A-III-4 Bitový AND

Zadanie si môžeme sformulovať nasledovne: Na vstupe sme dostali sadu prirodzených čísel. Našou úlohou je vybrať spomedzi nich niektoré tak, aby v dvojkovej sústave bitový AND vybratých čísel končil jednotkou nasledovanou čo najviac nulami.
Čiastočné body sa dali získať dvomi spôsobmi. Pre malé n funguje hrubá sila – prezrieme všetky podmnožiny a vyberieme najlepšiu. Ak sú malé všetky a_i, môžeme použiť dynamické programovanie: postupne pre každé k zostrojíme množinu všetkých bitových ANDov, ktoré sa dajú dosiahnuť nejakou kombináciou prvých k čísel zo vstupu. (Časová zložitosť takéhoto riešenia je teda $O(n2^b)$, kde b je počet bitov v spracúvaných číslach.)
K riešeniu za plný počet bodov nás privedú tri pozorovania.

Riešenie vždy existuje. Ak vyberieme len jedno číslo, bitový AND vybratých čísel je toto číslo, čiže máme nejaké platné riešenie.

Je len 30 možností pre to, koľkými nulami bude končiť výsledný bitový AND. A tesne pred týmito nulami musíme mať jednotku.

Pridávať čísla medzi vybraté (skoro) nikdy nič nepokazí. Predstavme si, že máme vybratú nejakú množinu čísel, ktorých bitový AND končí $100\ldots0$, pričom núl je presne z. Inými slovami, najpravejšia 1 v našom bitovom ANDe je na z-tej pozícii. Čo sa teraz stane, ak do našej množiny pridáme niektoré ďalšie číslo, ktoré má tiež jednotku na z-tej pozícii? Nič podstatné. Presnejšie, opäť dostaneme platné riešenie a toto riešenie bude rovnako dobré ako to doterajšie – na z-tej pozícii ostane 1, a na pozíciach napravo od nej ostanú 0, lebo AND s novým číslom nemá ako zmeniť 0 na 1.

Čo z toho všetkého vieme teda odvodiť?
Ak pre konkrétnu hodnotu z (t.j. konkrétny počet núl na konci bitového ANDu našich čísel) existujú *nejaké* platné riešenia, tak jedným z nich je určite riešenie pozostávajúce z *úplne všetkých* čísel, ktoré majú na z-tej pozícii jednotku.
A teda vieme našu úlohu riešiť nasledovne: Postupne pre každé z od 29 po 0 vypočítame bitový AND všetkých čísel, ktoré majú na z-tej pozícii jednotku. Zakaždým potom overíme, či má výsledný bitový AND na všetkých menej významných pozíciách nuly. Ak nie, pre toto z riešenie neexistuje. Ak áno, práve sme našli najlepšie možné riešenie (lebo ideme od najväčšieho z dodola), stačí ho vypísať a skončiť.
Vyššie popísané riešenie má časovú zložitosť $O(nb)$. V našej úlohe je $b = 30$ (všetky čísla na vstupe sú nanajvýš 30-bitové). Toto je malá konštanta, preto sa smelo môžeme tváriť, že naše riešenie je v princípe lineárne od počtu spracúvaných čísel.

Listing programu (Python)

```
N = int( input() )
A = [ int(_) for _ in input().split() ]
for b in range(31,-1,-1):
    chcem = 2 ** b
    B = [ x for x in A if x & chcem != 0 ]
    if B == []: continue
    Band = B[0]
    for b in B: Band = Band & b
    if Band & (chcem-1) == 0:
        print(len(B))
        print(' '.join(str(_) for _ in B))
        import sys
        sys.exit()
```

A-III-5 Samko stále stávkuje

Táto úloha má veľa rôznych riešení bežiacich v polynomiálnom čase.
Informácie, kto porazil koho, si môžeme reprezentovať ako orientovaný graf. Najjednoduchší možný spôsob je použiť maticu susednosti. Pre každý závod si do nej zaznačíme všetkých $\binom{n}{2}$ dvojíc (i, j) takých, že i porazil j. Na záver pustíme Floydov-Warshallov algoritmus, ktorý zistí, odkiaľ kam sa dá po hranách dostať. (Odborne tomu hovoríme, že vypočítame tranzitívny uzáver.) Takéto riešenie má časovú zložitosť $O(zn^2 + n^3)$.

Ako takéto riešenie zlepšiť? V prvom rade vieme ušetriť prácu pri zápise informácií. Ak bežec X dobehol prvý, Y druhý a Z tretí, stačí si zaznačiť, že X porazil Y a že Y porazil Z. Z toho si už vieme odvodiť, že aj X porazil Z. Pre každý závod teda stačí spraviť len $n-1$ orientovaných hrán. Nová časová zložitosť: $O(zn+n^3)$.
Druhým zlepšením je uvedomiť si, že náš nový graf môže byť omnoho redší ako pôvodný: máme n vrcholov a nanajvýš rádovo nz hrán. Ak bude $z < n$, oplatí sa nám viac z každého vrcholu zvlášť spustiť prehľadávanie (napr. do šírky), ktoré zistí, kam všade sa z neho vieme dostať po hranách. Časová zložitosť tohto riešenia je $O(zn^2)$.

Povieme, že dvaja bežci A a B sú *ekvivalentní*, ak si A trúfa na B a aj B si trúfa na A. Ekvivalentní bežci si zjavne trúfajú na tú istú množinu bežcov: ak si A trúfa na C, tak aj B si zjavne trúfa na C a naopak.
V grafovej terminológii: dvaja bežci sú ekvivalentní, ak ležia v tom istom silno súvislom komponente. Zdalo by sa teda, že nám pomôže, keď nájdeme silno súvislé komponenty v našom grafe. Z každého z nich by potom stačilo spustiť prehľadávanie len raz.
Samo o sebe však toto pozorovanie žiadne zlepšenie nepridá: ak napríklad všetky závody dopadli rovnako, budeme mať n jednoprvkových silno súvislých komponentov.

Na lepšie riešenie potrebujeme ešte jedno pozorovanie. Vo všeobecnosti platí, že keď zoberieme orientovaný graf a na každý jeho silno súvislý komponent sa budeme dívať ako na jeden veľký vrchol, môže nám vzniknúť úplne hocijaký acyklický graf. V našom prípade však vždy vznikne len jedna jediná cesta: prvá skupina ekvivalentných bežcov si bude trúfať na každého, druhá skupina si bude trúfať na každého okrem prvej skupiny, tretia na každého okrem prvých dvoch skupín, a tak ďalej.
V nasledujúcom texte si vysvetlíme, prečo je tomu tak, a navyše si aj povieme jednoduchý spôsob ako príslušné silno súvislé komponenty zostrojiť bez potreby použiť komplikovaný všeobecný algoritmus.

Vzorové riešenie

Pre jednoduchosť si predstavme, že na začiatku bežci ešte nie sú očíslovaní. Očíslujeme ich až v cieli prvého závodu v poradí, v akom dobehli. Už po prvom závode teda platí, že pre ľubovoľné $i < j$ si bežec i trúfa na bežca j.
Každý ďalší závod nám vyrobí $n-1$ nových orientovaných hrán. A čo spôsobia tie? Hrana idúca z menšieho čísla na väčšie nespôsobí vôbec nič, toto sme vedeli už po prvom závode. Hrana idúca z väčšieho čísla na menšie nám spôsobí, že niektoré dvojice bežcov začnú byť ekvivalentné.
Povedzme napríklad, že v druhom závode bežec 11 zdolal bežca 6. Toto znamená, že odteraz už navždy budú bežci s číslami od 6 po 11 ekvivalentní: keďže máme cyklus $6 \to 7 \to 8 \to 9 \to 10 \to 11 \to 6$, každý z nich si trúfa na každého iného z nich.
Samozrejme, ekvivalencia je tranzitívna. Ak nám jedna hrana spôsobí, že sú ekvivalentní bežci od 6 po 11 a iná hrana spôsobí, že sú ekvivalentní bežci od 3 po 7, tak nutne musia byť ekvivalentní úplne všetci bežci od 3 až po 11.
Pôvodne jednoznačné poradie od 1 po n sa nám teda pridaním všetkých nových hrán zmení na niekoľko po sebe idúcich intervalov takých, že vždy v rámci intervalu sú všetci bežci ekvivalentní.
Ako tieto intervaly zostrojiť? Najjednoduchšie je predstaviť si každú spätnú hranu (t.j. hranu zodpovedajúcu tomu že bežec s väčším číslom dobehol tesne pred bežcom s menším číslom) ako úsečku na reálnej osi. Zjednotením všetkých týchto úsečiek dostaneme hľadané disjunktné intervaly. No a zjednotenie úsečiek ľahko nájdeme pomocou „zametania“: stačí si usporiadať všetky začiatky a konce a postupne zľava doprava ich spracovať.
Takéto riešenie má časovú zložitosť $O(nz \log(nz))$, ak pri zametaní použijeme univerzálne knižničné triedenie, resp. dokonca $O(nz)$ ak použijeme triedenie v lineárnom čase (napr. CountSort).

Ešte jednoduchšie riešenie

Ak sa ešte trochu zamyslíme, budeme vedieť vzorové riešenie implementovať ešte jednoduchším spôsobom.
Naďalej si pre jednoduchosť predstavujme, že sú bežci očíslovaní podľa toho, ako dobehli do cieľa prvého závodu.
Už vieme, že bežci sa nám rozpadnú na niekoľko *tried ekvivalencie* – skupín, v ktorej sú každí dvaja bežci ekvivalentní. A tiež vieme aj to, že triedu tvorí vždy niekoľko po sebe očíslovaných bežcov. Ostáva teda určiť jediné: pre ktoré k ležia bežci k a $k+1$ v tej istej skupine?

To je ale tiež ľahké: je to práve vtedy, ak aspoň raz niektorý bežec s číslom väčším ako k porazil niektorého z bežcov s číslom menším alebo rovným k.
Inými slovami, medzi bežcami k a $k+1$ je hranica skupín práve vtedy, ak *v úplne každom závode* dobehli na prvých k miestach bežci s číslami 1 až k. No a takto sformulovanú podmienku už ľahko overíme v čase $O(nz)$. Dokonca to vieme spraviť aj bez prečíslovania bežcov. Stačí postupne čítať výsledky závodov. Prečítame všetky prvé miesta a zaznačíme si bežcov, ktorí sa na nich objavili. Potom spravíme to isté s druhými miestami, tretími miestami, a tak ďalej. No a vždy, keď sme práve spracovali všetky k-te miesta a platí, že doteraz sme videli len k rôznych bežcov, je za k-tym miestom koniec skupiny.

Listing programu (C++)

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

int main() {
    // načítame vstup
    int N, Z;
    cin >> N >> Z;
    vector< vector<int> > vysledky(Z, vector<int>(N));
    for (int z=0; z<Z; ++z) for (int n=0; n<N; ++n) cin >> vysledky[z][n];

    int videnych = 0, spracovanych = 0; // koľko bežcov sme už videli aspoň raz a koľko Z-krát?
    vector<bool> videl(N+1,false);      // ktorých bežcov sme už videli?
    vector<int> odpovede(N+1);          // na koľkých bežcov si trúfa bežec n?
    vector<int> aktualna_skupina;       // ktorí bežci patria do práve spracúvanej skupiny?

    for (int n=0; n<N; ++n) {
        // zaznačíme si všetkých bežcov ktorí niekedy dobehli na pozícii n+1
        for (int z=0; z<Z; ++z) if (!videl[vysledky[z][n]]) {
            videl[vysledky[z][n]] = true;
            ++videnych;
            aktualna_skupina.push_back( vysledky[z][n] );
        }
        // skontrolujeme, či práve neskončila skupina
        if (videnych == n+1) {
            for (int kto : aktualna_skupina) odpovede[kto] = N - 1 - spracovanych;
            spracovanych += aktualna_skupina.size();
            aktualna_skupina.clear();
        }
    }
    // vypíšeme odpovede
    for (int n=1; n<=N; ++n) cout << odpovede[n] << (n==N ? "\n" : " ");
}
```

A-III-6 Vykopávky

Riešenie hrubou silou má časovú zložitosť $O(p_r p_s d_r d_s)$: najprv obyčajným prehľadávaním „ofarbíme“ jednotlivé miestnosti, potom pre každú možnú polohu diery prejdeme všetky z nej dosiahnuteľné štvorce a zistíme, ktoré miestnosti sa medzi nimi aspoň raz vyskytli.
Lepšie riešenie môžeme založiť na nasledovnom pozorovaní: Predstavme si, že už sme spracovali nejakú polohu diery, a teda presne vieme, koľko štvorcov ktorej miestnosti je z nej priamo dosiahnuteľných. Čo sa teraz zmení, ak dieru posunieme o stĺpec doprava? Nejakých $d_r + 2$ štvorcov prestane byť dosiahnuteľných a nejakých iných $d_r + 2$ ich začne byť dosiahnuteľných. Stačí teda prejsť týchto $O(d_r)$ štvorcov a upraviť počty pre zodpovedajúce miestnosti a budeme mať spracovanú novú polohu diery.
Samotnú odpoveď, teda počet rôznych dosiahnuteľných miestností, si vieme tiež udržiavať počas toho, ako robíme vyššie popísané zmeny. Vždy, keď do oblasti diery pridáme štvorec patriaci novej miestnosti, zväčšíme si počítadlo rôznych miestností. A vždy, keď z oblasti diery odstránime štvorec a tým spôsobíme, že nejakej miestnosti klesne počet dosiahnuteľných štvorcov na nula, počítadlo rôznych možností zase znížime.
Ľahko nahliadneme, že toto riešenie má časovú zložitosť $O(p_r p_s d_r)$: ušetrili sme pri kontrole dier jeden ich rozmer. (Dieru pokrývajúcu najľavejších d_s stĺpcov síce napr. vždy prezrieme celú, ale to nám zložitosť nepokazí.)
Existujú aj asymptoticky efektívnejšie riešenia, sú však zbytočne komplikované na implementáciu. Na získanie plného počtu bodov nebolo lepšie riešenie potrebné.

Listing programu (C++)

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

const int stepR[] = {-1,1,0,0}, stepS[] = {0,0,-1,1};

int PR, PS, DR, DS;
vector<string> plan;
int PM;                           // počet miestností
vector< vector<int> > miestnost; // pre každé políčko číslo miestnosti kam patrí
int odpoved;
int dobre_miestnosti;             // koľko aktuálne máme dobrých miestností
vector<int> pocet_vnutri;         // pre každú miestnosť, koľko jej políčok vidíme

void uprav(int r, int s, int zmena) {
    // pridáme políčko (r,s) do aktuálnej diery alebo ho z nej odstránime
    if (r < 0 || r >= PR || s < 0 || s >= PS) return;
    if (plan[r][s] == '#') return;
    int m = miestnost[r][s];
    pocet_vnutri[m] += zmena;
    if (zmena == -1 && pocet_vnutri[m] == 0) --dobre_miestnosti;
    if (zmena == +1 && pocet_vnutri[m] == 1) ++dobre_miestnosti;
}

int main() {
    // načítame vstup
    cin >> PR >> PS >> DR >> DS;
    plan.resize(PR);
    for (int r=0; r<PR; ++r) cin >> plan[r];

    // ofarbíme si miestnosti prehľadávaním do šírky
    PM = 0;
    miestnost.resize( PR, vector<int>(PS,-1) );
    for (int r=0; r<PR; ++r) for (int s=0; s<PS; ++s) if (plan[r][s]=='.' && miestnost[r][s]==-1) {
        miestnost[r][s] = PM;
        queue<int> Q;
        Q.push(r); Q.push(s);
        while (!Q.empty()) {
            int cr = Q.front(); Q.pop();
            int cs = Q.front(); Q.pop();
            for (int d=0; d<4; ++d) {
                int nr = cr + stepR[d], ns = cs + stepS[d];
                if (nr < 0 || nr >= PR || ns < 0 || ns >= PS) continue;
                if (plan[nr][ns] == '#' || miestnost[nr][ns] != -1) continue;
                miestnost[nr][ns] = PM;
                Q.push(nr); Q.push(ns);
            }
        }
        ++PM;
    }

    // postupne skúšame všetky možné polohy diery
    odpoved = 0;
    dobre_miestnosti = 0;
    pocet_vnutri.resize(PM,0);
    for (int r0=0; r0+DR <= PR; ++r0) {
        // začneme s dierou úplne naľavo od plánu, skončíme úplne napravo
        for (int s0=-DS; s0 <= PS+2; ++s0) {
            // ideme posunúť ľavý horný roh diery o políčko doprava, na (r0,s0)

            // pridáme miestnosti, ktoré sme odkryli
            uprav( r0-1, s0+DS-1, +1 );
            for (int r=r0; r<r0+DR; ++r) uprav( r, s0+DS, +1 );
            uprav( r0+DR, s0+DS-1, +1 );

            // odstránime miestnosti, ktoré sme zakryli
            uprav( r0-1, s0-1, -1 );
            for (int r=r0; r<r0+DR; ++r) uprav( r, s0-2, -1 );
            uprav( r0+DR, s0-1, -1 );

            // pozrieme sa, či sme nezlepšili riešenie
            odpoved = max( odpoved, dobre_miestnosti );
        }
    }
    cout << odpoved << endl;
}
```

TRIDSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Eduard Batmendijn, Michal Forišek, Jaroslav Petrucha
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2017