Αλγόριθμοι και πολυπλοκότητα
Εργασία: 3η Προγραμματιστική

Διδάσκοντες: Παγουρτζής Α., Σούλιου Δ.,Φωτάκης Δ.

Ονοματεπώνυμο: Σεβαστού Νικολέτα
Α.Μ.: 711514 22 00015
email:nikolesev@gmail.com

ΑΛΜΑ

2023-2024

Άσκηση 1

Λύση

Algorithm 1 maxkruskal($G = (V(G), E(G)), w, N, M$)

```
1: // Συνολική πολυπλοκότητα O(M log M)
2: A ←emptyset()
3: // V(G) = {1, ..., N}
4: for e ∈ E(G) do
5:     t(e) = −w(e)  // αντίθετο από τη στιγμή κλεισίματος w(e) της ακμής e
6: end for
7: Q′ ←newQueue()// ουρά προτεραιότητας ελαχίστου με βάση τα κλειδιά w(·)
8: Q ←newQueue()// ουρά προτεραιότητας ελαχίστου με βάση τα κλειδιά t(·)
9: Q ← E(G)
10: for i = 1 to N do
11:     makeset(i)                                  // σύνολο που περιέχει το i
12: end for
13: i ← 0
14: while Q ≠ ∅ do
15:     e = {a, b} ←dequeue(Q)// έξοδος του ελάχιστου σε προτεραιότητα
        στοιχείου
16:     if set(a)≠set(b) then                  // set(a):εύρεση αντιπροσώπου
17:         i ← i + 1
18:         Enqueue(e, Q′)
19:         // Union(a,b):ένωση συνόλων όπου ανήκουν τα a, b και ορισμός
              κοινού αντιπροσώπου
20:         Union(a, b)
21:     end if
22: end while
23: return Q′, i
```

Algorithm 2 Transport($G = (V(G), E(G)), w, N, M, K$)

```
1: (Q, i) ←maxkruskal(G = (V(G), E(G)), w, N, M)
2: // Q είναι ουρά προτεραιότητας με τις ακμές του παραγόμενου δέντρου
     μεγίστου βάρους
3: if i ≤ k then
4:     print('infinity')
5: else
6:     for i = 1 to K + 1 do
7:         e ←dequeue(Q)
8:     end for
9:     tmax← w(e)
10:    return tmax
11: end if
```

Άσκηση 2

Λύση

Algorithm 3 President(G, N, M, A, B, K, c, T)

```
// G :αναπαράσταση γραφήματος με vector G (όπως στη C++) μεγέθους 1×N. Το λ-οστο κελί αντιστοιχεί στην κορυφή λ και έχει περιεχόμενο ένα vector G[λ] μεγέθους όσο οι γείτονες της κορυφής. Κάθε κελί G[λ][i] περιέχει ένα διδιάστατο πίνακα. G[λ][i][1] έχει τον γείτονα. G[λ][i][2] έχει την απόσταση μεταξύ λ και του i οστού γείτονα
// c :πίνακας διαστάσεων 1 × K
for v ∈ V(G) do
    d(v) ← ∞      // πεδίο για χρόνο άφιξης στην πόλη v από αφετηρία A
end for
d(A) ← T
TIME← newMatrix(1, K)
// TIME(i) είναι ο χρόνος άφιξης στην πόλη της στήλης i του προέδρου
// και είναι NULL αν δεν την επισκεφθεί.
t ← 0
for i = 1 to K − 1 do
    len=length(G[c(i)])
    for j = 1 to len do
        if G[c(i)][j][1] = c(i + 1) then
            TIME[c(i)] ← t
            t ← t + G[c(i)][j][2]
            // G[c(i)][j][2]: Βάρος ακμής των δύο διαδοχικών πόλεων
               c(i), c(i + 1) στο δρομολόγιο του προέδρου
        end if
    end for
end for
SHOULD_I_PASS(G, B,TIME)
return d(B)
// Συνολική πολυπλοκότητα O((K − 1)N + M log N)
```

Algorithm 4 SHOULD_I_PASS(G, B,TIME)

```
1: // Συνολική πολυπλοκότητα O(M log N)
2: Q ←newQueue()// ουρά προτεραιότητας ελαχίστου με βάση τα κλειδιά d(·)
3: Q ← V(G)
4: v ←dequeue(Q)
5: while v ≠ B do
6:     len← length(G[v])
7:     for i = 1 to len do
8:         u ← G[v][i][1]
9:         if TIME(u) < TIME(v) then
10:            maxi← v
11:            mini← u
12:        else
13:            maxi← u
14:            mini← v
15:        end if
16:        // περνά ο πρόεδρος από κάποια κατεύθυνση
17:        if TIME(u) ≠NULL & TIME(v) ≠NULL&(TIME(mini)≤
           d(v) ≤(TIME(maxi)-1)) then
18:            c ← 2G[v][i][2]                    // περιμένω
19:        else          // Δεν περνά ο πρόεδρος από κάποια κατεύθυνση
20:            c ← G[v][i][2]
21:        end if
22:        if d(u) > d(v) + c then
23:            d(u) ← d(v) + c
24:        end if
25:    end for
26:    v ←dequeue(Q)
27: end while
```

Πηγές

1. Vector in C++ STL