A-III-1 Odpoveď

Keďže číslo 42 sa v celej postupnosti vyskytuje práve raz, začneme tým, že nájdeme jeho pozíciu p. Zaujímajú nás teraz len tie súvislé podpostupnosti, ktoré túto pozíciu obsahujú.

Riešenia hrubou silou (aspoň kubická časová zložitosť)

Prvým a najjednoduchším riešením je zobrať si každý možný začiatok z a koniec k také, že $z \leq p \leq k$ a $k - z$ je párne, a pre každú dvojicu overiť, či medián tohoto úseku je naozaj 42. Priamočiara forma overovania: overovaný úsek si skopírujeme do pomocného poľa, to usporiadame a pozrieme sa na prostredný prvok. Takéto riešenie má časovú zložitosť $\Theta(n^3 \log n)$, lebo máme $\Theta(n^2)$ úsekov a pre väčšinu z nich na triedenie potrebujeme $\Theta(n \log n)$ krokov.

Existujú aj rôzne (pomerne komplikované) algoritmy, pomocou ktorých vieme medián n-prvkovej postupnosti nájsť v čase $\Theta(n)$. Použitie takéhoto algoritmu namiesto triedenia vedie k časovej zložitosti $\Theta(n^3)$.

Tú istú časovú zložitosť však vieme dosiahnuť aj oveľa ľahšie. Stačí si uvedomiť, že nepotrebujeme *nájsť medián*, len *overiť*, či je 42 mediánom. Pri overovaní nás nezaujíma rozmiestnenie prvkov. Jediné, čo potrebujeme overiť, sú ich počty: počet prvkov menších ako 42 musí byť rovný počtu prvkov väčších ako 42. A toto vieme veľmi ľahko overiť v čase lineárnom od dĺžky testovaného úseku. Takto teda dostávame ľahšie riešenie s časovou zložitosťou $\Theta(n^3)$.

Lepšie riešenie (kvadratická časová zložitosť)

Ako sme si všimli, nikdy nás nezaujímala konkrétne hodnota nejakého prvku, vždy sme sa len pýtali, či je menší alebo väčší ako 42. Upravme si teda naše vstupné pole tak, že namiesto čísla 42 dáme 0, čísla väčšie ako 42 zmeníme na $+1$ a čísla menšie na -1. Podmienku „úsek obsahuje rovnako veľa čísel väčších ako 42 a menších ako 42“ teraz vieme vyjadriť jednoducho ako „úsek má súčet 0“.

Teda platí, že konkrétna súvislá podpostupnosť má medián 42 práve vtedy, ak po úprave poľa obsahuje 0 a zároveň má súčet rovný 0. (Všimnite si, že nepotrebujeme uvažovať podmienku o nepárnej dĺžke. Tá totiž vyplýva zo zvyšných dvoch – každá postupnosť, ktorá obsahuje jednu 0 a rovnako veľa $+1$ a -1 musí mať nepárnu dĺžku.)

Kubické riešenie bolo zbytočne pomalé preto, že pre každý začiatok a koniec začínalo vždy rátať odznova. Skúsme to teda šikovnejšie. Postupne vyskúšame všetky z od 1 po p. Pre konkrétne z najskôr položíme $k = z$. V tejto chvíli je súčet úseku rovný hodnote na políčku z. Následne zväčšujeme k až po n a zakaždým v konštantnom čase prepočítame súčet zodpovedajúceho úseku. (Iba zoberieme doterajší súčet a pripočítame k nemu nasledujúci člen postupnosti.) No a vždy, keď $k \geq p$ a aktuálny súčet je rovný 0, sme našli jeden z vyhovujúcich úsekov.

Toto riešenie teda každú dvojicu $z \leq k$ spracuje v konštantnom čase, a teda má časovú zložitosť $\Theta(n^2)$.

(Poznámka: Na myšlienke „pre každý začiatok postupne zvyšujem koniec“ sa tiež dá navrhnúť riešenie s o trochu horšou časovou zložitosťou $\Theta(n^2 \log n)$. Pri ňom pracujeme priamo s pôvodnými hodnotami zo vstupu. Použijeme usporiadanú množinu (`multiset` v C++), do ktorej postupne vkladáme prvky postupnosti od z-teho ďalej. Pritom si udržiavame ukazovateľ (iterátor) na medián.)

Iné kvadratické riešenie

Aj vyššie uvedené kvadratické riešenie ešte robí veľakrát to isté: pre každé z prechádzame v cykle pre k aspoň od p po n a znova a znova sčitujeme tie isté prvky. Pokúsime sa túto neefektívnosť odstrániť. Najskôr to síce opäť povedie ku riešeniu s kvadratickou časovou zložitosťou, toto však nižšie vylepšíme.

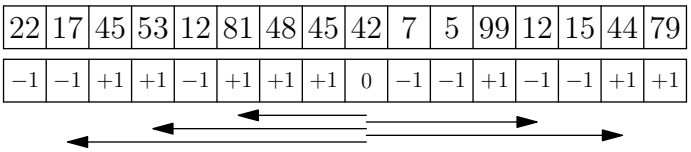
Začneme rovnakým predspracovaním ako vyššie uvedené kvadratické riešenie: zmeníme prvky na 0, $+1$ky a -1ky. Potom ale budeme prvky sčitovať len raz. Najskôr začneme od p a pre každé k si spočítame súčet β_k úseku od p po k. Toto celé vieme spraviť v lineárnom čase. No a následne spravíme to isté v opačnom smere: pre každé z od p po 1 spočítame súčet α_z úseku od z po p.

Pomocou takto predpočítaných hodnôt vieme spraviť ďalšie riešenie v čase $\Theta(n^2)$: vyskúšame všetky dvojice z, k pre ktoré platí $z \leq p \leq k$ a vyberieme tie z nich, kde $\alpha_z + \beta_k = 0$. (Alebo inými slovami, $\alpha_z = -\beta_k$.)

Vzorové riešenie

Ak chceme lepšiu ako kvadratickú časovú zložitosť, potrebujeme nájsť spôsob, ako zarátať viac dobrých úsekov naraz. Tu nám pomôže, keď si uvedomíme, že súčty α_z a β_k nadobúdajú hodnoty len od $-n$ po n.

Hlavnú myšlienku si najskôr ukážeme na obrázku:



- Horný riadok ukazuje vstupné pole, v dolnom sú hodnoty po úvodnej úprave.
- Šípky smerujúce doľava ukazujú tri rôzne úseky, pre ktoré platí $\alpha_z = 3$.
- Šípky smerujúce doprava ukazujú dva rôzne úseky, pre ktoré platí $\beta_k = -3$.
- Spojením hociktorého z týchto ľavých a hociktorého z týchto pravých úsekov dostaneme úsek so súčtom 0.

Formálne, pre každé s, nech $A[s]$ je počet z takých, že $\alpha_z = s$ a $B[s]$ nech je počet k takých, že $\beta_k = s$. Zvoľme si konkrétne s (medzi $-n$ a n, vrátane). Koľko existuje postupností takých, že $\alpha_z = s$ a $\beta_k = -s$? Presne $A[s] \times B[-s]$: máme totiž $A[s]$ možností, kde takáto postupnosť môže začínať, a nezávisle od toho $B[-s]$ možností, kde končí.

Polia A a B vieme vypočítať v lineárnom čase od n (takmer rovnakým algoritmom ako sme počítali hodnoty α_z a β_k). Aj následné vyskúšanie všetkých možných s a spočítanie dobrých podpostupností prebehne v lineárnom čase. Práve sme teda popísali riešenie s časovou zložitosťou $\Theta(n)$.

Listing programu:

```
#include <iostream>
#define MAXN 1234567

// drobny trik: deklarujeme A a B tak, aby do nich slo indexovat zapornymi cislami
int N, P[MAXN], Ashift[2*MAXN+1], Bshift[2*MAXN+1], *A=Ashift+MAXN, *B=Bshift+MAXN;

int main() {
    // nacitame a upravime vstup
    std::cin >> N;
    for (int n=0; n<N; ++n) { std::cin >> P[n]; P[n] = P[n]>42 ? 1 : P[n]<42 ? -1 : 0; }

    // najdeme nulu
    int p=0; while (P[p]!=0) ++p;

    // zistime sucty usekov zacinajucich a konciacich nulou
    int alpha=0, beta=0;
    for (int i=p; i>=0; --i) { alpha += P[i]; ++A[alpha]; }
    for (int i=p;  i<N; ++i) { beta  += P[i]; ++B[beta];  }

    // spocitame odpoved
    long long odpoved = 0;
    for (int i=-N; i<=N; ++i) odpoved += 1LL * A[i] * B[-i];
    std::cout << odpoved << "\n";
}
```

A-III-2 U obchodníka s rozličným kradnutým tovarom

V tomto vzoráku budeme predpokladať, že suma s, ktorú treba zaplatiť, je rádovo väčšia ako hodnota najväčšej mince c_n. (To úplne zodpovedá dnešným trhovým cenám modelov vesmírnych lodí a tiež ohraničeniam v zadaní úlohy.) Tento predpoklad využijeme len pri porovnávaní efektivity rôznych riešení – nebude na ňom závisieť ich korektnosť.

Stručný prehľad riešení

Základným korektným (ale veľmi pomalým) riešením je skúšanie všetkých možností platenia. Napríklad postupne skúšať všetky možné spôsoby použitia 1, 2, 3, ... mincí, až kým nenájdeme spôsob, ako zaplatiť práve požadovanú sumu. Takéto riešenie má časovú zložitosť rádovo úmernú n^m, kde m je počet mincí, ktoré treba v optimálnom riešení.

Existujú aspoň tri rôzne prístupy, ktoré vedú k riešeniu s časovou zložitosťou lineárnou od s:

1. Pre dostatočne veľa súm vypočítame optimálne zaplatenie bez vydávania a následne vyskúšame všetky potrebné možnosti pre sumu, ktorú Roberto obchodníkovi vydá. To ťažké na tomto riešení je určiť, ktoré možnosti už netreba skúšať.

2. Iné riešenie je založené na grafovom pohľade: Vrcholy grafu sú celé čísla predstavujúce sumu, ktorú ešte treba zaplatiť. Každá hrana predstavuje použitie jednej mince. V takomto grafe hľadáme najkratšiu cestu z 0 do s. Počas tohoto hľadania pre každú sumu „po ceste“ vypočítame najkratšiu vzdialenosť do nej – teda najmenší počet mincí potrebný na jej zaplatenie.

3. Časovú zložitosť zhruba kvadratickú od s (a ešte závisiacu aj od n a c_n) vieme dosiahnuť tak, že postupne pre každý počet mincí zostrojíme množinu všetkých súm, ktoré sa dajú daným počtom mincí zaplatiť.

   Vhodnými optimalizáciami (generovaním len niektorých vhodných súm, nie všetkých) sa dá toto riešenie vylepšiť tak, aby od s závisela jeho časová zložitosť len lineárne.

Vzorové riešenie navyše pridáva pozorovania, ktoré nám umožnia sumy s väčšie od istej hranice platiť pažravo. Tým dostávame riešenie, ktorého časová zložitosť závisí len od n a c_n.

Platba bez vydávania

Začnime riešením zjednodušenej úlohy, v ktorej musí obchodník zaplatiť presne sumu s bez toho, aby mu Roberto vydal. Na prvý pohľad by sa mohlo zdať, že pri platení je vždy optimálne použiť čo najväčšiu možnú mincu, no už vstup v zadaní nás presvedčí o opaku: Keď máme mince $(1, 5, 6)$ a chceme zaplatiť sumu 10, stačia nám na to dva kusy mincí: $5+5$. Ak by sme však použili mincu 6, zvyšok sumy by sme museli zaplatiť pomocou mince 1, teda dokopy by sme potrebovali päť kusov mincí.

Takéto *greedy* (pažravé) riešenie je síce nekorektné, no pomôže nám získať horný odhad správneho výsledku. Na zaplatenie sumy s vždy stačí menej ako $\lfloor s/c_n \rfloor + c_n$ mincí. Totiž jeden spôsob platenia je, že platíme mincou c_n, kým sa dá, a zvyšok (ktorý je určite menší ako c_n) zaplatíme mincou 1. Ak by sme počítali so všetkými druhmi mincí (nielen s 1 a c_n), mohli by sme dostať ešte lepší odhad, nám však postačí aj tento.

Označme najmenší počet kusov mincí potrebných na zaplatenie sumy i bez možnosti vydávania ako $P[i]$. Použijeme princíp *dynamického programovania*: Hodnoty $P[i]$ budeme počítať postupne od menších i k väčším a pri tom budeme používať už vyrátané hodnoty.

Na zaplatenie sumy 0 nám stačí 0 mincí, preto $P[0] = 0$. Keď máme zaplatiť sumu $i > 0$, môžeme začať ľubovoľnou mincou c_j, ktorej hodnota nepresahuje i. Použijeme tak jednu mincu a zostane nám zaplatiť $i - c_j$, na čo potrebujeme minimálne $P[i - c_j]$ mincí. Zo všetkých možných mincí c_j si samozrejme vyberieme tú najvýhodnejšiu. Hodnotu $P[i]$ teda vypočítame podľa vzťahu:

$$P[i] = \min \left\{ P[i - c_j] + 1 \;\middle|\; 1 \leq j \leq n \ \wedge \ c_j \leq i \right\}$$

Všimnite si, že pri výpočte $P[i]$ naozaj využívame len známe hodnoty. Aby sme nakoniec vedeli vypísať nejaký optimálny spôsob platenia, pre každú sumu i si zapamätáme aj to, ktorou mincou ju máme začať platiť.

V poli P si potrebujeme pamätať $O(s)$ hodnôt; každú z nich vypočítame v čase $O(n)$. Časová zložitosť je preto $O(sn)$ a pamäťová $O(s)$. Zdôrazňujeme, že ešte nejde o korektné riešenie úlohy zo zadania, len o pomocnú úvahu, ktorú ďalej využijeme.

Riešenie systémom „najskôr zaplať, potom vydaj“

Na optimálny spôsob platenia sa môžeme dívať tak, že najskôr obchodník Robertovi zaplatí nejakú sumu $s + x$ a následne mu Roberto vydá sumu x. Obchodník na to samozrejme v optimálnom riešení použije $P[s + x]$ mincí a Roberto $P[x]$. Stačí teda nájsť to optimálne $x \geq 0$, pre ktoré bude súčet $P[s+x]+P[x]$ najmenší možný.

Samozrejme, zatiaľ máme pre x nekonečne veľa možností a rozhodne by sme ich nechceli skúšať všetky. Otázka preto znie: aké najväčšie môže byť x?

Tu treba byť opatrný, pretože je veľmi ľahké oklamať sám seba napríklad tvrdením „Roberto nikdy nepoužije najväčšiu mincu“ alebo tvrdením „Roberto vždy bude vydávať nanajvýš s“.

Protipríklad na obe tieto tvrdenia: mince $(1, 90, 100)$ a suma $s = 70$. Jediným optimálnym riešením je, že obchodník zaplatí $90 + 90 + 90$ a Roberto mu vydá $100 + 100$ (teda dokopy 5 kusov mincí zmení majiteľa).

Jeden možný (aj keď nie najlepší) korektný horný odhad, po aké x treba skúšať:

Každý typ mince zjavne použije len jeden z nich – nemá zmysel, aby obchodník nejakými mincami platil a Roberto mu také isté vydal.

Ak Roberto v optimálnom riešení nepoužije mincu s cenou c_n: Každej inej mince použije najviac $c_n - 1$ kusov. (Ak by totiž použil c_n kusov mince s cenou c_i, bolo by výhodnejšie použiť c_i kusov mince s cenou c_n.) Dokopy teda Roberto použije určite menej ako nc_n mincí, každá má cenu menej ako c_n, a teda $x < nc_n^2$.

Ak Roberto použije mincu s cenou c_n: Znamená to, že túto mincu nechcel použiť obchodník. Rovnakou argumentáciou ako v predchádzajúcom prípade dostávame, že suma $s + x$, ktorú platil obchodník, je nutne menšia ako nc_n^2. A to isté teda tým skôr platí pre hodnotu x.

Výsledkom je teda nasledujúce riešenie: Vypočítame hodnoty v poli P od 0 po $s + nc_n^2$. Potom vyskúšame všetky x od 0 po nc_n^2 a nájdeme to, pre ktoré je súčet $P[s+x] + P[x]$ najmenší možný. Toto riešenie má časovú zložitosť $O(sn + n^2c_n^2)$.

Riešenie pomocou prehľadávania grafu

Pri návrhu riešenia zadanej úlohy nás môže pokúšať prístup, pri ktorom priamo počas jedného dynamického programovania budeme spracúvať aj možnosti, kedy platí obchodník, aj možnosti, kedy vydáva Roberto. Označme $Q[i]$ najmenší počet kusov mincí potrebných na zaplatenie sumy i, ak je povolené aj vydávanie. Tou istou úvahou ako pri poli P dostávame rekurentný vzťah:

$$Q[i] = \min \left\{ Q[i - c_j] + 1 \;\middle|\; 1 \le j \le n \;\wedge\; c_j \le i \right\} \;\cup\; \left\{ Q[i + c_j] + 1 \;\middle|\; 1 \le j \le n \right\}$$

Tu je ale problém: Tým, že sme povolili aj odčítanie, nám vo vzťahoch vznikli cykly: napr. hodnota $Q[7]$ závisí na hodnote $Q[7 + c_n]$ a zároveň aj $Q[7 + c_n]$ závisí na $Q[7]$. Tadiaľto teda cesta nevedie.

Tu je dobré si uvedomiť, že tá situácia, do ktorej sme sa dostali pri návrhu rekurencií, je totožná s tou, ktorú stretávame pri hľadaní najkratších ciest: vzdialenosť z Bratislavy do Popradu a z Bratislavy do Štrby spolu súvisia, lebo aj z Popradu môžeme ísť ďalej na Štrbu, aj zo Štrby do Popradu.

Na celý problém sa teda môžeme dívať ako na hľadanie najkratšej cesty v grafe. Vrcholmi grafu sú celé čísla, predstavujúce aktuálne zaplatenú sumu. Každá hrana má dĺžku 1 a zodpovedá použitiu mince či už jedným alebo druhým aktérom. V takomto grafe hľadáme najkratšiu cestu z vrcholu 0 do vrcholu s. Keďže všetky hrany majú jednotkovú dĺžku, môžeme použiť prehľadávanie do šírky.

Každý navštívený vrchol spracujeme v čase lineárnom od n (počtu mincí, a teda počtu vychádzajúcich hrán). Aby sme odhadli časovú aj pamäťovú zložitosť, stačí teda zhora odhadnúť počet navštívených vrcholov. Na to si pripomeňme, že máme odhad $Q[s] < \lfloor s/c_n \rfloor + c_n$. Prehľadávanie navštívi len tie sumy, ktoré sú od 0 vo vzdialenosti nanajvýš rovnej $Q[s]$.

Všetky sumy, ktoré sa dajú zaplatiť najviac k mincami, zjavne ležia v rozsahu od $-kc_n$ po kc_n. Po dosadení nášho odhadu pre $Q[s]$ dostávame, že naše prehľadávanie navštívi $O(s + c_n^2)$ vrcholov. Taká je teda aj jeho pamäťová zložitosť; časová zložitosť je potom $O(ns + nc_n^2)$.

Riešenie pomocou zostrojenia všetkých zaplatiteľných súm

Označme M_i množinu súm, ktoré sa dajú zaplatiť použitím presne i kusov mincí; zrejme $M_0 = \{0\}$. Ak $j \in M_i$, potom pre každú mincu c_k platí $j + c_k \in M_{i+1}$ (lebo stačí i mincami zaplatiť j a potom obchodník pridá mincu c_k) a $j - c_k \in M_{i+1}$ (Roberto vydá mincu c_k). Naopak, každá suma v M_{i+1} musela vzniknúť z nejakej sumy v M_i týmto spôsobom.

Pre sadu mincí $(1, 5)$ takto dostaneme $M_0 = \{0\}$, $M_1 = \{-5, -1, 1, 5\}$, $M_2 = \{-10, -6, -4, -2, 0, 2, 4, 6, 10\}$, $M_3 = \{-15, -11, -9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11, 15\}$, ...

Optimálnym počtom mincí na zaplatenie s je najmenšie také i, pre ktoré $s \in M_i$. Stačí nám teda postupne generovať $M_0, M_1, M_2, \ldots$, kým nezískame s.

Každú z množín M_i si môžeme reprezentovať poľom booleovských hodnôt, v ktorom si budeme pre jednotlivé sumy pamätať, či ich množina obsahuje alebo nie.[1] Iným spôsobom je pamätať si v poli pre každú sumu j najmenšie také i, pre ktoré $j \in M_i$. Z hľadiska optimálneho platenia nás totiž zaujíma len prvý výskyt sumy j. Pole s rovnakým významom sme si už skôr označili Q.

Vďaka odhadu $Q[s] < \lfloor s/c_n \rfloor + c_n$ vieme, že počet vygenerovaných množín bude nanajvýš $k = \lfloor s/c_n \rfloor + c_n - 1$ a ako sme ukázali už v predchádzajúcom riešení, rôznych hodnôt v záverečnej množine M_k (a teda aj v každej M_i) je $O(kc_n)$. Vygenerovať M_{i+1} z M_i vieme teda v čase $O(nkc_n)$. A celkovú časovú zložitosť dostávame ako súčet časov potrebných na vygenerovanie všetkých množín M_i. Celkovo teda vieme časovú zložitosť zhora odhadnúť ako $O(nk^2c_n)$. Po dosadení nášho odhadu za k dostávame teda horný odhad časovej zložitosti $O(ns^2/c_n + nc_n^3)$. Veľmi zjednodušene môžeme povedať, že čas behu tohto algoritmu rastie kvadraticky v závislosti od sumy s, ktorú platíme.

Optimalizácia predchádzajúceho riešenia

Jednu optimalizáciu práve popísaného riešenia sme už videli: je ňou grafové riešenie. Zatiaľ, čo riešenie s množinami M_i pre každú sumu postupne zostrojuje *všetky* počty mincí, na ktoré sa ju dá zaplatiť, grafové riešenie nájde len ten *najlepší* z nich. Teraz si ešte ukážeme jeden trochu iný spôsob, ako riešenie s množinami M_i vylepšiť. Aby sme vedeli generovať množiny M_i efektívnejšie, obmedzíme sa iba na určité poradia mincí pri platení. Zavedieme dve pravidlá:

1. Aktuálne zaplatená suma nesmie nikdy presiahnuť s.
2. Roberto smie vydávať iba vtedy, keď je aktuálna suma väčšia ako $s - c_n$.
   (Teda nesmie vydávať, ak sa ešte dá bez porušenia prvého pravidla platiť každou mincou.)

Napríklad pre sadu mincí $(1, 5)$ a $s = 9$ budú množiny M_i vyzerať takto: $M_0 = \{0\}$, $M_1 = \{1, 5\}$, $M_2 = \{0, 2, 4, 6\}$, $M_3 = \{1, 3, 5, 7, 9\}$. Napriek tomu, že nám niektoré sumy ubudli, suma 9 sa dá stále zaplatiť tromi mincami. Aj vo všeobecnosti platí, že sa počet mincí potrebných na zaplatenie s nezväčší. Poďme si ukázať, že vždy existuje také poradie platenia a vydávania, ktoré spĺňa obe pravidlá.

Nech v optimálnom riešení obchodník zaplatí mincami $a_1, a_2, \ldots, a_u$ a Roberto vydá mince $b_1, b_2, \ldots, b_v$. Platí teda $a_1 + a_2 + \cdots + a_u - b_1 - b_2 - \cdots - b_v = s$. Korektné poradie dostaneme jednoducho tak, že vždy, keď to nezakazuje pravidlo č. 1, zaplatíme ďalšou z mincí a_i. V opačnom prípade je aktuálna suma väčšia ako $s - c_n$ (inak by sme nemohli zaplatením ďalšou mincou porušiť pravidlo č. 1), preto vydáme ďalšou z mincí b_j. Uvedomte si, že skôr ako mince na platenie sa nám minú mince na vydávanie, a že tento postup sa zasekne, až keď použijeme všetky mince a dosiahneme sumu s.

Dodržiavaním týchto pravidiel teda nič nestratíme a navyše získame nasledujúce záruky: Pre každé optimálne riešenie existuje taká suma $r \in (s - c_n, s\rangle$ a poradie mincí, že najprv bez vydávania zaplatíme r a potom už s vydávaním doplatíme do sumy s. Počas tejto druhej fázy sa bude aktuálna suma pohybovať len v intervale $(s - 2c_n, s\rangle$.

Využime tieto záruky na urýchlenie nášho riešenia. Najprv spustíme algoritmus bez vydávania, čím si v čase $O(sn)$ nájdeme optimálny spôsob platenia bez vydávania pre všetky sumy od 0 po s. Z tých si necháme len interval $(s - c_n, s\rangle$. Získali sme tak „základ“ pre množiny M_i, platí totiž $j \in M_{P[j]}$. Vezmime najmenšiu z hodnôt $P[s-c_n+1], \ldots, P[s-1], P[s]$ a označme ju w. Ďalej budeme generovať množiny $M_{w+1}, M_{w+2}, \ldots$ naším pôvodným algoritmom, pričom sa obmedzíme len na sumy z intervalu $(s - 2c_n, s\rangle$. Takto časom vypočítame optimálny spôsob zaplatenia sumy s.

Pretože nás teraz zaujímajú len sumy z intervalu dĺžky $2c_n$, stačí nám $O(c_n)$ pamäte. Časová zložitosť sa zlepší na $O(nc_n^2)$, teda celé riešenie bude bežať v čase $O(ns + nc_n^2)$.

Ako riešiť vstupy pre $s \approx 10^{18}$

Intuícia nám radí, že ak je s príliš veľké, optimálne bude zaplatiť väčšinu mincami c_n. Poďme si podobné tvrdenie sformalizovať a dokázať.

[1] Polia zvyčajne nemôžeme indexovať zápornými číslami. Toto obmedzenie ľahko obídeme napríklad tak, že sume j priradíme index $j + t$, kde t je dostatočne veľké číslo. Alebo si môžeme pomôcť trikom, viď program k 1. úlohe.

Majme nejakú postupnosť mincí použitých obchodníkom na platenie, označme ich $d_1, d_2, \ldots, d_\ell$. Nech sa v tejto postupnosti ani raz nenachádza minca c_n a nech $\ell \geq c_n$.

Vezmime prefixové súčty $p_0 = 0$, $p_1 = d_1$, $p_2 = d_1 + d_2$, $\ldots$, $p_\ell = d_1 + d_2 + \cdots + d_\ell$. Pretože týchto súčtov je $\ell + 1$, čo je ostro viac ako c_n, existujú také dva súčty p_i a p_j $(i < j)$, ktoré dávajú rovnaký zvyšok po delení číslom c_n. To znamená, že ich rozdiel $p_j - p_i = d_{i+1} + \cdots + d_j$ je deliteľný c_n. Časť postupnosti od $(i+1)$-vej po j-tu mincu teda môžeme nahradiť niekoľkými mincami c_n, pričom celkový počet použitých mincí sa zmenší.

Najväčšia hodnota, akú môžeme získať použitím najviac $c_n - 1$ kusov mincí, ak nemáme k dispozícii žiadnu mincu c_n, je $(c_n - 1) \cdot c_{n-1}$. Teda ak $s > (c_n - 1) \cdot c_{n-1}$, potom pri optimálnom platení sumy s obchodník určite použije aspoň jednu mincu c_n – v opačnom prípade by potreboval aspoň c_n kusov mincí, čo by sa dalo zlepšiť podľa predchádzajúceho odseku.

Túto úvahu môžeme opakovať dovtedy, kým $s > (c_n - 1) \cdot c_{n-1}$; zakaždým znížime s o c_n a obchodníkovi zarátame použitie ďalšej mince. Efektívnejším spôsobom je pomocou delenia zistiť počet opakovaní tohto postupu a zmeny potom vykonať naraz v konštantnom čase.

Keďže suma, po ktorú musíme počítať hodnoty P, sa zmenší na $O(c_n^2)$, celková časová zložitosť nášho riešenia klesne na $O(nc_n^2)$.

Listing programu:

```
#include <cstdio>
#include <vector>
#define INF 1023456789
using namespace std;

int main() {
    long long s;
    int n;
    scanf("%lld%d", &s, &n);
    vector<int> C(2 * n);  // za n minci zo vstupu pridame ich opacne hodnoty
    for (int i = 0; i < n; ++i) {
        scanf("%d", &C[i]);
        C[i + n] = -C[i];
    }
    int cn = C[n - 1];  // najvaecsia minca
    vector<long long> R(2 * n, 0);  // pocty minci v optimalnom plateni

    // zabezpecime, ze s <= (c_n - 1) * c_{n - 1}
    int q = C[n - 2] * (cn - 1);
    if (s > q) {
        R[n - 1] = (s - q + cn - 1) / cn;
        s -= R[n - 1] * cn;
    }

    // optimalne platenie bez vydavania
    vector<int> P(s + 1, INF);
    P[0] = 0;
    vector<int> tah_P(s + 1, -1);
    for (int i = 1; i <= s; ++i)
        for (int j = 0; j < n; ++j)
            if (C[j] <= i && P[i] > P[i - C[j]] + 1) {
                P[i] = P[i - C[j]] + 1;
                tah_P[i] = j;
            }

    // Q inicializujeme hodnotami z P ...
    vector<int> Q(2 * cn, INF);
    vector<int> tah_Q(2 * cn, -1);
    int t = s - 2 * cn + 1;  // sumy v Q maju indexy posunute o t
    int w = INF;
    for (int i = max(0ll, s - cn + 1); i <= s; ++i) {
        Q[i - t] = P[i];
        w = min(w, P[i]);
    }

    // ... a dalej zlepsujeme, tentoraz aj s vydavanim
    for (int i = w; i != Q[s - t]; ++i)
        for (int j = 0; j < 2 * cn; ++j)
            if (Q[j] == i)  // z prvku mnoziny M_i vyrabame prvky M_{i+1}
                for (int k = 0; k < 2 * n; ++k)
                    if (0 <= j + C[k] && j + C[k] < 2 * cn && Q[j + C[k]] > i + 1) {
                        Q[j + C[k]] = i + 1;
                        tah_Q[j + C[k]] = k;
                    }

    // zrekonstruujeme optimalne platenie
    while (tah_Q[s - t] != -1) {
        ++R[tah_Q[s - t]];
        s -= C[tah_Q[s - t]];
    }
    while (tah_P[s] != -1) {
        ++R[tah_P[s]];
        s -= C[tah_P[s]];
    }

    for (int i = 0; i < 2 * n; ++i)
        printf("%lld%c", R[i], (i % n == n - 1) ? '\n' : ' ');
}
```

A-III-3 Zlomkové programy sa lúčia

Pri riešení tejto úlohy ste pravdepodobne prišli na to, že podúlohy b1 a b2 spolu súvisia. Pri hľadaní odmocniny z čísla x totiž hľadáme najmenšie y také, že $y^2 \geq x$. Riešenie podúlohy b1 sa teda pravdepodobne bude dať využiť pri riešení podúlohy b2.

Ale skôr, než sa na to pozrieme podrobnejšie, si vyriešme nesúvisiacu podúlohu a.

Budeme používať terminológiu, ktorú sme zaviedli v riešeniach krajského kola: na jednotlivé prvočísla v rozklade n sa dívame ako na premenné; ich exponenty sú hodnoty, ktoré sú v tých premenných uložené. Teda napr. číslo $2^7 5^4$ prečítame ako „v premennej #2 je uložená hodnota 7 a v premennej #5 hodnota 4“.

Podúloha a)

Medián troch premenných je zároveň rovný druhej najmenšej z nich. Vieme ho zostrojiť napr. nasledovne:

1. Kým sú všetky tri premenné kladné, každú zníž o 1 a zároveň zvýš medián o 1.
2. Kým sú práve dve premenné kladné, obe zníž o 1 a zároveň zvýš medián o 1.
3. V tejto chvíli už máme nájdený medián, ale ešte treba upratať:
   Kým je posledná premenná kladná, zníž ju o 1.

Samozrejme, my vopred nevieme, ktoré dve premenné budú kladné v kroku 2 a ktorá z nich bude kladná ešte aj v kroku 3. Tu je ale ľahká pomoc: do programu dáme všetky tri možnosti. Tá z nich, ktorá nastala, sa použije a tie, ktoré nenastali, sa použiť nebudú dať.

Výsledný program teda vyzerá nasledovne:

$$\left(\frac{7}{2\cdot 3\cdot 5} \;,\; \frac{7}{2\cdot 3} \;,\; \frac{7}{2\cdot 5} \;,\; \frac{7}{3\cdot 5} \;,\; \frac{1}{2} \;,\; \frac{1}{3} \;,\; \frac{1}{5} \right)$$

Príklad výpočtu pre $n = 4320 = 2^5 3^3 5^1$:

$$2^5 3^3 5^1 \xrightarrow{1} 2^4 3^2 7 \xrightarrow{2} 2^3 3^1 7^2 \xrightarrow{2} 2^2 7^3 \xrightarrow{5} 2^1 7^3 \xrightarrow{5} 7^3$$

Časová zložitosť nášho programu je priamo úmerná hodnote $\max(x, y, z)$.

Iné riešenie s rovnakou časovou zložitosťou sa dá založiť na myšlienke: pre tri čísla x, y a z je ich mediánom to, ktoré nie je ani najväčšie, ani najmenšie. Preto platí $\text{median}(x, y, z) = x + y + z - \max(x, y, z) - \min(x, y, z)$. No a sčítanie, odčítanie, maximum aj minimum už vieme naprogramovať – viď riešenia krajského kola.

Podúloha b1), prvé riešenie

Namiesto výpočtu druhej mocniny čísla x môžeme vyriešiť o trochu všeobecnejšiu úlohu: násobenie. Napíšeme teda zlomkový program, ktorý bude vedieť ľubovoľné číslo tvaru $5^y 7^z 47$ prerobiť na číslo 3^{yz}. A ak sa nám toto podarí, už sme vyhrali: stačí na jeho začiatok pridať zlomky, ktoré z čísla $2^x 47$ vyrobia číslo $5^x 7^x 47$ a máme program počítajúci druhú mocninu.

Ako teda vyriešiť násobenie? Prevedieme ho na sčítanie, ktoré už robiť vieme (viď riešenia krajského kola). Začneme s 3^0 a y-krát k tej 0 pripočítame z.

Jedno kolo pripočítavania bude vyzerať nasledovne:

1. Kým je premenná #7 kladná: zníž #7, zvýš #3, zvýš #11.
2. Kým je premenná #11 kladná: zníž #11, zvýš #7.
3. O jednu zníž #5.

Ak teda začneme s číslom $3^0 5^y 7^z$, po jednom kole pripočítavania budeme mať $3^z 5^{y-1} 7^z$, po ďalšom kole $3^{2z} 5^{y-2} 7^z$, a tak ďalej. A keď sa nám minie y, dostaneme číslo $3^{yz} 7^z$. Potom už len vyprázdnime premennú #7 a máme násobenie hotové.

Presnejšie, rovnako ako v niektorých úlohách domáceho a krajského kola, aj tu budeme potrebovať jedno prvočíslo navyše. Pomocou toho si budeme pamätať, či sme ešte vo fáze 1 (znižujeme premennú #7), alebo už vo fázach 2 a 3 (naspäť zvyšujeme premennú #7). Prvočísla 47 a 43 budú predstavovať fázu 1, prvočísla 59 a 53 fázu 2 a 3.

Program pre násobenie teda bude vyzerať nasledovne:

$$\left(\frac{3 \cdot 11 \cdot 43}{7 \cdot 47} \;,\; \frac{47}{43} \;,\; \frac{59}{47} \;,\; \frac{7 \cdot 53}{11 \cdot 59} \;,\; \frac{59}{53} \;,\; \frac{61}{5^2 \cdot 59} \;,\; \frac{47 \cdot 5}{61} \right)$$

Povšimite si posledné dva zlomky. Vždy, keď sa dostaneme na koniec cyklu, chceme znížiť premennú #5 a potom otestovať, či už je na nule. To spravíme tak, že sa ju pokúsime znížiť o 2 (predposledný zlomok) a ak sa nám to podarilo, je všetko ok, zvýšime ju o 1 a pokračujeme novou iteráciou cyklu.

Keď teraz pridáme aj inicializáciu a upratanie po skončení násobenia, dostávame kompletný program:

$$\left(\frac{5 \cdot 7}{2} \;,\; \frac{3 \cdot 11 \cdot 43}{7 \cdot 47} \;,\; \frac{47}{43} \;,\; \frac{59}{47} \;,\; \frac{7 \cdot 53}{11 \cdot 59} \;,\; \frac{59}{53} \;,\; \frac{61}{5^2 \cdot 59} \;,\; \frac{47 \cdot 5}{61} \;,\; \frac{1}{59} \;,\; \frac{1}{7} \;,\; \frac{1}{5} \right)$$

Aká je časová zložitosť tohto programu? Začneme s číslom 2^x. V $\Theta(x)$ krokoch si vyrobíme číslo, z ktorého začneme počítať násobenie. Každá iterácia násobenia zahŕňa $\Theta(x)$ krokov výpočtu v prvej, $\Theta(x)$ krokov v druhej a $\Theta(1)$ krokov v tretej fáze. Iterácií bude presne x. No a záverečné upratovanie má tiež len $\Theta(x)$ krokov. Dokopy teda dostávame časovú zložitosť $\Theta(x^2)$. A keďže potrebujeme rádovo takú veľkú hodnotu vyrobiť v premennej #3, lepšie to ani nejde.

Podúloha b1), druhé riešenie

Tentokrát nebudeme násobiť, ale rovno postupne počítať druhé mocniny od 1^2 až po x^2.

Presnejšie, v premennej #3, v ktorej má skončiť výstup, budeme mať hodnotu i^2, zatiaľ čo v premennej #5 budeme mať hodnotu $2i - 1$. Každá iterácia výpočtu bude vyzerať nasledovne:

1. Zväčši premennú #5 o dva. (Nová hodnota: $2i + 1$.)
2. Pridaj obsah premennej #5 do premennej #3. (Nová hodnota v #3: $i^2 + 2i + 1 = (i + 1)^2$.)

Zároveň pri každej iterácii znížime o jedna hodnotu vo vstupnej premennej #2. A keď tá klesne na z x na nulu, budeme mať v premennej #5 želanú hodnotu x^2.

Výsledný program:

$$\left(\frac{3 \cdot 5 \cdot 59}{2 \cdot 47} \;,\; \frac{5 \cdot 5 \cdot 43}{2 \cdot 59} \;,\; \frac{3 \cdot 7 \cdot 41}{5 \cdot 43} \;,\; \frac{43}{41} \;,\; \frac{37}{43} \;,\; \frac{5 \cdot 31}{7 \cdot 37} \;,\; \frac{37}{31} \;,\; \frac{59}{37} \;,\; \frac{1}{59} \;,\; \frac{1}{5} \;,\; \frac{1}{47} \right)$$

Trocha komentára k nemu:

- Prvý zlomok slúži na inicializáciu, použije sa len raz – v prvom kroku výpočtu.
- Druhý zlomok je začiatok cyklu: znížime premennú #2 a o dve zvýšime premennú #5.
- Tretí a štvrtý zlomok: znižujeme #5 a zvyšujeme #3. Zároveň zvyšujeme #7, aby sa nám pôvodná hodnota premennej #5 nestratila.
- Piaty až siedmy zlomok: keď sa #5 minie, presunieme obsah #7 späť do #5.
- Ôsmy zlomok: späť na začiatok cyklu.
- Deviaty a desiaty zlomok: Upratovanie – po poslednej iterácii cyklu zmažeme čo už nechceme.
- Jedenásty zlomok sa použije len pri $x = 0$.

Tentokrát je ešte očividnejšie, že časová zložitosť tohto programu je tiež $\Theta(x^2)$.
(Zaujímavosť: Pre ľubovoľné $x \geq 1$ spraví tento program presne o šesť krokov menej ako náš prvý program.)

Podúloha b2), pomalšie riešenie

Ako sme už uviedli na začiatku, úlohu „nájdi hornú celú časť odmocniny z x“ si môžeme preformulovať do podoby „nájdi najmenšie y také, že $y^2 \geq x$“.

Z toho vyplýva priamočiary algoritmus na riešenie podúlohy b2:

1. Nájdi riešenie podúlohy b1.
2. Postupne ho používaj pre $i = 1, 2, \ldots$, zakaždým vypočítaj i^2 a porovnaj vypočítanú hodnotu s x.

Akú bude mať tento algoritmus časovú zložitosť? Na vyskúšanie konkrétnej hodnoty i potrebujeme rádovo i^2 krokov. Dokopy teda spravíme krokov rádovo $1^2 + 2^2 + \cdots + y^2$. Z toho dostávame, že časová zložitosť je $\Theta(y^3) = \Theta(x\sqrt{x})$.

Toto riešenie by dostalo 4 body.

Podúloha b2), lepšie riešenie

Skúsenejší riešiteľ si isto uvedomí, kde sa dá ušetriť: Nebudeme odpovede skúšať sekvenčne všetky, ale použijeme upravené binárne vyhľadávanie.

V prvej fáze budeme teda skúšať hodnoty $i = 1, 2, 4, 8, \ldots$, až kým nenájdeme prvú, pre ktorú $i^2 \geq x$. V tejto chvíli vieme, že hľadané y leží niekde od $2^j + 1$ do 2^{j+1} pre nejaké j. V druhej fáze v tomto intervale správnu hodnotu y nájdeme binárnym vyhľadávaním.

Takto otestujeme dokopy $\Theta(\log y) = \Theta(\log x)$ rôznych hodnôt. Otestovanie jednej hodnoty vieme spraviť v čase $O(x)$, čím dostávame odhad časovej zložitosti $O(x \log x)$.

(Tento odhad je tesný. Počas druhej fázy riešenia vyskúšame rádovo $\log y$ hodnôt a každá hodnota, ktorú skúšame, je väčšia ako $y/2$. Časová zložitosť je teda $\Theta(x \log x)$.)

Toto riešenie by dostalo 6 bodov. Jeho implementácia je dosť nepríjemná, obzvlášť realizácia binárneho vyhľadávania vyžaduje doriešiť viacero technických detailov.

Podúloha b2), optimálne riešenie

Existuje však aj riešenie, ktoré je ešte efektívnejšie a navyše sa výrazne ľahšie implementuje: Upravíme naše druhé riešenie podúlohy b1. Teda budeme postupne skúšať $i = 1, 2, 3, \ldots$, lenže nebudeme vždy odznovu počítať hodnotu i^2. Namiesto toho budeme zároveň so zvyšovaním i zmenšovať x tak, aby po vyskúšaní i bola vo vstupnej premennej hodnota $x - i^2$. Akonáhle klesne vstupná premenná na nulu, vieme, že aktuálna hodnota i je hľadanou odpoveďou.

Jeden možný zlomkový program podľa tejto myšlienky:

$$\left(\frac{5 \cdot 67}{2 \cdot 61}, \frac{61}{67}, \frac{3 \cdot 7^2 \cdot 61}{2}, \frac{5 \cdot 47}{61}, \frac{11 \cdot 43}{5 \cdot 7 \cdot 47}, \frac{47}{43}, \frac{1}{7 \cdot 47}, \frac{3 \cdot 5 \cdot 11^2 \cdot 31}{47}, \frac{7 \cdot 37}{11 \cdot 31}, \frac{31}{37}, \frac{47}{31}, \frac{1}{11}, \frac{1}{7} \right)$$

Priebeh výpočtu, rozdelený na fázy:

1. Začíname s číslom 2^x.
   Pre $x = 0$ výpočet rovno skončil. Inak vieme použiť tretí zlomok, dostaneme $2^{x-1} \cdot 3 \cdot 7^2 \cdot 61$.
2. Dokola používame prvé dva zlomky, čím vyrobíme $3 \cdot 5^{x-1} \cdot 7^2 \cdot 61$.
3. Keď sa minú dvojky, použijeme štvrtý zlomok. Dostávame hodnotu $3 \cdot 5^x \cdot 7^2 \cdot 47$.
4. Kedykoľvek, keď sa dostaneme na toto miesto výpočtu, budeme mať číslo tvaru $3^i \cdot 5^{x-(i-1)^2} \cdot 7^{2i} \cdot 47$. (Keď sme tu prvýkrát, je $i = 1$.)
5. Používame piaty a šiesty zlomok, kým sa to dá. Ak všetko ide dobre, zmenšíme obsah premennej #5 o obsah premennej #7, teda o $2i$. Hodnota v premennej #5 sa teda zmení z $x-(i-1)^2$ na $x-(i-1)^2-2i = x-i^2-1$. Ak sa to celé podarilo, vidíme, že pôvodné x bolo ostro väčšie ako i^2, hodnota i (stále uložená v premennej #3) teda ešte nie je hľadanou odpoveďou.
6. V takomto prípade pokračujeme ďalej. Použije sa ôsmy zlomok, čím si zvýšime premennú #3 o jedno a premennú #11 (kde máme dočasne presunutú hodnotu premennej #7) o dve. Zároveň zvýšime premennú #5 o jedno.
   V tejto chvíli máme teda aktuálnu hodnotu $3^{i+1} \cdot 5^{x-i^2} \cdot 11^{2i+2} \cdot 31$.

7. Teraz sa dokola použije deviaty a desiaty zlomok na presun hodnoty premennej #11 späť do premennej #7. Keď to skončí, použije sa jedenásty zlomok.
   Tým dostávame hodnotu $3^{i+1} \cdot 5^{x-i^2} \cdot 7^{2i+2} \cdot 47$ a výpočet opäť pokračuje od vyššie popísanej fázy 4.
8. Ak sa vo fáze 5 vyprázdni premenná #5 skôr ako premenná #7, výpočet končí a v premennej #3 máme hľadanú odpoveď. Ešte potrebujeme vyprázdniť ostatné premenné.
   Toto začneme použitím siedmeho zlomku. Následne už jediné zlomky, ktoré sa ešte dajú použiť, sú posledné dva. Ich opakovaným použitím vyčistíme ostatné premenné a ostane nám nenulová len premenná #3.

Odhad časovej zložitosti: Fázy 1-3 majú zjavne časovú zložitosť $\Theta(x)$. Potom nasleduje niekoľko iterácií. V rámci každej iterácie sa v 5. fáze vykoná rádovo rovnako veľa krokov výpočtu ako v 7. fáze; ostatné fázy vykonáme v konštantnom čase. Celková časová zložitosť iterovania je teda priamo úmerná celkovému počtu krokov výpočtu v 5. fáze. No ale tých je $\Theta(x)$, lebo v 5. fáze zmenšujeme x až kým neklesne na nulu.

Tento algoritmus má teda celkovú časovú zložitosť $\Theta(x)$.

DVADSIATY SIEDMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Autori úloh: Michal Anderle, Vladimír Boža, Michal Forišek, Peter Fulla, Ján Hozza
Recenzent: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2012