Mohó algoritmusok módszertana

*Horváth Gyula * Szlávi Péter*
ELTE IK

2003

Vázlat:

0 „A mohóság dicsérete” – bevezetés

Gyakori kérdésfelvetés: „melyik a legjobb választás valamire?”, „hogyan kell elosztani valamit, hogy a legkevesebb vagy éppen a legtöbb legyen valamilyen jellemző?” stb. Ezek az ún. *optimalizációs problémák*.

Ilyen felvetésekre a szokásos *kereső módszerek* mindig megoldást jelentenek, de általában irdatlan mennyiségű eset megvizsgálását igénylik. Azok a bizonyos „haladóbb trükkök”, mint pl. a backtrack vagy a dinamikus programozás sem mindig tudnak segíteni.

Vannak viszont olyan helyzetek, amelyben létezik meglepően egyszerű és hatékony megoldás is. Az ötlet röviden megfogalmazható: válasszuk mindig az adott lépésben a legjobbnak tetszőt. Képszerűbben fogalmazva: a pillanat „csábításának” engedve, nem mérlegelve a távolabbi jövendőt döntünk az éppen legkedvezőbb mellett. „Carpe diem!” – mondta Horatius[1] a krisztusi időkben.

1. ábra. Hawaii nemes mohó (Moho nobilis)[3]

E felfogás időről időre reneszánszát élte s éli azóta is. Ugyanez az *életérzés* sugárzik a 11-13. századi Carmina Burana szövegéből és dallamából, de a jól ismert 20. századi Orff-i feldolgozásából is.

A mohóság, a hét főbűn egyike, számos gaztett indítórúgója nemcsak a történelmi relikviákban, hanem a tudományban is otthagyta nyomait. Példaként azokra a madarakra utalok, amelyek egy-egy nemtörődöm betelepülőnek, vagy mégcsak le sem telepedő, pénzsóvár rablónak köszönhetően tűntek el a Föld színéről. A közismert dodó mellett a rendszertan számos ilyent tart nyilván. Az érdekes latin (?) nevű „Moho nobilis” képét a mellékelt ábrán láthatjuk. Hogy véletlen-e a névválasztás: nem tudom, de elgondolkodtató, hogy az említettnek vannak hasonló sorsra jutott „névrokonai” is: Molokai mohó (Moho bishopi) és Oahu-mohó (Moho apicalis).[2]

2. ábra. A 11-13. sz.-i *Carmina Burana* kézirat egy illusztrációja[4]

3. ábra. Metallica a *Master Of Puppets* albumon szereplő logója[5]

[1] http://en.wikipedia.org/wiki/Horace
[2] http://www.foek.hu/dodo/folap.htm#mez
[3] http://www.foek.hu/dodo/madar/mono.htm
[4] http://en.wikipedia.org/wiki/Carmina_Burana
[5] http://en.wikipedia.org/wiki/Metallica

A mohóság körüli kalandozásomat befejezendő, utalok a napjaink globalizált világát átszövő zenei irányzatok egyikére, a heavy metal-ra, amely „hitvallásszerűen” terjeszti nótáiban ezt, az egyébként nem túl „szociális” életfelfogást: „Carpe Diem Baby”.[6] Mielőtt végleg elszállna merszünk a mohó algoritmusoktól, térjünk vissza az algoritmusok szelídebb világába!

Az elv: a biztos megoldást jelentő globális optimum felé a *lokális* optimumokon keresztül haladunk. Vajon a másik ide illő latin mondás, „fortiter in re, suaviter in modo”, népszerű magyar fordításban: „bátraké a szerencse” bejön-e mindig a merész programozónak?

Nyilván és sajnos: nem! Amit tehetünk az annyi, hogy mielőtt nekilátunk az ötlet kivitelezéséhez, az algoritmizáláshoz, elgondolkodunk, hogy lehetünk-e „mohók” abban a konkrét esetben, vagy sem? S most már ez nem etikai, hanem programhelyességi kérdés.

[6] http://en.wikipedia.org/wiki/Carpe_diem

1 Egy gondolatébresztő példa – a mohó algoritmusok „szókincse”

1.1 Egy „iskolapélda”

Az alábbi feladat „iskolapéldája”, mondhatni: „állatorvosi lova” a mohó algoritmusok világának. Nézzük a következő esemény-kiválasztási feladatot, mindjárt formalizálva:

Bemenete:

- E={1,...,n} (sorszámokkal adott) esemény vetélkedik egy közös erőforrásért
- minden esemény bekövetkezésének ideje: $[k_i,v_i)$, ahol $k_i<v_i$
- i és j események *kompatibilis*ek, ha $[k_i,v_i)\cap[k_j,v_j)=\varnothing$, azaz bekövetkezésük részben sem eshet egy időre

Kimenete:

- $M=\{e_1,e_2,\ldots,e_{Db}\}$ $e_i\in E$, valahogyan kiválasztott esemény(index)ek
- M maximális elemszámú
- e_i-k páronként kompatibilisek

1.2 A példa „ízlelgetése”

Megoldásként természetesnek látszik egy *backtrack* alapú megközelítés:

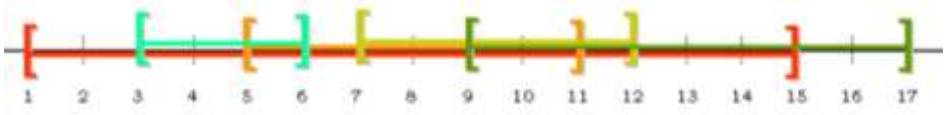
Induljunk ki az elsőből, majd vegyünk egy kompatibilist hozzá, és tegyünk így mindaddig, amíg csak lehet. Ezt tekintsük egy (nem biztos, hogy optimális) megoldásnak, amit feljegyzünk. Backtrack által előírt módon vissza-, majd előrelépve próbálunk újabb megoldáshoz jutni. Ilyenhez érve összevetjük, hogy jobbhoz jutottunk-e, mint a feljegyzett vagy sem. A jobbikat megjegyezzük. Az eljárás végén épp a globális optimum lesz a feljegyzett. Ez bizony igen munkaigényes algoritmus. Legyünk „mohók”, hátha itt nem bűn!

Ízlelgessük a problémát egy konkrét feladaton keresztül!

Mindenek előtt lépésekre bontjuk az eljárásunkat, hogy meglegyen az algoritmus vörös fonala. Ez itt: haladjunk eseményről eseményre, s döntsük el róla, hogy bevesszük-e a megoldásba vagy sem. Tehát semmi visszalépegetés, semmi próbálkozás, csak úgy durrbele!

k_i	v_i
1	15
3	6
7	12
5	11
9	17

ugyanez rajzosan, áttekinthetőbben:



4. ábra

A megoldással próbálkozzunk először „kézzel”, az intervallumok megadási sorrendjében, ahogy a fenti táblázatban szerepel. Kiválasztunk egy soron következőt, és a mögötte jövők között keresek alkalmasakat. Visszatekinteni nem érdemes, hiszen a kompatibilitás „kommutatív” fogalom; így ha volt olyan az előzőek között, akkor az éppen vizsgálttal számításba lett véve. Tehát megoldást nem veszítünk ez által a „nagyvonalúság” által. Persze az esetleges „hiábavalósága” miatt, a hatékonyság így is kérdéses.

1. [1,15) ⇒ nincs több választható, Db=1
2. [3,6) ⇒ [7,12) ⇒ nincs több választható, Db=2

3. ⇒ [9,17) ⇒ nincs több választható, Db=2
4. [7,12) ⇒ (ilyen próbálkozás már volt, nincs több választható) Db=1
5. [5,11) ⇒ nincs több választható, Db=1
6. [9,17) ⇒ ilyen próbálkozás már volt, nincs több választható, Db=1

Látszik, hogy a durrbele módszer *így nem* megy: csak a másodikként kiválasztott kezdő intervallum vezetett optimumhoz.

A „kézi" próbálkozást másodjára az intervallumok kezdete szerint haladva folytassuk; mondván: ne hagyjunk veszendőbe menni egyetlen percet se, így igyekszünk minél optimálisabb megoldáshoz jutni:

1. [1,15) ⇒ nincs több választható, Db=1

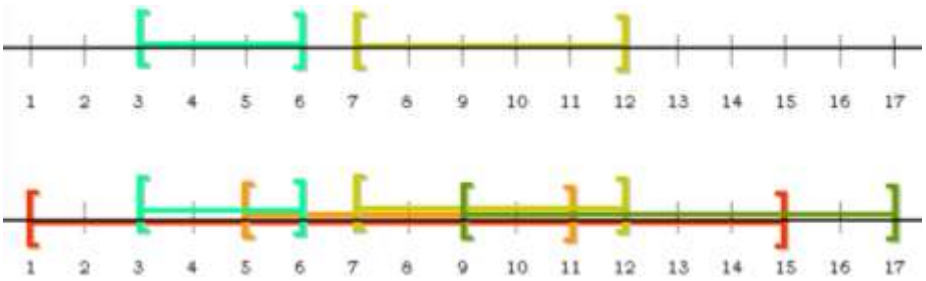
… de már ne is folytassuk! Hiszen épp az a törekvésünk, hogy „visszalépés nélkül", „nyílegyenesen" haladva találjunk rá a megoldásra. A fentit már nem bővíthetjük, s a korábbi próbálkozásainkból tudjuk ez nem optimális megoldás. Tehát következtetésünk: *ilyen sorrendben sem* mohóskodhatunk. Akkor hát mi lehet a vezérelvünk?

1.3 Egy beválónak tetsző mohó módszer

Mivel az események sorrendje a megoldásban érdektelen, ezért feltehetjük, hogy a bemenet v_i-k szerint növekvő (vagy ilyenné tehető). Az i.-hez érve beválasztjuk, ha (az eddigiekkel) kompatibilis, máskülönben nem. Ez logikusnak tűnő gondolat, mivel ekkor az i. mögött csupa olyan esemény található, amely befejezési ideje későbbi (pontosabban: nem előbbi), így a globális cél alapján az i. beválasztásának itt az utolsó esélye, de emellett legkevésbé így akadályoz más választást.

Az ötletet próbáljuk ki a konkrét problémára!

k_i	v_i
3	6
5	11
7	12
1	15
9	17



5. ábra.

Az elképzelt algoritmus pompásan működik a példára. Az az érzésünk, menni fog.

Az érzésen túl azonban be kellene látnunk, hogy ez *általában* is működőképes!

Tegyük föl, hogy M={e_1,e_2,…,e_u} *megoldás* is, *optimális* is! Itt az események indexelése legyen növekvő: $e_1<e_2<\ldots<e_u$.

A v-szerinti rendezés miatt az 1. a leghamarabb befejeződő esemény, azaz $v_1 \le v_i$. Ekkor nyilván: {**1**,e_2,…,e_u} is megoldás, hiszen $v_1 \le v_{e_1}$ és v_{e_1} kompatibilis volt mindegyikkel –feltevésünk szerint–, így ez igaz kell legyen 1-re is, és optimális is, hiszen elemszáma éppenannyi, mint M-é.

Tehát az optimális megoldás *kezdődhet* a *mohó*n választottal.

De vajon ez így folytatható-e?

Legyen E' az 1 választása után maradt kompatibilis események index-halmaza: E'={i | $v_1 \le k_i$ i= 2..n}!

Jelölje M' egy optimális megoldását az E'-vel jellemzett *redukált problémá*nak! Világos, hogy $\{1\}\cup M'$

1. *megoldás*a az eredetinek, mivel az 1. és az M'-beliek kompatibilisek E' definíciója miatt.
2. *optimális* is, hiszen $\{e_2,\ldots,e_u\}\subset E'$ is megoldása E'-nek; így M'-nek legalább u–1 eleme kell legyen, azaz $\{1\}\cup M'$-nek legalább u eleme van, aminél viszont több nem is lehet az M feltételezett „optimális megoldás" volta miatt.

Beláttuk tehát, hogy általában is működik a mohóság a feladatra.

1.4 Mit nyertünk?

Mielőtt további mohóskodásba fognánk, érdemes megvizsgálni, miért is fizetünk a fent körvonalazott matematikai bajlódással, mert az világos, hogy maga az algoritmus igen egyszerű.
A legtriviálisabb hozzáállást jelentő „*generáld és vizsgáld az összes lehetőséget*" *módszer*t ne formalizáljuk! Viszont becsüljük meg, mennyi munka árán jutna el az eredményhez!
Előre nem ismervén az optimális események számát, ez lehet: n vagy n–1 ..., legrosszabb esetben 1. Tehát generálni kell minden i-hez (i=1..n) az összes i eseményt tartalmazó *halmaz*t. Most nem is voltunk „bambák", hiszen eleve kihasználtuk, hogy az *események sorrendje közömbös*. (Így generálni nem is egyszerű!) Ezek száma: $\binom{n}{i}$. Az összes vizsgálatunk számát ezek összege közelíti: $\sum_{i=1}^{n}\binom{n}{i}=2^n-1$. Annak eldöntése, hogy egy esemény i-es kompatibilis eseményekből áll-e, i^2-nel arányos hasonlítást igényel, vagyis $n^{1,5}*2^{n-1}\le\sum_{i=1}^{n}i^2*\binom{n}{i}\le n^2*2^{n-1}$. Ez egy „horribilis" nagyságrendű számmá növekedik n növekedtével.
Meg lehet kísérelni a *backtrack* szokásos gondolatmenetével megoldást találni, hátha jelentős nyereséggel képes e problémán is úrrá lenni.

A lényegi eljárás (Free)Pascal-szerű algoritmusa az alábbi:

```
procedure FeldolgozasBT;//feldolgozás backtrack-kel
  type
    TEsemenyIndexek=array [1..MaxN] of integer;
    TKeresett=record van:boolean; melyik:integer; end;
  var
    i:integer;
    ker:TKeresett;
    //az aktuális eseményrendszer:
    aN:integer;
    aE:TEsemenyIndexek;
    //az eddig legjobb eseményrendszer:
    ljN:integer;
    ljE:TEsemenyIndexek;
begin
  ljN:=0; for i:=1 to MaxN do ljE[i]:=0;
  aN:=1; for i:=1 to MaxN do aE[i]:=0;
  while (aN>=1) do
  begin
    //az aN. helyre keressük a megoldást; aN-1.-ig OK
    ker:=VanJoEset(aN);
```

```
    if ker.van then
    begin
      aE[aN]:=ker.melyik; inc(aN);
    end
      else
    begin
      aE[aN]:=0; dec(aN);
    end;//if VanJoEset
    if aN-1>ljN then//ez eddig a legjobb
    begin
      ljN:=aN-1; ljE:=aE;
      if aN>N {N-nél jobb nem lehet, tehát tovább ne keressünk} then
      begin
        aN:=0;//kiléphetünk
      end;
    end;//if aN>ljN
  end;//while aN>=1
end;//FeldolgozasBT
function VanJoEset(const i:integer):
                              TKeresett;
  var k:TKeresett;
begin
  with k do
  begin
    melyik:=aE[i]+1;
    while (melyik<=N) and
          RosszEset(i,melyik) do
    begin
      inc(melyik);
    end;
    van:=melyik<=N
  end;//k
  VanJoEset:=k
end;//VanJoEset
function Kompatibilis(const i,j:integer):
                                    boolean;
  var tol1,ig1,tol2,ig2:integer;
begin
  tol1:= … az i. esemény kezdete …;
  ig1 := … az i. esemény vége …;
  tol2:= … a j. esemény kezdete …;
  ig2 := … a j. esemény kezdete …;
  Kompatibilis:=(tol1>=ig2) or (tol2>=ig1)
end;//Kompatibilis
function RosszEset(const i,melyik:integer):
                                    boolean;
  var j:integer;
begin
  j:=1;
  while (j<i) and Kompatibilis(aE[j],melyik)
    do inc(j);
  RosszEset:=j<i
end;//RosszEset
```

Néhány futási eredmény, esemény időpontok összehasonlítások számában mérve (H_N), feltüntetve a „bamba” algoritmus hasonló számát is (T_N):

N	H_N	T_N
5	51	179-400
6	836	470-1152
7	919	1185-3136

Az persze nyilvánvaló, hogy az **N** mellett a bemeneti sorozat milyensége is döntő a backtrack számára.

Kipróbálhatjuk ennek Lazarus-os megvalósítását: zip, exe.
Összevetésként nézzük meg a mohó megoldást is!

```
procedure FeldolgozasMH;//feldolgozás mohón
(*
  Ef: beN>0
*)
```

```
  type
    TEsemenyIndexek=array [1..MaxN] of integer;
  var
    i:integer;
    //a legjobb eseményrendszer:
    ljN:integer;
    ljE:TEsemenyIndexek;

begin//FeldolgozasMH
  //rendezés végidő szerint:
  Rendezes(et);
  ljN:=1; ljE[1]:=1;
  for i:=2 to N do
  begin
    if Kompatibilis(ljE[ljN],i) then
    begin
      inc(ljN); ljE[ljN]:=i
    end;//if Kompatibilis
  end;//for i
end;//FeldolgozasMH
procedure Rendezes(var et:…);
  var N,i,j,minj:integer;
begin
  for i:=1 to N-1 do
  begin
    minj:=i;
    for j:=i+1 to N do
    begin
      if ElobbVege(j,minj) then minj:=j
    end;//for j
    s:=et[minj]; et[minj]:=et[i]; et[i]:=s
  end;//for i
end;//Rendezes
```

Néhány futási eredmény, esemény időpontok összehasonlítások számában mérve (H_N), feltüntetve a „bamba” algoritmus hasonló számát is (T_N):

N	H_N	T_N
5	14	179-400
6	20	470-1152
7	27	1185-3136

Az persze nyilvánvaló, hogy az **N** mellett a bemeneti sorozat milyensége is döntő a mohó számára is. A rendezésbeli hasonlításokat is figyelmbe vettük.

```
function ElobbVege(const i,j:integer):
                                        boolean;
  var ig1,ig2:integer;
begin
  ig1:= … az i. esemény vége …;
  ig2:= … a j. esemény vége …;
  ElobbVege:=(ig2>=ig1)
  //"kétszer" egyszerűbb, mint a backtrack alapú
end;//ElobbVege

function Kompatibilis(const i,j:integer):
                                        boolean;
(*
  Rendezést kihasználó vizsgálat.
  Így fele olyan időigényes, mint a viszonyítás-
  ként használt backtrack-es hasonló célú fv.
*)
  var ig1,tol2:integer;
begin
  ig1:= … az i. esemény vége …;
  tol2:= … az i. esemény kezdete …;
  Kompatibilis:=(tol2>=ig1)
  //"kétszer" egyszerűbb, mint a backtrack alapú
end;//Kompatibilis
```

Kipróbálhatjuk ennek Lazarus-os megvalósítását: zip, exe.
Úgy hiszem, magáért beszél a hatékonyságnevekedés. A durva exponenciálisból egy lineárisan növekedő algoritmushoz jutottunk. Gondolom, már elfogadhatjuk az árát: a matematikai megfontolások fáradalmait.

1.5 A mohó módszerek algoritmikus vázlata – összegzés

Mit kell tehát tennünk egy feladat mohó megoldásának tervezésekor?

1. *Lépések*re bontjuk az eljárásunkat.
2. Keresünk a globális optimumra emlékeztető, ámde egy adott lépésnél eldönthető, lokálisan optimális kritériumot, azaz megfogalmazzuk a *mohó választás*t.
3. Mielőtt nekilátnánk az algoritmusnak, belátjuk a mohóság működését, azaz
   a) a „*mohó optimális választási tulajdonság*” – van-e optimális megoldás, ami mohó választással kezdődik
   b) az „*optimális részproblémák tulajdonság*” – a mohó választást hozzávéve a redukált probléma optimális megoldásához, az eredeti probléma megoldását kapjuk

   teljesülését.
4. Algoritmizálás.

Megismertük a mohó algoritmusok szókincsének szavait:

- lépés
- mohó választás
- mohó választási tulajdonság
- optimális részproblémák tulajdonság

Alkalmazzuk!

2 Első példázat a „zenekar”-ról

2.1 A feladat

*„Egy népszerű zenekar a következő 100 napra vonatkozó fellépéseit tervezi. Sok meghívása van fellépésre, ezek közül kell a zenekarnak választani, hogy melyeket fogadja el. Minden fellépés pontosan egy napot foglal el. Minden beérkezett meghívási igény egy (**k, v**) számpárral adott, ami azt jelenti, hogy az igénylő azt szeretné, hogy a zenekar olyan **n** sorszámú napon tartson nála koncertet, hogy **k≤n≤v**. A zenekarnak az a célja, hogy a lehető legtöbb fellépést elvállaljon (természetesen egy napon csak egyet).*
Készítsünk programot, amely kiszámítja, hogy mely meghívásokat fogadjunk el, hogy az összes fellépések száma a lehető legnagyobb legyen; a program adjon is meg egy beosztást!”

2.2 A megoldás

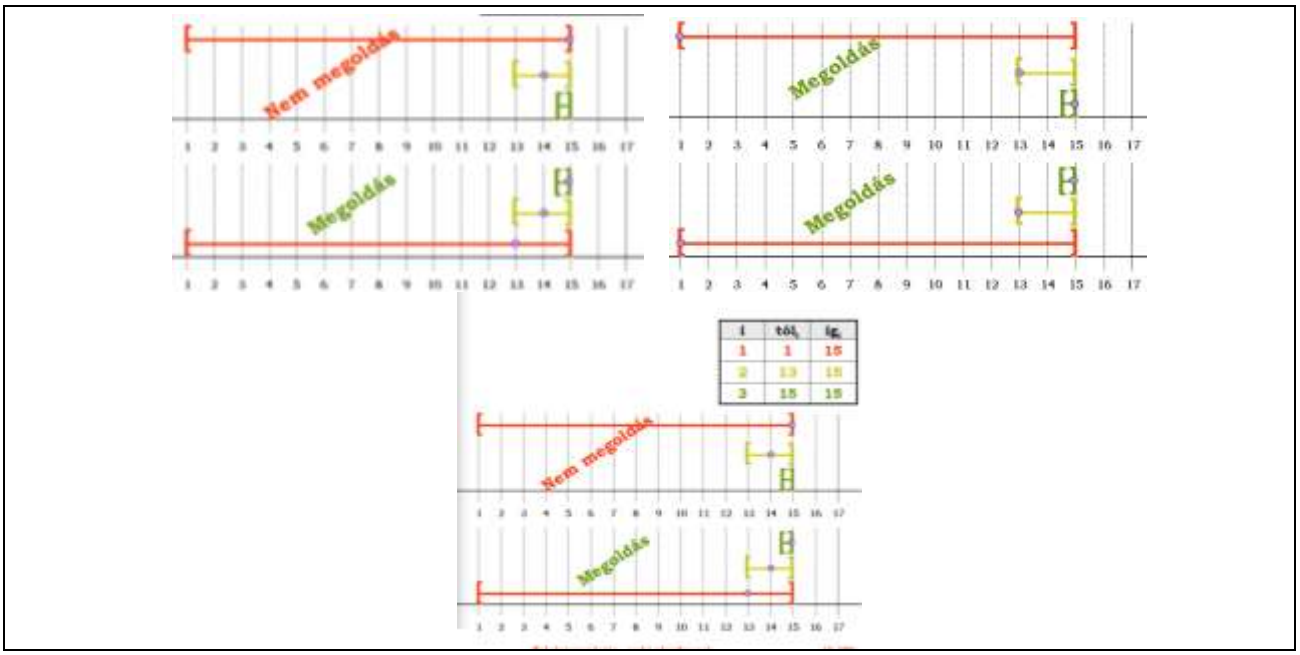
2.2.1 Lépésekre bontás

Sorra vesszük a fellépési ajánlatokat, s eldöntjük, hogy a (mohón) soron következő elvállalható-e vagy sem. Ha igen, feljegyezzük a (később tisztázandó módszerrel) hozzárendelt napot (hogy a későbbiek során e napot ne válasszuk újra), majd rátérünk a következőre.

2.2.2 Mohó választás

Kettős problémával állunk szemben: *intervallum*- és *nap*-választás. Válasszuk azt, amelyik

- a leghamarabb lezárul – ezt kell legsürgősebben választanunk,
- ha több ugyanazon a napon végződik, akkor közülük a legkorábban kezdődőt, hogy minél kevesebbel konkuráljon;
- a kiválasztott ajánlat-intervallumba eső legkorábbi, még választható nap lesz hozzárendelve. [7]



6. ábra. Az azonos befejezésű intervallumok figyelembevételi sorrendjének problémája

[7] Érdemes eltöprengeni a kezdetek szerinti sorrenden. (Egy ellenpélda: [1,5], [1,4], [2,2].)

2.2.3 A mohóság működésének belátása

Jelölések, fogalmak:

- Nap = {1,…,NapDb} – a választható *napok* sorszámainak halmaza
- Fl = {1,…,FlDb} – a *fellépések* sorszámainak halmaza
- k_i,v_i $i\in Fl$ – az i. fellépés *kezdő*, ill. *végső* napsorszáma
- NF = $\{(n_1,f_1),\ldots,(n_{Db},f_{Db})\}$, $n_i\neq n_j$, $f_i\neq f_j$ $(i\neq j)$, $n_i\in[k_{f_i},v_{f_i}]$, $f_i\in Fl$ i,j=1..Db – a *megoldás*, azaz a kiválasztott fellépések (f_i) és a hozzárendelt napsorszámok (n_i) halmaza
- NF *optimális megoldás*, ha nincs nagyobb elemszámú megoldás

Def. (fellépések rendezése)

f,f'$\in$Fl, $f <_f f'$ (f „kisebb/előbbi", mint f'), acsa

a) $v_f<v_{f'}$ (előbb ér véget) vagy

b) $v_f=v_{f'}$ és $k_f<k_{f'}$ (ha ugyanakkor végződik, akkor előbb kezdődik)

Def. (mohó választás)

Sz$\subseteq$Nap „szabad" napok és Fl' fellépések mellett (n^*,f^*) *mohó választás*, ha

a) $f^*=Min_{<_f}\{f\in Fl' \mid [k_f..v_f]\cap Sz\neq\emptyset\}$ – a „legkorábbi" fellépés, amely még elvállalható

b) $n^*=Min\{n\in Sz \mid n\in[k_{f^*}..v_{f^*}]\cap Sz\}$ – a „legkorábbi" fellépés legkorábbi napján

A továbbiakban az NF-et sorszámozzuk f-szerinti sorrendben:

NF = $\{(n_1,f_1),\ldots,(n_{Db},f_{Db})\}$ és $f_i<_f f_{i+1}$

Z1. Állítás (mohó választási tulajdonság):

Ha NF *optimális megoldás* és (n^*,f^*) *mohó választás*,
akkor $NF^* = \{(n^*,f^*),(n_2,f_2),..,(n_{Db},f_{Db})\}$ is *optimális megoldás*.

Bizonyítás:

NF optimális megoldás és (n^*,f^*) mohó választás, de $(_,f^*)\notin$NF $\Rightarrow$

$\exists(n_i,f_i)\in$NF: $n_i=n^*$

Ui. különben ellentmondásra jutnánk NF optimalitásával, hiszen ha $\forall n_i\neq n^*$, akkor NF-hez hozzávehetnénk +1-dikként…

Ez esetben helyettesíthetjük NF-ben (n_i,f_i)-t (n^*,f^*)-gal:

$NF' = \{(n_1,f_1),\ldots,(n_{i-1},f_{i-1}),(n^*,f^*),(n_{i+1},f_{i+1}),\ldots,(n_{Db},f_{Db})\} \Rightarrow NF'$ megoldás is és optimális is

Sorszámozzuk újra az NF'-t f-szerinti sorrend szerint, és megkapjuk

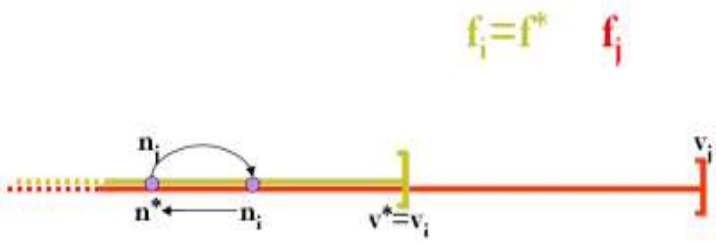
$NF^* = \{(n^*,f^*),(n_2,f_2),..,(n_{Db},f_{Db})\}$

A másik esetben, amikor $(_,f^*)\in$NF:

(n_i,f_i) helyettesítendő (n^*,f_i)-vel, ekkor (n^*,f^*)-gal szükségszerűen ütközést okoz valamely (n_j,f_j)-vel, amelyre éppen teljesül: $n_j=n^*$. (L. a mellékelt ábrát.)

Az (n^*,f^*) mohó választása miatt $v_{f^*}\leq v_{f_j}$, és így $n_i\leq v_{f_j}$.
Az ütközés ténye miatt $k_{f_j}\leq n_i$, így választható az (n_j,f_j) helyett (n_i,f_j).

Így az f^* beilleszthető, a korábban ütköző f_j megtartható a megváltozott napok mellett.



7. ábra. Az n* kiválaszthatóságának magyarázata.

Ez a módosított NF' megoldás is, optimális is. Az átsorszámozás után megkapjuk:

$NF^* = \{(n^*,f^*),(n_2,f_2),..,(n_{Db},f_{Db})\}$

♦

Z2. Állítás (optimális részproblémák tulajdonság):

Ha

1. $Sz \subseteq Nap$ „szabad” napok,
2. (n^*,f^*) *mohó választás* Sz mellett és
3. NF^* *optimális megoldás* $Sz\setminus\{n^*\}$-ra,

akkor

$(n^*,f^*)\cup NF^*$ *optimális megoldása* az Sz-nek.

Bizonyítás:

Először lássuk be, hogy *megoldás*!

NF^* (optimális) megoldás $Sz\setminus\{n^*\}$-ra $\Rightarrow$

$\forall(n_i,f_i)\in NF^* : n^* \neq n_i$

$\Rightarrow (n^*,f^*)\cup NF^*$ *megoldása* lesz az Sz-nek.

Az *optimális* voltát indirekt úton bizonyítjuk:

Legyen $Db=\|NF^*\|$.

Tegyük fel, hogy $\exists NF' = \{(n'_1,f'_1),..,(n'_L,f'_L)\}$ *optimális megoldás* Sz-re, amelyre $L>Db+1$.

A Z1. állításból következik, hogy $NF'' = \{(n^*,f^*),(n'_2,f'_2),..,(n'_L,f'_L)\}$ is *optimális megoldás*.

Világos, hogy az NF”-ből az (n^*,f^*)-t elhagyva egy megoldását kapjuk az $Sz\setminus\{n^*\}$-nak, ami jobb, mint az NF^*, hiszen több $(L-1>Db)$ elemet tartalmaz. Ez ellentmond az állítás 3. feltételének.

♦

2.2.4 A megvalósítás

Lássuk a megvalósítás részleteit, kezdve a reprezentációval, majd folytatva a megoldó eljárás implementálásával.

Ábrázolás [8]

```
Const
  MaxFellepes=255;        {max. fellépésszám; Ef: MaxFellepes<256}[9]
  NapDb=MaxFellepes;      {max. napszám, ami most nem is változik}
Type
  TKeresett=Record igaz:Boolean; nap:Byte End; {lin.kereséshez}
  TFellepes=Record
    db:Byte;
    mely:Array [0..MaxFellepes] of
              Record k,v,s:Byte; end; {fellépés: kezdet/vég/sorsz.}
  End;
  TNapok=Set of Byte;   {nap: melyik fellépés}
  TBeoszt=Array [1..MaxFellepes] of Byte; {beosztott fellépés indexek}
Var{Globális}
  Fel:TFellepes;          {fellépések}
  M:Byte;                 {beosztott fellépések száma}
  Beoszt:TBeoszt;         {beosztott fellépések sorszáma}
  FoglNapok:TNapok;
```

A lényegi rész algoritmusa

A mohó választás hatékony megvalósítása érdekében egyszer, a legelején rendezzük a fellépéseket a korábban definiált $<_f$ reláció szerint. Majd a lehetséges fellépés-ajánlatokon végig menve eldöntjük, hogy a mohó választással szóba kerülhet-e egyáltalán, vagyis van-e olyan nap, ami a hozzátartozó nap-intervallumba esik.

[8] Az inputot, ill. az outputot meghatározó adatokat színekkel megkülönböztettük.

[9] Későbbi ábrázolásnál kihasználjuk, hogy 1 bájtos adat. Szigorúan véve: ez nem olvasható ki a feladatból!

```
Procedure FellepestUtemez;
  Var i:Byte;
      OK:TKeresett;
Begin{FellepestUtemez}
  Rendezes(Fel); {a Fel rendezése v/k mező szerint}
  FoglNapok:=[Fel[1].k]; M:=1; Beoszt[1]:=Fel[1].s;
  For i:=2 to Fel.db do
  Begin
    OK:=VallalhatoE(i); {az i. fellépés elvállalható-e a beosztottak mellett}
    If OK.igaz then
    Begin
      FoglNap:=FoglNap+[OK.nap]; Inc(M); Beoszt[M]:=Fel[i].s;
    End{If};
  End{For i}
End{FellepestUtemez};
Function VallalhatoE(Const i:Byte):TKeresett;
  Var j:Word;
      k:TKeresett;
Begin
  j:=Fel[i].k;
  While (j<=Fel[i].v) and (j in FoglNap) do Inc(j);
  k.igaz:=j<=Fel[i].v;
  If k.igaz then k.nap:=j;
  VallalhatoE:=k
End;{VallalhatoE }
```

A futó program próbája: ZENEKAR.EXE.

Érdemes eltöprengeni a feladat apró módosításaival kapott változatain! Például: hogyan oldható meg, ha

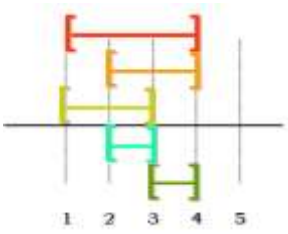
1. a meghívások nemcsak egy napra vonatkoznak, hanem a teljes intervallumra;
2. az egyes meghívásokhoz valamekkora fizetség tartozik; ez esetben természetesen az összfizetség maximumára kell törekedni;
3. a meghívások a teljes intervallumra vonatkoznak és tartoznak hozzájuk összegek is; ismét az összbevételt kell maximalizálni;
4. a meghívások a teljes intervallumra vonatkoznak, de most a lekötött összes napok számát kell maximalizálni.

2.3 Egy variáció

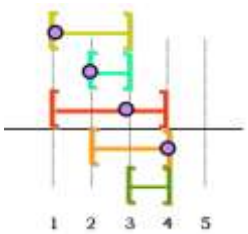
Vizsgájuk meg a 2. változatot, amelyben tehát *minden fellépés járjon valamennyi (h_i) haszonnal. Ekkor a célfeladat úgy módosul, hogy a lehető legtöbb haszonra kell törekedni.*
Ez az apró módosítás mennyiben érinti a megoldást? Az várható, hogy az eddigi törekvés, miszerint „minél több fellépést kell elvállalni”, nem feltétlenül hozza a legtöbb hasznot.

2.3.1 Egy ellenpélda

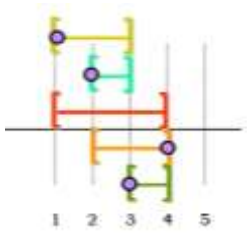
k_i	v_i	h_i
1	4	3
2	4	4
1	3	1
2	3	2
3	4	5



8. ábra



9. ábra



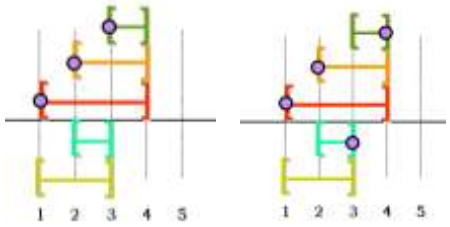
10. ábra

A 8. ábra mutatja az alaphelyzetet. A 9. a mohó módon (az intervallumvégek figyelembe vételével) rendezett, majd e szerint megoldott helyzetet ábrázolja. Látható, hogy az összhaszon: 1+2+3+4= =10. Könnyen észrevehető, hogy akár a 3. napra, akár a 4.-re vállalt fellépés helyett a legtöbb hasznot hozót (az eredeti táblában szereplő 5.-ket) választjuk, a haszon növelhető. (L. a 10. ábrát.)
A tapasztalat lesújtó: a „szokásos" mohó algoritmus nem jár sikerrel, legalábbis a „súlyozott" problémára. Lehet, hogy erre viszont egy másik mohó döntés hozza meg a várva várt sikert?

2.3.2 A logikus út megsejtése – újabb kudarc?

Az optimalizációs szempontunkkal összhangban válasszuk a rendezés szempontjául is a *hasznot*! A 11. ábra példája mutatja, hogy elérhető az előbb talált 12 értékű megoldás, mégha nem is ugyanazt találtuk. Tüzetesebb vizsgálattal az is kiderül, hogy lenne még ennél is jobb választás. A javasolthoz képesti módosítás: **3**. fellépés 5. napon, a **4**. fellépés a 3. napon; így az összhaszon: 5+4+3+2=14.

k_i	v_i	h_i
3	4	5
2	4	4
1	4	3
2	3	2
1	3	1



11. ábra. A vélt optimum és „cáfolata"

Ha pontosabban elemezzük a kudarcot, rájövünk, hogy az ütközések számának figyelmen kívül hagyása okozta. Ha tehát a „mohóságot" nem a haszonra, hanem az ütközés-számra vonatkoztatjuk, akkor talán rátalálunk az optimumra.

Mindenekelőtt számláljuk meg az egyes napokra hány ütközés esik! Az intervallumokat *rendezzük* most is *haszon* szerint. Csakhogy a kiválasztott intervallumnak a *legkevésbé ütköző* napját foglaljuk le, hogy ez által a legkevesebb intervallumot zárjunk ki a további versengésből.

Ezt az algoritmust a fenti példára alkalmazva, a várt optimumot kapjuk: hiszen az egyes napok ütközés-száma így alakul: 1→2, 2→4, 3→5, 4→3, 5→0. Haszonsorrendben végighaladva az intervallumokon, és lefoglalva a lehetséges, minimális ütközés-számú napot a „frekvenciákra" az alábbi értékeket kapjuk:

	1.	*2.*	*3.*	*4.*	*5.*	***nap***
0.	2	4	5	3	0	
1.	2	4	**4**	**[2]**	0	
2.	2	**[3]**	**3**	**[1]**	0	
3.	**[1]**	**[2]**	**2**	**[0]**	0	
4.	**[1]**	**[1]**	**[1]**	**[0]**	0	
5.	**[0]**	**[0]**	**[0]**	**[0]**	0	

lépés után

3 Második példázat a „fényképeszkedés”-ről

3.1 A feladat

„Egy rendezvényre N vendéget hívtak meg. Minden vendég előre jelezte, hogy mettől meddig lesz jelen. A szervezők fényképeken akarják megörökíteni a rendezvényen résztvevőket. Azt tervezik, hogy kiválasztanak néhány időpontot, és minden kiválasztott időpontban az akkor éppen jelenlevőkről csoportképet készítenek. Az a céljuk, hogy a lehető legkevesebb képet kelljen készíteni, de mindenki rajta legyen legalább egy képen.
Írjunk programot, amely kiszámítja, hogy legkevesebb hány fényképet kell készíteni; és megadja azokat az időpontokat is amikor csoportképet kell készíteni!”

3.2 A megoldás

3.2.1 Lépésekre bontás

Sorra vesszük a vendégeket, s eldöntjük, hogy a soron következő olyan-e, aki távozásakor még nem lett lefényképezve, azaz megérkezése óta még nem volt fényképezés. Ha igen, megjegyezzük, majd rátérünk a következőre.

3.2.2 Mohó választás

Válasszuk a *legkorábban távozó* vendéget, aki a *legutóbbi fényképezés óta érkezett*.

3.2.3 A mohóság működésének belátása

Jelölések, fogalmak:

- $V = \{v_1,\ldots,v_{vDb}\}$, ahol $v_i = (tól_i, ig_i)$ $(i=1..vDb)$ – a *vendégek* ott tartózkodása
- $FT = \{ft_1,\ldots,ft_{ftDb}\}$, ahol $ft_i<ft_{i+1}$ $(i=1..ftDb-1)$ – a *megoldás*, azaz a *fényképezés időpontjai*
- FT *optimális megoldás*, ha nincs kevesebb elemszámú megoldás

Def. (vendégek rendezése)

$v_i, v_j \in V$, $v_i \leq_{ig} v_j$ („kisebb/előbbi”), acsa $v_i.ig \leq v_j.ig$ (azaz előbb indul haza)

A továbbiakban a V-t $\leq_{ig}$-*rendezett*en ábrázoljuk, ami persze „csak” kezelési egyszerűsítést jelent.

Def. (mohó választás)

$W \subseteq V$ még le *nem* fényképezettek és ft *utolsó* fényképezés mellett ft^* *mohó választás*, ha $ft^*=v.ig$, amelyre $v:=Min_{\leq_{ig}} \{v \in W \mid v.tól>ft\}$ – az ft után érkezők közül a „legkorábban” távozó (időpontja).

F1. Állítás (mohó választási tulajdonság):

Ha $FT = \{ft_1,ft_2,\ldots,ft_{ftDb}\}$ *optimális megoldás* és ft^* *mohó választás*,
akkor $FT^* = \{ft^*,ft_2,\ldots,ft_{ftDb}\}$ is *optimális megoldás*.

Bizonyítás:

Ha $ft_1 \leq ft^*$, akkor
ft_1 helyett ft^* is szerepelhet, mivel ft^*-ig nincs távozás.
Így FT *megoldás* voltából következik FT^* *megoldás* volta, és
FT *optimalitás*a garantálja FT^* *optimalitás*át.
Az $ft_1>ft^*$ nem lehetséges, mivel
az első vendégről nem készülhetett volna fotó ($ft^*=v_1.ig<ft_1$), így FT *nem* lenne *megoldás* sem, ami ellentmondana feltételezésünknek.

♦

Jelölés: $FT_W = \{ft_1,\ldots,ft_k\}$, W-ben szereplő vendégek esetén *optimális megoldás*.

F2. Állítás (optimális részproblémák tulajdonság):

Ha

1. ft^* *mohó választás* és
2. $FT_W = \{ft_2,...,ft_{ftDb}\}$ *optimális megoldás* a $W = \{v_i \mid v_i.tól > ft^*\}$ esetén,

akkor

$FT_V = \{ft^*,ft_2,..,ft_{ftDb}\}$ *optimális megoldás* (azaz a teljes V esetén optimális megoldás).

Bizonyítás:

FT_V *megoldás*:

FT_W megoldás W-re $\Rightarrow$ $\forall v_i$-ről, aki ft^* után érkezett, készült fénykép
ft^* értelmezés $\Rightarrow$ ft^*-ig érkezetteket ft^*-kor lefényképeztek
} $\Rightarrow$ FT_V *megoldás*

FT_V *optimális* megoldás (indirekt)

Tfh: FT_V nem optimális megoldás, azaz

$\exists FT'' = \{ft''_1,ft''_2,..,ft''_k\} \neq FT_V$ *optimális megoldás*, amelyre $k<ftDb$.

Ekkor F1 állításból következik, hogy $\exists FT^* = (ft^*,ft''_2,...,ft''_k)$ is *optimális megoldás*, azaz az összes vendég fényképre kerül.

Hagyjuk el azon vendégeket, akik ft^*-kor jelen voltak, s így ekkor fényképre kerültek!

Ezek éppen a W-t alkotják.

Mivel FT^* megoldás során mindenkiről készült fénykép, és ft^*-kor a W-beliekről nem készülhetett, ezért róluk csak a későbbi időpontokban készülhetett.

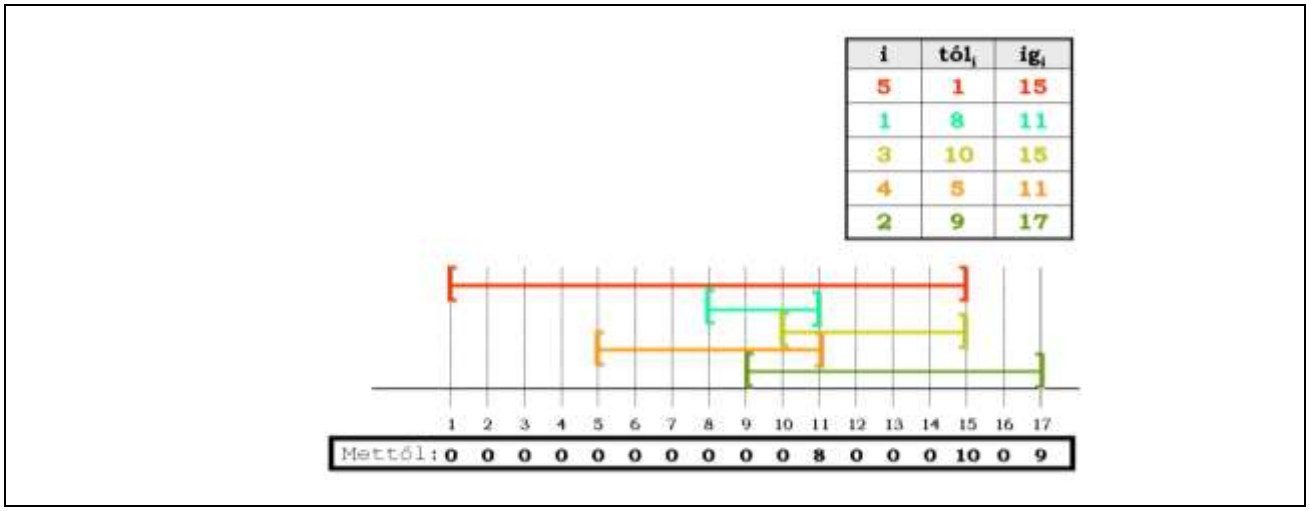
Ez azt jelenti, hogy $(ft''_2,...,ft''_k)$ megoldás W mellett, vagyis jobb, mint az optimálisnak feltételezett FT_W. Ez ellentmondás.

♦

3.2.4 A megvalósítás

Ábrázolás

A vendég-intervallumok idő-hatékonyabb tárolását jelenti, ha olyan tömbben gondolkodunk, amely minden (lehetséges) időponthoz megadja az akkor távozók legkésőbbi érkezési idejét. Azaz a legrövidebb ideig ott tartózkodókat, mint kritikusakat, adminisztráljuk csupán.



12. ábra. A „Mettől" tömb értelmezése.

Így egyrészt az azonos módon kezelhető vendégek számát is csökkenthetjük, ráadásul eleve rendezve lesz távozási idő szempontjából, ami a mohósághoz igen előnyös. Ezt mindjárt a beolvasáskor elintézhetjük.

```
Const
  MaxN=3000;                    {az intervallumok max. száma}
  MaxT=1000;                    {a megoldás időpontok max. száma}
Var
  vDb   :Word;                  {az intervallumok száma}
  Mettol:Array[1..MaxT] of Word; {az intervallumok:
                                  [Mettol[i],i), ha Mettol[i]>0}
  ftDB:Word;                    {a megoldás időpontok száma}
  FT  :Array[1..MaxN] of Word;  {a megoldás időpontok halmaza}
```

A lényegi rész algoritmusa

A beolvasást csak specialitása, érdekessége miatt mutatjuk be:

```
Procedure Beolvas;
{Globális Output: vDb, Mettol}
  Var  Bef:Text;
       i,tol,ig:Word;
Begin{Beolvas}
  For i:=1 to MaxT do Mettol[i]:=0;
  Assign(Bef,'fenykep.be'); Reset(Bef);
  ReadLn(Bef,vDb);
  For i:=1 to vDb do
  Begin
    ReadLn(Bef,tol,ig);
    If tol>Mettol[ig] then Mettol[ig]:=tol;
  End;
  Close(Bef);
End{Beolvas};
```

… és maga, a mohó eljárás:

```
Procedure FotoUtemezes;
{Globális input:  MaxT, Mettol
 Golbális output: ftDB, FT}
  Var  i,t:Integer;
       Utolso:Integer;
Begin{FotoUtemezes}
  Utolso:=0;{az utolso beválasztott időpont}
  ftDB:=0;  {a beválasztott időpontok száma}
  For t:=1 to MaxT do
  Begin
    If (Mettol[t]>0){volt távozás?} and
       (Utolso<Mettol[t]){az utolsó fényképezés óta jött?} then
    Begin
      Utolso:=t;
      Inc(ftDB);
      FT[ftDB]:=Utolso;
    End;
  End{For t};
End{FotoUtemezes};
```

A futó program próbája: FENYKEP.EXE.

4 Harmadik példázat az „ültetés”-ről

4.1 A feladat

„Egy rendezvényt olyan teremben tartanak, ahol M db ülőhely van. Az ülőhelyek 1-től M-ig sorszámozottak. A rendezvény szervezője megrendeléseket fogad. Minden megrendelés egy (A,B) számpárt tartalmaz, ami azt jelenti, hogy a megrendelő olyan ülőhelyet szeretne kapni, amelynek S sorszáma A és B közé esik (A≤S≤B).
Írjunk programot, amely kiszámítja, hogy a szervező a megrendelések alapján a legjobb esetben hány megrendelést tud kielégíteni; és meg is ad egy olyan jegykiosztást, amely kielégíti a megrendeléseket!”

4.2 A megoldás

Ez a feladat megegyezik az elsővel, csak a megfogalmazás más. Most azonban egy másik, ugyancsak mohó megoldását mutatjuk meg.

4.2.1 Lépésekre bontás

Nem az igények szerint haladva végzünk mohó választást, hanem az erőforrás, tehát az ülőhelyek szerint. Az adott ülőhelyhez választunk mohó módon megfelelő megrendelést.

4.2.2 Mohó választás

Az ülőhelyek sorszáma szint növekvően haladva, válasszuk azt a megrendelést, aminek ez az ülőhely megfelelő, és az igényelt intervallum jobbvégpontja a legkisebb. (Természetesen, ha nincs ilyen, akkor az adott szék üresen marad.)

4.2.3 A mohóság működésének belátása

Jelölések, fogalmak:

- Szék = {1,…,SzékDb} – a választható *székek* sorszáma
- Mr = {1,…,MrDb} – a *megrendelések* sorszámainak halmaza
- k_i, v_i i∈Mr – az i. megrendelés *kezdő*, ill. *végső* széksorszáma
- MSz = {(sz_1,m_1),…,(sz_{Db},m_{Db})}, $sz_i \neq sz_j$ (i≠j), $sz_i \in [k_{mi}, v_{mi}]$, m_i∈Mr i,j=1..Db – a *megoldás*, azaz a kiválasztott megrendelések (m_i) és a hozzárendelt széksorszámok (sz_i) halmaza
- MSz *optimális megoldás*, ha nincs nagyobb elemszámú megoldás
- V(sz)= {m∈Mr: $k_m \leq sz \leq v_m$ }, sz∈Szék; azon megrendelések halmaza, amelyeknek sz megfelelő.

Def. (megrendelések rendezése)

m,m’∈Mr, m $<_m$ m’ (m „kisebb/előbbi”, mint m’), acsa

- $v_m < v_{m'}$ (előbb ér véget) vagy
- $v_m = v_{m'}$ és $k_m < k_{m'}$ (az ugyanakkor végződik, akkor előbb kezdődik)

Def. (mohó választás)

Sz⊆Szék „szabad” székek és Mr’ megrendelések mellett (sz^*,m^*) *mohó választás*, ha

a) $sz^* = \text{Min}\{sz \mid sz \in Sz \text{ és } V(sz) \neq \emptyset\}$ – a legkisebb sorszámú igényelt szék

b) $m^* = \text{Min}_{<_m}\{V(sz^*)\}$ – a legkisebb jobbvégpontú megrendelés, amelynek sz* megfelelő.

Ü1. Állítás (mohó választási tulajdonság):

Ha MSz= {(sz_1,m_1),…,(sz_{Db},m_{Db})}, *optimális megoldás* az Sz, Mr bemenetnek, ahol
$sz_1 < sz_2 < \ldots < sz_{Db}$ és (m^*,sz^*) *mohó választás*,
akkor MSz^* = {(m^*,sz^*),(m_2,sz_2),..,(m_{Db},sz_{Db})} is *optimális megoldás*.

Bizonyítás:

Mivel sz* a legkisebb olyan szék, amelyre van igény, ezért $sz^* \le sz_1$

Két este lehet: $sz^*=sz_1$ vagy $sz^*< sz_1$

$sz^*=sz_1$: Ha m* nem szerepel az MSz optimális megoldásban, akkor (sz_1,m_1) helyett vehetjük (m^*,sz^*)-et. Ha m* szerepel az optimális megoldásban, azaz $m^*=m_i$, valamely i re, akkor $sz_i \le v_{m^*}$, de $\le v_{m^*} \le v_{m1^*}$, így (sz_1,m_1) helyett (sz*,m*)-et és (sz_i,m_i) helyett (sz_1,m_i) is optimális megoldás lesz.

$sz^*< sz_1$: Ekkor m* biztosan szerepel az MSz optimális megoldásban, mert egyébként (sz*,m*)-et hozzávéve Db+1 elemű megoldást kapnánk. Legyen $m^*=m_i$. Ekkor (szi,m_i) helyett (sz^*,m_i) vehető.

Ü2. Állítás (optimális részproblémák tulajdonság):

Ha

1. Sz={első,…,SzékDb} ⊆Szék „szabad” székek,
2. (sz^*,m^*) *mohó választás* az (Sz, Mr) bemenetre és
3. MSz^* *optimális megoldás* $(Sz\setminus\{sz^*\}, Mr\setminus\{m^*\})$-re, akkor $(sz^*,m^*)\cup MSz^*$ *optimális megoldása* az(Sz, Mr)-nek.

Bizonyítás:

Az $(Sz\setminus\{sz^*\}, Mr\setminus\{m^*\})$ redukált problémának legalább Db-1 elemű az optimális megoldása, mivel $\{(m_2,sz_2),..,(m_{Db},sz_{Db})\}$ megoldás. Db-nél nagyobb elemszámú megoldása nem lehet, mert ha lenne, akkor ahhoz hozzávéve (sz*,m*)-ot, a kiindulási (Sz, Mr) problémának lenne Db-nél nagyobb elemszámú megoldása. Tehát, mivel $(sz^*,m^*)\cup MSz^*$ megoldása az (Sz, Mr) problémának, optimális is, mert Db számú eleme van.

4.2.4 A megvalósítás

Ábrázolás

Minden ***v*** ülőhelyre egy láncba rakjuk az (***i, k***) számpárokat, ha az i-edik megrendelés a (***v, k***) pár. Ezáltal elkerülhetjük az intervallumoknak a kezdőpontjuk szerinti rendezését. A mohó választás megvalósítására prioritási sort használunk. Pontosabban, a prioritási sor adott x ülőhely vizsgálatakor azokat az (i, k_i) párokat tartalmazza, amelyekre teljesül, hogy $v_i \le x \le k_i$ és az i megrendelést még nem szerepel a megoldásban. A prioritási sor a k végpontok szerinti minimumos prioritási sor.

Az alábbi kód C++ nyelven ad meg egy lehetséges megoldást.

```
#include <stdlib.h>
#include <stdio.h>
#include <queue>
#define maxN 100001
#define maxM 10001

using namespace std;

    struct Par {
        int az, v;
        bool operator<( const Par &p ) const {
            return v > p.v;
        }
    };
    typedef struct Lanc{
        Par adat;
        Lanc *csat;
    }Lanc;
    Lanc* Igeny[maxM];
    int Ultet[maxM];
```

```
int main() {
    Lanc* p;
    int m,n,k,v, hany=0;
    Par q;
    scanf("%d %d",&m,&n);

    for (int i=1; i<=n; i++){
        scanf("%d %d",&k,&v);
        p = new Lanc;
        p->adat.az=i; p->adat.v=v;
        p->csat=Igeny[k];
        Igeny[k]=p;
    }
    priority_queue<Par> Q;
    for (int x=1; x<=m; x++){
        p=Igeny[x];
      // az x kezdőpontú megrendelések mennek a sorba
        while (p !=NULL){
            Q.push(p->adat);
            p=p->csat;
        }
        if (Q.size()>0 ){//a mohó választás
            hany++;
            q=Q.top(); Q.pop();
            Ultet[x]=q.az;// az x helyet a q.az megrendelés kapja
        }
      //az x végpontú megrendelések kivétele a sorból
        while(Q.size()>0 && Q.top().v==x)
            Q.pop();
    }//for x
    printf("%d\n",hany);
    for (int x=1; x<=m; x++)
        if (Ultet[x]>0)
            printf("%d %d\n",x, Ultet[x]);
    return 0;
}
```

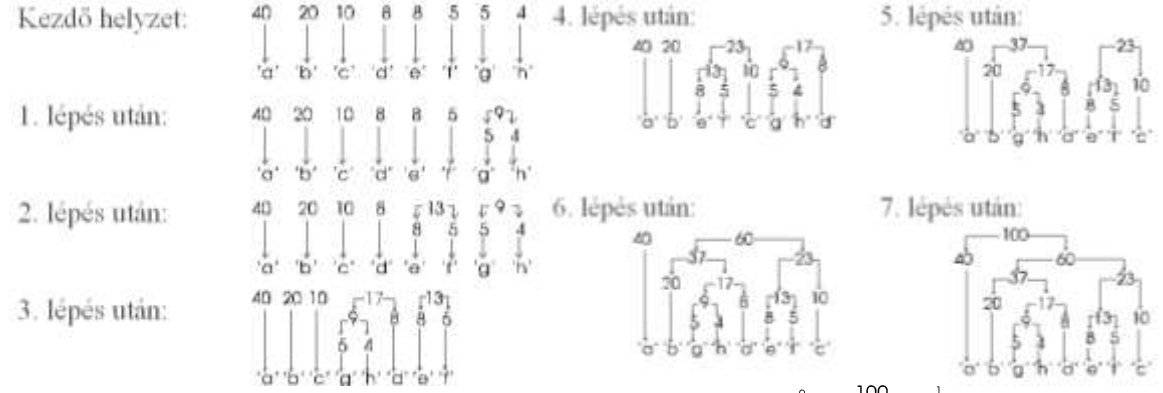
Az algoritmusa futási ideje

Ha a bementben az ülőhelyek száma m, az igények száma pedig n, akkor az algoritmus futási ideje legrosszabb esetben $O(n \cdot \log m)$.

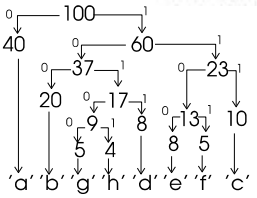
5 Konklúzió?

Néhány további klasszikus feladat, de most már részletes elemzés nélkül.

- Huffman-kódolás
  Adott egy karaktersorozat, tömörítsük úgy, hogy minden karakterhez egy új, változó bitszámú kódot rendelünk, amelyekből állítjuk össze a tömör kódsorozatot!
  Mohó választás: egy bináris fa felépítése a gyakoriságszerinti legkisebbek egyesítésével, kumulált gyakorisággal.



Végül a kódfa:



- Darabolási feladat
  Adott egy fémrúd, amelyet megadott számú és hosszúságú darabokra kell vágni. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. A vágások tetszőleges sorrendben végezhetők. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen két darabra vágunk. Milyen sorrendben kell a darabolást végezni, hogy minimális legyen a költség?
  Mohó választás: a legnagyobb darabot
- Pénzváltási feladat(ok)
  N Ft-ot kell váltani úgy, hogy a lehető legkevesebb pénzérmét használjunk fel, ha az érmék:
  a) 100, 50, 20, 10, 5, 2, 1 és N tetszőleges
  Mohó választás: a legnagyobb érméből, a lehető legtöbbet
  b) 3, 2 és N=7
  a mohó nem ad semmilyen eredményt!
  Hipotetikus ok: nem minden szám (1) állítható elő ezek összegeként.
  c) 8, 6, 4, 1 és N=10
  a mohó nem optimális eredményt ad.
  Hipotetikus ok: egymásnak nem egész-számú többszörösei (c_{i+1} † c_i).

Kérdések:

- Vannak-e felismerhető jegyei a feladatoknak?
- Láttuk: sablon az van. De, milyen csapdák vannak? Példák:
  - „Apró” módosítások – egy csipetnyi pénz a feltételek közé
  - Egyebek

6 Komolytalan befejezés, ha még nem lenne elég a jókedv

A jó kedvért és a mohósághoz néhány web-irodalmi adalék:

- „*Ami az embert jóllakottá teszi, az nem a táplálék mennyisége, hanem a mohóság hiánya.*”
- *„A gazdagságról*

  Bőség utáni vágyadat ne piszkítsa mohóság,
  Mert ínségben akkor segíthetsz másokon,
  Ha nem ülsz fenekeddel vagyonodon.
  A pénz, ha gazul szerzik s költik,
  Csak kárt okoz, s nem szolgál,
  Akár a víz, mely csónakod felbillenti inkább,
  Semmint szelíden vinné az árral tovább.”

  Lin Yun
  2001 nyarán
- Bhagavad-Gita: A Bhagavad-gítá a védikus irodalom egyik alapműve. Eredetileg a Mahábhárata című eposz része. Jelentősége a hinduizmuson belül hasonló mértékű, mint a keresztény vallásban a Bibliáé. A Bhagavad-gíta cím szó szerint azt jelenti, hogy „Isten éneke”. Tizennyolc fejezetből és hétszáz szanszkrit nyelvű (négysoros) versből áll a mű, amely több mint ötezer éve örvend köztiszteletnek. [10]

  Tizenhatodik fejezet [21]
  ***„…** Három kapu nyílik e pokolba: a kéj, a düh és a **mohóság**. Minden józan embernek meg kell válnia tőlük, mert a lélek lealacsonyodásához vezetnek. **…**”*

  Tizennegyedik fejezet [17]
  ***„…** A jóság kötőerejéből igazi tudás származik, a szenvedélyből **mohóság**, a tudatlanságból pedig ostobaság, őrültség és illúzió. **…**”*
- Karinthy Frigyes: A magyar irodalom egyik „alapműve”. Eredetileg a világirodalom része. Jelentősége a minden irodalmi izmuson belül hasonló, mint a keresztény vallásban a pápáé. A „Karinthy Frigyes” név szó szerint azt jelenti, hogy „Karinthy Frigyes”, bár jelenthetne egészen mást is. Tizennyolc milliárd neuronból, s legalább ennyi gliasejtből áll agya, a leginkább említésre érdemes „fej”-ezete, amely „papír-lenyomata” több mint ötven éve örvend köztiszteletnek.[11]

  „*A jótulajdonságokat szeretjük – a rosszakba szerelmesek vagyunk.*”
  Karinthy Frigyes
- Vicc:
  „– Asszony! Fantasztikus ember ez az Oszkár. Minden kell neki, amit csak meglát.
  – Akkor bemutathatnád neki a lányunkat...”
- És egy kép a mohóságról:

[10] http://www.krisna.hu/kiadvanyok/sp/bg/bev.htm
[11] SzP

7 Irodalom

Horváth Gyula: „*Mohó algoritmusok*”, NJSzT, 2003
Program-megoldások, Nemes Tihamér Országos Számítástechnikai Tanulmányi Verseny Bizottság
http://en.wikipedia.org/wiki/Main_Page
http://www.krisna.hu/kiadvanyok/sp/bg/bev.htm
Szlávi Péter: „*Mohó algoritmusok módszertana*”, e-jegyzet (PDF), 2003