**程式設計研討專題4**

**110403518 林晉宇**

1. **題意**

**UVA 10480 Sabotage**，給n個點(n<=50)，m條邊(m<=500)，求將1跟2分開的最小割，並輸出割包含的邊集合。這是一題解minimum-cut (maximum-flow)的問題，第一種作法會先使用一般的flow方法去解題，第二種方法實作隨機合併的做法，最後比較兩者的時間及第二種Randomize的準確率。

1. **使用Flow解題**

* **Max-flow / Min-cut 演算法:**

用來解最大流最小割(max-flow min-cut)的演算法主要有三種:

1. **Ford-Fulkerson**為三者中最早提出，時間複雜度為**O(fm)** (f為max-flow, m為邊數)
2. **Edmonds-Karp**，可以視為改進版的Ford-Fulkerson，時間複雜度為**O(m^2 n)**，不會受到max-flow值所影響
3. **Dinic**，複雜度為 **O(m n^2)**，一般邊的數量都大於點的數量，所以通常這比Edmonds-Karp快。

這邊我使用**Dinic**做法，可以直接求出max-flow(min-cut)值，而要判斷哪些邊為割的集合，就看邊的兩個端點是不是在同一群即可，如果不是(即原邊的一點跟1同一群，另一點跟2同一群)，則屬於割的邊集合，輸出該邊。

* **Dinic步驟**:

1. 一開始residual graph(殘量網路)即為原圖。
2. 循環:
   1. 用bfs去找當前residual graph的level graph(分層網路)。
   2. 用dfs去找當前level graph上的augmenting path(擴增路徑)，即一條可以從源點走向匯點的路徑。**[如果找不到擴增路徑，就終止循環]。**
   3. 更新residual graph，包含更新邊權、刪除飽和(邊權為0)的邊、建反向流量。回到第一步驟。

* **Pseudo code:**

1. **Dinic**

*int dinic()*

*ans <- 0*

*while(bfs()) //有增廣路徑*

*while(d <- dfs(s,inf)) //去找增廣路徑*

*ans+=d;*

*return ans;*

1. **bfs找分層圖(level graph)**

*ll bfs() //find level graph*

*dis[] <- {-1}*

*dis[1]=0*

*queue q*

*q.push(1)*

*while(!q.empty())*

*ll u=q.front(); q.pop();*

*for(遍歷u點的邊集合)*

*ll v=edge[i].v*

*if(dis[v]==-1&&edge[i].flow)*

*dis[v]=dis[u]+1; //dis 紀錄該點為分層圖的第幾層*

*q.push(v);*

*return dis[t]!=-1; //如果等於-1代表 就沒有augmenting path 可以直接終止循環*

1. **dfs找擴增路徑(augmentint path)**

ll dfs(ll u,ll flow) //find augmenting path

    if(u==t)    return flow; //如果當前該點為匯點 直接return

    for(遍歷u的邊集合)

        ll v=edge[i].v;

        if(dis[v]==dis[u]+1&&edge[i].flow)

            ll d=dfs(v,min(edge[i].flow,flow));

            if(d>0)

                edge[i].flow-=d;

                edge[i^1].flow+=d;

                return d;

    return 0;

1. **使用Randomized做法解題**

* **Karger algorithm:**

Karger是隨機算法，相比前面的flow演算法，優點是速度快，實現簡單，但未必找到的解一定是最小割。這裡用karger演算法來處理s-t cut的問題。

* **Karger步驟:**

1. 一開始Contracted graph即為原圖。
2. 當還有超過2點在contracted graph上時:
   1. 隨機在Contracted graph挑一個邊(u,v)，但此邊不能是(s,t)。
   2. 將u跟v兩點合併成一點，並更新contracted graph。
   3. 移除自環。
3. 最後剩下的兩點所連的邊即為最小割的可能答案。

* **實作方式:**

寫額外的struct來存圖的邊(edge)，而每當有兩點合併時，用並查集(Disjoint set union)去做處理，有加上路徑壓縮(data compression)以及按秩合併(union by rank)。

* **Pseudo code:**

1. **Karger**

*int Karger(struct Graph\* graph)*

*{*

*並查集初始化(將每個點的祖先設成自己)*

*v <- 點數*

*while(v>2)*

*{*

*隨機選一邊*

*if(邊的兩點屬於同一集合)    continue;*

*else if(邊的兩點包含1跟2) continue;*

*else*

*{*

*v--;*

*union();將兩點集合合併*

*}*

*}*

*cutedge <- 0;*

*for 遍歷所有邊*

*{*

*subset1 <- 當前邊所屬集合1*

*subset2 <- 當前邊所屬集合2*

*if(兩集合不同)    cutedge+=邊權;*

*}*

*return cutedge;*

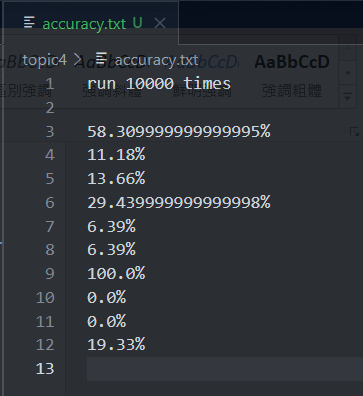
*}*

1. **比較**

* **Randomized 的準確率:**

測資使用UVA 10480原題的測資，使用網路上教的方法([連結](https://blog.ujoj.cc/post/get-uva-testdata/))，寫python把隱藏測資偷過來([測資連結](https://docs.google.com/document/d/1RHa_cQmDX8zxu3ttshhBZ7pDnOB7ULKNmuinEMPSb2k/edit?usp=sharing))，UVA這題總共有十筆資料，以下對比不同資料的準確率。(以dinic算出的答案作為標準答案，拿karger算出的結果來比對)。

|  |  |  |  |
| --- | --- | --- | --- |
| 資料 | 點數/邊數 | 邊權範圍 | 準確率(karger跑10000次) |
| 第一筆 | 5/10 | 100以內 | 58.3% |
| 第二筆 | 5/6 | 100以內 | 11.18% |
| 第三筆 | 5/6 | 100以內 | 13.66% |
| 第四筆 | 5/4 | 100以內 | 29.44% |
| 第五筆 | 5/8 | 100以內 | 6.39% |
| 第六筆 | 5/8 | 100以內 | 6.39% |
| 第七筆 | 2/1 | 100以內 | 100% |
| 第八筆 | 50/497 | 非常極端大部分100以內，但少數大至40000000 | 0% |
| 第九筆 | 50/497 | 非常極端大部分100以內，但少數大至40000000 | 0% |
| 第十筆 | 50/500 | 五位數以內 | 19.33% |



可以看出karger隨機算法在面對點數及邊權小的資料時，可能還有機會算出最小割，但基本上只要少數邊權非常極端或點數及邊數增加，準確率就會慘不忍睹。

* **時間比較:**

使用python產生隨機測資(連通圖)，以下為實驗的結果:

|  |  |  |
| --- | --- | --- |
| 資料筆數 | Flow (Dinic) | Randomized(Karger) |
| 200筆 | 0.035秒 | 0.021秒 |
| 2000筆 | 0.18秒 | 0.144秒 |
| 20000筆 | 1.795秒 | 1.225秒 |

可以看出Karger整體比flow快大約30%左右。

1. **結論**

這次報告讓我對flow中算最大流最小割的三種演算法有更深入的了解，也實際時做了dinic做法，而透過與隨機演算法karger的比較，可以發現雖然karger可能比較容易實作及速度較快，但準確率蠻低的，除非點數、邊數及邊權範圍較小及相近，準確率才有可能比較高。

1. **資料來源**
2. [Youtube - 13-1: 网络流问题基础 Network Flow Problems](https://www.youtube.com/watch?v=6DFWUgV5Osc&list=PLvOO0btloRnsbnIIbX6ywvD8OZUTT0_ID&index=9)
3. [Youtube - 13-2: Ford-Fulkerson Algorithm 寻找网络最大流](https://www.youtube.com/watch?v=8sLON0DqLZo&list=PLvOO0btloRnsbnIIbX6ywvD8OZUTT0_ID&index=9)
4. [Youtube - 13-3: Edmonds–Karp Algorithm 寻找网络最大流](https://www.youtube.com/watch?v=l-5W0ffPsDo&list=PLvOO0btloRnsbnIIbX6ywvD8OZUTT0_ID&index=10)
5. [Youtube - 13-4: Dinic's Algorithm 寻找网络最大流](https://www.youtube.com/watch?v=mtxzaGFLIAo&list=PLvOO0btloRnsbnIIbX6ywvD8OZUTT0_ID&index=11)
6. [Geeksforgeeks - Karger’s algorithm for Minimum Cut](https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/)