A-III-4 Diaľnica v Slovakistane

Budeme pracovať priamo s intervalmi častí – presnejšie, budeme si v nejakej vhodne zvolenej dátovej štruktúre udržiavať množinu aktuálne rozostavaných úsekov.
Pri každom pridaní nového intervalu budeme potrebovať túto množinu upraviť. Ak sa nový interval ani len nedotýka so žiadnym už skôr rozostavaným úsekom, len ho do množiny pridáme ako nový úsek. Ak sa s nejakými úsekmi dotýka alebo prekrýva, budeme ich potrebovať zjednotiť a nahradiť jedným novým úsekom.
Samozrejme, na to potrebujeme nájsť, kde sa nový interval nachádza vzhľadom na už existujúce úseky. Ak by sme si našu množinu úsekov pamätali ako obyčajný zoznam (či už usporiadaný alebo nie), spracovanie i-teho intervalu by trvalo $O(i)$ krokov a dokopy by sme dostali časovú zložitosť $O(q^2)$, resp. presnejšie $O(n+q\min(n,q))$.
Šikovnejšie bude použiť usporiadanú množinu (napr. set v C++), v ktorej budú aktuálne úseky zoradené podľa ich začiatkov (a teda aj koncov).
Vďaka tomu vieme v logaritmickej zložitosti zistiť pozíciu, na ktorú pridať nový interval – presnejšie, nájsť v aktuálnej množine buď miesto, kam ho vieme vložiť (ak sa so žiadnym existujúcim intervalom ani nedotýka), alebo prvý z úsekov, ktoré sa s práve pridávaným intervalom dotýkajú alebo prekrývajú.
V prvom prípade sme zjavne nový interval spracovali v logaritmickom čase a sme spokojní.
Čo s druhým prípadom? Úsek, ktorý sme našli, z množiny odstránime a náš interval nahradíme zjednotením s ním. Následne pokračujeme ďalším susedným úsekom, a to až do momentu, kým neprestanú mať s pridávaným intervalom prienik alebo dotyk.

Pri implementácii nám vedia pomôcť takzvané sentinely[1]. Na úplnom začiatku si do našej množiny vložíme úseky $[-2,-2]$ a $[n+2,n+2]$. Jeden z nich je úplne naľavo od diaľnice, druhý úplne napravo a je jasné, že sa s našimi intervalmi nikdy nebudú prekrývať. Vďaka nim nemusíme špeciálne ošetrovať prípady, keď sa naľavo alebo napravo od nášho intervalu už žiaden iný interval nenachádza a nemáme ho s čím porovnať.

Výsledná časová zložitosť takéhoto riešenia je $O(q \log q)$.
Pozor, nie je pravda, že každý z q intervalov pridáme v čase $O(\log q)$ – niektoré konkrétne intervaly budeme spracúvať dlho, lebo ich bude treba spájať s veľa existujúcimi úsekmi.
Pozrime sa ale na celý beh nášho algoritmu naraz. Pri spracovaní každého z q intervalov najskôr niekoľkokrát (občas vôbec, občas veľakrát) nejaký úsek z množiny odstránime a potom práve jeden nový úsek (zjednotenie nového intervalu so všetkými odstránenými úsekmi) vložíme. Dokopy za celý beh algoritmu teda do množiny len q-krát niečo vložíme. To ale znamená, že dokopy za celý beh algoritmu môžeme z množiny najviac q-krát niečo odstrániť. A keďže každú operáciu s množinou spravíme v čase logaritmickom od jej počtu prvkov, dostávame vyššie sľubovanú časovú zložitosť.

Listing programu (C++)

```
#include <cstdio>
#include <algorithm>
#include <set>
using namespace std;

set<pair<int, int> > useky;

auto spoj_intervaly(pair<int, int> prvy, pair<int, int> druhy) {
    pair<int, int> spojeny = {min(prvy.first, druhy.first), max(prvy.second, druhy.second)};
    useky.erase(prvy);
    useky.erase(druhy);
    useky.insert(spojeny);
    return useky.find(spojeny);
}

int main() {
    int n, q;
    scanf("%d %d", &n, &q);
    useky.insert({-2, -2});
    useky.insert({n + 2, n + 2});

    for(int i = 0; i < q; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        useky.insert({a, b + 1});
        auto novy = useky.find({a, b + 1});
```

[1] https://en.wikipedia.org/wiki/Sentinel_value

```
        while(true) {
            auto dalsi = next(novy);
            if(dalsi->first <= novy->second) {
                novy = spoj_intervaly(*novy, *dalsi);
            }
            else break;
        }

        while(true) {
            auto pred = prev(novy);
            if(novy->first <= pred->second) {
                novy = spoj_intervaly(*novy, *pred);
            }
            else break;
        }

        printf("%ld\n", useky.size() - 2);
    }
}
```

A-III-5 Planéta B-647

Všetky indexy v tomto riešení treba čítať modulo n, teda napríklad vrchol n je to isté ako vrchol 0.
Pre každý vrchol i môžeme definovať $d_i = t_i - t_{i+n/2}$: rozdiel medzi jeho teplotou a teplotou oproti. Hľadáme ľubovoľný vrchol, pre ktorý $d_i = 0$.
Keďže susedné hodnoty t_i sa vždy líšia práve o 1, ľahko nahliadneme, že susedné hodnoty d_i sú buď rovnaké alebo sa líšia práve o 2.
Ak by $n/2$ bolo nepárne, mali by hodnoty t_i a $t_{i+n/2}$ opačnú paritu a teda všetky hodnoty d_i by boli nepárne a riešenie by neexistovalo. My však máme $n = 100\,000$, a keďže $n/2$ je párne, t_i a $t_{i+n/2}$ majú rovnakú paritu a teda všetky d_i sú párne.
Všimnime si teraz, že $d_{i+n/2} = -d_i$: počítame rozdiel tých istých dvoch teplôt, len v opačnom poradí.
Predstavme si, že začneme v nejakom vrchole i. Ak $d_i = 0$, sme hotoví. Inak bez ujmy na všeobecnosti predpokladajme, že $d_i > 0$. Teraz pôjdeme po obvode planéty z vrcholu i do vrcholu $i + n/2$ a budeme sledovať, ako sa mení hodnota d. Na začiatku našej cesty je kladná, na jej konci je presne opačná, teda záporná. No a keďže v každom kroku sa hodnota d môže zmeniť len o presne 2, je zjavné, že cestou musela nadobudnúť všetky možné párne hodnoty medzi d_i a $-d_i$. Špeciálne teda niekedy po ceste musela byť táto hodnota nulová.
Z vyššie uvedenej úvahy vyplýva, že pre $n = 4k$ má naša úloha vždy nejaké riešenie – musí totiž existovať dvojica oproti sebe ležiacich vrcholov s rovnakou teplotou.
Navyše si môžeme rozmyslieť, že na vyššie popísanú úvahu nebolo nijak potrebné, aby začiatok a koniec našej cesty boli oproti sebe. Ak by sme mali ľubovoľné dva vrcholy i a j také, že $d_i > 0$ a $d_j < 0$, cestou z i do j nutne musíme stretnúť vrchol k pre ktorý $d_k = 0$.
Toto pozorovanie nám umožní použiť binárne vyhľadávanie. Začneme s intervalom od 0 po $n/2$. V každej iterácii sa potom pozrieme do stredu aktuálneho intervalu. Nech ide o vrchol m. Položíme dve otázky na teplotu (zistíme si t_m a $t_{m+n/2}$), z ktorých sa dozvieme hodnotu d_m. Ak $d_m = 0$, máme riešenie. V opačnom prípade zmenšíme interval hľadania na polovicu: m bude jeden koniec a druhý zoberieme tak, aby na ňom d mala opačné znamienko ako vo vrchole m.
Takéto riešenie stačilo na zisk 9 bodov. Posledný bod sa dal získať za pozorovanie, že v poslednej iterácii už nepotrebujeme explicitne zistiť hodnotu d: ak nám ostal len jeden kandidát, musí to byť on.

Listing programu (C++)

```
#include <bits/stdc++.h>

using namespace std;


int n = 100000;

int get_b (int i) {
        assert(i < n/2);
        int t1, t2;
        cout << "0␣" << i << endl << flush;
        cin >> t1;
        cout << "0␣" << i + n/2 << endl << flush;
```

```
        cin >> t2;
        return t1 - t2;
}

bool same_sgn (int x, int y) {
        return (x < 0 && y < 0) || (x > 0 && y > 0);
}

int main() {
        int l = 0, r = n/2;
        int b0 = get_b(l);
        if (b0 == 0) {
                cout << "1␣" << 0 << endl << flush;
                return 0;
        }

        while (r - l > 2) {
                int m = (r+l)/2;
                int b = get_b(m);
                if (b == 0) {
                        cout << "1␣" << m << endl << flush;
                        return 0;
                }
                if (same_sgn(b, b0)) {
                        l = m;
                        b0 = b;
                } else {
                        r = m;
                }
        }

        cout << "1␣" << l+1 << endl;

}
```

Bonusová otázka na zamyslenie

V spojitej verzii tejto úlohy nepotrebujeme celú planétu. Rovnakou úvahou ako v tomto vzorovom riešení sa dá dokázať, že keď máme planétu tvaru gule, na ktorej povrchu sa spojito mení teplota, tak niekde *na jej rovníku* vieme nájsť dva oproti sebe ležiace body s rovnakou teplotou.
Uvažujme teraz planétu tvaru gule, na povrchu ktorej sa aj teplota aj tlak vzduchu spojito menia. Musia vždy niekde na povrchu planéty existovať dva oproti sebe ležiace body ktoré majú *zároveň* aj rovnakú teplotu aj rovnaký tlak vzduchu?

A-III-6 Marshmallows

V prvej podúlohe si môžeme dovoliť vyskúšať všetky možné podmnožiny vrcholov, o každej overiť, či určuje súvislý podstrom, a ak áno, akú má kvalitu. Toto vieme elegantne implementovať pomocou bitových masiek: postupne budeme iterovať cez čísla od 0 do $2^n - 1$, pričom pre konkrétne číslo m vyberieme tie vrcholy, ktorých čísla zodpovedajú pozíciám jednotkových bitov v m.

Kvadratické riešenie

Ukážeme si, ako v lineárnom čase nájsť odpoveď pre konkrétnu polohu cenovky. Strom si zakoreníme vo vrchole s cenovkou. Pre každý podstrom si teraz chceme spočítať, akú najväčšiu kvalitu z neho vieme dostať, ak vyberieme (aspoň) jeho koreň. Toto spravíme rekurzívne – v podstate prehľadaním stromu do hĺbky.
Pre listy je to jednoduché: samotný list vybrať musíme a pod ním už nič nie je, takže odpoveď je 1 pre list s marshmallowom a -1 pre list s plastelínou.
Pre ostatné vrcholy to vypočítame pažravo: koreň podstromu musíme vybrať, a keďže výbery potomkov sú navzájom nezávislé, pre každého z nich môžeme zobrať jeho optimálne riešenie, resp. ho nezobrať vôbec, ak jeho optimálna hodnota bola záporná.
Pre konkrétnu polohu cenovky takto vieme zistiť optimálnu odpoveď v čase $O(n)$, dokopy má teda toto riešenie časovú zložitosť $O(n^2)$.

Vylepšenie pre malé stupne

Predstavme si, že sme vyššie popísaný algoritmus spustili pre jeden konkrétny koreň. Preň teraz poznáme optimálnu odpoveď. Pre ostatné vrcholy sme ale tiež spočítali niečo užitočné – optimálnu odpoveď, ak podstrom môžeme budovať len smerom dodola.
Čo nám chýba do toho, aby sme pre každý vrchol vedeli riešenie našej súťažnej úlohy? Zistiť, ako najlepšie z neho vieme podstrom rozšíriť smerom dohora. To teraz skúsime dopočítať.
Nech sme v nejakom vrchole x, ktorého optimálny podstrom hľadáme. Nech x má rodiča y. Ako prvé sa potrebujeme rozhodnúť, či y vyberieme do podstromu. Ak áno, tak môžeme nezávisle na sebe robiť dve veci: vybrať podstrom idúci nad y a vybrať podstromy smerujúce do ostatných detí vrcholu y (teda detí iných ako x).
Ak budeme vrcholy spracúvať od aktuálneho koreňa smerom dodola, vrchol y sme už spracovali pred vrcholom x, a teda vieme, aký najlepší podstrom vieme vybrať nad ním. Pre každého potomka y už z predchádzajúcej fázy riešenia vieme, aký najlepší podstrom vieme vybrať pod ním. No a z tohto už ľahko v čase $O(d)$ spočítame optimálny výber podstromu nad x. (Netreba pri tom zabudnúť na $+1$ alebo -1 za samotný vrchol y.)
Aj túto fázu riešenia vieme implementovať rekurzívnym prehľadávaním do hĺbky. V predchádzajúcom riešení sme najskôr rekurzívne spracovali všetkých potomkov vrcholu a až potom vrchol samotný, tu to musíme spraviť naopak – najskôr spracujeme vrchol a až potom sa postupne rekurzívne zavoláme na jeho potomkov.
Toto riešenie má časovú zložitosť $O(nd)$.

Lineárne riešenie

Od predchádzajúceho riešenia je už len kúsok k tomu optimálnemu. Nech s_v je optimálna kvalita podstromu vybratého z vrcholu v smerom dodola. V predchádzajúcom riešení sme potrebovali vedieť optimálnu kvalitu, ktorú vieme dostať výberom podstromov idúcich dodola do detí vrcholu y iných ako x. Zlepšenie bude spočívať v tom, že si uvedomíme, že toto nemusíme počítať v čase $O(d)$. Túto hodnotu totiž vieme v konštantnom čase vypočítať ako $s_y - \max(0, s_x)$.
Po tomto zlepšení už vieme všetko potrebné vypočítať v konštantnom čase pre vrchol, a teda dokopy dostávame riešenie s časovou zložitosťou $O(n)$.

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

const int NMAX = 11234567;

struct Nd
{
        vector<Nd *> e;
        int val;
        int opt, opt_up;

        int calc_up(Nd *from)
        {
                opt = val;
                for (Nd * it : e) if (it != from) {
                        opt += max(0, it->calc_up(this));
                }
                opt_up = opt;
                return opt;
        }

        void calc_down(Nd * from, int from_up)
        {
                opt += max(0, from_up);
                for (Nd * it : e) if (it != from) {
                        it->calc_down(this, opt - max(0, it->opt_up));
                }
        }
} nd[NMAX];

int main()
{
        int n;
        scanf("%d", &n);
        for (int i=0; i<n; i++) {
                char c;
                scanf("␣%c", &c);
```

```
                nd[i].val = c=='L' ? 1 : -1;
        }
        for (int i=0; i<n-1; i++) {
                int x, y;
                scanf("%d%d", &x, &y);
                nd[x].e.push_back(nd+y);
                nd[y].e.push_back(nd+x);
        }
        nd->calc_up(NULL);
        nd->calc_down(NULL, 0);
        for (int i=0; i<n; i++)
                printf("%d\n", nd[i].opt);
        return 0;
}
```

TRIDSIATY ÔSMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek, Jakub Šimo
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2023