ADA2021 Homework3

Problem5

討論對象：

```
b06303131 沈家睿
b08501098 柯晨緯
```

1. 我們只要直接輸出$deg(r)$即可，當使用adjacency-list時，要找到$deg(r)$只需要花$O(1)$時間。由於root所連到的子樹之間不可能有edge存在，否則就會形成一個cycle，且除了子樹之外就不會有其他connected component，不然就會有disconnected的部分，所以移除root之後的connected component數量就會是root所連接之子樹數量，也就是$deg(r)$。
2. v的子樹之間必不存在連接彼此的edge，否則這兩個互相連接的子樹在做dfs時應該會形成一個子樹；此外，v的子孫也不可能和深度$\leq depth(v)$的vertex有相連，否則就會產生v的子孫與祖先相連接的情形。因此計算$S(v,G)$直接output $deg(v)$即可，時間一樣是$O(1)$。(在此的深度是指dfs-tree中的深度，離root越遠深度越大)
3. 定義：$f(w_i) = \begin{cases} 1, & \text{if } up_T(w_i) = depth(v) \\ 0, & \text{otherwise} \end{cases}$，則$S(v,G) = \Sigma_{i=1}^{k} f(w_i) + 1$，因為如果$v$的child w的up_T小於$depth(v)$，代表當v移除後，$D_T(w)$都包含於v的parent所屬的component；而如果w的up_T等於$depth(v)$，則代表當v移除後，$D_T(w)$自己屬於一個component，因為$D_T(w)$內沒有任何一個點連到depth比$depth(v)$小的點；而up_T必不可能大於$depth(v)$，因為v的children必與v相連。綜合上述，如果$up_T = depth(v)$就要對$S(v,G)$加1，否則就不用，最後還要再多考慮v的parent所屬的component，故要再多加1。
4. 任意選一個點作為root r往下dfs(以遞迴實作)，過程中記錄每個vertex的depth以及其parent(depth為以dfs-tree的path去計算每個點離root的距離；root的parent為null)，如果traverse到一個vertex v，與其相連的點都已經visited，那在v就遍歷其所有相連點並找出相連點的depth之最小值，將那個值記錄在$up_T(v)$並且return該值；如果traverse到一個vertex v，與其相連的點沒有全部都被visited，那就對那些未visited的點做dfs，並取得它們return的up_T值，將這些up_T值和其他與v相連但在traverse v前就已經visited過的點之depth值做比較，將最小值設為$up_T(v)$並且return這個值；在此總共會花$O(|V|+|E|)$的時間，因為每個點都會去檢查接在其上的edge兩次。再來花$O(|V|)$的時間去檢查誰的parent是root，並把結果記錄於$S(r,G)$，而root以外的每個點v都都去檢查其所連接到的child之up_T並以subproblem(3)的方法計算其$S(v,G)$值，可以把v的adjacency-list遍歷一次並且跳過v的parent即可完成這個步驟，因為每個點都去遍歷自己的adjacency-list，故此步驟總共花$O(|V|+|E|)$的時間。所以計算所有vertex的$S(v,G)$共花$O(|V|+|E|)$時間即可。因為root在G裡的edge，相較其在dfs-tree裡，只會多back edges，而有沒有這些back edge並不會對拔掉root之後產生的connected component數量有所影響 (因為把root拔掉時這些back edges會跟著消失)，因此$S(r,G) = S(r,T)$ (T為G的dfs-tree)，所以計算$S(r,G)$只須知root在dfs-tree裡的degree即可。而其他點的$S(v,G)$的正確性就如同subproblem(3)所說的一樣。故得此演算法為正確，且時間複雜度為$O(|V|+|E|)$。

Problem6

討論對象：

```
b06303131 沈家睿
b08501098 柯晨緯
```

1. 此題可以直接使用Krusakal's algorithm，但是要在一開始sort edge的時候，由width大排到小，由於是使用disjoint set來檢查兩個點是否已經在同個component (同時要加上path compression和union by size)，且最多只有$2|E|+|V|-1$次的operation，故花在操作disjoint set上的時間為$O((2|E|+|V|-1)\alpha(|V|))$，其中$\alpha$是一個成長的很慢的函數，而sort邊花$O(E\log(E))$的時間，由於$E \leq V^2$，所以總共複雜度最多$O(E\log(V))$的時間。因為是先由大到小sort所有edges，可以知道每次取的edge都是目前所有剩餘的edge裡不會造成cycle的最大權重之edge，這樣的greddy choice便能使最終得出的tree為maximum spanning tree，此演算法正確性由此可知。
2. 假設maximum spanning tree(T)上存在兩點s,t，他們之間的widest path不在T上。我們定義s,t之間在T的path為P，把P上width最小的edge移除，此時會產生兩塊connected component C_1, C_2各自包含了s和t。此外我們定義s,t之間真正的widest path為P'，將P'上連接C_1, C_2的edge加在T上之後，由於多加上去的edge之width必大於被移除的edge(根據widest path定義得知)，我們可以得到wdith總和更大的tree，故T就不是一棵maximum spanning tree，得到矛盾。因此我們可以知道，在maximum spanning tree上，任兩點的path皆是widest path。由於找到真正的maximum spanning tree所需要花的時間就是Kruskal's algorithm所花的時間，且tree上edges的數量是保持所有點connected所需的最小數量，因此只需使用該演算法即可在$O(E\log(V))$時間即可找出E'。
3. **Claim:** A road $(u,v) \in E$ is downwards critical. $\Longleftrightarrow$ There is a shortest path from s to u or v that ends at (u,v).
   - $\Rightarrow$: 我們可以不失一般性地假設當$e=(u,v)$是downwards critical時，所有s到v的shortest path不會經過e。令s到v的shortest path總長為b，s到u的shortest path總長為a，由shortest path性質可知$a+d(e)>b$，此時如果要讓$d(e)$減少使得s到v的shortest path總長減少，$d(e)$至少需降到等於$b-a$，故沒有一個點$v \in V$距離s的shortest path會因$d(e)$減少任意正實數ϵ而其總長必因此下降，此與downwards critical的定義違背，可知當(u,v)是downwards critical時，s到u或v的shortest path一定存在一條會結束於(u,v)。
   - $\Leftarrow$: Trivial，不失一般性假設s到v的shortest path結束於$e=(u,v)$，這使得當$d(e)$任意減少一個正值都會使s到v的shortest path總長下降。

4. **Claim:** A road $(u,v) \in E$ is upwards critical. $\Longleftrightarrow$ All the shortest paths from s to u or v end at (u,v).
   - $\Rightarrow$: 我們可以不失一般性假設當$e=(u,v)$是upwards critical時，存在一條s到v的shortest path不會經過e，令其為P。令s到v的shortest path總長為L，此時P沒有經過e，因此無論$d(e)$增加多少，都不會因此改變s到v的shortest path總長，故沒有一個點$v \in V$距離s的shortest path會因$d(e)$增加任意正實數ϵ而其總長因此增加，此與upwards critical的定義違背，可知當(u,v)是upwards critical時，所有s到u或v的shortest path必結束於(u,v)。
   - $\Leftarrow$: Trivial，不失一般性地假設所有s到v的shortest path都會結束於$e=(u,v)$，這使得$d(e)$任意增加一個正值都會使s到v的shortest path總長增加。

5. 使用Dijkstra's algorithm先找出shortest path tree，同時在演算法進行的過程中，不斷地在relaxation的過程中去維護一個table M，M中記錄了每個點距離s的shortest path總長。由於Dijkstra's algorithm使用min-heap去實作，對min-heap的operation次數為$O(E)$，而每次operation所花的時間為$O(\log V)$的時間，故Dijkstra's algorithm所花時間為$O(E\cdot \log V)$。之後我們先建立兩個空的set D, U，分別儲存屬於downwards critical的edge以及屬於upwards critical的edge，並且建立一個大小為$V.size$的table π，$\pi[i]$為i這個vertex在與s的shortest path上，所有可能的parent。接著我們需要遍歷每個vertex $v \in V$，對每個點v，去檢查所有$u \in adj[v]$，只要$M[u] + d((u,v))$等於$M[v]$，便將(u,v)加入D (因爲代表(u,v)個edge是s到v的其中一條shortest path的最後一段road，因此由subproblem3的結論可知其屬於downwards critical)，並將u加入$\pi[v]$；直到每個$v \in V$都執行完以上操作後，我們再遍歷一次所有$v \in V$，只要$\pi[v].size$為1就將$(\pi[v][0], v)$加入U (因爲代表$(\pi[v][0], v)$這個edge是所有s到v的shortest path的最後一段road，因此由subproblem4的結論可知其屬於upwards critical)。執行完上述操作後，D, U內所存的edges就分別是所有downwards critical和upwards criticals了。在遍歷所有點並檢查與其相連的所有邊的過程，我們花了$O(|V| + |E|)$的時間。故整體演算法所花的時間是$O(E\log V)$。

6. 先將題目要找的環之條件改寫：

$$\frac{\Sigma_{i=1}^{n} k(v_i)}{\Sigma_{i=1}^{n} d(v_i, v_{i+1})} > K \;\Rightarrow\; K\cdot \Sigma_{i=1}^{n} d(v_i, v_{i+1}) - \Sigma_{i=1}^{n} k(v_i) < 0$$

$$\Rightarrow (K\cdot d(v_1,v_2) - \tfrac{1}{2}k(v_1) - \tfrac{1}{2}k(v_2)) + (K\cdot d(v_2,v_3) - \tfrac{1}{2}k(v_2) - \tfrac{1}{2}k(v_3)) + \cdots + (K\cdot d(v_n,v_1) - \tfrac{1}{2}k(v_n) - \tfrac{1}{2}k(v_1)) < 0$$

我們可以先將所有$(u,v) \in E$的weight改為$d'(u,v) = K\cdot d(u,v) - \frac{1}{2}k(u) - \frac{1}{2}k(v)$，並且用Bellman-Ford algorithm去找在每個edge的weight更新為d'之後是否存在negtive cycle，更新所有edge的weight只花$O(E)$，而Bellman-Ford algorithmn所花時間為$O(VE)$；且由以上數學式推倒可知，因為reweight之後如果有負環，代表該環滿足$\frac{\Sigma_{i=1}^{n} k(v_i)}{\Sigma_{i=1}^{n} d(v_i, v_{i+1})} > K$ (其中$v_1, v_2 \cdots v_n$為該環上所有vertex) ，因此我們只需$O(VE)$的時間即可成功找到題目所求之環是否存在。