演算法第四次程式作業

改進地方：

原本的演算法是跑V-1次，每次看E個邊去做relax，如 果在看第n次時都沒有發生relax ( n < V-1 )，那表示n之後的點也不用看了。

Pseudo code：

Bellman-Ford(G,s){

Initialize(G, s) ;

for i =1 to V-1 do

int a=0;

for each edge uv ∈ E do

if d[v] > d[u] + w(u, v)

d[v] = d[u] + w(u, v) ;

π[v]=u ;

a=1 ;

if a == 0

break ;

}

紅字為跟原本不同處。

輸出 Negative weight cycle 的方式：

跑完V-1次後會再跑一次確認是否可以再relax，如果可以代表有負迴圈，發生在可以relax的那個點，從那個點開始往它的parent找，直到找回一開始那個點，中間所經過的點即為負迴圈的點。

分析加速版的演算法複雜度：

Best case：跑第一次就break了，也就是 1\*E次，時間複雜度為O(E)

Worst case：跟原本的演算法一樣跑到V-1次，也就是 (V-1)\*E，時間複雜度為O(V\*E)

Average case：假設在某次break的機率都相等，也就是

(E+2E+….+(V-1\*E)) / (V-1) = (E+(V-1)\*E) / 2 = (V\*E) / 2 次，時間複雜度為O(V\*E)