華中科技大學

《算法设计与分析实践》

实 验 报 告

成　　绩：
评　　语：
教师签字：
评阅日期：

专业班级：计算机科学与技术 2407 班

学　　号：U202414802

姓　　名：朱俊瑞

指导教师：苏宙行

完成日期：2026.1.14

计算机科学与技术学院

2025 年 12 月

目 录

1 题目完成情况

本次实验中，我一共完成了 19 道题目，具体如下：

序号	题目编号	题目名称	类型	附加知识点
1	01-P2678	跳石头	分治	二分
2	03-P1228	地毯填补问题	分治	
3	04-P1220	关路灯	动态规划	搜索
4	08-P1437	敲砖块	动态规划	
5	09-P1434	滑雪	动态规划	记忆化搜索
6	11-P2018	消息传递	动态规划	树形
7	14-P2512	糖果传递	贪心	中位数
8	16-P1658	购物	贪心	
9	17-P1090	合并果子	贪心	优先队列
10	18-P1443	马的遍历	搜索	BFS
11	20-P2895	Meteor Shower S	搜索	BFS
12	23-P1784	数独	搜索	DFS
13	30-P3809	后缀排序	后缀数组	
14	33-P3374	树状数组 1	树状数组	
15	34-P3368	树状数组 2	树状数组	区间查询
16	36-P1816	忠诚	线段树	RMQ、ST 表
17	37-P3373	线段树 2	线段树	
18	42-P1111	修复公路	并查集	
19	47-P1314	聪明的质监员	前缀和	

以下是我的洛谷个人主页练习栏：

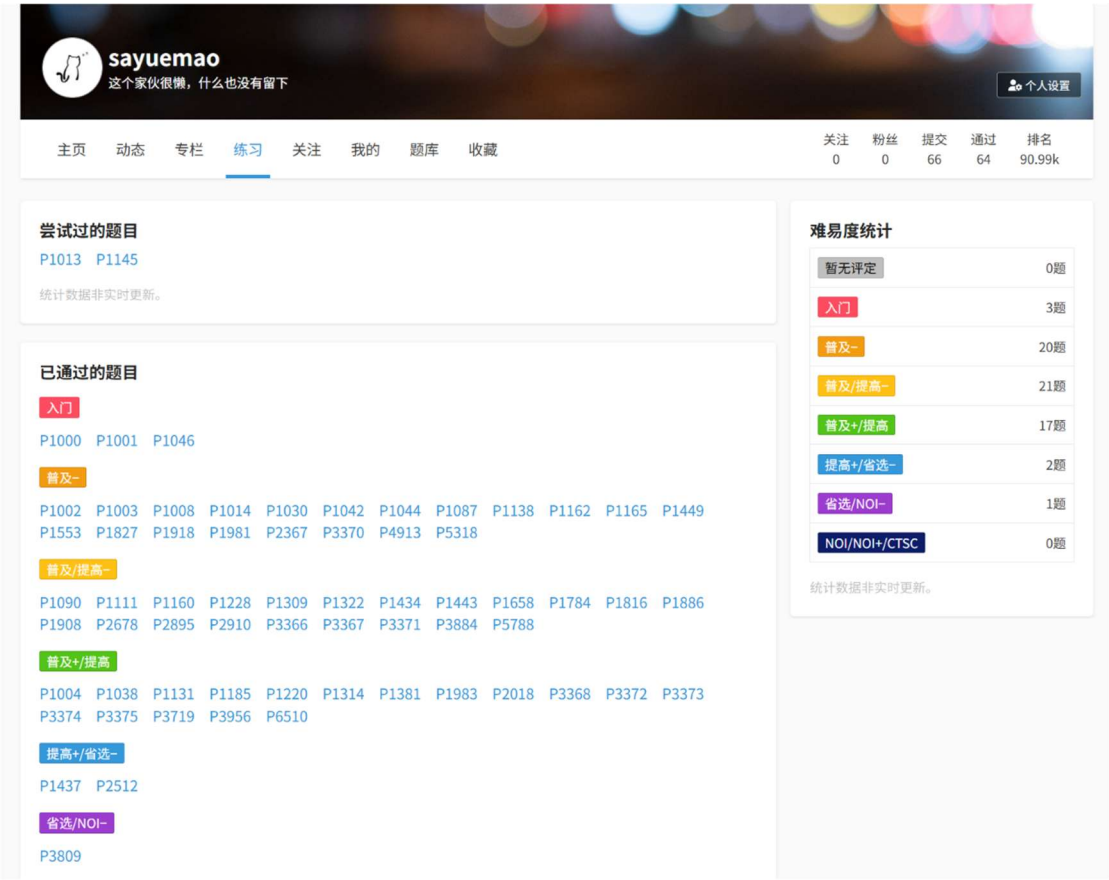

图 1-1 洛谷个人主页练习栏

以下为我完成各个题目的截图：

✓	P2678	[NOIP 2015 提高组] 跳石头	NOIP 提高组 2015	普及/提高-
–	P1242	新汉诺塔		提高+/省选-
✓	P1228	地毯填补问题		普及/提高-
✓	P1220	关路灯		普及+/提高
–	P4170	[CQOI2007] 涂色	各省省选 重庆 2007	提高+/省选-
–	P1854	[IOI 1999] 花店橱窗布置	IOI 1999	普及/提高-
–	P4999	烦人的数学作业		普及+/提高
✓	P1437	[HNOI2004] 敲砖块	各省省选 湖南 2004	提高+/省选-
✓	P1434	[SHOI2002] 滑雪	各省省选 上海 2002	普及/提高-
–	P4017	最大食物链计数		普及/提高-
✓	P2018	消息传递		普及+/提高
–	P1106	删数问题		普及/提高-
–	P1080	[NOIP 2012 提高组] 国王游戏	NOIP 提高组 2012	普及+/提高
✓	P2512	[HAOI2008] 糖果传递	各省省选 河南 2008	提高+/省选-
–	P2127	序列排序		提高+/省选-
✓	P1658	购物		普及/提高-
✓	P1090	[NOIP 2004 提高组] 合并果子	NOIP 提高组 2004	普及/提高-
✓	P1443	马的遍历		普及/提高-
–	P1135	奇怪的电梯		普及/提高-

图 1-2 题目通过截图 1

✓	P2895	[USACO08FEB] Meteor Shower S	USACO 2008	普及/提高-
–	P1825	[USACO11OPEN] Corn Maze S	USACO 2011	普及/提高-
–	P1433	吃奶酪		普及+/提高
✓	P1784	数独		普及/提高-
–	P1141	01迷宫		普及/提高-
–	P1019	[NOIP 2000 提高组] 单词接龙（疑似错题）	NOIP 提高组 2000	普及/提高-
–	P2349	金字塔		普及+/提高
–	P2324	[SCOI2005] 骑士精神	各省省选 四川 2005	提高+/省选-
–	P5691	[NOI2001] 方程的解数	NOI 2001	普及+/提高
–	P3067	[USACO12OPEN] Balanced Cow Subsets G	USACO 2012	提高+/省选-
✓	P3809	【模板】后缀排序	模板题	省选/NOI-
–	P2852	[USACO06DEC] Milk Patterns G	USACO 2006	普及+/提高
–	P2870	[USACO07DEC] Best Cow Line G	USACO 2007	普及+/提高
✓	P3374	【模板】树状数组 1	模板题	普及+/提高
✓	P3368	【模板】树状数组 2	模板题	普及+/提高
–	P3605	[USACO17JAN] Promotion Counting P	USACO 2017	提高+/省选-
✓	P1816	忠诚		普及/提高-
✓	P3373	【模板】线段树 2	模板题	普及+/提高
–	P1904	天际线		普及-
–	P1993	小 K 的农场		普及+/提高
–	P3275	[SCOI2011] 糖果	各省省选 四川 2011	提高+/省选-

图 1-3 题目通过截图 2


P1250 种树 普及/提高-
P1111 修复公路 普及/提高-
P1196 [NOI2002] 银河英雄传说 NOI 2002 普及+/提高
P2024 [NOI2001] 食物链 NOI 2001 普及+/提高
P1197 [JSOI2008] 星球大战 各省省选 江苏 2008 普及+/提高
P2756 飞行员配对方案问题 网络流与线性规划 24 题 普及+/提高
P1314 [NOIP 2011 提高组] 聪明的质监员 NOIP 提高组 2011 普及+/提高
P1726 上白泽慧音 普及+/提高
P3379 【模板】最近公共祖先（LCA） 模板题 普及/提高-
P3865 【模板】ST 表 & RMQ 问题 模板题 普及/提高-

图 1-4 题目通过截图 3

2 P2678 跳石头

2.1 题目描述

一年一度的“跳石头”比赛又要开始了！

这项比赛将在一条笔直的河道中进行，河道中分布着一些巨大岩石。组委会已经选择好了两块岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石（不含起点和终点的岩石）。在比赛过程中，选手们将从起点出发，每一步跳向相邻的岩石，直至到达终点。

为了提高比赛难度，组委会计划移走一些岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。由于预算限制，组委会至多从起点和终点之间移走 M 块岩石（不能移走起点和终点的岩石）。

2.2 输入格式

第一行包含三个整数 L,N,M，分别表示起点到终点的距离，起点和终点之间的岩石数，以及组委会至多移走的岩石数。保证 $L\geq 1$ 且 $N\geq M\geq 0$。

接下来 N 行，每行一个整数，第 i 行的整数 $Di(0<Di<L)$， 表示第 i 块岩石与起点的距离。这些岩石按与起点距离从小到大的顺序给出，且不会有两个岩石出现在同一个位置。

对于 20%的数据，$0\leq M\leq N\leq 10$。

对于 50% 的数据，$0\leq M\leq N\leq 100$。

对于 100% 的数据，$0\leq M\leq N\leq 50000,1\leq L\leq 10^9$。

2.3 输出格式

一个整数，即最短跳跃距离的最大值。

2.4 输入输出样例

图 2-1 P2678 跳石头输入输出样例

2.5 题目算法分析

若对移走的石头的每一种可能进行遍历，则需要的时间复杂度是 $O(M^N)$，在 $0\le M\le N\le 50000$ 条件下效率很低。而本题需要“在至多移除 M 块岩石的限制下，使选手跳跃的最短距离尽可能长”，是典型的“最大化最小值”问题。解决这个问题的关键是若某一距离 mid 是可行解（即移除≤M 块岩石可使所有跳跃距离≥mid），则所有小于 mid 的距离均为可行解；若 mid 不可行，则所有大于 mid 的距离均不可行。基于此特性，二分答案是最优解题思路。同时运用贪心使算法结果逐渐接近所求的最短距离最大值。

所求的最短跳跃距离的最大值，必然不会超出起点到终点的距离，所以我们可以对[1,L]这个区间进行二分寻找答案。具体来说，我们每一次二分寻找答案的时候，就是假设选手能跳的最远距离为 mid，然后我们遍历 1-N 块石头，假设为选手的下一块石头，如果移走之后还

可以小于这个 mid 的距离（即检验这个 mid 大小的距离是否大于当前下一个石头与当前模拟过程选手位置的距离），则说明这块石头需要被移除，因为它与当前模拟过程选手位置小于了我们假设的最短跳跃距离），并让记录移走石头的数目增加 1。遍历完后再检验这个移走石头数目是否满足小于 M 的要求。即每一次二分答案，对 mid 进行一次检验(调用 isSatisfied 函数)，是否可以作为一个最短距离，随后再对下一步二分的区间进行调整直至二分完全结束。

isSatisfied 函数：

```
01.   bool isSatisfied(int x)
02.   {
03.    int rockn = 0;
04.    int i = 0;
05.    int nowlength = 0;
06.    while(i < n+1)
07.    {
08.       ++i;
09.       if(dist[i] - nowlength < x)
10.       {
11.          ++rockn;//要被移除的石头 i
12.       }
13.       else
14.       {
15.          nowlength = dist[i];//更新当前人走到的石头，离起点的距
      离
16.       }
17.    }
18.    if(rockn > m)
19.       return false;
20.    else
21.       return true;
22.   }
```

二分答案判断：

```
01.    while(l <= r)
```

```
    {
        mid = l + (r-l)/2;
        if(isSatisfied(mid))
        {
            ans = mid;
            l = mid+1;
        }
        else
        {
            r = mid - 1;
        }
    }
```

如上代码，若当前 mid 满足可以是一个最短距离，就赋值给 ans 并让 l 等于 mid+1，以探寻可能的更大值即继续寻找最短跳跃距离的最大值；若当前 mid 不能满足，则说明 mid 过大，于是令 r 等于 mid-1 向左半区间寻找最短跳跃距离。

2.6 运行结果

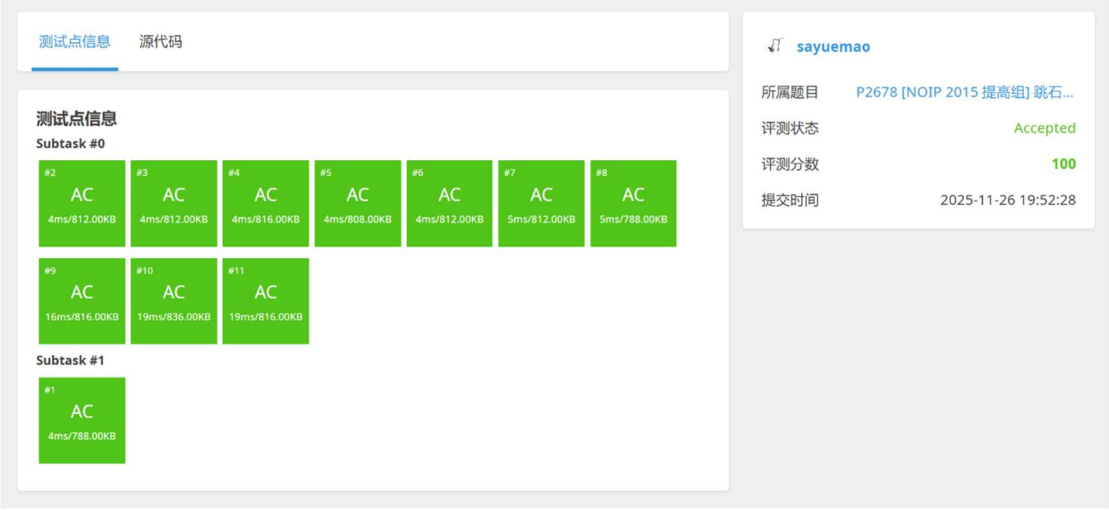

图 2-2 P2678 跳石头测试通过

代码在洛谷 OJ 上可正确测试通过。

2.7 性能分析

时间复杂度分析：二分答案的区间为[1, L]，二分次数为 O (logL)；每次二分需调用 isSatisfied 函数遍历 N 块岩石，时间复杂度为 O (N)。总时间复杂度为 O(NlogL)。

空间复杂度分析：仅用数组 dist 存储 N 块岩石与起点的距离（含终点位置），空间开销与岩石数量正相关。总空间复杂度为 O(N)。

2.8 优化思路

2.8.1 边界定义的精准化

早期实现常将终点作为特殊情况处理，例如在计算最后一段距离时单独判断（如 if j == len(DL)-1）。这可能会埋下隐患：当所有中间岩石被移除时，终点与起点的间距 L 可能被漏算。

优化方案：强制将起点（0）和终点（L）作为固定元素插入数组首尾，形成 rock[0...n+1]的结构（如样例中[0,2,11,14,17,21,25]）。遍历只需从 rock[1]到 rock[n]，终点自动参与间距计算。

2.8.2 考虑用位运算代替除法

在二分查找过程中，mid = (left1 + right1) / 2 可能会频繁计算。如果代码特别关注性能，可以考虑使用位运算代替除法。具体而言，可以使用位运算 mid = (left1+right1)>>1 相当于除以 2，并且在某些情况下可以略微加速计算。位运算的执行速度通常比除法操作要快。

3 P1434 滑雪

3.1 题目描述

Michael 喜欢滑雪。这并不奇怪，因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael 想知道在一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子：

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度会减小。在上面的例子中，一条可行的滑坡为 24−17−16−1（从 24 开始，在 1 结束）。当然 25－24－23－…－3－2－1 更长。事实上，这是最长的一条。

3.2 输入格式

输入的第一行为表示区域的二维数组的行数 R 和列数 C。下面是 R 行，每行有 C 个数，代表高度（两个数字之间用 1 个空格间隔）。

对于 100% 的数据，$1 \le R,C \le 100$。

3.3 输出格式

输出区域中最长滑坡的长度。

3.4 输入输出样例

图 2-1 P1434 滑雪输入输出样例

3.5 题目算法分析

若采用暴力深度搜索遍历所有可能的滑坡路径，每个点都可能被重复访问多次，在 R、C 最大为 100 的情况下，总运算量会达到 $O((R\times C)^2)$，即 10^8次以上，必然导致超时。本题核心需求是“寻找仅能向上下左右相邻且高度更低的点滑动的最长路径”，属于典型的 DAG 最长路径问题。因滑动规则限定“高度严格递减”，路径不存在环结构，且每个点的最长路径仅依赖于周边更高点的路径长度，因此采用“优先队列+动态规划”的组合算法，既保证无后效性，又能高效求解。

本题的核心思路是利用优先队列确保先处理高度更高的点，再通过动态规划数组记录每个点的最长滑坡长度。定义动态规划数组 step[r][c]，用于存储以坐标(i,j)为终点的最长滑坡长度，初始值均设为

1，因为每个点自身可作为长度为 1 的滑坡。结构体 node 存储点的坐标(x,y)和对应高度 val，通过重载<运算符将优先队列构建为大根堆，使高度越大的点优先级越高，确保处理当前点时，其周边所有更高点的最长路径已计算完成。

实现过程中首先读入二维高度数组 a[r][c]，将所有点的坐标和高度存入优先队列，同时初始化 step 数组为 1。随后进入循环，每次取出堆顶的最高点，依次检查其上下左右四个相邻方向：若相邻点在数组边界内且高度高于当前点（满足“从相邻点滑到当前点”的规则），则记录该相邻点的 step 值，并更新当前的最大 step 值 maxval。若存在这样的有效相邻点，说明当前点的最长路径可在该相邻点路径基础上延长 1，因此将 step[e.x][e.y]设为 maxval + 1。所有点处理完毕后，遍历 step 数组，其中的最大值即为整个区域的最长滑坡长度。

代码中 step 数组初始化为 1，即使某点周边无更高点，其自身也是一条合法的最短路径，保证了结果的完整性。这种算法通过优先队列优化处理顺序，使动态规划的无后效性得到满足，能高效应对题目给定的数据规模。

动态规划核心部分代码：

```
01.   while (!que.empty())
02.       {
03.           node e = que.top();
04.           que.pop();
05.           ll maxval = 1;
06.           bool hasroad = false;
07.           if (e.x - 1 >= 1 && a[e.x-1][e.y] > a[e.x][e.y])
08.           {
```

```
09.             if(maxval < step[e.x - 1][e.y]) maxval =
      step[e.x - 1][e.y];
10.             hasroad = true;
11.         }
12.         if (e.y - 1 >= 1 && a[e.x ][e.y-1] > a[e.x][e.y])
13.         {
14.             if (maxval < step[e.x][e.y-1])maxval =
      step[e.x][e.y-1];
15.             hasroad = true;
16.         }
17.         if (e.x + 1 <= r && a[e.x + 1][e.y] > a[e.x][e.y])
18.         {
19.             if (maxval < step[e.x + 1][e.y])maxval =
      step[e.x + 1][e.y];
20.             hasroad = true;
21.         }
22.         if (e.y+ 1 <= c && a[e.x ][e.y+1] > a[e.x][e.y])
23.         {
24.             if (maxval < step[e.x][e.y+1])maxval =
      step[e.x][e.y+1];
25.             hasroad = true;
26.         }
27.         if(hasroad) step[e.x][e.y] = maxval+1;
28.     }
```

检测上下左右四个方向的点，如果能下滑（即其高度小于当前点的高度，因为有相同高度的点存在所以需要进行这个判断），则再比较更新最大路径长度。

3.6 运行结果

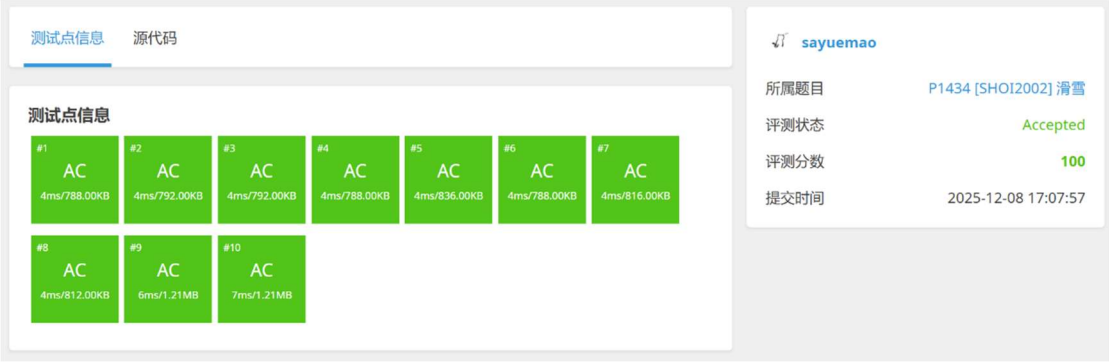

图 3-2 P1434 滑雪测试通过

代码在洛谷 OJ 上可正确测试通过。

3.7 性能分析

时间复杂度分析：每个点入队和出队的时间复杂度为 O(log (R×C))（优先队列的插入与弹出操作），共需处理 R×C 个点；每个点需遍历 4 个相邻方向，时间复杂度为 O(1)，整体遍历开销为 O(R×C)；总复杂度由优先队列操作主导，对于 R、C≤100 的规模（共 10^4个点），运算量约为 $10^4 \times 14 = 1.4 \times 10^5$，完全满足时间限制。所以总的时间复杂度为 O(R×C×log(R×C))。

空间复杂度分析：二维数组 a 和 step 各占用 O(R×C)空间，用于存储高度数据和路径长度；优先队列最多存储 R×C 个节点，整体空间开销与数组规模线性相关，无额外冗余存储。所以总的空间复杂度为 O(R×C)。

3.8 优化思路

3.8.1 方向数组简化代码

当前通过四次独立判断处理四个相邻方向，代码存在重复。可预先定义方向数组 int dir[4][2]={{-1,0},{1,0},{0,-1},{0,1}}，通过循环遍历四个方向，统一实现相邻点的边界判断和高度对比，减少重复代码，降低出错概率。

3.8.2 优先队列效率优化

当前使用大根堆实现高度更高的点先处理，可改用桶排序替代优

先队列进一步提升效率。统计所有高度的取值范围，按高度从大到小划分桶，将同高度的点存入对应桶中，再按高度降序遍历桶处理节点。该优化可将入队出队的时间复杂度从 O(log (R×C))降至 O(1)，适合高度值分布集中的场景。

4 P1443 马的遍历

4.1 题目描述

有一个 $n \times m$ 的棋盘，在某个点 (x,y) 上有一个马，要求你计算出马到达棋盘上任意一个点最少要走几步。

4.2 输入格式

输入只有一行四个整数，分别为 n,m,x,y。

对于全部的测试点，保证 $1 \le x \le n \le 400$，$1 \le y \le m \le 400$。

4.3 输出格式

一个 $n \times m$ 的矩阵，代表马到达某个点最少要走几步（不能到达则输出 -1）。

4.4 输入输出样例

输入 #1	复制	输出 #1	复制
3 3 1 1		0 3 2 3 -1 1 2 1 4	

图 4-1 P1443 马的遍历输入输出样例

4.5 题目算法分析

若采用深度优先搜索（DFS）求解，马的 8 种移动方向会导致棋盘节点被大量重复访问，时间复杂度会达到 $O((n \times m)^2)$，不仅效率较

低，还可能因递归栈溢出导致程序崩溃。本题核心需求是计算马到达棋盘任意点的最少步数，是典型的无权图最短路径问题——棋盘上每个点视为节点，马的合法移动视为权重为 1 的边，而广度优先搜索（BFS）的核心特性是按层序遍历，第一次到达节点时的步数即为最短步数，恰好适配该需求，因此采用 BFS 结合队列是最优解题思路。

本题核心思路是利用 BFS 的层序遍历特性，配合访问标记避免节点重复入队，确保每个节点仅被处理一次。BFS 的核心逻辑是先处理当前层的所有节点，再处理下一层，对应到题目中就是先处理走第 k 步能到达的所有点，再处理走第 k+1 步能到达的点，因此第一次到达某个点时的步数必然是最少步数。于是可以通过队列的数据结构存储待处理的节点实现 BFS 遍历逻辑，同时用访问标记记录已处理的节点，避免同一节点多次入队造成的效率浪费和步数错误。

首先确定马的初始位置，将其作为 BFS 的起点，初始化起点步数为 0，并标记为已访问后加入队列。随后进入循环，每次从队列中取出一个节点，依次检查马的 8 种“日”字形移动方向（即横向移动 1 格纵向移动 2 格、横向移动 2 格纵向移动 1 格的组合，共 8 种合法方向）。对于每个方向对应的新位置，判断其是否在棋盘范围内且未被访问，若满足条件，则将该新位置的步数设为当前节点步数+1，标记为已访问并加入队列。重复这一过程，直到队列为空，此时所有可达节点的最少步数都已计算完成，未被访问的节点则视为不可到达，输出-1 即可。

BFS 搜索核心代码：

```
01.      queue<node> q;
02.      q.push({x,y,0});
03.      while (!q.empty())
04.      {
05.          node t = q.front();
06.          q.pop();
07.          //八个方向
08.          if (t.x + 2 <= n && t.y + 1 <= m && visited[t.x +
     2][t.y + 1]==0)//右下左
09.          {
10.              chessboard[t.x + 2][t.y + 1] = t.bushu + 1;
11.              visited[t.x + 2][t.y + 1] = 1;
12.              q.push({t.x+2,t.y+1,chessboard[t.x + 2][t.y +
     1]});
13.          }
14.          if (t.x + 1 <= n && t.y + 2 <= m && visited[t.x +
     1][t.y + 2] == 0)//右下右
15.          {
16.              chessboard[t.x + 1][t.y + 2] = t.bushu + 1;
17.              visited[t.x + 1][t.y + 2] = 1;
18.              q.push({ t.x + 1,t.y + 2,chessboard[t.x +
     1][t.y + 2] });
19.          }
20.          if (t.x - 2 >= 1 && t.y + 1 <= m && visited[t.x -
     2][t.y + 1] == 0)//右上左
21.          {
22.              chessboard[t.x - 2][t.y + 1] = t.bushu + 1;
23.              visited[t.x - 2][t.y + 1] = 1;
24.              q.push({ t.x - 2,t.y + 1,chessboard[t.x -
     2][t.y + 1] });
25.          }
26.          if (t.x -1 >= 1 && t.y +2 <= m && visited[t.x -
     1][t.y + 2] == 0)//右上右
27.          {
28.              chessboard[t.x -1][t.y + 2] = t.bushu + 1;
29.              visited[t.x -1][t.y + 2] = 1;
30.              q.push({ t.x - 1,t.y + 2,chessboard[t.x -
     1][t.y + 2] });
31.          }
32.          if (t.x + 2 <= n && t.y - 1 >=1 && visited[t.x +
     2][t.y - 1] == 0)//左下右
33.          {
34.              chessboard[t.x + 2][t.y - 1] = t.bushu + 1;
35.              visited[t.x + 2][t.y - 1] = 1;
```

```
36.             q.push({ t.x + 2,t.y - 1,chessboard[t.x +
    2][t.y - 1] });
37.         }
38.         if (t.x + 1 <= n && t.y - 2 >= 1 && visited[t.x +
    1][t.y - 2] == 0)//左下左
39.         {
40.             chessboard[t.x + 1][t.y - 2] = t.bushu + 1;
41.             visited[t.x + 1][t.y - 2] = 1;
42.             q.push({ t.x + 1,t.y -2,chessboard[t.x + 1][t.y
    - 2] });
43.         }
44.         if (t.x -1 >= 1 && t.y -2 >= 1 && visited[t.x -
    1][t.y - 2] == 0)//左上左
45.         {
46.             chessboard[t.x - 1][t.y - 2] = t.bushu + 1;
47.             visited[t.x - 1][t.y - 2] = 1;
48.             q.push({ t.x -1,t.y -2,chessboard[t.x - 1][t.y
    - 2] });
49.         }
50.         if (t.x - 2 >= 1 && t.y - 1 >= 1 && visited[t.x -
    2][t.y - 1] == 0)//左上右
51.         {
52.             chessboard[t.x - 2][t.y - 1] = t.bushu + 1;
53.             visited[t.x - 2][t.y - 1] = 1;
54.             q.push({ t.x - 2,t.y - 1,chessboard[t.x -
    2][t.y - 1] });
55.         }
```

用 chessboard 二维数组记录每个棋盘点的步数。

4.6 运行结果

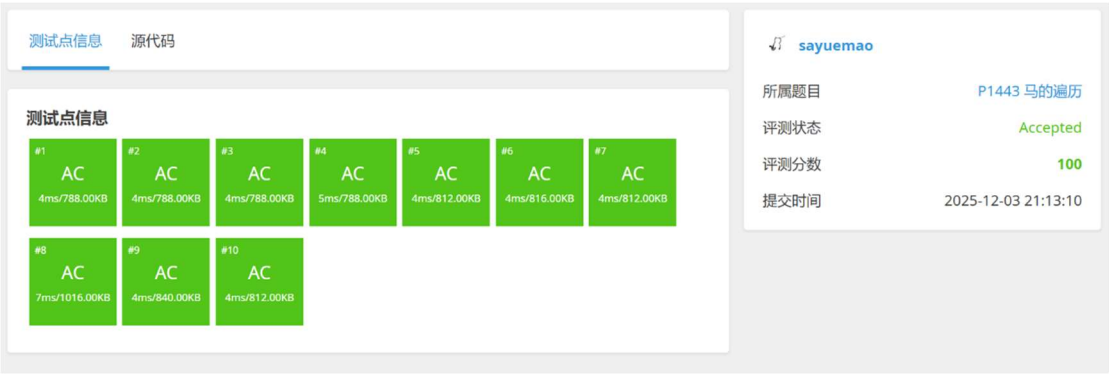

图 4-2 P1443 马的遍历测试通过

代码在洛谷 OJ 上可正确测试通过。

4.7 性能分析

时间复杂度分析：BFS 遍历过程中，每个棋盘节点最多入队和出队一次，棋盘总节点数为 n×m，因此时间复杂度为 O(n×m)。对于 n、m≤400 的规模，总运算量约为 1.6×10^5，完全满足时间限制。

空间复杂度分析：chessboard 数组和 visited 数组均占用 O(n×m)的空间，用于存储步数和访问状态；队列最多同时存储一层节点，最坏情况下（如棋盘全空）存储 O(n+m)个节点，整体空间复杂度为 O(n×m)，无额外冗余存储。

4.8 优化思路

4.8.1 简化移动方向判断：

当前代码通过 8 组独立判断处理马的移动方向，存在大量重复代码。可预先定义方向数组 int dir[8][2]={{2,1},{1,2},{-2,1},{-1,2},{2,-1},{1,-2},{-1,-2},{-2,-1}}，通过循环遍历 8 个方向，统一实现新坐标计算和边界判断，减少代码冗余，降低出错概率。

4.8.2 省略结构体中的 bushu 字段：

由于 chessboard[新 x][新 y]的值与节点的 bushu 字段完全一致（步数即 chessboard 存储的值），可省略结构体中的 bushu，在计算新节点步数时，直接通过 chessboard[t.x][t.y] + 1 获取，既简化了结构体定义，又减少了队列存储的空间开销。

5 P1111 修复公路

5.1 题目描述

A 地区在地震过后，连接所有村庄的公路都造成了损坏而无法通车。政府派人修复这些公路。

给出 A 地区的村庄数 N，和公路数 M，公路是双向的。并告诉你每条公路的连着哪两个村庄，并告诉你什么时候能修完这条公路。问最早什么时候任意两个村庄能够通车，即最早什么时候任意两条村庄都存在至少一条修复完成的道路（可以由多条公路连成一条道路）。

5.2 输入格式

第 1 行两个正整数 N,M。

下面 M 行，每行 3 个正整数 x,y,t，告诉你这条公路连着 x,y 两个村庄，在时间 t 时能修复完成这条公路。

$1\leq x,y\leq N\leq 10^3$，$1\leq M,t\leq 10^5$。

5.3 输出格式

如果全部公路修复完毕仍然存在两个村庄无法通车，则输出 -1，否则输出最早什么时候任意两个村庄能够通车。

5.4 输入输出样例

图 5-1 P1111 修复公路输入输出样例

5.5 题目算法分析

这道题是并查集的经典应用，思路是用贪心策略选早修复的公路，靠并查集维护村庄连通状态，什么时候所有村庄连成片，当前公路的修复时间就是答案。

具体做法：

1. 先把所有公路按修复时间 t 从小到大排序，确保每次处理的都是当前能用到的最早完工公路；
2. 初始化并查集：每个村庄的父节点都指向自己（fa[i]＝i），相当于每个村庄一开始都是独立的；再用一个变量 lastsinglenum 记录还需要多少次合并才能让所有村庄连通（即剩余的独立村庄个数）。n 个村庄要连通，最少需要合并 n-1 次，所以初始值设为 n-1；
3. 遍历排序后的公路，对每条公路连接的两个村庄做合并：用 findfather 函数找到两个村庄的所在集合的根节点，如果根节点不一样，说明两个村庄集合能合并，就把其中一个的根节点挂到另一个下面，同时 lastsinglenum 减 1（需要合并的次数少了一次）。每次合并后都要看看 lastsinglenum 是不是 0：如果是，

说明所有村庄已经连成一个整体，当前公路的修复时间就是题目所求的最早互通时间，直接输出；若遍历完所有公路，lastsinglenum 还大于 0，说明有些村庄连不上，输出-1。

查找集合根节点以及处理两个村庄之间存在公路：

```
01.   int findfather(int root)//迭代查找父亲
02.   {
03.       int t = root;
04.       while (t != fa[t]) t = fa[t];
05.       while (root != fa[root])
06.       {
07.          int a = root;
08.          root = fa[root];
09.          fa[a] = t;
10.       }
11.       return t;
12.   }
13.   void handle(int x, int y)
14.   {
15.       int xfather = findfather(x);
16.       int yfather = findfather(y);
17.       if (xfather != yfather)
18.       {
19.          fa[yfather] = xfather;
20.          --lastsinglenum;
21.       }
22.   }
```

fa 数组储存村庄 i 的所在集合的根节点。findfather 函数里同时使用了路径压缩（05-10 行），将查找过程中的每一个村庄的在 fa 数组对应的值都设置成根节点，这样下次查找时能更快找到根节点。

5.6 运行结果

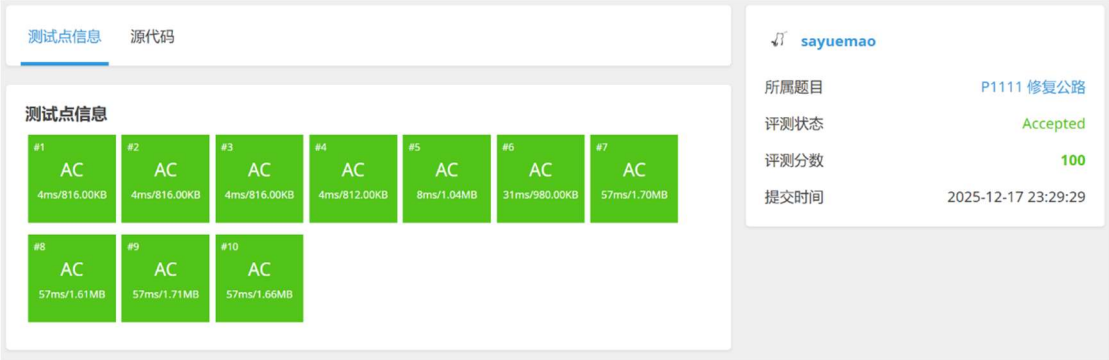

图 5-2 P1111 修复公路测试通过

代码在洛谷 OJ 上可正确测试通过。

5.7 性能分析

时间复杂度分析：M 条公路排序，使用 sort 函数时间复杂度可以近似是 O(MlogM)。并查集的查找和合并因为有路径压缩，近似 O(1)，所以整体时间复杂度 O(MlogM)。

空间复杂度分析：并查集的父节点数组 fa 占 O(N)空间，存储公路的结构体数组占 O(M)空间。

5.8 优化思路

5.8.1 并查集添加按秩合并

当前代码仅通过路径压缩优化并查集，可新增秩数组记录每个连通分量的深度（或节点数），合并时将深度较小的分量挂载到深度较大的分量上，避免连通分量过度失衡，进一步降低 findfather 的查询开销，让并查集操作更接近严格 O(1)。

5.8.2 排序策略优化

若输入中存在大量修复时间相同的公路，可采用稳定排序（如归并排序），确保同时间公路按输入顺序处理，避免因排序不稳定导致的无效合并；对于修复时间分布集中的场景，也可采用基数排序替代快速排序，减少排序过程中的比较操作，提升效率。

6 总结

6.1 实验总结

本次实验完成了洛谷平台不同类型的算法题目，我不仅巩固了分治、动态规划、贪心、搜索等知识，还学习了树状数组、线段树、并查集等算法。从最初对二分答案、BFS 等算法的机械套用，到后来能结合题目场景灵活调整逻辑，我的算法设计能力和编程能力都有了显著提升，也深刻体会到算法是解决问题的工具，理解原理比死记模板更重要。

通过本次实验，我积累了不少实用的解题经验：

1. **算法的灵活适配**：不同问题需要匹配最优算法。比如“跳石头”是典型的最大化最小值问题，二分答案+贪心验证是最优解；“马的遍历”作为无权图最短路径问题，BFS 的层序遍历特性刚好适配；“修复公路”则需要用并查集维护连通状态，搭配贪心策略优先处理早修复的公路。这让我明白，解题的关键是先分析问题本质，再选择合适的算法。
2. **细节决定成败**：算法题的坑往往藏在边界条件和特殊情况中。比如“跳石头”中需要将终点纳入数组遍历，否则会漏算最后一段距离；“马的遍历”中 8 个移动方向的边界判断不能遗漏，否则会导致数组越界；“修复公路”中并查集的路径压缩能大幅提升效率，少了这一步可能超时。当 WA 时可以下载出错的数据集看一下哪里出了问题，然后再根据相应代码逻辑逐步调试找

到错误点。

3. **模板的活学活用**：树状数组、线段树、并查集等基础算法都有固定模板，但实际题目中需要灵活调整。比如 “树状数组 2”的区间更新，需要结合差分思想对模板进行扩展。模板只是基础，理解算法的核心逻辑才能应对多样化的题目变形。

6.2 心得体会

1. **重视算法原理**：算法的核心不是代码本身，而是背后的逻辑推导比如二分答案的单调性、BFS 的最短路径特性、并查集的连通性维护，只有吃透原理，才能在不同题目中灵活运用。在算法设计与分析课堂上讲过的算法，都需要认真学习深入原理，才能真正在做题时派上用场。
2. **正视不足**：实验也暴露了我的短板：对复杂算法的优化理解不够深入，比如线段树的区间合并操作、树状数组的多维扩展；面对综合性题目时，拆解问题的能力不足，比如将动态规划与贪心结合的场景。后续我会针对性地加强这些薄弱点的练习，同时多总结同类题目的解题规律，形成自己的解题思路框架。另外我对算法的学习还不够透彻，了解的算法数量还不够多，还需要多加学习，主动探索，将算法学以致用。

总的来说，本次实验让我深刻体会到算法学习的 “循序渐进”——从基础模板到灵活运用，从单一算法到综合搭配，每一步都需要大量练习和思考。解未来我会继续积累题目经验，深化对算法原理的理解，

为后续的专业学习打下坚实基础。

6.3 课程建议

希望算法设计与分析实践课上能够多多增加交流的环节，让同学之间能够相互学习研究题解，这样或许可以让大家的算法水平更快进步。

附录（源代码）：

P2678 跳石头：

```
#include <iostream>
#include <algorithm>
using namespace std;
#define N 50005
// L 起点到终点距离     n 起点与终点间的岩石数量    m 组委会最多移走的岩石数
int n,m,L;
int dist[N];
int mid,ans;
bool isSatisfied(int x)
{
 int rockn = 0;
 int i = 0;
 int nowlength = 0;
 while(i < n+1)
 {
    ++i;
    if(dist[i] - nowlength < x)
    {
       ++rockn;//要被移除的石头 i
    }
    else
    {
       nowlength = dist[i];//更新当前人走到的石头，离起点的距离
    }
 }
 if(rockn > m)
    return false;
 else
    return true;
}
int main()
{
 cin>>L>>n>>m;
```

```
    for(int i = 1;i<=n;++i)
    {
        cin>>dist[i];
    }
    if(m>=n)
    {
        cout<<L;
        return 0;
    }
    dist[n+1] = L;
    int l = 1;
    int r = L;

    while(l <= r)
    {
        mid = l + (r-l)/2;
        if(isSatisfied(mid))
        {
            ans = mid;
            l = mid+1;
        }
        else
        {
            r = mid - 1;
        }
    }
    cout<<ans;
      return 0;
  }
```

P1434 滑雪

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
#define ll long long
ll r, c;
ll a[105][105],step[105][105];
struct node {
    ll x, y,val;
```

```
12.      bool operator < (const node& b) const
13.      {
14.          return val < b.val;
15.      }
16.  };
17.  priority_queue<node> que;
18.  int main() {
19.      cin >> r >> c;
20.      for (ll i = 1; i <= r; ++i)
21.      {
22.          for(ll j = 1;j <= c; ++j)
23.          {
24.              cin >> a[i][j];
25.              que.push({i,j,a[i][j]});
26.              step[i][j] = 1;
27.   }
28.      }
29.      while (!que.empty())
30.      {
31.          node e = que.top();
32.          que.pop();
33.          ll maxval = 1;
34.          bool hasroad = false;
35.          if (e.x - 1 >= 1 && a[e.x-1][e.y] > a[e.x][e.y])
36.          {
37.              if(maxval < step[e.x - 1][e.y]) maxval =
     step[e.x - 1][e.y];
38.              hasroad = true;
39.          }
40.          if (e.y - 1 >= 1 && a[e.x ][e.y-1] > a[e.x][e.y])
41.          {
42.              if (maxval < step[e.x][e.y-1])maxval =
     step[e.x][e.y-1];
43.              hasroad = true;
44.          }
45.          if (e.x + 1 <= r && a[e.x + 1][e.y] > a[e.x][e.y])
46.          {
47.              if (maxval < step[e.x + 1][e.y])maxval =
     step[e.x + 1][e.y];
48.              hasroad = true;
49.          }
50.          if (e.y+ 1 <= c && a[e.x ][e.y+1] > a[e.x][e.y])
51.          {
```

```
52.             if (maxval < step[e.x][e.y+1])maxval =
    step[e.x][e.y+1];
53.             hasroad = true;
54.         }
55.         if(hasroad) step[e.x][e.y] = maxval+1;
56.     }
57.     ll ans = 1;
58.     for (ll i = 1; i <= r; ++i)
59.     {
60.         for (ll j = 1; j <= c; ++j)
61.         {
62.             if (ans < step[i][j])
63.             {
64.                 ans = step[i][j];
65.             }
66.         }
67.     }
68.     cout << ans;
69.     return 0;
70. }
```

P1443 马的遍历

```
01. #include <iostream>
02. #include <algorithm>
03. #include <cstdio>
04. #include <cstring>
05. #include <queue>
06. using namespace std;
07. struct node {
08.     int x, y,bushu;
09. };
10. int n, m, x, y;
11. int chessboard[401][401],visited[401][401];
12. void Bfs(int x, int y)
13. {
14.     chessboard[x][y] = 0;
15.     visited[x][y] = 1;
16.     queue<node> q;
17.     q.push({x,y,0});
18.     while (!q.empty())
19.     {
20.         node t = q.front();
```

```
21.         q.pop();
22.         //八个方向
23.         if (t.x + 2 <= n && t.y + 1 <= m && visited[t.x +
    2][t.y + 1]==0)//右下左
24.         {
25.             chessboard[t.x + 2][t.y + 1] = t.bushu + 1;
26.             visited[t.x + 2][t.y + 1] = 1;
27.             q.push({t.x+2,t.y+1,chessboard[t.x + 2][t.y +
    1]});
28.         }
29.         if (t.x + 1 <= n && t.y + 2 <= m && visited[t.x +
    1][t.y + 2] == 0)//右下右
30.         {
31.             chessboard[t.x + 1][t.y + 2] = t.bushu + 1;
32.             visited[t.x + 1][t.y + 2] = 1;
33.             q.push({ t.x + 1,t.y + 2,chessboard[t.x +
    1][t.y + 2] });
34.         }
35.         if (t.x - 2 >= 1 && t.y + 1 <= m && visited[t.x -
    2][t.y + 1] == 0)//右上左
36.         {
37.             chessboard[t.x - 2][t.y + 1] = t.bushu + 1;
38.             visited[t.x - 2][t.y + 1] = 1;
39.             q.push({ t.x - 2,t.y + 1,chessboard[t.x -
    2][t.y + 1] });
40.         }
41.         if (t.x -1 >= 1 && t.y +2 <= m && visited[t.x -
    1][t.y + 2] == 0)//右上右
42.         {
43.             chessboard[t.x -1][t.y + 2] = t.bushu + 1;
44.             visited[t.x -1][t.y + 2] = 1;
45.             q.push({ t.x - 1,t.y + 2,chessboard[t.x -
    1][t.y + 2] });
46.         }
47.         if (t.x + 2 <= n && t.y - 1 >=1 && visited[t.x +
    2][t.y - 1] == 0)//左下右
48.         {
49.             chessboard[t.x + 2][t.y - 1] = t.bushu + 1;
50.             visited[t.x + 2][t.y - 1] = 1;
51.             q.push({ t.x + 2,t.y - 1,chessboard[t.x +
    2][t.y - 1] });
52.         }
53.         if (t.x + 1 <= n && t.y - 2 >= 1 && visited[t.x +
    1][t.y - 2] == 0)//左下左
```

```
54.         {
55.             chessboard[t.x + 1][t.y - 2] = t.bushu + 1;
56.             visited[t.x + 1][t.y - 2] = 1;
57.             q.push({ t.x + 1,t.y -2,chessboard[t.x + 1][t.y
    - 2] });
58.         }
59.         if (t.x -1 >= 1 && t.y -2 >= 1 && visited[t.x -
    1][t.y - 2] == 0)//左上左
60.         {
61.             chessboard[t.x - 1][t.y - 2] = t.bushu + 1;
62.             visited[t.x - 1][t.y - 2] = 1;
63.             q.push({ t.x -1,t.y -2,chessboard[t.x - 1][t.y
    - 2] });
64.         }
65.         if (t.x - 2 >= 1 && t.y - 1 >= 1 && visited[t.x -
    2][t.y - 1] == 0)//左上右
66.         {
67.             chessboard[t.x - 2][t.y - 1] = t.bushu + 1;
68.             visited[t.x - 2][t.y - 1] = 1;
69.             q.push({ t.x - 2,t.y - 1,chessboard[t.x -
    2][t.y - 1] });
70.         }
71.     }
72. }
73. int main() {
74.     cin >> n >> m >> x >> y;
75.     Bfs(x, y);
76.     for (int i = 1; i <= n; ++i)
77.     {
78.         for (int j = 1; j <= m; ++j)
79.         {
80.             if(visited[i][j]==0)
81.             {
82.                 chessboard[i][j] = -1;
83.          }
84.             cout << chessboard[i][j];
85.             if (j != m)
86.             {
87.                 cout << ' ';
88.             }
89.             else
90.             {
91.                 cout << endl;
92.             }
```

```
93.          }
94.      }
95.      return 0;
96.  }
```

P1111 修复公路

```
01.  #include <iostream>
02.  #include <algorithm>
03.  using namespace std;
04.  int n, m,mintime,lastsinglenum;
05.  int fa[1005];
06.  struct node {
07.      int x, y, t;
08.  }a[100005];
09.  bool cmp(const node& a, const node& b)
10.  {
11.      return a.t < b.t;
12.  }
13.  int findfather(int root)//迭代查找父亲
14.  {
15.      int t = root;
16.      while (t != fa[t]) t = fa[t];
17.      while (root != fa[root])
18.      {
19.          int a = root;
20.          root = fa[root];
21.          fa[a] = t;
22.      }
23.      return t;
24.  }
25.  void handle(int x, int y)
26.  {
27.      int xfather = findfather(x);
28.      int yfather = findfather(y);
29.      if (xfather != yfather)
30.      {
31.          fa[yfather] = xfather;
32.          --lastsinglenum;
33.      }
34.  }
35.  int main()
36.  {
```

```
37.         cin >> n >> m;
38.         lastsinglenum = n - 1;
39.         for (int i = 0; i < m; ++i)
40.         {
41.             cin >> a[i].x >> a[i].y >> a[i].t;
42.         }
43.         sort(a, a+m,cmp);
44.         for (int i = 0; i <= n; ++i)
45.         {
46.             fa[i] = i;
47.         }
48.         for (int i = 0; i < m; ++i)
49.         {
50.             node& nowroad = a[i];
51.             handle(nowroad.x, nowroad.y);
52.             if (lastsinglenum == 0)
53.             {
54.                 cout << nowroad.t;
55.                 return 0;
56.             }
57.         }
58.         cout << -1;
59.         return 0;
60.     }
```