第 5 周

Day21

1. PERKET

> **题目来源：** 洛谷
>
> **题目链接：** P2036 [COCI 2008/2009 #2] PERKET

【题目描述】

> Perket 是一种流行的美食。为了做好 Perket，厨师必须谨慎选择食材，以在保持传统风味的同时尽可能获得最全面的味道。你有 n 种可支配的配料。对于每一种配料，我们知道它们各自的酸度 s 和苦度 b 。当我们添加配料时，总的酸度为每一种配料的酸度总乘积；总的苦度为每一种配料的苦度的总和。
>
> 众所周知，美食应该做到口感适中，所以我们希望选取配料，以使得酸度和苦度的绝对差最小。
>
> 另外，我们必须添加至少一种配料，因为没有任何食物以水为配料的。

【输入描述】

> 第一行一个整数 n ，表示可供选用的食材种类数。
>
> 接下来 n 行，每行 2 个整数 s_i 和 b_i，表示第 i 种食材的酸度和苦度。
>
> 对于 100% 的数据，有 $1 \leq n \leq 10$，且将所有可用食材全部使用产生的总酸度和总苦度小于 1×10^9 ，酸度和苦度不同时为 1 和 0。

【输出描述】

> 一行一个整数，表示可能的总酸度和总苦度的最小绝对差。

【示例一】

> 输入：
>
> ```
> 1
> 3 10
> ```
>
> 输出：
>
> ```
> 7
> ```

【示例二】

> 输入：

```
2
3 8
5 8
```

输出：

```
1
```

【示例三】

输入：

```
4
1 7
2 6
3 8
4 9
```

输出：

```
1
```

【解法】

解法：枚举。

- 枚举出所有的选择情况即可。
- 可以用 dfs，也可以用二进制枚举。

【参考代码】

```cpp
#include <iostream>
#include <cmath>

using namespace std;

typedef long long LL;

const int N = 15;

LL n;
LL s[N], b[N];

int main()
{
    cin >> n;
```

```cpp
    for(int i = 0; i < n; i++) cin >> s[i] >> b[i];

    LL ret = 1e19;
    for(int st = 1; st < (1 << n); st++)
    {
        LL x = 1, y = 0;
        for(int i = 0; i < n; i++)
        {
            if((st >> i) & 1)
            {
                x *= s[i];
                y += b[i];
            }
        }
        ret = min(ret, abs(x - y));
    }

    cout << ret << endl;

    return 0;
}
```

2. 营救

> **题目来源：** 洛谷
>
> **题目链接：** P1396 营救

【题目描述】

> 妈妈下班回家，街坊邻居说小明被一群陌生人强行押上了警车！妈妈丰富的经验告诉她小明被带到了 t 区，而自己在 s 区。
>
> 该市有 m 条大道连接 n 个区，一条大道将两个区相连接，每个大道有一个拥挤度。小明的妈妈虽然很着急，但是不愿意拥挤的人潮冲乱了她优雅的步伐。所以请你帮她规划一条从 s 至 t 的路线，使得经过道路的拥挤度最大值最小。

【输入描述】

> 第一行有四个用空格隔开的 n，m，s，t，其含义见【题目描述】。
>
> 接下来 m 行，每行三个整数 u,v,w，表示有一条大道连接区 u 和区 v，且拥挤度为 w。
>
> **两个区之间可能存在多条大道。**

【输出描述】

> 输出一行一个整数，代表最大的拥挤度。

【示例一】

> 输入：
>
> 3 3 1 3
>
> 1 2 2
>
> 2 3 1
>
> 1 3 3
>
> 输出：
>
> 2

【解法】

解法：贪心(kk 算法) + 并查集。

- 从小到大选择每一条边，用并查集维护选择边之后的集合情况；
- 当 s 和 t 在同一个集合中时，说明选择的边可以形成通路。最后一次选择的边权就是结果。

【参考代码】

代码块

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1e4 + 10, M = 2e4 + 10;

int n, m, s, t;
struct node
{
    int u, v, w;
}e[M];

bool cmp(node& x, node& y)
{
    return x.w < y.w;
}

int fa[N];

int find(int x)
{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
```

```
}

void un(int x, int y)
{
    fa[find(x)] = find(y);
}

int main()
{
    cin >> n >> m >> s >> t;
    for(int i = 1; i <= m; i++) cin >> e[i].u >> e[i].v >> e[i].w;

    sort(e + 1, e + 1 + m, cmp);

    for(int i = 1; i <= n; i++) fa[i] = i;

    int ret = e[m].w;
    for(int i = 1; i <= m; i++)
    {
        int u = e[i].u, v = e[i].v, w = e[i].w;
        un(u, v);
        ret = w;

        if(find(s) == find(t)) break;
    }

    cout << ret << endl;

    return 0;
}
```


3. School Photo

> **题目来源：** 洛谷
>
> **题目链接：** P10710 [NOISG 2024 Prelim] School Photo

【题目描述】

> Zane 是 NOI 学校的校长。NOI 学校有 n 个班，每个班有 s 名同学。第 i 个班中的第 j 名同学的身高是 $a_{i,j}$。
>
> 现在 Zane 想从每个班上选出一名同学拍照，使得这 n 名同学中最高的同学和最低的同学的身高差最小。
>
> 请你输出这个最小值。

【输入描述】

> 第一行，两个整数 *n*,*s*；
>
> 接下来 *n* 行，每行 *s* 个整数，表示 *a*。

【输出描述】

> 一行一个整数表示答案。

【示例一】

> 输入：
>
> ```
> 2 3
> 2 1 8
> 5 4 7
> ```
>
> 输出：
>
> ```
> 1
> ```

【解法】

解法：双指针。

- 将所有人的身高与班级绑定，从小到大排序；
- 问题就变成在一个数组中，挑出连续的一段，使里面的学生包含所有班级。在所有的挑法中，找出最大身高 - 最小身高的最小值。
- 可以用滑动窗口来解决。

【参考代码】

代码块

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010, M = N * N;

int n, s, m;
struct node
{
    int h, id;
}a[M];

int cnt[N];
```

```cpp

bool cmp(node& x, node& y)
{
    return x.h < y.h;
}

int main()
{
    cin >> n >> s;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= s; j++)
        {
            m++;
            cin >> a[m].h;
            a[m].id = i;
        }
    }

    sort(a + 1, a + 1 + m, cmp);

    int ret = 1e9;
    for(int l = 1, r = 1, kind = 0; r <= m; r++)
    {
        // a[r]
        cnt[a[r].id]++;
        // 0 -> 1
        if(cnt[a[r].id] == 1) kind++;

        while(kind == n)
        {
            ret = min(ret, a[r].h - a[l].h);

            cnt[a[l].id]--;
            // 1 -> 0
            if(cnt[a[l].id] == 0) kind--;
            l++;
        }
    }

    cout << ret << endl;

    return 0;
}
```

4. 方格取数

> **题目来源：** 洛谷
>
> **题目链接：** P7074 [CSP-J2020] 方格取数

【题目描述】

> 设有 $n \times m$ 的方格图，每个方格中都有一个整数。现有一只小熊，想从图的左上角走到右下角，每一步只能向上、向下或向右走一格，并且不能重复经过已经走过的方格，也不能走出边界。小熊会取走所有经过的方格中的整数，求它能取到的整数之和的最大值。

【输入描述】

> 第一行有两个整数 n,m。
>
> 接下来 n 行每行 m 个整数，依次代表每个方格中的整数。

【输出描述】

> 一个整数，表示小熊能取到的整数之和的最大值。

【示例一】

> 输入：
>
> ```
> 3 4
> 1 -1 3 2
> 2 -1 4 -1
> -2 2 -3 -1
> ```
>
> 输出：
>
> ```
> 9
> ```

【解法】

解法：动态规划 - 路径类dp。

与之前的行走方式不同，这里多了一种往上走的方式。如果用之前的分析方式做题，会出现后效性。原因就是当前这个格子的值即会依赖上面，也会依赖下面，不是一个有向无环图。

可以通过定义两个状态表示，消除后效性。

1. 状态表示：

`f[i][j]` 表示：从上边或者左边走到 `[i, j]` 位置时，最大和；

`g[i][j]` 表示：从下边或者左边走到 `[i, j]` 位置时，最大和。

此时，每一种状态表示都是无后效性的。最终结果就在 `f[n][m]` 中。

2. 状态转移方程：

对于 `f[i][j]`：

- 从上边来：`f[i - 1][j]`；
- 从左边来：`f[i][j - 1]` 以及 `g[i][j - 1]`；

要的是三种情况的最大值。

对于 `g[i][j]`：

- 从下边来：`g[i + 1][j]`；
- 从左边来：`f[i][j - 1]` 以及 `g[i][j - 1]`；

也是三者的最大值。

3. 初始化：

两张表走初始化为负无穷，`f[1][0] = 0` 表示起点位置。

4. 填表顺序：

必须要一列一列的填！

因此，先从左往右填每一列：

- `f` 表从上往下每一行，
- `g` 表从下往上每一行。

【参考代码】

代码块

```cpp
#include <iostream>
#include <cstring>

using namespace std;

typedef long long LL;

const int N = 1010;

int n, m;
int a[N][N];
```

```cpp

LL f[N][N], g[N][N];

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            cin >> a[i][j];

    memset(f, -0x3f, sizeof f);
    memset(g, -0x3f, sizeof g);
    f[0][1] = 0;

    for(int j = 1; j <= m; j++)
    {
        // 先考虑 f 表
        for(int i = 1; i <= n; i++)
        {
            f[i][j] = max(f[i - 1][j], max(f[i][j - 1], g[i][j - 1])) + a[i][j];
        }

        // 填 g 表
        for(int i = n; i >= 1; i--)
        {
            g[i][j] = max(g[i + 1][j], max(f[i][j - 1], g[i][j - 1])) + a[i][j];
        }
    }

    cout << f[n][m] << endl;

    return 0;
}
```


Day22

1. 迷宫

> **题目来源：** 洛谷
>
> **题目链接：** P1605 迷宫

【题目描述】

> 给定一个 $N \times M$ 方格的迷宫，迷宫里有 T 处障碍，障碍处不可通过。
>
> 在迷宫中移动有上下左右四种方式，每次只能移动一个方格。数据保证起点上没有障碍。
>
> 给定起点坐标和终点坐标，每个方格最多经过一次，问有多少种从起点坐标到终点坐标的方案。

【输入描述】

> 第一行为三个正整数 N, M, T ，分别表示迷宫的长宽和障碍总数。
>
> 第二行为四个正整数 SX, SY, FX, FY ， SX, SY 代表起点坐标， FX, FY 代表终点坐标。
>
> 接下来 T 行，每行两个正整数，表示障碍点的坐标。
>
> 对于 100% 的数据， $1 \leq N, M \leq 5$，$1 \leq T \leq 10$，$1 \leq SX, FX \leq n$，$1 \leq SY, FY \leq m$

【输出描述】

> 输出从起点坐标到终点坐标的方案总数。

【示例一】

> 输入：
>
> ```
> 2 2 1
> 1 1 2 2
> 1 2
> ```
>
> 输出：
>
> ```
> 1
> ```

【解法】

解法：暴搜。

- 从起点开始搜索，把所有能走的路径都搜一遍即可。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 10;

int n, m, t;
int x1, y1, x2, y2;
bool e[N][N]; // 标记障碍物

```

```cpp
int ret;
bool st[N][N]; // 标记已经走过的位置

int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};

void dfs(int i, int j)
{
    if(i == x2 && j == y2)
    {
        ret++;
        return;
    }

    st[i][j] = true;
    for(int k = 0; k < 4; k++)
    {
        int x = i + dx[k], y = j + dy[k];
        if(x < 1 || x > n || y < 1 || y > m || e[x][y] || st[x][y]) continue;

        dfs(x, y);
    }
    // 恢复现场
    st[i][j] = false;
}

int main()
{
    cin >> n >> m >> t >> x1 >> y1 >> x2 >> y2;
    while(t--)
    {
        int a, b; cin >> a >> b;
        e[a][b] = true;
    }

    dfs(x1, y1);

    cout << ret << endl;

    return 0;
}
```


2. Teamwork

> **题目来源：** 洛谷
>
> **题目链接：** P5124 [USACO18DEC] Teamwork G

【题目描述】

> 在 Farmer John 最喜欢的节日里，他想要给他的朋友们赠送一些礼物。由于他并不擅长包装礼物，他想要获得他的奶牛们的帮助。你可能能够想到，奶牛们本身也不是很擅长包装礼物，而 Farmer John 即将得到这一教训。
>
> Farmer John 的 N 头奶牛（$1 \leqslant N \leqslant 104$）排成一行，方便起见依次编号为 $1 \cdots N$。奶牛 i 的包装礼物的技能水平为 si。她们的技能水平可能参差不齐，所以 FJ 决定把她的奶牛们分成小组。每一组可以包含任意不超过 K 头的连续的奶牛（$1 \leqslant K \leqslant 103$），并且一头奶牛不能属于多于一个小组。由于奶牛们会互相学习，这一组中每一头奶牛的技能水平会变成这一组中水平最高的奶牛的技能水平。
>
> 请帮助 FJ 求出，在他合理地安排分组的情况下，可以达到的技能水平之和的最大值。

【输入描述】

> 输入的第一行包含 N 和 K。以下 N 行按照 N 头奶牛的排列顺序依次给出她们的技能水平。技能水平是一个不超过 105 的正整数。

【输出描述】

> 输出 FJ 通过将连续的奶牛进行分组可以达到的最大技能水平和。

【示例一】

> 输入：
>
> ```
> 7 3
> 1
> 15
> 7
> 9
> 2
> 5
> 10
> ```
>
> 输出：
>
> ```
> 84
> ```

【解法】

解法：动态规划 - 线性 dp。

1. 状态表示：

`f[i]` 表示：只考虑前 `i` 个数，可以达到的最大技能水平和。

2. 状态转移方程：

对于 `i` 前的一个位置 `j`，其中 `(0 <= j && i - j <= k)`，那么最大技能水平和为：

- `f[j] + max(a[j + 1]...a[i]) × (i - j)`；
- 也就是只考虑前 `j` 个数的最大技能水平和以及 `[j + 1, i]` 区间的最大和。

在 `j` 变化的过程中，取最大值即可。

其中，`[j + 1, i]` 区间的最大值，可以在填表的过程中同时维护出来。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

const int N = 1e4 + 10;

int n, k;
int a[N];
int f[N];

int main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> a[i];

    for(int i = 1; i <= n; i++)
    {
        int t = a[i];
        for(int j = i - 1; j >= 0 && i - j <= k; j--)
        {
            f[i] = max(f[i], f[j] + t * (i - j));
            t = max(t, a[j]);
        }
    }

    cout << f[n] << endl;

    return 0;
```

```
29    }
```

3. Convention

> **题目来源：** 洛谷
>
> **题目链接：** P5119 [USACO18DEC] Convention S

【题目描述】

> 一场别开生面的牛吃草大会就要在 Farmer John 的农场举办了！
>
> 世界各地的奶牛将会到达当地的机场，前来参会并且吃草。具体地说，有 N 头奶牛到达了机场（$1 \leqslant N \leqslant 105$），其中奶牛 i 在时间 ti（$0 \leqslant ti \leqslant 109$）到达。Farmer John 安排了 M（$1 \leqslant M \leqslant 105$）辆大巴来机场接这些奶牛。每辆大巴可以乘坐 C 头奶牛（$1 \leqslant C \leqslant N$）。Farmer John 正在机场等待奶牛们到来，并且准备安排到达的奶牛们乘坐大巴。当最后一头乘坐某辆大巴的奶牛到达的时候，这辆大巴就可以发车了。Farmer John 想要做一个优秀的主办者，所以并不想让奶牛们在机场等待过长的时间。如果 Farmer John 合理地协调这些大巴，等待时间最长的奶牛等待的时间的最小值是多少？一头奶牛的等待时间等于她的到达时间与她乘坐的大巴的发车时间之差。
>
> 输入保证 $MC \geqslant N$。

【输入描述】

> 输入的第一行包含三个空格分隔的整数 N，M，和 C。第二行包含 N 个空格分隔的整数，表示每头奶牛到达的时间。

【输出描述】

> 输出一行，包含所有到达的奶牛中的最大等待时间的最小值。

【示例一】

> 输入：
>
> ```
> 6 3 2
> 1 1 10 14 4 3
> ```
>
> 输出：
>
> ```
> 4
> ```

【解法】

解法：二分答案。

二段性：

- 当等待时间增长的时候，所用的车辆在减少；
- 当等待时间减少的时候，所用的车辆在增加。

【参考代码】

代码块

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

int n, m, c;
int a[N];

int calc(int x)
{
    int cnt = 0;
    int l = 1, r = 1;
    while(r <= n)
    {
        while(r <= n && a[r] - a[l] <= x && r - l + 1 <= c) r++;
        cnt++;
        l = r;
    }
    return cnt;
}

int main()
{
    cin >> n >> m >> c;
    for(int i = 1; i <= n; i++) cin >> a[i];

    sort(a + 1, a + 1 + n);

    int l = 0, r = a[n] - a[1];
    while(l < r)
    {
        int mid = (l + r) / 2;
        if(calc(mid) <= m) r = mid;
        else l = mid + 1;
    }

    cout << l << endl;

    return 0;
}
```

4. Convention II

> **题目来源：** 洛谷
>
> **题目链接：** P5120 [USACO18DEC] Convention II S

【题目描述】

> 虽然在接机上耽误了挺长时间，Farmer John 为吃草爱好牛们举行的大会至今为止都非常顺利。大会吸引了世界各地的奶牛。
>
> 然而大会的重头戏看起来却给 Farmer John 带来了一些新的安排上的困扰。他的农场上的一块非常小的牧草地出产一种据某些识货的奶牛说是世界上最美味的品种的草。因此，所有参会的 N 头奶牛（$1 \leqslant N \leqslant 10^5$）都想要品尝一下这种草。由于这块牧草地小到仅能容纳一头奶牛，这很有可能会导致排起长龙。
>
> Farmer John 知道每头奶牛i计划到达这块特殊的牧草地的时间 ai，以及当轮到她时，她计划品尝这种草花费的时间 ti。当奶牛 i 开始吃草时，她会在离开前花费全部 ti 的时间，此时其他到达的奶牛需要排队等候。如果这块牧草地空出来的时候多头奶牛同时在等候，那么资历最深的奶牛将会是下一头品尝鲜草的奶牛。在这里，恰好在另一头奶牛吃完草离开时到达的奶牛被认为是“在等待的”。类似地，如果当没有奶牛在吃草的时候有多头奶牛同时到达，那么资历最深的奶牛是下一头吃草的奶牛。
>
> 请帮助 FJ 计算所有奶牛中在队伍里等待的时间（ai 到这头奶牛开始吃草之间的时间）的最大值。

【输入描述】

> 输入的第一行包含 N。以下 N 行按资历顺序给出了 N 头奶牛的信息（资历最深的奶牛排在最前面）。每行包含一头奶牛的 ai 和 ti。所有的 ti 为不超过 10^4 的正整数，所有 ai 为不超过 10^9 的正整数。

【输出描述】

> 输出所有奶牛中的最长等待时间。

【示例一】

> 输入：
>
> ```
> 5
> 25 3
> 105 30
> 20 50
> 10 17
> 100 10
> ```

> 输出：
>
> 10

【解法】

解法：用堆模拟整个流程。

- 用堆维护正在排队以及还未吃完草的奶牛，模拟整个吃草的过程。

【参考代码】

代码块

```cpp
#include <iostream>
#include <queue>
#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

int n;
struct node
{
    int id, a, t;

    // 按照 id 创建小根堆
    bool operator < (const node& x) const
    {
        return id > x.id;
    }

}e[N];

priority_queue<node> heap;

bool cmp(node& x, node& y)
{
    return x.a < y.a;
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        cin >> e[i].a >> e[i].t;
```

```
35          e[i].id = i;
36      }
37
38      sort(e + 1, e + 1 + n, cmp);
39
40      int i = 1, end = 0, ret = 0;
41      while(i <= n || heap.size())
42      {
43          while(i <= n && e[i].a <= end) heap.push(e[i++]);
44          if(heap.empty()) heap.push(e[i++]);
45
46          auto c = heap.top(); heap.pop();
47
48          if(c.a >= end) // 无需等待
49          {
50              end = c.a + c.t;
51          }
52          else
53          {
54              ret = max(ret, end - c.a);
55              end += c.t;
56          }
57      }
58
59      cout << ret << endl;
60
61      return 0;
62  }
```


Day23

1. 单词方阵

> **题目来源：** 洛谷
>
> **题目链接：** P1101 单词方阵

【题目描述】

> 给一 $n \times n$ 的字母方阵，内可能蕴含多个 `yizhong` 单词。单词在方阵中是沿着同一方向连续摆放的。摆放可沿着 8 个方向的任一方向，同一单词摆放时不再改变方向，单词与单词之间可以交叉，因此有可能共用字母。输出时，将不是单词的字母用 `*` 代替，以突出显示单词。

【输入描述】

> 第一行输入一个数 n。($7 \leq n \leq 100$)。

第二行开始输入 $n \times n$ 的字母矩阵。

【输出描述】

突出显示单词的 $n \times n$ 矩阵。

【示例一】

输入：

```
7
aaaaaaa
aaaaaaa
aaaaaaa
aaaaaaa
aaaaaaa
aaaaaaa
aaaaaaa
```

输出：

```
*******
*******
*******
*******
*******
*******
*******
```

【示例二】

输入：

```
8
qyizhong
gydthkjy
nwidghji
orbzsfgz
hhgrhwth
zzzzzozo
iwdfrgng
```

> yyyygggg
>
> 输出：
>
> *yizhong
>
> gy******
>
> n*i*****
>
> o**z****
>
> h*h*
>
> z****o**
>
> i*****n*
>
> y******g

【解法】

解法：暴力枚举。

- 找到每一个 y 字符位置，从八个方向尝试找出 yizhong 字符。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 110;

int n;
char a[N][N];
string s = "yizhong";

bool st[N][N];

int dx[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dy[] = {1, -1, 0, 0, 1, -1, 1, -1};

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        cin >> (a[i] + 1);

    // for(int i = 1; i <= n; i++)
```

```
23      // {
24      //     for(int j = 1; j <= n; j++)
25      //         cout << a[i][j];
26      //     cout << endl;
27      // }
28
29      for(int i = 1; i <= n; i++)
30          for(int j = 1; j <= n; j++)
31              if(a[i][j] == 'y')
32              {
33                  for(int k = 0; k < 8; k++)
34                  {
35                      int x = i, y = j;
36                      int cnt = 0;
37                      while(x >= 1 && x <= n && y >= 1 && y <= n && cnt < 7 &&
    a[x][y] == s[cnt])
38                      {
39                          cnt++;
40                          x = x + dx[k];
41                          y = y + dy[k];
42                      }
43
44                      if(cnt == 7) // 走成功
45                      {
46                          x = i, y = j;
47                          while(cnt--)
48                          {
49                              st[x][y] = true;
50                              x = x + dx[k];
51                              y = y + dy[k];
52                          }
53                      }
54                  }
55              }
56
57      for(int i = 1; i <= n; i++)
58      {
59          for(int j = 1; j <= n; j++)
60          {
61              if(st[i][j]) cout << a[i][j];
62              else cout << '*';
63          }
64          cout << endl;
65      }
66
67
68      return 0;
```

```
69    }
```

2. Watering the Fields

> **题目来源：** 洛谷
>
> **题目链接：** P2212 [USACO14MAR] Watering the Fields S

【题目描述】

> 给定 n 个点，第 i 个点的坐标为 (xi,yi)，如果想连通第 i 个点与第 j 个点，需要耗费的代价为两点的距离。第 i 个点与第 j 个点之间的距离使用欧几里得距离的平方进行计算，即：
>
> $$(xi-xj)2+(yi-yj)2$$
>
> 我们规定耗费代价小于 c 的两点无法连通，求使得每两点都能连通下的最小代价，如果无法连通输出 `-1`。

【输入描述】

> 第一行两个整数 n,c 代表点数与想要连通代价不能少于的一个数。
> 接下来 n 行每行两个整数 xi,yi 描述第 i 个点。

【输出描述】

> 一行一个整数代表使得每两点都能连通下的最小代价，如果无法连通输出 `-1`。

【示例一】

> 输入：
>
> ```
> 3 11
> 0 2
> 5 0
> 4 3
> ```
>
> 输出：
>
> ```
> 46
> ```

【解法】

解法：最小生成树。

- 按照题意建图，然后跑一遍最小生成树算法即可。

【参考代码】

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 2010, M = 2e6 + 10;

int n, c;
int x[N], y[N];

int m;
struct node
{
    int u, v, w;
}e[M];

int fa[N];

bool cmp(node& x, node& y)
{
    return x.w < y.w;
}

int find(int x)
{
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

int calc(int i, int j)
{
    return (x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]);
}

int main()
{
    cin >> n >> c;
    for(int i = 1; i <= n; i++) cin >> x[i] >> y[i];

    // 建图
    for(int i = 1; i <= n; i++)
        for(int j = i + 1; j <= n; j++)
        {
            int t = calc(i, j);
            if(t >= c)
            {
                m++;
                e[m] = {i, j, t};
```

```
                // e[m].u = i;
                // e[m].v = j;
                // e[m].w = t;
            }
        }

    for(int i = 1; i <= n; i++) fa[i] = i;

    sort(e + 1, e + 1 + m, cmp);

    int ret = 0, cnt = 0;
    for(int i = 1; i <= m; i++)
    {
        int u = e[i].u, v = e[i].v, w = e[i].w;
        int fu = find(u), fv = find(v);
        if(fu == fv) continue;

        cnt++;
        ret += w;
        fa[fu] = fv;

        if(cnt == n - 1) break;
    }

    if(cnt == n - 1) cout << ret << endl;
    else cout << -1 << endl;

    return 0;
}
```

3. Radio Contact

> **题目来源：** 洛谷
>
> **题目链接：** P3133 [USACO16JAN] Radio Contact G

【题目描述】

> FJ 失去了他最喜欢的牛铃，而 Bessie 已经同意帮助他找到它！他们用不同的路径搜索农场，通过无线电保持联系。
>
> 不幸的是，无线电中的电池电量不足，所以他们设法尽可能保持两者位置的距离最小，以节省电量。
>
> FJ 从位置(*fx*,*fy*) 开始，并计划遵循由 *N* 步组成的路径.Bessie 从位置 (*bx*,*by*) 开始，并遵循由 *M* 步组成的路径。每个步骤都是 `N`（北），`E`（东），`S`（南），或 `W`（西）。其中，东方向为 *x* 轴正方向，北方向为 *y* 轴正方向。两个路径可以经过相同的点。

> 在每个时间段，FJ 可以不移动，也可以沿着他的道路前进一步。无论哪个方向恰好在下一个（假设他还没有到达他的路径的最后位置）。Bessie 可以做出类似的选择。
>
> 在每个时间点（不包括从初始位置开始的第一步），他们的无线电消耗的能量等于它们之间距离的平方。
>
> 请帮助 FJ 和 Bessie 计划行动策略，使双方达到各自终点时，最大限度地减少消耗的能量总量。输出所消耗的最小的能量。

【输入描述】

> 第一行两个整数 N 和 M $(1 \leqslant N,M \leqslant 1000)$。
>
> 第二行两个整数 fx 和 fy。
>
> 第三行两个整数 bx 和 by $(0 \leqslant fx,fy,bx,by \leqslant 1000)$。
>
> 下一行为一个长度为 N 的字符串，描述 FJ 的路径。
>
> 最后一行为一个长度 M 的字符串，描述 Bessie 的路径。

【输出描述】

> 共一行一个整数，表示最小能量。

【示例一】

> 输入：
>
> ```
> 2 7
> 3 0
> 5 0
> NN
> NWWWWWN
> ```
>
> 输出：
>
> ```
> 28
> ```

【解法】

解法：动态规划。

1. 状态表示：

`f[i][j]` 表示：FJ 走了 `i` 步，B 走了 `j` 步，此时的最小能量。

2. 状态转移方程：

- FJ 动，B 动：`f[i - 1][j - 1] + dist[i, j]`；

- FJ 动，B 不动：`f[i - 1][j] + dist[i, j]`；
- FJ 不动，B 动：`f[i][j - 1] + dist[i, j]`；
- FJ 不动，B 不动：一定不是最优情况。

其中，`dist[i, j]` 表示 FJ 走了 `i` 步，B 走了 `j` 步后的距离。仅需预处理出来走了若干步之后的坐标即可。

3. 初始化：

- 第一行表示 FJ 一直不动，B 一直在动：`f[0][j] = f[0][j - 1] + dist[0, j]`
- 第一列表示 FJ 一直在动，B 一直不动：`f[i][0] = f[i - 1][0] + dist[i, 0]`

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

const int N = 1010;

int n, m;
struct node
{
    int x, y;
}a[N], b[N];

int f[N][N];

int dist(int i, int j)
{
    return (a[i].x - b[j].x) * (a[i].x - b[j].x) + (a[i].y - b[j].y) * (a[i].y - b[j].y);
}

int main()
{
    cin >> n >> m;
    cin >> a[0].x >> a[0].y >> b[0].x >> b[0].y;

    for(int i = 1; i <= n; i++)
    {
```

```
27          char ch; cin >> ch;
28          if(ch == 'N') a[i] = {a[i - 1].x, a[i - 1].y + 1};
29          else if(ch == 'S') a[i] = {a[i - 1].x, a[i - 1].y - 1};
30          else if(ch == 'W') a[i] = {a[i - 1].x - 1, a[i - 1].y};
31          else a[i] = {a[i - 1].x + 1, a[i - 1].y};
32      }
33
34      for(int i = 1; i <= m; i++)
35      {
36          char ch; cin >> ch;
37          if(ch == 'N') b[i] = {b[i - 1].x, b[i - 1].y + 1};
38          else if(ch == 'S') b[i] = {b[i - 1].x, b[i - 1].y - 1};
39          else if(ch == 'W') b[i] = {b[i - 1].x - 1, b[i - 1].y};
40          else b[i] = {b[i - 1].x + 1, b[i - 1].y};
41      }
42
43      // 初始化
44      for(int j = 1; j <= m; j++) f[0][j] = f[0][j - 1] + dist(0, j);
45      for(int i = 1; i <= n; i++) f[i][0] = f[i - 1][0] + dist(i, 0);
46
47      for(int i = 1; i <= n; i++)
48          for(int j = 1; j <= m; j++)
49              f[i][j] = min(f[i - 1][j - 1], min(f[i - 1][j], f[i][j - 1])) +
    dist(i, j);
50
51      cout << f[n][m] << endl;
52
53      return 0;
54  }
```

4. 单词背诵

> **题目来源：** 洛谷
>
> **题目链接：** P1381 单词背诵

【题目描述】

> 灵梦有 n 个单词想要背，但她想通过一篇文章中的一段来记住这些单词。
>
> 文章由 m 个单词构成，她想在文章中找出连续的一段，其中包含最多的她想要背的单词（重复的只算一个）。并且在背诵的单词量尽量多的情况下，还要使选出的文章段落尽量短，这样她就可以用尽量短的时间学习尽可能多的单词了。

【输入描述】

> 第 1 行一个数 n，接下来 n 行每行是一个长度不超过 10 的字符串，表示一个要背的单词。

> 接着是一个数 m，然后是 m 行长度不超过 10 的字符串，每个表示文章中的一个单词。
>
> - 对于 100% 的数据， $n \leq 1000, m \leq 10^5$ 。

【输出描述】

> 输出文件共 2 行。第 1 行为文章中最多包含的要背的单词数，第 2 行表示在文章中包含最多要背单词的最短的连续段的长度。

【示例一】

> 输入：
>
> 3
>
> hot
>
> dog
>
> milk
>
> 5
>
> hot
>
> dog
>
> dog
>
> milk
>
> hot
>
> 输出：
>
> 3
>
> 3

【解法】

同《逛画展》类似，只不过这次我们处理的对象是「字符串」，但是我们并不关心每一个字符串长什么样子，只用关心你在文章中「出现了没有」，以及「出现了几次」。因此我们在滑动窗口之前可以先「预处理」一下所有的字符串，把需要背的字符串标记成 $[1, n]$ 之间的数，比如： hot -> 1 ， dog -> 2 ， $milk$ -> 3 ；把文章中需要背的「单词」改成「相应的数」存起来，不需要背的单词搞一个「 $n+1$ 」或者「 -1 」等「无效的值」存起来。这样，我们就把处理「字符串」数组，改成了处理「数字」数组，方便了许多。

这道题在「滑动窗口」的过程中需要注意三点：

1. 要先统计一下文章中需要背的字符串的「种类」有多少个，方便我们后续滑动窗口求「最短长度」；

2. 滑动窗口的过程中会遇到「无效」的字符串，直接跳过即可；
3. 有可能文章中并「没有需要背」的单词，此时应该输出 $0, 0$。

【参考代码】

代码块

```cpp
#include <iostream>
#include <unordered_set>
#include <unordered_map>

using namespace std;

const int N = 1e5 + 10;

int n, m;
unordered_set<string> mp;
unordered_set<string> tmp;

string s[N];

int kind;
unordered_map<string, int> cnt;

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        string s; cin >> s;
        mp.insert(s);
    }

    cin >> m;
    for(int i = 1; i <= m; i++)
    {
        cin >> s[i];
        if(mp.count(s[i])) tmp.insert(s[i]);
    }

    cout << tmp.size() << endl;
    if(tmp.empty())
    {
        cout << 0 << endl;
        return 0;
    }
```

```
40
41      int ret = N;
42      for(int l = 1, r = 1; r <= m; r++)
43      {
44          if(!tmp.count(s[r])) continue;
45
46          cnt[s[r]]++;
47          if(cnt[s[r]] == 1) kind++;
48
49          while(kind == tmp.size())
50          {
51              ret = min(ret, r - l + 1);
52              cnt[s[l]]--;
53              if(cnt[s[l]] == 0) kind--;
54              l++;
55          }
56      }
57
58      cout << ret << endl;
59
60      return 0;
61  }
```

Day24

1. 自然数的拆分问题

> **题目来源：** 洛谷
>
> **题目链接：** P2404 自然数的拆分问题

【题目描述】

> 任何一个大于 1 的自然数 n，总可以拆分成若干个小于 n 的自然数之和。现在给你一个自然数 n，要求你求出 n 的拆分成一些数字的和。每个拆分后的序列中的数字从小到大排序。然后你需要输出这些序列，其中字典序小的序列需要优先输出。

【输入描述】

> 输入：待拆分的自然数 n。
>
> 数据保证，$2 \leq n \leq 8$。

【输出描述】

> 输出：若干数的加法式子。

【示例一】

> 输入：
>
> 7
>
> 输出：
>
> 1+1+1+1+1+1+1
>
> 1+1+1+1+1+2
>
> 1+1+1+1+3
>
> 1+1+1+2+2
>
> 1+1+1+4
>
> 1+1+2+3
>
> 1+1+5
>
> 1+2+2+2
>
> 1+2+4
>
> 1+3+3
>
> 1+6
>
> 2+2+3
>
> 2+5
>
> 3+4

【解法】

解法：暴搜。

- 用 dfs 枚举所有方案即可。

【参考代码】

```
#include <iostream>

using namespace std;

const int N = 10;

int n;
int sum;
int path[N];

```

```cpp
void dfs(int pos)
{
    if(sum > n) return;
    if(sum == n)
    {
        if(pos == 2) return;
        for(int i = 1; i < pos - 1; i++)
            cout << path[i] << "+";
        cout << path[pos - 1] << endl;

        return;
    }

    for(int i = path[pos - 1]; i <= n; i++)
    {
        sum += i;
        path[pos] = i;
        dfs(pos + 1);
        path[pos] = 0;
        sum -= i;
    }
}

int main()
{
    cin >> n;

    path[0] = 1;
    dfs(1);

    return 0;
}
```


2. The Great Revegetation

> **题目来源：** 洛谷
>
> **题目链接：** P1694 [USACO19FEB] The Great Revegetation B

【题目描述】

> 长时间的干旱使得 Farmer John 的 N 块草地上牧草匮乏。随着雨季即将到来，现在应当是重新种植的时候了。
>
> 在 Farmer John 的储物棚里有四个桶，每个桶里装着一种不同的草种。他想要在每块草地上播种其中一种草。作为一名奶农，Farmer John 想要确保他的每头奶牛都能得到丰富的食谱。他的 M 头奶

> 牛每一头都有两块喜爱的草地，他想要确保这两块草地种植不同种类的草，从而每头奶牛都可以有两种草可供选择。Farmer John 知道没有一块草地受到多于 3 头奶牛的喜爱。
>
> 请帮助 Farmer John 选择每块草地所种的草的种类，使得所有奶牛的营养需求都得到满足。

【输入描述】

> 输入的第一行包含 $2 \leqslant N \leqslant 100$ 和 $1 \leqslant M \leqslant 150$。以下 M 行，每行包含两个范围为 $1 \cdots N$ 的整数，为 Farmer John 的一头奶牛喜欢的两块草地。

【输出描述】

> 输出一个 N 位数，每一位均为 1…4 之一，表示每一块草地上所种的草的种类。第一位对应草地 1 的草的种类，第二位对应草地 2，以此类推。如果有多种可行的解，只需输出所有解中最小的 N 位数。

【示例一】

> 输入：
>
> 5 6
>
> 4 1
>
> 4 2
>
> 4 3
>
> 2 5
>
> 1 2
>
> 1 5
>
> 输出：
>
> 12133

【解法】

解法：暴力枚举即可。

- 根据相互作用关系建图，有边的两个结点的颜色不能相同。
- 从第一个位置开始从最小的颜色开始枚举，只要合法就去考虑下一个位置，直到所有位置都考虑完毕。

【参考代码】

代码块

```cpp
#include <iostream>
#include <vector>

using namespace std;

```

```cpp
const int N = 200;

int n, m;
vector<int> edges[N];

int ret[N];

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        int a, b; cin >> a >> b;
        edges[a].push_back(b);
        edges[b].push_back(a);
    }

    for(int a = 1; a <= n; a++)
    {
        for(int i = 1; i <= 4; i++)
        {
            bool flag = true;
            for(auto b : edges[a])
            {
                if(ret[b] == i)
                {
                    flag = false;
                    break;
                }
            }

            if(flag)
            {
                ret[a] = i;
                break;
            }
        }
    }

    for(int i = 1; i <= n; i++) cout << ret[i];

    return 0;
}
```

3. 香甜的黄油

> **题目来源：** 洛谷
>
> **题目链接：** P1828 [USACO3.2] 香甜的黄油 Sweet Butter

【题目描述】

> Farmer John 发现了做出全威斯康辛州最甜的黄油的方法：糖。
>
> 把糖放在一片牧场上，他知道 N 只奶牛会过来舔它，这样就能做出能卖好价钱的超甜黄油。当然，他将付出额外的费用在奶牛上。
>
> Farmer John 很狡猾。像以前的 Pavlov，他知道他可以训练这些奶牛，让它们在听到铃声时去一个特定的牧场。他打算将糖放在那里然后下午发出铃声，以至他可以在晚上挤奶。
>
> Farmer John 知道每只奶牛都在各自喜欢的牧场（一个牧场不一定只有一头牛）。给出各头牛在的牧场和牧场间的路线，找出使所有牛到达的路程和最短的牧场（他将把糖放在那）。

【输入描述】

> 第一行包含三个整数 N,P,C，分别表示奶牛数、牧场数和牧场间道路数。
>
> 第二行到第 $N+1$ 行，每行一个整数，其中第 i 行的整数表示第 $i-1$ 头奶牛所在的牧场号。
>
> 第 $N+2$ 行到第 $N+C+1$ 行，每行包含三个整数 A,B,D，表示牧场号为 A 和 B 的两个牧场之间有一条长度为 D 的双向道路相连。

【输出描述】

> 输出一行一个整数，表示奶牛必须行走的最小的距离和。

【示例一】

> 输入：
>
> ```
> 3 4 5
> 2
> 3
> 4
> 1 2 1
> 1 3 5
> 2 3 7
> 2 4 3
> 3 4 5
> ```
>
> 输出：
>
> ```
> 8
> ```

【解法】

解法：最短路。

- 以所有的牧场为起点，跑一次 heap - dijkstra 算法，统计出到达每一头奶牛的最短距离；
- 在所有的情况下取最小值。

【参考代码】

代码块

```cpp
#include <iostream>
#include <queue>
#include <cstring>
#include <vector>

using namespace std;

typedef pair<int, int> PII;

const int N = 810, INF = 0x3f3f3f3f;

int n, p, c;
int id[N];
vector<PII> edges[N];

int dist[N];
bool st[N];

int dijkstra(int s)
{
    memset(dist, 0x3f, sizeof dist);
    memset(st, 0, sizeof st);

    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, s});
    dist[s] = 0;

    while(heap.size())
    {
        auto t = heap.top(); heap.pop();
        int u = t.second;

        if(st[u]) continue;
        st[u] = true;

        // 松弛
```

```cpp
            for(auto& t : edges[u])
            {
                int v = t.first, w = t.second;
                if(dist[u] + w < dist[v])
                {
                    dist[v] = dist[u] + w;
                    heap.push({dist[v], v});
                }
            }
        }

        int sum = 0;
        for(int i = 1; i <= n; i++)
        {
            // id[i]
            if(dist[id[i]] == INF) return INF;
            sum += dist[id[i]];
        }

        return sum;
    }

    int main()
    {
        cin >> n >> p >> c;
        for(int i = 1; i <= n; i++) cin >> id[i];
        for(int i = 1; i <= c; i++)
        {
            int u, v, w; cin >> u >> v >> w;
            edges[u].push_back({v, w});
            edges[v].push_back({u, w});
        }

        int ret = INF;
        for(int i = 1; i <= p; i++)
        {
            ret = min(ret, dijkstra(i));
        }

        cout << ret << endl;

        return 0;
    }
```

4. 饥饿的奶牛

> **题目来源：** 洛谷
>
> **题目链接：** P1868 饥饿的奶牛

【题目描述】

> 有一条奶牛冲出了围栏，来到了一处圣地（对于奶牛来说），上面用牛语写着一段文字。
>
> 现用汉语翻译为：
>
> 有 N 个区间，每个区间 x,y 表示提供的 $x\sim y$ 共 $y-x+1$ 堆优质牧草。你可以选择任意区间但不能有重复的部分。
>
> 对于奶牛来说，自然是吃的越多越好，然而奶牛智商有限，现在请你帮助他。

【输入描述】

> 第一行一个整数 N。
>
> 接下来 N 行，每行两个数 x,y，描述一个区间。
>
> $1\le n\le 1.5\times 10^5, 0\le x\le y\le 3\times 10^6$。

【输出描述】

> 输出最多能吃到的牧草堆数。

【示例一】

> 输入：
>
> ```
> 3
> 1 3
> 7 8
> 3 4
> ```
>
> 输出：
>
> ```
> 5
> ```

【解法】

解法：动态规划 - 线性 dp + 预处理 + 二分优化。

贪心策略是不对的，因为我们如果在从前往后选择的过程中，先选一个区间长度较大的，后续的选择有可能不是最优的，比如 $[1,4]$、$[1,2]$、$[3,5]$。

正解应该是排序 + 动态规划。

为了方便考虑 i 位置的区间能放在谁后面，我们将所有区间按照右端点从小到大排序。这样对于 i 位置的区间，只能放在 i 位置之前的某个区间后面。动态规划的过程中，填表就不会混乱。

1. 状态表示：

   $f[i]$ 表示：最后一个区间必定选 i 区间，此时能选择出来的最长区间的长度。

   那么整个 f 表中的最大值就是结果。

2. 状态转移方程：

   对于 i 位置的区间，设 $j\ (1 \le j < i-1)$，只要 $a[j].y < a[i].x$，那么 i 位置的区间就能放在 j 位置的区间后面，此时 $f[i] = f[j] + a[i].y - a[i].x + 1$，然后取所有情况的最大值即可。但是，数据范围告诉我们，这样是会超时的，要去优化一下状态转移方程。

   在第二层循环中，我们要找的是能接在谁后面，并且要达到长度最大。因为 i 位置之前的 y 坐标都是升序的，所以 i 位置能接的区间一定是连续的一段，更新 $f[i]$，其实就是要的是这一段里面 f 的最大值。

   于是可以再创建一个 $g[i]$ 数组，用来记录 $[1, i]$ 区间内所有 f 的最大值。在更新 $f[i]$ 的值时，先二分能接的区间的右端点 r，然后用 $g[r] + a[i].y - a[i].x + 1$ 更新 $f[i]$。就能把第二层循环优化成 log 级别。

   注意二分时候的边界情况，因为有可能这个区间之前不存在右端点比它的左端点小的区间。

3. 初始化：

   不用初始化。

4. 填表顺序：

   从左往右。

【参考代码】

代码块

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 2e5 + 10;
```

```cpp

int n;
struct node
{
    int x, y;
}a[N];

int f[N], g[N];

bool cmp(node& a, node& b)
{
    return a.y < b.y;
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i].x >> a[i].y;

    sort(a + 1, a + 1 + n, cmp);

    for(int i = 1; i <= n; i++)
    {
        // 二分出 j 位置
        int l = 0, r = i - 1;
        while(l < r)
        {
            int mid = (l + r + 1) / 2;
            if(a[mid].y < a[i].x) l = mid;
            else r = mid - 1;
        }

        f[i] = g[l] + a[i].y - a[i].x + 1;
        g[i] = max(g[i - 1], f[i]);
    }

    cout << g[n] << endl;

    return 0;
}
```


Day25

1. 玩具谜题

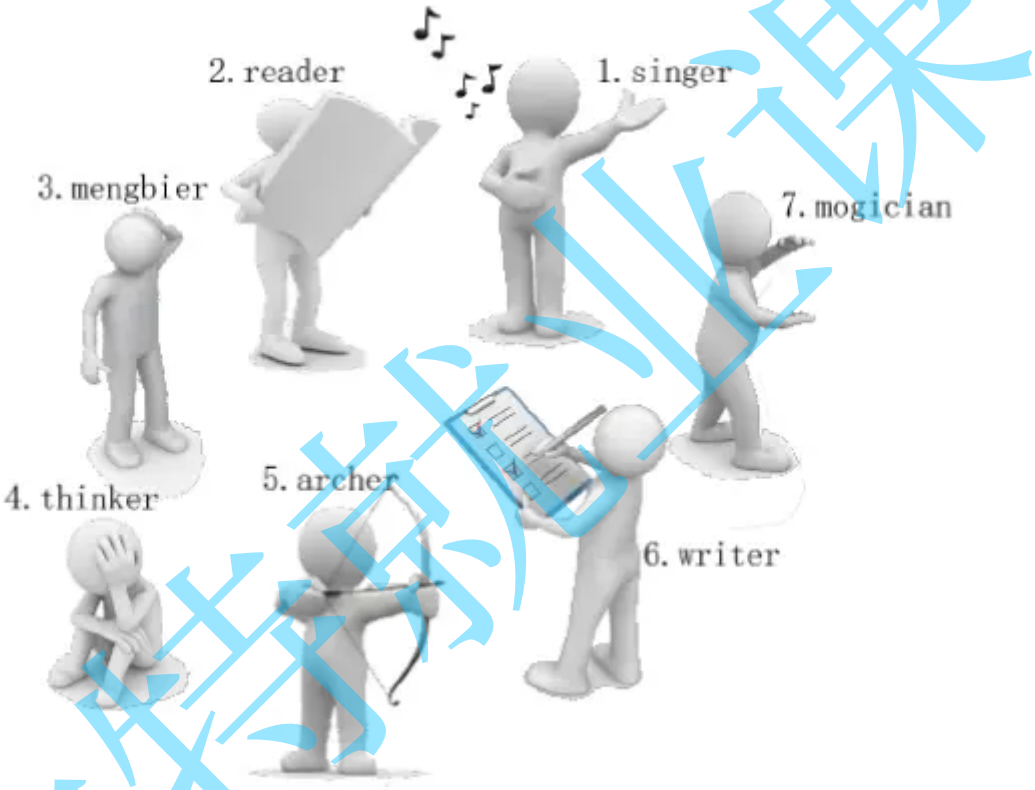
题目来源： 洛谷

题目链接： P1563 [NOIP 2016 提高组] 玩具谜题

【题目描述】

小南有一套可爱的玩具小人，它们各有不同的职业。

有一天，这些玩具小人把小南的眼镜藏了起来。小南发现玩具小人们围成了一个圈，它们有的面朝圈内，有的面朝圈外。如下图：



这时 singer 告诉小南一个谜题：“眼镜藏在我左数第 3 个玩具小人的右数第 1 个玩具小人的左数第 2 个玩具小人那里。”

小南发现，这个谜题中玩具小人的朝向非常关键，因为朝内和朝外的玩具小人的左右方向是相反的：面朝圈内的玩具小人，它的左边是顺时针方向，右边是逆时针方向；而面向圈外的玩具小人，它的左边是逆时针方向，右边是顺时针方向。

小南一边艰难地辨认着玩具小人，一边数着：

singer 朝内，左数第 3 个是 archer。

archer 朝外，右数第 1 个是 thinker。

thinker 朝外，左数第 2 个是 writer。

所以眼镜藏在 writer 这里！

虽然成功找回了眼镜，但小南并没有放心。如果下次有更多的玩具小人藏他的眼镜，或是谜题的长度更长，他可能就无法找到眼镜了。所以小南希望你写程序帮他解决类似的谜题。这样的谜题具体可以描述为：

有 n 个玩具小人围成一圈，已知它们的职业和朝向。现在第 1 个玩具小人告诉小南一个包含 m 条指令的谜题，其中第 z 条指令形如“向左数/右数第 s 个玩具小人”。你需要输出依次数完这些指令后，到达的玩具小人的职业。

【输入描述】

输入的第一行包含两个正整数 n,m，表示玩具小人的个数和指令的条数。

接下来 n 行，每行包含一个整数和一个字符串，以逆时针为顺序给出每个玩具小人的朝向和职业。其中 0 表示朝向圈内，1 表示朝向圈外。保证不会出现其他的数。字符串长度不超过 10 且仅由英文字母构成，字符串不为空，并且字符串两两不同。整数和字符串之间用一个空格隔开。

接下来 m 行，其中第 i 行包含两个整数 ai,si，表示第 i 条指令。若 ai=0，表示向左数 si 个人；若 ai=1，表示向右数 si 个人。 保证 ai 不会出现其他的数，$1 \leqslant si < n$。

【输出描述】

输出一个字符串，表示从第一个读入的小人开始，依次数完 m 条指令后到达的小人的职业。

【示例一】

输入：

```
7 3
0 singer
0 reader
0 mengbier
1 thinker
1 archer
0 writer
1 mogician
0 3
1 1
0 2
```

输出：

```
writer
```

【解法】

解法：模拟。

- 根据题意模拟即可。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

const int N = 1e5 + 10;

int n, m;
struct node
{
    int p;
    string s;
}a[N];

int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++) cin >> a[i].p >> a[i].s;

    int pos = 0;
    while(m--)
    {
        int x, y; cin >> x >> y;
        if(x == a[pos].p) pos = (pos - y + n) % n;
        else pos = (pos + y + n) % n;
    }

    cout << a[pos].s << endl;

    return 0;
}
```


2. 寻找道路

> **题目来源：** 洛谷
>
> **题目链接：** P2296 [NOIP 2014 提高组] 寻找道路

【题目描述】

> 在有向图 G 中，每条边的长度均为 1，现给定起点和终点，请你在图中找一条从起点到终点的路径，该路径满足以下条件：
>
> 1. 路径上的所有点的出边所指向的点都直接或间接与终点连通。
> 2. 在满足条件 1 的情况下使路径最短。
>
> 注意：图 G 中可能存在重边和自环，题目保证终点没有出边。
>
> 请你输出符合条件的路径的长度。

【输入描述】

> 第一行有两个用一个空格隔开的整数 n 和 m，表示图有 n 个点和 m 条边。
>
> 接下来的 m 行每行 2 个整数 x,y，之间用一个空格隔开，表示有一条边从点 x 指向点 y。
>
> 最后一行有两个用一个空格隔开的整数 s,t，表示起点为 s，终点为 t。

【输出描述】

> 输出只有一行，包含一个整数，表示满足题目描述的最短路径的长度。如果这样的路径不存在，输出 -1。

【示例一】

> 输入：
>
> ```
> 3 2
> 1 2
> 2 1
> 1 3
> ```
>
> 输出：
>
> ```
> -1
> ```

【解法】

解法：反图 + bfs 求最短路。

- 在反图上，从终点开始跑一次 dfs/bfs，标记一下能到达的点。
- 然后在 bfs 的过程中，只去更新那些所有出边都被标记过的点。

【参考代码】

代码块

```cpp
#include <iostream>
#include <queue>
#include <algorithm>
#include <cstring>
```

```cpp
#include <vector>

using namespace std;

const int N = 1e4 + 10, INF = 0x3f3f3f3f;

int n, m, s, t;
vector<int> e1[N]; // 原图
vector<int> e2[N]; // 反图

bool st[N]; // 标记终点能到达的点

void dfs(int u)
{
    st[u] = true;
    for(auto v : e2[u])
    {
        if(!st[v]) dfs(v);
    }
}

int dist[N];

bool check(int u)
{
    if(!st[u]) return false;

    for(auto v : e1[u])
    {
        if(!st[v]) return false;
    }
    return true;
}

int bfs(int s)
{
    if(!check(s)) return -1;

    memset(dist, 0x3f, sizeof dist);
    queue<int> q;
    q.push(s);
    dist[s] = 0;

    while(q.size())
    {
        auto u = q.front(); q.pop();

```

```cpp
        for(auto v : e1[u])
        {
            if(!check(v) || dist[v] != INF) continue;

            dist[v] = dist[u] + 1;
            q.push(v);
        }
    }

    if(dist[t] == INF) return -1;
    return dist[t];
}

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        int x, y; cin >> x >> y;
        e1[x].push_back(y);
        e2[y].push_back(x);
    }
    cin >> s >> t;

    // 跑反图
    dfs(t);

    // bfs 求最短路
    cout << bfs(s) << endl;

    return 0;
}
```


3. 插入排序

> **题目来源：** 洛谷
>
> **题目链接：** P7910 [CSP-J 2021] 插入排序

【题目描述】

> 插入排序是一种非常常见且简单的排序算法。小 Z 是一名大一的新生，今天 H 老师刚刚在上课的时候讲了插入排序算法。
>
> 假设比较两个元素的时间为 $O(1)$，则插入排序可以以 $O(n^2)$ 的时间复杂度完成长度为 n 的数组的排序。不妨假设这 n 个数字分别存储在 $a_1, a_2, \ldots, a_n$ 之中，则如下伪代码给出了插入排序算

法的一种最简单的实现方式：

这下面是 C/C++ 的示范代码：

```
for (int i = 1; i <= n; i++)
    for (int j = i; j >= 2; j--)
        if (a[j] < a[j-1]) {
            int t = a[j-1];
            a[j-1] = a[j];
            a[j] = t;
        }
```

这下面是 Pascal 的示范代码：

```
for i:=1 to n do
    for j:=i downto 2 do
        if a[j]<a[j-1] then
            begin
                t:=a[i];
                a[i]:=a[j];
                a[j]:=t;
            end;
```

为了帮助小 Z 更好的理解插入排序，小 Z 的老师 H 老师留下了这么一道家庭作业：

H 老师给了一个长度为 n 的数组 a，数组下标从 1 开始，并且数组中的所有元素均为非负整数。小 Z 需要支持在数组 a 上的 Q 次操作，操作共两种，参数分别如下：

$1\ x\ v$：这是第一种操作，会将 a 的第 x 个元素，也就是 a_x 的值，修改为 v。保证 $1 \le x \le n$，$1 \le v \le 10^9$。**注意这种操作会改变数组的元素，修改得到的数组会被保留，也会影响后续的操作。**

$2\ x$：这是第二种操作，假设 H 老师按照**上面的伪代码**对 a 数组进行排序，你需要告诉 H 老师原来 a 的第 x 个元素，也就是 a_x，在排序后的新数组所处的位置。保证 $1 \le x \le n$。**注意这种操作不会改变数组的元素，排序后的数组不会被保留，也不会影响后续的操作。**

H 老师不喜欢过多的修改，所以他保证类型 1 的操作次数不超过 5000。

小 Z 没有学过计算机竞赛，因此小 Z 并不会做这道题。他找到了你来帮助他解决这个问题。

【输入描述】

第一行，包含两个正整数 n, Q，表示数组长度和操作次数。

第二行，包含 n 个空格分隔的非负整数，其中第 i 个非负整数表示 a_i。

接下来 Q 行，每行 2~3 个正整数，表示一次操作，操作格式见**【题目描述】**。

对于所有测试数据，满足 $1 \leq n \leq 8000$，$1 \leq Q \leq 2 \times 10^5$，$1 \leq x \leq n$，$1 \leq v, a_i \leq 10^9$ 。

对于所有测试数据，保证在所有 Q 次操作中，至多有 5000 次操作属于类型一。

【输出描述】

对于每一次类型为 2 的询问，输出一行一个正整数表示答案。

【示例一】

输入：

```
3 4
3 2 1
2 3
1 3 2
2 2
2 3
```

输出：

```
1
1
2
```

说明：

在修改操作之前，假设 H 老师进行了一次插入排序，则原序列的三个元素在排序结束后所处的位置分别是 3,2,1。

在修改操作之后，假设 H 老师进行了一次插入排序，则原序列的三个元素在排序结束后所处的位置分别是 3,1,2。

注意虽然此时 $a_2 = a_3$，但是我们**不能将其视为相同的元素**。

【解法】

解法：模拟 + 排序。

先将整个数组排序：

- 对于 1 操作，修改对应位置之后，仅会让当前元素不合法，执行一次冒泡过程就可让整个数组有序；
- 对于 2 操作，维护一个数组，标记原数组排完序之后的位置。

【参考代码】

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 8010;

int n, Q;
struct node
{
    int x, id;
}a[N];

int mp[N]; // 维护排完序之后的下标

// 维护排序的稳定性
bool cmp(node& x, node& y)
{
    if(x.x == y.x) return x.id < y.id;
    else return x.x < y.x;
}

// 维护排序之后，之前的元素下标在哪
void get_id()
{
    for(int i = 1; i <= n; i++)
    {
        // a[i].x     a[i].id
        mp[a[i].id] = i;
    }
}

// 仅需一趟冒泡
void bubble_sort()
{
    // 先从左往右
    for(int i = 2; i <= n; i++)
        if(a[i].x < a[i - 1].x || (a[i].x == a[i - 1].x && a[i].id < a[i - 1].id))
            swap(a[i], a[i - 1]);

    // 再从右往左
    for(int i = n; i >= 2; i--)
        if(a[i].x < a[i - 1].x || (a[i].x == a[i - 1].x && a[i].id < a[i - 1].id))
            swap(a[i], a[i - 1]);
}
```

```cpp

int main()
{
    cin >> n >> Q;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i].x;
        a[i].id = i;
    }

    sort(a + 1, a + 1 + n, cmp);
    get_id();

    while(Q--)
    {
        int op, i, v;
        cin >> op >> i;
        if(op == 1)
        {
            cin >> v;
            // i -> mp[i]
            a[mp[i]].x = v;
            bubble_sort();
            get_id();
        }
        else
        {
            cout << mp[i] << endl;
        }
    }

    return 0;
}
```


4. 飞扬的小鸟

> **题目来源：** 洛谷
>
> **题目链接：** P1941 [NOIP 2014 提高组] 飞扬的小鸟

【题目描述】

> Flappy Bird 是一款风靡一时的休闲手机游戏。玩家需要不断控制点击手机屏幕的频率来调节小鸟的飞行高度，让小鸟顺利通过画面右方的管道缝隙。如果小鸟一不小心撞到了水管或者掉在地上的话，便宣告失败。

为了简化问题，我们对游戏规则进行了简化和改编:

游戏界面是一个长为 n，高为 m 的二维平面，其中有 k 个管道（忽略管道的宽度）。

小鸟始终在游戏界面内移动。小鸟从游戏界面最左边任意整数高度位置出发，到达游戏界面最右边时，游戏完成。

小鸟每个单位时间沿横坐标方向右移的距离为 1，竖直移动的距离由玩家控制。如果点击屏幕，小鸟就会上升一定高度 x，每个单位时间可以点击多次，效果叠加；如果不点击屏幕，小鸟就会下降一定高度 y。小鸟位于横坐标方向不同位置时，上升的高度 x 和下降的高度 y 可能互不相同。

小鸟高度等于 0 或者小鸟碰到管道时，游戏失败。小鸟高度为 m 时，无法再上升。

现在,请你判断是否可以完成游戏。如果可以，输出最少点击屏幕数；否则，输出小鸟最多可以通过多少个管道缝隙。

【输入描述】

第 1 行有 3 个整数 n,m,k，分别表示游戏界面的长度，高度和水管的数量，每两个整数之间用一个空格隔开；

接下来的 n 行，每行 2 个用一个空格隔开的整数 x 和 y，依次表示在横坐标位置 0~$n-1$ 上玩家点击屏幕后，小鸟在下一位置上升的高度 x，以及在这个位置上玩家不点击屏幕时，小鸟在下一位置下降的高度 y。

接下来 k 行，每行 3 个整数 p,l,h，每两个整数之间用一个空格隔开。每行表示一个管道，其中 p 表示管道的横坐标，l 表示此管道缝隙的下边沿高度，h 表示管道缝隙上边沿的高度（输入数据保证 p 各不相同，但不保证按照大小顺序给出）。

【输出描述】

共两行。

第一行，包含一个整数，如果可以成功完成游戏，则输出 1，否则输出 0。

第二行，包含一个整数，如果第一行为 1，则输出成功完成游戏需要最少点击屏幕数，否则，输出小鸟最多可以通过多少个管道缝隙。

【示例一】

输入：

```
10 10 6
3 9
9 9
1 2
1 3
1 2
1 1
```

```
2 1
2 1
1 6
2 2
1 2 7
5 1 5
6 3 5
7 5 8
8 7 9
9 1 3
```

输出：

```
1
6
```

【解法】

解法：动态规划 - 完全背包。

核心思路：先更新状态，然后处理管道。

1. 状态表示：

`f[i][j]` 表示：到达第 `i` 个位置时，高度为 `j`，此时的最小点击次数。

2. 状态转移方程：

- 在 `i - 1` 位置选择点击：
  a. 点击 `1` 次：`f[i - 1][j - x[i - 1]] + 1`；
  b. 点击 `2` 次：`f[i - 1][j - x[i - 1] * 2] + 2`；
  c. 以此类推

  用完全背包做优化：`f[i][j] = min(f[i - 1][j - x[i - 1]], f[i][j - x[i - 1]]) + 1`。
- 在 `i - 1` 位置不点击：`f[i][j] = min(f[i][j], f[i - 1][j + y[i - 1]])`。

两大细节：

- 先更新点击的情况，再更新不点击的情况：

因为点击的状态转移方程会用到当前这一行的值。如果先更新不点击的情况，这一行的状态就包含先下降再上升的情况。

- 天花板需要特殊处理：

  天花板不仅会被 `m - x[i - 1]` 的位置更新，也会被 `m - x[i - 1] + 1` 更新。

  也就是 `[m - x[i - 1], m]` 区间的高度更新。

3. 初始化：

   先把整个格子初始化为无穷，然后把第一列初始化为 `0`，表示起始位置。

4. 填表顺序：

   按照背包问题的方式，正常填表。

5. 最终结果：

- 如果结果存在，最后一行里面更新最小值；
- 如果结果不存在，在所有 `f` 表中，判断能通过的管道数量。

【参考代码】

代码块

```cpp
#include <iostream>
#include <cstring>

using namespace std;

const int N = 10010, M = 1010, INF = 0x3f3f3f3f;

int n, m, k;
int x[N], y[N];

// 管道信息
bool st[N];
int l[N], h[N];

int f[N][M];

int main()
{
    cin >> n >> m >> k;
```

```
20      for(int i = 0; i < n; i++) cin >> x[i] >> y[i];
21      // 初始化管道
22      for(int i = 1; i <= n; i++)
23      {
24          l[i] = 0;
25          h[i] = m + 1;
26      }
27      for(int i = 1; i <= k; i++)
28      {
29          int p; cin >> p;
30          cin >> l[p] >> h[p];
31          st[p] = true;
32      }
33
34      memset(f, 0x3f, sizeof f);
35      // 初始化
36      for(int j = 1; j <= m; j++) f[0][j] = 0;
37
38      for(int i = 1; i <= n; i++)
39      {
40          // 点击
41          for(int j = x[i - 1] + 1; j <= m; j++)
42          {
43              f[i][j] = min(f[i][j], f[i - 1][j - x[i - 1]] + 1);
44              f[i][j] = min(f[i][j], f[i][j - x[i - 1]] + 1);
45          }
46
47          // 特殊处理天花板
48          for(int k = m - x[i - 1]; k <= m; k++)
49          {
50              f[i][m] = min(f[i][m], f[i][k] + 1);
51              f[i][m] = min(f[i][m], f[i - 1][k] + 1);
52              // f[i][m] = min(f[i][k] + 1, f[i - 1][k] + 1);
53          }
54
55          // 不点击
56          for(int j = 1; j + y[i - 1] <= m; j++)
57              f[i][j] = min(f[i][j], f[i - 1][j + y[i - 1]]);
58
59          // 处理管道
60          for(int j = 0; j <= l[i]; j++) f[i][j] = INF;
61          for(int j = h[i]; j <= m; j++) f[i][j] = INF;
62      }
63
64      int ret = INF;
65      for(int j = 1; j <= m; j++) ret = min(ret, f[n][j]);
66
```

```
67      if(ret != INF) cout << 1 << endl << ret << endl;
68      else
69      {
70          cout << 0 << endl;
71          int cnt = 0;
72          for(int i = 1; i <= n; i++)
73              for(int j = 1; j <= m; j++)
74              {
75                  if(st[i] && f[i][j] != INF)
76                  {
77                      cnt++;
78                      break;
79                  }
80              }
81          cout << cnt << endl;
82      }
83
84      return 0;
85  }
```