演算法程式作業四

110403518 林晉宇

一、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。

二、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:
    i. 找出該路徑的 path flow。
    ii. 對該路徑的每一邊減去 path flow，並建反向邊。
    iii. 將 path flow 值加入答案。
    iv. 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});
        }
    }
}
```

三、Edmonds-Karp

以下為 Edmonds-Karp 的步驟:

1. 建圖。
2. 用 bfs 找增廣路徑，如果有:
    i. 找出該路徑的 path flow。
    ii. 對該路徑的每一邊減去 path flow，並建反向邊。
    iii. 將 path flow 值加入答案。
    iv. 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);
        }
```

```
    }
}
```

四、分析複雜度

Ford-Fulkerson: O(m*$\log_{M/(M-1)} f*$)（M 為 cut 中邊數最多的值，* 為最大 flow 值）

Edmonds-Karp: O(nm*2)

五、產生測資

以下為產生測資的 python 程式碼:

```python
import os
import random

os.system("g++ ek.cpp -o ek") # compile
os.system("g++ ff.cpp -o ff") # compile

open('output_ek.txt', 'w').close()
open('output_ff.txt', 'w').close()

f = open("testcase.in", "w")

n = 100 #最多 n*(n-1)
s = 1
t = n
c = n*n-n
f.write(f"{n} {s} {t} {c}\n")
for u in range(1, n+1):
    for v in range(1, n+1):
        if u == v:
            continue
        w = random.randint(1, 100000000)
        f.write(f"{u} {v} {w}\n")

os.system("./ek")
os.system("./ff")
```

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。

六、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。