**第5周上机实验报告**

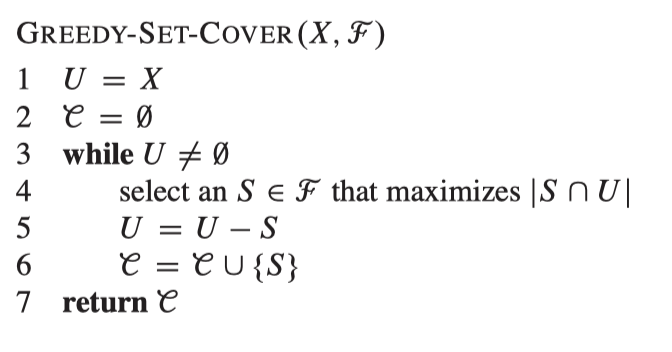
|  |  |
| --- | --- |
| 姓名： | 思子华 |
| 学号： | 2018202181 |
| 学院： | 信息学院 |
| 日期： | 2020.4.20 |

# 一、题目：

顶点覆盖问题（the set-covering problem）的贪心、近似算法实现。

2-近似算法的实现。时间复杂度为

**GREEDY-SET-COVER 伪代码**

****

# 二、算法思路：

## 具体描述算法的设计思想:

为了达到目标的时间复杂度，需要尽量降低while循环中各个操作的时间复杂度。采用以空间换时间的思路，除了用数组保存各个集合的元素以外，还维护了一些数据结构：

1. 布尔数组 vector<bool>covered: 记录结点是否已经被覆盖
2. Int类型二维数组 vector<vector<int>>node\_set: 按照结点记录其对应的所有集合
3. 链表数组 vector<list<int>>L : 根据的大小，将集合穿起来，L的size是初识图中最大子集的大小
4. Int L\_max 保存当前L中集合的最大size
5. Vector<int>set\_length 记录每一个集合中的元素的数量

**对应伪代码**

* U用vector<bool>covered 实现，U–S 是将中的元素均标为covered, 显然这是的。
* Select S 选取L中记录的最大size的集合中的第一个，即选取L[L\_max].front()。在L\_max维护好的情况下，这个操作是
* 用一个vector<int>储存最终结果的子集的集合的序号，res.push\_back(L[L\_max].front())是

于是，实现该算法的关键是如何维护和L以及L\_max。

**维护策略：**

* 对于选出的集合, 遍历其中每一个元素，单独进行维护
  + 对于的任意元素m，使用node\_set访问每一个包含m的集合
  + 将每一个包含m的集合的大小减1
    - Set\_length[j]的值减1
    - 将L中存储的（位置记作L[t]），移动到L[t-1]
* 最后更新L\_max
  + 若L中原L\_max处有元素，则不变
  + 否则，将L\_max减1，直到L\_max处有元素

## 算法时间复杂度分析

根据上述内容可知，伪代码中对应部分的实现是。

**下面证明****“维护”部分是。**

* 由于每一次维护部分作用了个点，只需证明每一个点的操作都是,便可以证明**“维护”部分是。**
* 单独考虑一个点，一个点被常数个集合覆盖，根据这个点去修改常数个集合的size，并且调整这些集合在L中的位置，这是O(1)的。

# 三、程序设计框架：

## 核心函数的名称和功能

### Greedy\_set\_cover()

满足伪代码架构的实现，update为“维护”数据结构部分的实现。

1. vector<**int**> greedy\_set\_cover()
2. {
3. vector<**int**> res;
4. **while**(n > 0)
5. {
6. n = n - L\_max;
8. res.push\_back(L[L\_max].front());
10. update(L\_max);
11. }
13. **return** res;
14. }

### Update

遍历选中的集合，对于每一个元素更新包含它的集合的大小以及该集合在L中的位置。最后，更新L\_max（当前所有集合中最大的size）

1. **void** update(**int** set\_size)
2. {
3. **int** index = L[set\_size].front();
4. L[set\_size].pop\_front();
6. **for**(auto vertex : S[index])
7. {
8. decrease(vertex);
9. }
11. **while**(L\_max>=0&&L[L\_max].empty())
12. {
13. L\_max--;
14. }
15. }

### Decrease()

利用node\_set，遍历该包含结点的所有集合，将该集合的size减1，再移动它在L中的位置。

1. **void** decrease(**int** vertex)
2. {
3. **if**(covered[vertex]) **return**;
5. covered[vertex] = **true**;
7. **for**(auto S\_i : node\_set[vertex])
8. {
9. **for**(list<**int**>::iterator l = L[ set\_length[S\_i] ].begin();l != L[ set\_length[S\_i] ].end();l++)
10. {
11. **if**(\*l == S\_i)
12. {
13. L[ set\_length[S\_i] ].erase(l);
14. **break**;
15. }
16. }
17. set\_length[S\_i] --;
18. L[ set\_length[S\_i] ].push\_back(S\_i);
19. }
20. }

## 各核心类、各核心函数之间的关系

----> 选中

greedy\_set\_cover ----> 更新 解集

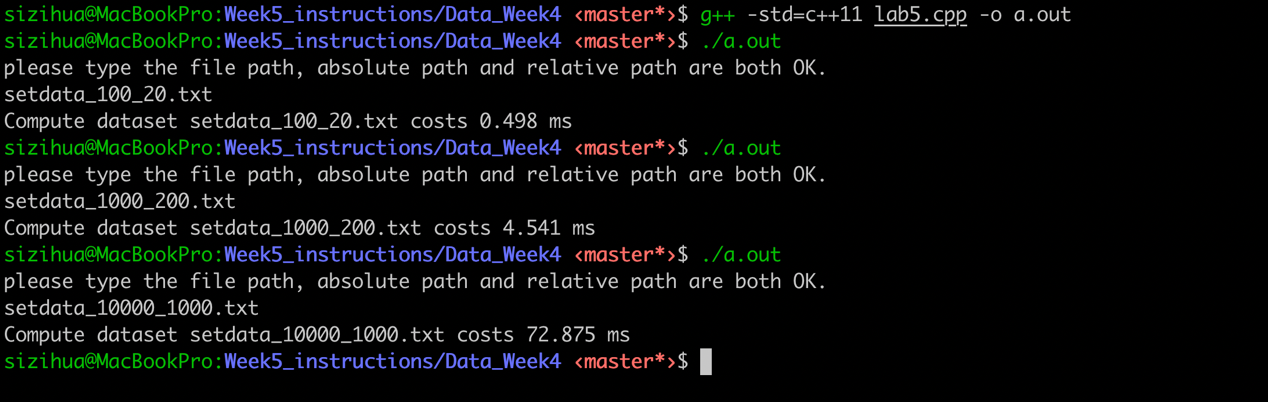
----> 维护数据结构update ----> 包含结点的每一个集合decrease

## 输入输出的格式等（输入文件请写为相对路径）

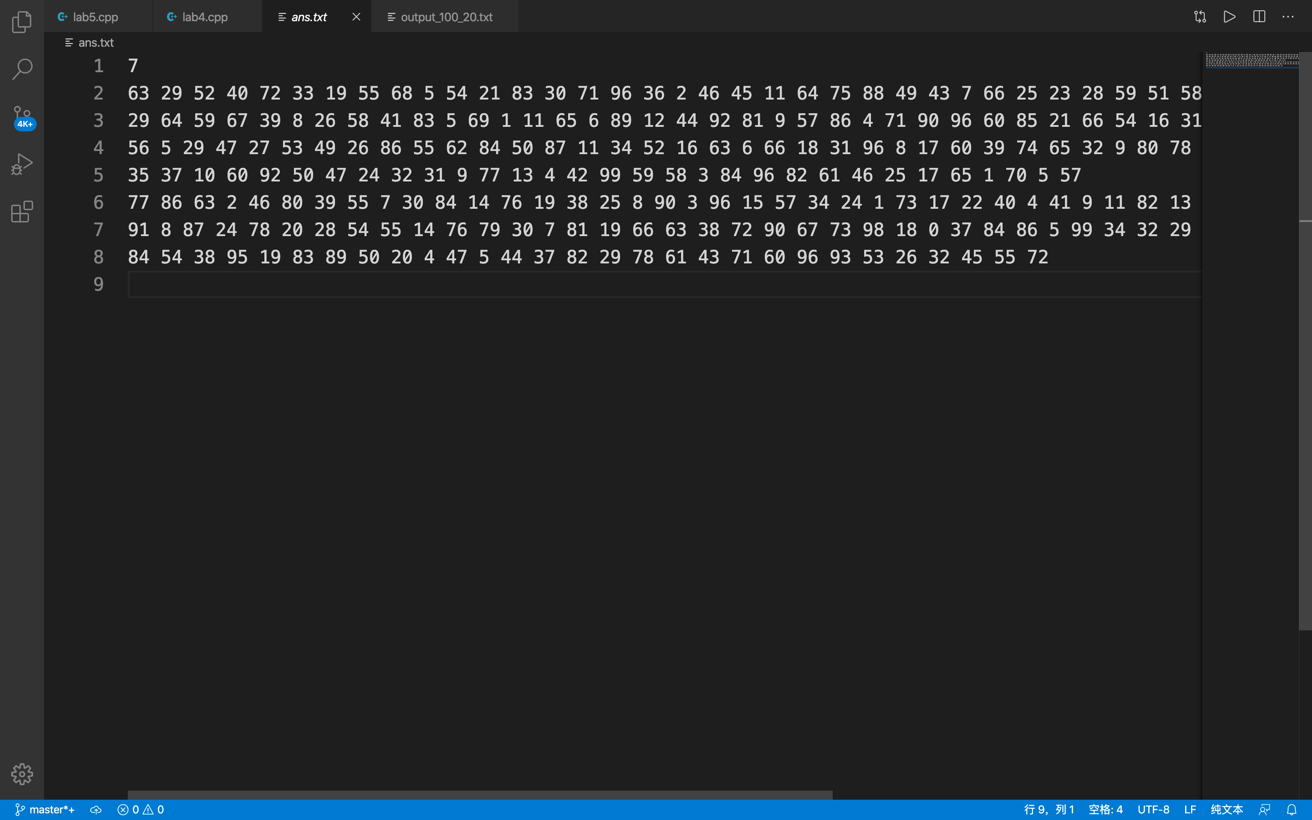
程序使用fstream中的open()函数打开文件，故输入格式为相对、绝对均可。

# 四、实验结果说明：

## （1）实验结果截图



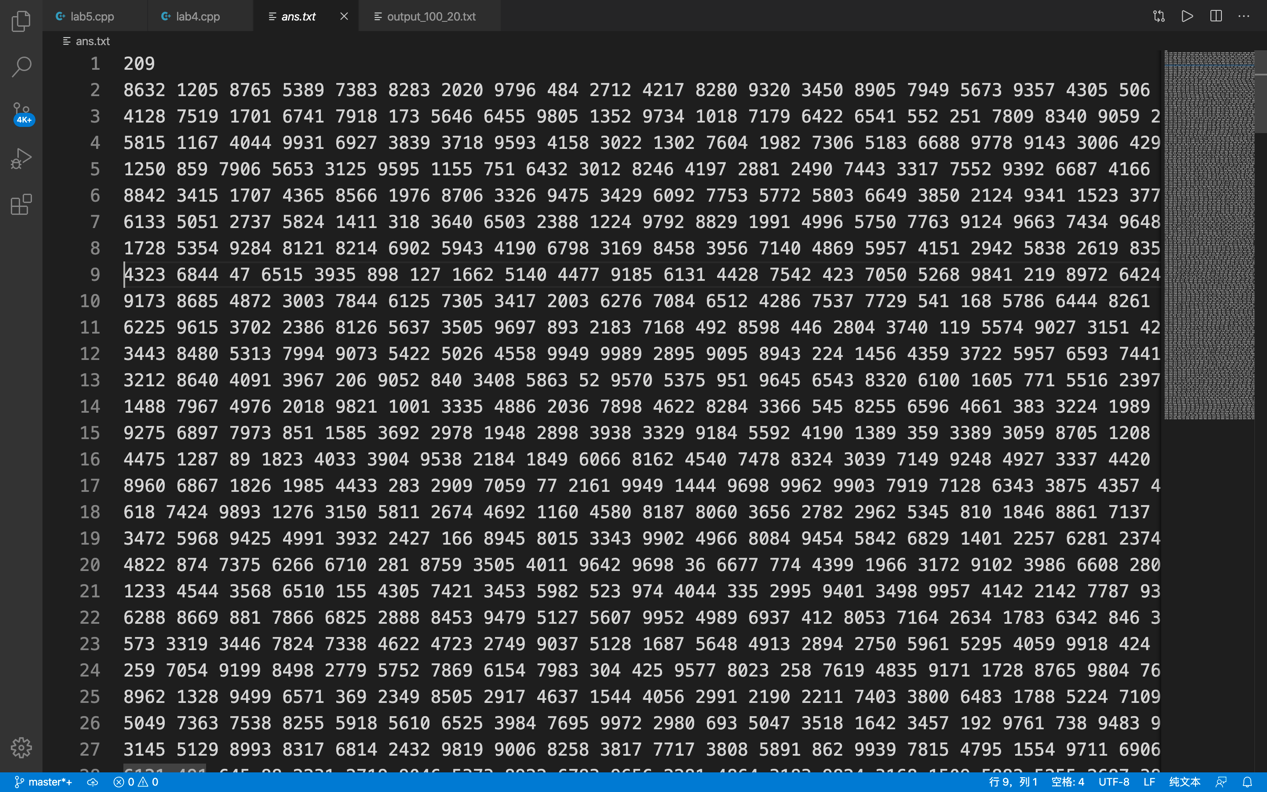
图表 a：运行程序



图表 b：运行结果1



图表 c：运行结果2



图表 d：运行结果3

## （2）实验结果的分析

从运行时间可以看出，随着输入的setdata以10倍的比例增大，运行时间也是以10倍左右的速率增大，验证了算法的运行时间复杂度是

# 五、个人总结（选填）

本次实验的实现，如果使用stl中的set，通过求交集等方式来实现，是非常容易写代码的，诚然，这种实现空间复杂度也比较低，但是set的各种操作的时间复杂度最终算下来是无法达到要求的。

最终采取的方法的空间复杂度较大，大概是O(nm)的（n代表结点数目，m代表集合数目）。这体现出算法很难同时在时间和空间复杂度上都表现良好，本实验便是用空间换时间。