Homework 3

B06303131 經濟五 沈家睿

December 15, 2021

Reference:
b08501098 柯晨緯 & b08902149 徐晨祐
b08502041 李芸芳 & b07207063 廖政華 & b09902068 凌暄

Problem 5 - Shi-Hei Robots

(1) 直接輸出 deg(r) 即可
因爲 T 爲一 DFS Tree 因此若移除接在 r 之 edge
即可得到 components 數
因爲 T 是 Tree 因此子樹間不會存在 edge
否則即形成 cycle
因此 deg(r) 即爲 s(r, T)

(2) 直接輸出 deg(v) 即可
分爲兩個 case 說明之
1. 若 v 有若干子樹存在於 T 時
任一子樹皆不可能存在邊相連
因若存在子樹相連的邊則會產生 back edge 連回 v
即發生與題目假設之矛盾
因此 deg(v) 即爲 s(v, G)
2. 任何 >depth(v) 之點皆不能與 $\leq$depth(v) 之點相連
(depth(x) 之定義爲 DFS Tree 中 root 至 x 的深度)
因爲若 >depth(v) 之點與 $\leq$depth(v) 之點相連
亦會產生 back edge 與題目假設出現矛盾
因此 deg(v) 即爲 s(v, G)

(3) 所求爲 $1+\sum_{i=1}^{k}(up_T(w_i) == depth_T(v))$
證明如下
因爲 $up_T(w_i)$ 之最大值爲 $depth_T(v)$
因此只會有兩種 case:
1.$up_T(w_i) < depth_T(v)$
若在此 case 下即代表存在一點 $x \in D_T(w_i)$
與 v 之 parent 所屬之 component 相連
因此若分離 v 則不會對 component 數有所影響
2.$up_T(w_i) == depth_T(v)$
若在此 case 下
則代表 $\forall x \in D_T(w_i)$ 皆不會與 v 之 parent 所屬之 component 相連
因此 $D_T(w_i)$ 可視爲一 component
因此在 $up_T(w_i) == depth_T(v)$ 時 component 數要加一
在最後額外的加一是來自 v 之 parent 所屬的 component

(4) 以遞迴實作 DFS 求解之
對 V 上任一點 s 做 DFS
graph 上之 vertex 內含有
1.depth: 從 s 至此點之深度
2.up: 即爲 (3) 所定義之 $up_T()$
3.parent: 此點之 parent id
4.adjacency list
5.color:
白: 未拜訪過
灰: 還未更新 up 值時
黑: 更新完 up 值並 return
此一 DFS 會記錄每點之 depth 並且更新 up 值
走訪方向爲往 color 是白色點往下做 DFS
走訪過去即將 color 轉爲灰色
要 return 前需將 color 轉爲黑色且計算此點 up 值
up 之計算方式分爲兩個 case
遍歷其 adjacency list 看各點之 color 狀態
1. 若存在至少一黑色點
根據所有黑色點之 up 值
以及此點之所有 adjacency node 的 depth 值取最小
即爲此點之 up 值
2. 若所有點皆爲灰色
此點之所有 adjacency node 的 depth 值取最小
即爲此點之 up 值
而此 DFS 所花時間爲 O(|V|+|E|)
且根據 DFS 會得到一 DFS Tree T
接著計算 deg(s,T)(deg(s, T) 爲 deg(s) in T)
用以 s(s,G) 之計算
s(s,G) 之答案即爲 deg(s,T)
原因爲 deg(s,G)=deg(s,T)+nums of back edge
而 back edge 數對求 componenet 不會有影響
因爲拔掉 s 後其所有 back edges 皆會消失
因此直接輸出 deg(s,T) 即可
而除了 s 以外之點 v 皆可利用其 adjacency node 之 up 值
以及 (3) 得到之結論可得 s(v, G)
在算 s(v,G) 時因有紀錄其 parent id
因此在遍歷 v 之 adjacency list 可跳過其 parent node
而算 s(v,G) 的複雜度爲 O(|V|+|E|)
綜上所述總時間複雜度爲 O(|V|+|E|)

Problem 6 - Lost in Pekoland

(1) 可以利用 Kruskal’s algorithm 求解
將 Kruskal’s algorithm 第一步之 sort 改爲由大到小排序
則每次從最大 weight 的邊開始取
剩餘之檢查是否有環之步驟皆保持不變
因由大到小取且保證不形成 cycle 都不變
這樣的 greedy choice 便可得到 maximum spanning tree
且本質上仍在跑 Kruskal’s algorithm
因此時間複雜度爲 $O(E \log V)$

(2) claim:maximum spanning tree 中從 s 到任意點 t 必含其 widest path
使用矛盾證法
假設在原 maximum spanning tree T 上
存在 s 到 t 之 path P 並不是其從 s 到 t 之 widest path
則必可將 P 之 minimal width e 移除
使 T 分割爲兩個 connected component
則因 s 到 t 存在一 widest path P'
因此必可在 P' 找到一邊 e' 使兩個 connected component
形成一棵新的樹 T'
而 T' 之 width 和高於 T
因此矛盾於 T 爲一 maximum spanning tree
故得證
而此題依然是使用 Kruskal’s algorithm 求解
因此時間複雜度爲 $O(E \log V)$

(3) 定義一函數 cost(x): 從 s 至 x 之 shortest path cost
令 e = (u, v)
($\Rightarrow$):
claim: 若 e 具 downwards critical
則有一 shortest path 從 s 到 u or v 且結束於 e
使用矛盾證法
假設所有從 s 到 u or v 之 shortest path 皆沒結束於 e
且 e 具 downwards critical
則 WLOG 假設 cost(v)+d(e) > cost(u)
則根據 downwards critical 之定義令減少之正實數爲 ϵ
則 cost(v)+d(e)-ϵ< cost(u)
移項後可得 d(e)-(cost(u)-cost(v))< ϵ
則可知 ϵ 必大於 d(e)-(cost(u)-cost(v))
矛盾於 downwards critical 之定義
故得證
($\Leftarrow$):
claim: 若有一 shortest path 從 s 到 u or v 且結束於 e
則 e 具 downwards critical
因 e 位於 shortest path 上
則若 e 之 cost 減少任意正實數
必至少使 cost(u) or cost(v) 降低
因此 e 滿足 downwards critical 之性質
故得證

(4) 令 e = (u, v)
claim:
e $\in$ E 且 e 具 upwards critical $\Longleftrightarrow$
所有從 s 到 u 或 v 之 shortest path 必含 e
($\Rightarrow$):
claim: 若 e 具 upwards critical
則所有 shortest path 從 s 到 u or v 必含 e
使用矛盾證法
存在一 shortest path 從 s 到 u or v 不含 e 且 e 具 upwards critical
則 WLOG 假設存在一條 shortest path P' 從 s 到 v 不經過 e
因 e 具 upwards critical
因此任意增加一正實數 ϵ 可使至少一 shortest path cost 增加
若 e 之 cost 增加
則到 v 之 shortest path 仍爲 P'
而到其他點之 shortest path 之 cost 也不變
因此矛盾於 e 具 upwards critical 之定義
故得證
($\Leftarrow$):
claim: 所有 shortest path 從 s 到 u or v 且必含 e
則 e 具 upwards critical
定義一函數 cost(x): 從 s 至 x 之 shortest path cost
因 e 位於所有從 s 到 u or v 之 shortest path 上
因此若 e 之 cost 增加任意正實數
必使 cost(u) or cost(v) 增加
因此 e 滿足 upwards critical 之性質
故得證

(5) 使用兩次 Dijkstra's algorithm 求解之
跑完第一次 Dijkstra 時可得每點從 s 至任一點之 shortest path cost
並使用一表格 Min Cost Table 將各點之 shortest path cost 紀錄下來
則此步驟花費 $O(E \log V)$ 時間
使用另一張表 Predecessor Table
記錄各點形成 shortest path 之 predecessor attribute
而跑第二次 Dijkstra 時
可在每點 relaxation 時同時比較 Min Cost Table 同點之值
若相同則將此點之 predecessor attribute 加入 Predecessor Table 中
則此步驟依然花費 $O(E \log V)$ 時間
接著根據 (3) 及 (4) 之結論
最後只要遍歷一次 Predecessor Table 中的每一點 u
將各邊加入 upwards critical 或 downwards critical 之集合回傳即可
若 u 只有一點 v 爲其 predecessor attribute
則將 (u, v) 加入 upwards critical 及 downwards critical 之集合
反之若 u 之 predecessor attribute 有 k 點且 k > 1
則將 (u, v_i) $i \in [1, k]$ 加入 downwards critical 之集合
而此步驟花費 O(V+E)
因此總時間複雜度爲 $O(E \log V)$

(6) $\frac{\sum_{i=1}^{n} k(v_i)}{\sum_{i=1}^{n} d(v_i,v_{i+1})} > K$

$=> \sum_{i=1}^{n} k(v_i) > K \sum_{i=1}^{n} d(v_i, v_{i+1})$

$K \sum_{i=1}^{n} d(v_i, v_{i+1}) - \sum_{i=1}^{n} k(v_i) < 0$

每條邊之 weight $d(v_i, v_{i+1})$

根據此公式 $d'(v_i, v_{i+1}) = K \cdot d(v_i, v_{i+1}) - \frac{k(v_i)+k(v_{i+1})}{2}$

reweight 成 $d'(v_i, v_{i+1})$

則 $K \sum_{i=1}^{n} d(v_i, v_{i+1}) - \sum_{i=1}^{n} k(v_i) < 0$

$= \sum_{i=1}^{n} d'(v_i, v_{i+1}) < 0$

每條邊有了新的 weight $d'(v_i, v_{i+1})$ 且有上述之不等式
則可利用 Bellman-Ford algorithm 來偵測圖上是否具有負環
也就是此題之 exciting cycle
而全部邊 reweight 所花時間爲 O(E)
Bellman-Ford algorithm 所花時間爲 O(VE)
因此總時間複雜度爲 O(VE)