上海大学计算机工程与科学学院

数据结构 (2) 个人作业报告

作业：第十周上机练习

姓名：林艺珺

学号：18120189

日期：2020 年 3 月 28 日

哈密顿路径

1. 问题描述

在图 G 中找出一条包含所有顶点的简单路径，该路径称为哈密顿路径。

2. 基本要求

（1）图 G 是非完全有向图，且图 G 不一定存在哈密顿路径；

（2）设计算法判断图 G 是否存在哈密顿路径，如果存在，输出一条哈密顿路径即可；

（3）分析算法的时间复杂度。

3. 选做

（1）哈密顿回路：在图 G 中找出一条包含所有顶点的简单回路，该回路称为哈密顿回路。

（2）当这个图是带权图时，求该图的最短哈密顿回路。

题目分析

判断哈密顿路径

根据哈密顿路径的定义，需要从一节点出发一次访问完所有结点。而学习过的深度优先遍历算法，即是通过不断访问邻接顶点实现的。从指定的结点 v 开始进行深度优先搜索的算法的步骤是：

1. 访问结点 v，并标记 v 已被访问；
2. 取顶点 v 的第一个邻接顶点 w；
3. 若顶点 w 不存在，返回；否则继续步骤 4；
4. 若顶点 w 未被访问，则访问结点 w，并标记 w 已被访问；否则转步骤 5；
5. 使 w 为顶点 v 的在原来 w 之后的下一个邻接顶点，转到步骤 3。

举个例子，以下图 1 为一非完全有向图，可判断其有且仅有一条哈密顿路径：A-B-D-C-E。对该图使用深度优先遍历算法，从结点 A 开始，可得访问顺序同样为：A-B-D-C-E。因此，判断哈密顿路径的办法为：以 n 阶非完全有向图 G 的每个结点为起点，依次进行一次深度优先遍历，若一次遍历并不是所有结点都被标记，那么以此点为起点的哈密顿路径数量为零，当所有结点都进行深度优先遍历后，路径数量仍为零的，则无哈密顿路径。

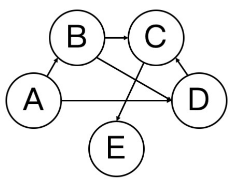

图 1: 存在哈密顿路径，但无哈密顿回路

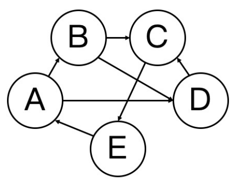

图 2: 存在哈密顿路径，也有哈密顿回路

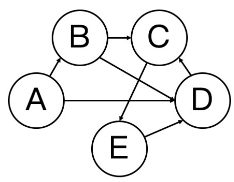

图 3: 两条哈密顿路径，没有哈密顿回路

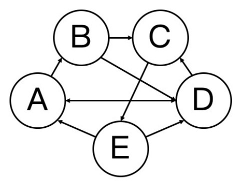

图 4: 两条哈密顿路径，两条哈密顿回路

判断哈密顿路径算法的时间复杂度

深度优先遍历本身使用递归算法，时间复杂度为 $O(n^2)$，而判断哈密顿路径时需要对 n 阶有向图的 n 个结点都进行一次深度优先遍历，因此时间复杂度为 $O(n^3)$。

判断哈密顿回路

判断哈密度回路是一个 NP 问题。

判断哈密顿回路在判断哈密顿路径的基础上，需要找到所有的哈密顿路径，然后判断路径的终点与起点是否存在方向上的连接，有则有哈密顿回路，所有哈密顿路径均无则无哈密顿回路。这种方法我未找到方式实现。

哈密顿回路的特点是，无论哪个结点作为起点，因此修改哈密顿路径的算法，在 DFS 中加入判断，当访问起点后，依然将其标注为未访问。从逻辑上来讲，这种方法一定存在遗漏的问题，但可以简单判断。

最短哈密顿回路

计算最短哈密顿回路也需要找到所有的哈密顿路径，然后判断路径的终点与起点是否存在方向上的连接，找到所有的哈密顿回路，然后进行权值相加，找出总权值最小的哈密顿回路。这种方法我未找到方式实现。

我不知道是不是我思考的方向有问题，我认为最短哈密顿回路需要求得所有哈密顿回路再进行计算。但是后来我发现可以用最短路径求解再计算最短哈密顿回路。

网络搜索相关资料的时候发现一道经典的 ACM 题——旅行商问题，其本质就是最短哈密顿回路问题。找到旅行商问题时，距离上交作业的时间已经来不及了，粗浅研究了一下算法，尝试着修改一下匹配我的数据格式，最终还是因为时间关系没有办法进行代码实现，之后会继续尝试。

主要思想如下：首先使用 Floyd 算法处理任意两个结点之间的最短距离，然后使用位运算求解，$1-(2^n-1)$ 的二进制表示代表结点 1—n(结点从 1 开始)，如 011 代表结点 1、2 被访问。dp[s][i] 表示在状态 s 中 (s 包含所有结点) 从 1 到 i 的最短路径长度，如 dp[13][3](13=1101 表示现有结点 1、3、4) 表示从结点 1 到结点 3 的最短路 (可能经过结点 4)。那么就会有状态转移方程：

$$dp[s][i] = min(dp[s][i], dp[t][j] + d[j][i])$$

其中 t 表示在当前所有结点群 s 中去除结点 i($t = s \oplus 2^{i-1}$)，d[j][i] 表示从 j(j 在 s 中) 到 i 的最短距离。在代码中，for(s=3;s<=(1«n)-1;s++) if(s&1) 先从 3 遍历到 2^n-1，且 s 一定要有结点 1。然后枚举状态 s 经过的结点 i，if(s==(1«(i-1))) 如果 s 只包含 i，这时候可以得到 $dp[s][i] = d[1][i]$；否则枚举 s 中的其他结点 j，并以 j 为中间点，求出最小的 dp[s][i]，类似 floyd 思想。

dp[s][i]=min_2(dp[s][i], dp[s^(1«(i-1))][j]+d[j][i]);

在 dfs() 中，dfs(s,id) 当前已访问的结点状态为 s，上一次递归访问结点 id，dp[s][id] 表示从 1 到 id(之间不经过 s 中的结点，1 除外) 的最短路径长度。递归边界是当 s=11⋯1，dp[s][id]=d[id][1]。枚举所有未访问的结点 i，将 i 加入状态 s 中得到 s1，dfs(s1,i)+d[i][id] 表示 1→i→id 的最短路长度。当枚举完所有的 i 后，就会得到当前 dp[s][id] 的最小值 mi，并且不需要更新。

源码程序

头文件及数据组织

```
#include <iostream>
using namespace std;
bool visited[1000];
bool exist;
int path[1000] = {−1};
int num = 0;
int n = 5;
char data[5] = {'A','B','C','D','E'};
//没有哈密顿路径
```

```
int w[5][5]= {{0,0,0,1,0},
               {0,0,1,1,0},
               {0,0,0,0,1},
               {0,0,1,0,0},
               {0,0,0,0,0}};
//有哈密顿路径, 但无哈密顿回路
int w[5][5]= {{0,1,0,1,0},
               {0,0,1,1,0},
               {0,0,0,0,1},
               {0,0,1,0,0},
               {0,0,0,0,0}};
//有哈密顿路径, 也有哈密顿回路
int w[5][5]= {{0,1,0,1,0},
               {0,0,1,1,0},
               {0,0,0,0,1},
               {0,0,1,0,0},
               {1,0,0,0,0}};
//有两条哈密顿路径, 没有哈密顿回路
int w[5][5]= {{0,1,0,1,0},
               {0,0,1,1,0},
               {0,0,0,0,1},
               {0,0,1,0,0},
               {0,0,0,1,0}};
//有两条哈密顿路径, 有两条哈密顿回路
int w[5][5]= {{0,1,0,1,0},
               {0,0,1,1,0},
               {0,0,0,0,1},
               {1,0,1,0,0},
               {1,0,0,1,0}};
```

有向无权图判断哈密顿路径算法

```
void DFS(int u, int cnt)
{
    visited[u] = true;
    path[cnt] = u;
    if (cnt == n)
    {
        exist = 1;
        return;
    }
    for (int i = 0; i < n && !exist; i++)
    {
```

```
        if(!visited[i] && w[u][i])
        {
            DFS(i, cnt + 1);
            visited[i] = false;
        }
    }
}
int main()
{
    cout << endl;
    cout << "int w[5][5]= {{0,1,0,1,0}," << endl;
    cout << "              {0,0,1,1,0}," << endl;
    cout << "              {0,0,0,0,1}," << endl;
    cout << "              {0,0,1,0,0}," << endl;
    cout << "              {1,0,0,0,0}};" << endl;
    for(int i = 0; i < n; i++)
    {
        exist = 0;
        DFS(i, 1);
        num++;
        if (exist)
        {
            cout << "一条哈密顿路径为: ";
            cout << data[path[1]];
            for (int j = 2; j <= n; j++)
            {
                cout << "–" << data[path[j]];
            }
            cout << "。 " << endl;
            break;
        }
    }
    if (!exist) {cout << "没有哈密顿路径。" << endl;}
    return 0;
}
```

有向无权图判断哈密度回路算法

```
void DFS(int u, int cnt)
{
    if (u != 0) visited[u] = true;
    path[cnt] = u;
    if (cnt == n)
```

```
    {
        exist = 1;
        return;
    }
    for (int i = 0; i < n && !exist; i++)
    {
        if(!visited[i] && w[u][i] && i != path[cnt + 1])
        {
            DFS(i, cnt + 1);
            visited[i] = false;
        }
    }
}
int main()
{
    for(int i = 0; i < n; i++)
    {
        exist = 0;
        DFS(i, 1);
        if (exist && w[path[n]][path[1]])
        {
            cout << "有哈密顿回路。路径为: ";
            cout << data[path[1]];
            for (int j = 2; j <= n; j++)
            {
                cout << "—" << data[path[j]];
            }
            cout << "—" << data[path[1]] << "。 " << endl;
        break;
        }
    }
    if (!exist) cout << "没有哈密顿回路。" << endl;
    return 0;
}
```

有向带权图计算最短哈密度回路算法

```
碍于时间关系未来得及研究与实现。
```

测试结果

判断有向无权图哈密顿路径

图 5: 样例一

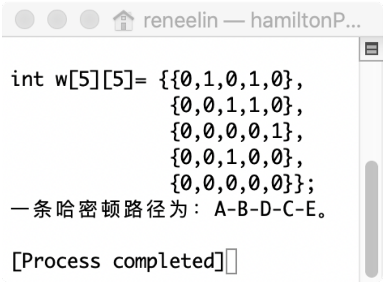

图 6: 样例二

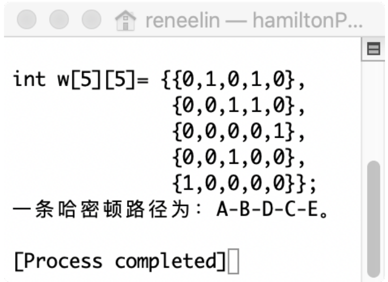

图 7: 样例三

判断有向无权图哈密顿回路

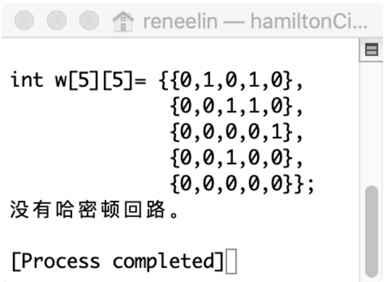

图 8: 样例一

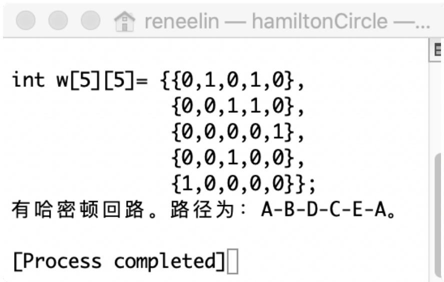

图 9: 样例二

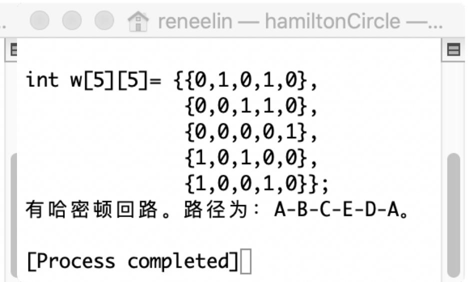

图 10: 样例三