**第3周上机实验报告**

# 一、题目：

在有向无环图DAG（Directed acyclic graph）中，哈密顿路径问题(Hamiltonian-path)可以在多项式时间内解决。请给出具体算法。

# 二、算法思路：

## 算法：

1. 对有向无环图进行拓扑排序(topological sort)
2. 验证排序结果
   1. 如果对于任意点u，以及排序结果中其下一个结点v，存在(u,v)，则哈密顿路径存在，并且就是拓扑排序返回结果
   2. 如果对于某个点u，以及排序结果中其下一个结点v，不存在(u,v)，则哈密顿路径不存在。

## 算法正确性证明：

如果每一个顶点都和下一个顶点彼此相邻，那么拓扑排序的结果为我们提供了一条哈密顿路径。如果有一个顶点和下一个顶点彼此不相邻，由拓扑排序的性质可知，不存在任何路径使两顶点相邻，于是不会存在哈密顿路径。

为了说明拓扑排序中相邻的顶点u和v，如果在图中不相邻（即没有(u,v)）便不存在从u到v的路径，采用反证法：假设存在从u到v的路径，沿着这条上的任何顶点，它们必然在u之后v之前，那么根据拓扑排序它们应该在u和v之间，这与u和v 在拓扑排序结果中相邻矛盾。因此， 不存在这样的路径。

# 三、程序设计框架：

## 核心函数的名称和功能

### 拓扑排序

使用DFS根据完成时间对结点进行排序，每当一个结点完成以后，将它插入到结果队列的开头。

1. list<**int**> topological\_sort(vector<set<**int**> >&G)
2. {
3. list<**int**> ans;
4. DFS(G,ans);
5. **return** ans;
6. }

### 验证拓扑排序的结果

遍历结果链表，判断第i个结点和第i+1个结点之间是否存在边，如果有就继续验证下一个结点，没有就说明不存在哈密顿路径。

1. **bool** judge(vector<set<**int**> >&G,list<**int**>ans)
2. {
3. **for**(list<**int**>::iterator it = ans.begin();it!=ans.end();it++)
4. {
5. list<**int**>::iterator next = it;
6. next++;
7. **if**(next == ans.end() )**return** **true**;
8. **if**(!G[\*it].count(\*next) )
9. {
10. **return** **false**;
11. }
12. }
13. **return** **true**;//make g++ happy
14. }

### DFS深度优先搜索

与普通DFS相比，在DFS-visit中，加入了在一个结点完成后将其添加进结果队列开头的操作。即增加了ans.push\_front(u)

1. **void** DFS(vector<set<**int**> >&G,list<**int**>&ans)
2. {
3. **int** size = G.size();
4. vector<**int**>color(size,0);//0 denotes white; 1 denotes black
6. **for**(**int** i = 0;i < size;i++)
7. {
8. **if**(color[i] == 0)
9. {
10. DFS\_visit(G,i,color,ans);
11. }
12. }
13. }
14. **void** DFS\_visit(vector<set<**int**> >&G,**int** u,vector<**int**>&color,list<**int**>&ans)
15. {
16. **int** size = G[u].size();
17. **for**(auto it:G[u])
18. {
19. **if**(color[it] == 0)
20. DFS\_visit(G,it,color,ans);
21. }
22. color[u] = 1;
24. ans.push\_front(u);
25. }

## 各核心函数之间的关系

拓扑排序函数由topological\_sort函数实现，其中拓扑排序由DFS算法实现，在DFS-visit中将完成后的结点添加到结果队列的开头，最终拓扑排序结果为队列ans。

验证拓扑排序结果是否构成哈密顿路径并输出结果由output函数进行。其中验证部分使用judge函数实现。

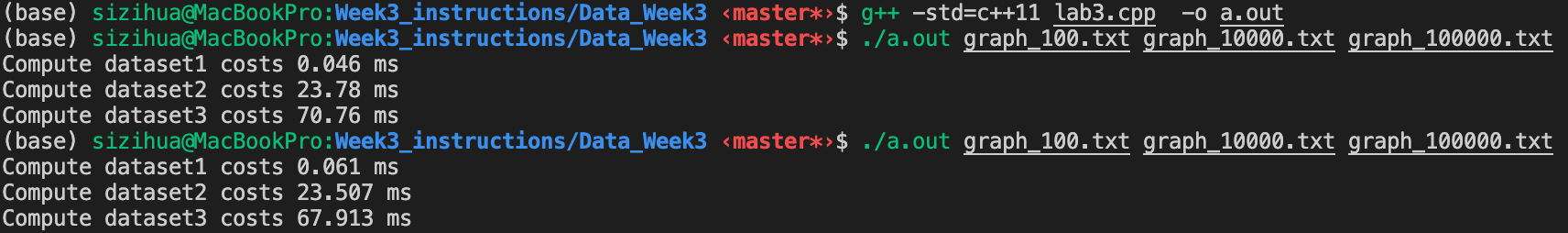
## 输入输出的格式等（输入文件请写为相对路径）

输入时，编译后运行时./a.out + 输入文件名即可，例如./a.out graph100.txt 。

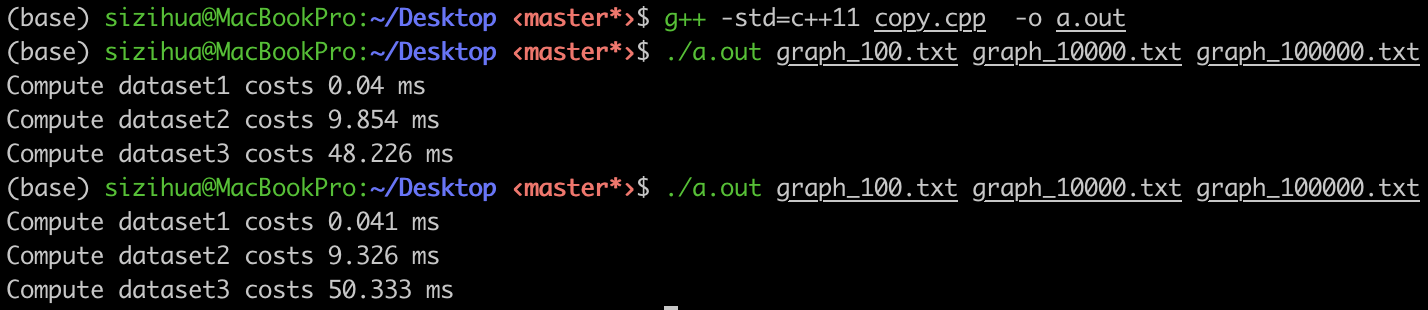
输出时，如果存在多个输入文件，输出文件按照编号与输入文件顺序对应。如ans1.txt对应第一个输入文件，ans2.txt对应第二个输入文件。

# 四、实验结果说明：

本次实验使用两种数据结构储存图实现算法以后，分别进行了测试。



图表 a：vector<set<int> >存储数据



图表 b：vector<vector<int> >存储数据

从理论上看，set和vector在stl中的实现都满足使用迭代器遍历的时间复杂度为O(n)，而且judge函数中查询点u的临接链表时，set由于底层使用红黑树实现具有O(log n)的时间复杂度，vector查询则是O(n)的，因此vector<vector<int> >实现的算法效率应该比vector<set<int> >实现的算法效率低。

然而实际测试中vector<vector<int> >的效率是比vector<set> >高的。这应该是因为vector的实现比较接近c语言中的数组，所以对于遍历操作，能够更加充分地利用cpu的cache进行加速，所以实际运行效率比较高。

# 五、个人总结（选填）

设计数据结构时，不仅要考虑理论上的时间复杂度，还需要考虑具体硬件实现时可能出现的情况。