数与图的完美结合

-------浅析差分约束系统

华中师大一附中　冯威

[摘要]

在面对多种多样的问题时，我们经常会碰到这样的情况：往往我们能够根据题目题面意思来建立一些简单的模型，但却面对这些模型无从下手。这时我们应该意识到，也许能够将这种模型与其他的模型之间搭起一座桥梁，使我们能够用更简单直接的方式解决它。这里我们介绍一种方法，它很好地将某些特殊的不等式组与图相联结，让复杂的问题简单化，将难处理的问题用我们所熟知的方法去解决，它便是**差分约束系统**。这里我们着重介绍**差分约束系统**的原理和其需要掌握的 **bellman-ford** 算法。然后通过 zju1508 和 zju1420 两道题目解析差分约束系统在信息学题目中的应用，并逐渐归纳解决这类问题的思考方向。

[目录]

[关键字] 差分约束系统、不等式、单元最短路径、转化

[正文]

在分析差分约束系统之前，我们首先介绍一个解决单元最短路径问题的 Bellman Ford 算法，它的应用十分广泛，在差分约束系统中更充当着重要的角色。

Bellman-ford 算法

算法简单介绍

这个算法能在更一般的情况下解决最短路的问题。何谓一般，一般在该算法下边的权值可以为负，可以运用该算法求有向图的单元最长路径或者最短路径。我们这里仅以最短路径为例。

Bellman ford 类似于 Dijkstra 算法，对每一个节点 v∈V，逐步减小从起点 s 到终点 v 最短路的估计量 dist[v]直到其达到真正的最短路径值 mindist[v]。Bellman-ford 算法同时返回一个布尔值，如果不存在从源结点可达的负权回路，算法返回布尔值 TRUE，反之返回 FALSE。

算法具体流程

1. 枚举每条边(u,v)∈E（G）。
2. 对枚举到的边进行一次更新操作。
3. 回到步骤 1，此过程重复 n-1 次，以确定没有更可以优化的情况。
4. 枚举每条边（u，v）若仍然存在可以更新的边，则说明有向图中出现了负权回路，于是返回布尔值 FALSE。
5. 返回布尔值 TRUE。

注：这里的更新操作是一种**松弛技术**，以单元最短路径为例这个操作就是保证 dist[v]<=dist[u]+w[u,v]，即 if dist[v]>dist[u]+w[u,v] then dist[v]=dist[u]+w[u,v]，如果是最长路径则是保证 dist[v]>=dist[u]+w[u,v]。

定义一个有向图 G=（V，E），w（u，v）表示由结点 u 到 v 的边的权值。

伪代码如下：

```
For i=1 to |V|G||-1
  Do for 每条边(u，v)∈E|G|
      Do 更新操作（u，v，w（u，v））
For 每条边（u，v）∈E|G|
  Do if 仍然有可更新内容 then return False
Return True
```

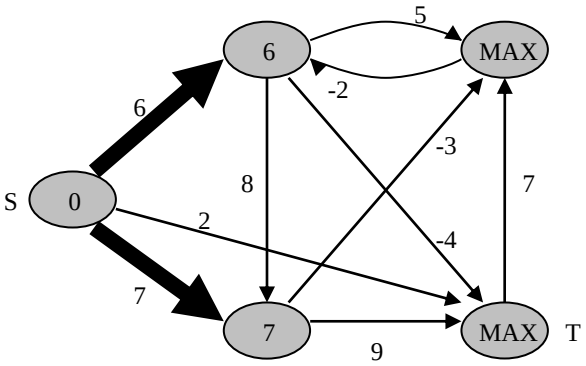
下图描述了该算法的操作过程，粗线条代表已更新线段

图例中，S 为源节点，粗线断覆盖的边表示最近一次执行更新操作的边。算法执行|V|-1 次操作，每次操作都对所有可以进行松弛操作的边进行扩展。

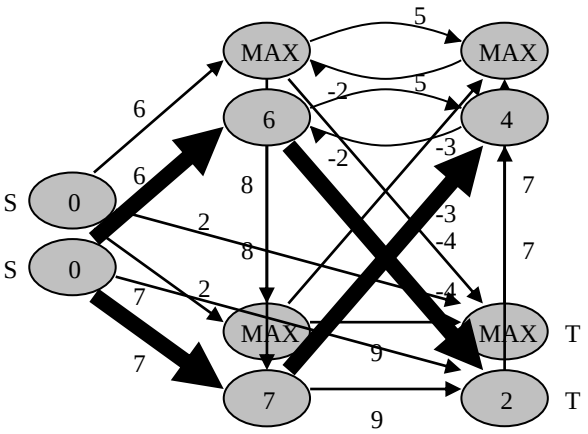
证明一下 Bellman-Ford 算法的正确性。

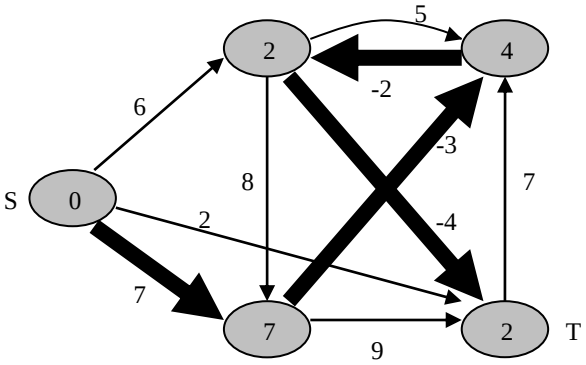
1．设 G=（V，E）为有向加权图，源节点为 S，加权函数为 w：E-〉R。如果有负权回路则 Bellman_ford 算法一定会返回布尔值 false,否则返回 TRUE。

证明略。

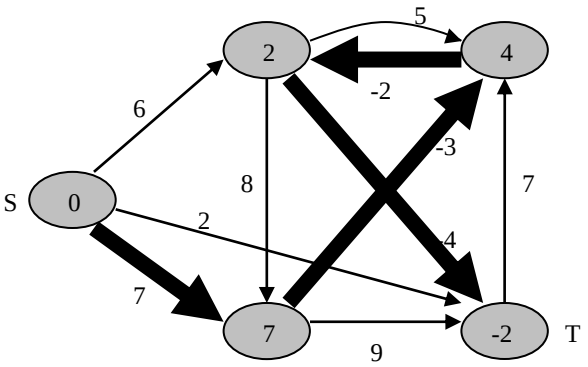


2．设 G=（V，E）为有向加权图，





源节点为 S 加权函数为 w：E-〉R，并且 G 不含从 s 可达的负权回路，则算法 Bellman_ford



终止时，对所有从 s 可达的结点 v 有 d[v]=mindist（s，v）。

证明：设 v 为从 s 可达的节点，且 p=<v0,v1,..,vk>为从 s 到 v 的一条最短路径，其中 v0=s，vk=v。因为路径 p 是简单路径，所以 k<=|V|-1。我们希望通过归纳证明对 i=0,1,…,k，在对 G 的边进行完第 i 趟操作后有 d[vi]=mindist(s,vi),且该等式此后一直保持成立，因为总共有 |V|-1 次操作，所以上述结论证明命题成立。

我们发现与 Dijkstra 不同，Bellman ford 更多的是对边进行操作，在稀疏图，即点多边少的图中，用 Bellman Ford 更能高效的解决单元最短路径问题。下面一道例题来进一步熟悉 Bellman Ford 并与 Dijkstra 作一下简单的时间上的比较。

例题一：ZJU2008 Invitation Cards

[题目大意]在有向加权图中 G（V，E），邮局要从起点 S 向其他 n 个节点发送邮件，于是派出 n 个邮递员，分别到达其他 n 个地点发送，然后回到起点 S，求出所有邮递员所经过的总路程的最小值。

[数据范围] 1 <= P,Q <= 1000000。P 表示节点数目，Q 表示边的个数。

[题目分析]这道题算法很简单，即求出从 s 到任意点的最短路径，求其和 ANS1，再将所有有向边反向（这个过程将求从另外 n 个结点到 s 的最短路径转化为从 s 到其他 n 个点的最短路径），再求一次 s 到任意点的最短路径，求其和 ANS2，Ans=Ans1+Ans2 即为所求。

但是，此题难在数据量上，无论是 Dijkstra 的 $O(V^2)$ 的算法，还是 Bellmanford 的 O(VE) 的算法都无法在数据规模最大的情况下在 5s 的时间内得出结果，于是需要做出一点优化。

关于 Dijkstra 的优化：用堆维护使寻找需要扩展的点的过程复杂度降低为 1。复杂度大幅降低。

关于 Bellman ford 的优化：可以将 Bellman ford 需要扩展的点放进队列中，扩展顺序有了新的变化。用一个队列 queue 表示需要更新的点，每次取队列头指针 fp 所指的点 u，搜索所有的边(u,v)属于 E，如果 dist[u]+w[u,v] 比 dist[v]更优则更新 dist[v]，尾指针 rp 后挪一位，将 v 点加入队列 queue。这样的优化避免了很多重复的操作，事实证明它的效率仅次于 Dijkstra+heap 的效率。下面用 3 种不同的方法来解这道题目，看看结果如何。

我们用以下三种方法求最短路径:

1. Dijkstra
2. Bellman Ford
3. Dijkstra&Heap

在 ZJU Online Judge 中进行评测结果如下：

算法	运行所有数据所用时间
Dijkstra	〉5s （TLE）
Bellman Ford	1.46s
Dijkstra&Heap	1.24s

差分约束系统

对于解决差分约束系统问题的操作过程和使用原理，我们通过下面一道简单的题目进行了解。

引例：[例题二] Zju1508 Interval

题目大意：

有一个序列，题目用 n 个整数组合 [ai,bi,ci]来描述它，[ai，bi，ci]表示在该序列中处于[ai，bi]这个区间的整数至少有 ci 个。如果存在这样的序列，请求出满足题目要求的最短的序列长度是多少。如果不存在则输出 -1。

输入：第一行包括一个整数 n，表示区间个数，以下 n 行每行描述这些区间，第 i+1 行三个整数 ai，bi，ci，由空格隔开，其中 0<=ai<=bi<=50000 而且 1<=ci<=bi-ai+1。

输出：一行，输出满足要求的序列的长度的最小值。

[浅析]初看此题感觉无从下手，我们将范围和个数进行量化，让问题看起来更加简单些。

将问题数字化：

若记，$t_i = \begin{cases} 1 & \text{如果i存在于序列之中} \\ 0 & \text{如果i不存在于序列中} \end{cases}$

那么本题约束条件即为 $\sum_{j=a_i}^{b_i} t_j \geq c_i \ (i = 1,2,\ldots,n)$

建立不等式模型：

这样的描述使一个约束条件所牵涉的变量太多，不妨设 $S_i = \sum_{j=1}^{i} t_j$ (i=1,2,…,n)

约束条件即可用以下不等式表示：

$$S_{b_i} - S_{a_i - 1} \geq c_i (i = 1,2,\ldots,n)$$

值得注意的是，这样定义的 S 若仅仅满足约束条件的要求是不能完整体现它的意义的，S 中的各个组成之间并不是相对独立的，他们存在着联系。

由于 $S_i = \sum_{j=1}^{i} t_j$，且 t 要么为 1，要么为 0，则 S_i 一定比 S_{i-1} 大 ，且至多大 1。于是有：

$$S_i - S_{i-1} \geq 0 (i = 1,2,\ldots,n)$$
$$S_{i-1} - S_i \geq -1 (i = 1,2,\ldots,n)$$

那么如何找到满足要求的这样一组 S，且使其个数最少呢？

这里我们看到最少就会联想到贪心，确实这道题目用贪心可行，且不失为一种很好的做法，但在这里我们不作多的介绍。我们将针对这个不等式组进行联想，迁移。注意到不等式中只涉及到了 2 个未知数，用减号相连接。这样的式子是否与过去曾经学过的 Bellmanford 中的松弛操作所达到的效果相类似呢？

联想：

我们需要寻找一个满足以下要求的 S 数组

$$S_{b_i} - S_{a_i - 1} \geq c_i (i = 1,2,\ldots,n)$$
$$S_i - S_{i-1} \geq 0 (i = 1,2,\ldots,n)$$
$$S_{i-1} - S_i \geq -1 (i = 1,2,\ldots,n)$$

而 Bellman-Ford 每次的更新操作为

$$\text{If } d[u] + w[u,v] \leq d[v] \text{ then } d[v] \leftarrow d[u] + w[u,v]$$

即在经过若干次更新操作将要保证 d[v]<=d[u]+w[u,v]

这里我们也有一个已知量-----边的权值，即 w[u,v]，于是整理一下得

d[u]-d[v]>=w[u,v]

经过这样的变形，不难看出，两个式子有着极为惊人的相似！

于是做出如下的转化：

1. 我们将 $S_0, S_1, \ldots, S_n$，看作 n+1 个点 $V_0, V_1, \ldots, V_n$。

2. 若 S_i 和 S_j 之间有着约束关系，例如 $S_{i-1}-S_i >= -1$,那么我们从结点 V_{i-1} 向结点 V_i 连上一条有向边，边的权值为-1。

这样如果我们从 V_0 出发，求出结点 V_0 到 V_n 的最短路径，则 S_n 为满足要求情况下的最小值。相反如果我们发现在 Bellmanford 算法执行的过程中存在有负权回路，则说明不存在满足要求的式子。

于是通过合理地建立数学模型，将不等式图形化，用 Bellmanford 作为武器，最终此题得到了圆满的解决！

思考：

本题只要求输出最少的个数，那么能否通过上述方法构造出相应的数列呢？答案是肯定的。其实在 Bellmanford 的过程中，我们已经得到了所有结点上 S 的值，那么由 $T_i = S_i - S_{i-1}$ 求出所有 T，当 T_i 为 1 则输出 i，即可输出满足题目要求的序列。

总结：

对于 ZJU1508 的问题，最终可以化为一类线性不等式定义的线性函数。而这类问题其实是可以转化为单元最短路径问题，从而用刚才所准备到的 Bellman Ford 算法来解决它。

线性程序设计：

我们为线性程序设计问题制定一个严格的数学描述：

给定一个 m*n 矩阵 A，维向量 b 和维向量 c，我们希望找出由 n 个元素组成的向量 x，在由 Ax<=b 所给出的 m 个约束条件下，使目标函数 $\sum_{i-1}^{n} c_i x_i$ 达到最大值。

其实很多问题都可以通过这样一个线性程序设计框架来进行描述。在实际的问题中，也经常要对其进行分析和解决。上述例题使我们对这一类线性程序设计问题提供了一个多项式时间的算法。它将一类特殊的线性不等式与图论紧密联系在了一起。这类特殊的线性不等

式，我们称它为**差分约束系统**，它是可以用单元最短路径来求解的。

差分约束系统：

差分约束系统是一个线性程序设计中特殊的一种，线性程序设计中矩阵 A 的每一行包含一个 1 和一个-1，A 的所有其他元素均为 0。由 Ax<=b 给出的约束条件形成 m 个差分约束的集合，其中包含 n 个未知单元。每个约束条件均可够成简单的不等式如下：$x_j - x_i \le b_k$（1<=I,j<=n,1<=k<=m）

简单举例：

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \le \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix}$$

找出未知量x_1, x_2, x_3, x_4, x_5,并且满足以下差分约束条件：

$$\begin{aligned} x_1 - x_2 &<= 0 \\ x_1 - x_5 &<= -1 \\ x_2 - x_5 &<= 1 \\ x_3 - x_1 &<= 5 \\ x_4 - x_1 &<= 4 \\ x_4 - x_3 &<= -1 \\ x_5 - x_3 &<= -3 \\ x_5 - x_4 &<= -3 \end{aligned}$$

需要注意的是，用 Bellmanford 求出的具体答案只是众多答案中的一种，答案并不唯一，但答案之间却也有着联系。这里我们并不对其进行专门的探讨。

了解了差分约束系统的整个过程，下面再看另外一道题目，让我们对差分约束系统的应用能够有更加深刻的认识。

[例题三] zju1420 Cashier Employment 出纳员问题

题目中文翻译：

Tehran 的一家每天 24 小时营业的超市，需要一批出纳员来满足它的需要。超市经理雇佣你来帮他解决他的问题—超市在每天的不同时段需要不同数目的出纳员（例如：午夜时只需一小批，而下午则需要很多）来为顾客提供优质服务。他希望雇佣最少数目的出纳员。

经理已经提供你一天的每一小时需要出纳员的最少数量—R(0), R(1), ..., R(23)。R（0）表示从午夜到上午 1：00 需要出纳员的最少数目，R（1）表示上午 1：00 到 2：00 之间需要的，等等。每一天，这些数据都是相同的。有 N 人申请这项工作，每个申请者 I 在每 24 小时中，从一个特定的时刻开始连续工作恰好 8 小时，定义 tI （0 <= tI <= 23）为上面提到的开始时刻。也就是说，如果第 I 个申请者被录取，他（她）将从 tI 时刻开始连续工作 8 小时。

你将编写一个程序，输入 R（I）（I = 0..23）和 tI （I = 1..N），它们都是非负整数，计算为满足上述限制需要雇佣的最少出纳员数目。在每一时刻可以有比对应的 R（I）更多的出纳员在工作。

输入

输入文件的第一行为测试点个数（<= 20）。每组测试数据的第一行为 24 个整数表示 R（0），R（1），...， R（23）（R（I）<= 1000）。接下来一行是 N，表示申请者数目（0 <= N <= 1000），接下来每行包含一个整数 tI （0 <= tI <= 23）。两组测试数据之间没有空行。

输出

对于每个测试点，输出只有一行，包含一个整数，表示需要出纳员的最少数目。如果无解，你应当输出“No Solution”。

样例输入

```
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10
```

样例输出

```
1
```

我们按刚才所讲到的方法对此题进行处理。
这题很容易想到如下的不等式模型：

设 num[i]为 i 时刻能够开始工作的人数，x[i]为实际雇佣的人数，那么 x[I]<=num[I]。
设 r[i]为 i 时刻至少需要工作的人数，于是有如下关系：

x[I-7]+x[I-6]+x[I-5]+x[I-4]+x[I-3]+x[I-2]+x[I-1]+x[I]>=r[I]

设 s[I]=x[1]+x[2]…+x[I]，得到

0<=s[I]-s[I-1]<=num[I]， 0<=I<=23

s[I]-s[I-8]>=r[I]， 8<=I<=23

s[23]+s[I]-s[I+16]>=r[I]， 0<=I<=7

对于以上的几组不等式，我们采用一种非常笨拙的办法处理这一系列的不等式(其实也是让零乱的式子变得更加整齐，易于处理)。首先我们要明白差分约束系统的应用对象（它通常针对多个二项相减的不等式的）于是我们将上面的所有式子都转化成两项未知项在左边，另外的常数项在右边，且中间用>=连接的式子，即:

s[I]-s[I-1]>=0 (0<=I<=23)

s[I-1]-s[I]>=-num[I] (0<=I<=23)

s[I]-s[I-8]>=r[I] (8<=I<=23)

s[I]-s[I+16]>=r[I]-s[23] (0<=I<= 7)

这里出现了小的困难，我们发现以上式子并不是标准的差分约束系统，因为在最后一个式子中出现了三个未知单位。但是注意到其中跟随 I 变化的只有两个，于是 s[23]就变得特殊起来，看来是需要我们单独处理，于是我们把 s[23]当作已知量放在右边。

经过这样的整理，整个图就很容易创建了，将所有形如 A-B>=C 的式子 我们从节点 B 引出一条有向边指向 A 边的权值为 C （这里注意由于左右确定，式子又是统一的>=的不等式，所以 A 和 B 是相对确定的，边是一定是指向 A 的） ，图就建成了 。

最后枚举所有 s[23]的可能值，对于每一个 s[23]，我们都进行一次常规差分约束系统问题的求解，判断这种情况是否可行，如果可行求出需要的最优值，记录到 Ans 中，最后的 Ans 的值即为所求。

程序见附录。

结语：

对于许多复杂的问题，我们通常选择将不够清晰、难以处理的模型转化为容易理解、易于处理的模型。就像用已知的知识作为工具去探索未知领域一样，联想、发散、转化将成为相当有用的武器。本文选择了差分约束系统这样一个平台，通过介绍差分约束系统的相关知识和其在信息学问题中的应用以小见大，为读者提供一个解题的思路和技巧。

附录：

参考文献：《金牌之路竞赛解题指导—高中计算机》 [王建德，周咏基]

《Introduction to Algorithms》 [H.Cormen]

例题信息

http://acm.zju.edu.cn/

http://acm.zju.edu.cn/show_problem.php?pid=2008

http://acm.zju.edu.cn/show_problem.php?pid=1508

http://acm.zju.edu.cn/show_problem.php?pid=1420

例一：ZJU2008

Invitation Cards

Time limit: 5 Seconds Memory limit: 32768K
Total Submit: 70 Accepted Submit: 20

In the age of television, not many people attend theater performances. Antique Comedians of Malidinesia are aware of this fact. They want to propagate theater and, most of all, Antique Comedies. They have printed invitation cards with all the necessary information and with the programme. A lot of students were hired to distribute these invitations among the people. Each student volunteer has assigned exactly one bus stop and he or she stays there the whole day and gives invitation to people travelling by bus. A special course was taken where students learned how to influence people and what is the difference between influencing and robbery.

The transport system is very special: all lines are unidirectional and connect exactly two stops. Buses leave the originating stop with passangers each half an hour. After reaching the destination stop they return empty to the originating stop, where they wait until the next full half an hour, e.g. X:00 or X:30, where 'X' denotes the hour. The fee for transport between two stops is given by special tables and is payable on the spot. The lines are planned in such a way, that each round trip (i.e. a journey starting and finishing at the same stop) passes through a Central Checkpoint Stop (CCS) where each passenger has to pass a thorough check including body scan.

All the ACM student members leave the CCS each morning. Each volunteer is to move to one predetermined stop to invite passengers. There are as many volunteers as stops. At the end of the day, all students travel back to CCS. You are to write a computer program that helps ACM to minimize the amount of money to pay every day for the transport of their employees.

Input

The input consists of N cases. The first line of the input contains only positive integer N. Then follow the cases. Each case begins with a line containing exactly two integers P and Q, 1 <= P,Q <= 1000000. P is the number of stops including CCS and Q the number of bus lines. Then there are Q lines, each describing one bus line. Each of the lines contains exactly three numbers - the originating stop, the destination stop and the price. The CCS is designated by number 1. Prices are positive integers the sum of which is smaller than 1000000000. You can also assume it is always possible to get from any stop to any other stop.

Output

For each case, print one line containing the minimum amount of money to be paid each day by ACM for the travel costs of its volunteers.

Sample Input

```
2
2 2
1 2 13
2 1 33
4 6
1 2 10
2 1 60
1 3 20
3 4 10
2 4 5
4 1 50
```

Sample Output

```
46
210
```

Problem Source: *Central Europe 1998*

[参考程序]：

```
const
  maxn=1000000;
  maxm=1000000;

type
  Tedge     =  record
                 x,y,long:longint;
               end;
  waytype   =  array[1..maxn]of longint;
  Tdijkstra = object
                 h,p:array[1..maxn]of longint;
                 tot:longint;
                 procedure work(var f:waytype);
               end;
```

```
  TBellman =   object
                  state:array[1..maxn*2]of longint;
                  procedure bellman_ford(var f:waytype);
                end;

var
  index        :   array[0..maxn]of longint;
  edge         :   array[1..maxm]of Tedge;
  BellMan      :   Tbellman;
 // dijkstra    :   Tdijkstra;
  way1,way2 :   waytype;
  n,m,i,ans,
  long,x,y,
  o,casen      :   longint;

procedure qsort(fp,rp:longint);
  var i,j:longint;
      x,t:Tedge;
  begin
    i:=fp;j:=rp;x:=edge[(i+j)div 2];
    repeat
      while (edge[j].x>x.x) do dec(j);
      while (edge[i].x<x.x) do inc(i);
      if i<=j then begin
        t:=edge[i];edge[i]:=edge[j];edge[j]:=t;
        inc(i);dec(j);
      end;
    until i>j;
    if i<rp then qsort(i,rp);
    if fp<j then qsort(fp,j);
  end;

procedure makeindex;
  var i,j:longint;
  begin
    index[0]:=0;j:=1;
    for i:=1 to n do begin
      while ((j<=m)and(edge[j].x=i))do inc(j);
      index[i]:=j-1;
    end;
  end;

procedure change;
  var
```

```
        i,t : longint;
    begin
      for i:=1 to m do begin
        t:=edge[i].x;edge[i].x:=edge[i].y;edge[i].y:=t;
      end;
      qsort(1,m);
      makeindex;
    end;


procedure Tbellman.bellman_ford(var f:waytype);
    var fp,rp,temp : longint;
    begin
      fp:=1;rp:=1;
      fillchar(f,sizeof(f),$FF);
      qsort(1,m);
      state[fp]:=1;f[1]:=0;
      while fp<=rp do begin
        for i:=index[state[fp]-1]+1 to index[state[fp]] do begin
          temp:=f[state[fp]]+edge[i].long;
          if (temp<f[edge[i].y]) or (f[edge[i].y]<0) then begin
            inc(rp);
            state[rp]:=edge[i].y;
            f[edge[i].y]:=temp;
          end;
        end;
        inc(fp);
      end;
    end;

procedure Tdijkstra.work(var f:waytype);
    var u,v,i,temp:longint;

    procedure swap(var x:longint;y:longint);
    var t:longint;
    begin
      p[h[x]]:=y;p[h[y]]:=x;
      t:=h[x];h[x]:=h[y];h[y]:=t;
      x:=y;
    end;

    procedure up(x:longint);
    begin
      while x>1 do
```

```
        if f[h[x]]<f[h[x shr 1]] then swap(x,x shr 1) else exit;
    end;

    procedure down(x:longint);
    var l,r:longint;
    begin
      while true do begin
        l:=x*2;r:=l+1;
        if l>tot then exit;
        if (l=tot)then if (f[h[l]]<f[h[x]]) then swap(x,l) else exit;
        if (f[h[x]]>f[h[l]])and(f[h[l]]<f[h[r]]) then swap(x,l) else
        if (f[h[x]]>f[h[r]])then swap(x,r) else exit;
      end;
    end;

    begin
      fillchar(f,sizeof(f),$FF);
      fillchar(p,sizeof(p),0);
      f[1]:=0;tot:=1;h[1]:=1;
      while true do begin
        if tot=0 then break;
        v:=h[1];p[h[1]]:=0;h[1]:=h[tot];p[h[tot]]:=1;
        h[tot]:=0;dec(tot);
        if tot>1 then down(1);

        for i:=index[v-1]+1 to index[v] do begin
          temp:=f[v]+edge[i].long;u:=edge[i].y;
          if (f[u]>temp)or(f[u]<0) then begin
            f[u]:=temp;
            if p[u]=0 then begin
              inc(tot);
              h[tot]:=u;
              p[u]:=tot;
              up(tot);
            end else up(p[u]);
          end;
        end;
      end;
    end;

begin
    readln(casen);
    for o:=1 to casen do begin
      readln(n,m);
```

```
      fillchar(edge,sizeof(edge),0);
      for i:=1 to m do
        with edge[i] do
          readln(x,y,long);

      qsort(1,m);makeindex;ans:=0;
      bellman.bellman_ford(way1);
//      dijkstra.work(way1);
      for i:=1 to n do inc(ans,way1[i]);
      change;
      bellman.bellman_ford(way2);
//      dijkstra.work(way2);
      for i:=1 to n do inc(ans,way2[i]);
      writeln(ans);
   end;
end.
```

例二：ZJU1508

Intervals

Time limit: 10 Seconds Memory limit: 32768K
Total Submit: 150 Accepted Submit: 49

You are given n closed, integer intervals [ai, bi] and n integers c1, ..., cn.

Write a program that:

> reads the number of intervals, their endpoints and integers c1, ..., cn from the standard input,

> computes the minimal size of a set Z of integers which has at least ci common elements with interval [ai, bi], for each i = 1, 2, ..., n,

> writes the answer to the standard output.

Input

The first line of the input contains an integer n (1 <= n <= 50 000) - the number of intervals. The following n lines describe the intervals. The i+1-th line of the input contains three integers ai, bi and ci

separated by single spaces and such that 0 <= ai <= bi <= 50 000 and 1 <= ci <= bi - ai + 1.

Process to the end of file.

Output

The output contains exactly one integer equal to the minimal size of set Z sharing at least ci elements with interval [ai, bi], for each i = 1, 2, ..., n.

Sample Input

```
5
3 7 3
8 10 3
6 8 1
1 3 1
10 11 1
```

Sample Output

```
6
```

Problem Source: *Southwestern Europe 2002*

[参考程序]：

```
{$MODE FPC}
{$M 0,1025}
{$R-,Q-,S-,I-}

Program T_1508;
Type
  TEdge = Record
            s, t, w : LongInt;
          End;

Var
```

```
  edge : Array [ 1..50000 ] Of TEdge;
  d : Array [ 0..50000 ] Of LongInt;
  n, i, a, b, c, max : LongInt;

Procedure Bellman_Ford;
Var
  i : LongInt;
  flag : Boolean;

Begin
  For i := 0 To max Do
     d[ i ] := 0;

  Repeat
     Flag := True;

     For i := 1 To n Do
     With edge[ i ] Do
        If (d[ s ] + w < d[ t ]) Then
        Begin
           d[ t ] := d[ s ] + w;
           Flag := False;
        End;

     For i := 1 To max Do
        If (d[ i-1 ] + 1 < d[ i ]) Then
        Begin
           d[ i ] := d[ i-1 ] + 1;
           Flag := False;
        End;

     For i := max DownTo 1 Do
        If (d[ i ] < d[ i-1 ]) Then
        Begin
           d[ i-1 ] := d[ i ];
           Flag := False;
        End;
  Until Flag;
End;

Begin
  While NOT EOF Do
  Begin
     Readln(n);
```

```
    max := 0;
    For i := 1 To n Do
    Begin
      Readln(a, b, c);
      If (b > max) Then max := b;
      edge[ i ].s := b; edge[ i ].t := a - 1; edge[ i ].w := -c;
    End;
    Bellman_Ford;

    Writeln(d[ max ] - d[ 0 ]);
 End;
End.
```

例三：ZJU1420

Cashier Employment

Time limit: 10 Seconds Memory limit: 32768K
Total Submit: 34 Accepted Submit: 10

A supermarket in Tehran is open 24 hours a day every day and needs a number of cashiers to fit its need. The supermarket manager has hired you to help him, solve his problem. The problem is that the supermarket needs different number of cashiers at different times of each day (for example, a few cashiers after midnight, and many in the afternoon) to provide good service to its customers, and he wants to hire the least number of cashiers for this job.

The manager has provided you with the least number of cashiers needed for every one-hour slot of the day. This data is given as R(0), R(1), ..., R(23): R(0) represents the least number of cashiers needed from midnight to 1:00 A.M., R(1) shows this number for duration of 1:00 A.M. to 2:00 A.M., and so on. Note that these numbers are the same every day. There are N qualified applicants for this job. Each applicant i works non-stop once each 24 hours in a shift of exactly 8 hours starting from a specified hour, say ti (0 <= ti <= 23), exactly from the start of the hour mentioned. That is, if the ith applicant is hired, he/she will work starting from ti o'clock sharp for 8 hours. Cashiers do not replace one another and work exactly as scheduled, and there are enough cash registers and counters for those who are hired.

You are to write a program to read the R(i)'s for i=0...23 and ti's for i=1...N that are all, non-negative integer numbers and compute the least number of cashiers needed to be employed to meet the mentioned constraints.

Note that there can be more cashiers than the least number needed for a specific slot.

Input

The first line of input is the number of test cases for this problem (at most 20). Each test case starts with 24 integer numbers representing the R(0), R(1), ..., R(23) in one line (R(i) can be at most 1000). Then there is N, number of applicants in another line (0 <= N <= 1000), after which come N lines each containing one ti (0 <= ti <= 23). There are no blank lines between test cases.

Output

For each test case, the output should be written in one line, which is the least number of cashiers needed.

If there is no solution for the test case, you should write No Solution for that case.

Sample Input

```
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10
```

Sample Output

```
1
```

Problem Source: *Asia 2000, Tehran (Iran)*

[参考程序]:

```
const
  maxn=230;

var
  g:array[-1..maxn,1..4]of record
      n,v:integer;
    end;
  d,xu,num,a:array[-1..maxn]of integer;
  ans:integer;
  x,n,casen,o,i:integer;
  flag:boolean;

procedure add(a,b,c:integer);
  begin
    inc(num[a]);
    g[a,num[a]].n:=b;
    g[a,num[a]].v:=c;
  end;

procedure init;
  var i:integer;
  begin
    fillchar(g,sizeof(g),0);
    for i:=0 to 23 do begin
      if i<=7   then add(i+16,i,xu[i]-ans)
                else add(i-8,i,xu[i]);
      add(i-1,i,0);
      add(i,i-1,-a[i]);
    end;
  end;

function bellman_ford:boolean;
  var i,j,k:integer;
      ff:boolean;
  begin
    bellman_ford:=false;
    fillchar(d,sizeof(d),0);
    for i:=1 to 23 do begin
      ff:=true;
      for j:=-1 to 23 do
        for k:=1 to num[j] do
          if d[j]+g[j,k].v>d[g[j,k].n] then begin
            d[g[j,k].n]:=d[j]+g[j,k].v;ff:=false;
```

```
                end;
          if ff then break;
       end;
       for j:=0 to 23 do if d[j]-d[j-1]>a[j] then exit;
       for j:=-1 to 23 do
          for k:=1 to num[j] do
             if d[j]+g[j,k].v>d[g[j,k].n] then exit;
   {    for i:=0 to 23 do begin
          if (i<=7)and (d[i]-d[i+16]<xu[i]-d[23]) or
              (i>=8)and (d[i]-d[i-8]<xu[i]) then exit
       end;}
       bellman_ford:=n<>4//d[23]=ans;
   end;

begin
//    assign(input,'e:\input.txt');reset(input);assign(output,'e:\output.txt');rewrite(output);
   readln(casen);
   for o:=1 to casen do begin
      for i:=0 to 23 do read(xu[i]);readln;readln(n);
      for i:=1 to n do begin read(x);inc(a[x]);end;
      flag:=false;
      for ans:=0 to n do begin
         fillchar(d,sizeof(d),0);
         fillchar(num,sizeof(num),0);
         init;
         if bellman_ford then begin
            flag:=true;
            break;
         end;
      end;
      if flag then writeln(ans) else writeln('No Solution');
   end;
//    close(output);close(input);
end.
```