Lesson1

> **DFS（执着）**
>
> 数据结构：栈 空间：O(h) 在空间上比BFS有优势
>
> 不具有最短性
>
> 两个特性
>
> - 回溯
>
> 回溯完之后注意**恢复现场**
> - 剪枝
>
> 顺序与搜索树
>
> - e1: 全排列问题

```cpp
#include <iostream>

using namespace std;

const int N = 10;

int n;
int path[N];
bool st[N];

void dfs(int u) {
    if (u == n) {   //退出条件
        for (int i=0; i<n; i++) printf("%d ", path[i]);
        puts("");
        return ;
    }

    for (int i=1; i<=n; i++) {
        if (!st[i]) {
            path[u] = i;
            st[i] = true;
            dfs(u+1);
            st[i] = false;  //恢复现场
        }
    }
}


int main(void) {
    scanf("%d", &n);

    dfs(0);

    return 0;
}
```

> - e2: n皇后问题
>
> 第一种搜索顺序:

```cpp
#include <iostream>

using namespace std;

const int N = 20;

int n;
char g[N][N];
bool col[N], dg[N], udg[N];  //dg对角线，udg反对角线，dg数量是n的二倍，所以N=20

//全排列搜索顺序
void dfs(int u) {
    if (u == n) {
        for (int i=0; i<n; i++) puts(g[i]);
        puts("");
        return ;
    }

    for (int i=0; i<n; i++) {
        if (!col[i] && !dg[i+u] && !udg[i-u+n]) {
            g[u][i] = 'Q';
```

```
            col[i] = dg[i+u] = udg[i-u+n] = true;
            dfs(u+1);
            //恢复现场
            g[u][i] = '.';
            col[i] = dg[i+u] = udg[i-u+n] = false;
        }
    }
}

int main(void) {
    scanf("%d", &n);

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            g[i][j] = '.';

    dfs(0);

    return 0;
}
```

第二种搜索顺序:

```
#include <iostream>

using namespace std;

const int N = 20;

int n;
char g[N][N];
bool row[N], col[N], dg[N], udg[N];

void dfs(int x, int y, int s) {
    if (y == n) y = 0, x++;

    if (x == n) {
        if (s == n) {
            for (int i=0; i<n; i++) puts(g[i]);
            puts("");
        }

        return ;
    }

    //不放皇后
    dfs(x, y+1, s);

    //放皇后
    if (!row[x] && !col[y] && !dg[x+y] && !udg[y-x+n]) {
        g[x][y] = 'Q';
        row[x] = col[y] = dg[x+y] = udg[y-x+n] = true;
        dfs(x, y+1, s+1);
        g[x][y] = '.';
        row[x] = col[y] = dg[x+y] = udg[y-x+n] = false;
    }
}

int main(void) {
    scanf("%d", &n);

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            g[i][j] = '.';

    dfs(0, 0, 0);

    return 0;
}
```

BFS（稳重，层层遍历）

数据结构：队列 空间：O(2^h)

当每条边权重相同时，能找到最短路（DFS不具备）

```
queue <- 初始
while queue不空 {
    t <- 对头
    扩展 t 所有邻点
}
```

- e1：走迷宫

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

using namespace std;

typedef pair<int, int> PII;

const int N = 110;

int n, m;
int g[N][N], d[N][N];   //d[i][j]记录起点到点(i, j)的距离
PII q[N*N], pre[N][N];  //pre记录路径，逆序
int hh, tt = -1;

int bfs() {
    memset(d, -1, sizeof d);

    d[0][0] = 0;
    q[++tt] = {0, 0};

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

    while (hh <= tt) {
        auto t = q[hh++];

        for (int i=0; i<4; i++) {
            int x = t.first+dx[i], y = t.second+dy[i];

            if (x>=0 && x<n && y>=0 && y<m && !g[x][y] && d[x][y] == -1) {
                d[x][y] = d[t.first][t.second]+1;
                pre[x][y] = t;
                q[++tt] = {x, y};
            }
        }
    }

    //输出路径
    // int x = n-1, y = m-1;
    // while (x || y) {
    //     cout << x << " " << y <<endl;
    //     auto t = pre[x][y];
    //     x = t.first, y = t.second;
    // }

    return d[n-1][m-1];
}

int main(void) {
    scanf("%d%d", &n, &m);

    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++)
            scanf("%d", &g[i][j]);

    cout << bfs() << endl;

    return 0;
}

```

- e2: 八数码问题

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

using namespace std;

queue<string> q;
```

```cpp
unordered_map<string, int> d;

int bfs(string state) {
    d[state] = 0;
    q.push(state);

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

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

        if (t == "12345678x") return d[t];

        int dist = d[t];
        int k = t.find('x');
        int x = k/3, y = k%3;

        for (int i=0; i<4; i++) {
            int a = x+dx[i], b = y+dy[i];

            if (a>=0 && a<3 && b>=0 && b<3) {
                swap(t[a*3+b], t[k]);

                if (!d.count(t)) {
                    d[t] = dist+1;
                    q.push(t);
                }

                swap(t[a*3+b], t[k]);   //恢复现场
            }
        }
    }

    return -1;
}

int main(void) {
    char s[2];
    string state;

    for (int i=0; i<9; i++) {
        scanf("%s", s);
        state += *s;
    }

    cout << bfs(state);

    return 0;
}

```

树与图的存储

树是一种特殊的图（无环连通图 ）

图的类型

- 有向图

  存储方式

  - 邻接矩阵，使用二维数组 g[a, b] （不能保存重边）

    空间复杂度n^2，适用于稠密图

  - **邻接表**（每个结点开一个单链表，与拉链法哈希表一直）

    适用于**稀疏图**

```cpp
const int N = 100010, M = N*2;  //N代表结点数，M代表边数

int h[N], e[M], ne[M], idx;

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

- 无向图

对于一条无向边，建两条有向边

树与图的深度优先遍历

- 遍历方式

```
const int N = 100010, M = N*2;

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

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

void dfs(int u) {
    st[u] = true;   //已经被遍历

    for (int i=h[u]; i!=-1; i=ne[i]) {
        int j = ne[i];
        if (!st[j]) dfs(j);
    }
}
```

- e1: 树的重心

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

using namespace std;

const int N = 100010, M = 2*N;

int n;
int h[N], e[M], ne[M], idx;
bool st[N];
int ans = N;

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

//返回以u为根的子树的结点个数（包含u）
int dfs(int u) {
    st[u] = true;

    int sum = 1, res = 0;
    for (int i=h[u]; i!=-1; i=ne[i]) {
        int j = e[i];
        if (!st[j]) {
            int s = dfs(j);
            sum += s;
            res = max(res, s);
        }
    }

    res = max(res, n-sum);
    ans = min(ans, res);

    return sum;

}

int main(void) {
    scanf("%d", &n);

    memset(h, -1, sizeof h);

    for (int i=0; i<n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }

    dfs(1);

    cout << ans;

```

```
    return 0;
}
```

树与图的宽度优先遍历

- e1: 图中点的层次

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

using namespace std;

const int N = 100010, M = N;

int n, m;
int h[N], e[M], ne[M], idx;
int q[M], hh, tt = -1;
int d[N];   //每个点到1号点的距离

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

int bfs() {
    memset(d, -1, sizeof d);

    d[1] = 0;
    q[++tt] = 1;

    while (hh <= tt) {
        int t = q[hh++];

        for (int i=h[t]; i!=-1; i=ne[i]) {
            int j = e[i];
            if (d[j] == -1) {
                d[j] = d[t]+1;
                q[++tt] = j;
            }
        }
    }

    return d[n];
}

int main(void) {
    scanf("%d%d", &n, &m);

    //注意初始化的位置
    memset(h, -1, sizeof h);

    for (int i=0; i<m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    cout << bfs() << endl;


    return 0;
}

```

拓扑排序（有向图宽搜的应用）

有环图不存在拓扑排序。可以证明，有向无环图一定存在拓扑序列

度数

- 入度：有多少边指入，**入度为0**的点可以作为**拓扑序列的起点**
- 出度：有多少边指出

步骤

```
queue <- 所有入队为0的点
while queue不空 {
    t <- 队头
    枚举t的所有出边 t -> j
        删掉 t -> j, d[j]--;
        if (d[j] == 0) queue <- j;
}
```

- e1: 有向图的拓扑序列

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

using namespace std;

const int N = 100010, M = 100010;

int n, m;
int h[N], e[M], ne[M], idx;
int q[N], hh, tt = -1;
int d[N];   //d[i]表示第i个结点的入度

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

bool toposort() {
    //将所有入度为0的点入队
    for (int i=1; i<=n; i++)
        if (!d[i]) q[++tt] = i;

    while (hh <= tt) {
        int t = q[hh++];

        for (int i=h[t]; i!=-1; i=ne[i]) {
            int j = e[i];
            d[j]--;
            if (d[j] == 0) q[++tt] = j;
        }
    }

    return tt == n-1;
}

int main(void) {
    scanf("%d%d", &n, &m);

    memset(h, -1, sizeof h);

    for (int i=0; i<m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        d[b]++;
    }

    if (toposort())
        for (int i=0; i<n; i++) printf("%d ", q[i]);
    else puts("-1");

    return 0;
}
```