Ανάλυση και Σχεδιασμός Αλγορίθμων

Ιωάννης Χατζήνας Ιωάννης Λώλος

Απρίλιος-Μάιος 2024

Πρόβλημα 1

Ύπαρξη Εφικτού Δρομολογίου $s \to t$

Το πρόβλημα της διαπίστωσης ύπαρξης κάποιας εφικτής διαδρομής, ανάμεσα σε δύο κόμβους $s - t$ ενός ακατεύθυντου γράφου G (χωρίς να ληφθεί κάποιος περιορισμός για τα βάρη), είναι απλό και μπορεί να λυθεί με τη χρήση είτε Διερεύνυσης κατα Βάθος, είτε Διερεύνυσης κατα Πλάτος. Θα επιλέξουμε αυθαίρετα τον αλγόριθμο DFS[1] και θα τον τροποποιήσουμε αναλόγως, ώστε να λαμβάνει υπ΄όψιν την περιορισμένη χιλιομετρική απόσταση L που μπορεί να καλυφθεί με γεμάτο ρεζερβουάρ και τα μήκη l_e των οδικών αρτηριών και τελικά να επιστρέφει ΑΛΗΘΕΣ αν ή διαδρομή είναι εφικτή, αλλιώς ΨΕΥΔΕΣ. Οι αλλαγές που εφαρμόσαμε στον αλγόριθμο είναι οι εξής:

- Παραλήψαμε οτιδήποτε σχετικό με τους προκάτοχους κάθε κόμβου και τον χρόνο επίσκεψης τους, καθώς δεν αποτελούν πληροφορίες χρήσιμες στον αλγόριθμο μας.
- Καλέσαμε την υπορρουτίνα ΕΠΙΣΚΕΨΗ με εκκίνηση μόνο τον κόμβο s, αφού δεν θέλουμε να επισκεφθούμε ολόκληρο το γράφημα, αλλά μόνο όσους κόμβους μπορούμε να φτάσουμε ξεκινώντας από τον s
- Προσθέσαμε στις γραμμές 5-8 την συνθήκη τερματισμού του αλγορίθμου
- Η υπορρουτίνα ΕΠΙΣΚΕΨΗ δέχεται πλέον το όρισμα L
- Στη γραμμή 11 περιορίσαμε τους γειτονικούς κόμβους που θα επισκευθούμε μόνο σε αυτούς για τους οποίους το βάρος της ακμής (u, v) είναι μικρότερο από L

Συνοπτικά ο αλγόριθμος ξεκινάει από τον κόμβο s και επισκέπτεται όλους τους κόμβους του γράφου, τους οποίους μπορεί να φτάσει μέσω ακμών που έχουν βάρος μικρότερο του L. Όσοι κόμβοι επισκέπτονται απο τον αλγόριθμο βάφονται μελανοί. Στο τέλος, αν ο κόμβος t είναι μελανός τότε επιστρέφεται ΑΛΗΘΕΣ, αλλιώς ΨΕΥΔΕΣ.

Όσον αφορά τον χρόνο εκτέλεσης, στην χειρότερη περίπτωση (όταν δηλαδή πρέπει να επισκευθούμε όλους τους κόμβους του γράφου) ισχύει:

- Ο βρόγχος των γραμμών 2-3 απαιτεί χρόνο $O(V)$
- Η συνθήκη των γραμμών 5-8 απαιτεί χρόνο $O(1)$
- Η κλήση της υπορρουτίνας ΕΠΙΣΚΕΨΗ, μαζι με όλες τις αναδρομικές κλήσεις της θα εξερευνήσει όλες τις ακμές και όλους τους κόμβους του γράφου, επομένως απαιτεί χρόνο $O(|E| + |V|)$

Συνολικά λοιπόν ο αλγόριθμος έχει χρονική πολυπλοκότητα $O(|V| + |E|)$

Algorithm 1 Ύπαρξη Εφικτού Δρομολογίου $s \to t$

```
1:  ΥΠΑΡΞΗ ΔΙΑΔΡΟΜΗΣ(G,L,s,t)
2:      Για κάθε u ∈ G.V
3:          u.χρώμα = ΛΕΥΚΟ
4:      ΕΠΙΣΚΕΨΗ(G,s,L)
5:      Αν t.χρώμα == ΜΕΛΑΝΟ τότε
6:          Επίστρεψε ΑΛΗΘΕΣ
7:      Αλλιώς
8:          Επίστρεψε ΨΕΥΔΕΣ

9:  ΕΠΙΣΚΕΨΗ(G,u,L)
10:     u.χρώμα == ΓΚΡΙΖΟ
11:     Για κάθε v ∈ G.adj[u] με w(v,u) ≤ L
12:         Αν v.χρώμα == ΛΕΥΚΟ τότε
13:             ΕΠΙΣΚΕΨΗ(G,v,L)
14:     u.χρώμα = ΜΕΛΑΝΟ
```

Εύρεση Ελάχιστης Ποσότητας Καυσίμων

Για την εύρεση της ελάχιστης ποσότητας καυσίμων που απαιτείται για την ολοκλήρωση της διαδρομής θα προτείνουμε μια brute force προσέγγιση, δεδομένου ότι έχουμε στην διάθεση μας έναν γρήγορο αλγόριθμο πιστοποίησης της ύπαρξης δυνατού δρομολογίου $s \to t$ για δοσμένη ποσότητα καυσίμων L. Θα δοκιμάσουμε τον αλγόριθμο 1 για όλα τα δυνατά μήκη l_e και θα κρατήσουμε το μικρότερο μήκος για το οποίο ο αλγόριθμος τερματίζει επιτυχώς. Με αυτόν τον τρόπο μπορούμε να πετύχουμε χρονική πολυπλοκότητα $O((|V| + |E|)|E|)$, καθώς πρέπει να καλέσουμε $|E|$ φορές τον αλγόριθμο 1, ο οποίος απαιτεί χρόνο $O(|E| + |V|)$. Αυτό δεν είναι το βέλτιστο που μπορεί να επιτευχθεί. Μπορούμε να πετύχουμε πολυπλοκότητα $O((|V| + |E|) \log |E|)$, αν αντί να κάνουμε γραμμική αναζήτηση στο σύνολο των l_e, τα ταξινομήσουμε και ψάξουμε τη λύση με δυαδική αναζήτηση. Η ταξινόμηση μπορεί να επιτευχθεί με ταχύτητα έως και $O(E \log E)$, αν χρησιμοποιήσουμε πχ τον αλγόριθμο mergesort, ενώ πλέον η υπόλοιπη διαδικασία χρειάζεται μόνο $O((|V| + |E|) \log E)$ χρόνο. Η συνολική χρονική πολυπλοκότητα είναι: $O((|V| + |E|) \log E)$. Συμπληρωματικά, αν υποθέσουμε ότι κάθε ζεύγος κόμβων συνδέεται από το πολύ μία ακμή, τότε ο γράφος έχει το πολύ $\frac{u(u-1)}{2}$ ακμές, όπου

u το πλήθος κόμβων. Επομένως $O((|V| + |E|) \log E) = O((|V| + |E|) \log V)$ (απόδειξη σε παράρτημα).

Algorithm 2 Εύρεση Ελάχιστης Ποσότητας Καυσίμων

```
1:  ΕΥΡΕΣΗ ΕΛΑΧΙΣΤΗΣ ΠΟΣΟΤΗΤΑΣ ΚΑΥΣΙΜΩΝ(G, s, t)
2:      ΤΑΞΙΝΟΜΗΣΗ(G.l)                          ▷ με χρήση mergesort
3:      L = 0, R = G.E.μέγεθος
4:      Όσο L ≠ R
5:          M = ⌊(L + R)/2⌋
6:          Αν ΥΠΑΡΞΗ ΔΙΑΔΡΟΜΗΣ(G,G.l_M,s,t) τότε
7:              ελάχιστο= G.l_M
8:              R = M
9:          Αλλιώς
10:             L = M
11:     Επίστρεψε ελάχιστο
```

Πρόβλημα 2

Έστω t_i ο χρόνος εξυπηρέτησης του i-οστού πολίτη στην ουρά και w_i ο χρόνος αναμονής του. Ο χρόνος w_i δίνεται από το άθροισμα των χρόνων εξυπηρέτησης των μπροστινών πολιτών:

$$w_i = \sum_{j=1}^{i-1} t_j \tag{1}$$

Ο συνολικός χρόνος αναμονής W για n πολίτες δίνεται ως εξής:

$$W = \sum_{i=1}^{n} w_i \tag{2}$$

Επομένως από τις εξισώσεις (1),(2) προκύπτει:

$$W = \sum_{i=1}^{n} \sum_{j=1}^{i-1} t_j = \sum_{i=1}^{n} (n - i) t_i$$

Παρατηρούμε ότι ο χρόνος εξυπηρέτησης των πρώτων πολιτών στην ουρά $(t_1, t_2, ...)$ πολλαπλασιάζεται με έναν παράγοντα $n - i$, ο οποίος μειώνεται όσο αυξάνεται το i. Επομένως για να ελαχιστοποιήσουμε το άθροισμα, και επομένως τον συνολικό χρόνο αναμονής W αρκεί να ταξινομήσουμε τους πολίτες στην ουρά, ώστε οι πρώτοι να έχουν τον μικρότερο χρόνο εξυπηρέτησης. Θα το κάνουμε χρησιμοποιώντας τον αλγόριθμο ταξινόμησης Mergesort[2], ο οποίος είναι της τάξης $O(n \log n)$

Algorithm 3 Mergesort

```
1:  ΣΥΓΧΩΝΕΥΤΙΚΗ ΤΑΞΙΝΟΜΗΣΗ(A,p,q)
2:      Αν p < r τότε
3:          q = ⌊(p + r)/2⌋
4:          ΣΥΓΧΩΝΕΥΤΙΚΗ ΤΑΞΙΝΟΜΗΣΗ(A,p,q)
5:          ΣΥΓΧΩΝΕΥΤΙΚΗ ΤΑΞΙΝΟΜΗΣΗ(A,q+1,r)
6:          ΣΥΓΧΩΝΕΥΣΗ(A,p,q,r)

7:  ΣΥΓΧΩΝΕΥΣΗ(A,p,q,r)
8:      n_1 = q − p + 1
9:      n_2 = r − q
10:     Έστω L[1..n_1 + 1] και R[1..n_2 + 1] δύο νέες συστοιχίες
11:     Για i = 1 έως n_1
12:         L[i] = A[p + i − 1]
13:     Για j = 1 έως n_2
14:         R[j] = A[q + j]
15:     L[n_1 + 1] = ∞
16:     R[n_2 + 1] = ∞
17:     i = 1
18:     j = 1
19:     Για k = p έως r
20:         Αν L[i] ≤ R[j] τότε
21:             A[k] = L[i]
22:             i = i + 1
23:         Αλλιώς
24:             A[k] = R[j]
25:             j = j + 1
```

Πρόβλημα 3

Έστω S η συμβολοσειρά μας και $c_1, c_2, ..., c_m$ οι m τομές, προφανώς με $1 \leq c_i \leq n$ για κάθε $i \in [1, m]$. Το ελάχιστο κόστος του να γίνουν όλες οι τομές που βρίσκονται ανάμεσα στην i-οστή και την j-οστή τομή της συμβολοσειράς μπορεί να προσδιοριστεί από την εξής αναδρομική συνάρτηση:

Algorithm 4 Αναδρομικό Ελάχιστο Κόστος

```
1: ΑΝΑΔΡΟΜΙΚΟ ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ(S,τομές,i,j)
2:     Αν j − i ≤ 1 τότε
3:         Επίστρεψε 0
4:     Αλλιώς
5:         ελάχιστοΚόστος = ∞
6:         Για k από i μέχρι j
7:             κόστος = c_j − c_i+ ΑΝΑΔΡΟΜΙΚΟ ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ(S,τομές, i, k) + ΑΝΑΔΡΟΜΙΚΟ ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ(S,τομές, k, j)
8:             ελάχιστοΚόστος = min (ελάχιστοΚόστος, κόστος)
9:         Επίστρεψε ελάχιστοΚόστος
```

Το έλαχιστο συνολικό κόστος όλων των τομών μπορεί να προσδιοριστεί από την ΑΝΑΔΡΟΜΙΚΟ ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ(S,0,$m+1$) (Καταχρηστικά θα χρησιμοποιήσουμε το c_0 για να δηλώσουμε την αρχή της συμβολοσειράς και το c_{m+1} για το τέλος της.), ωστόσο ο παραπάνω αλγόριθμος δεν είναι αποδοτικός και μπορεί να βελτιωθεί με τη χρήση δυναμικού προγραμματισμού. Αυτό θα το επιτύχουμε χρησιμοποιώντας έναν $(m+2) \times (m+2)$ πίνακα κόστους, στην κάθε θέση ΚΟΣΤΟΣ$[i][j]$ του οποίου θα αποθηκεύσουμε το κόστος διαχωρισμού της συμβολοσειράς από την τομή c_i μέχρι την τομή c_j.

Algorithm 5 Δυναμικό Ελάχιστο Κόστος

```
1: ΔΥΝΑΜΙΚΟ ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ(S,τομές)
2:     Έστω ΚΟΣΤΟΣ[0..m + 1][0..m + 1] νέα συστοιχία
3:     ΚΟΣΤΟΣ[i][j] = 0 για κάθε i, j
4:     Για απόσταση από 2 έως m + 1
5:         Για i από 0 έως m + 1−απόσταση
6:             j = i+απόσταση
7:             ΚΟΣΤΟΣ[i][j] = ∞
8:             Για k από i + 1 μέχρι j − 1
9:                 κόστος = ΚΟΣΤΟΣ[i][k] + ΚΟΣΤΟΣ[k][j] + c_j − c_i
10:                ΚΟΣΤΟΣ[i][j] = min(ΚΟΣΤΟΣ[i][j],κόστος)
11:    Επίστρεψε ΚΟΣΤΟΣ[0][m + 1]
```

Ο νέος αλγόριθμος χρειάζεται $O(m^2)$ για την αρχικοποίηση του πίνακα ΚΟΣΤΟΣ, ενώ έχει τρεις εμφωλευμένους βρόγχους, ο μετρητής του καθενός εκ των

οποίων λαμβάνει το πολύ $m + 2$ τιμές. Οι υπόλοιπες εντολές μπορούν να εκτελεστούν σε σταθερό χρόνο. Συμπεραίνουμε πως ο δυναμικός αλγόριθμος έχει $O(m^3)$ χρόνο εκτέλεσης, το οποίο είναι μια βελτίωση σε σχέση με τον αναδρομικό αλγόριθμο, ο οποίος έχει εκθετικό χρόνο εκτέλεσης.

Παράρτημα Α

Απόδειξη $O((|V| + |E|) \log E) = O((|V| + |E|) \log V)$

Σύμφωνα με την υπόθεση μας, στην χειρότερη περίπτωση $E = \frac{V(V-1)}{2}$, όπου E, V τα πλήθη των ακμών και κόμβων αντίστοιχα. Επομένως:

$$
\begin{aligned}
E &= \frac{V(V-1)}{2} \Rightarrow \\
\log E &= \log \frac{V(V-1)}{2} \Rightarrow \\
\log E &= \log V + \log (V-1) - \log 2 \Rightarrow \\
(|V| + |E|) \log E &= (|V| + |E|)(\log V + \log (V-1) - \log 2) \Rightarrow \\
O((|V| + |E|) \log E) &= O((|V| + |E|)(\log V + \log (V-1) - \log 2)) \Rightarrow \\
O((|V| + |E|) \log E) &= O((|V| + |E|) \log V)
\end{aligned}
$$

Αναφορές

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest Clifford Stein. *Εισαγωγή στους Αλγορίθμους*. Πανεπιστημιακές Εκδόσεις Κρήτης, 2021, σσ. 598–602.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest Clifford Stein. *Εισαγωγή στους Αλγορίθμους*. Πανεπιστημιακές Εκδόσεις Κρήτης, 2021, σσ. 33–37.