1. 图的存储

- 邻接矩阵

```
int n,e;    //n为顶点数，e为边数
cin >> n >> e;
//邻接矩阵
const int DISCONNECT = -1;
vector<vector<int> > map(n,vector<int>(n,DISCONNECT));

while(e--){
    int from,to;
    cin>>from>>to;
    map[from][to] = map[to][from] = 1;
}

for(int i = 0;i < n; i++){
    for(int j = 0;j < n; j++){
        cout<<map[i][j]<<" ";
    }
    cout<<endl;
}
```

- 有权重的邻接矩阵

```
int n,e;    //n为顶点数，e为边数
cin >> n >> e;
//有权重的邻接矩阵
const int DISCONNECT = INT_MAX;
vector<vector<int> > map(n,vector<int>(n,DISCONNECT));

while(e--){
    int from,to,weight;
    cin >> from >> to >> weight;
    map[from][to] = map[to][from] = weight;
}

for(int i = 0;i < n; i++){
    for(int j = 0;j < n; j++){
        cout<<map[i][j]<<" ";
    }
    cout<<endl;
}
```

- 邻接表

```
int n,e;    //n为顶点数，e为边数
cin >> n >> e;
//邻接表
const int DISCONNECT = INT_MAX;
vector<vector<int> > map(n,vector<int>());

while(e--){
    int from,to;
    cin >> from >> to;
    map[from].push_back(to);
    map[to].push_back(from);
}
```

- 有权重的邻接表

```
int n,e;    //n为顶点数，e为边数
cin >> n >> e;
//邻接表的权重
struct node
{
    int index;
    int weight;
    //按从大到小的顺序输出
    bool operator<(node n){
        if(n.index!=index)
            return index<n.index;
    }
};
vector<vector<node> > map(n,vector<node>());
while(e--){
    int from,to,weight;
    cin >> from >> to >> weight;
    map[from].push_back( {to,weight} );
    //map[to].push_back( {from,weight} );
}
for(int i = 0;i < n; i++){
    if(!map[i].empty()){
        cout<<i<<":";
        sort(map[i].begin(),map[i].end());
        for(auto it=map[i].begin();it!=map[i].end();it++){
            cout<<"("<<i<<","<<it->index<<","<<it->weight<<")";
        }
        cout<<endl;
    }
}
```

2.图的遍历

- DFS

邻接矩阵：

```
vector<vector<int>> Grap(2000,vector<int>(2000,-1));
bool vis[2000]={false};
//领接矩阵DFS
void DFS(int n,int str)
{
    cout<<str<<endl;
    vis[str]=true;

    for(int i=0; i<n; i++){
        if(Grap[str][i]!=-1 && vis[i]==false)
            DFS(n,i);
    }
}
```

邻接表：

方法1.

```cpp
vector<vector<int>> biao(2000,vector<int>());
bool vis[2000]={false};
//领接表
void DFS(int n,int str)
{
    cout<<str<<endl;
    vis[str]=true;

    for(auto &NextIndex:biao[str]){
        if(vis[NextIndex]==false){
            DFS(n,NextIndex);
        }
    }
}
```

方法2.

题目：https://pintia.cn/problem-sets/1375335383906332672/problems/1375479256565624832

```cpp
void DFS(int n,int str)
{
    cout<<str+1<<" ";
    vis[str]=true;

    for(auto it=biao[str].rbegin();it!=biao[str].rend();it++){
        if(vis[(*it)]==false){
            DFS(n,(*it));
        }
    }
}
```

- BFS

邻接矩阵：

```cpp
vector<vector<int>> Grap(2000,vector<int>(2000,-1));
bool vis[2000]={false};
//邻接矩阵
void DFS(int n,int str)
{
    queue<int> q;
    q.push(str);
    vis[str]=true;

    while(!q.empty()){
        int cur=q.front();
        q.pop();
        cout<<cur<<endl;

        for(int i=0; i<n; i++){
            if(vis[i]==false && Grap[cur][i]!=-1){
                q.push(i);
                vis[i]=true;
            }
        }
    }
}
```

邻接表：

```cpp
vector<vector<int>> biao(2000,vector<int>());
bool vis[2000]={false};
//领接表
void DFS(int n,int str)
{
    queue<int> q;
    q.push(str);
    vis[str]=true;

    while(!q.empty()){
        int cur=q.front();
        q.pop();
        cout<<cur<<endl;

        //遍历每一个和它相连的点
        for(auto nextindex:biao[cur]){
            if(vis[nextindex]==false){
                vis[nextindex]=true;
                q.push(nextindex);
            }
        }
    }
}
```

记录层数：

```cpp
// 邻接表-BFS-记录层数？
void bfs_biao_level(int startIndex) {
    queue<int> que;
    que.push(startIndex);
    vis[startIndex] = true;
    int level = 1;

    while (!que.empty()) {
        queue<int> next;
        cout << "level = " << level++ << endl;
        while (!que.empty()) {
            int curIndex = que.front();
            que.pop();

            cout << curIndex << endl;

            // 遍历每一个和他相连的点
            for (const auto &nextIndex:biao[curIndex]) {
                if (!vis[nextIndex]) {
                    vis[nextIndex] = true;
                    next.push(nextIndex);
                }
            }
        }
        que = next;
    }
}
```

3. 有向图的拓扑序列

有向无环图一定存在一个拓扑序列（有向无环图又被称为拓扑图）

拓扑序列：对于图中的每一条边（x,y），x在序列中都出现在y之前，则称为拓扑序列。