P-III-1 Oplocení zahrádek

Nikde na pozemku se nebudou setkávat čtyři konce plotů (dva vodorovné a dva svislé). Kdyby tomu tak bylo, můžeme například oba vodorovné ploty spojit do jednoho a ušetříme tím jeden poplatek.

Představme si, že všechny poplatky seřadíme podle ceny od nejvyšší po nejnižší, v případě rovnosti zvolíme nějaké jedno konkrétní pořadí. Řádek nebo sloupec, který je umístěn dříve ve zvoleném pořadí, nazveme důležitější. Stejné označení použijeme i pro ploty v těchto řádcích.

Uvažujme libovolnou křižovatku řádku a sloupce. Klíčovým pozorováním je, že se nikdy nevyplatí, aby touto křižovatkou procházel plot v méně důležitém směru. Kdybychom totiž takový plot rozdělili na dva a naopak spojili bychom oba ploty v důležitějším směru, dostaneme alespoň stejně dobré řešení.

Vezměme si nyní optimální řešení, v němž výše uvedené pravidlo není nikde porušeno. Je zjevné, že tím je podoba optimálního řešení jednoznačně určena: v každém řádku/sloupci je plot přerušen přesně na těch místech, kde křižuje důležitější sloupce/řádky.

Na základě tohoto pozorování už dokážeme efektivně spočítat celkovou cenu poplatků. Potřebujeme jenom pro každý řádek/sloupec zjistit, kolik důležitějších sloupců/řádků ho křižuje. To uděláme tak, že si je všechny výše popsaným způsobem uspořádáme a potom je postupně zpracováváme, přičemž si pamatujeme, kolik řádků a kolik sloupců jsme už zpracovali.

Při dobré implementaci takto dostaneme řešení s časovou složitostí $\mathcal{O}((d+s)\cdot\log(d+s))$. Na asymptotickou časovou složitost nemá vliv to, zda uspořádáme ceny pro každý rozměr zvlášť, nebo všechny dohromady.

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

int main() {
    int D, S;
    cin >> D >> S;
    vector<int> svisle(D-1), vodorovne(S-1);
    vector< pair<int,int> > moznosti;
    for (int d=0; d<D-1; ++d) {
        cin >> svisle[d];
        moznosti.emplace_back(svisle[d],0);
    }
    for (int s=0; s<S-1; ++s) {
        cin >> vodorovne[s];
        moznosti.emplace_back(vodorovne[s],1);
    }
    sort( moznosti.begin(), moznosti.end() );
    reverse( moznosti.begin(), moznosti.end() );
```

```
    vector<int> zpracoval(2,0);
    int odpoved = 0;
    for (auto moznost : moznosti) {
        int cena = moznost.first, pocet = 1 + zpracoval[1-moznost.second];
        odpoved += cena * pocet;
        ++zpracoval[moznost.second];
    }
    cout << odpoved << endl;
}
```

Dodatečné komentáře

Na výše popsané řešení ve skutečnosti nemá žádný vliv poslední podmínka ze zadání: ta, že ploty je třeba stavět vždy tak, aby se žádný z nich nedal prodloužit. Naše řešení ji zjevně splňuje a zároveň z důkazu jeho správnosti vyplývá, že tato podmínka je zbytečná. Kdybychom ji v zadání neměli, optimální řešení by stále mělo stejnou cenu.

Díky této dodatečné podmínce je ale možné i bez výše uvedeného důkazu vytvořit 6-bodové řešení s časovou složitostí $\mathcal{O}(d^2s^2(d+s))$. Při tomto řešení si uvědomíme, že prvním postaveným plotem musí nutně být některý celý řádek nebo některý celý sloupec. Vyzkoušíme postupně všechny možnosti. Každá nám problém rozdělí na dva menší obdélníkové podproblémy téhož typu, které je třeba vyřešit každý zvlášť. To provedeme rekurzivně, přičemž výsledky memoizujeme. Jiný, ale ekvivalentní pohled je postup zdola nahoru (iterativní dynamické programování): postupně najdeme optimální řešení pro každou z $\mathcal{O}(d^2s^2)$ obdélníkových podoblastí vstupu, od nejmenších po největší.

Komplikovanější, ale také korektní důkaz tvrzení, na němž je založeno vzorové řešení, je možné provést matematickou indukcí. Přesněji, indukcí podle obsahu obdélníka dokážeme, že pro jakýkoliv obdélník s daným obsahem platí, že jako první můžeme postavit plot s maximálním poplatkem. Kdyby totiž bylo optimální jako první postavit jiný plot kolmý na ten, který chceme, z indukčního předpokladu víme, že v každé polovině pozemku je optimální jako první postavit kus našeho plotu. Nyní můžeme tyto první tři kroky bez zhoršení celkové ceny nahradit postavením našeho plotu a dvou částí toho druhého. Rozmyslete si, jak tento argument dokončit pro řešení začínající postavením jiného plotu rovnoběžného s tím, který chceme.

P-III-2 Horolezci

Triviální pozorování: Cena výstroje každého z horolezců bude rovna některé z výšek na vstupu. (Přesněji, maximu z výšek vrcholů, na které půjde.)

Základní netriviální pozorování: Vždy je optimální použít nejvýše tři horolezce.

Důkaz: Uvažujme libovolný plán expedice, který používá více než tři horolezce. Vybereme si toho horolezce H, který má nejlevnější výstroj. Potom dokážeme vytvořit levnější plán expedice takto: Horolezce H necháme doma a výstroj mu vůbec nekoupíme. Každý den, kdy měl horolezec H lézt, přiřadíme jinému horolezci (ne nutně všechny dny stejnému). Toto lze vždy provést, neboť:

1. Každý jiný horolezec má aspoň tak dobrou výstroj jako H, takže může lézt na vrchol, na který měl jít H.
2. Máme alespoň tři další horolezce a nejvýše dva z nich lezou v předchozím a následujícím dni. Zbývá tedy alespoň jeden horolezec, který si tento den může přidat mezi svoje.

Poměrně přímočaře tak dostáváme řešení s časovou složitostí $\mathcal{O}(n2^n)$: Předpokládáme, že máme přesně tři horolezce. Vyzkoušíme všechny možnosti, který den přiřadit kterému horolezci. Přitom pro každý den máme jen dvě možnosti, neboť ho nemůžeme přiřadit tomu horolezci, který měl předcházející den. Pro každou možnost spočítáme, kdo potřebuje jakou výstroj (přitom když některého horolezce nepoužijeme vůbec, zůstane mu cena výstroje nula).

Úloha má více různých řešení s polynomiální časovou složitostí, většinou založených na vhodném použití dynamického programování. Základní řešení je založeno na pozorování, že když se postupně rozhodujeme, kdo půjde na který vrchol, tak existuje jen $\mathcal{O}(n^4)$ zajímavých stavů: potřebujeme vědět, kolik vrcholů jsme už zpracovali ($n+1$ možností), pro každého horolezce jeho dosud nejvyšší přidělený vrchol (do n^3 možností) a číslo horolezce, který šel na předcházející vrchol (3 možnosti).

Ověření výstrojí

Zamyslíme se nad jednodušší úlohou: Někdo nám řekl, jaké výstroje naši tři horolezci mají. Naším úkolem už je pouze ověřit, zda takto vybaveni zvládnou celou expedici.

To umíme poměrně snadno ověřit v lineárním čase: postupně pro každé i od 1 do n a pro každého horolezce zjistíme, zda je možné naplánovat prvních i dní tak, aby všechno odpovídalo podmínkám a aby v i-tý den lezl právě on. Použijeme k tomu jednoduché pozorování: naplánovat prvních i dní a skončit horolezcem x dokážeme právě tehdy, když x může vylézt do výšky v_i a je možné naplánovat prvních $i-1$ dní tak, abychom skončili horolezcem jiným než x.

Kubické řešení

Jeden z horolezců zjevně musí mít výstroj na nejvyšší vrchol na vstupu. Vyzkoušíme všech $\mathcal{O}(n^2)$ možností pro zbývající dva horolezce: pro každého z nich je to buď výška některého jiného vrcholu ze vstupu, nebo nula, pokud ho vůbec na expedici nevezmeme. Každou z možností výše popsaným postupem v čase $\mathcal{O}(n)$ zkontrolujeme a vybereme nejlevnější z těch, které vyhovují.

Kvadratické řešení

Horolezce si označíme 1, 2 a 3. Horolezec 1 bude mít výstroj na nejvyšší vrchol na vstupu. Vyzkoušíme všech $\mathcal{O}(n)$ možností pro cenu výstroje horolezce 2. Když víme, jakou výstroj mají tito dva, můžeme použít podobný postup, jako výše popsané ověřování, jenom s jedním rozdílem: místo toho, abychom si jen pamatovali, které situace se dají dosáhnout a které ne, budeme si pamatovat, pro jakou nejmenší cenu výstroje horolezce 3 je která situace dosažitelná.

Toto řešení tedy vyzkouší $\mathcal{O}(n)$ možností, každou z nich v čase $\mathcal{O}(n)$, takže jeho časová složitost je kvadratická vzhledem k počtu dní expedice.

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

const int NEKONECNO = INT_MAX/3;
int N;
vector<int> V;

int dopocitej_tretiho(int prvni, int druhy) {
    vector< vector<int> > opt(N, vector<int>(3,NEKONECNO));
    // opt[i][j] == optimální cena výstroje třetího horolezce, při níž lze
    // naplánovat dny 0..i tak, aby poslední den lezl horolezec j+1.

    if (prvni >= V[0]) opt[0][0] = 0;
    if (druhy >= V[0]) opt[0][1] = 0;
    opt[0][2] = V[0];

    for (int n=1; n<N; ++n) {
        if (prvni >= V[n])  opt[n][0] = min( opt[n-1][1], opt[n-1][2] );
        if (druhy >= V[n]) opt[n][1] = min( opt[n-1][0], opt[n-1][2] );
        opt[n][2] = max( V[n], min( opt[n-1][0], opt[n-1][1] ) );
    }
    return min( opt[N-1][0], min( opt[N-1][1], opt[N-1][2] ) );
}

int main() {
    cin >> N;
    V.resize(N);
    for (int &v : V) cin >> v;
    int prvni = *max_element( V.begin(), V.end() );
    int nejlepsi = prvni + dopocitej_tretiho(prvni,0);
    for (int druhy : V)
        nejlepsi = min( nejlepsi, prvni + druhy + dopocitej_tretiho(prvni,druhy) );
    cout << nejlepsi << endl;
}
```

Zrychlujeme

Celé řešení můžeme ještě o kousek zrychlit. Podívejme se na to, jak se situace postupně mění, když cenu výstroje horolezce 2 budeme postupně snižovat. Na začátku budou mít horolezci 1 i 2 výstroj dostatečnou na všechny vrcholy a horolezec 3 si vystačí s nulovou cenou.

S každým snížením se stane, že se nějaký vrchol stane dostupným pouze pro horolezce 1. Můžeme si představit, že tyto nejvyšší vrcholy nám plán rozdělují na samostatné a od této chvíle již nezávislé úseky. Nutnou cenu vybavení pro horolezce 3 dostaneme jako maximum z jeho nutného vybavení ze všech těchto úseků. Každým snížením ceny výstroje horolezce 2 se tak jen nějaký úsek rozdělí na dva další.

Zbývá si rozmyslet, jak efektivně poznat úseky a jak v nich dopočítávat cenu výstroje horolezce 3. V rámci jednoho úseku ale platí, že kromě krajních vrcholů, které jsou pevně přiřazené horolezci 1 mají všechny ostatní nižší výšku a může je absolvovat horolezec 1 i 2. Úseky liché délky tak snadno porkyjeme posloupností typu $1212\ldots121$ a horolezce 3 nepotřebujeme. Pokud má úsek sudou délku, není možné jej obsadit střídavě horolezci 1 a 2 tak, aby se jedničkou začínalo i končilo. Horolezci 3 však můžeme přiřadit ten nejnižší vrchol z celého úseku a zbytek doplnit od krajů pomoci 1 a 2.

Hranice úseků i jejich minima můžeme nejsnáze spočítat tím, že je nebudeme postupně dělit, ale naopak je postupně pospojujeme. Tím dostáváme řešení, které vyzkouší $\mathcal{O}(n)$ možností a na každou spotřebuje $\mathcal{O}(1)$ času. Protože ale vrcholy potřebujeme procházet v pořadí podle velikosti (nejprve od nejmenšího po největší pro předpočítání minim v úsecích a následně od největšího po nejmenší pro dopočítání ceny výstroje třetího horolezce), musíme je na začátku seřadit v čase $\mathcal{O}(n \log n)$.

P-III-3 Exaktní exponenciální algoritmy

Obě soutěžní úlohy jsou vlastně úlohami o barvení grafů. Graf, který obarvujeme, má ve vrcholech jednotlivá zvířata a hrany představují konflikty mezi nimi. Různé barvy představují různé výběhy. Vrcholům grafu chceme přiřadit barvy tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu.

V podúloze A se ptáme, zda lze zadaný graf obarvit třemi barvami (které budeme pro jednoduchost nazývat červená, zelená a modrá).

V podúloze B se ptáme, jaký nejmenší počet barev nám postačí na obarvení zadaného grafu.

(Pro zajímavost uvedeme, že pro tyto úlohy nejen že neznáme žádný efektivní algoritmus, ale máme i dobrý důvod domnívat se, že takové algoritmy pro ně neexistují – jedná se o tzv. NP-těžké problémy. Efektivní exponenciální algoritmy jsou tedy nejlepším exaktním řešením těchto problémů, jaké známe.)

Podúloha A, pomalejší řešení

Všech možností, jak lze zvířata rozmístit do výběhů, je 3^n. Chceme-li tedy dosáhnout časové složitosti $\hat{O}(2^n)$, musíme dělat něco šikovněji. Ukážeme si dva možné přístupy.

První je založen jen na šikovnějším zkoušení možností. Začneme tím, že si graf rozdělíme na komponenty souvislosti. Zjevně stačí zkontrolovat každou z nich zvlášť. Když nyní máme souvislý graf, všechny možnosti jeho obarvení budeme zkoušet backtrackingem. Počáteční vrchol můžeme bez újmy na obecnosti obarvit červenou barvou. Každý další vrchol určený na obarvení zvolíme tak, aby sousedil s alespoň jedním vrcholem, který je už obarven. Tak zajistíme, že pro každý vrchol budeme mít na výběr nejvýše dvě možné barvy (neboť mu nesmíme dát stejnou barvu, jakou už má jeho soused). Celkově tedy v nejhorším případě vyzkoušíme nejvýše 2^{n-1} různých obarvení.

Druhý způsob je založen na pozorovaní, že tutéž otázku pro dvě barvy namísto tří už umíme zodpovědět efektivně (tj. v polynomiálním čase). Jedná se vlastně o otázku, zda je graf bipartitní. S efektivním řešením této úlohy jsme se setkali v krajském kole. Úlohu pro tři barvy nyní můžeme vyřešit tak, že vyzkoušíme všech 2^n možností, které vrcholy jsou červené. Pro každou z nich v polynomiálním čase ověříme, zda lze zbývající vrcholy obarvit zelenou a modrou barvou.

Podúloha A, vzorové řešení

Vzorové řešení získáme tak, že vylepšíme druhé z výše uvedených pomalejších řešení. Budeme při tom využívat ještě jeden pojem ze studijního textu: nezávislé

množiny. Když se podíváme na korektně obarvený graf, je zjevné, že každá barva určuje jednu nezávislou množinu jeho vrcholů. Výše uvedené řešení bychom tedy mohli vylepšit tak, že namísto všech 2^n podmnožin vrcholů budeme zkoušet jen ty podmnožiny, které jsou nezávislé. Těch ale stále může být řádově 2^n, takže samo o sobě toto pozorování nestačí. Jsme ale na dobré cestě.

Nezávislou množinu vrcholů X nazveme *nezvětšitelnou,* jestliže do ní nemůžeme přidat žádný další vrchol grafu tak, aby zůstala nezávislou.

Tvrzení 1: Pokud je možné zadaný graf korektně obarvit třemi barvami, potom ho lze obarvit takovým způsobem, aby množina červených vrcholů byla nezvětšitelná.

Důkaz: Uvažujme libovolné platné obarvení. Postupně se podíváme na každý vrchol, který není červený. Jestliže ani žádný jeho soused není červený, přebarvíme ho na červeno. Když tento proces skončí, máme opět platné obarvení a zároveň je zjevné, že nová množina červených vrcholů je nezvětšitelná.

Tvrzení 2: Všech možných nezvětšitelných nezávislých množin je nejvýše $3^{n/3}$ a všechny je můžeme vygenerovat v čase $\hat{O}(3^{n/3})$, neboli $\hat{O}(1.4423^n)$.

Důkaz: „Lepší algoritmus 2“ ze studijního textu ve skutečnosti řeší přesně tuto úlohu.

Získali jsme řešení podúlohy A v čase $\hat{O}(1.4423^n)$. Ještě jednou si ho stručně shrneme: použijeme Lepší algoritmus 2 ze studijního textu na vygenerování všech možností, které vrcholy budou červené. Pro každou z nich v polynomiálním čase ověříme, zda je možné zbytek grafu obarvit dvěma barvami.

Podúloha B, exponenciální řešení

V nejhorším případě budeme potřebovat n barev – pokud se žádná dvě zvířata nesnesou, potřebuje mít každé vlastní výběh. Přímočaré zkoušení všech možností backtrackingem bude proto mít časovou složitost až řádově n^n, tedy ještě horší než exponenciální.

Abychom dosáhli exponenciální časové složitosti, nebudeme vrcholy barvit jeden po druhém, ale budeme jednu po druhé přidávat celé barvy.

Náš algoritmus si můžeme představit jako rekurzivní funkci, která na vstupu dostane množinu ještě neobarvených vrcholů a na výstupu vrátí nejmenší počet dalších barev potřebný na jejich korektní obarvení. Tělo této funkce bude vypadat následovně: Je-li množina prázdná (tzn. už jsme obarvili všechno), vrátíme nulu. Jinak vezmeme následující barvu a vyzkoušíme všechny možnosti, které neobarvené vrcholy ji dostanou a které ne. Pro každou z těchto možností rekurzivním voláním naší funkce zjistíme, kolik dalších barev je ještě zapotřebí k dokončení barvení, a vybereme nejlepší z těchto možností.

Nyní stačí uvědomit si, že můžeme použít memoizaci – pro každou z 2^n podmnožin vrcholů jednou spočítáme její optimální řešení a zapamatujeme si ho.

Graf, který obarvujeme, má 2^n podmnožin vrcholů, potřebujeme tedy zodpovědět 2^n otázek. U každé z nich vyzkoušíme nejvýše 2^n možností, které vrcholy dostanou následující barvu. Celkově proto můžeme časovou složitost shora odhadnout $\hat{O}(4^n)$ – úspěšně jsme tak sestrojili řešení s exponenciální časovou složitostí.

Totéž řešení můžeme zformulovat také jako iterativní dynamické programování: postupně pro každou podmnožinu vrcholů zadaného grafu, od nejmenších po největší, zjistíme optimální potřebný počet barev vyzkoušením výše popsaných možností. V praxi by se takové řešení dobře implementovalo pomocí bitových masek.

Podúloha B, vzorové řešení

Výše uvedený odhad časové složitosti není těsný a šikovnější úvahou ho dokážeme vylepšit. Stačí si všimnout, že vždy, když zkoušíme nějakou konkrétní možnost, jak přidat jednu novou barvu, je každý vrchol v jednom ze tří stavů: už obarvený, právě dostává barvu, nebo zůstává neobarvený. Celkem při zodpovídání všech otázek tedy vyzkoušíme jen 3^n možností, takže při šikovné implementaci bude mít výše popsané řešení časovou složitost dokonce $\hat{O}(3^n)$.

Popsaný algoritmus dokážeme ještě vylepšit podobným způsobem, jako jsme to udělali v podúloze A. Stačí si uvědomit, že i zde platí analogie Tvrzení 1: pokaždé, když zkoušíme všechny možnosti pro novou barvu, stačí zkoušet ty, které jsou nezvětšitelné. Již přímočarý odhad ukazuje, že takto pro každou z 2^n otázek vyzkoušíme nejvýše 1.4423^n možností, jak přidat novou barvu, takže budeme mít časovou složitost $\hat{O}(2.8846^n)$.

Přesnější analýzou tohoto algoritmu je možné odvodit ještě těsnější horní odhad jeho časové složitosti: $\hat{O}(2.4423^n)$.