**演算法程式作業四**

110403518 林晉宇

1. **Maximum Flow**

這次作業實作找出最大流的兩種方法，並設計測資比較兩種方法的耗時。第一種為使用Ford-Fulkerson，在尋找augmenting path時，每次找使flow增加最大的路徑；第二種為使用Edmonds-Karp，與前者的差異在於尋找augmenting path時，使用bfs來確保找到的路徑為最短路。

1. **Ford-Fulkerson**

**以下為Ford-Fulkerson的步驟:**

1. 建圖。
2. 找flow值最大的增廣路徑，如果有:
   * 1. 找出該路徑的path flow。
     2. 對該路徑的每一邊減去path flow，並建反向邊。
     3. 將path flow值加入答案。
     4. 繼續找下一個增廣路徑。
3. 當沒有增廣路徑時，中止程式。

**資料結構:**

1. 由於點數(n)最大只會到100，所以我選擇使用鄰接矩陣存圖(**G**)。
2. Vis陣列代表每點是否被訪問過；parent陣列存每一點的祖先(即從哪裡來的)；dis[v]陣列代表從源點到v所經過的邊最大的權重。

**如何找出當下flow值最大的增廣路徑:**

我的作法與dijkstra演算法雷同，只是在初始化以及鬆弛操作的條件判斷不太一樣。首先，dis陣列一開始會初使化為0，而鬆弛操作的條件判斷為**if(dis[v] < G[u][v] and !vis[v])**，即如果v點可以從其他邊權更大的邊過來，這時就更新v點的dis以及parent。以下為pseudo code。

int parent[] -> -1, dis[] -> 0; //初始化

bool vis[] -> 0; //初始化

priority\_queue<pair<int,int>> pq; //max heap

pq.push({dis[0], s}); //0代表dis, s代表源點

while(!pq.empty()){

    int w, u = pq.pop();

    if(vis[u])  continue;

    vis[u] = true;

    for(int v = 所有 u 指向的點){

        if(dis[v] < u->v的權重 and !vis[v]){ //relaxing

            dis[v] = u->v的權重;

            parent[v] = u;

            pq.push({dis[v], v});

        }

    }

}

1. **Edmonds-Karp**

**以下為Edmonds-Karp的步驟:**

1.建圖。

2.用bfs找增廣路徑，如果有:

* + 1. 找出該路徑的path flow。
    2. 對該路徑的每一邊減去path flow，並建反向邊。
    3. 將path flow值加入答案。
    4. 繼續找下一個增廣路徑。

3.當沒有增廣路徑時，中止程式。

**資料結構:**

1.由於點數(n)最大只會到100，所以我選擇使用鄰接矩陣存圖(**G**)。

2.Vis陣列代表每點是否被訪問過；parent陣列存每一點的祖先(即從哪裡來的)。

int parent[] -> -1; //初始化

bool vis[] -> 0; //初始化

queue<int> q;

q.push(s);

while(!q.empty()){

    int u = q.pop();

    if(vis[u])  continue;

    vis[u] = true;

    for(int v = 所有u指向的邊){

        if(!vis[v] and u->v邊權大於0){

            parent[v] = u;

            q.push(v);

        }

    }

}

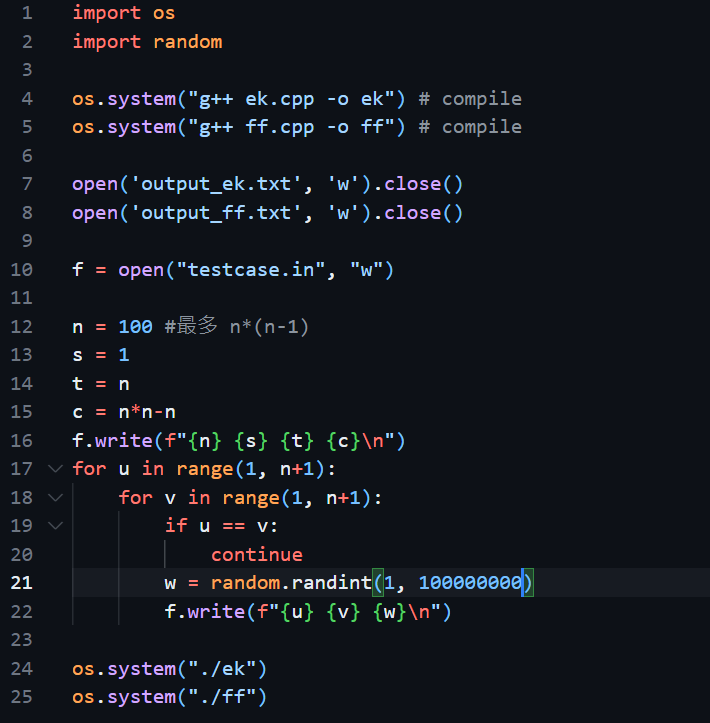
1. **分析複雜度**

**Ford-Fulkerson: O(m\*) (M為cut中邊數最多的值, \*為最大flow值)**

**Edmonds-Karp:** **O(nm\*2)**

1. **產生測資**

以下為產生測資的python程式碼:

****

由於測資的範圍不大，所以我將點及邊的數量直接設成最大值，即100點以及9900條邊，邊權範圍為1~1e8。

1. **結果**

透過上述python程式進行兩種方法的比較，我執行了兩百次，最後平均兩者的執行時間:

1. Ford-folkerson: **約0.02375秒**
2. Edmond-karp: **近乎0秒**

可以看到雖然ford-folkerson每次找augmenting path時，是找path最大的路徑，看似比較省時，但實際上結果卻是edmond-karp的用bfs找最短路徑比較快，當然另一種原因可能是ford-folkerson這樣的做法還會受到flow的大小影響。總之，兩種做法對於n最大只有100的情況下，程式運行的時間差別並不大，我認為都是十分有效率的。