演算法程式作業四

110502529 陳柏燊

EK Pseudo code:

```
path //adjenecy-matrix also as residual flow
nei //adjececy-list record neighbor use c++ uordered_set implement

while input(u,v,w):
    path[u][v] += w //since may have different edge such that have same (u,v)
    nei[u].inert(v)
    nei[v].inert(u)
```

(data structure and input process)

```
parent[|V|] //record parent
def bfs():
    for i=1 to V: parent[i] = 0
    queue.push({source, inf})

    while !queue.empty():
        nowd, nowf = queue.pop()
        for i in nei[nowd]:
            if path[now][i] > 0 and !parent[i]:
                parent[i] = now
                if i == sink:
                    return min(nowf, path[now][i])
                queue.push({i, min(nowf, path[now][i])})

    return 0
```

(use bfs to find the shortest path)

bfs will update parent and will return the max flow of the shortest path

```
ans = 0
while(flow = bfs()):
    ans += flow
    cur = sink
    while (cur != source):
        from = p[cur];
        path[from][cur] -= flow;
        path[cur][from] += flow;
        cur = from;

print ans
```

(main Ford-Fulkerson process)

Use bfs to find if there exit any augmenting path, if yes, add it to ans and update path

Time complexity O(nm*(n + m)) = O(nm^2)

Maxpath Pseudo code:

since data structure and main Ford-Fulkerson is same as I did in ek so I won't paste it again.

Let focus on how to find the maxflow of all augmenting path.

```
def maxpath():
    priority_queue<flow, point> q
    // sort by flow from big to little

    maflow[|V|]
    // record the max flow from source of the residual graph

    q.push(inf, source)
    maflow[source] = inf

    while !q.empty():

        nowf, nowd = q.pop()

        if nowf != maxflow[nowd]: continue

        for x in nei[nowd]:
            if path[nowd][x] > 0 and maxflow[x] < min(nowf, path[nowd][x]):
                parent[x] = nowd
                maflow[x] = min(nowf, path[nowd][x])
                q.push(maflow[x], x)

        return maxflow[sink]
```

My algorithm to find the max flow Is modify from Dijkstra algorithm which is to solve shortest path.

Focus on maxflow, since we want to find the "max" flow so we set it initial value to 0 not to inf.

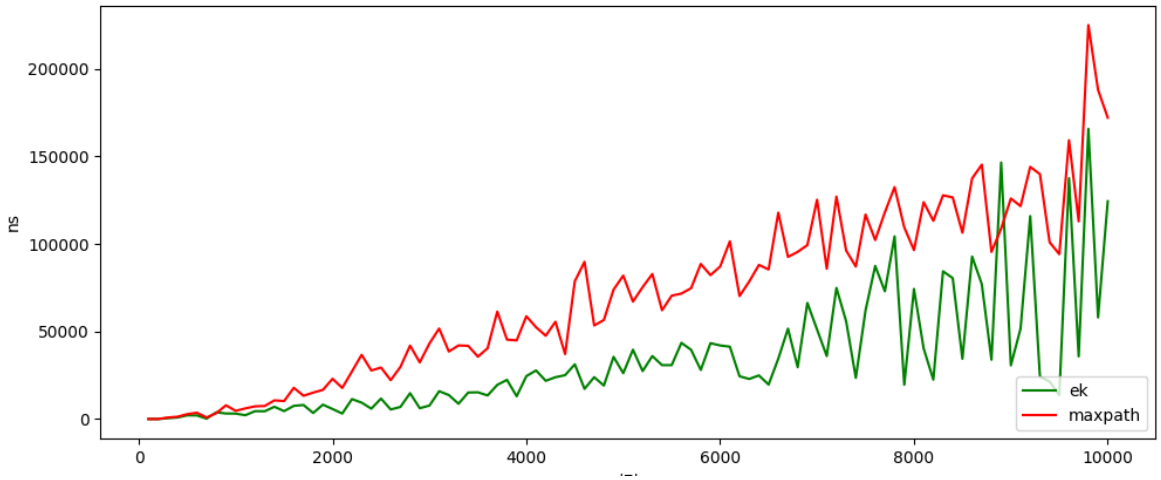
Also focus on the for loop in the while statement, we can see the condition in the if statement, it maxflow[x] "<" min(nowf, path[nowd][x]) since "max" flow so it's < not >.

Time complexity O($1 + \log_{M/(M-1)} f^*$ *(ElogV))

相當於找一最長路徑 P 長度為 $= \mathbf{Min}_{e \in P}\, w(e)$,
上式中 f^* 為最大 flow 的值, M 為 cut 中邊數最
多的數值 (see Sedgewick's book).

Designed input and compare time

N = 100 M = 100 ~ 10000



觀察圖可以發現 maxpath 比 ek 慢，我測了好多次都這樣，我想最主要的原因應該是，用 random 產生邊權，無法做到產生大量回邊，這導致了 ek，通常不會做到 worst case，所以都比 maxpath 來的快。