

6.19 《图的存储&遍历》课堂笔记

本周作业

1、P7771

要求：

- (1) 做到 50 分
- (2) 根据参考代码完善本题
- (3) 思考有没有提高分数的思路

解题思路

- ①先判断是否存在欧拉回路（通过入度和出度全部相等判断）
- ②统计起点和终点的个数（通过入度和出度判断），同时要存在入度-出度 >1 或者出度-入度 >1 的情况，则直接输出 No
- ③如果是存在欧拉回路，则从 1 开始进行深搜，并输出路径
- ④如果是存在欧拉路，则从起点开始进行深搜，并输出路径

```
1  #include <iostream>
2  using namespace std;
3  int g[1010][1010];
4  int ru[100010];
5  int chu[100010];
6  int path[1000000];
7  int n, m, sta, en;
8  void dfs(int x){
9      for(int i = 1; i <= n; i++){
10         if(g[x][i] > 0){
11             g[x][i]--;
12             dfs(i);
13         }
14     }
15     en++;
16     path[en] = x;
17 }
18 int main(){
19     cin >> n >> m;
20     int x, y, cnt1 = 0, cnt2 = 0;
21     for(int i = 1; i <= m; i++){
22         cin >> x >> y;
23         g[x][y]++;
24         ru[y]++;
25         chu[x]++;
26     }
27     bool flag = false;
28     for(int i = 1; i <= n; i++){//判断是否是回路
29         if(chu[i] != ru[i]){
30             flag = true;
31         }
32     }
```

```

33 |     for(int i = 1; i <= n; i++){
34 |         if(chu[i] == ru[i] + 1){
35 |             sta = i;
36 |             cnt1++; //统计起点
37 |         }
38 |         if(ru[i] == chu[i] + 1){
39 |             cnt2++; //统计起点
40 |         }
41 |         if(chu[i] - ru[i] >= 2 || ru[i] - chu[i] >= 2){
42 |             cout << "No";
43 |             return 0;
44 |         }
45 |     }
46 |     if(!flag){
47 |         dfs(1);
48 |         for(int i = en; i >= 1; i--){
49 |             cout << path[i] << " ";
50 |         }
51 |         return 0;
52 |     }
53 |     if(cnt1 == 1 && cnt2 == 1){
54 |         dfs(sta);
55 |         for(int i = en; i >= 1; i--){
56 |             cout << path[i] << " ";
57 |         }
58 |         return 0;
59 |     }
60 |     cout << "No";
61 |     return 0;
62 | }

```

2、P3916(要求做到 60 分)

3、复习这段时间学习的容器、算法、图的存储与访问、邻接矩阵、欧拉图等知识

上周作业

1、洛谷 P1918 (用 map 方法)

```

1 | #include <iostream>
2 | #include <algorithm>
3 | #include <map>
4 | using namespace std;
5 | const int N = 100010;
6 | int n;
7 | map<int, int> ma;
8 | int Q;
9 | int main() {
10 |     int curInt = 0;
11 |     int loc = 0;
12 |     cin >> n;
13 |     for(int i = 1; i <= n; i++) {
14 |         cin >> curInt;
15 |         ma[curInt] = i;
16 |     }

```

```

17 |     cin >> Q;
18 |     for(int i = 1; i <= Q; i++) {
19 |         map<int, int> :: iterator it = ma.begin();
20 |         cin >> loc;
21 |         it = ma.find(loc);
22 |         if(it == ma.end()) {
23 |             cout << 0 << endl;
24 |         }
25 |         else {
26 |             cout << (*it).second << endl;
27 |         }
28 |     }
29 |
30 |     return 0;
31 | }

```

2、Wood.cpp

作业2 wood.cpp (set写)

博艾市有一个木材仓库，里面可以存储各种长度的木材，但是保证没有两个木材的长度是相同的。作为仓库负责人，你有时候会进货，有时候会出货，因此需要维护这个库存。有不超过 100000 条的操作：

进货，格式1：在仓库中放入一根长度为 Length(不超过 10^9) 的木材。如果已经有相同长度的木材那么输出Already Exist。

出货，格式2：从仓库中取出长度为 Length 的木材。如果没有刚好长度的木材，取出仓库中存在的和要求长度最接近的木材。如果有多根木材符合要求，取出比较短的一根。输出取出的木材长度。如果仓库是空的，输出Empty。

输入输出样例

输入 #1

```

7
11
15
13
23
23
23
23

```

输出 #1

```

3
1
5
Empty

```

```

#include <iostream>
#include <set>
using namespace std;

set<int> _wareHouse;
int _n = 0;
int _op = 0;
int _len = 0;

void Inp() {
    if(_wareHouse.count(_len) == 1) {
        cout << "Already Exist" << endl;
    }
    else {
        _wareHouse.insert(_len);
    }
}

```

```

void Outp() {
    if(_wareHouse.empty()) {
        cout << "Empty" << endl;
    }
    else {
        if(_wareHouse.count(_len) == 1) {
            cout << _len << endl;
            _wareHouse.erase(_len);
        }
        //删除相似值
        else {
            _wareHouse.insert(_len);
            set<int>::iterator it = _wareHouse.find(_len);
            set<int>::iterator it1 = it; //it1动, it不会动, 只保存原位置
            set<int>::iterator it2 = it; //it2动, it不会动, 只保存原位置
            set<int>::iterator it3 = it; //提前动, 为了和end()比较
            it3++;
            //已经是最短
            if(it1 == _wareHouse.begin()) {
                it1++;
                cout << *it1 << endl;
                _wareHouse.erase(it1); //删除后一个位置
            }
            //已经是最长
            //错误写法else if(++it1 == _wareHouse.end()) {
            else if(it3 == _wareHouse.end()) {
                it1--;
                cout << *it1 << endl;
                _wareHouse.erase(it1); //删除前一个位置
            }
            //中间
            else {
                it1--;
                it2++;
                if(_len - *it1 <= *it2 - _len) {
                    cout << *it1 << endl; //左边差距小取左的
                    _wareHouse.erase(it1); //删除前一个位置
                }
                else {
                    cout << *it2 << endl;
                    _wareHouse.erase(it2); //删除后一个位置
                }
            }
            _wareHouse.erase(it); //删除原位置元素
        }
    }
}

int main () {
    cin >> _n;
    for(int i = 1; i <= _n; i++) {
        cin >> _op >> _len;
        if(_op == 1) {
            Inp();
        }
        else {
            Outp();
        }
    }
    return 0;
}

```

课堂内容

1、图的概念

定义：图（Graph）是由顶点的有穷非空集合和顶点之间边的集合组成，通常表示为： $G(V,E)$ ，其中， G 表示一个图， V 是图 G 中顶点的集合， E 是图 G 中边的集合。

在图中需要注意的是：

- (1) 线性表中我们把数据元素叫元素，树中将数据元素叫结点，在图中数据元素，我们则称之为**顶点**（ V ）。
- (2) 线性表可以没有元素，称为空表；树中可以没有节点，称为空树；但是，在图中**不允许没有顶点**（有穷非空性）。
- (3) 线性表中的各元素是线性关系，树中的各元素是层次关系，而图中各顶点的关系是用**边**来表示（边集可以为空）。

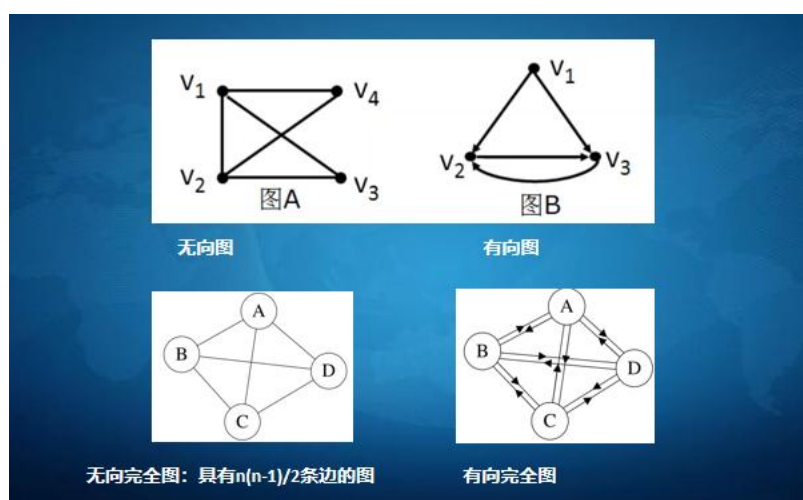
2、图的种类

有向图：带方向（箭头）的图

无向图：不带方向（箭头）的图

无向完全图：具有 $n(n-1)/2$ 条边的无向图

有向完全图：具有 $n(n-1)$ 条边的有向图



3、图的相关概念

(1) 顶点的度

顶点 V_i 的度 (Degree) 是指在图中与 V_i 相关联的边的条数。对于有向图来说, 有入度 (In-degree) 和出度 (Out-degree) 之分, 有向图顶点的度等于该顶点的入度和出度之和。

- **入度:** 有向图的某个顶点作为终点的次数和
- **出度:** 有向图的某个顶点作为起点的次数和

(2) 邻接

若无向图中的两个顶点 V_1 和 V_2 存在一条边 (V_1, V_2) , 则称顶点 V_1 和 V_2 邻接;

若有向图中存在一条边 $\langle V_3, V_2 \rangle$, 则称顶点 V_3 与顶点 V_2 邻接, 且是 V_3 邻接到 V_2 ;

(3) 连通图

图 G 中任意两个顶点 u, v 之间存在一条从 u 通过若干条边、点到达 v 的通路, 则图 G 是连通的

(4) 强连通图

有向图中任意两点都连通的最大子图。

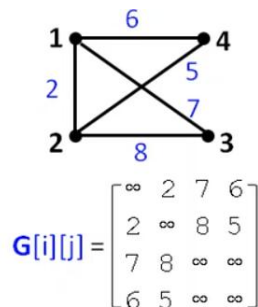
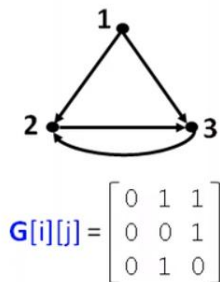
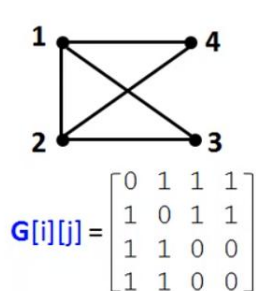
4、图的存储

图的邻接矩阵 (Adjacency Matrix) 存储方式是用一个二维数组存储图中的各顶点的邻接关系和权值。

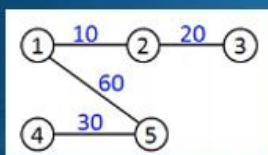
$G[i][j]$: 存储从 i 点到 j 点的边的权值, i 表示起点, j 表示终点

$G[i][j]=1$ 或权值 点 i 与点 j 之间有边时

$G[i][j]=0$ 或无穷 点 i 与点 j 之间无边时



5、邻接矩阵的存储和访问



样例输入:

```

5 4
1 2 10
2 3 20
4 5 30
1 5 60
  
```

```

#include<bits/stdc++.h>
using namespace std;
int mapp[110][110];
int n,m;
int x,y,num;

int main(){
    memset(mapp,0x7f,sizeof(mapp));
    cin >> n >> m;
    for(int i=1; i<=m; i++){
        cin >> x >> y >> num;
        mapp[x][y] = num;
        mapp[y][x] = num;
    }
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            cout << setw(12) << mapp[i][j];
        }
        cout << endl;
    }
    return 0;
}
  
```

6、关于 memset 的要点:

- (1) memset 可以快速的初始化数组，且速度是 for 循环 8 倍
- (2) 如果想赋值具体内容只能给 0 或者 -1
- (3) 正无穷 1: 0x3f (1061109567)，介于 int 最大值一半和 1e9 之间，可以用于预防相加溢出
- (4) 正无穷 2: 0x7f，介于 2e9 到 int 之间
- (5) 负无穷: 128，刚好比 int 最小值大一点

7、邻接矩阵的 DFS:

```

int mapp[110][110];
bool vis[110];
int n,m;
int x,y,num;

void dfs(int x){
    vis[x] = true;
    cout << x << " ";
    for(int i=1; i<=n; i++){
        if(mapp[x][i] != -1 && vis[i] == false){
            dfs(i);
        }
    }
}

int main(){
    memset(mapp,-1,sizeof(mapp));
    memset(vis,false,sizeof(vis));
    cin >> n >> m;
    for(int i=1; i<=m; i++){
        cin >> x >> y >> num;
        mapp[x][y] = num;
        mapp[y][x] = num;
    }
    dfs(1);
    return 0;
}
  
```

8、欧拉图

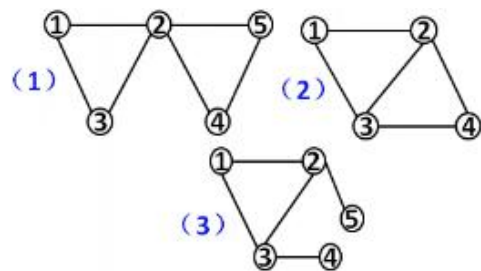
欧拉路：该路径经过图的每一条边且仅经过一次。

欧拉回路：如果欧拉路起点和终点相同，则称“欧拉回路”。

定理 1：存在欧拉路的条件：图是连通的，有且只有两个奇点或者没有奇点
(且以奇点为起点才能得到欧拉路)

定理 2：存在欧拉回路的条件：图是连通的，没有奇点

奇点：跟一个点相连的边的数目为奇数



如上图用定理判断：

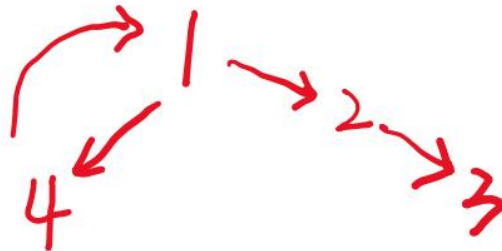
图 1 有欧拉路和欧拉回路，图 2 有欧拉路没有欧拉回路，图 3 没有欧拉路

9、有向图的欧拉路

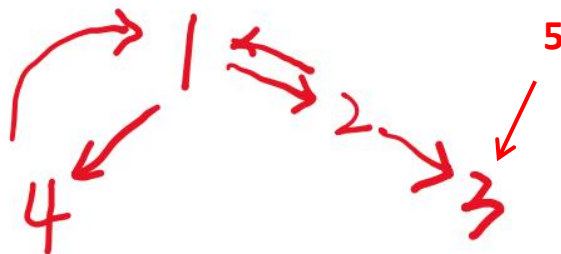
在有向图中，要计算欧拉路需要计算出度和入度

有向图的欧拉路判断：只有两个点的入度 \neq 出度，其中一个点的出度比入度多 1 (该点为起点)，另一个点的入度比出度多 1 (该点为终点)，其余点的入度和出度相等。

下图存在欧拉图



下图不存在欧拉图 (没有起点)



有向图的欧拉回路判断：每个点的入度出度都相同，任意点作为起点都可以找到回路。从 1 开始保证字典序最小。

课堂练习

1、寻找欧拉路

题目：对一个图寻找欧拉路或欧拉回路（保证）
输入格式：第一行 n, m ($n, m < 100$) 表示 n 个点， m 条边
以下 m 行为每条边连接的两点

输出格式：

欧拉路

输入样例1：

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

输出样例1：

1 2 3 4 5 1

输入样例2：

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

输出样例2：

2 1 3 2 4 3

解题思路：

$du[]$ 存储各个顶点的度

$path[]$ 存储欧拉路路径

题目中保证一定有欧拉路，所以找到奇点后进行深搜

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int mapp[110][110];
4  int path[110], du[110];
5  int len;
6  int n, m;
7  int x, y;
8  int startt = 1;
9
10 void dfs(int x){
11     for(int i=1; i<=n; i++){
12         if(mapp[x][i] == 1){
13             mapp[x][i] = 0; //删边
14             mapp[i][x] = 0;
15             dfs(i);
16         }
17     }
18     len++;
19     path[len]=x;
20 }
```

```

22 □ int main(){
23     cin >> n >> m;
24 □     for(int i=1; i<=m; i++){
25         cin >> x >> y;
26         mapp[x][y] = 1;
27         mapp[y][x] = 1;
28         du[x]++; //统计每个点的度数
29         du[y]++;
30     }
31 □     for(int i=1; i<=n; i++){
32 □         if(du[i]%2 == 1){ //判断有没有奇点存在
33             startt = i;
34             break;
35         }
36     }
37     dfs(startt);
38 □     for(int i=len; i>=1; i--){ //逆序输出
39         cout << path[i] << " ";
40     }
41     return 0;
42 }

```