程序设计实习
算法基础

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

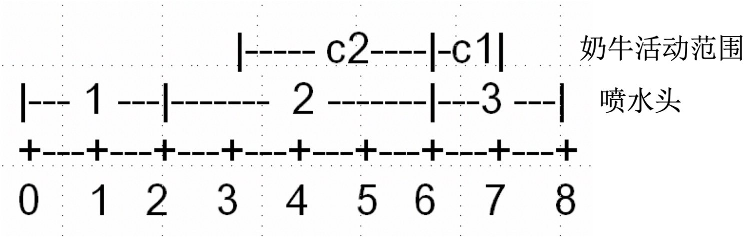
2024 年 5 月 31 日

在一片草场上: 有一条长度为 L $(1 \le L \le 1,000,000$，L 为偶数) 的线段。John 的 N $(1 \le N \le 1000)$ 头奶牛都沿着草场上这条线段吃草，每头牛的活动范围是一个开区间 (S, E)，S，E 都是整数。不同奶牛的活动范围可以有重叠。
John 要在这条线段上安装喷水头灌溉草场。每个喷水头的喷洒半径可以随意调节，调节范围是 $[A, B](1 \le A \le B \le 1000)$，$A$,$B$ 都是整数。要求

- 线段上的每个整点恰好位于一个喷水头的喷洒范围内
- 每头奶牛的活动范围要位于一个喷水头的喷洒范围内
- 任何喷水头的喷洒范围不可越过线段的两端 (左端是 0, 右端是 L)

请问，John 最少需要安装多少个喷水头。

例题 09：Dividing the Path 灌溉草场



在位置2和6，喷水头的喷洒范围不算重叠

例题 09：Dividing the Path 灌溉草场

输入
第 1 行：整数 N、L。
第 2 行：整数 A、B。
第 3 到 $N+2$ 行：每行两个整数 S、E $(0 \leq S < E \leq L)$，表示某头牛活动范围的起点和终点在线段上的坐标 (即到线段起点的距离)。

输出
最少需要安装的多少个喷水头；若没有符合要求的喷水头安装方案，则输出 -1。

输入样例
2 8
1 2
6 7
3 6
输出样例
3

例题 09：Dividing the Path 灌溉草场

从线段的起点向终点安装喷水头，令 $f(X)$ 表示：所安装喷水头的喷洒范围恰好覆盖直线上的区间 $[0, X]$ 时，最少需要多少个喷水头。

显然，X 应满足下列条件

- X 为偶数
- X 所在位置不会出现奶牛，即 X 不属于任何一个 (S, E)
- $X \geq 2A$
- 当 $X > 2B$ 时，存在 $Y \in [X - 2B, X - 2A]$ 且 Y 满足上述三个条件，使得 $f(X) = f(Y) + 1$

例题 09：Dividing the Path 灌溉草场

递推计算 $f(X)$

- $f(X)=\infty$:X 是奇数
- $f(X)=\infty$:$X<2A$
- $f(X)=\infty$:X 处可能有奶牛出没
- $f(X)=1$:$2A\leq X\leq 2B$ 、且 X 位于任何奶牛的活动范围之外
- $f(X)=1+\min\{f(Y):Y\in[X-2B,X-2A]$, Y 位于任何奶牛的活动范围之外$\}$:$X>2B$

递推计算 $f(X)$

- $f(X) = \infty$:X 是奇数
- $f(X) = \infty$:$X < 2A$
- $f(X) = \infty$:X 处可能有奶牛出没
- $f(X) = 1$:$2A \leq X \leq 2B$ 、且 X 位于任何奶牛的活动范围之外
- $f(X) = 1 + \min\{f(Y) : Y \in [X-2B, X-2A]$, Y 位于任何奶牛的活动范围之外$\}$:$X > 2B$

对每个 X 求 $f(X)$，都要遍历区间 $[X-2B, X-2A]$ 去寻找其中最小的 $f(Y)$, 则时间复杂度为: $L \times B = 1000000 \times 1000$，太慢

快速找到 $[X-2B, X-2A]$ 中使得 $f(Y)$ 最小的元素是问题求解速度的关键。

递推计算 $f(X)$

- $f(X) = \infty$:X 是奇数
- $f(X) = \infty$:$X < 2A$
- $f(X) = \infty$:X 处可能有奶牛出没
- $f(X) = 1$:$2A \leq X \leq 2B$ 、且 X 位于任何奶牛的活动范围之外
- $f(X) = 1 + \min\{f(Y) : Y \in [X-2B, X-2A]$, Y 位于任何奶牛的活动范围之外$\}$:$X > 2B$

对每个 X 求 $f(X)$，都要遍历区间 $[X-2B, X-2A]$ 去寻找其中最小的 $f(Y)$, 则时间复杂度为：$L \times B = 1000000 \times 1000$，太慢

快速找到 $[X-2B, X-2A]$ 中使得 $f(Y)$ 最小的元素是问题求解速度的关键。

可以使用优先队列 priority_queue!（multiset 也可以）！
求 $F(X)$ 时，若坐标属于 $[X-2B, X-2A]$ 的二元组 $(i, F(i))$ 都保存在一个 priority_queue 中，并根据 $F(i)$ 值排序，则队头的元素就能确保是 $F(i)$ 值最小的。

例题 09：Dividing the Path 灌溉草场

在求 X 点的 $F(X)$ 时，队列中最小点大于 $X-2A$ 的点，表示 $F(X)$ 无解。

队列中可以出现坐标小于 $X-2B$ 的点。这样的点若出现在队头，则直接将其抛弃。

求出 X 点的 F 值后，将 $(X, F(X))$ 放入队列

队列里只要存坐标为偶数的点即可

例题 09：Dividing the Path 灌溉草场

```cpp
#include <iostream>
#include <cstring>
#include <queue>
using namespace std;
const int INFINITE = 1<<30;
const int MAXL = 1000010;
int F[MAXL];  // F[L] 就是答案
int cowThere[MAXL]; //cowThere[i] 为 1 表示点 i 有奶牛
int N,L,A,B;
struct Fx {
  int x;
  int f;
  bool operator<(const Fx & a) const {
    if (a.f == f) return x > a.x;
    return f > a.f;
  }
  Fx(int xx=0,int ff=0):x(xx),f(ff) { }
};// 在优先队列里， f 值越小的越优先
priority_queue<Fx> qFx;
```

例题 09：Dividing the Path 灌溉草场

```
int main() {
  cin >> N >> L; cin >> A >> B;
  A <<= 1; B <<= 1; //A,B 的定义变为覆盖的直径
  memset(cowThere,0,sizeof(cowThere));
  for (int i = 0; i < N; ++i) {
    int s, e;
    cin >> s >> e;
    ++cowThere[s+1]; //从 s+1 起进入一个奶牛区
    --cowThere[e]; //从 e 起退出一个奶牛区
  }
  int inCows = 0; //表示当前点位于多少头奶牛的活动范围之内
  for (int i = 0; i <= L; ++i) { //算出每个点是否有奶牛
    F[i] = INFINITE;
    inCows += cowThere[i];
    cowThere[i] = inCows > 0;
  }
  for (int i = A; i <= B; i += 2) {//初始化队列
    if (cowThere[i]) continue;
    F[i] = 1;
    qFx.push(Fx(i, 1));
  }
  for (int i = B + 2; i <= L; i += 2 ) {
    if (cowThere[i]) continue;
    while (!qFx.empty() && qFx.top().x < i - B) qFx.pop();
    if (!qFx.empty() && qFx.top().x <= i - A) {
      F[i] = qFx.top().f + 1;
      qFx.push(Fx(i, F[i]));
    }
  }
  if (F[L] == INFINITE) cout << -1 <<endl;
  else cout << F[L] << endl;
  return 0;
} // 复杂度: O(nlogn)
```

手工实现优先队列的方法

如果一个队列满足以下条件:

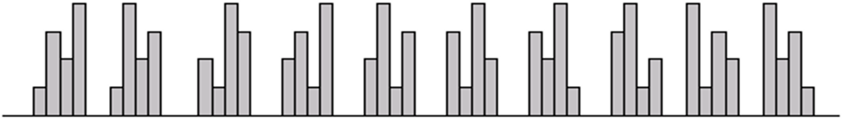
1. 开始为空
2. 每在队尾加入一个元素 a 之前，都从现有队尾往前删除元素，一直删到碰到小于 a 的元素为止，然后再加入 a

那么队列就是递增的，当然队头的元素，一定是队列中最小的

例题 10：一个美妙的栅栏

N 个木棒, 长度分别为 $1, 2, \ldots, N$.
构成美妙的栅栏

- 除了两端的木棒外，每一跟木棒，要么比它左右的两根都长，要么比它左右的两根都短。
- 即木棒呈现波浪状分布, 这一根比上一根长了，那下一根就比这一根短，或反过来



All cute fences made of $N = 4$ planks, ordered by their catalogue numbers.

问题: 符合上述条件的栅栏建法有很多种，对于满足条件的所有栅栏, 按照字典序 (从左到右, 从低到高) 排序。
给定一个栅栏的排序号，请输出该栅栏, 即每一个木棒的长度.

例题 10：一个美妙的栅栏

输入数据
第一行是测试数据的组数 K $(1 \le K \le 100)$。接下来的 K 行，每一行描述一组输入数据.
每一组输入数据包括两个整数 N 和 C. N ($1 \le N \le 20$) 表示栅栏的木棒数，C 表示要找的栅栏的排列号.

输出数据
输出第 C 个栅栏，即每一个木棒的长度

设 20 个木棒可组成的栅栏数是 T; 我们假设 T 可以用 64-bit 长整数表示，$1 < C \le T$

输入样例

2
2 1
3 3

输出样例

1 2
2 3 1

例题 10：一个美妙的栅栏

解题思路

问题抽象：给定 1 到 N 这 N 个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列，问第 C 个排列是一个怎样的排列

总体思想
排列计数 $+$ 动规

例题 10：一个美妙的栅栏

排列计数

如 1，2，3，4 的全排列，共有 4！种，求第 10 个的排列是（从 1 计起)?

先试首位是 1，后 234 有 3！ =6 种 <10, 说明首位 1 偏小，问题转换成求 2 开头的第（10-6=4）个排列，而 3！ =6 >= 4，说明首位恰是 2。

第二位先试 1（1 没用过），后面 2！ =2 个 <4,1 偏小，换成 3（2 用过了）为第二位，待求序号也再减去 2!，剩下 2 了。而此时 2!>=2，说明第二位恰好是 3。

第三位先试 1，但后面 1！ <2，因此改用 4。末位则是 1 了。

这样得出，第 10 个排列是 2-3-4-1。

例题 10：一个美妙的栅栏

共 N 根长度不一的木棒。本题待求方案的序号为 C

先假设第 1 短的木棒作为第一根，看此时的方案数 $P(N,1)$ 是否 $\geq C$, 如果否，则应该用第二短的作为第一根，C 减去 $P(N,1)$ ，再看此时方案数 $P(N,2)$ 和 C 比如何。如果还 $< C$ ，则应以第三短的作为第一根,C 再减去 $P(N,2)$ ....

$P(i,j)$ 表示：有 i 根木棒，以其中第 j 短的作为第一根，在此情形下能构成的美妙栅栏数目。

若发现第 i 短的作为第一根时，方案数已经不小于 C，(即 $P(N,i) \geq C$,C 是不断减小的) 则确定应该以第 i 短的作为第一根，然后再去确定第二根....

即接下来试 $P(N-1,1)$, $P(N-1,2)$......

解题关键: 求出所有 $P(i,j)$

例题 10：一个美妙的栅栏

动规解题思路

令 $S(i)$ 表示由 i 根木棒构成的合法方案集合

令 $B[i][k]$ 表示：$S(i)$ 中以这 i 根木棒里第 k 短的木棒打头的方案数

要看看 $B[i][k]$ 和 $B[i-1][n]$ 的关系 $(n = 1 \ldots i-1)$

动规解题思路

在选定了某根木棒 x 作为 i 根木棒中的第一根木棒的情况下，假定剩下 $i-1$ 根木棒的合法方案数是 $A[i-1]$。但是，这 $A[i-1]$ 种方案，并不是每种都能和 x 形成新的合法方案。将第一根比第二根长的方案称为 DOWN 方案，第一根比第二根短的称为 UP 方案，则，$S(i-1)$ 中，第一根木棒比 x 长的 DOWN 方案，以及第一根木棒比 x 短的 UP 方案，才能和 x 构成 $S(i)$ 中的方案。

$$B[i][k] = \sum_{M=k}^{i-1} B[i-1][M]_{(DOWN)} + \sum_{N=1}^{k-1} B[i-1][N]_{(UP)}$$

没法直接推。

例题 10：一个美妙的栅栏

动规解题思路

于是把 B 分类细化，即加一维.

$B[i][k] = C[i][k][DOWN] + C[i][k][UP]$
$C[i][k][DOWN]$ 是 $S(i)$ 中以第 k 短的木棒打头的 DOWN 方案数。然后试图对 C 进行动规

$$C[i][k][UP] = \sum_{M=k}^{i-1} C[i-1][M][DOWN]$$

$$C[i][k][DOWN] = \sum_{N=1}^{k-1} C[i-1][N][UP]$$

初始条件：$C[1][1][UP] = C[1][1][DOWN] = 1$

例题 10：一个美妙的栅栏

经验：当选取的状态，难以进行递推时（分解出的子问题和原问题形式不一样，或不具有无后效性），考虑将状态增加限制条件后分类细化，即增加维度，然后在新的状态上尝试递推

例题 10：一个美妙的栅栏

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int UP =0;
const int DOWN =1;
const int MAXN = 25;
long long C[MAXN][MAXN][2];  //C[i][k][DOWN] 是 S(i) 中以第 k 短的木棒打头的 DOWN 方案数,C[i][k][UP] 是 S(i)
void Init(int n) {
  memset(C,0,sizeof(C));
  C[1][1][UP] = C[1][1][DOWN] = 1;
  for (int i = 2 ;i <= n; ++i) {
    for (int k = 1; k <= i; ++k) { //枚举第一根木棒的长度
      for (int M = k; M < i ; ++M) //枚举第二根木棒的长度
        C[i][k][UP] += C[i-1][M][DOWN];
      for (int N = 1; N <= k-1; ++N) //枚举第二根木棒的长度
        C[i][k][DOWN] += C[i-1][N][UP];
    }
  }
//总方案数是 Sum{ C[n][k][DOWN] + C[n][k][UP] }  k = 1.. n;
}
void Print(int n,long long cc);
int main() {
  int T,n;
  long long c;
  Init(20);
  scanf("%d",&T);
  while(T--)     {
    scanf("%d %lld",&n,&c);
    Print(n,c);
  }
  return 0;
}
```

例题 10：一个美妙的栅栏

```cpp
void Print(int n,long long cc) {
  int used[MAXN]; //木棒是否用过
  int seq[MAXN]; //最终要输出的答案
  memset(used,0,sizeof(used));
  for (int i = 1; i <= n; ++i) {//依次确定每一个位置 i 的木棒序号
    int No = 0; //位置 i 的木棒 k 是剩下的木棒里的第 No 短的,No 从 1 开始算 ^^I
    long long skipped = 0; //已经跳过的方案数
    int k;
    for (k = 1; k <= n; ++k) {
      if (used[k]) continue; //长度为 k 的木棒用过就尝试下一个
      ++No; //k 是剩下的木棒里的第 No 短的
      if (i == 1) {
          skipped = C[n][No][UP]+C[n][No][DOWN];
      } else {
        if (k > seq[i-1] && (i <= 2 || seq[i-2] > seq[i-1])) //合法放置
          skipped = C[n-i+1][No][DOWN];
        else if(k < seq[i-1] && (i <= 2 || seq[i-2] < seq[i-1]))
          skipped = C[n-i+1][No][UP];
      } //if( i == 1)
      if (skipped >= cc) break;
      cc -= skipped;
    }// for( k = 1; k <=n; ++k)
    used[k] = 1;
    seq[i] = k;
  }
  for (int i = 1; i <= n; ++i)
    cout << seq[i] << " ";
  cout << endl;
}
```

状态压缩动态规划

有时，状态相当复杂，看上去需要很多空间，比如一个数组才能表示一个状态，那么就需要对状态进行某种编码，进行压缩表示。

比如：状态和某个集合有关，集合里可以有一些元素，没有另一些元素，那么就可以用一个整数表示该集合，每个元素对应于一个 bit，有该元素，则该 bit 就是 1。

例题 11：TSP 问题

N 个城市，编号 1 到 N。$(N \leq 16)$。
任意两个城市间都有路，A->B 和 B->A 的路可能不一样长。
已知所有路的长度，问经每个城市恰好一次的最短路径的长度

例题 11：TSP 问题

用 $dp[s][j]$ 表示经过集合 s 中的每个点恰好一次，且最后走的点是 j ($j \in s$) 的最佳路径的长度。

最终就是要求：

$$\min_{0 \le j < N} dp[all][j]$$

all 是所有点的集合

例题 11：TSP 问题

状态方程：

$$dp[s][j] = \min_{j \in s, s' = s - j, k \in s'} \{dp[s'][k] + w[k][j]\}$$

($w[k][j]$ 是 k 到 j 的边权值)

边界条件：$dp[\{i\}][i] = 0$

例题 11：TSP 问题

如何表示点集 s ?

由于只有 16 个点，可以用一个 short 变量表示点集。每个点对应一个 bit。例如：

$5 = 0000000000000101_2$

5 代表的点集是 0,2

全部 n 个点的点集，对应的整数是：$(1 << n) - 1$

最终要求：$\min\limits_{0 \le j < n} dp[(1 << n) - 1][j]$

例题 11：TSP 问题

如何进行集合操作?

位运算。例：从集合 i 中去掉点 j，得到新集合 s':

$s' = s\&(\sim(1 << j))$
或
$s' = s - (1 << j)$

状态数目：$dp[s][j]$ s: $0-(2^n-1)$ j: $0-(n-1)$

状态转移：$O(n)$

总时间：$O(n^2 2^n)$

硬枚举: $O(n!)$

例题 12：大炮阵地

司令部的将军们打算在 N*M($1 \le n \le 100, 1 \le m \le 10$) 的网格地图上部署他们的大炮。一个 N*M 的地图由 N 行 M 列组成，地图的每一格可能是山地（用"H" 表示），也可能是平原（用"P" 表示），如下图。在每一格平原地形上最多可以布置一门大炮（山地上不能够部署大炮）；

P	P	H	P	H	H	P	P
P	H	P	H	P	H	P	P
P	P	P	H	H	H	P	H
H	P	H	P	P	P	P	H
H	P	P	P	P	H	P	H
H	P	P	H	P	H	H	P
H	H	H	P	P	P	P	H

如果在地图中的灰色所标识的平原上部署一门大炮，则图中的黑色的网格表示它能够攻击到的区域：沿横向左右各两格，沿纵向上下各两格。图上其它白色网格均攻击不到。从图上可见大炮的攻击范围不受地形的影响。
现在，将军们规划如何部署大炮，在防止误伤的前提下（保证任何两门大炮之间不能互相攻击，即任何一门大炮都不在其他支大炮的攻击范围内），在整个地图区域内最多能够摆放多少大炮。

例题 12：大炮阵地

思路：如果用 $dp[i]$ 表示前 i 行所能放的最多大炮数目，能否形成递推关系?

例题 12：大炮阵地

思路：如果用 $dp[i]$ 表示前 i 行所能放的最多大炮数目，能否形成递推关系？

显然不能。因为不满足无后效性

例题 12：大炮阵地

思路：如果用 $dp[i]$ 表示前 i 行所能放的最多大炮数目，能否形成递推关系？

显然不能。因为不满足无后效性

按照加限制条件加维度的思想，加个限制条件：
$dp[i][j]$ 表示第 i 行的大炮布局为 j 的前提下，前 i 行所能放的最多大炮数目

例题 12：大炮阵地

思路：如果用 $dp[i]$ 表示前 i 行所能放的最多大炮数目，能否形成递推关系?

显然不能。因为不满足无后效性

按照加限制条件加维度的思想，加个限制条件：
$dp[i][j]$ 表示第 i 行的大炮布局为 j 的前提下，前 i 行所能放的最多大炮数目

布局为 j 体现了状态压缩。j 是个 10 位二进制数，表示一行大炮的一种布局。有大炮的位置，对应位为 1，没有大炮的位置，对应位为 0

例题 12：大炮阵地

思路：如果用 $dp[i]$ 表示前 i 行所能放的最多大炮数目，能否形成递推关系？

显然不能。因为不满足无后效性

按照加限制条件加维度的思想，加个限制条件：
$dp[i][j]$ 表示第 i 行的大炮布局为 j 的前提下，前 i 行所能放的最多大炮数目

布局为 j 体现了状态压缩。j 是个 10 位二进制数，表示一行大炮的一种布局。有大炮的位置，对应位为 1，没有大炮的位置，对应位为 0

依然不满足无后效性。因仅从 $dp[i-1][k]$ $(k = 0 \ldots 1023)$ 无法推出 $dp[i][j]$。达成 $dp[i-1][k]$ 可能有多种方案，有的方案允许第 i 行布局为 j, 有的方案不允许第 i 行布局为 j，然而却没有信息可以用来进行分辨。

例题 12：大炮阵地

再加限制条件，再加一维 (多加的状态变量用于补充必要的信息)：

$dp[i][j][k]$ 表示第 i 行布局为 j, 第 $i-1$ 行布局为 k 时, 前 i 行的最多大炮数目。

1. j, k 这两种布局必须相容。否则 $dp[i][j][k] = 0$
2. $dp[i][j][k] = \max\limits_{m=0...1023} dp[i-1][k][m] + Num(j)$,
   $Num(j)$ 为布局 j 中大炮的数目, j 和 m 必须相容, k 和 m 必须相容。此时满足无后效性
3. 初始条件：
   $dp[0][j][0] = Num(j)$
   $$dp[1][i][j] = \begin{cases} 0, & \text{i,j 不相容} \\ dp[0][j][0] + Num(i), & \text{i,j 相容} \end{cases}$$

例题 12：大炮阵地

问题：dp 数组为：

```
int dp[100][1024][1024]
```

太大，时间复杂度和空间复杂度都太高。

解决：
每一行里最多能放 4 个大炮。就算全是平地，能放大炮的方案数目也不超过 60 (用一遍 dfs 可以全部求出)

算出一行在全平地情况下所有大炮的排列方案，存入数组 $state[70]$

```
int dp[100][70][70]
```

足矣

$dp[i][j][k]$ 表示第 i 行布局为 $state[j]$, 第 $i-1$ 行布局为 $state[k]$ 时, 前 i 行的最多大炮数目。

例题 12：大炮阵地

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
const int MAXN = 105, MAXM = 10;
int state[MAXN], num[MAXN], pstate[MAXN],cntOne[1 << MAXM];
int dp[MAXN][MAXN][MAXN];
int main() {
  int n, m, cnt = 0;
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    for (int j = 0; j < m; j++)
      if(s[j] == 'H') pstate[i] += (1 << j);
  }
  for (int j = 0; j < (1 << m); j++) {
    cntOne[j] = cntOne[j/2] + j % 2;
    if ((j & (j << 1)) || (j & (j << 2))) continue;
    num[cnt] = cntOne[j];
    state[cnt++] = j;
  }
  for (int i = 0; i < cnt; i++) {
    if (pstate[0] & state[i]) continue;
    dp[0][i][0] = num[i];
  }
```

例题 12：大炮阵地

```
  for (int i = 0; i < cnt; i++) {
    if (pstate[1] & state[i]) continue;
    for (int j = 0; j < cnt; j++) {
      if (pstate[0] & state[j]) continue;
      if (state[i] & state[j]) continue;
      dp[1][i][j] = dp[0][j][0] + num[i];
    }
  }
  for (int i = 2; i < n; i++) {
    for (int j = 0; j < cnt; j++) {
      if (pstate[i] & state[j]) continue;
      for (int k = 0; k < cnt; k++) {
        if (pstate[i - 1] & state[k]) continue;
        if (state[j] & state[k]) continue;
        for (int m = 0; m < cnt; m++) {
          if (pstate[i - 2] & state[m]) continue;
          if ((state[j] & state[m]) || (state[k] & state[m])) continue;
          dp[i][j][k] = max(dp[i][j][k], dp[i - 1][k][m] + num[j]);
        }
      }
    }
  }
  cout << *max_element(&dp[n-1][0][0], &dp[n-1][cnt-1][cnt - 1]) << endl;
  return 0;
}
```

例题 13：课程大作业

小明是北京大学信息科学技术学院三年级本科生。他喜欢参加各式各样的校园社团。这个学期就要结束了，每个课程大作业的截止时间也快到了，可是小明还没有开始做。每一门课程都有一个课程大作业，每个课程大作业都有截止时间。如果提交时间超过截止时间 X 天，那么他将会被扣掉 X 分。对于每个大作业，小明要花费一天或者若干天来完成。他不能同时做多个大作业，只有他完成了当前的项目，才可以开始一个新的项目。小明希望你可以帮助他规划出一个最好的办法 (完成大作业的顺序) 来减少扣分。

例题 13：课程大作业

输入
输入包含若干测试样例。
输入的第一行是一个正整数 T，代表测试样例数目。
对于每组测试样例，第一行为正整数 N（$1 \leq N \leq 15$）代表课程数目。
接下来 N 行，每行包含一个字符串 S（不多于 50 个字符）代表课程名称和两个整数 D（代表大作业截止时间）和 C（完成该大作业需要的时间）。
注意所有的课程在输入中出现的顺序按照字典序排列。

输出
对于每组测试样例，请输出最小的扣分以及相应的课程完成的顺序。
如果最优方案有多个，请输出字典序靠前的方案。

例题 13：课程大作业

样例输入

```
2
3
Computer 3 3
English 20 1
Math 3 2
3
Computer 3 3
English 6 3
Math 6 3
```

样例输出

```
2
Computer
Math
English
3
Computer
English
Math
```

解题思路：

$dp[s]$ 表示已经完成的作业集合为 s 时，所能达到的最少扣分
状态方程：$dp[s] = \min_{j \in s, s' = s - j} dp[s'] + c(j)$
$c(j)$ 表示，完成 s' 后，再完成 j 所造成的扣分

例题 13：课程大作业

解题思路：
由于要记录完成作业的过程，所以，在每个状态，不但要记录到达该状态的最小扣分，还要记录当初是从哪个状态转移到目前这个状态的（即计算出 $dp[s]$ 时，$dp[s]$ 里面应该要记录计算时选出的最优的那个 s'，这样从终态出发，就能往回依次找到状态转移的路径（作业完成的顺序）

解题思路：
由于要记录完成作业的过程，所以，在每个状态，不但要记录到达该状态的最小扣分，还要记录当初是从哪个状态转移到目前这个状态的（即计算出 $dp[s]$ 时，$dp[s]$ 里面应该要记录计算时选出的最优的那个 s'，这样从终态出发，就能往回依次找到状态转移的路径（作业完成的顺序）
dp 数组可以如下定义：

```
struct Node  {
  int pre;       //上一个状态（比当前状态完成的作业少了 1 个）
  int minScore; //到达当前状态的最低扣分
  int last;      //当前状态下，最后完成的作业的编号
  int finishDay; //作业 last 完成的时间
} dp[(1 << 16) + 10] ;
```

则 $dp[i]$ 代表状态 i 的情况,i 的上一个状态就是 $dp[i].pre$

例题 13：课程大作业

解题思路：
由于要记录完成作业的过程，所以，在每个状态，不但要记录到达该状态的最小扣分，还要记录当初是从哪个状态转移到目前这个状态的（即计算出 $dp[s]$ 时，$dp[s]$ 里面应该要记录计算时选出的最优的那个 s'，这样从终态出发，就能往回依次找到状态转移的路径（作业完成的顺序）
dp 数组可以如下定义：

```
struct Node  {
  int pre;       //上一个状态（比当前状态完成的作业少了 1 个）
  int minScore; //到达当前状态的最低扣分
  int last;      //当前状态下，最后完成的作业的编号
  int finishDay; //作业 last 完成的时间
} dp[(1 << 16) + 10] ;
```

则 $dp[i]$ 代表状态 i 的情况,i 的上一个状态就是 $dp[i].pre$
边界条件:

$dp[0].minScore = 0$

递推顺序: $dp[0]$->$dp[1]$->$dp[2]$ ...->$dp[1 << m-1]$ (共 m 个大作业)

字典序问题：
按如下公式计算 $dp[s].minScore$ 时:
$dp[s] = \min_{j \in s, s' = s - j} dp[s'] + c(j)$
如果发现有一个新的 s', 导致 $dp[s'] + c(j)$ (先完成 s'，再完成作业 j) 和当前 $dp[s]$ 相等，则由 s 出发，
$dp[s].last- > dp[s.pre].last- > dp[dp[s.pre].pre].last- > \ldots$ 就是当前作业完成顺序的逆。$j- > dp[s'].last- > dp[[dp[s'].pre]].last- > \ldots$ 就是另一条同样优的作业完成顺序的逆。比较这两个顺序的字典序。如果从 j 出发的更小，则更新 $dp[s].last$ 为 j, $dp[s].pre$ 为 s' , 相应的 $finishDay$ 也更新