Predaj starého vína

Jedna nemenovaná francúzska grófska rodina sa v poslednej dobe dostala do finančných ťažkostí. Na ich zmiernenie sa gróf rozhodol urobiť drastické opatrenie – začať predávať nejaké fľaše vín z rodinného vinohradu. Fľaše, ktoré plánuje predať, sú momentálne uložené v jednom rade na polici vo vinnej pivnici. Uskladňovanie kvalitného vína je veľmi chúlostivá vec. Napríklad by nemali byť žiadne medzery v rade vín – toto by porušilo prúdenie vzduchu v pivnici, čo má vplyv na zmeny teploty vzduchu. Preto fľaše môžu byť odoberané iba z krajov radu. Víno je pre grófa veľmi cenné, a preto sa rozhodol, že jeho rodina predá iba jednu fľašu vína ročne. Fľaše na jeho polici sú rôznych druhov. Ak by predal všetky tento rok, cena i-tej fľaše by bola w_i. Naštastie pre grófa, kvalitné vína získavajú na cene s rastúcim vekom. Pre jednoduchosť predpokladajme, že cena i-tej fľaše na polici je v y-tom roku rovná $w_i \cdot y$.

Úloha

Pre dané začiatočné ceny vín a ich usporiadanie na polici nájdite optimálne poradie, v ktorom ich treba predávať a spočítajte celkovú sumu peňazí, ktoré vie získať gróf pri dodržaní tohto poradia.

Vstup

Prvý riadok vstupu obsahuje jedno celé číslo N ($1 \leq N \leq 2000$) – počet fliaš vína na polici. Nasleduje N riadkov, i-ty riadok obsahuje jedno celé číslo w_i ($1 \leq w_i \leq 1000$) – začiatočnú cenu i-tej fľaše zľava.

Výstup

Výstupom je jediný riadok s jedným celým číslom – maximálnou možnou sumou peňazí, čo vie gróf získať na predaji vín.

Príklad

Vstup

N=5

1 3 1 5 2

Výstup

43

(Fľaše treba predať v poradí: #1, #5, #2, #3, #4. Ich ceny budú 1, 4, 9, 4 a 25, čo je dokopy 43.)

Greedy algoritmus

- Pozorovanie: Čím drahšie víno, tým viac narastie jeho cena každý rok.
- Chceme teda lacné vína predať ako prvé a drahé až nakoniec.
- Keby sme mohli predávať vína v ľubovoľnom poradí, zjavne by sme ich predali v poradí od najlacnejšieho po najdrahšie.

Greedy algoritmus

- Bohužiaľ, nemôžeme predávať vína v ľubovoľnom poradí, a teda musíme náš algoritmus nejako prispôsobiť.
- Držme sa uvedenej myšlienky, a navrhnime takýto algoritmus:
- V každom (k)roku vezmeme tú z dvoch krajných fliaš, ktorá je lacnejšia.

Greedy algoritmus

- Na vzorovom vstupe:
- 1 3 1 5 2
- 3 1 5 2
- 3 1 5
- 1 5
- 5

- Za fľaše postupne dostaneme:
- 1 . 1
- + 2 . 2
- + 3 . 3
- + 4 . 1
- + 5 . 5
- = 43, správne!

Greedy algoritmus

- Bude to fungovať pre každý vstup?

Greedy algoritmus

- Bude to fungovať pre každý vstup?
- Nie!
- Protipríklad: 10 1 1 1 9 9 9
- Greedy: 1.9 + 2.9 + 3.9 + 4.1 + 5.1 + 6.1 + 7.10 = 9+18+27+4+5+6+70 = 139
- Optimálne: 1.10 + 2.1 + 3.1 + 4.1 + 5.9 + 6.9 +7.9 = 10+2+3+4+45+54+63 = 181

Dynamické programovanie

- Našli sme síce rýchle, no len v obmedzených prípadoch funkčné riešenie.
- Poďme nájsť riešenie, ktoré funguje vždy!
- Vyskúšame dynamické programovanie – pokúsime sa odvodiť riešenie problému od riešenia nejakého menšieho podproblému.

Dynamické programovanie

- Formulácia podproblému je pomerne jednoduchá:
- Podproblémom je ten istý problém s nejakou podmnožinou pôvodnej množiny fliaš a so začiatočným rokom r. (Teda ceny vín budú hneď $r.w_i$.)

Dynamické programovanie

- Pozorovanie: Keďže fľaše nikdy nemôžeme brať z iného miesta ako z kraja, nemôže vzniknúť ľubovoľný takto definovaný podproblém.
- Napríklad z problému so zadaním 1 2 3 4 5 sa vieme dostať k podproblému 2 3 4 alebo 3 4 5, ale nikdy nie napríklad k 1 3 4.
- Takisto, k ľubovoľnému podproblému sa nevieme dostať v ľubovoľnom roku, rok musí byť vždy (N – počet ostávajúcich fliaš + 1).

Dynamické programovanie

- Vďaka tomu vieme podproblém jednoznačne určiť aj bez toho, aby sme museli vymenovať všetky ostávajúce fľaše vína a definovať rok.
- Ak zadáme rok r a číslo prvej fľaše a, podproblém je jednoznačne určený – počet fliaš musí byť N–r+1, teda množina ostávajúcich fliaš bude
  a, a+1, ..., a+(n–r).

Dynamické programovanie

- Teda nech naj[r][a] je najväčšia cena, ktorú vieme získať za fľaše začínajúce a-tou fľašou, ktoré začneme predávať v roku r.
- Skúsme teda nájsť „magický vzorec“ pre naj[r][a].

Dynamické programovanie

- Ak nám ostáva len jedna fľaša (v N-tom roku), potom nemáme na výber, len predať túto fľašu, teda
- ak r=n, potom naj[r][a] = n*w[a]
- Ak nám ostáva viac fliaš (r<n), potom musíme zistiť (vyskúšať), či je výhodnejšie predať najprv ľavú alebo pravú.

Dynamické programovanie

- Ak predáme najprv ľavú, dostaneme
- r*w[a] + naj[r+1][a+1]
- (cenu za túto fľašu v roku r + najlepšiu možnú cenu za fľaše začínajúce a+1-vou fľašou v r+1-vom roku)
- Ak predáme najprv pravú (tj. a+(n-r)-tú) fľašu, dostaneme
- r*w[a+(n-r)] + naj[r+1][a]

Dynamické programovanie

- Hodnota naj[r][a] bude, samozrejme, lepšia z týchto dvoch možností, teda
- ak r<n, potom
- naj[r][a] = max(r*w[a] + naj[r+1][a+1], r*w[a+(n-r)] + naj[r+1][a])

Dynamické programovanie

- Hodnota, ktorú chceme vyrátať, je najlepšia cena za fľaše začínajúce prvou fľašou, ktoré začneme predávať v prvom roku, teda naj[1][1]. (Počet fliaš je a+(N-r) = 1+N-1 = N)
- Už potrebujeme len určiť, v akom poradí jednotlivé hodnoty vypočítame, aby sme vždy mali hodnoty, na ktorých aktuálna hodnota závisí, už vypočítané, a môžeme sa pustiť do programovania.

Dynamické programovanie

- Aktuálna hodnota závisí na hodnotách naj[r+1][a] a naj[r+1][a+1], teda budeme musieť počítať „odzadu“, začať naj[n][...] a skončiť pri naj[1][1].
- Hodnoty naj[n][...] nezávisia na žiadnych ďalších hodnotách, teda ich nie je problém vypočítať ako prvé.

Dynamické programovanie

```
import java.util.Scanner;

public class PredajStarehoVina {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();

        int[] w = new int[n+1];
        for (int i = 1; i <= n; ++i) {
            w[i] = s.nextInt();
        }

        int[][] naj = new int[n+1][n+1];

        // pouzijeme vzorec pre naj[n][...] (na nicom nezavisiace)
        for (int a = 1; a <= n; ++a) {
            naj[n][a] = n*w[a];
        }

        // postupne vyratame naj[n-1][...], naj[n-2][...], ..., naj[1][1]
        for (int r = n-1; r >= 1; --r) {
            for (int a = 1; a <= r; ++a) {  // a moze byt najviac r, inak by a+(n-r) > n
                naj[r][a] = Math.max(r*w[a]       + naj[r+1][a+1],
                                     r*w[a+(n-r)] + naj[r+1][a]);
            }
        }

        System.out.println(naj[1][1]);
    }
}
```

Dynamické programovanie

- Časová zložitosť: $O(N^2)$
- (dominantné sú dva vnorené cykly, príkaz vovnútri sa vykoná (n-1 + n-2 + ... + 1)-krát)
- Pamäťová zložitosť: $O(N^2)$
- (ukladáme si medzivýsledky do poľa N.N)

Dynamické programovanie

- Na záver ešte jedno vylepšenie:
- Nepotrebujeme si pamätať všetkých N.N medzivýsledkov, keďže nás zaujíma len jeden, a na výpočet jedného „riadku“ potrebujeme vždy len jeden nasledujúci riadok.
- Použime teda namiesto naj[n+1][n+1] len pole naj[2][n+1], čo nám zlepší pamäťovú zložitosť na O(N).
- Ako to jednoducho implementovať?
- Iba doplníme zvyšok po delení dvoma!

Dynamické programovanie

```
import java.util.Scanner;

public class PredajStarehoVina2 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();

        int[] w = new int[n+1];
        for (int i = 1; i <= n; ++i) {
            w[i] = s.nextInt();
        }

        int[][] naj = new int[2][n+1];

        // pouzijeme vzorec pre naj[n][...] (na nicom nezavisiace)
        for (int a = 1; a <= n; ++a) {
            naj[n%2][a] = n*w[a];
        }

        // postupne vyratame naj[n-1][...], naj[n-2][...], ..., naj[1][1]
        for (int r = n-1; r >= 1; --r) {
            for (int a = 1; a <= r; ++a) {  // a moze byt najviac r, inak by a+(n-r) > n
                naj[r%2][a] = Math.max(r*w[a]       + naj[(r+1)%2][a+1],
                                       r*w[a+(n-r)] + naj[(r+1)%2][a]);
            }
        }

        System.out.println(naj[1][1]);  // 1%2 = 1
    }
}
```