































#include<iostream>

using namespace std;

const int N = 1000;

int a[N]; // 存储集合元素

bool vis[N]; // 存储集合元素的状态

int subSum;  //子集当前和

int flag = 0;//标记是否找到过解

int n, c;

void print(int i) {//输出符合条件的子集的元素

for (int j = 0; j <= i; j++) {

if (vis[j] == true) {

if (j != i)printf("%d ", a[j]);

else printf("%d\n", a[j]);

}

}

}

void search(int i) {

//小于，就继续往后加数；大于，就回溯返回上一节点；等于，就输出，再回溯返回上一节点

if (i >= n) return; //超出范围

vis[i] = true;

subSum += a[i];

if (subSum == c) {//满足 输出

flag = 1;//表示我找到过符合条件的子集

print(i);

}

else if (subSum < c) {// 不足 继续取数

search(i + 1);

}

vis[i] = false;

subSum -= a[i];

search(i + 1);

return;

}

int main() {

cin >> n >> c;

for (int i = 0; i < n; i++) cin >> a[i];

search(0);

if (!flag) printf("No Solution!");

return 0;

}

#include<bits/stdc++.h>

using namespace std;

int n, m;

int bestx[10];

int B[10][10];

int x[10], low[10], high[10];

int bestd = 0;

int len(int ii){

for(int i = 1; i <= m; i ++ ){

high[i] = 0;

low[i] = n + 1;

}

for(int i = 1; i <= ii; i ++ )

for (int k = 1; k <= m; k ++ )

if(B[x[i]][k] > 0){

if(i < low[k])

low[k] = i;

if(i > high[k])

high[k] = i;

}

int tmp = 0;

for(int k = 1; k <= m; k ++ )

if(low[k] <= n && high[k] > 0 && tmp < high[k] - low[k])

tmp = high[k] - low[k];

return tmp;

}

void swap(int\* x, int i, int j){

int tmp;

tmp = x[i];

x[i] = x[j];

x[j] = tmp;

}

void backtrack(int i){

if(i == n){int tmp = len(i);

if(tmp < bestd){

bestd = tmp;

for(int j = 1; j <= n; j ++ )

bestx[j] = x[j];

}

}

else{// 若不是末尾；

for(int j = i; j <= n; j ++ ){

swap(x, i, j);

int ld = len(i);

if(ld < bestd)

backtrack(i + 1);

swap(x, i, j);

}

}

}

int arrangeBoards(){

bestd = n + 1;

for(int i = 1; i <= n; i ++ )

x[i] = i;

backtrack(1);

return bestd;

}

int main(){

cin >> n;

cin >> m;

vector<int> temp(m);

for(int i = 1; i <= n; i ++ ){

for (int j = 1; j <= m; j ++ ) {

cin >> B[i][j] ;

}

}

int minLen = arrangeBoards();

cout << minLen << endl;

for(int i = 1; i <= n; i ++ )

cout<<bestx[i]<<" ";

return 0;

}

#include <bits/stdc++.h>

using namespace std;

int n;

int boy[21][21], girl[21][21];

int Max = INT\_MIN; //MAX代表男女双方竞赛优势的总和的最大值 用来返回指定整数类型所能表示的最小值。

int sum = 0;

int res[21][21]; //data[i][j]用于存放男运动员i配对后的双方竞赛优势

int maxSum[21];   //保存每个男生匹配后可达到的最大双方竞赛优势

int book[21];    //标记女运动员是否已经匹配 0未匹配 1已匹配

//Max：40 -> 52

void dfs(int t){

    if(t>=n)  //t到达n后，代表全部标记访问了，得到了最大值

    {

        Max = max(Max, sum);

        return;

    }

    int cnt = 0;

    //求t及t之后男生匹配女生的最大值的和

    for (int i = t; i < n;i++){

        cnt += maxSum[i];

        //假设的贪心的让每个男运动员匹配最优的女运动员

    }

    //剪枝函数：之前t个已经匹配好的男女运动员的sum与

    //之后的t->n-1个男女匹配的最大值加起来得到的Max比较

    //若前者<=Max，剪枝

    if(sum+cnt<Max)

        return;

        //若cnt>=Max，要继续向下搜索

        //从第t个男生开始匹配，找未匹配的女生

    for (int i = 0; i < n;i++){

        if(!book[i]){//若第i个女生未匹配

            book[i] = 1;

            sum += res[t][i];

            dfs(t + 1);

            book[i] = 0; //若第t个男生匹配女生i得到的sum不大于Max，则回溯

            sum -= res[t][i];

        }

    }

}

int main(){

    cin >> n;

    for (int i = 0; i < n;i++){

        for (int j = 0; j < n;j++){

            cin >> boy[i][j];

        }

    }

    for (int i = 0; i < n;i++){

        for (int j = 0; j < n;j++){

            cin >> girl[i][j];

        }

    }

    for (int i = 0; i < n;i++){

        for (int j = 0; j < n;j++){

            //对每个男生都求男女双方竞赛优势，能得到i\*j种结果

            res[i][j] = boy[i][j] \* girl[j][i];

            //记录每个男生匹配后可达到的最大双方竞赛优势，用于后面的剪枝

            maxSum[i] = max(maxSum[i], res[i][j]);

        }

    }

    dfs(0);

    cout << Max << endl;

    return 0;

}

#include <bits/stdc++.h>

using namespace std;

int n,m;

int depth=1;   //子集树的深度。一层一层的找，最后若形成子集树，说明变换成功，此深度就是最少变换次数

string str="";   //回溯的记录结果

bool dfs(int n,int curDepth)

{

    if(curDepth>depth)   //当前搜索层超过子集树的深度，返回

        return false;

    int operation=n;   //进行两步操作：左子树：n\*3；右子树：n/2;

    for(int i=0;i<2;i++)

    {

        i==0?operation=3\*n:operation=n/2;   //如果i=0，进入左子树；否则，进入右子树

        if(operation==m||dfs(operation,curDepth+1))   //如果经过上述操作后，得到的数operation = m，或者经过下一次操作后，得到的数operation = m

        {

            i==0?str+="f":str+="g";   //左子树操作：加“f”；右子树操作：加“g”

            return true;   //满足条件就可以返回了

        }

    }

    return false;

}

int main()

{

    cin>>n>>m;

    while(!dfs(n,1))   //从第一层开始搜索。当dfs返回false,表示没有变换成功；否则，继续下一层循环，直到子集树建成

    {

        depth++;   //深度就是最小的变换次数

    }

    cout<<depth<<endl;

    cout<<str<<endl;

    return 0;

}

#include<iostream>

using namespace std;

#define MAX 50

int n, m;

int W[MAX] = { 0 };

int select[MAX] = { 0 };

int bestS[MAX] = {0}, minSum = 10000;

struct E {

int left;

int right;

} E[MAX] = {0};

bool isCover(int nowSelect[]) {

bool flag=true;

for (int i = 1; i <= m; i++) {

if (nowSelect[E[i].left] == 0 && nowSelect[E[i].right] == 0) {

flag = false;

break;

}

}

return flag;

}

void find\_min\_cover(int nowSelect[],int position,int sum) {

if (position == n + 1) {

if (isCover(nowSelect)) {

minSum = sum;

copy(nowSelect, nowSelect + n + 1, bestS);

}

}

else {

if (sum + W[position] < minSum) {//当前顶点选中

nowSelect[position] = 1;

find\_min\_cover(nowSelect, position + 1, sum + W[position]);

nowSelect[position] = 0;

}

//当前顶点未选中

find\_min\_cover(nowSelect, position + 1, sum);

}

}

int main() {

cin >> n >> m;

for (int i = 1; i <= n; i++) {

cin >> W[i];

}

int v1, v2;

for (int i = 1; i <= m; i++) {

cin >> v1 >> v2;

E[i].left = v1;

E[i].right = v2;

}

find\_min\_cover(select, 1, 0);

cout << minSum << endl;

for (int i = 1; i <= n; i++) {

cout << bestS[i] << " ";

}

return 0;

}

#include<iostream>

#include<cstring>

using namespace std;

#define MAX\_NODE 999

int max\_cut=0;

int n,m;

int cut\_pos[MAX\_NODE];

void cal\_max\_cut(int n,int m,bool is\_connected[][MAX\_NODE],int cut[],int i)

{

    if(i==n+1)

    {

        int count=0;

        for(int i=1;i<=n;i++)

        {

            for(int j=i+1;j<=n;j++)

            {

                if(is\_connected[i][j] && cut[i]!=cut[j])

                    count++;

            }

        }

        if(count>max\_cut)

        {

            max\_cut=count;

            for(int i=1;i<=n;i++)

            {

                if(cut[i] == 2)

                {

                    cut\_pos[i]=0;

                    continue;

                }

                cut\_pos[i]=cut[i];

            }

        }

        return;

    }

    cut[i]=1;

    cal\_max\_cut(n,m,is\_connected,cut,i+1);

    cut[i]=2;

    cal\_max\_cut(n,m,is\_connected,cut,i+1);

}

int main()

{

    cin>>n>>m;

    bool is\_connected[MAX\_NODE][MAX\_NODE];

    memset(is\_connected,false,sizeof(is\_connected));

    for(int i=0;i<m;i++)

    {

        int x,y;

        cin>>x>>y;

        is\_connected[x][y]=true;

        is\_connected[y][x]=true;

    }

    int cut[n+1];

    memset(cut,0,sizeof(cut));

    cal\_max\_cut(n,m,is\_connected,cut,1);

    cout<<max\_cut<<endl;

    for(int i=1;i<=n;i++)

        cout<<cut\_pos[i]<<" ";

    return 0;

}

#include <bits/stdc++.h>

using namespace std;

int n;    //部件数量

int m;    //供应商数量

int d;    //价格上限

int bestw;   //最小的重量

int\*\* c = NULL;    //二维数组，每个部件不同商家的价格

int\*\* w = NULL;     //二维数组，每个部件不同商家的重量

//每一个部件的信息

class Node {

public:

    int weight;        //当前已选机器的重量和

    int val;           //当前已选机器的价值和

    int source;        //哪个供货商

    int level;         //第几层,也代表了第几个部件

    int priority;      //优先级

    Node\* father;

};

Node\* leaf;//叶子结点

void Input() {

    scanf("%d %d %d",&n,&m,&d);

    w = (int\*\*)malloc(sizeof(int\*)\*(n + 1));

    c = (int\*\*)malloc(sizeof(int\*)\*(n + 1));

    for (int i = 1; i <= n; i++) {

        w[i] = (int\*)malloc(sizeof(int)\*(m + 1));

        c[i] = (int\*)malloc(sizeof(int)\*(m + 1));

    }

    leaf = NULL;

    for (int i = 1; i <= n; i++)

        for (int j = 1; j <= m; j++)

            scanf("%d",&c[i][j]);

    for (int i = 1; i <= n; i++)

        for (int j = 1; j <= m; j++)

            scanf("%d",&w[i][j]);

}

//优化的优先级设定

bool operator<(Node a, Node b)  //level按照减序

{

    if (a.priority == b.priority)return a.level < b.level;  //如果重量相同，选择level大的。

    return a.priority > b.priority;//否则，重量小的先出队

}

//计算当前节点的优先级

void QueuePriority(Node a) {

    int currentMinW;

    a.priority = a.val;

    //int temp\_min\_c = INT\_MAX;

    for (int i = a.level + 1; i <= n; i++)//选出剩余的部件在售货商中购买的最小质量,就是选择每一层最小的质量

    {

        currentMinW = 999999;

        for (int j = 1; j <= m; j++)  //每一层找最小的

        {

            currentMinW = currentMinW < w[i][j] ? currentMinW : w[i][j];//从m个商家中选择当层重量最小的

        }

        a.priority += currentMinW;

    }

}

//约束函数

bool constraint(Node\* pNode, int i) {

    return pNode->val + c[pNode->level + 1][i] <= d || pNode->weight + w[pNode->level + 1][i] <= bestw;

}

//创建节点

Node createNode(int level, Node\* father, int source, int val, int weight) {

    Node newNode{};

    newNode.level = level;//层次加1

    newNode.father = father;

    newNode.source = source;

    newNode.val = val;

    newNode.weight = weight;

    return newNode;

}

void MinWeightMachine() {

    int i, j;

    bestw = 999999;

    Node initial{};

    initial = createNode(0, NULL, 0, 0, 0);

    QueuePriority(initial);      //计算优先级

    priority\_queue<Node> heap;   //用优先队列，建立一个最小堆。加入进去就会自动排好序的。

    heap.push(initial);

    while (!heap.empty()) {

        Node\* pNode = new Node(heap.top());

        heap.pop();//队首元素作为父节点出队，即优先级值小的活结点先扩展

        if (pNode->level == n)//到达叶节点，不能扩展 ,得到一个解

        {

            if (pNode->weight < bestw)   //更新

            {

                bestw = pNode->weight;

                //MinValue  = pNode ->val;

                leaf = pNode;   //记录是最后是哪个结点数据,便于回溯找最优解

            }

        }

        else {

            for (i = 1; i <= m; i++)//扩展结点，依次选择每个售货商，每次都是m叉树

            {

                //可行性剪枝和限界剪枝

                if (constraint(pNode, i)) {

                    Node newNode{};//生儿子结点

                    newNode = createNode(pNode->level + 1, pNode, i, pNode->val + c[pNode->level + 1][i], pNode->weight + w[pNode->level + 1][i]);

                    QueuePriority(newNode);     //计算优先值

                    heap.push(newNode);//儿子入队

                }

            }

        }

    }

}

void Output() {

    printf("%d\n",bestw);

    int\* result = (int\*)malloc(sizeof(int)\*(n + 1));

    for (int i = n; i >= 1; i--) {

        result[i] = leaf->source;//从最后叶子结点回溯到根节点

        leaf = leaf->father;

    }

    for (int i = 1; i <= n; i++)

        if(i==1) printf("%d",result[i]);

        else printf(" %d",result[i]);

    putchar('\n');

}

int main() {

    Input();

    MinWeightMachine();

    Output();

    return 0;

}

#include <cstring>

#include <cstdio>

#define maxn 20

#define lowbit(x) (x & (-x))

#define maxs (1 << maxn)

#define INF 0x7f

int line[maxn][maxn], ans[maxn], anscnt = 0, dp[maxs], cnt[maxs], n, rs, pre[maxs], qpow[maxn];

int main(void)

{

scanf("%d", &n);

for (int i = 0; i < n; ++i)

for (int j = i + 1; j < n; ++j)

scanf("%d", &line[i][j]), line[j][i] = line[i][j];

memset(dp, INF, sizeof(dp));

rs = 1 << n;

qpow[0] = 1;

for (int i = 1; i < n; ++i)

qpow[i] = qpow[i - 1] << 1;

for (int i = 1, temp = 0, old = 0; i < rs; ++i, temp = lowbit(i), old = i - temp)

{

int pos = 0;

cnt[i] = cnt[old];

while (temp >>= 1)

++pos;

for (int j = 0; j < n; ++j, old >>= 1)

if (old & 1)

cnt[i] -= line[pos][j];

else cnt[i] += line[pos][j];

}

dp[0] = 0;

for (int i = 1, temp = i; i < rs; ++i, temp = i)

for (int j = 0; temp; ++j, temp >>= 1)

if ((temp & 1))

if (dp[i] > dp[i - qpow[j]] + cnt[i - qpow[j]])

{

dp[i] = dp[i - qpow[j]] + cnt[i - qpow[j]];

pre[i] = j;

}

printf("%d\n", dp[--rs]);

while (rs)

{

ans[anscnt++] = pre[rs];

rs -= qpow[pre[rs]];

}

for (int i = 0; i < anscnt; ++i)

printf("%d ", ans[i] + 1);

return 0;

}

#include <iostream>

#include <fstream>

#include <algorithm>

#include <functional>

#include <queue>

using namespace std;

class Node{

public:

static int k;

static int n;

Node();

Node(const Node &other);

void compute(int i);//当前节点的所需time

int time;

int t;

int \*select;

int \*mac;

};

int Node::k = 0;

int Node::n = 0;

Node::Node(){

time = 0;

t = 0;

select = new int[n + 1];

mac = new int[k + 1];

int i;

for (i = 0; i <= n; ++i)

{

select[i] = 0;

}

for (i = 0; i<= k; ++i)

{

mac[i] = 0;

}

}

Node::Node(const Node &other){

t = other.t;

time = other.time;

mac = new int[k + 1];

select = new int[n + 1];

int i;

for (i = 1; i <= k; ++i)

{

mac[i] = other.mac[i];

}

for (i = 1; i <= n; ++i)

{

select[i] = other.select[i];

}

}

void Node::compute(int i){

if (time < mac[i])

{

time = mac[i];

}

}

class Schedule{

public:

Schedule(int n, int k, int \*data){

this->data = data;

this->n = n;

this->k = k;

MinMac = new int[n + 1];

MinTime = INT\_MAX;

};

int getMT(){return MinTime;}

int \*getMM(){return MinMac;}

void ExScedule();

protected:

int n;

int k;

int \*data;

int MinTime;

int \*MinMac;

};

void Schedule::ExScedule(){

queue<Node> Q;

Node::k = k;

Node::n = n;

Node initial;

Q.push(initial);

while(!Q.empty()){

Node father = Q.front();

Q.pop();

if (father.t == n)

{

if (father.time < MinTime)

{

MinTime = father.time;

for (int i = 1; i <= n; ++i)

{

this->MinMac[i] = father.select[i];

}

}

}

else

{

for (int i = 1; i <= k; ++i)

{

Node Child(father);

Child.t++;

Child.select[Child.t] = i;

Child.mac[i] += data[Child.t];

Child.compute(i);

if (Child.time <= MinTime)

{

Q.push(Child);

}

}

}

}

}

int main(){

int n,k,i,j;

cin>>n>>k;

int \*data = new int[n + 1];

for (i = 1; i <= n; ++i){cin>>data[i];}

Schedule test(n,k,data);

test.ExScedule();

cout<<test.getMT()<<endl;

return 0;

}