

Nemrekurzív preorder bejárás veremmel

Ismét feltesszük, hogy a fa a g gyökérpontja által van megadva elsőfiú testvér reprezentációval, és az M műveletet akarjuk minden ponton végrehajtani.

```
PreorderV(g,M)          // Preorder bejárás veremmel.
if (g=nil) then return //Üres fa
Letesit (V:Verem)
Verembe (V, g)
while (NemUres (V))
  {VeremBol (V, p)
  M(p)
  if p.Testver!=nil then Verembe (V, p.Testver)
  if p.Elsofiu!=nil then Verembe (V, p.Elsofiu) }
```

Példa: $V=[1], p=1, V=[], M(1), V=[2], p=2, V=[], M(2), V=[8], V=[8,3], p=3, V=[8], M(3), V=[8,4], p=4, V=[8], M(4), V=[8,5], p=5, V=[8], M(5), V=[8,6], p=6, V=[8], M(6), V=[8,7], p=7, V=[8], M(7), p=8, V=[], M(8), V=[9], p=9, V=[], M(9)$

Helyesség

Tekintsük a while ciklus végrehajtásának egy adott pillanatát. Legyen $B = \{p_1, \dots, p_k\}$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a veremből kikerültek. Legyen $V = \{q_1, \dots, q_m\}$ a V verem tartalma, ezek az aktív pontok. A következő négy állítás teljesül.

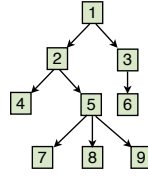
- Az B sorozatban a pontok helyes preorder sorrendben vannak.
- A preorder bejárásban p_k -t közvetlenül q_m követi.
- A preorder sorrendben q_i megelőzi q_{i-1} -et ($i = 2, \dots, m$).
- $B \cap V = \emptyset$ és a fa bármely $p \notin B$ pontjára, pontosan egy olyan $q \in V$ pont van, hogy p leszarmazottja q -nak az elsőfiú-testvér fában.

Szintszerinti bejárás

Egy F fa bármely két p és q pontjára p akkor és csak akkor előzi meg q -t a szintszerinti bejárásban, ha $d(p) < d(q)$ vagy $d(p) = d(q)$ és p balrább esik, mint q .

Az algoritmus

```
SzintBejar (g,M)
if (g=nil) then return //Üres fa
Letesit (S: Sor)
SorBa (S, g)
while (Elemszam (S) !=0)
  {SorBol (S, p)
  M(p)
  p=p.Elsofiu;
  while (p!=nil) //p gyerekeit berakjuk S-be
    {SorBa (S, p)
    p=p.Testver}}
```



1. ábra.

Példa: $S=[1], p=1, S=[], M(1), p=2, S=[2], p=3, S=[2,3], p=Nil, p=2, S=[3], M(2), p=4, S=[3,4], p=5, S=[3,4,5], p=Nil, p=3, S=[6,7], p=8, s=[6,7,8], p=9, S=[6,7,8,9], p=Nil, p=6, S=[7,8,9], M(6), p=Nil, p=7, S=[8,9], M(7), p=Nil, p=8, S=[9], M(8), p=Nil, p=9, S=[], M(9), p=nil.$

Helyesség

Tekintsük a külső while ciklus végrehajtásának egy adott pillanatát. Legyen $B = \{p_1, \dots, p_k\}$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a sorból kikerültek. Legyen $S = \{q_1, \dots, q_m\}$ a S sor aktuális tartalma, ezek az aktív pontok. A következő négy állítás teljesül.

- Az B sorozatban a pontok szintszerinti sorrendben vannak.
- Az S sorban a pontok szintszerinti sorrendben vannak.
- $d(q_m) \leq d(q_1) + 1$
- A szintszerinti bejárásban p_k megelőzi q_1 -et.

Bejárás Adagolóval

Ha nem fontos a szintszerinti bejárési sorrend, a fenti algoritmusban az aktív pontok tárolására minden olyan absztrakt adattípus használható lenne, amely rendelkezik az alábbi műveletekkel.

Értékhalmoz: E Adagol = $A \subseteq E$ Műveletek: A : Adagoló, $x : E$

- $\{Igaz\}$ Letesit(A) $\{A = \emptyset\}$
- $\{A = A\}$ Megszuntet(A) $\{Igaz\}$
- $\{A = A\}$ Uresit(A) $\{A = \emptyset\}$
- $\{S = [a_1, \dots, a_n]\}$ Betesz(A, x) $\{A = Pre(A) \cup \{x\}\}$
- $\{A \neq \emptyset\}$ Kivesz(A, x) $\{x \in Pre(A) \wedge Pre(A) = A \cup \{x\}\}$
- $\{A = A\}$ Elemszam(A) $\{Elemszam = |A|\}$

Mohó algoritmusok

Optimalizálási probléma megoldására szolgáló algoritmus sokszor olyan lépések sorozatából áll, ahol minden lépésben adott halmazból választhatunk. Ezt gyakran dinamikus programozás alapján oldjuk meg, de előfordul, hogy a dinamikus programozás túl sok esetet vizsgál annak érdekében, hogy az optimális választást meghatározza.

A mohó algoritmus mindig az adott lépésben optimálisnak látszó választást teszi. Vagyis, a lokális optimumot választja abban a reményben, hogy ez globális optimumhoz fog majd vezetni.

Mohó algoritmus nem mindig ad optimális megoldást, azonban sok probléma megoldható mohó algoritmussal.

Eseménykiválasztási probléma

Tegyük fel, hogy adott események egy $S = \{a_1, a_2, \dots, a_n\}$ n elemű halmaza, amelyek egy közös erőforrást, például egy előadótermet kívánnak használni. A teremben egy időben csak egy esemény folyhat. Az a_i esemény az s_i kezdő időpont és az f_i befejezési időpont által adott, ahol $s_i < f_i$. A cél egy maximális halmazát kiválasztani kompatibilis eseményeknek (i,j kompatibilis, ha $s_i \geq f_j$ vagy $s_j \geq f_i$).

Feltesszük, hogy az S eseményhalmaz elemeit a befejezési idejük szerint nemcsökkenő sorrendbe rendeztünk.

Példa:

1. táblázat. Az eseménykiválasztási probléma egy inputja

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Mohó algoritmus

MOHÓ-ESEMÉNY-KIVÁLASZTÓ(s, f)

```
n:=hossz[s]
Letesit(A:Halmaz)
Bovit(A,1)
i:=1
for m=2 to n
  {if s[m]>=f[i]
   then {Bovit(A,m)
        i:=m}}
return A
```

Példa: $A = \{1, 4, 8, 11\}$

A helyesség igazolása

1. Lemma Létezik olyan optimális megoldás, amely az 1 eseménnyel kezdődik.

Bizonyítás Legyen A egy tetszőleges optimális megoldás, legyen k a legkisebb indexű esemény. Ha $k = 1$, akkor az állítás nyilvánvaló, egyébként legyen A' az a halmaz, amit A -ból kapunk úgy, hogy k -t kicseréljük 1-re. A Lemma teljesül, mert A' is optimális.

2. Lemma Ha A tartalmazza 1-et és A optimális megoldása az S problémának, akkor $A \setminus \{1\}$ optimális megoldása az S' problémának, ahol $S' = \{i \in S : s_i \geq f_1\}$

Bizonyítás Ha nem lenne optimális megoldása S' -nek, akkor egy jobb megoldást kiegészítve 1-el, A -nál jobb megoldását kapnánk az S problémának.

A mohó megoldó stratégia elemei

1. Fogalmazzuk meg az optimalizációs feladatot úgy, hogy minden egyes választás hatására egy megoldandó részprobléma keletkezzék.

2. Bizonyítsuk be, hogy mindig van olyan optimális megoldása az eredeti problémának, amely tartalmazza a mohó választást, tehát a mohó választás mindig biztonságos.

3. Mutassuk meg, hogy a mohó választással olyan részprobléma keletkezik, amelynek egy optimális megoldásához hozzávéve a mohó választást, az eredeti probléma egy optimális megoldását kapjuk.

Kapcsolat a dinamikus programozással

A feladat megoldható lenne dinamikus programozással is.

Legyen $S_{i,j} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$, tehát $S_{i,j}$ azokat az S -beli eseményeket tartalmazza, amelyek a_i befejeződése után kezdődnek, és befejeződnek a_j kezdete előtt, azaz kompatibilisek mind a_i -vel mind pedig a_j -vel.

Tegyük fel, hogy $A_{i,j}$ egy optimális megoldása az $S_{i,j}$ részproblémának és $a_k \in A_{i,j}$. Ekkor az $A_{i,k}$ megoldás optimális megoldása kell legyen az $S_{i,k}$ részproblémának, és az $A_{k,j}$ megoldás optimális megoldása kell legyen az $S_{k,j}$ részproblémának.

Legyen $c[i, j]$ az $S_{i,j}$ részprobléma optimális megoldásában az események száma, ekkor

$c[i, j] = 0$, ha $S_{i,j} = \emptyset$ és

$c[i, j] = \max_{i < k < j, a_k \in S_{i,j}} \{c[i, k] + c[k, j] + 1\}$ egyébként.

A rekurzióknak a mohó megoldás mindig optimális megoldását adja.

Hátizsák feladat

Egy adott hátizsákba tárgyakat akarunk pakolni. Adott n tárgy minden tárgynak van egy fontossági értéke ($f[i]$), és egy súlya ($s[i]$), a hátizsákba maximum összesen S súlyt pakolhatunk. Az $s[i]$ és S értékek egészek. Szeretnénk úgy választani tárgyakat, hogy az összfontosság maximális legyen. Tehát feladatunk, hogy kiválasszuk a tárgyaknak olyan halmazai közül, amelyekre az összsúly nem haladja meg S -t azt, amelyre maximális az összfontosság.

A töredékes változat abban különbözik az előzőtől, hogy a tárgyak töredéke is választható, nem kell 0-1 bináris választást tenni.

Megoldó algoritmusok

A töredékes hátizsák feladatot optimálisan megoldja egy mohó algoritmus. A feladat megoldásához számítsuk ki minden tárgyra a $f[i]/s[i]$ fontosság per súly hányadosát. A mohó stratégia szerint mindig a legnagyobb hányadosú tárgyból választunk amennyit csak lehet. Ha elfogyott, de még nem telt meg a hátizsák, akkor a következő legnagyobb hányadosú tárgyból választunk amennyit csak lehet, és így tovább, amíg a hátizsák meg nem telik. Mivel a tárgyakat az érték per súly hányados szerint kell rendeznie, a mohó algoritmus futási ideje $O(n \log n)$ lesz.

Ez a mohó algoritmus nem ad feltétlenül optimális megoldást a 0-1 hátizsák feladatra. Ezt igazolja a következő példa: $s = [10, 20, 30]$, $f = [60, 100, 120]$, $S = 50$.

Huffman kód

A Huffman-kód széles körben használt és nagyon hatékony módszer adatállományok tömörítésére. Az elérhető megtakarítás 20%-tól 90%-ig terjedhet, a tömörítendő adatállomány sajátosságainak függvényében. A kódolandó adatállományt karaktersorozatnak tekintjük. A Huffman féle mohó algoritmus egy táblázatot használ az egyes karakterek előfordulási gyakoriságára, hogy meghatározza, hogyan lehet a karaktereket optimálisan ábrázolni bináris jelsorozattal.

Kódolási módszerek

2. táblázat. Karakterek kódolása

karakter gyakoriság	a	b	c	d	e	f
fix hosszú kód	000	001	010	011	100	101
változó hosszú kód	0	101	100	111	1101	1100

Kézenfekvő megoldás fix hosszú kódokat használni, ekkor a betűk kódolásához 3 bit kell, így a teljes kód hossza: 300.

A változó hosszú kód alkalmazása tekintélyes megtakarítást eredményez, ha gyakori karaktereknek rövid, ritkán előforduló karaktereknek hosszabb kódszavakat feleltetünk meg. A fenti példában szereplő kód esetén a kódoláshoz szükséges bitek száma: $(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) = 224$.

A kódot vissza is kell tudnunk fejteni. A továbbiakban csak olyan kódszavakat tekintünk, amelyekre igaz, hogy egyik sem kezdőszelete a másoknak. Az ilyen kódolást prefix-kódnak vagy prefix mentes kódnak nevezzük.

Kódok ábrázolása

Az olyan bináris fa, amelynek levelei a kódolandó karakterek, egy alkalmas ábrázolása a prefixmentes kódoknak. Ekkor egy karakter kódját a fa gyökerétől az adott karakterig vezető út ábrázolja, a 0 azt jelenti, hogy balra megyünk, az 1 pedig, hogy jobbra megyünk az úton a fában.

Ekkor minden karakter kódjának hossza a fában a megfelelő levél mélysége, így a teljes állomány kódolásának várható hossza egy adott T fára:

$$B_T = \sum_{c \in C} f(c) d_T(c),$$

ahol C a karakterek halmaza, $f(c)$ a c karakter gyakorisága, $d_T(c)$ pedig a c -nek megfelelő levél mélysége a T fában.

Huffman kód algoritmus

A kód fájának megszerkesztéséhez egy prioritási sort használunk, amely a fa pontjait az f érték alapján tárolja. Eredményképpen a fa gyökerét adja az algoritmus vissza.

```

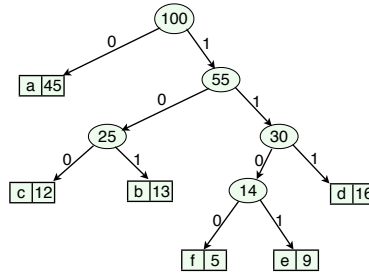
HUFFMAN(C)
n:=|C|
Letesit(Q:PrisOr)
for (c in C)
    Sorba(Q, c)
for i=1 to n-1
    {új z csúcs létesítése
    Sorbol(Q, x)
    bal[z]:=x
    Sorbol(Q, y)
    jobb[z]:=y
    f[z]:=f[x]+ f[y]
    Sorba(Q, z) }
Sorbol(Q, x)
Return x

```

Az algoritmus futási ideje $O(n \log n)$.

Megjegyzés: A fenti pszeudókód által kapott algoritmus nem alkalmas arra, hogy az egyes pontok kódját hatékonyan megkapjuk belőle. Egy hatékony megvalósítás során, a fa konstruálásánál, az adott pont fiainál meg kell jegyeznünk, hogy bal vagy jobb fiúk.

Példa:



2. ábra.

Helyesség

1. Lemma Legyen C tetszőleges karakter halmaz, és legyen $f[c]$ a $c \in C$ karakter gyakorisága. Legyen x és y a két legkisebb gyakoriságú karakter C -ben. Ekkor létezik olyan optimális prefix-kód, amely esetén az x -hez és y -hoz tartozó kódszó hossza megegyezik, és a két kódszó csak az utolsó bitben különbözik.

Bizonyítás: A bizonyítás alapötlete az, hogy vegyünk egy optimális prefix-kódot ábrázoló T fát és módosítsuk úgy, hogy a fában x és y a két legmélyebben lévő testvércsúcs legyen. Ha ezt meg tudjuk tenni, akkor a hozzájuk tartozó kódszavak valóban azonos hosszúságúak lesznek és csak az utolsó bitben különböznek.

Legyen a és b a T fában a két legmélyebb testvércsúcs. Az általánosság megszorítása nélkül feltehetjük, hogy $f[a] \leq f[b]$ és $f[x] \leq f[y]$. Mivel $f[x] \leq f[y]$ a két legkisebb gyakoriság, ezért azt kapjuk, hogy $f[x] \leq f[a]$ és $f[y] \leq f[b]$.

Cseréljük ki az a és x pontokat T -ben, legyen a kapott fa T' . Majd cseréljük ki a b és y pontokat T' -ben a kapott fa legyen T'' .

Ekkor kiszámolva a költségeket:

$$B(T) - B(T') = (f[a] - f[x])(d_T(a) - d_T(x)) \geq 0$$

$$B(T') - B(T'') = (f[b] - f[y])(d_{T'}(b) - d_{T'}(y)) \geq 0.$$

Következésképpen T'' is optimális.

A konstrukció teljesíti az optimális részproblémák tulajdonságot.

2. Lemma Legyen C tetszőleges ábécé, és minden $c \in C$ karakter gyakorisága $f[c]$. Legyen x és y a két legkisebb gyakoriságú karakter C -ben. Tekintsük azt a C' ábécét, amelyet C -ből úgy kapunk, hogy eltávolítjuk az x és y karaktert, majd hozzáadunk egy új z karaktert, tehát $C' = C \setminus \{x, y\} \cup \{z\}$. Az f gyakoriságok C' -re megegyeznek a C -beli gyakoriságokkal, kivéve z esetét, amelyre $f[z] = f[x] + f[y]$. Legyen T' olyan fa, amely optimális prefix-kódját ábrázolja a C' ábécének. Ekkor az a T fa, amelyet úgy kapunk, hogy a z levélcsúcsához hozzákapcsoljuk gyerek csúcsként x -et és y -t, olyan fa lesz, amely a C ábécé optimális prefix-kódját ábrázolja.

Bizonyítás: A formulák behelyettesítésével adódik, hogy $B(T) = B(T') + f[x] + f[y]$.

Ezt követően indirekt bizonyítunk. Tegyük fel, hogy T nem optimális prefix-kódfa a C ábécére. Ekkor létezik olyan T'' kódfa C -re, hogy $B(T'') < B(T)$. Az általánosság megszorítása nélkül (az 1. lemma alapján) feltehetjük, hogy x és y testvérek. Legyen T^* az a fa, amelyet T'' -ből úgy kapunk, hogy eltávolítjuk az x és y csúcsokat, és ezek közös z szülőjének gyakorisága az $f[z] = f[x] + f[y]$ érték lesz.

Ekkor $B(T^*) = B(T'') - f[x] - f[y] < B(T) - f[x] - f[y] = B(T')$, ami ellentmondás.

Egy vágási feladat

Adott egy fémrúd, amelyet megadott számú és hosszúságú darabokra kell felválni. A vágások tetszőleges sorrendben elvégezhetőek, az a lényeg, hogy a végén a kívánt darabszámú és hosszúságú darabok keletkezzenek. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. A célunk optimalizálni a műveletsor teljes költségét.

Adjunk meg olyan algoritmust, amely kiszámítja a vágási műveletsor optimális összköltségét. Az algoritmus futási ideje $O(n * \log n)$ legyen, ha n a darabok száma!

Megoldás

- Ábrázoljuk a vágások sorozatát egy fában, a levelek a végső darabok és minden vágás egy belső pontnak felel meg.
- Ekkor minden darab annyi vágásban vesz részt, amilyen mélyen a fában van.
- Tehát egy olyan fát kell építeni, amelyben $B_T = \sum_{c \in C} l(c) d_T(c)$ minimális ahol C a darabok halmaza, $l(c)$ a c darab hossza, $d_T(c)$ pedig a c -nek megfelelő levél mélysége a T fában.
- A feladat ekvivalens a Huffman kód feladatával.

Egy ütemezési feladat

Feladat Vegyük azt az ütemezési modellt, ahol adottak az $\{1, \dots, n\}$ munkák továbbá a j munkának van egy p_j végrehajtási ideje és egy w_j súlya. A cél a munkáknak egy olyan sorrendjét megadni, amely sorrendre a befejezési idők súlyozott összege minimális.

Példa ($p_1 = 2, w_1 = 2$), ($p_2 = 4, w_2 = 3$), ($p_3 = 2, w_3 = 1$).

Ekkor az 1, 2, 3 sorrendre a befejezési idők 2, 6, 8 és a célfüggvény $2 \cdot 2 + 6 \cdot 3 + 8 \cdot 1 = 30$.

A 2, 1, 3 sorrendre a befejezési idők 4, 6, 8 és a célfüggvény $4 \cdot 3 + 6 \cdot 2 + 8 \cdot 1 = 32$.

Mohó megoldások:

- Elsőnek azt hajtsuk végre, akinek kicsi a végrehajtási ideje.
- Elsőnek azt hajtsuk végre, akinek nagy a súlya.

Egyik sem ad optimális algoritmust.

Tétel A munkákat a p_j/w_j érték szerint monoton növekvő sorrendbe rendezve egy optimális sorrendet kapunk.

Lemma Van olyan optimális megoldás, amelyben az elsőnek végrehajtott munkára minimális a p_j/w_j érték.

Bizonyítás: Vegyük azt az optimális megoldást, amelyben a legkorábban szerepel olyan munka, amelyre a p_j/w_j érték minimális. Legyen ez a megoldás OPT és tegyük fel, hogy az i -edik munka az első olyan, amire a p_j/w_j érték minimális. Ha $i = 1$, akkor a lemma teljesül.

Ellenkező esetben legyen k és r OPT $i-1$ -edik és i -edik munkája. Vegyük azt az OPT' megoldást, amelyet úgy kapunk OPT-ből, hogy felcseréljük k és r sorrendjét. Ekkor OPT célfüggvényértékéből kivonva OPT' célfüggvényértékét a

$$w_k p_k + w_r (p_k + p_r) - w_r p_r - w_k (p_r + p_k) = w_r p_k - w_k p_r > 0$$

értéket kapjuk, ami ellentmond OPT optimalitásának.

Lemma Egy optimális megoldás tetszőleges végszelete optimális megoldása annak az ütemezési feladatnak, amelyben az input a végszeletben szereplő munkák halmaza.

Bizonyítás Ha nem lenne optimális megoldás, akkor egy jobb megoldást kiegészítve a hiányzó kezdőszelettel, a kiinduló megoldásnál jobb megoldását kapnánk az eredeti problémának.

Egy másik ütemezési feladat

Feladat Vegyük azt az ütemezési problémát, ahol egy gép van, minden munkának van egy végrehajtási ideje és egy határideje. A cél a maximális késedelem (befejezési időből kivont határidő) minimalizálása.

Mohó algoritmus: a munkákat határidő szerint rendezzük és ebben a sorrendben hajtjuk végre.

Tétel A Mohó algoritmus optimális megoldást ad.

Lemma Van olyan optimális megoldás, amelyben az elsőnek végrehajtott munkának a legkisebb a határideje.

Bizonyítás: Tekintsünk egy olyan ütemezést legyen S , amely nem egy legkisebb határidővel rendelkező munkát ütemez elsőnek. Legyen a legkisebb határidővel rendelkező munka j .

Tegyük fel, hogy S valamilyen i_1, i_2, \dots, i_k, j munkasorrenddel kezdődik. Vegyük S' -t, amelyet úgy kapunk S -ből, hogy a munkák ezen kezdeti sorrendjét kicseréljük a j, i_1, i_2, \dots, i_k sorrendre. Az új ütemezésre az i_1, \dots, i_k munkák később fejeződnek be, viszont a befejezési idejük nem lesz nagyobb, mint j -nek az S ütemezésben, ezért a késedelmük sem, hiszen j határideje minimális volt. Következésképpen az így kapott megoldás is optimális lesz, amivel a lemmát igazoltuk.

Huzalozásos feladat

Adottak fekete és fehér korongjaink egy egyenesen mindkettőből n darab. A feladat az, hogy a lehető legrövidebb huzalmennyiséggel kössünk össze n darab fekete-fehér párt.

Mohó algoritmus: balról jobbra vegyük az első fekete-fehér párt, töröljük őket az inputból majd folytassuk ugyanígy.

Lemma Van olyan optimális megoldás, ami tartalmazza balról jobbra nézve az első fekete fehér párt.

Lemma A törlés után kapott feladat optimális megoldásához hozzávéve a törölt párt a telejs feladat optimális megoldását kapjuk.

Kiskérdések a ZH utáni hétre

- Szintszerinti bejárás
- Eseményválasztás algoritmusa
- Huffman kód algoritmusa
- Huffman kód algoritmusának végrehajtása, az optimális fa felépítése, a kódok megadása egy megadott példán

Szorgalmi feladat

Adott az egyenesen pontoknak egy $x_1 \leq x_2 \leq \dots \leq x_n$ egy halmaza. A feladat az, hogy fedjük le a pontokat körökkel úgy, hogy a $\sum(1 + d_i)$ érték minimális legyen, ahol d_i az i -edik kör átmérője. Adjunk egy lineáris idejű algoritmust, amely megad egy optimális lefedést.

Beküldés: cimreh@inf.u-szeged.hu,

Pszedókode vagy forrás +helyesség igazolás + futási idő elemzés

- első négy megoldó 8-8 pont
- ötödiktől-nyolcadik megoldó 4-4 pont