Εργασία: Ανάλυση και Σχεδιασμός Αλγορίθμων

Panagiotis Koutris   Christos Alexopoulos

May 2024

Πρόβλημα 1

Ερώτημα 1^o

Για να βρούμε αν υπάρχει δρομολόγιο μεταξύ των πόλεων s και t θα χρησιμοποιήσουμε έναν αλγόριθμο BFS (Breadth - First - Search) με 2 διαφορές. Η πρώτη θα είναι η προσθήκη ενός επιπλέον ελέγχου στους γειτονικούς κόμβους, για το αν η μεταξύ απόσταση των εκάστοτε 2 πόλεων είναι μικρότερη του L. Στη περίπτωση που αυτό ισχύει περνάμε από αυτή την ακμή/δρόμο και συνεχίζουμε την αναζήτησή μας από αυτόν τον κόμβο, αλλιώς τον αγνοούμε και απορρίπτουμε την εκάστοτε διαδρομή. Η δεύτερη διαφορά θα είναι ο έλεγχος κάθε φορά για το αν ο κόμβος που βγάζουμε από την ουρά είναι ο t. Όταν και άμα τον συναντήσουμε, ο αλγόριθμος θα επιστέφει true, ενώ άμα αδειάσει η ουρά χωρίς να τον έχουμε πετύχει θα επιστρέφει false. Η χρονικη πολυπλοκότητα αυτού του αλγορίθμου είναι $O(V+E)$. Στο παράρτημα παρατίθεται σχετικός ψευδοκώδικας.

Ερώτημα 2^o

Για την εύρεση των ελάχιστων καυσίμων που είναι απαραίτητα για να ταξιδέψουμε από την πόλη s στην πόλη t θα εξερευνήσουμε όλες τις διαδρομές μεταξύ αυτών των 2 κόμβων του γράφου. Πιο συγκεκριμένα θα βρίσκουμε για κάθε μία από αυτές τις διαδρομές τη μέγιστη απόσταση μεταξύ 2 πόλεων και από όλες αυτές τις αποστάσεις θα επιλέγουμε τη μικρότερη, η οποία θα αποτελεί και την ελάχιστη χωρητικότητα καυσίμων. Η εξερεύνηση του γράφου θα γίνεται με τον αλγόριθμο Dijkstra και η πολυπλοκότητα του αλγορίθμου θα είναι $O(|V|) \cdot T(Extract - Min) + O(|V| + |E|) \cdot T(Decrease - Value)$, θα εξαρτάται δηλαδή από τη δομή δεδομένων που θα χρησιμοποιήσουμε για την υλοποίηση του. Στην περίπτωση που επιλέξουμε για την υλοποίηση έναν δυαδικό σωρό η πολυπλοκότητα του αλγορίθμου μας θα είναι $O((|V| + |E|) \cdot log|V|)$.

Πρόβλημα 2

Ερώτημα 1^o

Από τη στιγμή που γνωρίζουμε εκ των προτέρων τον χρόνο εξυπηρέτησης του κάθε πολίτη, το πρώτο μας βήμα θα είναι η ταξονόμηση των n πολιτών κατά αύξουσα σειρά. Στην συνέχεια οι πολίτες θα εξυπηρετήθουν με τη σειρά αυτής της ταξινόμησης, δηλαδή πρώτα οι πολίτες με τον μικρότερο χρόνο αναμονής. Ο λόγος για τον οποίο επιλέγουμε αυτή τη σειρά είναι γιατί ο χρόνος αναμονής κάθε πολίτη προστίθεται σε αυτόν όλων των επομένων. Επομένως με αυτόν τον τρόπο ελαχιστοποιούμε τον συνολικό χρόνο αναμονής. Ο αλγόριθμος αυτός είναι αποδοτικός γιατί έχει χρονική πολυπλοκότητα $O(nloogn+n) = O(nlogn)$, επειδή η ταξινονόμηση (Merge Sort ή Quick Sort ή Heap Sort) έχει πολυπλοκότητα $O(nlogn)$ και η εξυπηρέτηση των πολιτών $O(n)$. Στο παράρτημα παρατίθεται σχετικός ψευδοκώδικας και ένα απλό παράδειγμα ωστε να αποδειχθεί η ορθότητα του αλγορίθμου μας.

Πρόβλημα 3

Ερώτημα 1^o

Αξιοποιώντας τις βασικές αρχές του δυναμικού προγραμμτισμού ορίζουμε ως το βασικό μας υποπρόβλημα την εύρεση του ελάχιστου υπολογιστικού κόστους για το χωρισμό ενός substring σε επιμερή τμήματα. Η εύρεση του κόστους σε ένα substring βρίσκεται από την εξής σχέση: $C[i,j] = min(C[i,j], C[k-1,j-1] + cost(k,i))$ για κάθε $i\epsilon[1,n], j\epsilon[1,m]$ και $k\epsilon[1,i]$, οπου: $cost(k,i) = i-k+1$, C[i,j] είναι το ελάχιστο κόστος για τον χωρισμό ενός substring μήκους i σε j+1 κομμάτια και η μεταβλητή k συμβολίζει για κάθε τιμή των i και j όλα τα δυνατά σημεία στα οποία μπορούμε να χωρίσουμε το substring. Φυσικά αυτή η σχέση προαπαιτεί την αρχικοποίηση όλων των ελαχίστων κερδών σε $+\infty$ εκτός από τις τιμές C[i,0] οι οποίες αρχικοποιούνται σε 0 καθώς για τον χωρισμό μιας συμβολοσειράς σε 0 επιμερή τμήματα δεν απαιτείται καθόλου κόστος. Έτσι, θα καταλήξουμε στο ζητούμενο ελάχιστο υπολογιστικό κόστος για τις τιμές i=n και j=m, δηλαδή θα βρούμε το C[n,m].Στο παράρτημα παρατίθεται σχετικός ψευδοκώδικας.

Ερώτημα 2^o

Η χρονική πολυπλοκότητα του αλγορίθμου αυτού είναι $O(n+n^2 \cdot m) = O(n^2 \cdot m)$ επειδή το πρώτο for loop εκτελείται n φορές και τα 3 υπόλοιπα εμφωλευμένα loop εκτελούνται στη χειρότερη περίπτωση m, n και n φορές αντίστοιχα με τη σειρά που γράφτηκαν στον ψευδοκώδικα.

Παράρτημα

Πρόβλημα 1

Ερώτημα 1^o

Ψευδοκώδικας

```
for each vertex u ∈ G - {s}
  u.π = NIL
  u.d = ∞
  u.color = WHITE
s.π = NIL
s.d = 0
s.color = GRAY
Q ≠ 0
ENQUEUE(Q, s)
while Q ≠ 0
  u = DEQUEUE(Q)
  for each vertex v ∈ G.Adj[u]
     if v.color == WHITE && l[u,v]<L //l[u,v]:distance between cities u & v
     v.color = GRAY
     v.π = u
     v.d = u.d + 1
     if v==t return true //if we find t
     ENQUEUE(Q, v)
u.color = BLACK
end

return false //if we dont find t
```

Πρόβλημα 2

Ψευδοκώδικας

```
Let serviceTimes[] be the array of each citizens service time
Let waitingTimes[] be the array of each citizens waiting time

MergeSort(serviceTimes[]);

totalTime = 0;
for i from 1 to n:
    if i=1: waitingTimes[i] = serviceTimes[i];
    else: waitingTimes[i] = serviceTimes[i] + WaitingTimes[i-1];
totalTime = totalTime + waitingTimes[i] ;
end

return totalTime;
```

Παράδειγμα

Έστω ότι έχουμε 3 πολίτες A,B,C με χρόνους αναμονής $t_A = 1$,$t_B = 2$ και $t_C = 3$ μονάδες χρόνου αντίστοιχα.Θα υπολογίσουμε για κάθε δυνατή σειρά εξυπηρέτησης τον συνολικό χρόνο αναμονής T των πολιτών.

i)Για σειρά A,B,C : T = 1 + (1 + 2) + (3 + 3) = 10
ii)Για σειρά A,C,B : T = 1 + (1 + 3) + (4 + 2) = 11
iii)Για σειρά B,A,C : T = 2 + (2 + 1) + (3 + 3) = 11
iv)Για σειρά B,C,A : T = 2 + (2 + 3) + (5 + 1) = 13
v)Για σειρά C,A,B : T = 3 + (3 + 1) + (4 + 2) = 13
vi)Για σειρά C,B,A : T = 3 + (3 + 2) + (5 + 1) = 14

Από τους παραπάνω υπολογισμούς παρατηρούμε ότι πράγματι με τον αλγόριθμό μας (Περίπτωση i) έχουμε τον ελάχιστο συνολικό χρόνο αναμονής.

Πρόβλημα 3

Ψευδοκώδικας

```
Let C[a,b] be the array of the minimum computational cost
of cutting the a character long string into b+1 parts

cost(k,i) = i-k+1

For each i from 1 to n:
    C[i, 0] = 0
For each j from 1 to m:
    For each i from j to n:
        C[i, j] = infinity
        For each k from 1 to i:
            C[i, j] = min(C[i, j], C[k-1, j-1] + cost(k, i))

return C[n,m]
```