

华中科技大学

2019

算法实践

课程设计报告

专 业：	计算机科学与技术
班 级：	ACM1901班
学 号：	U201915035
姓 名：	邹雅
电 话：	15058667378
邮 件：	1542527211@qq.com
完成日期：	2022. 1. 6



华中科技大学课程设计报告

目录

一、课程设计任务的完成情况	1
1.1 POJ题完成情况	1
1.2 学习过程及成效	2
二、小组工作情况的汇报	3
2.1 负责任务	3
2.2 工作情况	4
三、专题报告——差分约束系统	4
3.1 一般性描述	4
3.2 解题报告	9
四、总结	14
4.1 已完成工作	14
4.2 需要进一步完善的地方	14
4.3 心得体会	14

华中科技大学课程设计报告

一、课程设计任务的完成情况

1.1 POJ题完成情况

POJ 题一共完成 10 道，在平台上跑出来的是 8 题；另两题由于无法找到平台测试故而未做测试，可以通过本地测试。

POJ1182

 练习	1182: 食物链	Accepted	640kB	48ms	1936 B	G++	前天
--	-----------	----------	-------	------	--------	-----	----

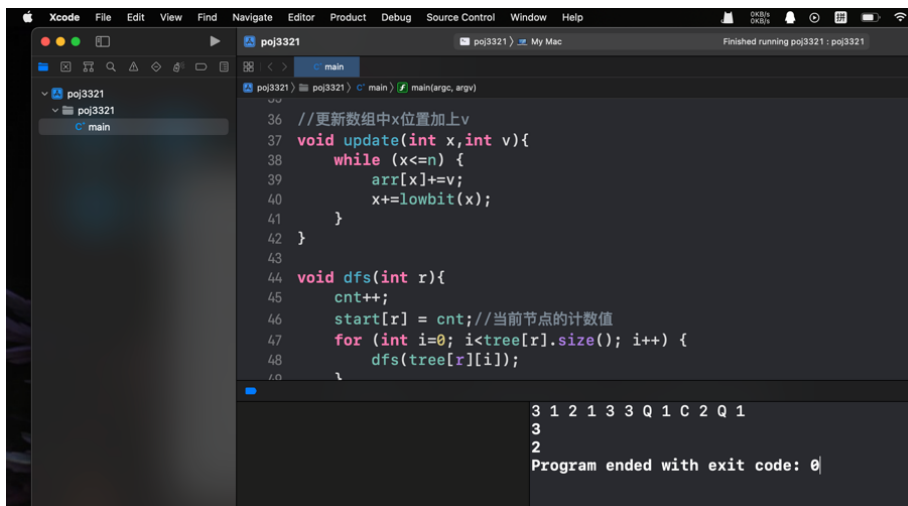
POJ1417

 练习	1417: True Liars	Accepted	760kB	1ms	3973 B	G++	前天
--	------------------	----------	-------	-----	--------	-----	----

POJ1990

 练习	1990: MooFest	Accepted	828kB	11ms	1286 B	G++	昨天
--	---------------	----------	-------	------	--------	-----	----


POJ3321




```
36 //更新数组中x位置加上v
37 void update(int x,int v){
38     while (x<=n) {
39         arr[x]+=v;
40         x+=lowbit(x);
41     }
42 }
43
44 void dfs(int r){
45     cnt++;
46     start[r] = cnt;//当前节点的计数值
47     for (int i=0; i<tree[r].size(); i++) {
48         dfs(tree[r][i]);
49     }
50 }
```

3 1 2 1 3 3 Q 1 C 2 Q 1
3
2
Program ended with exit code: 0

POJ1470

 练习	1470: Closest Common Ancestors	Accepted	3580kB	159ms	2333 B	G++	刚刚
--	--------------------------------	----------	--------	-------	--------	-----	----

POJ1986

比赛	题目	结果	内存	时间	代码长度	语言	提交时间
 练习	1986: Distance Queries	Accepted	7892kB	148ms	2559 B	G++	刚刚

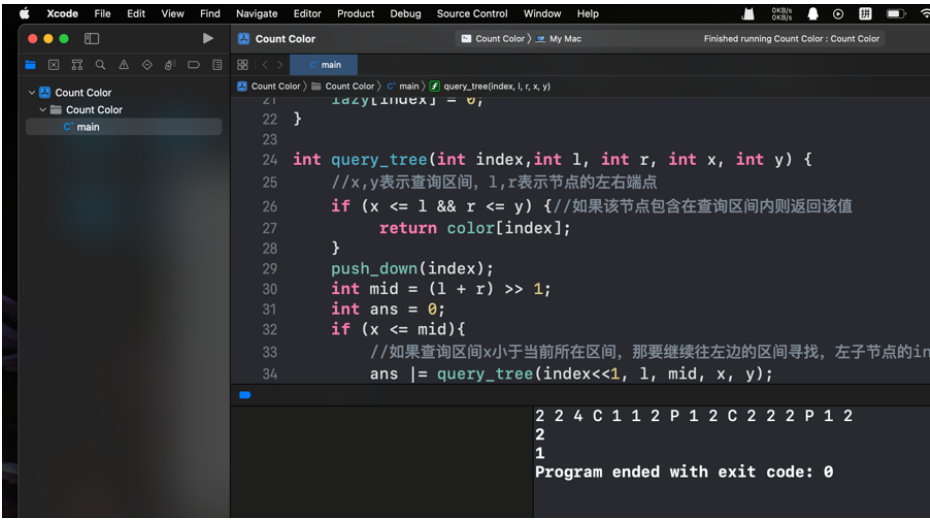
Candies

比赛	题目	结果	内存	时间	代码长度	语言	提交时间
 练习	3424: Candies	Accepted	2728kB	554ms	2158 B	G++	12小时前

Cashier Employment

 练习	1275: Cashier Employment	Accepted	256kB	12ms	3876 B	G++	20小时前
--	--------------------------	----------	-------	------	--------	-----	-------

Count Color



City Horizon

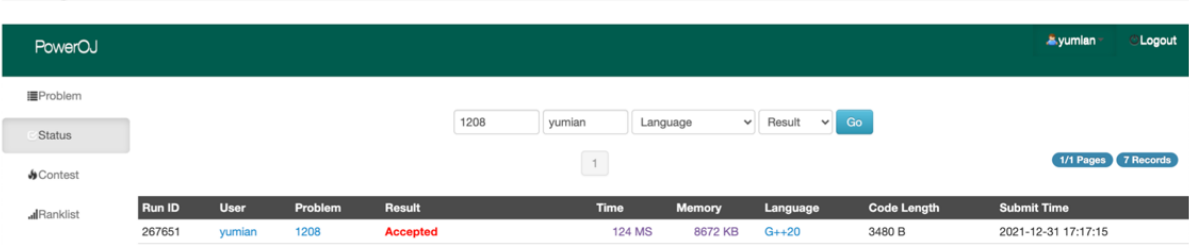


图 1-1 POJ 题解测试截图

1.2 学习过程及成效

在课程期间，第一个星期我学习了并查集和树状数组；第二个星期我学习了 LCA/RMQ 专题和差分约束系统，并和小组同学一起准备了并查集专题的答辩；课程结束后由于离代码截止还有一周，我学习了线段树。

学习并查集期间，True Liars 这道题卡了我好一段时间。一开始在考虑如何归约成并查集问题上觉得迷惑，在思考清楚用边带权并查集建立相互有关系的人这一点后，建立并查集完成了建模。但是接下去又被解决是否存在唯一的好人组合问题难住了，通过搜寻资料学习和思考，发现可以归约成 01 背包问题。在这道题里我深入地理解了边带权并查集

以及更好地应用动态规划，发现了 01 背包问题的变体问题有更广阔的应用以便于日后举一反三。

树状数组相对来说比较简单，这个数据结构设计得很巧妙，打开了我的思路。在 MooFest 这道题中，核心思路是通过对牛的数量以及牛的 x 位置坐标量建立两个树状数组。一开始只想到了用 x 坐标建立树状数组，然后对牛一一进行处理，这种处理次数需要达到 $O(n^2)$ ，而建立了牛的数量树状数组后，处理就简单了很多，只需一遍处理即可。

LCA 最近公共祖先算法有很多解法，受限于时间，我学习了用 Tarjan 算法来求解，之后可以继续深入学习。例如，LCA 可以转换成 RMQ 区间查询问题来进行求解。差分约束系统我会在下面细讲，在此不多做赘述。

线段树也是一个很巧妙的数据结构，在树状数组的基础之上又加深了一个层次。Count Color 这道题将颜色保存成二进制的形式是很巧妙的，一开始我的思路是用数组标记，然后去遍历，虽然说也可以实现，但是会加大处理复杂度，在线段树的层面上，这道题用标准的 query 和 update 函数进行改写即可实现。天际线的题，之前做过一道类似的用扫描线的解法来做，而这次用线段树来实现。想清楚线段树保存的内容以及更新法之后，这道题中的离散化思想也是让我有学习到的地方，通过把离散的大空间在小空间内映射，可以加快处理速度以及占据的内存。

在每个专题中我都学到了一个新的知识点，这些算法知识对我来说是全新的也是很有用的，而通过做题，又学会了一些很巧妙的算法思路。同时在这次算法实践的工作过程中，我也加深了自己的搜索资料学习的能力，以及提取关键信息的能力。

二、小组工作情况的汇报

2.1 负责任务

在本次算法实践课程设计实验的小组工作过程中，我主要负责的工作是完成答辩所需的 ppt 的制作。具体来说，第一部分是完成理论部分的 ppt 制作，提取关键答辩的关键信息进行展示；第二部分是根据各小组成员所提供的相关文字初稿和 ppt 初稿等资料，将其进行系列整合并配以图片、动画等展示成果，最终设计完成最后答辩所使用的 ppt。

姓名	分工
邹雅	理论部分的 ppt 以及整合 ppt
刘旭东	理论引入、解题思路和汇报
李恒庄	POJ1417 文字和初步的 ppt
张家荣	POJ1182 文字和初步的 ppt、部分理论

表 2-1 小组分工情况

2.2 工作情况

在本次课程设计实验中，我的主要任务是 ppt 的设计制作。在 ppt 的制作过程中，我主要遇到了内容选取上的问题，这些问题主要源于我对相关算法停留在了相对比较浅的认识上，没有很好地把控住相关算法的核心思想，以及没有很好地将并查集的知识点融会贯通。之后在结合学习资料以及小组成员提供的文字整理资料，以及与各小组成员间的讨论，我们确定了 ppt 制作的具体内容和所要展示的具体材料。最后顺利在课程答辩前完成了 ppt 的制作，虽然我肯定还有着相当大的进步空间，不过这已经是现阶段我所能做到的最好了，最终也是顺利通过了答辩。

在制作 ppt 的过程中，通过反复查看各种资料以及小组成员的文字稿，思考如何把最核心的内容展示在 ppt 上来抓住主干要点，我能更加深入地理解了并查集的内容，通过了解了并查集的演变从而明白了并查集能解决的问题，对边带权并查集也有了更多的认识，其中张家荣同学总结整理了对关系运算、解决并查集问题过程中的三大问题等等理论内容，也让我学到了很多。总体而言，尽管之前对并查集已经进行了学习并且写了题，但是和组员沟通、整理 ppt 这些过程，我的收获很多。

当然，在这次课程设计中我发现了我自身存在的一些问题，如和小组成员之间的相互沟通还不够积极、任务总是很容易拖延到临近截止日期等。同时我也深刻认识到，不管是什么任务，只要涉及到小组合作，同学之间充分的交流是非常重要的。我希望能够在将来的工作中能做得更好。

三、专题报告——差分约束系统

3.1 一般性描述

3.1.1 基本概念

差分约束系统 (system of difference constraints)，是求解关于一组变量的特殊不等式组的方法。例如下所示为一个不等式组：

$$\begin{cases} x_1 - x_2 \leq 0 \\ x_1 - x_3 \leq 6 \\ x_3 - x_2 \leq 2 \end{cases}$$

特点是全都是两个未知数的差小于等于某个常数（大于等于也可以，因为左右乘以-1 或者直接左右移项就可以化成小于等于的形式）。这个不等式组要么无解，要么就有无限组解。因为任何两个数加上一个数以后，它们之间的关系（差）是不变的，这个差分约束系统中的所有不等式都不会被破坏。

不等式组的每一个不等式称之为一个约束条件。如果一个系统由 n 个变量和 m 个约束条件组成，其中每个约束条件形如 $x_j - x_i \leq b_k$ (这里的 $i, j \in [1, n], k \in [1, m]$)，则称其为差分约束系统。通俗一点地说，差分约束系统就是一些不等式的组，而我们的目标是通过给定的约束不等式组求出某些变量的最大值或某些变量的最小值或者该差分约束系统是否有解等等问题。

3.1.2 相关原理

<最长路&最短路>

最长路的定义与最短路是对偶的，故这里仅介绍最短路。

对加权图 $G(V, E)$: $w(u, v)$ 及源点 $s \in V$ 。若对于某一点 $v \in V$ ，存在一条从 s 到 v 的路径 p^* 使得 $w(p^*) \leq w(p)$ ，其中 p 为从 s 到 v 的任意路径，则称 p^* 为从 s 到 v 的最短路， $w(p^*)$ 为最短路长度，记做 $D[v]$ 。如果图中不存在负权环，则对于任意 $v \in V$ ， $D[v]$ 都是一个确定的有界值。

求解差分约束系统可以转换为求解图 G 中每个节点的最短路。

<三角不等式>

差分约束大部分时候就是找到问题的约束条件并且转换为对应的不等式。我们来看约束条件的通式 $x_j - x_i \leq b_k$ ，移项得 $x_j \leq b_k + x_i$ 。观察可得和最短路转移中的三角形不等式 $d_y \leq d_x + edge_i$ 很像，也就是说求最大值的过程类似于最短路算法中的松弛操作。所以说在得到不等式组之后，我们可以转化成最短路径问题来求解。

为了求解最短路径，需要使用到三角不等式，即：

$$B - A \leq c \quad (1)$$

$$C - B \leq a \quad (2)$$

$$C - A \leq b \quad (3)$$

若要求 $C-A$ 的最大值，可以知道 $\max(C-A) = \min(b, a + c)$ ，正好对应图 3-1 中 C 到 A 的最短路：

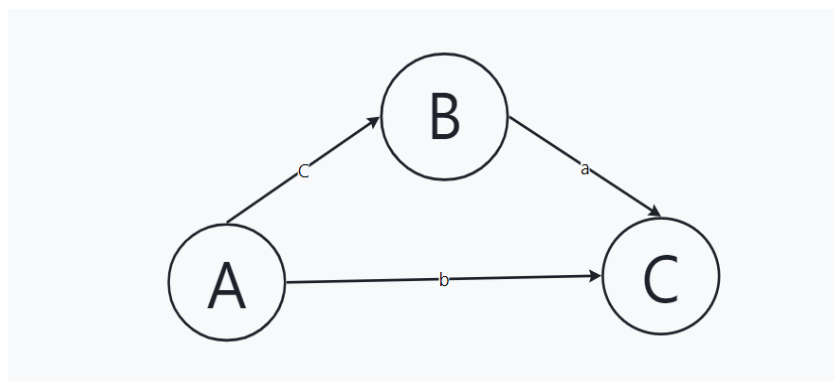


图 3-1 AC间最短路径例图

我们经常使用三角不等式，在最短路径问题求解过程中进行松弛操作。

〈松弛操作〉

松弛操作是指通过测试源点 s 到某一个节点 v 之间的最短路径，利用 s 到节点 u 之间的最短路径距离加上节点 u 与 v 之间的边权重，并与当前的 s 到 v 的最短路径估计进行比较，如果前者更小，那么就要对 v 的最短路径，以及 v 的前驱节点进行更新。松弛操作的时间复杂度为 $O(1)$ ，伪代码如下所示：

```

RELAX( $u, v, w$ ):
  if  $v.d > u.d + w(u, v)$ :
     $v.d = u.d + w(u, v)$ 
     $v.pre = u$ 
  
```

3.1.3 常用算法

1 Dijkstra算法

Dijkstra 算法是一个基于贪心算法、广度优先、动态规划的基础之上的算法，目的是求一个图中一个点与其他全部点之间最短路径的算法。

算法核心思想为维护一个节点集合 S ，从起始节点到在这个集合中的每一个点对最短路径都已经被找到，算法重复从除了这个节点集合以外的其余节点中取出一个最短路径估计最小的节点 u ，把 u 加入到集合 S 中，然后再把所有从 u 出发的边进行松弛，一直求解直到遍历完所有节点。

如下图例子中，以 D 点为起始节点，此时我们已经把 C 、 D 、 E 、 F 节点加入到节点集合 S 中，接下去我们要找一个到这些节点最短的路径的节点继续操作。我们观察节点到 D 的距离和前驱节点如下表所示。那么接下去我们要把 G 加入到节点集合 S 中，并根据 G 节点的最短路径距离和前驱节点进行更新路径信息。

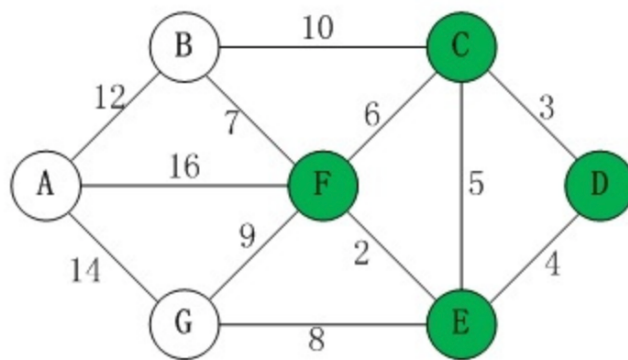


图 3-2 最短路径图例子

节点	距离	前驱节点
D	0	/
C	3	D
E	4	D
F	6	E
B	13	C
A	22	F
G	12	E

表 3-1 节点距离以及前驱节点

看完这个核心思想，我们就能理解为什么说 Dijkstra 算法是贪心算法，因为每一次加入节点时，我们都选取了最短路径的节点来进行松弛操作。

该算法要求所有边的权重均为非负值，即对于所有的边 $(u, v) \in E$ ， $\omega(u, v) \geq 0$ ，即不能有负权重的边，更不能有负权重的环。伪代码如图 3-3：

```

DIJKSTRA( $G, w, s$ )
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S = \emptyset$ 
3   $Q = G.V$ 
4  while  $Q \neq \emptyset$ 
5       $u = \text{EXTRACT-MIN}(Q)$ 
6       $S = S \cup \{u\}$ 
7      for each vertex  $v \in G.Adj[u]$ 
8          RELAX( $u, v, w$ )
    
```

图 3-3 Dijkstra 算法伪代码

2 SPFA算法

SPFA 算法解决的是一般情况下的单源最短路径问题，是对 Bellman-ford 算法的队列优化。相对于 Dijkstra 算法来说 SPFA 算法可以在权值为负数的时候使用，并且可以检测到负权环。假如说所跑的图上存在这样一个环路，那么算法将告诉我们不存在解决方案；如果不存在负权环，那么算法将给出最短路径和它们的权重。

实现思路本质上也是通过松弛操作来降低源节点到各个节点之间的最短路径值，直至整个图的最短路径不再变化为止。

Bellman-Ford 算法规定，外循环进行总节点个数的次数，内循环对每条边进行松弛操作，进行总边数的次数，内循环中每一次循环就是对该边进行松弛操作。

下面是该算法简单的实现思路：

用 dis 数组记录源点到有向图上任意一点距离，其中源点到自身距离为 0，到其他点距离标记为无穷（一个存在的最大整数）。将源点入队，并重复以下步骤：

1. 队列中的队首元素 x 出队
2. 遍历所有以队首为起点的有向边 (x, i) ，若 $dis[x] + w(x, i) < dis[i]$ ，则更新 $dis[i]$
3. 如果点 i 不在队列中，则 i 入队
4. 若队列为空，跳出循环；否则继续跳转到 1 进行执行

如果需要对负环是否存在的判定，只需要把上述的双重循环再进行一遍即可。如果在这次循环还会改变更新最短路径，那么说明存在负环，直接退出表示当前算法不存在最短路径解。

3.1.4 差分约束系统的基本解法过程

1. 根据条件把题意通过各种变量的相关约束表达出来得到不等式组，注意要发掘出隐含的不等式，比如说前后两个变量之间隐含的不等式关系，以及一些隐藏的内部逻辑。
2. 进行不等式组的标准化的，根据得到的不等式组进行建图：
 - 如果要求取最小值，那么求出最长路，那么将不等式全部化成 $x_i - x_j \geq k$ 的形式，这样建立从 j 到 i 并且权值为 k 的边，如果不等式组中有 $x_i - x_j > k$ ，因为一般题目都是对整形变量的约束，化为 $x_i - x_j \geq k + 1$ 即可，如果 $x_i - x_j = k$ 呢，那么可以变为如下两个： $x_i - x_j \geq k$ ， $x_i - x_j \leq k$ ，进一步变为 $x_j - x_i \geq -k$ 的形式，再建立两条边即可。
 - 如果求取的是最大值，那么求取最短路，将不等式全部化成 $x_i - x_j \leq k$ 的形式，这样建立从 j 到 i 并且权值为 k 的边，如果像上面的两种情况，那么同样地标准化就行了。
 - 如果要判断差分约束系统是否存在解，一般都是判断环，选择求最短路或者最长路求

解都行，只是不等式标准化时候不同。判环的话，用SPFA算法即可得到，我们在后续会介绍。

3. 建好图之后一般直接SPFA算法或Bellman-Ford算法求解，因为一般存在负边，针对一些特定的题目，如果我们确保了边权值不可能为负时，我们则可以采用Dijkstra算法来进行最短路径的寻找。

3.2 解题报告

3.2.1 Candies

问题描述

在幼儿园的日子里，飞鼠是他的班班长。校长偶尔会给飞鼠班的孩子们带来一大袋糖果，并让飞鼠分发。所有的孩子都非常喜欢糖果，而且他们经常比较糖果的数量和别人。孩子A可能有想法，如果另一个孩子B在某些方面比他好，那么B有理由比他应得的更多的糖果，不然的话不论他有多少糖果他都不应该比B得到一定数量的糖果更少，这样他就会感到不满，然后到校长那里去抱怨“飞鼠”分配不公。

当时史努比和飞鼠一起上课。飞鼠总是拿他的糖果和史努比的比较。他让这些数字之间的差别尽可能大，同时又能让每个孩子都满意。现在他又从班主任那里得到了一袋糖果，他能从中得到怎么样最大的不同呢？

输入输出

输入是单个的测试用例。测试用例从一行两个整数N和M开始，N和M分别不超过30000和15000。N是孩子课堂上和孩子们的数量为从1到N。史努比和flymouse的编号总是1, N。然后M行各拿三个整数，按顺序为A、B和c，这意味着孩子A认为孩子B的糖果不应该比他多c个。

输出一行为所需的最大差值。差值是有限的。

解题分析

差分约束系统中，最重要的就是找到约束，根据约束建立图，然后套用图的最短路算法即可。分析题目可以看出，题目中给出了M行的条件，就是孩子A认为孩子B的糖果不应该比他多c个。所以在本题中，约束很明显，就是输入的B的糖果减去A的糖果小于等于c个，转化成伪代码进行描述就是 $candies[v] - candies[u] \leq w$ ，把这M行的约束输入并保存即可。同时由于糖果数不会小于0，所以可以直接用最简单的Dijkstra算法。

算法设计

图的构建通过数组的方式来实现，用 `pre[maxn]`，`last[maxn]` 两个 `int` 类型的数组来记录记录边的顺序，`to[maxn]` 来记录边的目的节点，`weight[maxn]` 来记录边的权值，边的顺序依照加入的顺序，通过 `cnt` 累加得到一个固定的 `index`。为了在 Dijkstra 算法中，记录每个节点到起始节点到距离便于后续的更新和计算，利用一个 `dis[maxn]` 数组来进行保存，每次更新时都把最短路径填入数组。

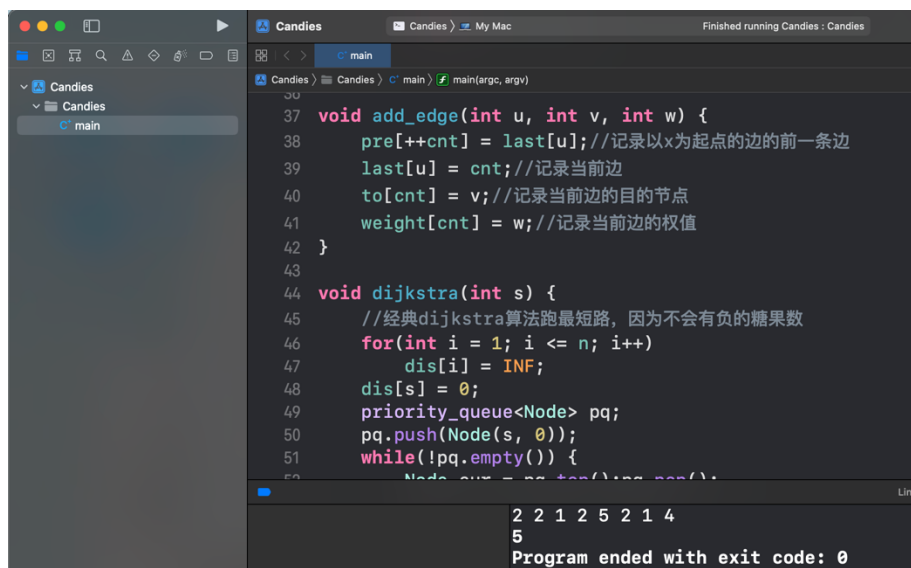
再设计一个结构体 `Node`，用来在 Dijkstra 算法中保存节点的 `index` 和最短路径权值。当节点放在优先队列中时，路径权值越小优先级越高对应也在队列中也越靠前，便于 Dijkstra 算法选取最短路径。

```
struct Node {
    int index, v;
    Node () {}
    Node (int a, int b) : index(a), v(b) {}
    bool friend operator < (Node x, Node y) {
        //传入 Node 节点的 v 值越大，在优先队列中的优先值越小
        return x.v > y.v;
    }
};
```

按照上述分析，在输入时，读取 `M` 个约束，构建好完整的图，然后跑 Dijkstra 算法来得到一个从起始到结束的最短路径，这个值就是满足约束且得到的最大糖果差距，直接输出即可。

测试分析

首先，我们在本地进行用例测试，然后在上传到远程 OJ 上进行测试，测试通过截图如第一部分所示。本地用例测试结果如下：



```
37 void add_edge(int u, int v, int w) {
38     pre[++cnt] = last[u]; //记录以x为起点的边的前一条边
39     last[u] = cnt; //记录当前边
40     to[cnt] = v; //记录当前边的目的节点
41     weight[cnt] = w; //记录当前边的权值
42 }
43
44 void dijkstra(int s) {
45     //经典dijkstra算法跑最短路，因为不会有负糖果数
46     for(int i = 1; i <= n; i++)
47         dis[i] = INF;
48     dis[s] = 0;
49     priority_queue<Node> pq;
50     pq.push(Node(s, 0));
51     while(!pq.empty()) {
52         Node cur = pq.top(); pq.pop();
53         for(int i = 1; i <= n; i++) {
54             if (weight[pre[i]] + dis[cur.index] < dis[i]) {
55                 dis[i] = weight[pre[i]] + dis[cur.index];
56                 last[i] = pre[i];
57                 to[i] = cur.index;
58                 pq.push(Node(i, dis[i]));
59             }
60         }
61     }
62 }
```

2 2 1 2 5 2 1 4
5
Program ended with exit code: 0

图 3-4 Candies 问题测试用例

时间复杂度分析： $O(n^2)$. 算法的时间复杂度核心来自 Dijkstra 算法，而 Dijkstra 算法的核心算法是在更新各个最短路径，得到的时间复杂度为 $O(n^2)$.

3.2.2 Cashier Employment

问题描述

德黑兰的一家超市每天 24 小时营业，需要一些收银员来满足其需求。超市经理雇你来帮他解决问题。问题是超市在每天的不同时间需要不同数量的收银员(例如，午夜以后有几个收银员，下午有很多的收银员)来为顾客提供良好的服务，他希望雇佣最少数量的收银员来做这项工作。

在一天中每一小时的时间里，经理会为你提供最少数量的收银员。该数据表示为 $R(0)$, $R(1)$, \dots , $R(23)$: $R(0)$ 表示从午夜到凌晨 1 点所需的最少收银员数量， $R(1)$ 表示从凌晨 1 点到凌晨 2 点期间所需的这个数字，以此类推。注意这些数字每天都是一样的。有 N 个合格的申请人申请这份工作。每个申请人 i 每 24 小时工作一次，正好从一个指定的小时开始不间断工作 8 个小时，例如 t_i ($0 \leq t_i \leq 23$)，正好从一个小时的开始提到。也就是说，如果第一个申请人被录用，他/她将从准时开始工作 8 小时。收银员不会互相替换，而是严格按照预定时间工作，而且有足够的收银机和柜台供那些被雇佣的人使用。

你要写一个程序来读取数据，雇佣的人数和时间都是非负整数，计算出满足上述约束条件所需要的最少收银员数量。请注意，对于一个特定的时间段，收银员的数量可能比所需的最少数量要多。

输入输出

输入的第一行是这个问题的测试用例的数量(最多 20 个)。每个测试用例从 24 个整数开始，代表 $R(0)$, $R(1)$, \dots , $R(23)$ (在一行 $R(i)$ 最多为 1000)。然后在另一行是 N ($0 \leq N \leq 1000$)，是申请人的数量，在这之后的 N 行，每一行包含一个 t_i ($0 \leq t_i \leq 23$)。测试用例之间没有空行。

对于每个测试用例，输出应该写在一行中，这是所需收银员的最少数量。如果测试用例没有解决方案，你应该为该用例输出 No Solution。

解题分析

同样的在差分约束系统中，最重要的就是找到约束，根据约束建立图，然后套用图的最短路算法即可。分析该题目题意，我们最容易看出来的第一个约束就是在某个时间段雇佣的人数应该要多于或者等于所需要的收银员数量。第二，由于给出了每个时间段来应聘的人

数，那么我们在这个时间段雇佣的人数也不能多于这个人数，因为不会多出人来给我们雇佣。这两个约束是非常直观的，而另一个约束很难直接想到，我们会在下面进行讨论。

为了接下来讨论更加方便我们做如下的定义：

$R[i]$ 表示在第 i 个小时至少也要的人数；

$Ts[i]$ 表示应聘者中可以在第 i 个小时开始工作的人数；

$s[i]$ 表示前 i 个小时雇佣的人数。

差分约束如下：

$$1. 0 \leq s[i] - s[i-1] \leq work[i]$$

某一个小时雇佣的人数不能超过在这个小时来应聘的总人数，因为 $s[i]$ 代表了前 i 个小时雇佣的总人数，那么两个相邻值之差就是在这个 i 小时雇佣的人数。我们发现这个小时雇佣的人数必须比这个小时来应聘的人少，因为人不会多出来。而且这个雇佣的人数必须大于等于 0，因为在这一个小时我们不能雇佣负数个人。

$$2. i \geq 8 \quad need[i] \leq s[i] - s[i - 8]$$

在 $i \geq 8$ 时，我们要考虑的是八个小时前雇佣的人已经下班了，那么 i 时雇佣的人 $s[i]$ 减去八个小时前雇佣的人数 $s[i - 8]$ 也就是当前已经下班的人数，就是当前 i 时所在上班的全部人数，这个人数必须要大于此时所需要的人。

我们提前定义一个 $s[24]$ 代表了前 24 小时雇佣的总人数，也就是我们最终要求出的雇佣人数。

$$i < 8 \quad s[i] + s[24] - s[16 + i] \geq need[i]$$

$i < 8$ 时，此时存在雇佣的人跨天工作的现象，在 7 点前都存在前一天的员工还在工作。而由于每天每个时间点雇佣的人数都是一样的，我们可以把这个人数转化成当天晚上雇佣的人。也就是说前 24 小时雇佣的总人数 $s[24]$ 减去前 $16+i$ 个小时雇佣的人 $s[16 + i]$ 就代表了之前一天雇佣的人到 i 时还在工作的人数，这个人数加上当天到 i 时雇佣的人数 $s[i]$ 就是 i 时还在工作的总人数，这个人数必须要比所需要的人更多。简单来说就是能工作到 i 点的人，加上第二天雇佣的工作到 i 点的人，要比所需要的人多。

$$3. s[24] - s[0] \geq sum$$

sum 表示最终我们一天内雇佣的总人数。这个 sum 和上述我们提到的 $s[24]$ 要求是一致的。由于出现了 sum 这个未知数，求解的时候我们需要从小到大枚举 sum 的值，直到 SPFA 求解最短路成功时即可停止，这个求解成功最后是通过我们计算出的 $s[24]$ 需要与 sum 保持一致来判断的，如果两者相等了，那么就是求解成功。或者也可以采用二分枚举 sum 以加快速度，我在这里中没有做这个处理。用 SPFA 算法是因为有的约束可能会出

现负数的现象，而 SPFA 是利用队列对 Bellman-Ford 算法进行优化的算法，可以处理负权值的图的最短路径问题，也可以自动判别出负权环路。

算法设计

我们建立一个结构体用来保存所构建的图中边的数据，结构体中具体数据如下所示：

```
struct Edge {  
    int u, v, w; //边的起始点，终止点和权值  
    int next;  
};
```

Edge中的next变量用于保存图的构建关系，保存该边同样一个起始节点的下一条边，那么为了保存住这个边的关系，我们自然需要一个数组head[maxn]来进行按序保存。

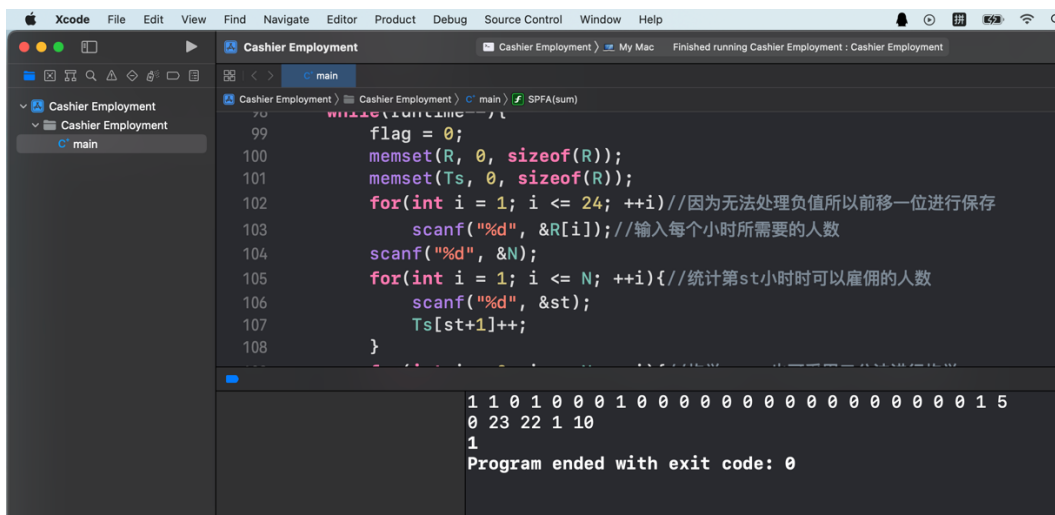
除此之外，我们还需要一个类型为Edge的数组edges[maxn]来进行保存所有的边。在跑 SPFA算法时，我们需要三个辅助的数组：dist[maxn]用来保存从源节点开始到当前节点的最短路径距离，vis[maxn]用来标记节点是否被取走过，used[maxn]标记该节点被使用的次数，如果这个次数超过了24，那么直接返回，不能继续进行算法也不能作为一种可行解。

SPFA算法采用队列queue的数据结构进行存储，队列中保存节点的顺序号，如果该节点有松弛操作的更新并且当前未被标记，则放入队列中，这样子把队列遍历直至空为止，也就是把所有边的松弛操作循环所有的节点数次。而在每次循环中还有一个小的循环，根据head[maxn]中保存的顺序来进行遍历，每次都取当前边中保存的next下一条边来进行更新。

SPFA算法跑完双循环之后，我们如何判断当前是否可以满足雇佣以及雇佣的人数呢？如上小节所述，我们需要判断跑完的dist[24]和我们用来枚举的sum值是否一致，如果一致那么满足条件，此时枚举到的sum值就是所需要的雇佣最小值，而如果不一致，则需要继续枚举。假如说枚举完来应聘的总人数都没找到一个满足条件的解的话，那么我们就知道了此时不存在解，我们直接输出No Solution即可。

测试分析

首先，我们在本地进行用例测试，然后在上传到远程OJ上进行测试，测试通过截图如第一部分所示。本地用例测试结果如下：



```
99     flag = 0;
100    memset(R, 0, sizeof(R));
101    memset(Ts, 0, sizeof(R));
102    for(int i = 1; i <= 24; ++i) //因为无法处理负值所以前移一位进行保存
103        scanf("%d", &R[i]); //输入每个小时所需要的人数
104    scanf("%d", &N);
105    for(int i = 1; i <= N; ++i) { //统计第st小时时可以雇佣的人数
106        scanf("%d", &st);
107        Ts[st+1]++;
108    }
```

```
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
1
Program ended with exit code: 0
```

图 3-5 Cashier Employment 问题测试用例

时间复杂度分析： $O(nm_e)$. n 表示节点数目， m_e 表示边的数目。算法的时间复杂度核心来自 SPFA 算法，而 SPFA 算法的核心算法是在更新各个最短路径，内外循环的逻辑在上面已经解释清楚了，得到的时间复杂度为 $O(nm_e)$ 。

四、总结

4.1 已完成工作

在这次算法实践的课程学习中，我总共完成了五个专题的学习，分别是并查集、树状数组、LCA 和 RMQ 问题、线段树和差分约束系统。同时，完成了对应五个专题的各自两道题。对于并查集专题，我和小组成员一起完成了答辩，我通过总结制作 ppt，更加深入了解了并查集。对于差分约束系统，我完成了这份报告的撰写，对这个专题以及这两道题又重新再理清了一次思路。

4.2 需要进一步完善的地方

其实在之前的学习过程中，由于在学期接近尾声，时间有很大的限制。不管是理论学习还是题目的分析理解，都是浅尝辄止，停留在一个了解这些理论并且理解了的学习层面上，没有做到掌握的层次。相对而言并查集和差分约束系统由于有二次学习和整理书面资料，对这两者的掌握更加深刻。所以我想在后续的学习生活中，应该要继续加强对这些学过的知识的理解 and 应用。应该要深入地去理解这些算法本质是什么，和哪些数据结构能产生关系，有哪些常见的应用场景，有哪些经典的建模方式。这些问题应该要贯穿在后续学习的过程中，加强对其本质的思考。

4.3 心得体会

在学习算法实践这门课之前，其实我们也学习了算法设计与分析。在算法设计与分析课程中，我们学习了相对而言更加基础的贪心、分治、动态规划、网络流等等算法，对于算法

是什么有了一个初步的认识。计算机算法，抽象而言就是计算机解决问题的方法和步骤。我们在学习的这些算法，其实就是前人在万千的试验中总结出来的解决问题的规律和模版。

在算法实践中，我又接触到了这些相对而言更加进阶的算法，并且通过了一些算法题来进行实践。学习理论的过程中我时常感叹这些算法设计的精妙，很多时候我们都很难找到一个完美的解决问题的办法，需要权衡优点和缺点，需要进行审慎地思考。

而在解决实际算法问题中，我又更深一步认识到了根据算法进行建模的重要性。像我这样初入门的人，思考一道算法题，很难能直接看到这道题能用什么算法，或者是不知道如何建模。那如何进行提升呢？我想，空想是没用的，我们既然已经站在了巨人的肩膀上，那就更好地去利用这些学习的资源。

作为计算机科学与技术专业的学生，打好一个良好的算法基础是非常重要的。我想从算法实践这门课出发，一方面对已经学过的这些算法进行更加深入地理解，去了解更多的应用场景、建模方法，去接触更多的题；另一方面，从这些算法进行广度拓展，我相信还有很多的算法需要我们进行探索，比如这次课程我还未完成的线段树学习，后续的学习中要进行安排学习。

华中科技大学课程设计报告

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字: 

二、对课程设计的学术评语（教师填写）

三、对课程设计的评分（教师填写）

评分项目 (分值)	报告撰写 (50 分)	课设过程 (50 分)	最终评定 (100 分)
得分			

指导教师签字: _____