**第4周上机实验报告**

# 一、题目：

图G = (V,E) 的独立集（independent set）是子集 使得E中的每条边至多与中的一个顶点相关联。**独立集问题**是要找出G中具有最大规模的独立集。

当G为二分图时，给出一个多项式时间算法来求解独立集问题。

## 知识点解释

* 二分图 (bipartite graph)：有点资料也叫作二部图。它是如果一个图可以分为两个子集X和Y，且X和Y通过有边连接。通俗点说，就是X的每一个边的另一个端点只能是Y里的一个顶点，Y的每个边的另一个端点只能是X的一个顶点。
* 最大匹配（maximum matching）：在二分图中，最大匹配包含的边（连接X中顶点x和Y中顶点y的边）是所有匹配中边数最多的。
* 完全匹配（perfact matching）：在二分图中，有一个连接X和Y的匹配，（现在主要关注X），如果X中所有的顶点都在Y中有对应的匹配，而不管Y中是否所有的点都在X中是否有匹配，我们把这样的匹配叫作（从X到Y的）完全匹配，也叫作饱和X的匹配。完全匹配需要|X|<=|Y|，就是需要X的顶点的个数少于Y的顶点的个数。 注：匈牙利算法所求出的是二分图的最大匹配，而该最大匹配不一定是完备，所以也不一定是完美匹配。
* 最小顶点覆盖(minimum vertex cover)：实质是个点集，点集里面的点能覆盖所有的边，最小顶点覆盖就是满足这个要求的点集中点数最小的那个。

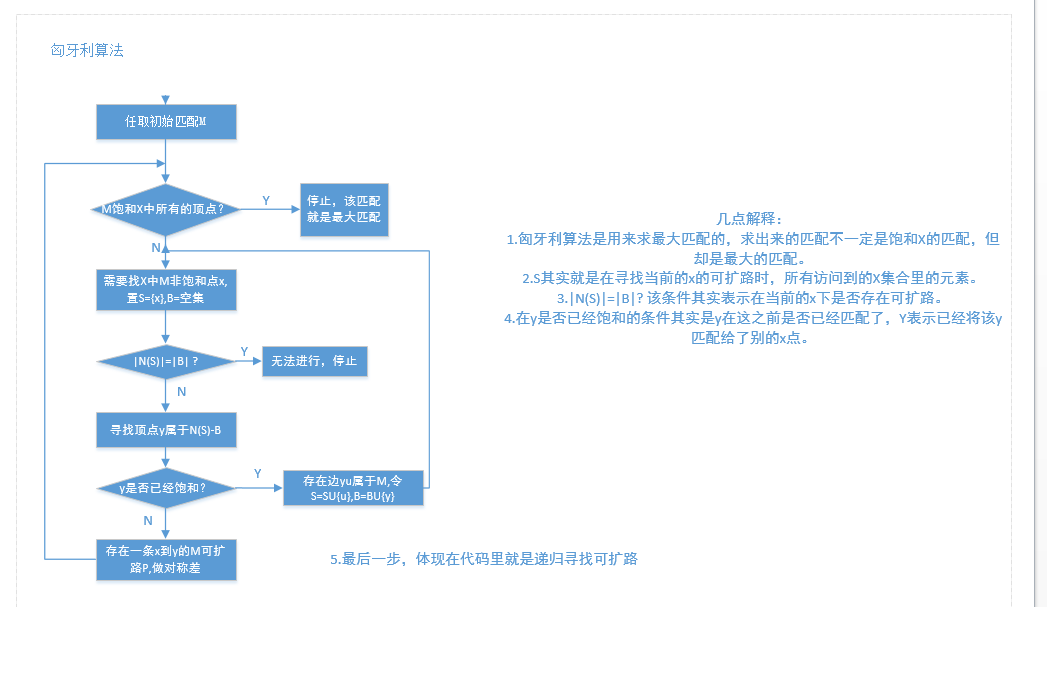
# 二、算法思路：

## 整体流程

* 使用匈牙利算法寻找最大匹配。
* 最小点覆盖集的大小等于最大匹配的大小，根据匹配结果**标注**出最小点覆盖集的顶点。
* 将二分图中属于最小点覆盖集的顶点去掉，剩下的就是一个最大独立集。

## 匈牙利算法思路

* 匈牙利算法寻找最大匹配，就是通过不断寻找原有匹配M的增广路径，因为找到一条M匹配的增广路径，就意味着一个更大的匹配M' , 其恰好比M 多一条边。
* 对于图来说，最大匹配不是唯一的，但是最大匹配的大小是唯一的。



图表 a：匈牙利算法流程图[[1]](#footnote-1)

## 标注最小点覆盖集思路

用一个例子进行说明：

* 红色边为最大匹配选中的匹配边
* 红色点为最小点覆盖选中的点

我们可以看到任何一条边都包含一个红点，这就是二分图的最小点覆盖。

图表 b：最小点覆盖集示例

**标注过程**：

* 先做最大匹配（匈牙利算法）
* 之后我们每次从左边不在匹配边中的一个点开始去按照：未匹配边 -> 匹配边 -> 未匹配边 -> 匹配边 …… 匹配边与未匹配边交替选择的顺序，标记途中经过的顶点，则最后一条经过的边必定为匹配边。
* 这里我开了两个数组x\_mathch和y\_match分别记录二分图左侧部分（left\_part）是否被标记和右侧部分（right\_part）是否被标记，上面的例子中3、4、B被标记。但最小点覆盖的点集并不是这些点组成的集合，而是左侧未被标记的点和右侧被标记的点组成的集合S才是最终答案，例子中为1、2、B。
* 最大独立集是图中所有非最小点覆盖集的点

## 算法正确性证明

### 二分图中：最大独立集 = 所有顶点 – 最小点覆盖集

最小点覆盖的点覆盖了所有的边，因此任意一条边上都有一个点属于最小点覆盖集。所以移除这些点，每条边上就只剩下一个顶点了，这些点自然就是独立集。由于选取的是最小点覆盖集，所以剩下的集合便是最大独立集。

### 二分图中最小点覆盖集的点数等于最大匹配数

实际上，这是图论中的柯尼希定理。本课为算法课，在此不展开了。

### 标注出的点为最小点覆盖集[[2]](#footnote-2)

记选出的点集为P，最大匹配的边集为E。

* P覆盖了所有的边（反证法）：
  + 假设有一条边左右两端点均不在P集合中，则左端点被标记，右端点未被标记。
  + 分类讨论，如果说当前边不在边集E中，则左端点为未被匹配点，又因为当前边不是匹配边，所以必然会从左端点沿当前边开始标记，那么右端点必然被标记，矛盾。如果说当前边在边集E中，其左端点被标记，那么必然是通过右端点链接过来的，则右端点被标记，矛盾。
  + 综上，我们可以知道点集P覆盖了所有边。
* ｜P｜=｜E｜：
  + 点集P中的左侧点都是匹配点，而右侧点也都为匹配点，又因为一条匹配边上左侧点和右侧点不可能同时在或者不在点集P中，那么E集合中任意一条边左右两端点中都恰好仅有一个点在点集P中，所以|P| = |E|。
* ｜P｜是最小点覆盖数：
  + |P| = 最大匹配数，如果另一个点集的元素个数比|P|更小，那么这个点集必然无法包含所有的匹配边，连所有的匹配边都无法全部包含，怎么可能包含所有边呢。所以|P| 是最小的点覆盖数。

## 时间复杂度

读入文件是(E)的。

匈牙利算法是（nE）的，即（）

标注结点是 ,即

综上，算法是的。

# 三、程序设计框架：

## 核心函数的名称和功能：

### 最大独立集

* 使用匈牙利算法寻找最大匹配
* 根据最大匹配标注最小点覆盖集

1. **int** MaxIndependentSet()
2. {
3. **int** ans = hungary();
4. mem\_set\_false(x\_match);
5. **int** m = y\_match.size();
6. **for**(**int** i =0;i < m;i++)
7. {
8. **if**(y\_match[i]!=-1) x\_match[y\_match[i]] = **true**;
9. }
10. **for**(**int** i = 0;i < m;i++)
11. {
12. **if**(!x\_match[i])mark(i);
13. }
14. **return** m - ans;
15. }

### 匈牙利算法

使用DFS实现的匈牙利算法

1. **int** find(**int** x)
2. {
3. **if**(x == -1)**return** 0;
4. **if**(x\_match[x] == 1) **return** 0;
5. x\_match[x] = **true**;
6. **int** n = bipartite\_graph[x].size();
7. **for**(**int** i = 0;i < n;i++)
8. {
9. **int** y = bipartite\_graph[x][i];
10. **if**(y\_match[y]==-1 || find(y\_match[y]))
11. {
12. y\_match[y] = x;
13. **return** 1;
14. }
15. }
16. **return** 0;
17. }
18. /\*匈牙利算法\*/
19. **int** hungary()
20. {
21. **int** ans = 0;
22. **int** n = bipartite\_graph.size();
23. **for**(**int** i = 0;i < n;i++)
24. {
25. mem\_set\_false(x\_match);
26. ans+=find(i);
27. }
28. **return** ans;
29. }

### 标注函数

1. /\*标记最小点覆盖集 左侧部分为被标记的点 右侧被标记的点是最小点覆盖集\*/
2. **void** mark(**int** x)
3. {
4. **if**(vx[x]) **return**;
5. vx[x] = **true**;
6. **int** n = bipartite\_graph[x].size();
7. **for**(**int** i =0;i < n;i++)
8. {
9. **int** y = bipartite\_graph[x][i];
10. **if**(y\_match[y]>=0 && !vy[y])
11. {
12. vy[y] = **true**;
13. mark(y\_match[y]);
14. }
15. }
16. }

## 各核心函数之间的关系：

---------> hungary() -------> find()

MaxindependentSet()

---------> mark()

## 输入输出的格式等（输入文件请写为相对路径）：

执行./a.out以后在命令行中输入文件路径即可。

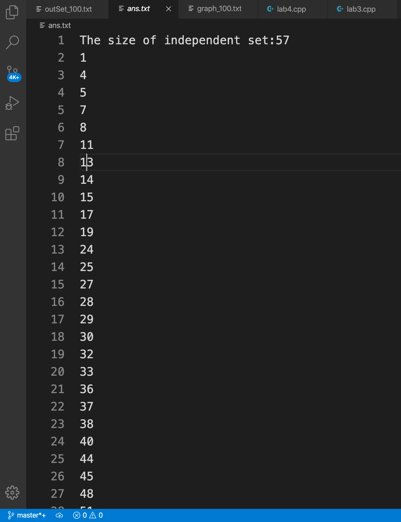
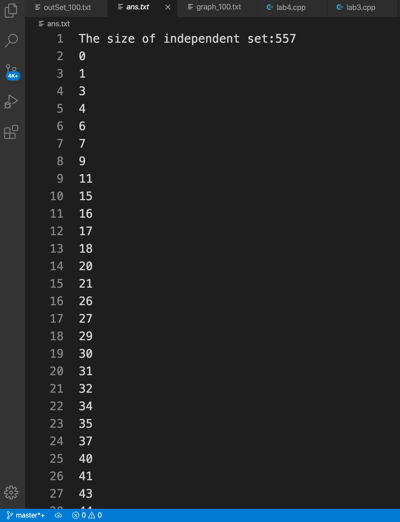
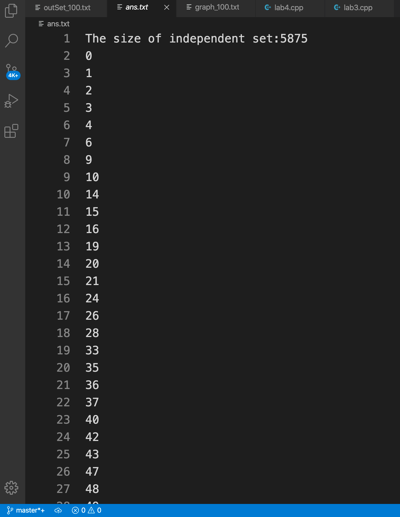
根据个人的环境，具体输入即可。使用string保存输入路径，用fstream的open函数打开，所以相对路径，绝对路径均可。

# 四、实验结果说明：

## 实验结果截图



图表 c：命令行运行

图表 d：运行结果

（2）实验结果的分析

随着输入规模10倍的扩大，运行时间基本也在1000倍的扩大，所以基本满足的结论。

1. 源自网络：<https://blog.csdn.net/michaelhan3/article/details/51706380> [↑](#footnote-ref-1)
2. 源自网络：<https://blog.csdn.net/SkeletonKing233/article/details/104923374> [↑](#footnote-ref-2)