Algorithms - Assignment 2

Καζγκούτης Αθανάσιος Παπαδόπουλος Δημήτριος-Λάζαρος

May 24, 2023

Πρόβλημα 1

Η συνάρτηση MaxSuccessPath παίρνει για ορίσματα τον γράφο G = (V,E), το βάρος κάθε ακμής (p) το οποίο εκφράζει την πιθανότητα ότι ένα πακέτο το οποίο στέλνεται από τη μία συσκευή θα φτάσει στην άλλη χωρίς να χαθεί, τον αφετηριακό κόμβο (s) και τον κόμβο προορισμού (t). Στην γραμμή 2 εκτελείται η απόδοση αρχικών τιμών. Στην γραμμή 3 ο αλγόριθμος δημιουργεί ένα κενό σύνολο (D) στο οποίο θα εισάγει τους κόμβους των οποίων οι τελικές πιθανότητες από την αφετηρία (s) έχουν ήδη προσδιοριστεί. Στην γραμμή 4 δημιουργούμε μία ουρά προτεραιότητας μεγίστου Q για τους κόμβους με κλειδιά τις τιμές των πεδίων p και ορίζουμε ως αρχικό περιεχόμενο το σύνολο των κόμβων. Στην συνέχεια κάθε φορά που διατρέχεται ο βρόχος while στην γραμμή 6 αφαιρείται από την ουρά ένας κόμβος u που έχει την μέγιστη εκτίμηση πιθανότητας (u.p) και προστίθεται στο σύνολο (D) στην γραμμή 9. Στις γραμμές 10-11 ο αλγόριθμος χαλαρώνει όλες τις ακμές (u,v) που εκκινούν από τον u. Στην γραμμή 7-8 ελέγχουμε αν ο κόμβος που βγήκε από την Q και άρα έχει προσδιοριστεί η τελική του πιθανότητα είναι ο κόμβος προορισμού (t), ώστε να τυπώσουμε την διαδρομή s→t.

```
1  function MaxSuccessPath(G,p,s,t)
2      Initialize(G,s)
3      D = ∅
4      Q = G.V
5      while Q ≠ ∅
6          u = ExtractMaxQ
7          if (u == t)
8              PrintPath(u)
9          D = D ∪ {u}
10         for each vertex v ∈ G.Adj[u]
11             Relax(u,v,p)
```

Η συνάρτηση Initialize αποδίδει τις αρχικές τιμές, όπου μετά το κάλεσμά της κάθε κόμβος v δεν θα έχει προκάτοχο (v.π = NULL), η πιθανότητα του αφετηριακού κόμβου (s) θα είναι (s.p = 1) ενώ για όλους τους υπόλοιπους θα είναι ίση με το 0.

```
function Initialize(G,s)
    for each vertex v ∈ G.V
        v.p = 0
        v.π = NULL
    s.p = 1
```

Η διαδικαίσα της Χαλάρωσης μίας ακμής (u,v) έχει ως εξής: ελέγχουμε εάν μπορούμε να βελτιώσουμε την μέγιστη πιθανότητα (p) της διαδρομής για τον κόμβο v διερχόμενοι μεσω του u. Στην περίπτωση που η διαδρομή με την μέγιστη πιθανότητα που έχει βρεθεί μέχρι στιγμής έως τον v μπορεί να βελτιωθεί διερχόμενοι από τον u, τότε ενημερώνουμε την εκτίμηση (v.p) και τον προκάτοχο (v.π). Το p_{uv} είναι η πιθανότητα σωστης λήψης μυνήματος ανάμεσα στους κόμβους (u,v).

```
function Relax(u,v,p)
    if (v.p < u.p · p_uv)
        v.p = u.p · p_uv
        v.π = u
```

Η συνάρτηση printPath παίρνει για όρισμα τον κόμβο προορισμού (u) για τον οποίο θα βρεί το μονοπάτι που οδηγεί σε αυτόν. Χρησιμοποιούμε τον πίνακα R όπου θα αποθηκεύουμε τους κόμβους. Στην γραμμή 3 ο πρώτος κόμβος στον R είναι ο κόμβος προορισμού και τρέχουμε την while μέχρι να βρούμε κόμβο που να μην έχει προκάτοχο δηλαδή να είναι ο αφετηριακός. Στην γραμμή 5 αποθηκεύουμε στον R τον προκάτοχο του αντίστοιχου κόμβου και στην γραμμή 6 u με τον προκάτοχο του ώστε να τρέξουμε το μονοπάτι ανάποδα. Τέλος στις γραμμές 8-9 τυπώνουμε το μονοπάτι ξεκινώντας από το τέλος του πίνακα R για να είναι οι κόμβοι με την σωστή σειρά.

```
1 function printPath(u)
2     i = 1
3     R[0] = u
4     while u.π ≠ NULL
5         R[i] = u.π
6         u = u.π
7         i = i + 1
8     for j = i-1 : -1 : 0
9         print → R[j]
```

Πρόβλημα 2

- **Το πρόβλημα**
  Το πρόβλημα που πρέπει να λύσουμε είναι να βρούμε το μέγιστο προσδοκώμενο συνολικό κέρδος όταν έχουμε (n) πιθανές τοποθεσίες.

- **Τα υποπροβλήματα**
  Θεωρούμε $P[i]$ είναι το μέγιστο προσδοκώμενο συνολικό κέρδος αν ανοίξουμε τα εστιατόρια από το σημείο m_1 μέχρι το m_i. Επομένως τα υποπροβλήματα είναι να βρούμε τα μέγιστα κέρδη για $\forall i \in [0, n]$. Όπου για $i = n, P[n]$ είναι η λύση στο πρόβλημα που αναζητάμε.

- **Η βέλτιστη Υποδομή**
  Αρχικά στην βασική περίπτωση αν $i = 0$ τότε δεν υπάρχει καμία τοποθεσία και το κέρδος είναι μηδέν $P[0] = 0$. Στην συνέχεια αν το $i > 0$ υπάρχουν δύο επιλογές

  1. Να μην ανοίξουμε εστιατόριο στην τοποθεσία i
     Τότε το μέγιστο κέρδος $P[i]$ θα είναι το μέγιστο κέρδος των προηγούμενων $i - 1$ τοποθεσιών, δηλαδή $P[i] = P[i - 1]$
  2. Να ανοίξουμε εστιατόριο στην τοποθεσία i
     Τότε το συνολικό κέρδος θα είναι το κέρδος από το άνοιγμα του εστιατορίου στην τοποθεσία m_i δηλαδή το p_i σύν το μέγιστο κέρδος από τα εστιατόρια μέχρι την τοποθεσία m_d δηλαδή $P[d]$ όπου d είναι ο μεγαλύτερος δείκτης για τον οποίο ισχύει ότι $d < i$ και $m_d \leq m_i - k$ (επειδή δύο εστιατόρια πρέπει να απέχουν μεταξύ τους τουλάχιστον k μέτρα). Επομένως $P[i] = p_i + P[d]$

Η συνάρτηση MaxProfit παίρνει για ορίσματα τις τοποθεσίες σε μέτρα απο την αρχή της Εγνατίας (m) και τα προσδοκώμενα κέρδη κάθε τοποθεσίας (p). Η for τρέχει n φορές όπου για κάθε i υπολογίζει ποιο κέρδος απο τις δύο επιλογές που αναλύσαμε στην βέλτιστη υποδομή είναι μεγαλύτερο και το αποθηκεύει στο $P[i]$ και τέλος επιστρέφει την λύση στο αρχικό πρόβλημά μας το $P[n]$.

```
function MaxProfit(m,p)
    P[0] = 0
    for i=1 to n
        d = FindMaxIndex-d(i)
        P[i] = max{P[i-1], p_i + P[d]}
    return P[n]
```

Η συνάρτηση FindMaxIndex−d βρίσκει τον μεγαλύτερο δείκτη για τον οποίο ισχύει ότι $d < i$ και $m_d \leq m_i - k$. Στις γραμμές 2-3 ελέγχει αν το $i = 1$ γιατί τότε δεν υπάρχει προηγούμενος δείκτης και για αυτό επιστρέφει 0 ώστε στην MaxProfit να χρησιμοποιήσει $P[d] = P[0]$. Στην while ελέγχει κάθε φορά αν η τοποθεσία με δείτη d απέχει λιγότερο από k μέτρα απο τον i ώστε να μειώσει και άλλο τον δείκτη d μέχρι να βρεί τον μεγαλύτερο d που να απέχει τουλάχιστον k μετρα απο τον i. Στις γραμμές 7-8 κάνει έναν επιπλέον έλεγχο αν το d μειούμενο συνεχώς φτάσει στην τιμή 0, να επιστρέψει 0 για τον ίδιο λόγο με την παραπάνω if στις γραμμές 2-3. Η περίπτωση το d να φτασει στο 0 γίνεται αν οι τοποθεσίες των εστιατορίων είναι πολύ κοντά και μαζεμένες σε χώρο μικρότερο απο k μέτρα.

```
1 function FindMaxIndex-d(i)
2     if(i=1)
3         return 0
4     d = i-1
```

```
    while(m_d ≥ m_i − k)
        d = d − 1
        if(d=0)
            return 0
    return d
```