Az előző részben megoldottunk néhány problémát mohó algoritmusok segítségével. A kitűzött feladatok mindegyikénél sikerült egy optimális megoldást találni úgy, hogy a megoldást végző algoritmus nem vizsgált meg minden lehetséges esetet, hanem minden lépésnél egy, az addigi ismeretek alapján optimális lehetőséget választott. Ezzel ugyanakkor egyszerűsítette a problémát, vagyis a továbbiakban egy kisebb halmazból kellett választani, és az eredetivel ekvivalens feladatot megoldani.

Az előző részben a megoldások során nem igazoltuk, hogy a mohó stratégiával talált eredmény valóban egy optimális megoldás. Bár ez sok esetben szinte magától értetődik, mégis érdemes ezeket a könnyebb példákat is végiggondolni.

Nézzük először a címletezés problémáját. A ma használatos forint érmék vagy papírpénzek esetén sikerült mohó módszerrel megoldást adnunk. Az eljárás bármely pénzösszeg esetén a lehető legnagyobb olyan címletből választ, amely még nem nagyobb az összegnél. Ez a mohó módszer azonban nem működik pl. az 1, 5, 7, 10 címleteknél, hiszen a 14-et öt érmével fizeti ki, pedig kettővel is lehet. Tehát nem minden címletezési problémára ad optimális megoldást a mohó módszer.

Vizsgáljunk olyan címleteket, ahol a pénzértékek közelebb állnak a megszokotthoz, pl. nem fordul elő, hogy két kisebb értékű címlet összege meghalad egy nagyobb értékű címletet. Oldjuk meg a címletezés feladatát például azzal a feltétellel, hogy bármely címlet legalább kétszerese a csökkenő sorrendben utána következőnek.

Ekkor elég egyértelműnek tűnik, hogy a mohó módszerrel kapott eredmény optimális. De hogyan tudnánk ezt bizonyítani? Mivel annyira nyilvánvaló a dolog, próbáljuk meg az indirekt bizonyítást. Tegyük föl, hogy van egy optimális megoldás, amely nem azonos a mohó módszerrel kapott megoldással, sőt, kevesebb számú pénzzel fizet ki egy bizonyos pénzösszeget. Vizsgáljuk a két megoldást a címletek szerint csökkenő sorrendben, és keressük az első címletet, ahol darabszámban eltérés van. Nézzük azt a pénzösszeget, ami a közösen, azonos számban választott nagyobb pénzek fölhasználása után megmaradt. Ha a mohó módszer esetén van kevesebb számú ebből a címletből, az azt jelenti, hogy a fennmaradó, kifizetésre váró pénzösszeg kisebb, mint a címlet. De ekkor a másik, optimális megoldás nagyobb összeget fizetett ki, mint szükséges, hiszen ebből a címletből még legalább egyet választott. Ez azt jelenti, hogy a másik megoldás nem lehet optimális megoldás, sőt, nem is helyes címletezés. Ha a mohó módszer esetén van több az első különböző számú címletből, akkor az előbb definiált fennmaradó összeg nem kisebb, mint a címlet. Ha a mohó megoldás egy ilyen címlettel kifizet egy összeget, akkor a nem mohó megoldásban a feltétel szerint ennek az összegnek a kifizetéséhez legalább két pénzdarab szükséges. Ha tehát ezekkel a pénzösszegekkel mindkét megoldás szerint fizetünk, akkor a mohó megoldás egy darab, míg a másik legalább kettő darab pénzzel fizeti ki ugyanakkora összeget. Folytassuk a címletek összehasonlítását a kisebb címletek felé haladva, és tegyük ugyanazt, mint az előbb. Nyilván ugyanazok az esetek lehetségesek, mint az előbb, vagyis nem fordulhat elő, hogy a második, optimálisnak mondott megoldás kevesebb pénzdarabot használ föl azonos összegek kifizetésére. Ez azt jelenti, hogy a másik optimális megoldás vagy mégsem optimális, vagy megegyezik a mohó algoritmussal kapott megoldással.

A címletezés problémakört még folytatjuk majd, mivel további izgalmas kérdéseket vet föl, hogy hogyan oldható meg más feltételekkel, vagy akár bármilyen feltétel nélkül a probléma.

Nézzük meg a mohó megoldást az előző részben tárgyalt fényképezés feladatnál. Megállapítottuk, hogy a feladat ekvivalens azzal, hogy N darab balról zárt, jobbról nyílt $[E_i, T_i)$ intervallumhoz $(1 \leq i \leq N)$, melyek végpontjai egészek, meg kell keresnünk a lehető legkisebb K számú F_j egész pontot $(1 \leq j \leq K)$, amelyekre teljesül, hogy mindegyik intervallumban szerepel közülük legalább egy. A mohó módszert itt úgy alkalmaztuk, hogy a T_i értékek szerint nem csökkenő sorrendbe állítottuk az intervallumokat, majd kiválasztottuk az elsőt, és az intervallum végénél eggyel kisebb egész lett F_1. Ezután elhagytuk azokat az intervallumokat, amelyek tartalmazták F_1-et, és a megmaradtak közül választottuk a következő legkisebb T_i értékűt, illetve F_2 az annál eggyel kisebb egész lett.

Nézzük, hogy most hogyan igazolható, hogy a mohó megoldás optimális. A módszer abból indult ki, hogy a sorrendben első intervallum végpontja előtt kell lennie egy F_1 pontnak, és ez akkor van benne a lehető legtöbb további intervallumban, ha $T_1 - 1$-hez tesszük. Megint nyilvánvalónak látszik, hogy az F_1 érték a „lehető legjobb" a többi lehetséges közül. Próbáljunk meg ismét egy indirekt bizonyítást. Tegyük föl, hogy van egy optimális, a mohó algoritmussal kapott K számú pontnál kevesebb pontot tartalmazó megoldás. Rendezzük most is az intervallumokat a végpontjaik szerint növekvően, és tegyük ezt a nem mohó megoldásban szereplő K' számú F'_j egésszel (a mohó megoldás F_j értékei eleve növekvő sorrendben vannak). Nézzük az azonos indexű egészek egymáshoz viszonyított helyzetét. F_1-nél nem lehet nagyobb F'_1, mivel ekkor a nem mohó megoldásban a rendezett sorrendben első intervallum nem tartalmazná egyik F'_j egészet sem, hiszen a legkisebbet sem tartalmazza. Így biztosan teljesül, hogy $F'_1 \leq F_1$. Így a mohó megoldás első pontja biztosan legalább annyi intervallumban szerepel, mint a nem mohó megoldás első pontja. Ebből következik, hogy a két pont által tartalmazott intervallumok elhagyása után a mohó megoldás legalább azokat az intervallumokat (vagy többet) lefed, amelyeket a másik megoldás. Tehát a nem mohó megoldásban is biztosan szerepel a mohó megoldás második pontját meghatározó intervallum. Ekkor az előbbi gondolatmenettel tudjuk igazolni, hogy $F'_2 \leq F_2$, és indukcióval a többi értékre is. Ezzel azonban ellentmondást kapunk, hiszen így a mohó megoldás és a másik, optimális megoldás azonos számú pontot tartalmaz. Tehát helytelen az a föltevés, hogy a mohó megoldás nem optimális.

A címletezéshez képest talán jobban látszik, hogy az intervallumok elhelyezkedésétől függően több optimális megoldás is van, és a mohó algoritmus ezek közül ad egyet.

Az elmúlt évek OKTV és Nemes Tihamér programozási versenyfeladatai között sok olyan problémát találunk, amelyek mohó módszer segítségével oldhatók meg. A http://mester.inf.elte.hu weboldalon a Mohó algoritmusok témakörben összegyűjtve is megtaláljuk őket. Gyakorlásként oldjuk meg a következő feladatot (a feladat pontos szövege

a http://nemes.inf.elte.hu oldalon érhető el.)

Informatika OKTV 2013/14. Második forduló – Gépek

Egy vállalkozó a következő N napra megrendeléseket fogad. Minden munkát egy nap alatt tud elvégezni, amihez egy gépet használ. M megrendelés érkezett, minden megrendelésben szerepel, hogy az igényelt munkát milyen határidőig kell elvégezni. A vállalkozónak ki kell számítani, hogy legkevesebb hány gépre van szükség, hogy minden igényelt munkát határidőre el tudjon végezni.

Készítsünk programot, amely kiszámítja, hogy legkevesebb hány gép kell ahhoz, hogy minden megrendelt munkát határidőre el tudjon végezni a vállalkozó! A program adja is meg, hogy ez egyes megrendelést melyik napon, melyik gépen végezze el a vállalkozó!

A program bemenete a napok N és a munkák M száma, valamint a munkák $1 \leq h_i \leq N$ határideje ($1 \leq i \leq M$). A program kimenete a szükséges gépek G száma, és minden munkára egy (n_i, g_i) számpár, amely megadja, hogy az i-edik munkát hányadik napon és hányas sorszámú gépen végezzük el.

Mivel mohó stratégiát szeretnénk alkalmazni, ezért próbáljuk ismét valamilyen szempont szerint sorrendbe állítani a lehetőségeket. A legegyszerűbb sorrend a munkák határideje alapján adódik, tehát rendezzünk a határidők szerint növekvő sorrendbe. A sorrendben legelső munkához biztosan szükséges egy gép. De melyik napon dolgozzon ez a gép? Olyan napot kell választani, amely a legkevésbé növeli a szükséges gépek számát. Mivel a munkák közül az első ér véget legelőször, vagyis a többi munka határideje az ő határidejénél nem kisebb, ezért az első munkát végezzük az első napon. Így marad a legtöbb lehetőség arra, hogy ez a gép egy másik napon egy határidős munkát befejezzen. Járjunk el hasonlóan a következő határidős munkával: ha lehet, akkor ugyanennek a gépnek a következő szabad napjára tegyük, hiszen minél kevesebb gépet szeretnénk, ugyanakkor minél több más határidős munkára akarunk szabad napot hagyni. Ez egészen addig folytatható, amíg el nem érünk egy olyan h_j határidejű munkához, amely határidejekor már minden napon dolgozik az első gép. Ekkor mindenképp szükséges egy új gépet beüzemelni. Mivel ezt a gépet is a lehető legjobban szeretnénk kihasználni, ezért ezt a munkát a második gép első munkanapjára időzítjük. Folytatassuk a következő munkával, de most már két géppel. Ha az első fölszabadul ennek a munkának a határidejéig, akkor az első gép első következő szabad napján dolgozzon ezen a munkán, ha pedig csak a második gép szabad, akkor az végezze a feladatot annak a következő szabad napján. Ha egyik gép sem szabad, akkor egy harmadik gépre lesz szükség, amelynél szintén az első munkanapon dolgozzon először.

A fentiek alapján megadható egy mohó algoritmus, amely meghatározza a szükséges gépek számát, és el is készít egy beosztást. Nyilván más optimális beosztás is lehetséges, de úgy gondoljuk, hogy a mohó algoritmussal készített beosztással a munkákat határidőre végzi el a vállalkozó, ugyanakkor a lehető legkevesebb gépet használja. Azt természetesen most is igazolni kellene, hogy megoldásunk egy optimális megoldás. Érdemes volna most is indirekt bizonyítást választani, és megmutatni, hogy kevesebb számú géppel nem lehet megoldani a problémát. Akinek kedve van hozzá, végezze el a bizonyítást.

Vizsgáljuk meg először azt a kérdést, hogy mennyi gépre van szükség. Mikor kell az összes gépet használnunk? Általában, meg tudnánk-e mondani, hogy a t-edik időpontban hány gépre van szükség? Igen, ha összeszámoljuk az összes olyan munkát, amelyet a t időpontig be kell befejezni, és elosztjuk a t-vel, a napok számával, akkor megkapjuk, hogy átlagosan hány gép kell. Ha az értéket fölfelé kerekítjük, akkor megkapjuk, hogy legalább hány gép kell. Ezek szerint ha minden $1 \leq t \leq N$ időpontra meghatározzuk a gépek számát, akkor azok maximuma a keresett G érték. Ehhez nem szükséges tudnunk, hogy melyik munka mikor ér véget, csak azt, hogy hány munka határideje esik a t-edik napra. Vegyünk föl tehát egy N méretű hm tömböt, amelynek $hm[t]$ eleme megadja a t időpontban befejezendő munkák számát.

```
Határidők-összegzése eljárás(h, M, N)
    hm[] := 0                 // hm minden értéke kezdetben zérus
    Ciklus i := 1-től M-ig    // minden munka határidejét megnézzük
        t := h[i]             // a h[i] munka határideje t
        hm[t] := hm[t] + 1    // a t időpontban befejeződő munkák megszámolása
    Ciklus vége
Határidők-összegzése eljárás vége
```

A hm tömb kiszámítása után könnyen megválaszolható, hogy mennyi gép kell. Végigmegyünk sorban a hm értékeken, és megnézzük, hogy az addig lejárt határidőkhöz hány gép kell, ha feltételezzük, hogy minden gép az első naptól üzemel.

```
Szükséges-gépek-száma eljárás(hm, N)
    msz := 0                      // az eddig befejeződött munkák száma
    G := 0                        // az eddig szükséges gépek száma
    Ciklus t := 1-től N-ig        // az összes napot megvizsgáljuk
        msz := msz + hm[t]        // az eddigekhez hozzávesszük a most befejeződő munkákat
        gsz := ⌈msz/t⌉            // ennyi gép kell naponta a t-edik napig
        Ha gsz > G akkor G := gsz // ha ez az új maximum
    Ciklus vége
Szükséges-gépek-száma eljárás vége
```

Figyeljük meg, hogy G kiszámításához nem is kellett rendezni a h bemeneti adatsorozatot, tehát ha csak a gépek

maximális számának meghatározása lenne a cél, akkor egy $N+M$-mel arányos lépésszámú algoritmussal kész lennénk. Ez alapvetően annak köszönhető, hogy a feladatban szereplő intervallumok bal oldala azonos. Az olvasóra bízzuk a folytatást, tehát annak az eljárásnak a kitalálását, amely megadja, hogy melyik munka melyik gépen történjen. Az egyik lehetséges út az adatok rendezése, és a munkák fenti leírás szerinti beosztása. A teljes megoldás lépésszáma így $M \cdot \log M + N$. De ha valaki még ennél is hatékonyabb algoritmust szeretne, akkor próbálja meg az adatok rendezése nélkül megadni, hogy melyik gép melyik munkán dolgozzon, annak ismeretében, hogy a szükséges gépek száma már ismert.