[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.*

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 [1], 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 $w : E \to [0, 1]$. Εφ' όσον τα βάρη των ακμών είναι μη αρνητικά, δύναται να χρησιμοποιηθεί ο αλγόριθμος του Dijkstra ως βάση.

Οι διαφορές με τον κλασικό αλγόριθμο του Dijkstra είναι οι ακόλουθες. Αρχικά, αναζήτουμε το μονοπάτι μέγιστου μήκους–πιθανότητας μεταξύ δύο κορυφών s και t αντί του μονοπατιού ελάχιστου μήκους. Επομένως, καθώς εξετάζουμε τις κορυφές του γράφου, η συνθήκη για την ενημέρωση των αποστάσεων των κορυφών από την αφετηρία s είναι $\mathcal{D}[v] < \mathcal{D}[u] \cdot w(u, v),\ \forall (u, v) \in E$.

Λόγω της μορφής της παραπάνω ανισότητας, οι αρχικές αποστάσεις όλων των κορυφών έχουν τεθεί ίσες 0, με εξαίρεση την απόσταση της κορυφής s η οποία αρχικοποιείται σε $\mathcal{D}[s] = 1.0$.

Τέλος, τα γεγονότα είναι ανεξάρτητα. Δηλαδή, για δύο ακμές $(x, y), (y, z) \in E$ η πιθανότητα επιτυχούς μετάδοσης του σήματος από την κορυφή x στην κορυφή z, μέσω των ακμών αυτών είναι $w(x, y) \cdot w(y, z)$. Προκειμένου το γινόμενο $w(x, y) \cdot w(y, z)$ να παίρνει την μέγιστη δυνατή τιμή, πρέπει και τα $w(x, y), w(y, z)$ να είναι τα κατά το δυνατόν μέγιστα.

Σε κάθε επανάληψη του βρόχου των γραμμών 13 – 19 επιλέγουμε και αφαιρούμε από την ουρά Q την κορυφή με τη μέγιστη απόσταση-πιθανότητα και προχωράμε στη διερεύνηση των κορυφών της γειτονιάς της και την ενημέρωση των αποστάσεών τους (αν χρειάζεται). Έτσι, στο τέλος ο πίνακας π περιέχει τον γονέα κάθε κορυφής στο δένδρο μέγιστης πιθανότητας (ο γονέας της ρίζας s είναι **NIL**) και στον πίνακα $\mathcal{D}$ είναι καταχωρημένη η απόσταση-πιθανότητα κάθε κορυφής από τη ρίζα του δένδρου.

Algorithm 1 The algorithm is based on the implementation of Dijkstra's algorithm by T. H. Cormen and C. E. Leiserson and R. L. Rivest and C. Stein.[2]

```
1:  procedure MAXPP(G, s, t, w)
2:      MaxQueue Q ← ∅ ▷ maximum priority queue
3:      array P ← ∅
4:      array π ← ∅ · |V|
5:      array D ← ∅ · |V|
6:      π[s] ← NIL
7:      D[s] ← 1.0
8:      for all v ∈ V − {s} do
9:          π[v] ← NIL
10:         D[v] ← 0
11:     for all v ∈ V do
12:         insert(Q, v, D[v])
13:     while Q ≠ do
14:         u ← extract_max (Q)
15:         for all v ∈ AdjLst[u] do
16:             if D[v] < D[u] · w(u, v) then
17:                 D[v] ← D[u] · w(u, v)
18:                 update_priority (Q, v, D[v])
19:                 π[v] ← u
20:     if π[t] = NIL then
21:         return The requested path does not exist.
22:     P ← P ∪ {t}
23:     int i ← −1
24:     while π[P[i]] ≠ NIL do
25:         P[i − 1] ← π[P[i]]
26:         i ← i − 1
27:     return P
```

Για την κατασκευή του μονοπατιού μέγιστης πιθανότητας επιτυχούς αποστολής από την κορυφή s προς την κορυφή t –εάν υπάρχει– ακολουθούμε τα εξής βήματα. Αρχικά, σε έναν κενό πίνακα $\mathcal{P}$ προσθέτουμε την κορυφή t. Σκοπός μας είναι να διασχίσουμε το δένδρο από την κορυφή t προς την κορυφή s προσθέτοντας την κορυφή που συναντάμε σε κάθε βήμα, στο μονοπάτι $\mathcal{P}$.

Μετά την εκτέλεση της γραμμής 22 ο πίνακας $\mathcal{P}$ περιέχει μία μόνο κορυφή, την t. Ελέγχουμε εάν ο γονέας της είναι ίσος με **NIL**. Εάν όχι, προσθέτουμε τον γονέα στην πρώτη θέση του $\mathcal{P}$ και επαναλαμβάνουμε τον προηγούμενο έλεγχο για τον γονέα της κορυφής του μονοπατιού που προστέθηκε τελευταία. Η διαδικασία επαναλαμβάνεται έως ότου ο γονέας της τελευταίας κορυφής που προστέθηκε στο μονοπάτι $\mathcal{P}$ να είναι **NIL**. Τότε, η κορυφή αυτή είναι αναγκαστικά η s και το μονοπάτι έχει τη μορφή $\mathcal{P} = \langle s, v_0, \dots, v_k, t\rangle$.

Σε ό,τι αφορά τον χρόνο εκτέλεσης του Αλγορίθμου 1 ισχύουν τα παρακάτω.

Στην γραμμή 2 δημιουργείται ένα κενό maximum priority queue Q σε χρόνο $\mathcal{O}(1)$. Στις γραμμές 2, 3 και 4 δημιουργούνται τρεις νέοι κενοί πίνακες, ο καθένας σε σταθερό χρόνο.

Σε κάθε μία από τις γραμμές 6 και 7 πραγματοποιείται μία καταχώρηση σε συγκεκριμένη θέση των πινάκων π και $\mathcal{D}$ αντίστοιχα, σε σταθερό χρόνο. Συνεπώς, το σύνολο των γραμμών $2-7$ εκτελείται σε χρόνο $\mathcal{O}(1)$.

Το σώμα της **for** των γραμμών $8-10$ εκτελείται $|V|$ φορές. Κάθε μία από τις γραμμές 9 και 10 εκτελεί σε σταθερό χρόνο μία καταχώρηση σε συγκεκριμένη θέση των πινάκων π και $\mathcal{D}$ αντίστοιχα. Άρα, ο συνολικός χρόνος εκτέλεσης των γραμμών $8-10$ είναι $\mathcal{O}(|V|-1) = \mathcal{O}(|V|)$.

Η γραμμή 12 τρέχει σε χρόνο $T_{Q,\text{insert}}$, $|V|$ φορές. Ο χρόνος $T_{Q,\text{insert}}$ εξαρτάται από την υλοποίηση του priority queue. Τότε, ο συνολικός χρόνος εκτέλεσης των γραμμών $11-12$ είναι $|V| \cdot T_{Q,\text{insert}}$.

Το σώμα του **while** των γραμμών $13-19$ εκτελείται $|V|$ φορές. Η μέθοδος extract_max$(\cdot,\cdot)$ για το Q εκτελείται σε χρόνο $T_{Q,\text{extract\_max}}$ και η καταχώρηση του αποτελέσματος γίνεται σε σταθερό χρόνο. Δηλαδή, η γραμμή 14 εκτελείται σε $T_{Q,\text{extract\_max}} + \mathcal{O}(1) = T_{Q,\text{extract\_max}}$ χρόνο. Το σώμα της **for** των γραμμών $15-19$ εκτελείται το πολύ $|E|$ φορές. Σε κάθε εκτέλεση, ο έλεγχος της συνθήκης της γραμμής 16 πραγματοποιείται σε σταθερό χρόνο. Ο πολλαπλασιασμός και η καταχώρηση του αποτελέσματος στη γραμμή 17 εκτελούνται σε σταθερό χρόνο. Ομοίως και η καταχώρηση της γραμμής 19. Η μέθοδος update_priority$(\cdot,\cdot,\cdot)$ απαιτεί χρόνο $T_{Q,\text{update\_priority}}$ και εξαρτάται από την υλοποίηση της Q.

Ο έλεγχος της γραμμής 20 εκτελείται μία φορά σε σταθερό χρόνο και αν είναι αληθής, ο program counter προχωράει στην γραμμή 21 η οποία εκτελείται σε χρόνο $\mathcal{O}(1)$ και τερματίζει την διαδικασία.

Οι καταχωρήσεις των γραμμών 22 και 23 απαιτούν σταθερό χρόνο.

Το σώμα του **while** των γραμμών $24-26$ μπορεί να εκτελεστεί έως και $|V|$ φορές. Κάθε μία από τις γραμμές 25 και 26 τρέχει σε σταθερό χρόνο $\mathcal{O}(1)$. Άρα, ο συνολικός χρόνος εκτέλεσης των γραμμών $24-26$ είναι $\mathcal{O}(|V|)$.

Η γραμμή 27 τρέχει σε σταθερό χρόνο. Τελικά, ο συνολικός χρόνος εκτέλεσης του Αλγορίθμου 1 είναι $T(|V|,|E|) = \mathcal{O}(1) + \mathcal{O}(|V|) + |V| \cdot T_{Q,\text{insert}} + |V| \cdot T_{Q,\text{extract\_min}} + |E| \cdot T_{Q,\text{update\_priority}} + \mathcal{O}(|V|)$. Δηλαδή,

$$\begin{aligned} T(|V|,|E|) = &|V| \cdot T_{Q,\text{insert}} \\ &+|V| \cdot T_{Q,\text{extract\_min}} \\ &+|E| \cdot T_{Q,\text{update\_priority}}. \end{aligned} \tag{1}$$

Η σχέση (1) είναι σύμφωνη με τον χρόνο εκτέλεσης του αλγορίθμου του Dijkstra[2][1] και εξαρτάται από την υλοποίηση του priority queue.

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 απαραίτητα. Στην περίπτωση αυτή η προηγούμενη λύση $\left(R_j + p_i\right)$ 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 $R_i = \max\left\{p_i, R_j + p_i \cdot \delta(m_i, m_j)\right\}$.

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

$$R_i = \max\left\{p_i, \max_{j<i}\left\{R_j + \delta[m_i, m_j] \cdot p_i\right\}\right\} \quad (2)$$

Υπολογισμός βέλτιστης λύσης

Βάσει της σχέσης (2) προκύπτει ο Αλγόριθμος 2. 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.

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] ← R[j] + p[i]
10:             else
11:                 R[i] ← max{R[i], R[j]}
12:     return R[n]
```

Ορθότητα αλγορίθμου

Loop invariant *Μετά από κάθε i-στή επανάληψη του βρόχου των γραμμών $6 - 11$, η θέση i του πίνακα R περιέχει το μέγιστο προσδοκώμενο κέρδος έχοντας εστιατόρια από το σημείο m_1 έως και m_i τηρώντας τον περιορισμό σχετικά με τις αποστάσεις δύο οποιονδήποτε εστιατορίων.*

Αρχικοποίηση

Πριν την είσοδο του program counter στον επαναληπτικό βρόχο των γραμμών $6 - 11$, στην θέση 1 του πίνακα R είναι αποθηκευμένο το προσδοκώμενο κέρδος του εστιατορίου της τοποθεσίας m_1, $R[1] = p[1]$, το οποίο είναι όντως το μέγιστο προσδοκώμενο συνολικό κέρδος στην περίπτωση που έχουμε στη διάθεσή μας μία μόνο τοποθεσία. Το loop invariant ικανοποιείται πριν την πρώτη εκτέλεση του έξω επαναληπτικού βρόχου.

Διατήρηση

Έστω πως το loop invariant ικανοποιείται μετά την i-στή εκτέλεση του επαναληπτικού βρόχου των γραμμών $6 - 11$, δηλαδή $R[i]$ είναι η τιμή του μεγίστου προσδοκώμενου κέρδους έχοντας διαθέσιμες τις τοποθεσίες m_1 έως και m_i. Για την επόμενη εκτέλεση του εξωτερικού βρόχου $(i + 1)$ πριν την πρώτη εκτέλεση του εσωτερικού επαναληπτικού βρόχου είναι $R[i + 1] = p[i + 1]$ και για κάθε εκτέλεση του έσω βρόχου $(\forall j \in [1, i])$ υπάρχουν δύο περιπτώσεις.

i. Οι τοποθεσίες $m[i+1]$ και $m[j]$ ικανοποιούν τη συνθήκη $m[i+1]-m[j] \geqslant k$. Τότε, στο μέγιστο προσδοκώμενο κέρδος με διαθέσιμες τοποθεσίες τις $m_1, \dots, m_j$, $j \leqslant i < i+1$ προστίθεται το κέρδος του εστιατορίου της τοποθεσίας $m[i+1]$ και το άθροισμα αυτό καταχωρείται στην θέση $R[i+1]$.

ii. Οι τοποθεσίες $m[i+1]$ και $m[j]$ δεν ικανοποιούν τη συνθήκη $m[i+1]-m[j] \geqslant k$. Τότε, μεταξύ του μεγίστου προσδοκώμενου κέρδους έχοντας διαθέσιμες τις τοποθεσίες $m_1, \dots, m_{i+1}$, $R[i+1]$ —η οποία πριν την πρώτη εκτέλεση της γραμμής 8 για δεδομένο i περιέχει την τιμή $p[i+1]$— και του μεγίστου προσδοκώμενου κέρδους έχοντας διαθέσιμες τις τοποθεσίες $m_1, \dots, m_j$, $R[j]$, για $j \leqslant i < i+1$ επιλέγεται το μέγιστο και καταχωρείται στη θέση $R[i+1]$.

Άρα, με το πέρας της $(i+1)$-στής εκτέλεσης του εξωτερικού επαναληπτικού βρόχου η θέση $R[i+1]$ περιέχει το μέγιστο προσδοκώμενο κέρδος έχοντας διαθέσιμες τις τοποθεσίες $m_1, \dots, m_{i+1}$. Το loop invariant διατηρείται.

Τερματισμός

Η συνθήκη η οποία επιφέρει τον τερματισμό του εξωτερικού επαναληπτικού βρόχου (γραμμές $6-11$) είναι $i = n+1 > n$ αφού, μετά το πέρας κάθε εκτέλεσης του βρόχου, ο μετρητής i προσαυξάνεται κατά 1. Για $i = n+1$, το σώμα του βρόχου δεν εκτελείται. Αυτό σημαίνει πως στη θέση $R[n]$ είναι καταχωρημένο το μέγιστο προσδοκώμενο κέρδος έχοντας διαθέσιμες τις τοποθεσίες $m_1, \dots, m_n$.

Βάσει των παραπάνω, προκύπτει η ορθότητα του αλγορίθμου. Αξίζει να σημειωθεί πως ο δυναμικός προγραμματισμός εφαρμόστηκε στο συγκεκριμένο πρόβλημα βελτιστοποίησης διότι υπήρχε καλώς ορισμένη βέλτιστη υποδομή και επειδή υπάρχει επικάλυψη των υποπροβλημάτων, δηλαδή σε κάθε αναδρομή συναντώνται συγκεκριμένα υποπροβλήματα αντί κάθε φορά να δημιουργούνται νέα όπως στην τεχνική *διαίρει & βασίλευε*.[2]

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 Αλγορίθμου 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

$$\sum_{i=2}^{n}\left(\sum_{j=1}^{i}1\right) = \sum_{i=2}^{n} i = \frac{1}{2}n^2 + \frac{1}{2}n - 1 \text{ φορές.}$$

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 $T(n) = \mathcal{O}(1) + n \cdot \mathcal{O}(1) + \left(\frac{1}{2}n^2 + \frac{1}{2}n - 1\right) \cdot \mathcal{O}(1)$. Δηλαδή,

$$T(n) = \mathcal{O}(n^2). \tag{3}$$

Αναφορές

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

[2] 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.