Решења задатака - Изборно такмичење

8. јун 2024.

Најбољи збир

Аутор: Огњен Тешић

Идеја је да се користе префиксне суме за брзо израчунавање збира елемената поднизова и збира елемената које треба искључити. Прво, израчунава се префиксна сума pref, где pref[i] представља збир свих елемената од почетка низа до позиције i. Даље, израчунава се префиксна сума p2 „по модулу", где p2[i] представља збир свих елемената који се морају искључити (сваки S-ти елемент) од почетка низа до позиције i.

На пример, ако је $S = 3$, онда p2[i] садржи збир свих елемената на позицијама $j < i$ и притом i и j дају исти остатак при дељењу са 3. Затим, за сваки могући подниз дужине U, рачуна се збир тог подниза користећи префиксну суму pref, и одузимајући збир елемената који се морају искључити користећи префиксну суму p2. Пошто је потребно одузети збир сваког S-тог елемента у поднизовима, p2 омогућава брзо добијање ових вредности без потребе за поновним рачунањем.

На крају, проласком кроз све могуће поднизове дужине U и израчунавањем збирова се проналази највећи збир. Сложеност овог решења је $\mathcal{O}(N)$.

Четири делиоца

Аутор: Огњен Тешић

Ако је растављање броја x на просте чиниоце $p_1^{k_1} \cdots p_m^{k_m}$, тада је број делилаца броја x једнак $(k_1+1)\cdots(k_m+1)$. Пошто је по условима задатка број делилаца броја x мора бити 4, мора да важи да је $m=1$ и $k_1+1=3$ тј. $x=p^3$, за неки прост број p или да је $m=2$ и $k_1+1=2$ и $k_2+1=2$, тј. $x=p\cdot q$ за неке просте бројеве p и q $(p\neq q)$.

Пошто мора да важи да је најмањи прост чинилац већи од 100, важи $p\geq 101$, па је $q\leq \frac{x}{101}$. Највећа могућа вредност броја x који испитујемо је b па је потребно одредити све просте бројеве који су већи или једнаки 101, а мањи или једнаки $\frac{b}{101}$. То ефикасно можемо урадити Ератостеновим ситом. Њих чувамо у сортираном низу.

Нека је $f(n)$ број таквих бројева који су мањи или једнаки n. Како бисмо одредили колико је таквих бројева у интервалу $[a,b]$ можемо користити формулу $f(b)-f(a-1)$.

Да одредимо вредност функције $f(n)$ примењујемо следећи алгоритам:
Бројеве облика p^3 налазимо тако што пролазимо кроз просте бројеве све док је $p^3\leq n$.
Бројеве облика $p\cdot q$ налазимо тако што итерирамо по броју p. Без умањења општости можемо претпоставити да је $p<q$. Сада је за свако p број бројева који могу бити q једнак броју претходно нађених простих бројева у интервалу $[p+1,\frac{n}{p}]$. Број таквих бројева можемо наћи бинарном претрагом, или техником два показивача.

Најспорији део овог алгоритма је само Ератостеново сито, пошто пролаз кроз све просте бројеве је бржи него пролаз кроз све бројеве. Нека је $m=\frac{b}{100}$. Сложеност решења је $\mathcal{O}(m\cdot\log(\log m))$.

Слојеви

Аутор: Милан Вугделија

Да бисмо за сваку тачку могли брзо да израчунамо слој коме она припада, потребно је да тачке буду у неком погодном редоследу. Један такав редослед је опадајући по x координати. На међусобни однос и редослед тачака са истом x координатом вратићемо се ускоро. Након сортирања, прва тачка на коју наилазимо је максимална (не постоји тачка са већом x координатом), па она припада првом слоју. Претпоставимо да након тога наилазимо на тачку са мањом x и мањом y координатом. Та тачка, јасно, припада другом слоју. Да бисмо за неку следећу тачку, која има још мање x, могли да одредимо којем слоју припада, довољно је да за сваки започети слој s знамо највећу y координату међу тачкама за које је већ установљено да припадају слоју s. Нека је за први слој тај маскимум $ymax_0$, за други слој $ymax_1$ итд. Приметимо да низ $ymax$ мора да буде строго опадајући, што следи из дефиниције слојева.

Даље, нека нова тачка на коју наилазимо (звaћемо је текућа тачка) има координате x и y. Уколико за неки слој s важи $ymax_{s-1} > y$, то значи да текућом тачком доминира нека тачка слоја s. Према томе, текућа тачка не припада слоју s, нити било ком ранијем слоју (са мањим редним бројем). Ако при томе још за следећи слој $s+1$ важи $ymax_s \leq y$, онда ниједна од досадашњих тачака из слоја $s+1$ не доминира текућом тачком. Наредне тачке које имају исту x координату као текућа, или мању, такође не доминирају текућом тачком. Према томе, текућа тачка припада слоју $s+1$.

Уколико постоји више тачака са истом x координатом, оне све припадају истом слоју. Ако бисмо од две такве тачке прво обрадили ону са већом y координатом, поредећи y координату следеће тачке само са максималном y координатом слоја коме припадају обе те тачке, могли бисмо да дођемо до погрешног закључка да друга тачка не припада истом слоју као прва. Да не бисмо компликовали прогргам узимањем у обзир и x координата (које би биле битне само код тачака са истом x координатом), боље је да тачке са истом x координатом обрађујемо почевши од оне са најмањом y координатом. Према томе, приликом почетног сортирања тачака опадајуће по x, тачке са истом x координатом треба да сортирамо растуће по y. При оваквом редоследу обраде тачака, када за текућу тачку пронађемо први слој $s+1$, чија је максимална y координата $ymax_s$ мања или једнака y координати текуће тачке, треба још да запамтимо да текућа тачка припада слоју $s+1$ и да ажурирамо $ymax_s$ на нови максимум.

Како је низ $ymax$ строго опадајући, проналажење траженог слоја може да се обави бинарном претрагом. То нас доводи до решења, коме је за сваки задатак потребно време сразмерно са $n \cdot \log n$.

Решења подзадатака су једноставнија од главног решења, па их нећемо објашњавати једнако детаљно.

1 Решење случаја када има до 50 тачака

Тачке којима још није одређен слој звaћемо необрађене тачке. Када је укупан број тачака довољно мали, тачке можемо да обрађујемо слој по слој. За сваку необрађену тачку проверавамо да ли нека друга необрађена тачка њом доминира. Уколико се не нађе тачка која полазном тачком домирира, полазној тачки додељујемо текући слој, а у противном је остављамо за касније и памтимо да су неке тачке остале необрађене. Овај поступак можемо да пнављамо док има необрађених тачака. Време потребно за решење једног задатка на овај начин сразмерно је са $n^2 \cdot s$, где је n број тачака, а s број слојева.

2 Решење случаја када има до 5 слојева

Претходно решење, у коме се тачке обрађују слој по слој, може да се побољша на много начина. Једно једноставно побољшање је да тачке прво сортирамо као у главном решењу. Тада се у једном пролазу кроз низ тачака, за сваку тачку брзо одређује да ли је она максимална (не рачунајући обрађене тачке), тј. да ли припада текућем слоју. Време потребно за решење једног задатка на овај начин сразмерно је са $n \cdot s$, где је n број тачака, а s број слојева.

3 Решење случаја када свака тачка једног слоја доминира свим тачкама наредног слоја

И у овом случају је корисно да тачке сортирамо као у главном решењу. Уз ово додатно ограничење, тачке ће у низу већ бити распоређене по слојевима. Почетак новог слоја можемо да препознамо по томе што текућом тачком доминира тачка која јој претходи.

4 Решење случаја када има до 2 тачке по слоју

Ако се у слоју налази само једна тачка, она мора да има максимално x и максимално y међу свим необрађеним тачкама. Слично томе, ако се у слоју налазе две тачке, онда једна од њих мора да има максимално x, а друга максимално y, не рачунајући већ обрађене тачке. Приметимо да ово не искључује случај када је једна од тачака максимална по обе координате, или случај када су обе тачке максималне по обе координате (различите тачке могу да имају обе координате једнаке). Да бисмо могли брзо да пронађемо тачке са максималном x, односно y координатом, можемо да користимо два реда са приоритетом (хипа), од којих нам један враћа тачке опадајуће по x, а други опадајуће по y.

Приметимо још само да приликом провере да ли смо у текућем слоју нашли једну или две тачке, поредимо редне бројеве тачака а не координате, јер ни једнакост тачака по обе координате не гарантује да се ради о истој тачки.

Мешалица 4

Аутор: Огњен Нешковић

```cpp
#include <algorithm>
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int dfs(int start_node, vector<bool>& visited, const vector<vector<int>>& graph) {
    stack<int> stack;
    stack.push(start_node);
    visited[start_node] = true;

    int component_size = 0;

    while (!stack.empty()) {
        int node = stack.top();
        stack.pop();
        component_size++;

        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                stack.push(neighbor);
            }
        }
    }

    return component_size;
}

int main()
{
        ios::sync_with_stdio(false);
        cin.tie(0);

        int n, k, cnt_perms; cin >> n >> k >> cnt_perms;
        vector<vector<int>> graph(n, vector<int>());
        for (int i = 0; i < cnt_perms; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        int x; cin >> x; x--;
                        graph[j].push_back(x);
                        graph[x].push_back(j);
                }
        }

        vector<bool> visited(n);
        vector<int> component_sizes;
        for (int i = 0; i < n; i++)
                if (!visited[i])
                        component_sizes.push_back(dfs(i, visited, graph));

        vector<bool> possible_sum(n + 1);
        possible_sum[0] = 1;
        for (int i = 0; i < component_sizes.size(); i++)
        {
                vector<bool> possible_sum_next = possible_sum;
                int x = component_sizes[i];
                for (int j = 0; j <= n; j++)
                        if (possible_sum[j] && x + j <= n)
                                possible_sum_next[x + j] = 1;

                possible_sum = possible_sum_next;
        }

        int solution = INT_MAX;
        for (int i = 0; i <= n; i++)
                if (possible_sum[i])
                        solution = min(solution, abs(i - k));
        cout << solution;

        return 0;
}
```