# 图论



## Flood Fill

算法用途：找到某个点所在的**连通块**

基于不同的用途，可以额外维护更多的信息，比如**连通块中点的个数**

基本实现有两种：$\mathrm{BFS}$ 与 $\mathrm{DFS}$

时间复杂度均为 $O(n+m)$，其中 $n$ 为点数，$m$ 为边数

---

### 基础：

原题链接：AcWing 1113. 红与黑

#### BFS 写法

- `st[i][j]` 表示 $(i,j)$ 已经遍历过
- 只要当前点没有出界、没有被遍历过、当前点在连通块内，我们就将当前点加入到队列中去
- 如果我希望统计连通块中点的数量，那么我需要在**每次遍历一个新的点时就统计一次**，即在 `for` 循环内统计
- 如果我希望统计连通块的数量，那么在 $\mathrm{Flood\ Fill}$ 的入口出进行通统计

完整代码：

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

#define x first
#define y second

using namespace std;
typedef pair<int, int> PII;
const int N = 25;
```

```
char g[N][N];
bool st[N][N];

int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};

int n, m;

void bfs(int sx, int sy, int& sum)
{
    queue<PII>q;
    q.push({sx, sy});
    st[sx][sy] = true;

    while(q.size())
    {
        auto it = q.front();
        q.pop();
        int x = it.x, y = it.y;
        for(int i = 0; i < 4; i ++)
        {
            int nx = x + dx[i], ny = y + dy[i];
            if(nx < 0 || ny < 0 || nx >= m || ny >= n) continue;
            if(st[nx][ny]) continue;
            if(g[nx][ny] == '#') continue;

            q.push({nx, ny});
            st[nx][ny] = true;
            sum++;

        }
    }
}

int main()
{
    while(cin >> n >> m, n || m)
    {
        memset(g, 0, sizeof g);
        memset(st, false, sizeof st);
        for(int i = 0; i < m; i ++)
            cin >> g[i];
        int cnt = 0;
        for(int i = 0; i < m; i ++)
            for(int j = 0; j < n; j ++)
                if(g[i][j] == '@')
                {
                    int sum = 0;
                    bfs(i, j, sum);
                    cnt += sum;
                }
        cout << cnt + 1 << endl;
    }
```

```
    return 0;
}
```

**DFS 写法**

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


using namespace std;
typedef pair<int, int> PII;
const int N = 25;

char g[N][N];
bool st[N][N];

int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};

int n, m;

int dfs(int sx, int sy)
{
    st[sx][sy] = true;
    int cnt = 1;

    for(int i = 0; i < 4; i ++)
    {
        int nx = sx + dx[i], ny = sy + dy[i];
        if(nx < 0 || ny < 0 || nx >= m || ny >= n) continue;
        if(g[nx][ny] == '#') continue;
        if(st[nx][ny]) continue;

        cnt += dfs(nx, ny);
    }
    return cnt;
}

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

```
                }
            cout << cnt << endl;
        }

        return 0;
    }
```

## 变式

原题链接：AcWing 1233. 全球变暖

对于每一块连通块（陆地），我们 BFS 时统计出连通块中点的数量 $cnt$，同时再统计边界的数量 $bount$ （只要某个点上下左右存在海洋，那么就是边界）

只要 $cnt == bount$ 那么这块陆地将会被淹没，我们统计所有不会被淹没的陆地即可

完整代码：

```
#include <iostream>
#include <queue>

#define x first
#define y second

using namespace std;

const int N = 1e3 + 10;
typedef pair<int, int> PII;

char g[N][N];
bool st[N][N];

int n;

int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};

void bfs(int sx, int sy, int& cnt, int& bound)
{
    queue<PII>q;
    q.push({sx, sy});
    st[sx][sy] = true;
    while(q.size())
    {
        auto t = q.front();
        q.pop();

        st[t.x][t.y] = true;
        cnt++;

        bool is_bount = false;

        for(int i = 0; i < 4; i ++)
```

```
        {
            int nx = t.x + dx[i], ny = t.y + dy[i];
            if(nx >= 0 && ny >= 0 && nx < n && ny < n)
            {
                if(st[nx][ny]) continue;
                if(g[nx][ny] == '.') is_bount = true;
                else
                {
                    q.push({nx, ny});
                    st[nx][ny] = true;
                }
            }
        }
        if(is_bount) bound++;
    }

}

int main()
{
    cin >> n;
    for(int i = 0; i < n; i ++)
        cin >> g[i];
    int ans = 0;
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n; j ++)
            if(!st[i][j] && g[i][j] == '#')//当前点没有被遍历过并且是陆地
            {
                int cnt = 0, bound = 0;//cnt表示陆地数量，bound表示会被淹没的数量
                bfs(i, j, cnt, bound);
                if(cnt == bound) ans++;
            }
    cout << ans << endl;
    return 0;
}
```

---

## BFS

BFS 中，每个点只会被遍历一次，因此时间复杂度为 $O(n+m)$，其中 $n$ 为点数，$m$ 为边数

BFS 需要额外记录点的坐标，因此需要用 `pair<int, int>` 来存储 其次，我们还需要确定哪些点已经遍历过，因此需要用 `st[i][j]` 来标记 $(i,j)$ 这个点是否已经遍历过

在初始化时，我们将**起点**压入队列，**并用 `st` 标记起点**

---

原题链接：AcWing 1562. 微博转发

如果 $A$ 关注了 $B$，那么 $A$ 便会转发 $B$ 的帖子，因此我们建立一条从 $B$ 指向 $A$ 的**有向边**

由于层数最大为 $L$ 层，因此我们需要统计**所有路径长度不超过 $L$ 的点的数量**

由于点的边权全部都是 $1$，因此我们直接用 $\mathrm{BFS}$ 来统计即可

- 点的边权不同，我们考虑 $\mathrm{SPFA}$
- 如果点的边权只有 $0$ 和 $1$，考虑双端队列
- 如果点的边权只有 $1$，考虑 $\mathrm{BFS}$

完整代码：

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

using namespace std;

const int N = 1e5 + 10;

int h[N], e[N], ne[N], idx;
bool st[N];

void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

int n, L;

int bfs(int u)
{
    memset(st, false, sizeof st);

    queue<int>q;
    q.push(u);
    st[u] = true;

    int cnt = 0;

    for(int i = 1; i <= L; i ++)//由于我们需要遍历 L 层，因此在队列遍历的时候只能遍历本层的节点，因此需要取队列的大小
    {
        int sz = q.size();
        while(sz--)
        {
            int t = q.front();
            q.pop();
            for(int j = h[t]; j != -1; j = ne[j])
            {
                int k = e[j];
                if(!st[k])
                {
                    st[k] = true;
                    q.push(k);
                    cnt++;
```

```
                }
            }
        }
    }
    return cnt;
}

int main()
{
    memset(h, -1, sizeof h);
    cin >> n >> L;
    for(int i = 1; i <= n; i ++)
    {
        int cnt = 0;
        cin >> cnt;
        for(int j = 1; j <= cnt; j ++)
        {
            int x;
            cin >> x;
            add(x, i);
        }
    }
    int k;
    cin >> k;
    while(k--)
    {
        int q;
        cin >> q;
        cout << bfs(q) << endl;
    }
    return 0;
}
```

# 拓扑排序

## 基础

只有有向图才有拓扑排序，无向图是没有拓扑排序的