Увод у алгоритме - Рок 3 - 2025

Ово је испит из предмета увод у алгоритме из трећег испитног рока 2025. године преузет са странице `https://jason.matf.bg.ac.rs/test/ispitni%20rok%203/01%20Podela%20niza`. На тој страници се решења задатака могу тестирати и мој савет је да пре читања решења, пробате да урадите задатак и тек ако и тада не успете, да погледате ова решења. Срећно у припреми испита ☺

1. Поделити низ позитивних целих бројева на три дела тако да збир елемената из првог и трећег дела буде исти. Израчунати највећи могући збир елемената првог дела овако подељеног низа. (Приметимо да оваква подела сигурно постоји, први и трећи део низа могу бити празни и самим тим збир елемената тих делова је 0.)

   Са стандардног улаза се учитава број $n \in [0, 10^5]$, а затим и n целих бројева из интервала $[0, 10^3]$.

   На стандардни излаз исписати број који представља највећи могући збир елемената првог дела низа при оваквој подели.

Пример 1

```
Улаз:
5
1 1 1 1 4
Излаз:
4
```

Пример 2

```
Улаз:
7
1 1 3 12 3 3 2
Излаз:
5
```

Решење: Задатак решавамо техником два показивача. Такође чуваћемо збир са леве и збир са десне стране поред два показивача који одређују до ког елемента узимамо елементе у збир са које стране. Уколико је збир са леве стране мањи или једнак од збира са десне стране увећавамо збир са његове стране за актуелну вредност и померамо индекс удесно, исто радимо уколико је мањи са десне стране за десну страну. У ситуацијама када су збирови једнаки ми добијамо нови максимални збир и чувамо га. Излазимо из петље када се поднизови "сретну".

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n; cin >> n;
    vector<int> a(n);

    for(int i = 0; i < n; i++) cin >> a[i];
    int z1 = 0, z2 = 0;
    int p = 0, q = n-1, maksZbir = 0;
    while(p < q) {
        if(z1 <= z2) z1 += a[p++];
```

```
        else z2 += a[q--];
        if(z1 == z2) maksZbir = z1;
    }

    cout << maksZbir << endl;

    return 0;
}
```

2. Никола жели да купи све јастуке у једној продавници јастука, али пошто је дошао пешака не мозе понети све јастуке од једном. Он може понети два или три јастука по једном доласку у продавницу. Како Никола узима све јастуке, власник продавнице је одлучио да му да специфичан попуст. Уколико Никола купује 2 јастука при доласку онда је попуст 500 динара, али ако купује 3 јастука онда је попуст 20% за та три јастука. Никола увек купује прве јастуке које му продавац донесе, али може да одлучи да ли ће купити 2 или 3 јастука. Написати програм који помаже Николи да прође што је јефтиније могуће при куповини свих јастука.

   Са стандардног улаза се добија број јастука у продавници $n \in [2, 100]$, а затим и цена за сваки јастук, оним редом којим их продавац доноси Николи. Цена једног јастука је број који је дељив са 5 и припада интервалу [1000,5000].

   На стандардни излаз исписати минимални износ којим Никола може купити све јастуке из продавнице.

Пример 1

```
Улаз:
5
1000 1000 1000 2000 2000
Излаз:
5500
```

Пример 2

```
Улаз:
5
1000 1500 2000 2500 3000
Излаз:
8000
```

Решење: Наивно решење за овај задатак би било да гледамо сваку могућу комбинацију јастука коју никола може да купи, 2 па 3 или 3 па 2 и која је најнижа цена у тим ситуацијама. То решење би било прихватљиво с обзиром на мало $n \leq 100$. Много лепше, а и ефикацније решење је динамичким програмирањем. Када Никола купује само два јастука он нема избора и мора њих да купи за попуст од 500 динара, а када купује 3 јастука не може да купи 2 јастука па тако евентуално да добије најбољу цену јер по спецификацији задатка он може да купи само 2 или само 3 јастука. ДП низ можемо формирати на почетку око тих случајева, прва два и прва три јастука. Даље гледамо када је цена мања, када је цена најниже цене за јастуке до предпоследњег у збиру цене за два јастука са попустом или цена најниже цене за јастуке до оног на позицији $i - 3$ у збиру са ценом са попустом за три последња јастука нижа и то нам је најбоља цена за куповину јастука до те позиције. На крају у дп низу на крајњој позицији нам је и најнижа цена јастука и тако смо помогли Николи у куповини јастука. ☺

```
#include <iostream>
#include <vector>
```

```
#include <climits>

using namespace std;

int main() {
    int n; cin >> n;
    vector<long long> a(n);
    vector<long long> dp(n, 0);
    for(int i = 0; i < n; i++) cin >> a[i];

    dp[0] = INT_MAX;
    dp[1] = a[0] + a[1] - 500;
    dp[2] = (a[0] + a[1] + a[2]) * 4 / 5;

    for(int i = 3; i < n; i++) {
        long long prvi = dp[i - 2] + a[i - 1] + a[i] - 500;
        long long drugi = dp[i - 3] + (a[i - 2] + a[i - 1] + a[i]) * 4 / 5;
        dp[i] = min(prvi, drugi);
    }

    cout << dp[n - 1] << endl;

    return 0;
}
```

3. За дати број k и низ од n целих бројева. Потребно је проверити да ли је могуће добити број k од елемената из низа тако што крећемо од елемента на индексу 0, бирамо да ли га додајемо или одузимамо од тренутне суме, затим идемо на индекс који има вредност тренутне суме и исто то радимо са тим елементом. Сваки елемент низа мозе се користити највише једном за једно k.

   Са стандардног улаза се добијају број и $n \in [0, 10^5]$, а затим и n целих бројева из интервала $[10, 10]$. Затим се добија 5 бројева који представљају разлицте упите k.

   За сваки упит испиисати ниску 'MOGUCE' ако је могуће добити број k на претходно описан начин, иначе исписати ниску 'NIJE MOGUCE'.

Пример 1

```
Улаз:
6
3 7 -2 -1 3 1
8
100
7
6
50
Излаз:
MOGUCE
NIJE MOGUCE
MOGUCE
NIJE MOGUCE
NIJE MOGUCE
```

Пример 2

```
Улаз:
10
8 2 3 7 6 5 1 2 1 9
9
13
1000
```

```
9
3
Излаз:
MOGUCE
NIJE MOGUCE
NIJE MOGUCE
MOGUCE
NIJE MOGUCE
```

Решење: Са обзиром на то да ми не можемо да проверимо по упиту да ли смо ми дошли до те вредности на тражени начин морамо да приступимо задатку на други начин. Најбоље нам је да кренемо од почетка, да додамо и одузмемо број на првој позицији, даље за обе суме проверимо да ли је индекс у границама низа, ако јесте одузмемо и саберемо вредност на суму и у оба случаја уђемо дубље у рекурзију. Сваку суму коју добијемо на том путу можемо да упишемо у скуп да знамо да је упит могућ. На крају након краја рекурзије проверавамо упите и исписујемо тражени одговор у зависности од присуства вредности у скупу. (У коду је коришћена мапа са логичким вредностима, али боље је користити скуп)

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
#include <unordered_map>

using namespace std;

unordered_map<int, bool> moguceSume;

void proveri(int sum, const vector<int> &a, unordered_set<int> &posecenjo) {
    if(sum < 0 || sum >= a.size()) return;
    int add = sum + a[sum];
    int sub = sum - a[sum];
    if(posecenjo.find(add) == posecenjo.end()) {
        posecenjo.insert(add);
        moguceSume[add] = true;
        proveri(add, a, posecenjo);
        posecenjo.erase(add);
    }

    if(posecenjo.find(sub) == posecenjo.end()) {
        posecenjo.insert(sub);
        moguceSume[sub] = true;
        proveri(sub, a, posecenjo);
    }
}

int main() {
    int n; cin >> n;
```

```
    vector<int> a(n);
    for(int i = 0; i < n; i++) cin >> a[i];

    unordered_set<int> poseceno;
    proveri(0, a, poseceno);

    for(int i = 0; i < 5; i++) {
        int k; cin >> k;
        if(moguceSume[k]) cout << "MOGUCE" << endl;
        else cout << "NIJE MOGUCE" << endl;
    }

    return 0;
}
```

4. На омоту чоколаде са лешником пише да произвођач гарантује да се у квадрату од $a \cdot a$ коцкица сигурно налази бар један лешник. Никола је купио једну чоколаду и занима га који је највећи квадрат који може пронаћи тако да се у њему не налази лешник. Написати програм који налази највећи квадрат сачињен од $x \cdot x$ коцкица тако да не садржи ни један лешник. Временска сложеност треба бити $O(n^2)$, а просторна сложеност $O(n^2)$.

   Са стандардног улаза се учитава цео број n, а затим и n^2 бројева (0 или 1) који представљају да ли одређена коцкица чоколаде садржи лешник или не.

   Исписати један број, који представља дужину странице највећег траженог квадрата.

Пример 1

```
Улаз:
3
0 1 0
1 0 0
0 0 1
Излаз:
1
```

Пример 2

```
Улаз:
4
1 0 0 1
0 0 1 1
0 0 0 0
1 1 1 1
Излаз:
2
```

Решење: Овај задатак такође решавамо динамичким програмирањем. Уколико је присутан лешник ништа нећемо радити. Уколико смо на левој или горњој ивици и нема лешника свакако у ДП матрицу уписујемо један јер не може више поља дотле да буде без лешника (већи квадрат). У остатку матрице уколико нема лешника на пољу налазимо минимум поља изнад, лево и изнад лево дијагонално од тренутног. Величина макисмалног квадрата без лешника на том пољу је збир минимума са јединицом јер ако на пример имамо изнад квадрат 5·5, а лево један само првог реда, ништа нам не значи овај који је изнад у формирању новог целог квадрата и зато је ту онда величина највећег празног квадрата 2. На крају исписујемо максималну вредност коју смо уписали у ДП матрицу. У ДП матрици вредност на пољу је вредност максималног квадрата без лешника који се завршава (доњи десни угао је) у том пољу.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int n; cin >> n;
    vector<vector<int>> cokolada(n, vector<int>(n, 0));
    vector<vector<int>> dp(n, vector<int>(n, 0));

    int maksDp = 0;

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> cokolada[i][j];
            if(cokolada[i][j]) continue;
            if(i == 0 || j == 0) dp[i][j] = 1;
            else dp[i][j] = min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]})
            + 1;
            maksDp = max(maksDp, dp[i][j]);
        }
    }

    cout << maksDp << endl;

    return 0;
}
```