Basketball Exercise

Innen: Algowiki

Tartalomjegyzék

Feladat

Egy kosárlabda-mérkőzéshez szeretnénk egy ütőképes csapatot összeállítani. Ehhez a játékosok két n hosszú sorba állnak fel egymás mellé. A választás során a sorokon elölről végig haladva választhatjuk vagy a bal, vagy a jobb oldali játékost, vagy megtehetjük hogy egyikőjüket sem választjuk. Arra azonban ügyelnünk kell, hogy két közvetlenül egymás után választott játékos nem kerülhet ki ugyanabból a sorból. Egy csapat tetszőleges számú játékosból állhat.

Minden játékosnak ismerjük a magasságát, a biztos győzelem érdekében pedig szeretnénk úgy összeválogatni a csapatunkat, hogy a csapattagok magasságainak összege a lehető legnagyobb legyen. Mekkora lehet a legnagyobb összeg, ha kezdetben bármelyik sorból választhatunk?

Bemenet:

Az első sor egy n számot tartalmaz, a sorok hosszait. $(1 \leq n \leq 10^5)$

A második sor a bal oldali sor n db játékosának magasságait tartalmazza szóközökkel elválasztva.
$(1 \leq h_{1,i} \leq 10^9)$

A harmadik sor a jobb oldali sor n db játékosának magasságait tartalmazza szóközökkel elválasztva.
$(1 \leq h_{2,i} \leq 10^9)$

Kimenet:

Egyetlen szám, a lehető legnagyobb magasságösszeg.

Eredeti feladat:

Codeforces - Basketball Exercise (https://codeforces.com/problemset/problem/1195/C)

Megoldási ötletek

Hibás megoldási ötletek

Tipikusan rossz eredményre vezet általában a mohó megközelítés, vagyis az az elv, hogy minél több embert választunk, annál jobb, és minden választásnál vegyük a magasabbikat. Ez az algoritmus az alábbi bemenetre 15-öt adna (*bal, jobb, bal*):

```
3
4 2 1
2 10 96
```

pedig látható, hogy a *bal, senki, jobb* választással a legjobb érték 100.

Egy ennél kifinomultabb, de szintén legtöbbször helytelen mohó algoritmus lehet ennek az előzőnek egy továbbgondolása: minden i-edik választásnál megvizsgáljuk, hogy az előzővel (i-1 -edik) ellentétes sorból való választással jobban járnánk-e, mintha inkább az azonos sorbeli játékost választanánk, és az előzőt kihagynánk.

Ez a módszer már 100-at adna a fenti példára, de 13-at az alábbira (*bal, jobb, senki, bal*):

```
4
2 5 1 7
1 4 4 3
```

ahol is pedig 17 a helyes válasz (*jobb, bal, jobb, bal*).

Helyes, de lassú megoldás

Brute-force: valamiféle rekurzióval számoljuk ki, mi lenne mindhárom választási lehetőség eredménye, és ennek adjuk vissza a maximumát. Ezzel a feladatot kisebb részfeladatok eredményeitől tennénk függővé, ami alapból nem rossz gondolat (lásd megoldás), de sok esetet feleslegesen sokszor újra kéne számolni hozzá, ami polinomiális időben nem futna le semmiképpen sem.

Segítség, gondolatébresztő

Vezessük vissza az alapfeladatot részfeladatok megoldásaira, de vegyük észre, hogy egy részeredményt általában többször is felhasználunk. Nem lehetne-e ezeket az eredményeket valahogy egyszer kiszámolni, és később csak hivatkozni rájuk? Hogy épülnek egymásra a részeredmények? Tudunk-e esetleg találni valamiféle sorrendiséget a kiszámításuk között?

Megoldás

Fontos gondolatok

A részfeladatokra való visszavezetésről már eszünkbe juthat a dinamikus programozás. Az alábbiakban a "klasszikus" táblázatos megoldást részletezzük, bár érdemes megemlíteni, hogy a feladat ugyanolyan helyesen megoldható (bár a rekurzív hívások miatt valamivel lassabb) a rekurzív megközelítéssel és memoizációval is (vagyis a már kiszámolt részeredmények cache-elésével).

Részletes megoldás

A megoldás során kiszámított részeredményeket érdemes egy három soros, n+1 oszlopos táblázatban rögzíteni. Az első sor i-edik oszlopába (jelöljük N_i -vel) írjuk be azt a maximális összeget, amit akkor kaphatunk, ha az i-edik választásnál egyik sorból sem választunk senkit. Hasonlóképp a második sor i-edik oszlopába (L_i) kerüljön az a maximális összeg, amikor az i-edik választásnál a bal oldali sorból választunk játékost, a harmadik sor i-edik oszlopába (R_i) pedig ha a jobb oldaliból. A táblázatot kezdjük el hátulról kitölteni, az n+1 -edik oszlop elemei jelentsék a kezdőállapotot, amikor még egy embert sem válogattunk a csapatunkba, vagyis ekkor még az összmagasság mindenhol 0.

Ekkor észrevehetjük, hogy a táblázat i-edik oszlopának értékei mind csak az i+1 -edik oszlop értékeitől függenek, mégpedig a következőképpen:

a) $N_i = max(L_{i+1}, R_{i+1})$

Vagyis ha az i-edik pozíciónál nem választunk senkit, akkor a legnagyobb összeg az eddigi legjobb csapat magasságösszege.

Megjegyzés: Itt egy fontos észrevétel, hogy az eddigi legjobb csapatból fölösleges vizsgálnunk azt az esetet, amikor a következő körben sem választunk senkit, ugyanis ez az eset biztosan nem adhat maximális összeget. Ezt belátni egyszerű: tegyük fel, hogy a maximális összegű megoldásban létezik olyan i, hogy $h_{1,i}, h_{2,i}, h_{1,i+1}, h_{2,i+1}$ *közül egyet sem válogattunk be. Mivel azonban minden játékos magassága pozitív, négyük közül a kiválasztási szabályoknak eleget téve és az eddig kiválasztott játékosokat meg nem változtatva kiválasztható legalább még egy, akivel együtt az összeg még nagyobb lenne. Ez pedig ellentmond annak a feltételezésnek, hogy ez a választás eleve maximális összegű volt.*

b) $L_i = h_{1,i} + max(R_{i+1}, N_{i+1})$

Vagyis ha az i-dik pozíciónál a bal oldali játékost választjuk, akkor a maximális összeg az ő magassága + az eddigi legjobb csapat összmagassága a választási szabályok megtartásával (vagyis ha a következő körben nem választhatunk újra balról).

c) $R_i = h_{2,i} + max(L_{i+1}, N_{i+1})$

Hasonlóképp az előzőhöz.

A táblázat kitöltése után a végeredmény $max(L_1, R_1)$ lesz, hiszen kezdetben bármelyik oldalról választhattunk.

Megjegyzés: N_1 *vizsgálata itt az a) pontban leírt gondolatmenet alapján hasonlóképp igazolhatóan fölösleges (indirekt feltevéssel).*

Komplexitás

Ez az elv egy egyszerű rekurzív formulára támaszkodik, azonban erősen kihasználja a hívások során kiszámolt részeredmények közti sorrendiséget. A megfelelő reprezentáció megválasztásával elértük, hogy egyetlen ciklussal $O(n)$ futásidő alatt kiszámoljuk a végeredményt. A részeredmények tárolására alkalmas táblázat felvehető $O(n)$ tárhelyen, azonban az implementációban kihasználjuk, hogy mindig kizárólag csak a következő oszlop elemeire kell hivatkoznunk, vagyis elegendő N_{i+1}, L_{i+1} és R_{i+1} eltárolása. Ezáltal a tényleges tárhelyigény levihető $O(1)$-re!

Implementáció [becsuk]

```
1 #include <iostream>
2 using namespace std;
3
```

```cpp

// Az osszmagasssag nagy lehet, hasznaljunk minel nagyobb egesz tipust
typedef unsigned long long ulong;


int main()
{
    int N;
    cin >> N;

    // Beolvassuk a bal oldali sor magassagait
    int* L = new int[N];
    for (int l = 0; l < N; ++l)
        cin >> L[l];

    // Beolvassuk a jobb oldali sor magassagait
    int* R = new int[N];
    for (int r = 0; r < N; ++r)
        cin >> R[r];

    // Hatulrol indulva: ha senkit nem valasztunk, az osszmagassag minden esetben 0
    ulong NMax = 0;
    ulong LMax = 0;
    ulong RMax = 0;
    // Visszafele haladva valasszunk jatekosokat
    for (int i = N - 1; i >= 0; --i)
    {
        // Az i-edik valasztas utani legnagyobb összmagassag
        ulong n = LMax > RMax ? LMax : RMax;// ha senkit nem valasztunk
        ulong l = L[i] + (RMax > NMax ? RMax : NMax);// ha a bal oldali jatekost valasztjuk
        ulong r = R[i] + (LMax > NMax ? LMax : NMax);// ha a jobb oldali jatekost valasztjuk
        // A kovetkezo korben mar ezekre az ertekekre hivatkozunk majd, mint eddigi legnagyobb osszegek
        NMax = n;
        LMax = l;
        RMax = r;
    }

    // A kezdeti valasztas mindegy (vagy bal, vagy jobb): amelyik nagyobb, az a valasz
    cout << (LMax > RMax ? LMax : RMax) << endl;
    return 0;
}
```

A lap eredeti címe: „https://algowiki.miraheze.org/w/index.php?title=Basketball_Exercise&oldid=1341”