双指针、BFS与图论

一、AcWing 1238. 日志统计

【题目描述】

小明维护着一个程序员论坛。现在他收集了一份“点赞”日志，日志共有N行。

其中每一行的格式是：

```
ts id
```

表示在ts时刻编号id的帖子收到一个“赞”。

现在小明想统计有哪些帖子曾经是“热帖”。

如果一个帖子曾在任意一个长度为D的时间段内收到不少于K个赞，小明就认为这个帖子曾是“热帖”。

具体来说，如果存在某个时刻T满足该帖在$[T, T+D)$这段时间内（注意是左闭右开区间）收到不少于K个赞，该帖就曾是“热帖”。

给定日志，请你帮助小明统计出所有曾是“热帖”的帖子编号。

【输入格式】

第一行包含三个整数N, D, K。

以下N行每行一条日志，包含两个整数ts和id。

【输出格式】

按从小到大的顺序输出热帖id。

每个id占一行。

【数据范围】

$1 \le K \le N \le 10^5$

$0 \le ts, id \le 10^5$

$1 \le D \le 10000$

【输入样例】

```
7 10 2
0 1
0 10
10 10
10 1
9 1
100 3
100 3
```

【输出样例】

```
1
3
```

【分析】

我们先将所有的日志按时间排好序，然后从前往后遍历每一份日志i。日志j表示在日志i之前的且与日志i的时间差距小于D的最早的日志，即循环每一份日志$op[i]$时，我们先将$op[i].id$的点赞数量加一，然后判断日志j的时间是否已经在区间外，即是否有$op[i].time - op[j].time \geq D$，如果已经超出时间范围，那么将$op[j].id$的点赞数减一，然后指针$j$向后移，直到$op[j]$的时间在范围内。然后判断$op[i].id$的点赞数是否已经大于等于$K$，如果满足那么标记一下这个$id$是热贴，最后遍历每个$id$输出所有热贴即可。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define X first
#define Y second
using namespace std;

const int N = 100010;
typedef pair<int, int> PII;
PII op[N];
bool st[N];
int cnt[N];
int n, d, k;

int main()
{
```

```
    cin >> n >> d >> k;
    for (int i = 0; i < n; i++) cin >> op[i].X >> op[i].Y;
    sort(op, op + n);
    for (int i = 0, j = 0; i < n; i++)
    {
        cnt[op[i].Y]++;
        while (op[i].X - op[j].X >= d)
        {
            cnt[op[j].Y]--;
            j++;
        }
        if (cnt[op[i].Y] >= k) st[op[i].Y] = true;
    }
    for (int i = 0; i <= 100000; i++)
        if (st[i]) cout << i << endl;
    return 0;
}
```

二、AcWing 1101. 献给阿尔吉侬的花束

【题目描述】

阿尔吉侬是一只聪明又慵懒的小白鼠，它最擅长的就是走各种各样的迷宫。

今天它要挑战一个非常大的迷宫，研究员们为了鼓励阿尔吉侬尽快到达终点，就在终点放了一块阿尔吉侬最喜欢的奶酪。

现在研究员们想知道，如果阿尔吉侬足够聪明，它最少需要多少时间就能吃到奶酪。

迷宫用一个$R \times C$的字符矩阵来表示。

字符`S`表示阿尔吉侬所在的位置，字符`E`表示奶酪所在的位置，字符`#`表示墙壁，字符`.`表示可以通行。

阿尔吉侬在1个单位时间内可以从当前的位置走到它上下左右四个方向上的任意一个位置，但不能走出地图边界。

【输入格式】

第一行是一个正整数T，表示一共有T组数据。

每一组数据的第一行包含了两个用空格分开的正整数R和C，表示地图是一个$R \times C$的矩阵。

接下来的R行描述了地图的具体内容，每一行包含了C个字符。字符含义如题目描述中所述。保证有且仅有一个S和E。

【输出格式】

对于每一组数据，输出阿尔吉侬吃到奶酪的最少单位时间。

若阿尔吉侬无法吃到奶酪，则输出`oop!`。

每组数据的输出结果占一行。

【数据范围】

$1 < T \leq 10$

$2 \leq R, C \leq 200$

【输入样例】

```
3
3 4
.S..
###.
..E.
3 4
.S..
.E..
....
3 4
.S..
####
..E.
```

【输出样例】

```
5
1
oop!
```

【分析】

BFS裸题，直接看代码~

【代码】

```
#include <iostream>

#include <cstring>
```

```cpp
#include <algorithm>
#include <queue>
using namespace std;

typedef pair<int, int> PII;
const int N = 210;
char g[N][N];
int dis[N][N];
int n, m, sx, sy;
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };

int bfs(int sx, int sy)
{
    memset(dis, -1, sizeof dis);
    queue<PII> Q;
    Q.push({ sx, sy });
    dis[sx][sy] = 0;
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        int x = t.first, y = t.second;
        for (int i = 0; i < 4; i++)
        {
            int nx = x + dx[i], ny = y + dy[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && g[nx][ny] != '#' && !~dis[nx][ny])
            {
                dis[nx][ny] = dis[x][y] + 1, Q.push({ nx, ny });
                if (g[nx][ny] == 'E') return dis[nx][ny];
            }
        }
    }
    return -1;
}

int main()
{
    int T;
    cin >> T;
    while (T--)
    {

        cin >> n >> m;
```

```
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (cin >> g[i][j], g[i][j] == 'S') sx = i, sy = j;
        int res = bfs(sx, sy);
        if (!~res) puts("oop!");
        else cout << res << endl;
    }
    return 0;
}
```

三、AcWing 1113. 红与黑

【题目描述】

有一间长方形的房子，地上铺了红色、黑色两种颜色的正方形瓷砖。

你站在其中一块黑色的瓷砖上，只能向相邻（上下左右四个方向）的黑色瓷砖移动。

请写一个程序，计算你总共能够到达多少块黑色的瓷砖。

【输入格式】

输入包括多个数据集合。

每个数据集合的第一行是两个整数W和H，分别表示x方向和y方向瓷砖的数量。

在接下来的H行中，每行包括W个字符。每个字符表示一块瓷砖的颜色，规则如下：

1. `.`：黑色的瓷砖；
2. `#`：红色的瓷砖；
3. `@`：黑色的瓷砖，并且你站在这块瓷砖上。该字符在每个数据集合中唯一出现一次。

当在一行中读入的是两个零时，表示输入结束。

【输出格式】

对每个数据集合，分别输出一行，显示你从初始位置出发能到达的瓷砖数（记数时包括初始位置的瓷砖）。

【数据范围】

$1 \leq W, H \leq 20$

【输入样例】

```
6 9
....#.
.....#
......
......
......
......
......
#@...#
.#..#.
0 0
```

【输出样例】

```
45
```

【分析】

本题是求起点所在的连通块中点的数量，唯一坑点是输入n,m时是和传统的输入相反的。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 25;
char g[N][N];
bool st[N][N];
int n, m, sx, sy;
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };

int dfs(int x, int y)
{
    st[x][y] = true;
    int res = 1;
    for (int i = 0; i < 4; i++)
    {
        int nx = x + dx[i], ny = y + dy[i];

        if (nx >= 0 && nx < n && ny >= 0 && ny < m && !st[nx][ny] &&
```

```
    g[nx][ny] == '.')
            res += dfs(nx, ny);
    }
    return res;
}

int main()
{
    while (cin >> m >> n, m || n)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (cin >> g[i][j], g[i][j] == '@') sx = i, sy = j;
        memset(st, false, sizeof st);
        cout << dfs(sx, sy) << endl;
    }
    return 0;
}
```

四、AcWing 1224. 交换瓶子

【题目描述】

有N个瓶子，编号$1 \sim N$，放在架子上。

比如有5个瓶子：

```
2 1 3 5 4
```

要求每次拿起2个瓶子，交换它们的位置。

经过若干次后，使得瓶子的序号为：

```
1 2 3 4 5
```

对于这么简单的情况，显然，至少需要交换2次就可以复位。

如果瓶子更多呢？你可以通过编程来解决。

【输入格式】

第一行包含一个整数N，表示瓶子数量。

第二行包含N个整数，表示瓶子目前的排列状况。

【输出格式】

输出一个正整数，表示至少交换多少次，才能完成排序。

【数据范围】

$1 \leq N \leq 10000$

【输入样例1】

```
5
3 1 2 5 4
```

【输出样例1】

```
3
```

【输入样例2】

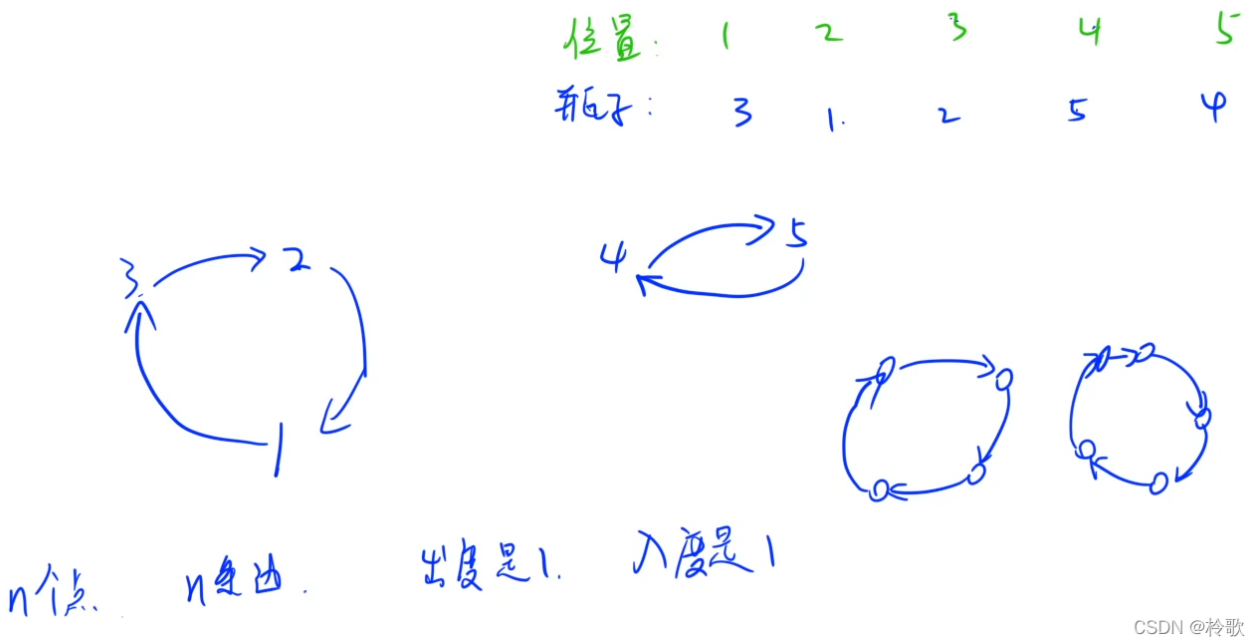
```
5
5 4 3 2 1
```

【输出样例2】

```
2
```

【分析】

我们将每个瓶子向它所应该在的位置上的那个瓶子连一条有向边，如下图所示：



CSDN @柃歌

那么显然每个瓶子的入度和出度都为1。初始时有若干个环，记为cnt，我们的目的是让整个图变成n个自环，而我们每次交换最多只能分裂出一个自环，因此需要的最少交换次数就是$n-cnt$。

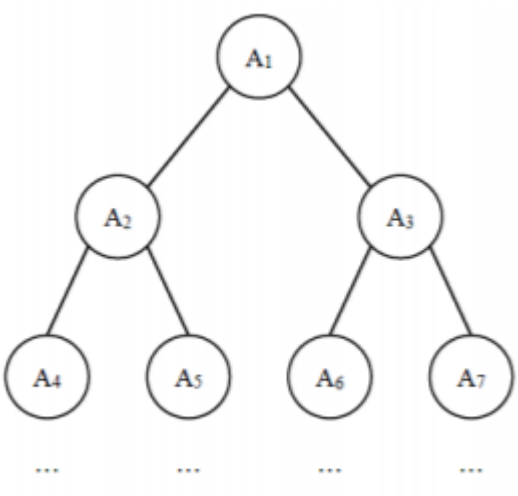
【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 10010;
int idx[N];//表示每个位置上的瓶子编号
bool st[N];
int n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> idx[i];
    int cnt = 0;//环的数量
    for (int i = 1; i <= n; i++)
        if (!st[i])
        {
            cnt++;
            for (int j = i; !st[j]; j = idx[j]) st[j] = true;
        }
    cout << n - cnt << endl;
    return 0;
}
```

五、AcWing 1240. 完全二叉树的权值

【题目描述】

给定一棵包含N个节点的完全二叉树，树上每个节点都有一个权值，按从上到下、从左到右的顺序依次是$A_1, A_2, \ldots, A_N$，如下图所示：



现在小明要把相同深度的节点的权值加在一起，他想知道哪个深度的节点权值之和最大？

如果有多个深度的权值和同为最大，请你输出其中最小的深度。

注：根的深度是1。

【输入格式】

第一行包含一个整数N。

第二行包含N个整数$A_1, A_2, \ldots, A_N$。

【输出格式】

输出一个整数代表答案。

【数据范围】

$1 \leq N \leq 10^5$

$-10^5 \leq A_i \leq 10^5$

【输入样例】

```
7
1 6 5 4 3 2 1
```

【输出样例】

```
2
```

【分析】

从第一层开始，将每一层的所有结点权值相加，判断是否为最大值，如果为最大值则更新一下最优解的层数res。对于一个完全二叉树，第$layer$层的节点数量最多为$2^{layer-1}$个，因此我们在循环每一层的时候要注意当前下标要在界内且当前这层累加的总结点数不超过$2^{layer-1}$个。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = 100010;
int a[N];
int n, res;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    LL maxv = -1e18, layer = 1;//layer表示当前层数
    for (int i = 0; i < n; layer++)
    {
        LL sum = 0, cnt = 0;
        while (i < n && cnt < 1 << layer - 1) sum += a[i++], cnt++;
        if (sum > maxv) res = layer, maxv = sum;
    }
    cout << res << endl;
    return 0;
}
```

六、AcWing 1096. 地牢大师

【题目描述】

你现在被困在一个三维地牢中，需要找到最快脱离的出路！

地牢由若干个单位立方体组成，其中部分不含岩石障碍可以直接通过，部分包含岩石障碍无法通过。

向北，向南，向东，向西，向上或向下移动一个单元距离均需要一分钟。

你不能沿对角线移动，迷宫边界都是坚硬的岩石，你不能走出边界范围。

请问，你有可能逃脱吗？

如果可以，需要多长时间？

【输入格式】

输入包含多组测试数据。

每组数据第一行包含三个整数L, R, C分别表示地牢层数，以及每一层地牢的行数和列数。

接下来是L个R行C列的字符矩阵，用来表示每一层地牢的具体状况。

每个字符用来描述一个地牢单元的具体状况。

其中，充满岩石障碍的单元格用`#`表示，不含障碍的空单元格用`.`表示，你的起始位置用`S`表示，终点用`E`表示。

每一个字符矩阵后面都会包含一个空行。

当输入一行为`0 0 0`时，表示输入终止。

【输出格式】

每组数据输出一个结果，每个结果占一行。

如果能够逃脱地牢，则输出`Escaped in x minute(s).`，其中`x`为逃脱所需最短时间。

如果不能逃脱地牢，则输出`Trapped!`。

【数据范围】

$1 \leq L, R, C \leq 100$

【输入样例】

```
3 4 5
S....
.###.
.##..
###.#

#####
#####
##.##
##...


#####
```

```
#####
#.###
####E

1 3 3
S##
#E#
###

0 0 0
```

【输出样例】

```
Escaped in 11 minute(s).
Trapped!
```

【分析】

三维的迷宫问题，直接看代码~

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
typedef pair<PII, int> PIII;
const int N = 110;
char g[N][N][N];
int dis[N][N][N];
int l, r, c, sl, sr, sc;
int dx[6] = { -1, 1, 0, 0, 0, 0 };
int dy[6] = { 0, 0, -1, 0, 1, 0 };
int dz[6] = { 0, 0, 0, 1, 0, -1 };

int bfs(int sl, int sr, int sc)
{
```

```
    memset(dis, -1, sizeof dis);
    queue<PIII> Q;
    Q.push({ { sl, sr }, sc });
    dis[sl][sr][sc] = 0;
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        int x = t.X.X, y = t.X.Y, z = t.Y;
        for (int i = 0; i < 6; i++)
        {
            int nx = x + dx[i], ny = y + dy[i], nz = z + dz[i];
            if (nx >= 0 && nx < l && ny >= 0 && ny < r && nz >= 0 && nz < c && g[nx][ny][nz] != '#' && !~dis[nx][ny][nz])
            {
                dis[nx][ny][nz] = dis[x][y][z] + 1, Q.push({ { nx, ny }, nz });
                if (g[nx][ny][nz] == 'E') return dis[nx][ny][nz];
            }
        }
    }
    return -1;
}

int main()
{
    ios::sync_with_stdio(false);
    while (cin >> l >> r >> c, l || r || c)
    {
        for (int i = 0; i < l; i++)
            for (int j = 0; j < r; j++)
                for (int k = 0; k < c; k++)
                    if (cin >> g[i][j][k], g[i][j][k] == 'S')
                        sl = i, sr = j, sc = k;
        int res = bfs(sl, sr, sc);
        if (!~res) cout << "Trapped!\n";
        else cout << "Escaped in " << res << " minute(s).\n";
    }
    return 0;
}
```

七、AcWing 1233. 全球变暖

【题目描述】

你有一张某海域$N \times N$像素的照片，`.`表示海洋、`#`表示陆地，如下所示：

```
.......
.##....
.##....
....##.
..####.
...###.
.......
```

其中“上下左右”四个方向上连在一起的一片陆地组成一座岛屿，例如上图就有2座岛屿。

由于全球变暖导致了海面上升，科学家预测未来几十年，岛屿边缘一个像素的范围会被海水淹没。

具体来说如果一块陆地像素与海洋相邻（上下左右四个相邻像素中有海洋），它就会被淹没。

例如上图中的海域未来会变成如下样子：

```
.......
.......
.......
.......
....#..
.......
.......
```

请你计算：依照科学家的预测，照片中有多少岛屿会被完全淹没。

【输入格式】

第一行包含一个整数N。

以下N行N列，包含一个由字符`#`和`.`构成的$N \times N$字符矩阵，代表一张海域照片，`#`表示陆地，`.`表示海洋。

照片保证第1行、第1列、第N行、第N列的像素都是海洋。

【输出格式】

一个整数表示答案。

【数据范围】

$1 \leq N \leq 1000$

【输入样例1】

```
7
.......
.##....
.##....
....##.
..####.
...###.
.......
```

【输出样例1】

```
1
```

【输入样例2】

```
9
.........
.##.##...
.#####...
.##.##...
.........
.##.#....
.#.###...
.#..#....
.........
```

【输出样例2】

```
1
```

【分析】

我们在搜索每个连通块（岛屿）的时候需要记录这个连通块是否存在至少一个陆地，其上下左右四个方向都为陆地，如果存在，那么这个岛屿就不会被淹没，否则这个岛屿就会全部被淹没，答案加一。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 1010;
char g[N][N];
bool st[N][N], flag;//flag表示当前连通块是否存在一个其上下左右都是陆地的点
int n, res;
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };

void dfs(int x, int y)
{
    st[x][y] = true;
    bool f = true;
    for (int i = 0; i < 4; i++)
    {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < n)
        {
            if (g[nx][ny] == '.') f = false;//只要四周有一个位置是海那么这个点就会被淹没
            if (g[nx][ny] == '#' && !st[nx][ny]) dfs(nx, ny);
        }
    }
    if (f) flag = true;
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%s", g[i]);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (g[i][j] == '#' && !st[i][j])
            {
                flag = false, dfs(i, j);
                if (!flag) res++;
            }
    printf("%d\n", res);
    return 0;
}
```

八、AcWing 1207. 大臣的旅费

【题目描述】

很久以前，T王国空前繁荣。

为了更好地管理国家，王国修建了大量的快速路，用于连接首都和王国内的各大城市。

为节省经费，T国的大臣们经过思考，制定了一套优秀的修建方案，使得任何一个大城市都能从首都直接或者通过其他大城市间接到达。

同时，如果不重复经过大城市，从首都到达每个大城市的方案都是唯一的。

J是T国重要大臣，他巡查于各大城市之间，体察民情。

所以，从一个城市马不停蹄地到另一个城市成了J最常做的事情。

他有一个钱袋，用于存放往来城市间的路费。

聪明的J发现，如果不在某个城市停下来修整，在连续行进过程中，他所花的路费与他已走过的距离有关，在走第x千米到第$x+1$千米这一千米中（x是整数），他花费的路费是$x+10$这么多。也就是说走1千米花费11，走2千米要花费23。

J大臣想知道：他从某一个城市出发，中间不休息，到达另一个城市，所有可能花费的路费中最多是多少呢？

【输入格式】

输入的第一行包含一个整数n，表示包括首都在内的T王国的城市数。

城市从1开始依次编号，1号城市为首都。

接下来$n-1$行，描述T国的高速路（T国的高速路一定是$n-1$条）。

每行三个整数P_i, Q_i, D_i，表示城市P_i和城市Q_i之间有一条双向高速路，长度为D_i千米。

【输出格式】

输出一个整数，表示大臣J最多花费的路费是多少。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq P_i, Q_i \leq n$

$1 \leq D_i \leq 1000$

【输入样例】

```
5
1  2  2
1  3  1
2  4  5
2  5  4
```

【输出样例】

```
135
```

【分析】

假设走过的路程为s千米，那么花费为：$(10+1)+(10+2)+\cdots+(10+s)=10*s+(1+s)*s/2$。所以我们需要做的就是求出可能的最远的路程。

本题的图是一棵树，树中最长的路径也称为树的直径，那么如何求出这个直径呢？

1. 首先随便选择一个点作为起点（假设为1号点），找出距离1号点最远的点x，则x一定是树的某一条直径的端点；
2. 以点x为起点再求一遍该点到其他点的距离，最远的距离即为树的直径，也就是树中最长的路径。

求树中两点间的距离可以使用DFS也可以使用BFS，本题使用DFS求解。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = 100010;
int e[N << 1], ne[N << 1], d[N << 1], h[N], idx;
int dis[N];
int n;

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

```

```cpp
void dfs(int u, int fa, int w)
{
    dis[u] = w;
    for (int i = h[u]; ~i; i = ne[i])
        if (e[i] != fa) dfs(e[i], u, w + d[i]);
}

int main()
{
    cin >> n;
    memset(h, -1, sizeof h);
    for (int i = 1; i < n; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    dfs(1, -1, 0);//随机选个起点然后求出该点到其他点的距离
    int x = 1;//表示离起点最远的点
    for (int i = 2; i <= n; i++)
        if (dis[i] > dis[x]) x = i;
    dfs(x, -1, 0);//再以x为起点求一遍该点到其他点的距离
    int s = 0;//表示离x的最远距离
    for (int i = 1; i <= n; i++) s = max(s, dis[i]);
    cout << 10 * s + (LL)(1 + s) * s / 2 << endl;
    return 0;
}
```

九、AcWing 826. 单链表

【题目描述】

实现一个单链表，链表初始为空，支持三种操作：

- 向链表头插入一个数；
- 删除第k个插入的数后面的数；
- 在第k个插入的数后插入一个数。

现在要对该链表进行M次操作，进行完所有操作后，从头到尾输出整个链表。

注意：题目中第k个插入的数并不是指当前链表的第k个数。例如操作过程中一共插入了n个数，则按照插入的时间顺序，这n个数依次为：第1个插入的数，第2个插入的数，...，第n个插入的数。

【输入格式】

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令（数据保证合法），操作命令可能为以下几种：

1. `H x`，表示向链表头插入一个数x。
2. `D k`，表示删除第k个插入的数后面的数（当k为0时，表示删除头结点）。
3. `I k x`，表示在第k个插入的数后面插入一个数x（此操作中k均大于0）。

【输出格式】

共一行，将整个链表从头到尾输出。

【数据范围】

$1 \leq M \leq 100000$

【输入样例】

```
10
H 9
I 1 1
D 1
D 0
H 6
I 3 6
I 4 5
I 4 5
I 3 4
D 6
```

【输出样例】

```
6 4 6 5
```

【分析】

复习一下单链表裸题，直接看代码~

【代码】

```
#include <iostream>

using namespace std;
```

```cpp

const int N = 100010;
int e[N], ne[N], head, idx;
int n;

void add_to_head(int x)
{
    e[idx] = x, ne[idx] = head, head = idx++;
}

void add(int k, int x)
{
    e[idx] = x, ne[idx] = ne[k], ne[k] = idx++;
}

void del(int k)
{
    ne[k] = ne[ne[k]];
}

int main()
{
    cin >> n;
    head = -1, idx = 0;
    while (n--)
    {
        char op;
        int k, x;
        cin >> op;
        if (op == 'H') { cin >> x; add_to_head(x); }
        else if (op == 'I') { cin >> k >> x; add(k - 1, x); }
        else
        {
            cin >> k;
            if (!k) head = ne[head];
            else del(k - 1);
        }
    }
    for (int i = head; ~i; i = ne[i]) cout << e[i] << ' ';
    return 0;
}
```