Увод у алгоритме - Рок 1 - 2025

Ово је испит из предмета увод у алгоритме из првог рока 2025. године преузет са странице `https://jason.matf.bg.ac.rs/test/ispitni%20rok%201/01%20Asteroidi`. 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. Научници посматрају систем звезде "Ецнус" који је специфичан по томе што су му све планете поређане у правој линији. Такође, свака планета има по један сателит који кружи по орбити која је облика кружнице. С времена на време, научници детектују присуство астероида у систему, при чему се и они увек појављују на линији на којој леже планете.

   Познате су координате свих планета у систему (може се сматрати да све планете леже на x оси координатног система) и полупречници орбита њихових сателита. Потребно је за сваки детектовани астероид испитати унутар колико орбита се налази. Временска сложеност треба да буде $O(nlogn+mlogn)$, а просторна сложеност $O(n)$, при чему је n број планета, а m број астероида.

   Са стандардног улаза се уноси природан број n ($1 \leq n \leq 10000$). Након тога, уносе се подаци о планетама у облику c r (реални бројеви), где је c координата центра планете, а r полупречник орбите њеног сателита. Затим се уноси природан број m ($1 \leq m \leq 1000$), а након тога и m упита облика a, где је a координата на којој је детектован астероид.

   За сваки упит исписати колико постоји орбита у којима се детектовани астероид налази.

Пример 1

```
Улаз:
3
1 2.5
6 3.5
7 1.25
4
6.75
0.35
14.2
3.15
Излаз:
2
1
0
2
```

Пример 2

```
Улаз:
4
2 1.5
4 2.5
7 10.5
11 2
3
2.1
12.9
17.5
Излаз:
3
2
1
```

Решење: Да израчунамо у колико се орбита налази неки астероид, нама су потребне леве и десне ивице орбита планета и онда ћемо број астероида рачунати као разлику броја орбита које су почеле до тада (орбите чија је лева ивица мања од

координате астероида) и броја оних које се завршавају пре координате астероида. Да бисмо ефикасно израчунали ову вредност за сваки упит прво је потребно сортирати векторе са левим и десним ивицама орбита. Након тога примењујемо функцију `lower_bound` за сваку координату астероида у оба низа. Приметимо да смо за број отворених увећали за вредност, а за број затворених смањили вредност координате у аргументу функције, то смо урадили да бисмо постигли да буду укључене све орбите са мањом десном ивицом, односно укључени сви крајеви орбита до те тачке (без ње) јер рачунамо да је астероид у орбити иако је тачно на ивици.

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

using namespace std;

int main() {
    int n; cin >> n;
    double c, r;
    vector<double> l(n);
    vector<double> d(n);

    for(int i = 0; i < n; i++) {
        cin >> c >> r;
        l[i] = c - r;
        d[i] = c + r;
    }

    sort(l.begin(), l.end());
    sort(d.begin(), d.end());

    int m; cin >> m;
    double tmp;
    for(int i = 0; i < m; i++) {
        cin >> tmp;
        auto opened = lower_bound(l.begin(), l.end(), tmp + 0.000001);
        auto closed = lower_bound(d.begin(), d.end(), tmp - 0.000001);
        int rez = distance(l.begin(), opened) - distance(d.begin(), closed);
        cout << rez << endl;
    }

    return 0;
}
```

2. Бинарна пирамида се формира на следећи начин: на врху се налази 0, а сваки следећи ред се добија тако што се свака цифра 0 из претходног реда замени са 0 1, а свака цифра 1 са 1 0.

   За сваки од м упита одредити k-ту цифру n-тог реда пирамиде. Индекси редова и цифара крећу од 1. Временска сложеност треба да буде O($m \cdot n$), а просторна O(1).

Са стандардног улаза се учитавају број m ($1 \leq m \leq 1000$), а затим и m упита облика n k.

На стандардни излаз за сваки од m упита исписати k-ту цифру n-тог реда пирамиде.

Пример 1

```
Улаз:
3
2 1
3 3
4 7
Излаз:
0
1
0
```

Пример 2

```
Улаз:
4
4 16
3 2
10 1
53 76
Излаз:
1
1
0
0
```

Решење: Из поставке задатка можемо видети да је увек са леве стране испод неке цифре иста та цифра, а измена само када се иде на десну страну. Решење је у томе да се враћамо уназад до врха пирамиде од n-тог реда до првог, адекватно k је пола од оног тренутног (јер се број елемената у реду увек дуплира). Такође гледамо да ли је у питању било спуштање удесно, на то нам указује парност од k. На крају само узмемо остатак при дељењу са два броја спуштања удесно. То ради јер крећемо од нуле и једно спуштање удесно значи да прелазимо на јединицу, а још једно значи да се враћамо на нулу, итд.

```
#include <iostream>

using namespace std;

int main() {
    int m, n, k;
    cin >> m;

    for(int i = 0; i < m; i++) {
        cin >> n >> k;
        int desna = 0;

        while(n > 1) {
            n--;
            desna += (k % 2 == 0) ? 1 : 0;
            k = (k + 1) / 2;
        }

        cout << (desna % 2) << endl;
    }

    return 0;
```

```
}
```

3. У књижари је монтирана нова полица. Радница је добила задатак да поређа н књига из серијала "ASCII" на полицу. Све књиге у овом серијалу су специфичне по томе што им је наслов само један карактер. Једино што јој је битно приликом ређања је да се књига под називом "@" налази пре књиге под називом "#".

   Са стандардног улаза се уноси број књига n ($1 \leq n \leq 8$), а затим и n карактера који представљају наслове књига. Подразумевати да књиге "@" и "#" постоје међу књигама.

   Исписати све могуће редоследе књига, лексикографски растуће.

Пример 1

```
Улаз:
4
A B # @
Излаз:
@ # A B
@ # B A
@ A # B
@ A B #
@ B # A
@ B A #
A @ # B
A @ B #
A B @ #
B @ # A
B @ A #
B A @ #
```

Пример 2

```
Улаз:
4
$ # @ !
Излаз:
! $ @ #
! @ # $
! @ $ #
$ ! @ #
$ @ ! #
$ @ # !
@ ! # $
@ ! $ #
@ # ! $
@ # $ !
@ $ ! #
@ $ # !
```

Решење: Ово је класичан задатак генерисања комбинација. На почетку када учитамо све, сортирамо низ карактера јер треба да испишемо распореде лексикографски растуће. У рекурзивном позиву поставимо базни случај да када смо попунили сва поља испишемо комбинацију. Иначе пролазимо кроз све књиге у низу. Подешавамо логичку променљиву уколико испунимо специјални услов, и прескачемо итерацију уколико га није могуће испунити са књигом која је на реду. Додајемо тај карактер у комбинацију и бришемо додату књигу из вектора, пре него што позовемо са измењеним векторима процедуру поново.

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

using namespace std;

void generisi(int k, int n, vector<char> knjige, vector<char> kombinacija, bool bi
    if(k == n) {
        for(char x : kombinacija) cout << x << " ";
```

```
        cout << endl;
        return;
    }
    kombinacija.push_back(' ');
    for(int i = 0; i < knjige.size(); i++) {
        if(knjige[i] == '#' && !bioAt) continue;
        if(knjige[i] == '@') bioAt = true;
        kombinacija[k] = knjige[i];
        vector<char> knjigeNov = knjige;
        knjigeNov.erase(knjigeNov.begin() + i);
        generisi(k + 1, n, knjigeNov, kombinacija, bioAt);
        if(knjige[i] == '@') bioAt = false;
    }
}

int main() {
    int n; cin >> n;
    vector<char> knjige(n);

    for(int i = 0; i < n; i++) cin >> knjige[i];

    sort(begin(knjige), end(knjige));
    generisi(0, n, knjige, {}, false);

    return 0;
}
```

4. Веверица се налази у горњем левом ћошку дворишта квадратног облика. Она жели да се попне на дрво које се налази у доњем десном ћошку дворишта и успут да покупи барем један жир, при чему жели да се креће само доле или десно.

   Написати програм који исписује број могућих путања којима веверица може доћи до дрвета. Временска и просторна сложеност треба да буду $O(n^2)$

   Са стандардног улаза се учитава димензија странице дворишта n, а затим и матрица димензија $n \cdot n$ чији су елементи 0 или 1. Уколико је елемент 1 на том месту у дворишту се налази жир.

   Исписати број могућих путања којим веверица може доћи до дрвета крећући се доле или десно, а да притом покупи барем један жир.

Пример 1

```
Улаз:
4
0 0 1 0
1 0 0 0
0 1 0 0
1 0 1 0
Излаз:
18
```

Пример 2

```
Улаз:
4
0 1 0 0
0 0 1 1
1 0 1 1
0 0 0 0
Излаз:
19
```

Решење: Решење задатка се крије у томе да израчунамо на колико начина веверица може да стигне до краја без да покупи један једини жир (☹ јадна веверица у том случају). Када израчунамо тај број, њега одузмемо од броја свих осталих путања до доњег десног ћошка матрице. То рачунамо тако што у матрицу путања ставимо збир путања до поља пре (лево и горе) то је и број путања до те тачке, с обзиром на то да се веверица може кретати десно и доле. Матрицу путања где избегавамо поља са жиром рачунамо на исти начин само што ставимо нулу на поља на којима је жир. (такође је потребно узети у обзир специјалне случајеве у рачуну поља на левој и горњој ивици матрице)

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

using namespace std;

int main() {
    int n; cin >> n;
    vector<vector<long long>> koraci(n, vector<long long>(n, 0));
    vector<vector<long long>> koraciBZ(n, vector<long long>(n, 0));

    bool tmp;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> tmp;
            if(i == 0 && j == 0) {
                koraci[i][j] = 1;
                koraciBZ[i][j] = 1;
                continue;
            }
            if(i == 0) {
                koraci[i][j] = koraci[i][j - 1];
                koraciBZ[i][j] = tmp ? 0 : koraciBZ[i][j - 1];
            } else if(j == 0) {
                koraci[i][j] = koraci[i - 1][j];
                koraciBZ[i][j] = tmp ? 0 : koraciBZ[i - 1][j];
            } else {
                koraci[i][j] = koraci[i - 1][j] + koraci[i][j - 1];
                koraciBZ[i][j] = tmp ? 0 : koraciBZ[i - 1][j] + koraciBZ[i][j - 1]
            }
        }
    }

    long long rez = koraci[n - 1][n - 1] - koraciBZ[n - 1][n - 1];
    cout << rez << endl;

    return 0;
}
```