Δευτερη Εργασία

Βασίλειος Σκαρλάτος | 4107

Ιούνιος 2023

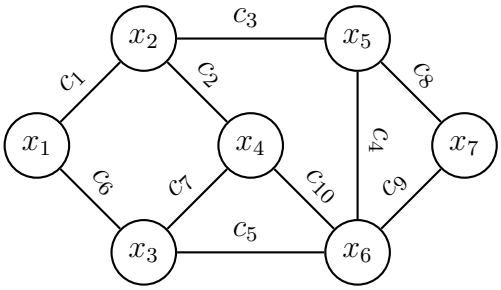
1 Πρώτη Άσκηση

Σε αυτήν την άσκηση μας ζητάται να αναπτύξουμε αλγόριθμο που να υπολογίζει ένα έγκυρο υποσύνολο E' με το ελάχιστο συνολικό βάρος όπου το E' προκύπτει ως εξής:

Έστω $G = (V, E)$ ένα **μη κατευθυνόμενο**, γράφημα με θετικό βάρος c_e σε κάθε ακμή $e \in E$. Ονομάζουμε ένα υποσύνολο ακμών $E' \subseteq E$ έγκυρο, αν όταν **αφαιρεθεί** το E' από το σύνολο ακμών E, το γράφημα **δεν περιέχει κύκλο**: δηλαδή το γράφημα $G = (V, E/E')$ δεν περιέχει κανένα κύκλο.

Λύση:

Μας δίνεται ένας γράφος $G = (V, E)$:



Ο συγκεκριμένος γράφος είναι προφανές ότι έχει πολλούς κύκλους όμως είναι πιθανόν ο γράφος που θα μας δωθεί να μήν έχει και **κανέναν**.

Αυτό που θέλουμε ουσιαστικά είναι να αφαιρέσουμε τους κύκλους από το γράφημα με τέτοιο τρόπο ώστε το συνολικό βάρος των ακμών $e \in E' \subseteq E$ που αφαιρούνται, να έχουν το ελάχιστο δυνατό βάρος.

Για να το καταφέρουμε αυτό, θα ¨χτίσουμε' τον γράφο μέχρι το σημείο που θα έχει όλα τα nodes του και κανέναν κύκλο, ξεκινώντας από τις ακμές με το μεγαλύτερο βάρος.

Βήματα αλγορίθμου:

1. Δημιουργούμε μία κενή λίστα E'.
2. Δημιουργούμε μία κενή λίστα για να αποθηκεύουμε nodes.
3. Κάνουμε φθίνουσα ταξινόμιση την λίστα των ακμών E.
4. Για κάθε ακμή της λίστας E:
   - Αν και τα 2 nodes της βρίσκονται στην λίστα του βήματος 2. , τότε αποθηκεύουμε την ακμή στην λίστα E'.
   - Αν το node 1 της ακμής είναι μέσα στην λίστα των nodes, τότε αποθηκεύουμε σε αυτή την λίστα το node 2 της ακμής.
   - (Αντίστοιχα) Αν το node 2 της ακμής είναι μέσα στην λίστα των nodes, τότε αποθηκεύουμε σε αυτή την λίστα το node 1 της ακμής.
   - Αλλιώς αποθηκεύουμε και τα 2 nodes της ακμής στον πίνακα των nodes.

Ψευδοκώδικας:

Θεωρούμε δεδομένο ότι οι ακμές $e \in E$ είναι **αντικείμενα** με attributes, τα nodes 1 και 2, αλλά και το βάρος c_e.
Επίσης, δεδομένη θεωρείται συνάρτηση **QS** (από QuickSort) Η οποία παίρνει σαν είσοδο μία λίστα ακμών, την ταξινομεί με βάση τα βάρη c σε **φθίνουσα** σειρά και επιστρέφει την ταξινομιμένη λίστα.
Τελος, θεωρούμε E, την λίστα των ακμών και e.n1, το node 1 της ακμής e και e.n2, το node 2 της ακμής e.

E′=[ ]
nodes=[ ]
E2=QS(E)

```
FOR EACH e IN E2:
   IF ((e.n1 IN nodes) AND (e.n2 IN nodes)):
      E'.append(e)
   IF (e.n1 IN nodes):
      nodes.append(e.n2)
   IF (e.n2 IN nodes):
      nodes.append(e.n1)
   ELSE:
      nodes.append(e.n1)
      nodes.append(e.n2)
return E'
```

Πολυπλοκότητα:

Ας αναλύσουμε την πολυπλοκότητα του αλγορίθμου μας με βάση τον παραπάνω ψευδοκώδικα:

- Οι πρώτες 2 γραμμές είναι αρχικοποιήσεις των λιστών, επομένως έχουν πολυπλοκότητα $O(1)$
- Η τρίτη γραμμή κάνει χρήση της **QuickSort**, που είναι γνωστό ότι έχει πολυπλοκότητα $O(\log E)$, Όπου E, ο αριθμός των ακμών του δεδομένου γράφου.
- Η τέταρτη γραμμή κάνει χρήση της λούπας **FOR**, επομένως πρέπει να πολλαπλασιάσουμε την πολυπλοκότητα του βρόχου με $O(E)$
- Μέσα στις **IF**, δηλαδή στις γραμμές 5,7,9, υπάρχουν οι εντολές αναζήτησης **IN** που έχουν πολυπλοκότητα $O(V)$. Όπου V, ο αριθμός των κόμβων του δεδομένου γράφου.
- Οι υπόλοιπες γραμμές είναι εντολές ανάθεσης, άρα έχουν πολυπλοκότητα $O(1)$

Άρα συνολικά έχουμε

$$O(1) + O(\log E) + O(E * V) \;=\; O(E * V)$$

$$\forall\, G \;=\; (E, V)$$

Ορθότητα:

Ο Αλγόριθμος είναι ορθός διότι ¨χτίζει' ένα δένδρο το οποίο προκύπτει από τον δεδομένο γράφο, το οποίο έχει όλους του κόμβους του γράφου αλλά μόνο τις ακμές με τα μεγαλύτερα βάρη. Αυτό σημαίνει ότι οι ακμές που δεν κατέληξαν στο δένδρο -και άρα κατέληξαν στο Ε΄-, έχουν το ελάχιστο συνολικό βάρος.

2 Δεύτερη Άσκηση

Σε αυτήν την άσκηση μας ζητάται να αναπτύξουμε αλγόριθμο που να επιλέγει τον μέγιστο αριθμό καλεσμένων σε μία δεξίωση όπου πρέπει όλοι να έχουν τουλάχιστον 4 γνωστούς και τουλάχιστον 4 άγνωστους:

Λύση:

Εφόσον μας δίνονται τα ζευγάρια γώστών, είναι εύκολο να κατασκευάσουμε έναν πίνακα $n \times n$ για τους n υποψληφιους καλεσμένους, βάζοντας 1 στα ζευγάρια που γνωρίζονται και 0 στα ζευγάρια που δεν γνωρίζονται (Πίνακα γειντίασης δηλαδή):

Candidates	a_1	a_2	...	a_n
a_1	-	1	...	0
a_2	1	-	...	1
$\vdots$	$\vdots$	$\vdots$	-	$\vdots$
a_n	0	1	...	-

Βήματα αλγορίθμου:

1. Δημιουργούμε τον παραπάνω πίνακα.
2. Διατρέχουμε τον πίνακα ανά σειρά και ελέγχουμε σειρά-σειρά αν ο υποψήφιος πληρεί τα κριτήρια (Δηλαδή αν έχει τουλάχιστον 4 άσους ΚΑΙ 4 μηδενικά).
3. Αν τα πληρεί τον τοποθετούμε σε έναν νέο πίνακα, ο οποίος δουλεύει με τον ίδιο τρόπο με τον πίνακα που διατρέχουμε, αλλιώς, τον προσπερνάμε. Κρατάμε και τους 2 πίνακες.
4. Επαναλαμβάνουμε τα βήματα 2 και 3 μέχρι ο νέος πίνακας να έχει ίδια διάσταση με τον παλιό, καθώς αυτό σημαίνει ότι όλοι οι καλεσμένοι πληρούν τα κριτήρια της δεξίωσης.

Πολυπλοκότητα:

Ας αναλύσουμε την πολυπλοκότητα του αλγορίθμου μας με βάση τα παραπάνω βήματα:

- Εφόσον μας δίνονται ζεύγη γνωστών, η κατασκευή του πίνακα απαιτεί να αναζητήσουμε για κάθε άτομο όλα τα ζεύγη του και να τοποθετήσουμε 1 στις αντίστοιχες θέσεις του πίνακα και 0 σε όλες τις άλλες, δηλαδή έχουμε $O(n^2)$.
- Ο έλεγχος του πίνακα έχει πολυπλοκότητα $O(n^2)$.
- Η δημιουργία του καινούριου πίνακα, γίνεται παράλληλα με το προηγούμενο βήμα, οπότε συμπεριλαμβάνεται στο $O(n^2)$.
- Τα βήματα 2 και 3, στην χειρότερη περίοτωση (αυτή δηλαδή στην οποία σε κάθε επανάληψη ¨διώχνουμε' έναν υποψήφιο καλεσμένο μέχρι να μην μείνει κανένας), γίνονται n φορές.

Επομένως η συνολική πολυπλοκότητα προκύπτει ως εξής:

$$O(n^2) + O(n * n^2) = O(n^3)$$

για n υποψήφιους καλεσμένους.

Ορθότητα: [Εις άτοπον απαγωγή]

Έστω οτι η τελική λίστα δεν είναι σωστή, αυτό σημαίνει ότι έχουμε ή κάποια άτομα παραπάνω ή κάποια λιγότερα:

- Αν έχουμε λιγότερα άτομα:
  - Αυτό σημαίνει ότι υπάρχουν άτομα τα οποία πληρούν τα κρητήρια πρόσκλησης στην δεξίωση αλλά ο αλγόριθμος τους ¨έδιωξε¨.
  - Όμως ο αλγόριθμος αφαιρεί μόνο άτομα τα οποία δεν πληρούν τα κρητήρια.
  - Άρα καταλήγουμε σε **άτοπο**.
- Αν έχουμε περισσότερα άτομα:
  - Αυτό σημαίνει ότι υπάρχουν άτομα τα οποία δεν πληρούν τα κρητήρια πρόσκλησης τα οποία όμως κατέληξαν προσκεκλημένα.

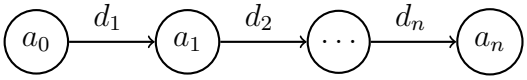
– Ο αλγόριθμος τερματίζει και μας δίνει την τελική λίστα όταν έχει κάνει έναν γύρο ελέγχου και δεν βρει κανέναν που δεν πληρεί τα κρητήρια πρόσκλησης.
– Άρα όλοι οι προσκεκλημένοι πληρούν τα κρητήρια.
– Άρα καταλήγουμε σε **άτοπο**.

3 Τρίτη Άσκηση

Σε αυτήν την άσκηση μας ζητάται να αναπτύξουμε αλγόριθμο που να επιλέγει το ταξίδι με την ελάχιστη δυνατή ποινή μεταξύ του σημείου εκκίνησης και του τερματισμού:

Λύση:

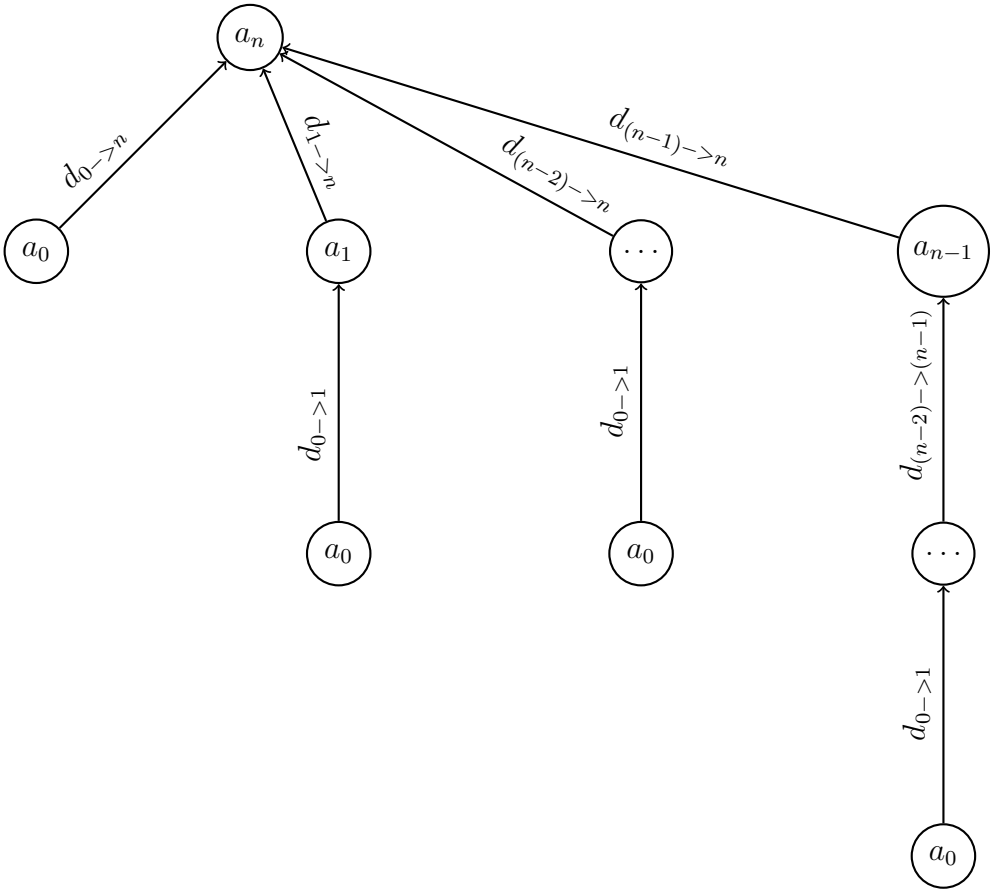
Αυτό που μας δίνεται είναι τα ξενοδοχεία και οι θέσεις τους σε σχέση με το σημείο μηδέν. Όμως μπορούμε εύκολα να υπολογίσουμε τις αποστάσεις μεταξύ των ξενοδοχείων και να καταλήξουμε με το εξής γράφημα.



Όπου d, οι αποστάσεις.

Το ερώτημα που καλούμαστε αρχικά να απαντήσουμε, είναι:
Με ποιούς τρόπους μπορώ να φτάσω στο a_n;

Απαντάμε αυτό το ερώτημα σχεδιάζοντας το δένδρο με όλες τις πιθανές διαδρομές:



Στο παραπάνω δένδρο, υπάρχουν όλες οι πιθανές διαδρομές από το σημείο μηδέν μέχρι τον τελικό προορισμό μαζί με τις αποστάσεις d για κάθε υπο-διαδρομή. Άρα μπορούμε να υπολογίσουμε την ποινή της κάθε διαδρομής:

$$penalty = \sum_{i=1}^{N} (200 - d_i)^2$$

Όπου N, ο αριθμός τον υπο-διαδρομών και d, οι αποστάσεις μεταξύ των ξεκοδοχείων.

Θα μπορούσαμε να πολύ απλά να ελέγξουμε όλες τις πιθανές διαδρομές και να επιλέξουμε αυτήν με το ελάχιστο κόστος, αλλά ένας τέτοιος αλγόριθμος δεν θα ήταν αποδοτικός. Οπότε θα προσεγγίσουμε το πρόβλημα αλλιώς. Συγκεκριμένα, με **Δυναμικό προγραμματισμό**.

Αρχικά, θα αποθηκεύσουμε σε έναν πίνακα την πιο αποδοτική διαδρομή προσέλευσης σε κάθε ξενοδοχείο (όσον αφορά το κόστος), δηλαδή το **opt[j]** ξεκινώντας από το σημείο μηδέν, έτσι για κάθε επόμενο ξενοδοχείο, θα έχουμε ήδη αποθηκευμένα τα κόστη προσέλευσης των προηγουμένων, μειώνοντας τους ελέγχους που πρέπει να κάνουμε και κατά συνέπεια την πολυπλοκότητα του αλγορίθμου μας.

Δηλαδή, το υποπρόβλημα την περίπτωσή μας είναι το κόστος της διαδρομής προς το ξενοδοχείο.

Βήματα αλγορίθμου:

1. Δημιουργόμε έναν πίνακα και αποθηκεύουμε διαδρομή κόστους 0 στην πρώτη θέση. (καθώς το κόστος προσέλευσης στο σημείο 0, είναι 0)
2. Για κάθε ξενοδοχείο (ξεκινόντας από το πρώτο):
   - Υπολογίζουμε τα κόστη προέλευσης σε αυτό το ξενοδοχείο από κάθε προηγόυμενο ξενοδοχείο, δηλαδή από όλες τις προηγούμενες θέσεις του πίνακα.
   - Προσθέτουμε σε αυτά τα κόστη, τα κόστη προσέλευσης στα αντίστοιχα ξενοδοχεία και από τα σύνολα, αποθηκεύμουμε στον πίνακα το ελάχιστο μαζί με την αντίστοιχη διαδρομή.

Ψευδοκώδικας:

Θεωρούμε δεδομένες συναρτήσεις: findCost(a,b) η οποία υπολογίζει το κόστος από το ξενοδοχείο a στο b, findMin(x,y,arr[ ]) η οποία βρίσκει το ελάχιστο κόστος του πίνακα από την θέση x μέχρι y, findMinPos(x,y,arr[ ]), το ίδιο αλλά επιστρέφει την θέση που το βρήκε, δηλαδή την διαδρομή. Επίσης, θεωρούμε γνωστή την ύπαρξη αντικειμένων διαδρομών route τα οποία έχουν τις εξής μεθόδους:

- copy: αντιγράφει ένα άλλο αντικείμενο route.
- add: προσθέτει ξενοδοχεία στην διαδρομή.
- getCost: επιστρέφει το κόστος..
- setCost: βάζει κόστος.

Τέλος, θεωρούμε αντικείμενα Hotel για τα ξενοδοχεία και πίνακα Hotels δεδομένο.

```
routes=[ ]
route0= new route()
route0.setCost(0)
routes.append(route0)
FOR i IN RANGE (1,n):
   h=Hotels[i]
   costs=[ ]
   FOR j IN RANGE (0,i-1):
      costs.append(routes[j].getCost()+findCost(routes[j],h))
   Pos=findMinPos(0,i,costs[ ])
   R=new route()
   R.copy(routes[pos])
   R.add(h)
   R.setCost(findMin(0,i,costs[ ]))
   routes.append(R)
return routes[n]
```

Πολυπλοκότητα:

- Η πολυπλοκότητα δημιουργίας του πρώτου αντικειμένου είναι O(1).
- Η Εξωτερική λούπα θα γίνει n φορές.
- Η εσωτερική λούπα θα γίνει το πολύ n φορές (ανά εξωτερική λούπα).

Επομένως η συνολική πολυπλοκότητα προκύπτει ως εξής:

$$O(1) + O(n * n) = O(n^2)$$

Ορθότητα:

Όπως αναφέραμε, ο συγκεκριμένος αλγόριθμος υλοποιεί δυναμικό προγραμματισμό, δηλαδή το opt[j] είναι ορισμένο με βάση τα προηγούμενα opt[ ]. Το opt[0] είναι καλά ορισμένο καθώς opt[0]=0, άρα το opt[1] είναι επίσης καλά ορπισμένο, με βάση αυτό, το opt[2] υποθέτουμε οτι είναι επίσης καλά ορισμένο κ.ο.κ. άρα το opt[n-1] υποθέτουμε οτι είναι καλά ορισμένο άρα το opt[n] είναι καλά ορισμένο άρα και ο αλγόριθμος είναι ορθός.