

D_Double's Journey

享受思考的过程

UVa 103 - Stacking Boxes

分类: [解题报告](#) [动态规划](#) [UVA OJ](#) 2012-08-19 00:04 258人阅读 [评论\(0\)](#) [收藏](#) [举报](#)

【题目链接】

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=114&page=show_problem&problem=39


【原题】





Stacking Boxes

Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an n -dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its ``lower-class'' cousin.

The Problem

Consider an n -dimensional ``box'' given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box  (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of n -dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes  such that each box  nests in box  ().

A box $D = (d_1, d_2, \dots, d_n)$ nests in a box $E = (e_1, e_2, \dots, e_n)$ if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E . This loosely corresponds to turning box D to see if it will fit in box E . However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).




For example, the box $D = (2,6)$ nests in the box $E = (7,3)$ since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E . The box $D = (9,5,7,3)$ does NOT nest in the box $E = (2,10,6,8)$ since no rearrangement of D results in a box that satisfies the nesting property, but $F = (9,5,7,1)$ does nest in box E since F can be rearranged as (1,9,5,7) which nests in E .

Formally, we define nesting as follows: box $D = (d_1, d_2, \dots, d_n)$ nests in box $E = (e_1, e_2, \dots, e_n)$ if there is a permutation π of $\{1, 2, \dots, n\}$ such that $(d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(n)})$ ``fits'' in (e_1, e_2, \dots, e_n) i.e., if $d_{\pi(i)} < e_i$ for all i .

The Input

The input consists of a series of box sequences. Each box sequence begins with a line

consisting of the the number of boxes k in the sequence followed by the dimensionality of the boxes, n (on the same line.)

This line is followed by k lines, one line per box with the n measurements of each box on one line separated by one or more spaces. The  line in the sequence () gives the measurements for the  box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the k boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The ``smallest'' or ``innermost'' box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

```
5
3 1 2 4 5
4
7 2 5 6
```

【题目大意】

给 n 维图形，它们的边长是 $\{d_1, d_2, d_3 \dots d_n\}$ ，对于两个 n 维图形，如果满足其中一个的所有边长按照任意顺序都一一对应小于另一个的边长，那么就锁可以嵌套到另一个。例如 $a\{1, 2\}$ ， $b\{2, 3\}$ ， a 所有边长都已一一对应小于 b 的边长，所以 a 能嵌套于 b 。

给 k 个 n 维图形，求它们最多可以连续嵌套多少个。

【分析与总结】

首先要判断两个图形是否可以嵌套，只需要把所有图形边长都按照从小到达排好序，那么对于a, b两个，只要按顺序一一比较它们的边数，如果满足所有 $a_i < b_i$ ，就所以a能嵌套于b。

然后，就是解法，这题有两种解法。

第一种就是所谓的用记忆化搜索求DAG模型（详见《算法入门经典》p161）。

我们可以用图的邻接矩阵来表示所有的关系，a“可嵌套于”b，那么就是 $G[a][b]=1$ 。然后这个问题就可以转化成求这张图的一个最长路径长度是多少。

第二种方法，假设a可嵌套于b用 $a < b$ 来表示，那么最长的一串就是 $a < b < c < d < e \dots$ ，可以看出非常像是一个“最长递增子序列”，这里的“递增”是指“维度”递增。

【代码】

方法一：记忆化搜索DAG模型

```
/*
 * UVa: 103 - Stacking Boxes
 * 记忆化搜索
 * Time: 0.016s
 * Author: D_Double
 */
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
int G[32][32], arr[32][12], d[32], n, k;
bool first;

//判断arr[a]是否比arr[b]大
inline bool isBigger(int a, int b){
    for(int i=0; i<k; ++i)
        if(arr[a][i]<=arr[b][i])return false;
    return true;
}

int dp(int i){
    if(d[i]!=-1)return d[i];
    int &ans=d[i]=1;
    for(int j=0; j<n; ++j)if(G[i][j]){
        ans = max(ans, dp(j)+1);
    }
    return ans;
}
```

```

void Print(int i){
    if(first) printf(" %d",i+1);
    else {printf("%d",i+1); first=true;}
    // printf("%d ", i+1);
    for(int j=0; j<n; ++j) if(G[i][j]&& d[j]+1==d[i]){
        Print(j);
        break;
    }
}

int main(){
    while(~scanf("%d%d",&n,&k)){
        for(int i=0; i<n; ++i){
            for(int j=0; j<k; ++j)
                scanf("%d",&arr[i][j]);
            sort(arr[i], arr[i]+k);
        }
        memset(G, 0, sizeof(G));
        for(int i=0; i<n-1; ++i){
            for(int j=i+1; j<n; ++j){
                if(isBigger(j,i))
                    G[i][j]=1;
                else if(isBigger(i,j))
                    G[j][i]=1;
            }
        }
        memset(d, -1, sizeof(d));
        int ans=-1, pos;
        for(int i=0; i<n; ++i){
            int t=dp(i);
            if(t>ans){
                ans=t;
                pos=i;
            }
        }
        printf("%d\n", ans);
        first=false;
        Print(pos);
        printf("\n");
    }
    return 0;
}

```

方法二：最长递增子序列

```

/*
 * UVa: 103 - Stacking Boxes
 * 最长递增子序列
 * Time: 0.056s
 * Author: D_Double
 */
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
int G[32][32], d[32], s[32],n, k;

```

```

bool first;

struct Node{
    int no;
    int A[12];
    int k;
    void Sort(){
        sort(A,A+k);
    }
    friend bool operator < (const Node&a, const Node&b){
        for(int i=0; i<a.k; ++i){
            if(a.A[i]>b.A[i])return false;
        }
        return true;
    }
}arr[32];

//判断arr[a]是否比arr[b]大
inline bool isBigger(int a, int b){
    for(int i=0; i<k; ++i)
        if(arr[a].A[i]<=arr[b].A[i])return false;
    return true;
}

void print(int i){
    if(s[i]!=i)print(s[i]);
    printf("%d ", arr[i].no);
}

int main(){
    while(~scanf("%d%d",&n,&k)){

        for(int i=0; i<n; ++i){
            for(int j=0; j<k; ++j)
                scanf("%d",&arr[i].A[j]);
            arr[i].no=i+1;
            arr[i].k=k;
            arr[i].Sort();
        }
        sort(arr,arr+n);
        memset(G, 0, sizeof(G));
        for(int i=0; i<n-1; ++i){
            for(int j=i+1; j<n; ++j){
                if(isBigger(j,i))
                    G[i][j]=1;
            }
        }
        int maxVal=1,pos=0;
        s[0]=0; d[0]=1;
        for(int i=1; i<n; ++i){
            d[i]=1; s[i]=i;
            for(int j=0; j<i; ++j)if(G[j][i]&& d[i]<d[j]+1){
                s[i]=j;
                d[i]=d[j]+1;
            }
            if(d[i]>maxVal){
                pos=i;
                maxVal=d[i];
            }
        }
        printf("%d\n", maxVal);
        print(pos);
    }
}

```

```
        printf("\n");  
    }  
    return 0;  
}
```

—— 生命的意义，在于赋予它意义。

原创 <http://blog.csdn.net/shuangde800> , By D_Double (转载
请标明)