITED;
      p=G[u].successors;
      while (p != NULL) {
            v = p->nodeName;
            if (G[v].mark==UNVISITED)
                  dfs(v, G);
            p = p->next;   }
}
```

dfs fa keresése 1.



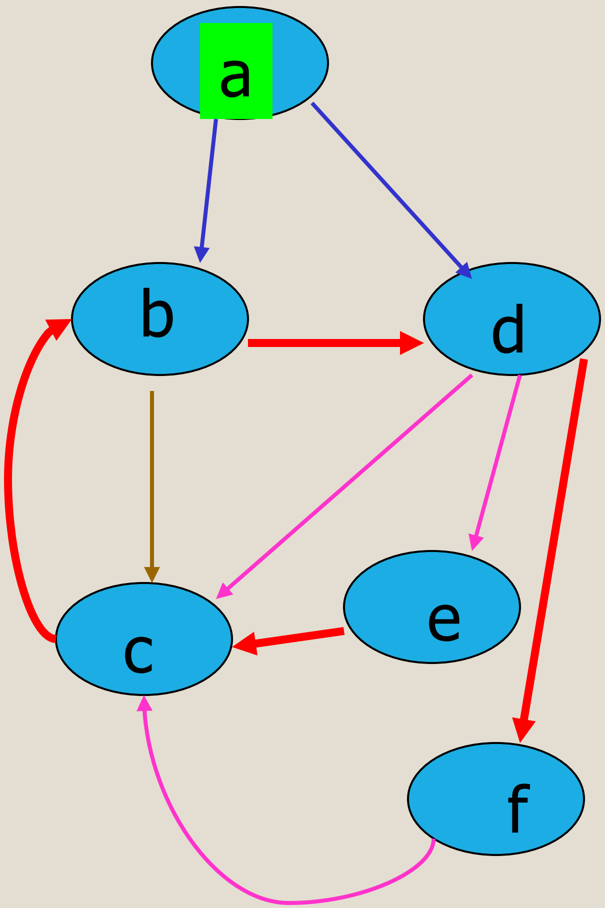
a *piros nyilak* a szülő-gyerek kapcsolatot jelentik a dfs *fában*

előre mutató él: a→d
(de d nem gyereke a-nak)
visszamutató él: c→b
(c leszármazottja b-nek)
keresztélek:
a többi
mindegyik keresztél jobbról balra megy, vagyis később látogatott éltől korábban látogatott élig

dfs fa keresése 2.

Mi lett volna, ha nem az a csúcsból indulunk?
Az a csúcsot sosem találtuk volna meg.
Ekkor több fát találunk, amelyek **erdőt** alkotnak.

Pl. induljunk az e csúcsból
az eredmény a piros fa, ill. az a csúcs
önmagában (elfajult fa)



Előremutató él nincs
visszamutató él b b →c
Keresztélek: itt is jobbról balra
később látogatott éltől korábban
látogatott élig mennek

Az általános dfsForest algoritmus

```
void dfsForest( GRAPH G){
NODE u;
for (u=0; u<MAX; u++)
     G[u].mark=UNVISITED;
for (u=0; u<MAX; u++)
     if (G[u].mark==UNVISITED)
     dfs(u, G);
}
```

Forest = erdő

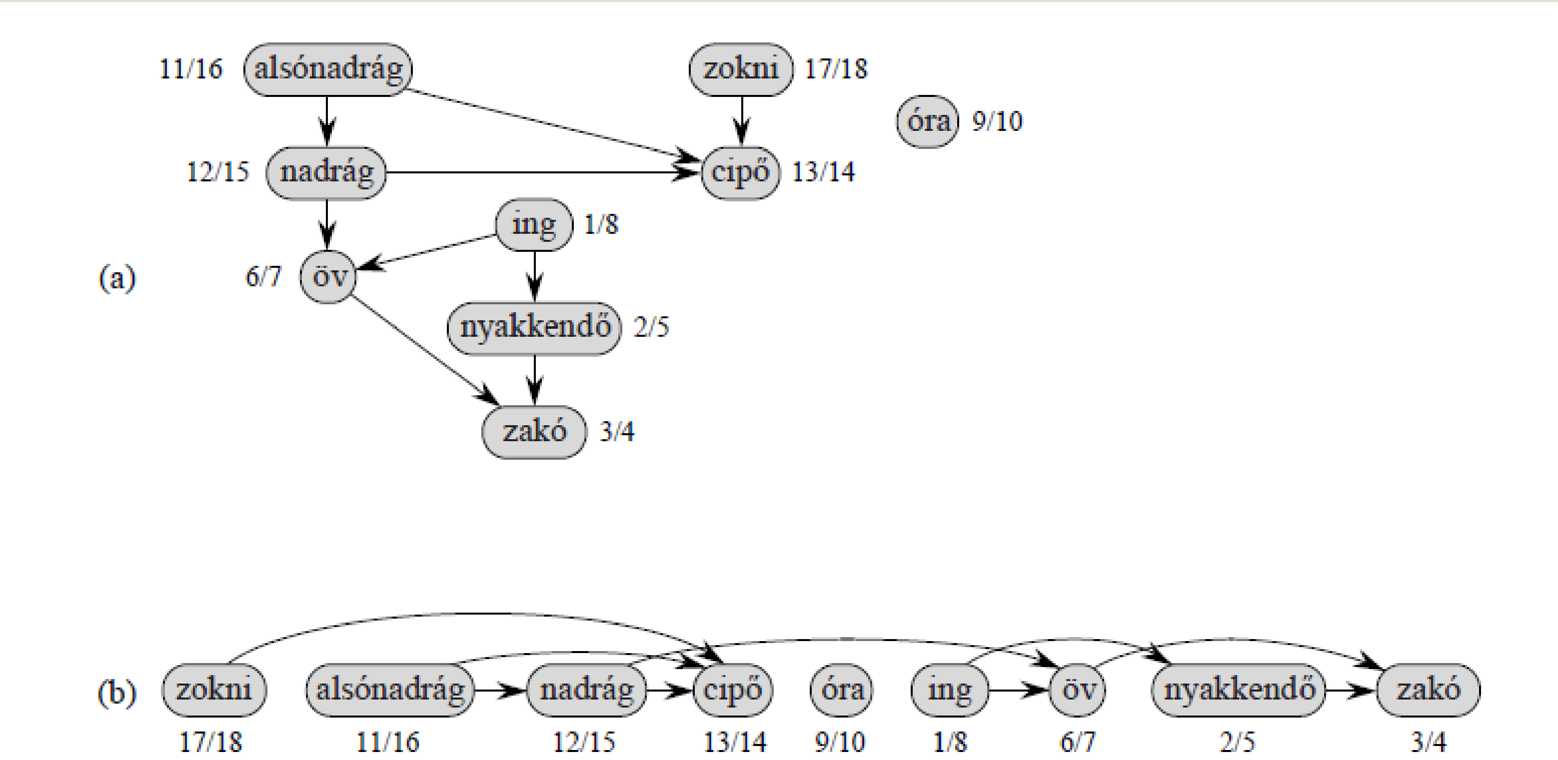
Irányított gráf postorder bejárása

```
int k; // csúcsok számolása
void dfs(NODE u, GRAPH G){
LIST p; // u szomszédsági listáján fut
NODE v; // p által mutatott csúcs
G[u].mark=VISITED;
   p=G[u].successors;
   while (p != NULL) {
     v = p->nodeName;
     if (G[v].mark==UNVISITED)
          dfs(v, G);
     p = p->next;   }
  ++k;
 G[u].postorder=k;
}
```

Új: ami piros
k globális változó
„visszafelé" számozza be a csúcsokat

```
void dfsForest( GRAPH G){
NODE u;
k = 0;
for (u=0; u<MAX; u++)
   G[u].mark=UNVISITED;
for (u=0; u<MAX; u++)
   if (G[u].mark==UNVISITED)
     dfs(u, G);
}
```

Topológiai rendezés



Topológiai rendezés: egy *részben rendezést* *teljes rendezéssé* egészít ki

Tegyük fel, hogy valamilyen folyamatban a feladatok időrendi precedenciájára vonatkozó szabályaink vannak, melyet az R részben rendezési reláció fejez ki. **Ennek lezárása** teljes rendezéssé - ezt nevezik topológiai rendezésnek - megadja a feladatok elvégzésének olyan lehetséges ütemezéseit, amelyek a kezdeti feltételeket kielégítik.

A topológiai rendezés tehát nem egyértelmű, szemben a **tranzitív lezárással, amely egyértelmű** eredményt ad.

Topológiai rendezés: egy egyszerű példa

Az orvos a műtőben egyes műtétekkor a sebészkesztyű alá fémszálakkal megerősített „vaskesztyűt” is húz, amely védi a szikével történő megvágás ellen.
Adott precedencia: (bal vaskesztyű, bal sebészkesztyű), (jobb vaskesztyű, jobb sebészkesztyű). Ez egy **részben rendezés**.
Lehetséges teljes rendezések:
Muszáj vaskesztyűvel kezdeni, ≤ jelenti a sorrendet.

Jobb vaskesztyűvel indít:	**Bal** vaskesztyűvel indít:
JVK ≤ JSK ≤ BVK ≤ BSK	BVK ≤ JVK ≤ JSK ≤ BSK
JVK ≤ BVK ≤ JSK ≤ BSK	BVK ≤ JVK ≤ BSK ≤ JSK
JVK ≤ BVK ≤ BSK ≤ JSK	BVK ≤ BSK ≤ JVK ≤ JSK

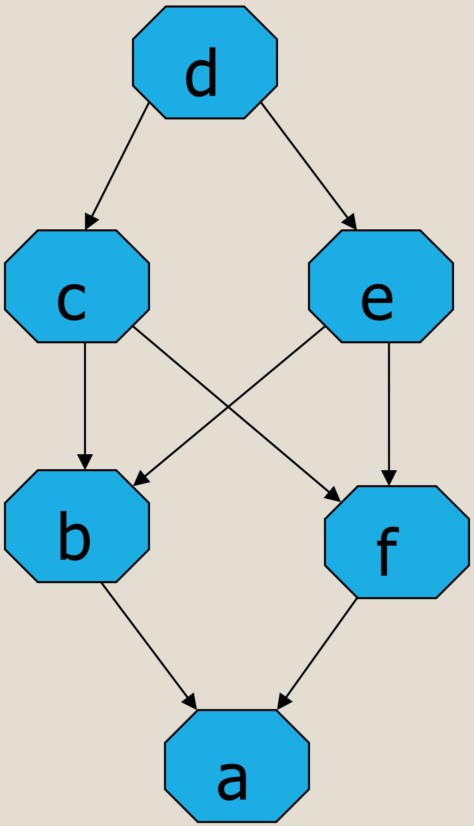
Topológiai rendezési algoritmus

- Legyen G egy aciklikus irányított gráf. (Csak aciklikus gráfot lehet topológiailag rendezni.)
- A dfsForest algoritmussal segítségével (ld. Irányított gráf postorder bejárása) fordított postorder sorrendet is kaphatunk: $(v_1,v_2,\ldots,v_n)$ ahol v_1 a postorder bejárás szerinti n-dik csúcs, v_2 az (n-1)-dik csúcs stb.
- Ebben a listában minden él előre mutat, tehát ez a lista egy topológiai rendezést valósít meg. A fordított postorder sorrendet FILO veremmel valósítjuk meg.

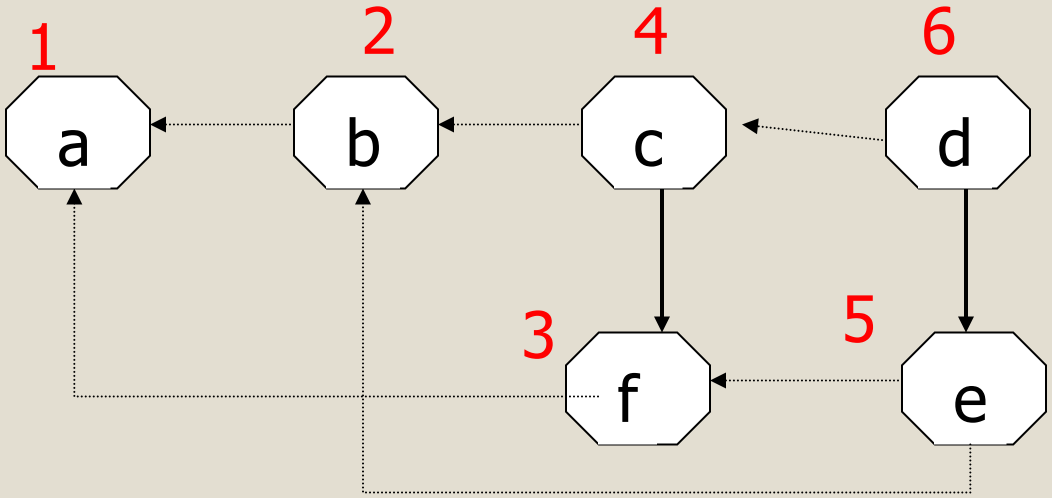
Topológiai rendezés **alkalmazása**

- Tegyük fel, hogy egy programban a nem-rekurzív függvényhívásokat egy G gráf írja le. Célunk, hogy a függvényeket olyan sorrendben elemezzük, hogy egy függvény elemzésekor az általa hívott függ-vényeket már áttekintettük.
- Megoldás: a postorder (fordított topológiai) rendezés
- Mindig szükséges viszont az aciklus teszt a rekurzivitás felderítésére.

Példa topológiai rendezésre



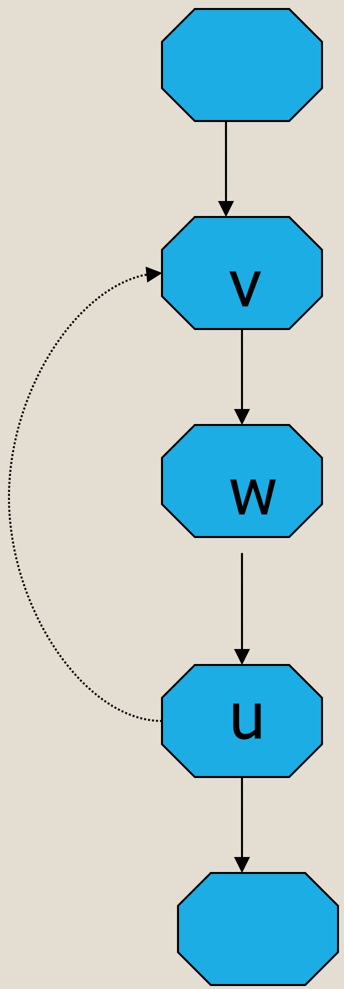
irányított aciklikus gráf



dfs erdő
(a csúcsok sorrendjében
történő kereséssel)

Egyik topológiai rendezés:
d,e,c,f,b,a

A visszamutató ív ciklust jelent



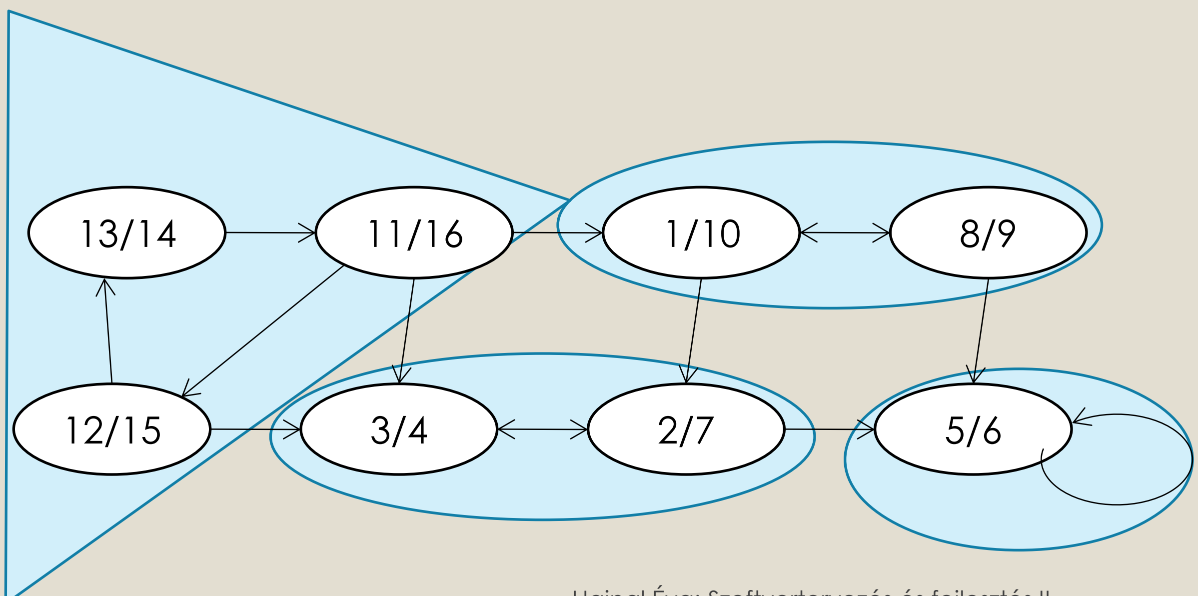
Tegyük fel, hogy **n** a csúcsok **m** pedig az élek száma, és **n** $\leq$ **m**
Ekkor **O(m)** lépésben megvalósíthatjuk a gráf postorder bejárását. Ha v $\rightarrow$ w előre mutató él, **w** p.o. száma < **v** p.o. száma. Ha u$\rightarrow$ v visszamutató él (**u** p.o. száma $\leq$ **v** p.o.) száma, akkor ez ciklust jelez.
Fordítva, egy ciklus élei között kell, hogy legyen visszamutató él.
Tf. ui. hogy $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$
v_1 postorder száma p_i, i=1,2,...
Ha k=1, $v_1 \rightarrow v_1$ visszamutató él.
Tf. k>1 és a fenti láncban v_1 és v_k között nincs visszamutató él. Ekkor $p_1 > p_2 > \ldots > p_k$ azaz $p_1 > p_k$ tehát $v_k \rightarrow v_1$ visszamutató él.

Irányított gráf aciklikus voltának tesztelése

```
BBOLEAN testAcyclic(GRAPH G){
NODE u, v;
LIST p;
dfs Forest (G);
   for (u=0; u < MAX; u++){
     p = G[u].successors;
     while (p != NULL) {
          v = p->nodeName;
          if (G[u].postorder <= G[v].postorder)
               return FALSE;
          p=p->next;
     }
   }
   RETURN true;
}
```

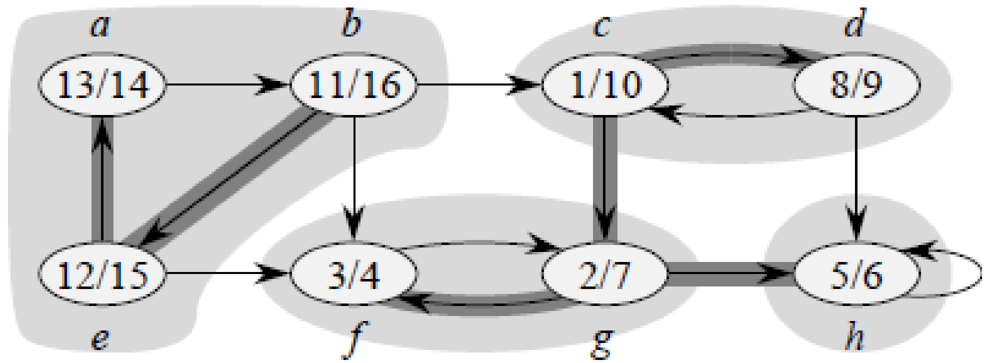
Irányított gráfok összefüggő komponensei

- Minden irányított (irányítatlan) gráf összefüggő komponensekre bontható
- Összefüggő komponens: ha bármely két csúcsa között van út
- A fenti öf. komponensek maximálisak, azaz bármely csúcsot hozzávéve már nem összefüggőek
- Összefüggő a gráf, ha egyetlen öf. komponensből áll

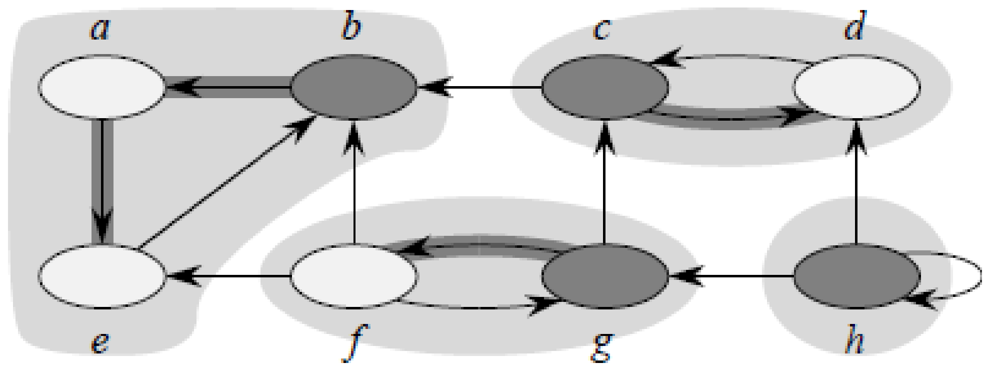


Erősen összefüggő komponensek

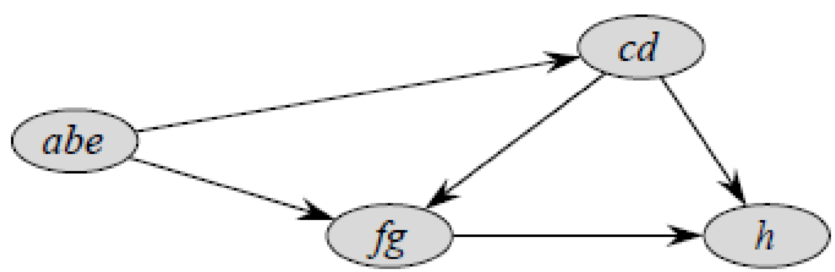
(a)



(b)



(c)



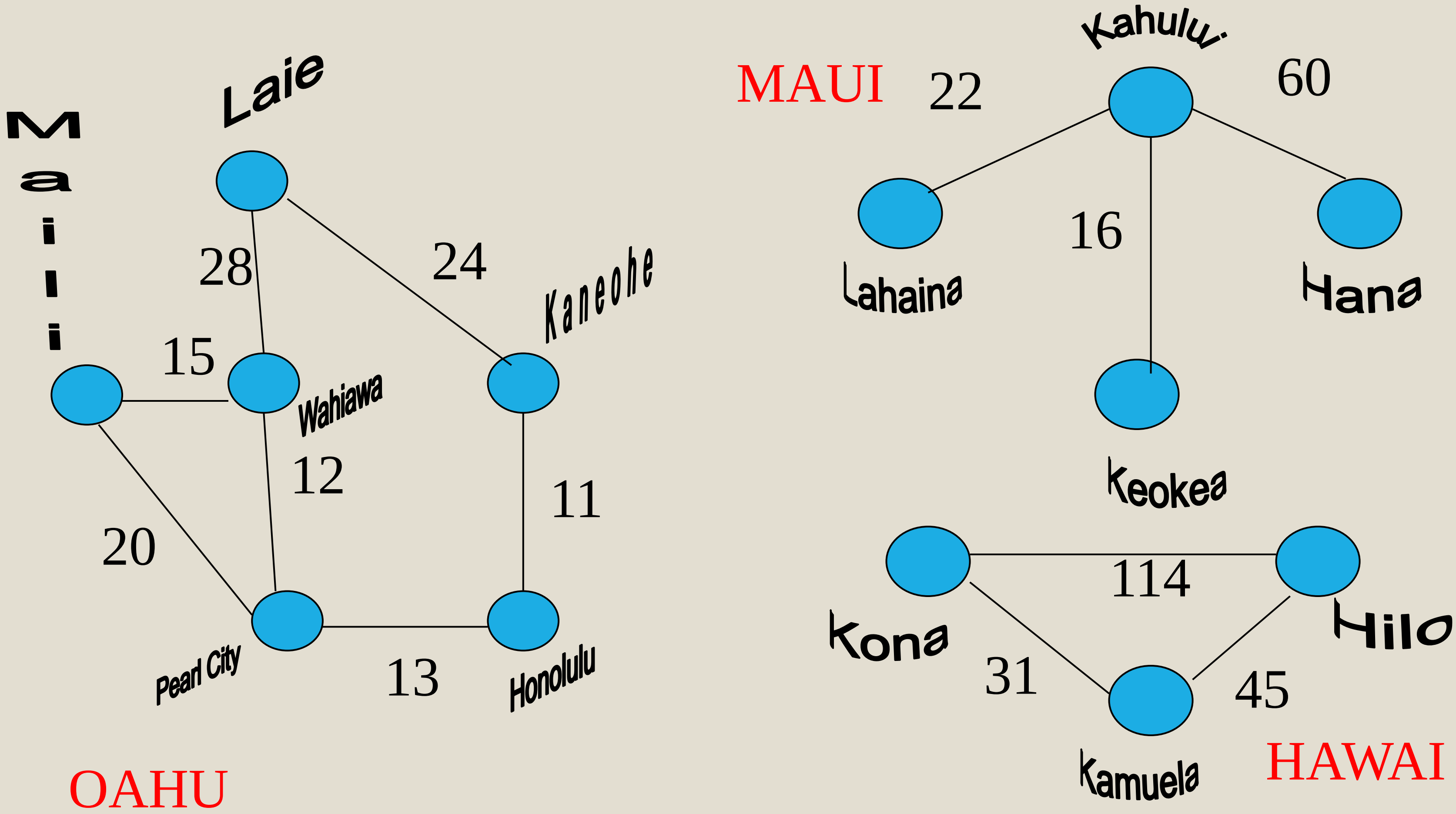
Algoritmus

- 1 MK(G) hívása; minden u csúcsra kiszámítjuk az $f[u]$ elhagyási időpontot
- 2 GT meghatározása
- 3 MK(GT) hívása, de MK fő ciklusában a csúcsokat $f[u]$ szerint csökkenő
- sorrendben vizsgáljuk (ahogy az első sorban kiszámítjuk)
- 4 a 3. lépésben kapott mélységi erdő egyes fáinak csúcsait írjuk ki, mint
- erősen összefüggő komponenseket

Összefüggő komponensek megkeresése

1. Legyen G egy irányítatlan gráf, jelölje G_0 azokat a csúcsokat, amelyekből nem vezet ki él. Ekkor G_0 minden pontja egy-egy komponens. (G_0 persze üreshalmaz is lehet.)
2. Tegyük fel, hogy i db élt megvizsgáltunk, és az ezeket tartalmazó G_i gráf összefüggő komponenseit megtaláltuk. (indukciós feltevés) Vegyünk egy újabb {u,v} élt. Ha ezek G_i ugyanazon komponensébe tartoznak, G_{i+1} komponensei megegyeznek G_i komponenseivel.
3. Ha sem u, sem v nincs G_i –ben, {u,v} egy újabb komponens.
4. Ha u G_i egyik komponensében van, és v nincs abban, akkor u komponenséhez csatlakoztatjuk v-t és a v-hez kapcsolódó G_i –beli összefüggő komponenst.

Kövessük nyomon az előbbi algoritmust a Hawai szigetek példáján! Az élek sorrendjét a súlyok nagyságrendje szerint vesszük.



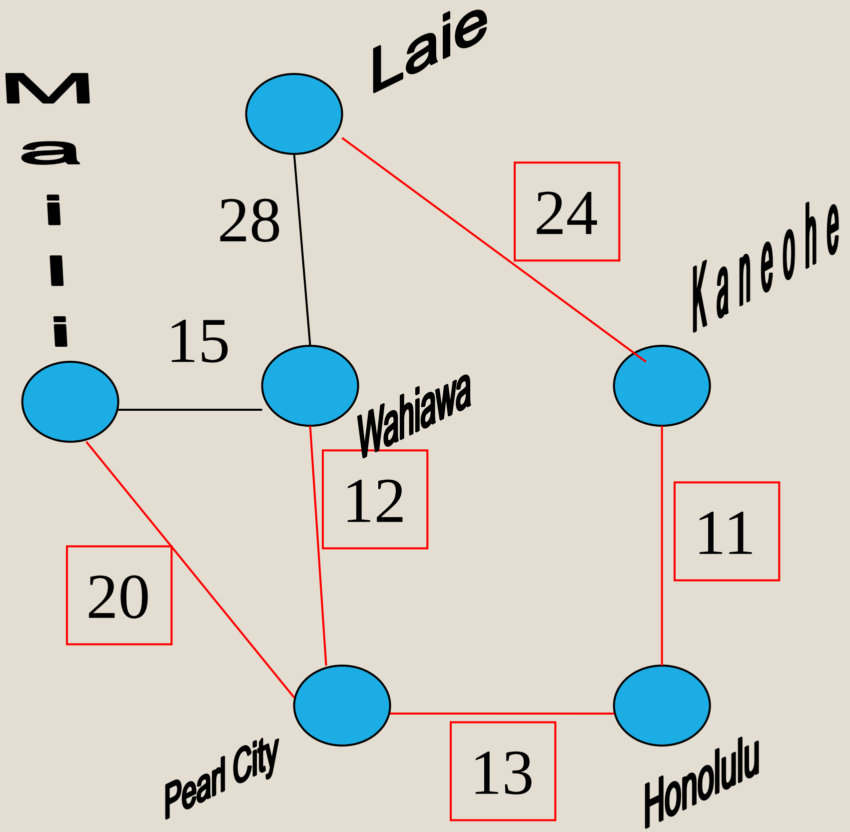
Minimális feszítőfa keresése

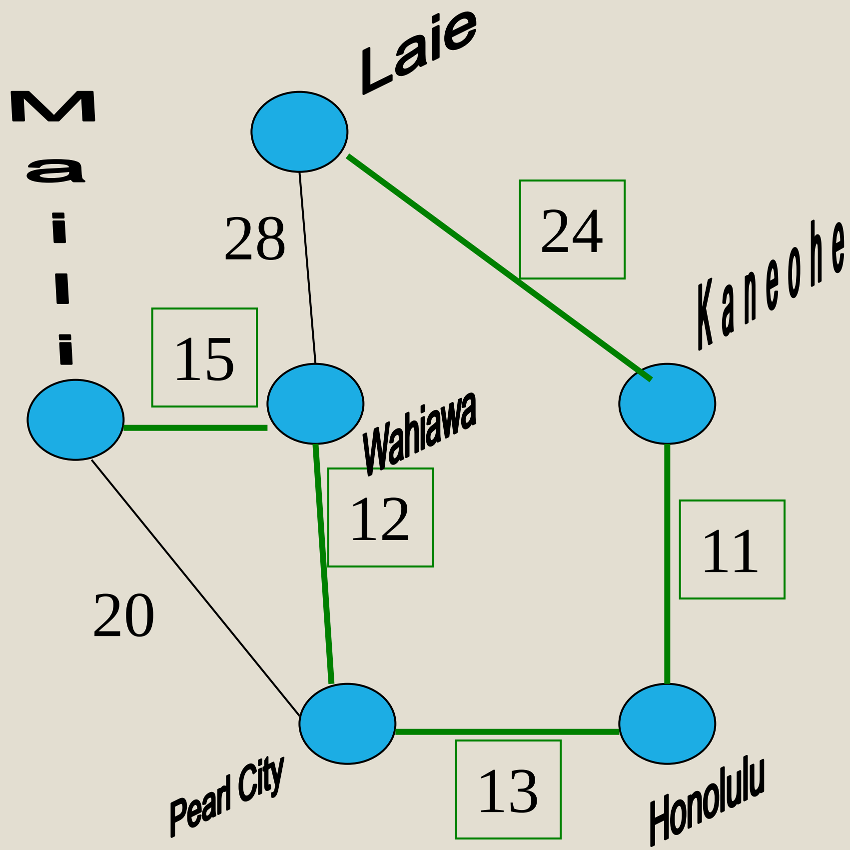
Irányítatlan súlyozott gráfban nemcsak az öf. komponenseket akarjuk megkeresni, hanem a komponens legyen fa (itt: ciklusmentes részgráf, gyökérrel, levéllel, gyerekkel nem foglalkozunk)

Feszítőfa: csúcsai azonosak a komponens csúcsaival, élei a komponens éleinek részhalmazát alkotják, ciklust nem tartalmaz

Ráadásul a feszítőfa legyen minimális abban az értelemben, hogy a súlyösszege minimális legyen (a lehetséges feszítőfák közül)

Ugyanazon gráf két különböző feszítőfája

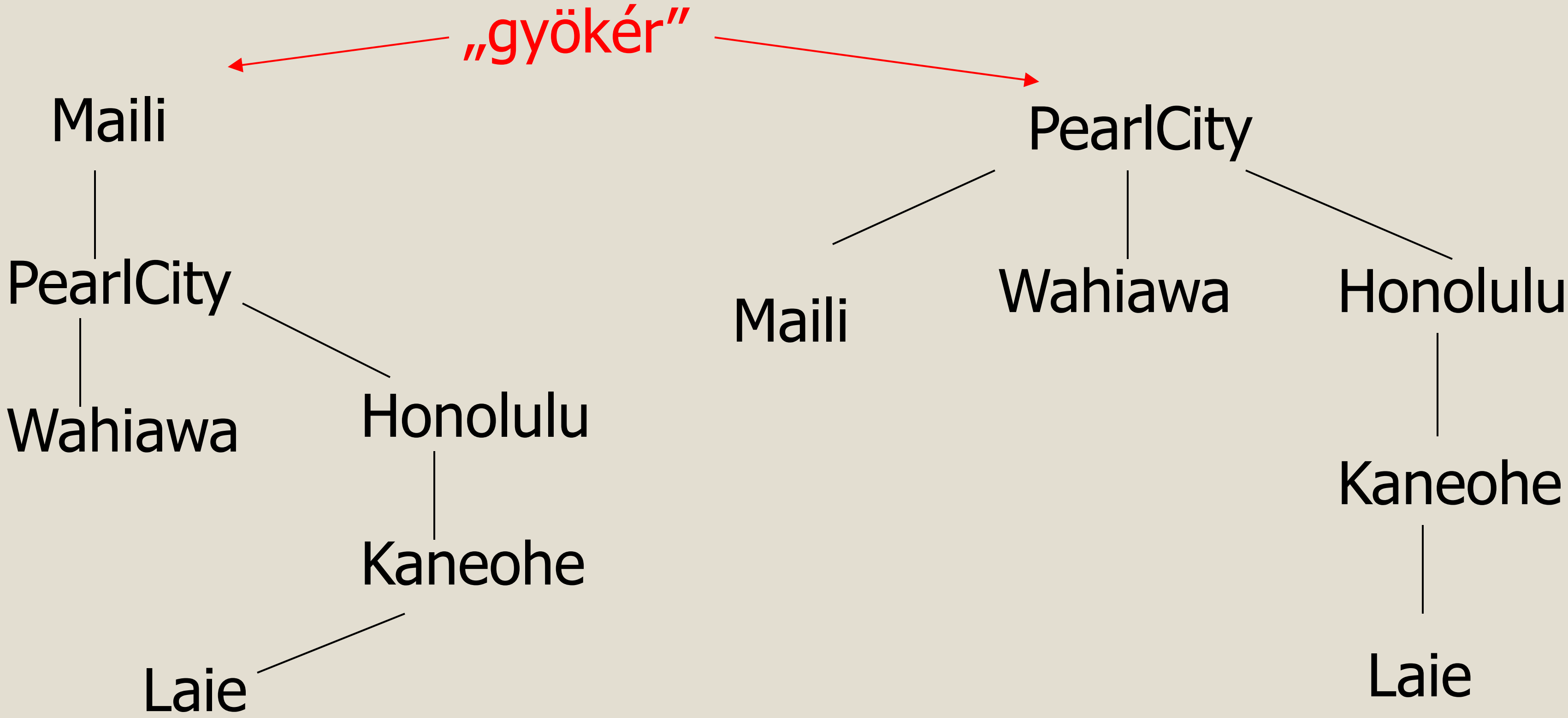




Piros feszítőfa,
összúlya 80

Zöld feszítőfa,
összsúlya 75

A fa gyökértelen, a gyerekek sorrendje tetszőleges



Ha nem kitüntetett a gyökér, a két gráf ugyanaz, mindkettő a piros fa!

Kruskal algoritmus a minimális feszítőfa megkeresésére

- Az éleket sorbarendezzük a súlyok szerint
- ha egy él két csúcsa különböző komponensbe tartozik, kiválasztjuk a feszítőfába és egyesítjük a két részkomponenst, egyébként nem választjuk ki az élt és nem egyesítünk
- Eredmény OAHU szigetén a zöld feszítőfa

Miért működik a Kruskal algoritmus?

- Legyen G egy összefüggő, irányítatlan súlyozott gráf.
- Rendezzük az éleket súly szerint növekvő sorrendbe. Ha van két azonos súly, az egyikhez adjunk hozzá egy kis ε értéket, hogy a két él különböző súlyú legyen, de a sorrend csak közöttük változzon. Esetleg többször is alkalmazzuk e trükköt úgy, hogy minden él különböző legyen, de $\sum\varepsilon$ még mindig kisebb, mint bármely két „régi" él különbsége.
- Így a generált feszítőfa egyértelmű lesz.
- A Kruskal algoritmus mohó algoritmus (greedy algorithm) abban az értelemben, hogy minden pillanatban az akkor legjobb lépést tesszük meg. Ez nem mindig vezet globális optimumhoz, de a Kruskal algoritmus esetében igen. (Nem bizonyítjuk.)

Köszönöm a figyelmet!