[TH2802] Δεύτερη Εργασία – Δυναμικός Προγραμματισμός

Ομάδα 31

Καπετάνιος Αντώνιος [AEM 10417] (kapetaat@ece.auth.gr)
Κολιαμήτρα Ελευθερία [AEM 10422] (eleftheak@ece.auth.gr)
Μπαξεβάνης Γεώργιος [AEM 10301] (geompakar@ece.auth.gr)

Πρόβλημα 1

Πρόβλημα 1 *Ένα δίκτυο υπολογιστών αναπαρίσταται από έναν γράφο $G = (V, E)$ όπου οι κόμβοι αναπαριστούν συσκευές και οι ακμές αναπαριστούν τις συνδέσεις μεταξύ των συσκευών. Κάθε ακμή (u, v) έχει ένα βάρος το οποίο εκφράζει την πιθανότητα p_{uv} ότι ένα πακέτο το οποίο στέλνεται από τη συσκευή u θα φτάσει στην συσκευή v χωρίς να χαθεί. Οι πιθανότητες είναι ανεξάρτητες. Ζητείται το μονοπάτι από την συσκευή s προς τη συσκευή t με τη μέγιστη πιθανότητα επιτυχούς αποστολής.*

Πρόταση 1 *Given a longest-paths problem, create a copy of the given edge-weighted DAG that is identical to the original, except that all edge weights are negated. Then the shortest path in this copy is the longest path in the original.*[3]

Λαμβάνοντας υπ' όψιν τη φύση του Προβλήματος 1 και την Πρόταση 1 προκύπτουν τα ακόλουθα συμπεράσματα. Αρχικά, εφ' όσον ο γράφος G αναπαριστά ένα πραγματικό δίκτυο συσκευών, δεν δύναται να υπάρχει ακμή $(u, v) \in E$, μεταξύ δύο διαφορετικών κορυφών $u, v \in V$, η οποία να έχει βάρος $w((u, v)) = p_{uv} = 1$, διότι αυτό θα σήμαινε πως υπάρχει μηδενική πιθανότητα αποτυχίας της αποστολής σε ένα πραγματικό σύστημα. Επιπλέον, η άρνηση της συνάρτησης βάρους των ακμών του G υπολογίζεται ως εξής

$$\forall e \in E : \ \neg w(e) = 1.0 - w(e) > 0. \qquad (1)$$

καθότι η πιθανότητα είναι φραγμένη ποσότητα στο διάστημα $[0, 1.0]$ και έχουμε θεωρήσει πως το σύστημα δεν είναι ιδανικό ($\nexists e \in E : w(e) = 1$).

Έστω, ο γράφος $\neg G = (V', E')$ με βάρη, τέτοιος ώστε $V' = V$, $E' = E$ και τα βάρη των ακμών του να δίνονται από την $w' : E' \to [0, 1]$, όπου $w' = \neg w$. Η σχέση (1) εξασφαλίζει πως στον $\neg G$ δεν υπάρχουν ακμές με βάρος 0. Αυτό επιτρέπει τη χρήση της λογικής του αλγορίθμου του Dijkstra[1] για την εύρεση του μονοπατιού ελάχιστου μήκους ή στην προκειμένη ελάχιστης πιθανότητας αποτυχίας στον $\neg G$ και κατά συνέπεια, βάσει της Πρότασης 1, της διαδρομής μέγιστης πιθανότητας επιτυχούς αποστολής στον G.

Προτείνουμε τον Αλγόριθμο 1. Δέχεται ως είσοδο έναν γράφο $G = (V, E)$ ο οποίος αναπαρίσταται με adjacency lists τα οποία επιτρέπουν την πρόσβαση στη γειτονιά μιας κορυφής σε γραμμικό χρόνο ως προς το μέγεθος της γειτονιάς [2], μία κορυφή s η οποία είναι το αρχικό άκρο του ζητούμενου μονοπατιού, μία κορυφή t η οποία είναι το τερματικό άκρο του μονοπατιού και την συνάρτηση βαρών των ακμών $w : E \to [0, 1]$.

Στις γραμμές $2-3$ υπολογίζεται το negated βάρος όλων των ακμών. Το σώμα του βρόχου εκτελείται $|E|$ φορές και κάθε εκτέλεσή του χρειάζεται σταθερό χρόνο $\mathcal{O}(1)$. Επομένως, ο βρόχος επανάληψης των γραμμών $2-3$ χρειάζεται $\mathcal{O}(|E|)$ χρόνο.

Έπειτα, στη γραμμή 4 δημιουργείται ένα κενό Fibonacci heap $\mathcal{F}$ σε χρόνο $\Theta(1)$ [4]. Στην γραμμή 5 δημιουργείται ένας κενός πίνακας $\mathcal{P}$, στον οποίο θα αποθηκευτεί το ζητούμενο μονοπάτι, σε σταθερό χρόνο $\mathcal{O}(1)$. Στις γραμμές 6 και 7 δημιουργούνται δύο κενοί πίνακες, π και $\mathcal{L}$ αντιστοίχως, μεγέθους $|V|$. Άρα, ο χρόνος εκτέλεσης των γραμμών $5-7$ είναι $3\mathcal{O}(1) = \mathcal{O}(1)$.

Στις γραμμές 8 και 9 πραγματοποιούνται δύο καταχωρήσεις, μία στον πίνακα π και μία στον πίνακα $\mathcal{L}$, οι οποίες απαιτούν σταθερό χρόνο. Στις γραμμές $10-12$ αρχικοποιούμε τις υπόλοιπες θέσεις των προαναφερθέντων πινάκων. Το σώμα του for εκτελείται $|V| - 1$ φορές και κάθε φορά απαιτεί χρόνο $2\mathcal{O}(1) = \mathcal{O}(1)$. Άρα, συνολικά οι γραμμές 8 έως και 12 απαιτούν $\mathcal{O}(|V|)$ χρόνο.

Στις γραμμές $13-14$ εισάγουμε στο Fibonacci heap όλες τις κορυφές του γράφου G χρησιμοποιώντας ως κλειδί την τιμή που τους αντιστοιχεί στον πίνακα $\mathcal{L}$. Η μέθοδος insert(□,□) για ένα Fibonacci heap απαιτεί $\Theta(1)$ [4]. Επομένως, οι γραμμές $13-14$ χρειάζονται $\Theta(|V|)$ χρόνο.

Στην γραμμή 16 η μέθοδος extract_min() που αφορά

[1] Αν υπήρχαν ακμές με βάρος 0, δεδομένου πως το μήκος του μονοπατιού θα υπολογιστεί χρήσει του γινομένου $p_{e_i} \cdot p_{e_j}$, θα υπήρχε ακμή μέσω της οποίας διαρκώς θα μπορούμε να παίρνουμε μονοπάτι με μήκος 0 με συνέπεια η διαδικασία να είναι ατέρμων. Επιπλέον, η σχέση (1) επιτρέπει την επέκταση της Πρότασης 1 σε μη κατευθυνόμενους γράφους με κυκλώματα.

το Fibonaccci heap έχει χρόνο εκτέλεσης $\mathcal{O}(\lg|V|)$ [4]. Η κατάχωρηση απαιτεί σταθερό χρόνο και επομένως ο συνολικός χρόνος κάθε εκτέλεσης της γραμμής 16 είναι $\mathcal{O}(\lg|V|)$. Το σώμα του while μπορεί να εκτελεσθεί έως και $|V|$ φορές.

Algorithm 1 The algorithm is based on Dijktstra's algorithm implementation by T. H. Cormen and C. E. Leiserson and R. L. Rivest and C. Stein [4].

```
 1: procedure MAXPP(G, s, t, w)
 2:     for all e ∈ E do
 3:         w(e) ← 1.0 − w(e)
 4:     FibonacciHeap F ← ∅
 5:     array P ← ∅
 6:     array π ← ∅ · |V|
 7:     array L ← ∅ · |V|
 8:     π[s] ← NIL
 9:     L[s] ← 0
10:     for all v ∈ V − {s} do
11:         π[v] ← NIL
12:         L[v] ← 1.0
13:     for all v ∈ V do
14:         F.insert(v, L[v])
15:     while F ≠ ∅ do
16:         u ← F.extract_min()
17:         P ← P ∪ {u}
18:         for all v ∈ u.AdjLst AND u ≠ s do
19:             if L[v] > L[u] · w(u, v) then
20:                 L[v] ← L[u] · w(u, v)
21:                 π[v] ← u
22:                 F.decrease_key(v, L[v])
23:         if u == s then
24:             for all v ∈ u.AdjLst do
25:                 if L[v] > w(u, v) then
26:                     L[v] ← w(u, v)
27:                     π[v] ← u
28:                     F.decrease_key(v, L[v])
29:     return P, π
```

Η γραμμή 17 εκτελείται κάθε φορά σε σταθερό χρόνο. Σε σταθερό χρόνο εκτελείται και κάθε μία από τις γραμμές 19, 20, 21 και 22 [4]. Το σώμα της for των γραμμών $18-22$ εκτελείται το πολύ $|E|$ φορές. Άρα ο χρόνος εκτέλεσης είναι $\mathcal{O}(|E|)$.

Η γραμμή 23 εκτελείται $|V|$ φορές σε σταθερό χρόνο ενώ το σώμα της (γραμμές $24-28$) εκτελείται μία φορά μόνο. Το σώμα της for εκτελείται το πολύ $|E|$ φορές και κάθε μία από τις γραμμές $25-28$ χρειάζεται σταθερό χρόνο. Επομένως, οι γραμμές $23-28$ απαιτούν $\mathcal{O}(|E|)$ χρόνο.

Τέλος, η γραμμή 29 έχει σταθερό χρόνο εκτέλεσης $\mathcal{O}(1)$. Βάσει όλων των παραπάνω, ο χρόνος εκτέλεσης του Αλγορίθμου 1 είναι $T(|V|,|E|) = \mathcal{O}(|E|) + \mathcal{O}(|V|) + \mathcal{O}(|V|\lg|V| + |E|)$, δηλαδή

$$T(|V|,|E|) = \mathcal{O}(|V|\lg(|V|) + |E|). \qquad (2)$$

Η σχέση (2) είναι σύμφωνη με τον χρόνο εκτέλεσης του αλγορίθμου του Dijkstra με Fibonaccci heap. [1][2][4]

Το μονοπάτι ελάχιστης πιθανότητας αποτυχούς αποστολής στον $\neg G$ το οποίο ταυτίζεται με το μονοπάτι μέγιστης πιθανότητας επιτυχημένης αποστολής είναι αποθηκευμένο με τη μορφή κορυφών στον πίνακα $\mathcal{P} = \langle s, \dots, v_i, \dots, t\rangle$. Στον πίνακα π είναι αποθηκευμένος για κάθε κορυφή v του G ο γονέας της στο μονοπάτι $\mathcal{P}$ (εάν η v είναι η s ή $v \notin \mathcal{P}$, τότε $\pi[v] =$ **NIL**).

Πρόβλημα 2

Πρόβλημα 2 *Μια αλυσίδα fast food πρόκειται να ανοίξει μια σειρά από εστιατόρια κατά μήκος της Εγνατίας. Οι n πιθανές τοποθεσίες έχουν αποστάσεις από την αρχή της Εγνατίας σε αύξουσα σειρά $m_1, m_2, \dots, m_n$ σε μέτρα. Το προσδοκώμενο κέρδος από το άνοιγμα ενός εστιατορίου στην τοποθεσία i είναι p_i, $i = 1, 2, \dots n$. Σε κάθε τοποθεσία η αλυσίδα μπορεί να ανοίξει μόνο ένα εστιατόριο. Επιπλέον, δύο εστιατόρια πρέπει να απέχουν μεταξύ τους τουλάχιστον k μέτρα.*

Μέγιστο προσδοκώμενο κέρδος

Βέλτιστη υποδομή

Συμβολίζουμε το μέγιστο προδοκώμενο κέρδος, έχοντας στη διάθεσή μας τις τοποθεσίες $m_1, \dots, m_i$, με R_i. Χρειάζεται να σπάσουμε το πρόβλημα στα εξής δύο· στο μέγιστο προσδοκώμενο κέρδος για το σύνολο των τοποθεσιών πριν την m_i και το προσδοκώμενο κέρδος της τοποθεσίας m_i. Αρχικά, υποθέτουμε πως $m_i - m_j \geqslant k$. Τότε, $R_i = R_j + p_i$ όπου R_j είναι το μέγιστο κέρδος έχοντας στη διάθεσή μας τις τοποθεσίες $m_1, \dots, m_{i-1}$ και p_i το προσδοκώμενο κέρδος ενός εστιατορίου στην τοποθεσία m_i. Θα αποδείξουμε πως η R_j είναι μέρος της R_i.

Έστω πως R_j δεν είναι το μέγιστο προσδοκώμενο κέρδος έχοντας στη διάθεσή μας τις τοποθεσίες $m_1, \dots, m_{i-1}$. Τότε, υπάρχει μία τιμή $R_f > R_j$. Αυτό συνεπάγεται πως υπάρχει μία νέα λύση R'_i η οποία είναι μεγαλύτερη της μέγιστης. Άτοπον. ■

Τώρα θα εξετάσουμε τη γενική περίπτωση όπου η συνθήκη $m_i - m_j \geqslant k$ δεν ικανοποιείται απαραί-

τητα. Στην περίπτωση αυτή η προηγούμενη λύση $(R_j + p_i)$ παίρνει τη μορφή $R_j + p_i \cdot \delta[m_i, m_j]$, όπου $\delta : [m_1, \dots, m_n] \times [m_1, \dots, m_n] \to \{0, 1\}$ με τύπο

$$\delta[m_x, m_y] = \begin{cases} 0, & \text{if } m_x - m_y < k \\ 1, & \text{if } m_x - m_y \geqslant k \end{cases}.$$

Στην περίπτωση όπου $\delta[m_i, m_{i-1}] = 0$, τότε δύναται να είναι $p_i > R_j$. Επομένως, η βέλτιστη υποδομή είναι $R_i = \max\{p_i, R_j + p_i \cdot \delta(m_i, m_j)\}$.

Αναδρομική λύση

Εκφράζουμε την βέλτιστη λύση ενός προβλήματος μεγέθους i συναρτήσει των βέλτιστων λύσεων υποπροβλημάτων μεγέθους $j < i$. Στην προκειμένη αναζητούμε το μέγιστο προσδοκώμενο κέρδος, R_i, όταν είναι διαθέσιμες οι τοποθεσίες $m_j, \dots, m_i,\ 1 \leqslant j \leqslant i \leqslant n$. Θέτωντας $R_1 = p_1$ για $j = i = 1$, για $m_j, \dots, m_i,\ 1 \leqslant j < i \leqslant n$ είναι

$$R_i = \max\left\{p_i, \max_{j<i}\left\{R_j + \delta[m_i, m_j] \cdot p_i\right\}\right\} \quad (3)$$

Υπολογισμός βέλτιστης λύσης

Η βέλτιστη λύση κάθε υποπροβλήματος αποθηκεύεται σε έναν πίνακα R. Επομένως, λύνοντας τα προβλήματα κατά αύξουσα σειρά ως προς το μέγεθός τους, κάθε αναδρομή δεν χρειάζεται να υπολογίσει εκ νέου την βέλτιση λύση για κάθε υποπρόβλημα αλλά να την αναζητήσει στον πίνακα R. Βάσει της σχέσης (3) προκύπτει ο Αλγόριθμος 2.

Algorithm 2 The procedure takes as input an array p of size n containing the expected profit of each of the possible locations, an array m of size n containing the distance of each possible location from a fixed origin point and a positive real constant k.

```
1:  procedure RESTAURANTS(p[n], m[n], k)
2:      int n ← length_of(p)
3:      array R[1, ..., n] ← ∅
4:      for all p_i ∈ p do
5:          R[p_i] ← p_i
6:      for i = [2, n] do
7:          for j = [1, i) do
8:              if m[i] − m[j] ⩾ k then
9:                  R[i] ← max{p[i], R[j] + p[i]}
10:             else
11:                 R[i] ← max{R[i], R[j]}
12:     return R[n]
```

Ορθότητα αλγορίθμου

Χρόνος εκτέλεσης αλγορίθμου

Στην γραμμή 2 του Αλγορίθμου 2 υπολογίζεται το μέγεθος του πίνακα p σε σταθερό χρόνο και το αποτέλεσμα καταχωρείται σε μία νέα μεταβλητή n σε σταθερό χρόνο $\mathcal{O}(1)$.

Στην γραμμή 3 του Αλγόριθμος 2 δημιουργείται ένας νέος κενός πίνακας μεγέθους n σε σταθερό χρόνο $\mathcal{O}(1)$.

Το σώμα του επαναληπτικού βρόχου των γραμμών $4-5$ εκτελείται n φορές. Σε κάθε εκτέλεση της γραμμής 5 πραγματοποιείται καταχώρηση μίας τιμής σε κάποια θέση του πίνακα R το οποίο απαιτεί σταθερό χρόνο $\mathcal{O}(1)$. Επομένως, συνολικά η for των γραμμών $4-5$ χρειάζεται $\mathcal{O}(n)$ χρόνο.

Το σώμα του for της γραμμής 6 εκτελείται $n-1$ φορές. Το σώμα του for της γραμμής 7 εκτελείται $2 \cdot (n-1)$ φορές. Ο έλεγχος της συνθήκης της γραμμής 8 γίνεται σε σταθερό χρόνο $\mathcal{O}(1)$. Η γραμμή 9 πρώτα υπολογίζει το άθροισμα $R[j] + p[i]$ σε σταθερό χρόνο. Έπειτα, υπολογίζει το μέγιστο μεταξύ των $p[i]$ και $R[j] + p[i]$ σε σταθερό χρόνο και τέλος καταχωρεί το αποτέλεσμα στον πίνακα R σε επίσης σταθερό χρόνο. Συνεπώς, η γραμμή 9 εκτελεί σειριακά εργασίες σταθερού χρόνου. Αυτό σημαίνει πως ο χρόνος εκτέλεσής της είναι $\mathcal{O}(1)$.

Η γραμμή 11 εκτελεί τις παρόμοιες ενέργειες με την γραμμή 9 πάνω σε διαφορετικές τιμές. Επομένως, η ανάλυση της προηγούμενης παραγράφου για τη γραμμή 9 εφαρμόζεται και για την γραμμή 11 και προκύπτει πως ο χρόνος εκτέλεσής της είναι $\mathcal{O}(1)$.

Βάσει των παραπάνω ο συνολικός χρόνος εκτέλεσης του Αλγορίθμου 2 είναι $T(n) = \mathcal{O}(1) + n \cdot \mathcal{O}(1) + (n-1) \cdot [(n-2) \cdot \mathcal{O}(1) + \mathcal{O}(1)]$. Δηλαδή,

$$T(n) = \mathcal{O}(n^2). \quad (4)$$

Αναφορές

[1] Dasgupta, S. and Papadimitriou, C.H. and Vazirani, U.V. *Algorithms*. McGraw-Hill Higher Education, 2006. ISBN: 9780077388492.

[2] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson/Addison-Wesley, 2006. ISBN: 9780321295354.

[3] Robert Sedgewick and Kevin Wayne. *Algorithms*. Fourth Edition. Addison-Wesley, 2011. ISBN: 9780321573513.

[4] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*. Third Edition. The MIT Press, 2009. ISBN: 9780262033848.