Kombinatorikus feladatok

Ládák: Egy vállalat udvarán egyetlen sorban vannak az elszállításra várakozó üres ládák. Három különböző típusú láda van, jelölje ezeket A, B és C. Minden láda a felső oldalán nyitott kocka alakú. Az A-típusú láda a legnagyobb és a C-típusú a legkisebb. Tehát minden C-típusú láda belerakható A-típusú és B-típusú ládába, minden B-típusú belerakható A-típusúba és A-típusúba belerakható B-típusú, majd ebbe egy C-típusú. Az a cél, hogy a ládákat úgy pakoljuk össze, hogy a lehető legkevesebb összepakolt láda legyen. A pakolást olyan robot végzi, amely a ládasor felett tud mozogni mindkét irányban, de ládát csak balról jobbra mozogva tud szállítani. Írjon programot, amely megadja, hogy legkevesebb hány ládába lehet összepakolni a ládasort!

Ha csak kétféle betű fordul elő a sorozatban, A és B, akkor a megoldás egyszerű. Legyen x az aktuálisan különálló B ládák száma, y az A ládáké.

Balról-jobbra haladva, ha A betű esetén maradt el B (x>0), amit nem raktunk bele A-ba, akkor azt rakjuk bele az aktuális A-ba. Azaz $x := x - 1$, és $y := y + 1$.

Ha az aktuális betű B, akkor növeljük az y számláló értékét, $y := y + 1$.

A megoldás $x + y$.

Ha három féle láda van. Legyen x,y,z,w az aktuálisan C-t, B-t, B-ben C-t, és tetszőleges tartalmú *A* ládák száma. Ekkor balról jobbra haladva.

Ha az aktuális láda A, akkor növeljük w-t. Utána x és y pozitív, akkor mindkettőt csökkentjük 1-el (pl B,C,A sorrend). Egyébként ha $z > 0$ csökkentjük 1-el. Egyébként az x és y közül azt csökkentjük, amelyik pozitív.

Ha az aktuális láda *B*, akkor ha $x > 0$, akkor x-et csökkentjük 1-el, z-t növeljük 1-el. Egyébként y-ot növeljük 1-el.

Ha az aktuális láda *C*, akkor x-et növeljük 1-el.

Permutációk Az első n számból képezhető permutációkat a lexikografikus sorrend szerint sorba rendezzük. Egy adott permutációra adjuk meg a rákövetkezőt.

Példa:

- Rakov(34125)=34152
- Rakov(12543)=13245
- Rakov(14532)=15234
- Rakov(54321)=-

1. Meg kell találnunk azt a legutolsó elemet, amit megnövelhetünk. Ez az utolsó olyan elem lesz, ami mögött van nála nagyobb.
2. Az elem helyére a mögötte levő elemek közül a legkisebb nála magyobbat írjuk.
3. A maradék elemekből a lehető legkisebb számot képezzük, és ezzel zárjuk a permutációt.

A processzorgyártó cégek megállapodtak abban, hogy milyen rendszert alkalmaznak az általuk gyártott processzorok egyedi azonosítására. Minden cég kap egy betűkészletet, és ezekből kell az azonosító kódot képeznie, úgy, hogy minden betű meghatározott számszor szerepeljen az azonosítóban. Például egy cég azt kapta, hogy minden azonosítója pontosan 3 db a betűt, 2 db b betűt és 1 db c betűt tartalmazhat. Készítsen olyan programot, amely adott kódra meghatározza a lexikografikus (ábécé szerinti) sorrendben rákövetkező szabályos kódot, ha van rákövetkező!

Példa:

- Rakov(aaabbc)=aaabcb
- Rakov(bbacaa)=bbcaaa
- Rakov(acbbaa)=baaabc
- Rakov(cbbaaa)=-

A megoldás ugyanazokból a lépésekből áll.

Interaktív feladatok

Bináris keresés: Anna gondolt egy számot 1 és $2^n - 1$ között, és Beának ki kell találni. Minden tippjére megkapja a választ, eltalálta -e a számot, és amennyiben nem talált, megtudja, hogy a gondolt szám nagyobb vagy kisebb a tippjénél.

Kezdetben $a = 0$, $f = 2^n$ a tipp pedig $g = 2^{n-1}$. Utána amíg nem talált:

- Ha a gondolt szám nagyobb, akkor
  $a := g$, $f := f$, $g := (a+f)/2$
- Ha a gondolt szám kisebb, akkor
  $a := a$, $f := g$, $g := (a+f)/2$

Medián keresése Adott n tárgy, melyeket 1-től n-ig számozunk, ahol n páratlan. Minden tárgy különböző súlyú (természetes számok), de magukat a súlyokat nem ismerjük. Minden y súlyra igaz, hogy $1 \leq y \leq n$. Mediánnak nevezzük azt a tárgyat, amelyiknél ugyanannyi könnyebb, mint nehezebb tárgy van. Írjunk programot, amely meghatározza a mediánt! A tárgyak súlyát egy olyan eszközzel hasonlíthatjuk össze, amely három tárgy közül megadja a mediánt.

Hagymahámozó algoritmus: Határozzuk meg a legnagyobb és legkisebb elemeket, hagyjuk el őket a maradék halmaz mediánja ugyanaz, mint az eredeti halmazé.

Egy elem akkor és csak akkor szélső elem, ha nem mediánja egyetlen hármasnak sem. Így minden lekérdezéssel ki tudunk zárni egy elemet, tehát $n-2$ kérdéssel meghatározhatjuk a szélső elemeket. Következésképpen a halmaz mediánja $(n-2)+(n-4)+(n-6)+\cdots+1 = O(n^2)$ kérdéssel meghatározható a hagymahámozó algoritmussal.

Számjáték: Egy játéktábla pozitív egész számok sorozata, ami 2n számot tartalmaz. Két játékos felváltva lép, egy lépésnem a játékos a számsorozat bal vagy jobb végéről elvesz egy számot. A játék akkor ér véget, ha a számok elfogytak. Az a játékos nyer, akinél az elvett számok összege több. Adjunk egy eljárást, amivel az első játékos legalább döntetlen elér. A második játékos lépéseit egy már adott számítógépes program szolgáltatja.

Megoldás Az első játékos kényszerítheti a másodikat, hogy csak a páratlan vagy csak a páros indexű számokat vegye el, így könnyen elérheti, hoyg ne veszítsen.

Maxszámjáték A játék csak a céljaiban különbözik az előző játéktól. Most a játékos célja a lehető legnagyobb érték összegyűjtése.

Az optimális stratégia dinamikus programozással meghatározható. Legyen minden $i \leq j$-re $OPT(i,j)$ az $a_i, \ldots, a_j$ számsorozatból az első játékos által összegyűjthető összeg.

- Ha $i = j$, akkor ez az érték 0 mivel az utolsó elemet a második játékos veszi el.
- Ha $i+j$ páratlan, akkor az első játékos jön, és a két szélső elem közül a neki jobbat választja, tehát ekkor $OPT(i,j) = \max\{a_i + OPT(i+1,j), a_j + OPT(i,j-1)\}$.
- Ha $i+j$ páros, akkor a második játékos jön, ő azt választja, ami az elsőnek rosszabb, azaz ekkor az érték $OPT(i,j) = \min\{OPT(i+1,j), OPT(i,j-1)\}$.

Ezen összefüggésekkel dinamikus programozással minden állásra kiszámítható a maximálisan nyerhető összeg és az ehhez szükséges lépés.

Visszalépéses algoritmus

8 királynő problémá Helyezzünk el az $n \times n$-es sakktáblán n királynőt, hogy egyik se üsse a másikat!

Megoldástér: n darab mező, amiket a koordinátákkal adhatunk meg: $\{(x_1,y_1),\ldots,(x_n,y_n)\}$, $1 \leq x_i, y_i \leq n$.
A királynők akkor és csak akkor nem ütik egymást, ha

- $x_i \neq x_j$, ha $i \neq j$
- $y_i \neq y_j$, ha $i \neq j$
- $x_i + y_i \neq x_j + y_j$, ha $i \neq j$
- $x_i - y_i \neq x_j - y_j$, ha $i \neq j$

Az első feltétel alapján feltehetjük, hogy $y_i = i$, így a királynők elhelyezését egy $(x_1, \ldots, x_n)$ $1 \leq x_i \leq n$ vektorral írjuk le, ami annak az elhelyezkedésnek felel meg, hogy az i-edik sorban a királynő az x_j oszlopban van.

- $x_i \neq x_j$, ha $i \neq j$
- $x_i + i \neq x_j + j$, ha $i \neq j$
- $x_i - i \neq x_j - j$, ha $i \neq j$

Megoldástér ábrázolás

Ekkor a bejárandó megoldáskezdemények $(x_1, \ldots, x_k)$ vektorok $1 \leq x_i \leq n$, $k \leq n$. A megoldáskezdemények egy fában ábrázolhatóak, a fát leíró, a bejárásához használható függvények:

$EFiu(x_1, \ldots, x_k) = (x_1, \ldots, x_k, 1)$, ha k<n és Nil ha k=n.

$Testver(x_1, \ldots, x_k) = (x_1, \ldots, x_{k-1}, x_k + 1)$, ha $x_k < n$ és Nil egyébként.

$APA(x_1, \ldots, x_k) = (x_1, \ldots, x_{k-1})$, ha $k \geq 1$ és Nil egyébként.

Kimerítő keresés

A kimerítő keresés vagy nyers erő algoritmusa során az egész megoldástért bejárjuk, megvalósítható egy fabejárással.

Megoldás keresése visszalépéses algoritmussal (backtracking)

Az alapötlet az, hogy ha a fabejárás során, az olyan részfákat, amelyekben nincs lehetséges megoldás nem járjuk be, ha ilyen fához jutunk visszalépünk. Ennek megvalósításához szükség van egy függvényre, amely megadja, ha nem tartalmaz lehetséges megoldást a részfa. A LehetMego függvénytől ezt várjuk el, ha a függvény hamis értéket ad, a részfa nem tartalmaz lehetséges megoldást, igaz válasz esetén nem feltétlenül kell, hogy tartalmazzon megoldást a részfa.

A LehetMego függvény megadja, ha nem kiterjeszthető a megoldáskezdemény, mert két királynő már üti egymást.

Tehát $LehetMego(x_1, \ldots, x_k) = True$, akkor és csak akkor ha minden $i \neq j$ esetén teljesül, hogy $x_i \neq x_j$, $x_i + i \neq x_j + j$, $x_i - i \neq x_j - j$

Használunk még egy Megoldas függvényt, amely akkor és csak akkor ad igaz értéket, ha az adott pont lehetséges megoldás. Esetünkben $Megoldas(x_1, \ldots, x_k) = True$, akkor és csak akkor teljesül, ha $LehetMego(x_1, \ldots, x_k)$ és $k = n$.

A keretalgoritmus

A fenti függvények alapján az alábbi általános keretalgoritmust építjük fel. Egy adott probléma esetén meg kell adni a megoldástér fáját és a problémához tartozó függvényeket.

```
Visszalep(F)
  If F=Nil Then return //Üres fa
  If Megoldas(F)
     Then Print "F" // Megoldas kiiratasa
          return
      Else p:=F.Elsofiu
           While (p!=Nil) // a gyerekeken sorbamenve
                If LehetMego(p) Then Visszalep(p)
                p:=p.Testver
```

A fenti változat egyetlen megoldást keres meg, de a harmadik negyedik sor változtatásával könnyen átírható az összes megoldás megkeresésére. A kiiratás helyett a megoldást bele kell tenni egy halmazba, a negyedik "return" sort törölni. Az összes megoldás megkeresése felhasználható optimalizálási feladatok megoldására is.

Pénzváltási probléma

Az pénzváltási problémában adottak $p_1,\ldots,p_n$ pénzérmék és egy E összeg, a feladat kiválasztani az érmék egy S halmazát, amelyre teljesül, hogy $\sum_{i\in S} p_i = E$.

A megoldástér a pénzérmék összes lehetséges részhalmaza. Egy lehetséges reprezentáció egy n szintből álló teljes bináris fa, amely a halmaz bitvektorát tárolja. A részhalmazoknak a levelek felelnek meg. A levélhez tartozó halmazt a levélig a gyökérből vezető út alapján kapjuk meg. Ha az i-dik szinten balra lépünk az i-dik elemet nem választjuk be, ha jobbra, akkor az i-dik elemet beválasztjuk a halmazba.

Egy másik reprezentációval a megoldástér pontjait (a pénzek részhalmazát) a pénzek indexeinek az $S \subseteq \{1,\ldots,n\}$ halmazának $X = (i_1,\ldots,i_k)$ növekvő felsorolásáként reprezentáljuk.

Ekkor a bejáráshoz használt függvények

$EFiu(i_1,\ldots,i_k) = (i_1,\ldots,i_k,i_k+1)$, ha $i_k < n$ és Nil ha $i_k = n$.

$Testver(i_1,\ldots,i_{k-1},i_k) = (i_1,\ldots,i_{k-1},i_k+1)$, ha $i_k < n$ és Nil ha $i_k = n$.

$Apa(i_1,\ldots,i_{k-1},i_k) = (i_1,\ldots,i_{k-1})$, ha $k \geq 1$ és Nil egyébként.

A LehetMego és Megoldás függvényeket a következőképpen kaphatjuk meg.

A LehetMego függvény megadja, ha az eddig meghozott döntéseket nem lehet kiegészíteni egy felbontássá, vagy azért mert már nagyobb összeget választottunk, vagy azért mert minden további elemet kiválasztva sem érhetjük el E-t. Tehát $LehetMego(i_1,\ldots,i_k) = True$ akkor és csak akkor, ha

$$\sum_{j=1}^{k} p_{i_j} \leq E \wedge \sum_{j=1}^{k} p_{i_j} + \sum_{j=i_k+1}^{n} p_j \geq E.$$

Továbbá $Megoldas(i_1,\ldots,i_k) = True$ akkor és csak akkor, ha $\sum_{j=1}^{k} p_{i_j} = E$

Adatszerkezet meghatározása Adott egy adatszerkezet, amelyről nem tudjuk, hogy lista vagy nyalóka (egy lista végére illesztve egy körlánc). Van két bélyeg. Az adatszerkezettel a következő lépéseket tehetjük.

- egy bélyeget a helyéről felvehetünk

-egy bélyeget a helyéről a rákövetkező helyre továbbmozdíthatunk

-egy bélyeg helyére mellé a másikat is odatehetjük

-a legelső elemre bélyeget tehetünk.

Határozzuk meg a méretben lineáris időben, melyik adatszerkezetről van szó!

Megoldás A nyalóka adatszerkezetet úgy ismerhetjük fel, ha találunk kört. Kört pedig úgy találhatunk, hogy egy bélyeg helyétől elindulva a másik bélyeggel mindig továbblépve újra megtaláljuk a bélyeget. Két módszer lehetséges:

Probálgatós Minden k-ra megnézzük, hogy van -e a kiindulási ponttól 2^k lépésre levő pontot tartalmazó legfeljebb 2^k hosszú kör. Ezt úgy tehetjük meg, hogy a 2^k lépésnyire levő bélyeg helyéről elindulunk a másik bélyeggel, teszünk 2^k lépést. Ha újra látjuk az első bélyeget van kör, ha nem, akkor áthelyezzük az első bélyeget is a második mellé, és rátérünk a 2^{k+1} hosszú kör ellenőrzésére.

Ötletes Elinditjuk a két bélyegzőt, az egyiket kétszer olyan gyorsan. (Időegységenként 2-t lép, mig a lassabb csak egyet). Ha nyalóka adatszerkezet van, lineáris időben lekörözi a gyors a lassút, és megtaláljuk, hogy van kör, egyébként pedig nyilvánvalóan lineáris időben eljutunk a lista végére.