Друштво математичара Србије
Адреса: Кнез Михаилова 35/4,
11000 Београд
сајт: www.dms.rs
е-маил: pom@dms.rs

Решење проблема Б за Април 2012
Подматрица

Проблем Б месеца априла припада класи комбинаторних и *backtrack* проблема. Сва пристигла решења су базирана на сличној идеји - идеји која решава и проблем са већим ограничењима матрице. Овде излажемо алгоритам и имплементацију овог приступа. Поред тога изнећемо још једну имплементацију ове идеје, са мало бољом сложеношћу. Напоменимо и то да је већина ученика направила превид код специјалног случаја: када су сви елементи полазне матрице негативни.

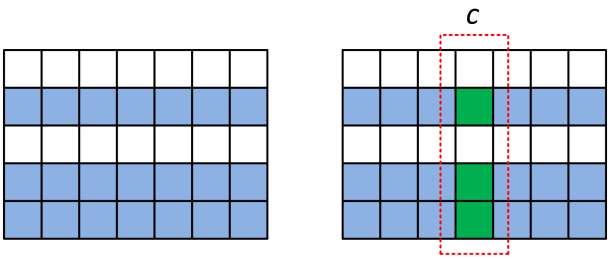
Алгоритам: На самом почетку, приметимо да су ограничења за димензију матрице јако мала: $n, m \leq 16$. Ово нас у старту наводи на то да не можемо "брзо" наћи тражену подматрицу (односно њену суму). Означимо са Δ скуп свих подматрица. Потребно је "паметно" претражити овај скуп и издвојити ону подматрицу са највећом сумом. Под "паметном" претрагом подразумевамо ону којој нећемо морати да тестирамо све подматрице, веће да што већи број искључимо без самог сумирања. Овако теоријски речено, можда, звучи чудно али ћемо ускоро видети, на овом конректном примеру, да идеја није тешка.

При решавању оваквих проблема, неопходно је проценити величину скупа претраге, како би знали потребну брзину претраге. Која је кардиналност нашег скупа Δ? Из дефиниције видимо да је подматрица одређена селектовањем неких врста и колона. Број врста је једнак n, тако да постоји 2^n различитих начина за селектовање врсти (Зашто?). Аналогно, постоји 2^m начина за селектовање колона. Ово нас доводи до чињенице да је $|\Delta| = 2^n \cdot 2^m = 2^{n+m}$. Сада видимо да наивни приступ, који тестира све подматрице из скупа Δ, не задовољава временска ограничења. Из овог разлога смо тражену, горе поменуту, претрагу назвали "паметном".

Посматрајмо скуп матрица које се добијају фиксирањем врста $v = \{v_1, v_2, .., .v_s\}$, $s \in [1, n]$. Овај скуп има 2^m подматрица, јер избор колона не зависи од избора врсти. Да ли за овако фиксиран скуп врсти морамо испитати свих 2^m избора колона? Одговор је негативан. Наиме, од нас се тражи подматрица са највећим збиром, а фиксирањем врсти једнозначно можемо одредити подматрицу из скупа Δ_v са максималним збиром.

Који услов треба да задовољава колона c да би ушла у тражену подматрицу из скупа Δ_v? Како је нама потребна подматрица са највећим збиром, колону c ћемо селектовати једино

у случају да је сума њених елемената, добијених у пресеку са врстама v, позитивна. Заиста, уколико је ова сума негативна, она ће само смањити крању суму подматрице. Специјалан случај је када су суме свих колона негативне (добијамо "празну" подматрицу) - тада селектујемо само колону са највећим збиром (чиме добијемо негативну суму, али свакако максималну).



Слика 1. Пример селектованих врста (плава боја). Означена колона c се селектује уколико је сума елемената добијених у пресеку са означеним врстама (зелена боја) позитивна.

Овим смо у потпуности описали тражени алгоритам: за сваки избор врсти v, одређујемо подматрицу са највећим збиром (једнозначим избором колона), а као резултат враћамо максималну добијену суму над свим могућим изборима врсти. Сложеност овог алгоритма, у најбољом импелемнтацији коју ћемо изнети у даљем тексту, је $O(2^n \cdot n \cdot m)$.

Имплементација 01: Као што смо видели, алгоритам има два, главна, независна корака: избор врсти и селектовање колона за дати избор врсти. За први корака, потребно је наћи све могуће начине за селектовање врсти. Ово је еквивалентно претрази свих подскупова скупа са n елемената. Постоје два могућа приступа: а) рекурзивни којим иницијализујемо одређени бинарни низ (чији елементи указују на то да ли је колона селектована или не); б) помоћу битмаски. Како је ограничење за димензију матрице 16, једноставнији приступ, који ћемо овде изложити, је преко битмаски.

Наиме, посматрајмо природни број k из опсега $[1, 2^n - 1]$. Овај број можемо посматрати као n-то битни број (додавањем водећих нула). Са друге стране имамо подскупове скупа са n елемената $V = \{1, 2, \ldots, n\}$. Између ова два скупа постоји бијекција. Заиста, броју k додељујемо подскуп K дефинисан као: врста i улази у подскуп акко је $(i-1)$-бит у броју k постављен на јединицу. Примера ради, за $n = 5$, број $k = 11$, који у бинаром запису има облик $(01011)_2$, одређује подскуп $\{1, 2, 4\}$.

У овом случају нема потребе за рекурзивном ицијализацијом низа који маркира одабране врсте. У ту сврху можемо користити обичан бројач k који ће узимати вредности од 1 до $2^n - 1$ (вредност 0 изостављамо јер у нашем случају празан скуп нема смисла). Ово је једноставно за имплементацију и доста брже од рекурзивног приступа (подсетимо се да су бинарне операције брже од оних у декадном систему). Такође, нема додатног губљења времена и меморије потребне за рекурзивне методе.

Други део алгортима је обичан грамзиви метод, који само сумира елементе колона за тренутно селектоване врсте. Овај део не захтева додатна објашњења везана за имплементацију.

Преостаје још да се размотри специјалан случај: када је за одређени избор врста, сума свих колона негативна. Тада добијамо да ни једна колона не улази у избор. У том случају је највећа сума подматрице из скупа Δ_v једнака 0 па она неће утицати на крајњи резултат. Међутим, шта се добија у случају када ово важи за сваки избор v? Тада елементи матрице

Псеудо код алгоритма проблема Подматрица

```
solution = 0;
for k ← 1 to 2^n − 1 do
    currentSum = 0;
    for j ← 1 to m do
        columnSum = 0;
        for i ← 1 to n do
            if (i − 1) цифра у бинаром запису број k је једнака јединици then
                columnSum = columnSum + a[i][j];
            end
        end
        if columnSum > 0 then
            currentSum = currentSum + columnSum;
        end
    end
    if currentSum > solution then
        solution = currentSum;
    end
end
if solution = 0 then
    solution = највећи елемент у матрици;
end
return solution;
```

a морају бити негативни или једнаки нула. Уколико са $solution$ означимо крајње решење, оно ће бити позивитно ако и само ако матрица a има барем један позитиван елемент. Дакле, уколико је $solution$ једнако нули на крају претраге, као резултат враћамо највећи елемент матрице a (који је мањи или једнак нули).

Напомена: "Показали" смо да је сложеност алгоритма једнака $O(2^n \cdot n \cdot m)$, при чему смо користили апроксимацију да у при сваком селектовању врста, у просеку, обилазимо $O(n)$ врста. Иако ово није интуитивно јасно, заинтересовани читалац може ово показати рачунајући суму $\frac{\sum_{k=0}^{n} k\binom{n}{k}}{2^n}$, где се као крајњи резултат добија вредност $\frac{n}{2}$.

Имплементација 02: Горе описана идеја се може имплементирати и у бољој сложености $O(2^n \cdot m)$, при чему ће меморијска сложеност бити $O(2^n)$. Наиме, идеја је да се памти додатна матрица sum димензије $2^n \times m$ дефинисана као:

$$sum[izborVrsta][kolona] = \text{ сума елемената из изабраних врста за дату колону}$$

Као што смо видели ова матрица може да се попуни у сложености $O(2^n \cdot n \cdot m)$. Проблем код прве имплементације је био тај што су се ове суме рачунале увек посебно, не ослањајући се на неке претходне резултате. Ово нас наводи на метод динамичког програмирања. Дати индекс $izborKolona$, као и у првој имплементацији, је природан број од 0 до $2^n - 1$ који посматрамо као маску која маркира изабране врсте.

Означимо са $f(x)$ број који се добија брисањем најмање цифре јединице у бинарном запису број x, а са $g(x)$ позицијију јединице најмање тежине. Примера ради, $f(6) = f((110)_2) =$

$(100)_2 = 4$, док је $g(6) = 2$. Сада можемо дефинисати следећу рекурентну везу

$$sum[izborVrsta][kolona] = sum[f(izborVrsta)][kolona] + a[g(izborVrsta)][kolona]$$

Заиста, $sum[f(izborVrsta)][kolona]$ представља суму свих елемената из дате колоне сем елемента из врсте означене последњом јединицом у маски $izborVrsta$. Једноставним додавањем овог елемента, $a[g(izborVrsta)][kolona]$, добија се тражена сума. Дакле, овим смо извршили иницијализацију матрице sum у сложености $O(2^n \cdot m)$. Уз помоћ ове матрице, алгоритам можемо једноставно имплементирати у датој сложености, јер сада за сваки избор врста, додајемо оне колоне чија је сума позитивна а њу већ имамо израчуату.

Међутим, на почетку смо напоменули да је меморијска сложеност овог алгоритма 2^n, а описана матрица sum је свакако већа. Можемо приметити да нама није потребна целокупна матрица sum. Уз помоћ додатног низа dp можемо рачунати матрицу sum само по колонама (дакле фиксирамо на почетку колону и за њу рачунамо дате суме), док у низу dp на позицији $izborVrsta$ додајемо суму текуће колоне ако је позитивна.

Имплементацију ове идеје (са избаченим специјални случајем) коју је послао ученик Борис Грубић, Гимн. "J.J. Змај" из Новог Сада, можете погледати овде.

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int MAX_N = 16;

int n, m;
int dp[1 << MAX_N], sum[1 << MAX_N], a[MAX_N][MAX_N], LOG[MAX_N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> a[i][j];
    memset(LOG, 0, sizeof(LOG));
    memset(dp, 0, sizeof(dp));
    for (int i = 0; 1 << i < MAX_N; ++i) LOG[1 << i] = i;
    for (int col = 0; col < m; ++col) {
        memset(sum, 0, sizeof(sum));
        for (int mask = 1; mask < 1 << n; ++mask) {
            int lastBit = mask & -mask;
            int row = LOG[lastBit];
            sum[mask] = sum[mask ^ lastBit] + a[row][col];
            dp[mask] += max(0, sum[mask]);
        }
    }
    int res = 0;
    for (int i = 1; i < 1 << n; ++i)
        res = max(res, dp[i]);
    cout << res << endl;
    return 0;
}
```

Напомена: Сложеност изложеног алгоритма није симетрична у односу на вредности n и m. Овај алгоритам решава проблем и када за димензију матрице a важе следећа ограничења: $n \leq 16$ и $m \leq 10^3$. Другим речима, потребно је да једна димензија матрице буде "мала" (уколико је број колона "мали" можемо увек посматрати транспоновану матрицу, јер је тражена вредност инваријантна у односу на ову операцију).

Решење задатка припремио:
Андреја Илић,
Природно математички факултет, Ниш