2η Σειρά Ασκήσεων, Αλγόριθμοι και Πολυπλοκότητα

Πέτρος Αυγερίνος 03115074

18 Δεκέμβρη, 2023

Contents

1 Πολύχρωμος Πεζόδρομος

Το πρόβλημα θα λυθεί με Δυναμικό Προγραμματισμό. Θα ορίσουμε δύο δείκτες i, j οι οποίοι εκφράζουν το διάστημα πλακών που επιθυμούμε να βάψουμε και c_i το χρώμα που θα βάψουμε την πλάκα i. Για μία μέρα, μπορώ να βάψω το χρώμα c_i από το σημείο i μέχρι κ επόμενες πλάκες όπου $c_\kappa = c_i$. Έτσι υπολοίπονται $j-(\kappa+1)$ πλάκες οι οποίες θα πρέπει να βαφτούν μέ άλλο χρώμα κάποια επόμενη μέρα. Αναδρομικά αναζητώ τις ελάχιστες δυνατές ημέρες που χρειάζονται για να βαφτούν οι πλάκες. Σαφώς για $i > j$ ο ελάχιστος αριθμός ημερών είναι μηδενικός, ενώ για $j = i$ αρκεί μονάχα μία μέρα. Ο διαχωρισμός που ορίσαμε προηγούμενως προκαλεί την άθροιση των δύο διαφορετικών συνόλων πλακών, έτσι για την εύρεση του ελάχιστου αριθμού ημερών για την βαφή των πλακών προκύπτει η εξής αναδρομική σχέση:

$$\text{min\_days}(i,j) = \begin{cases} 1 + min_{i\le\kappa\le j}\{\text{min\_days}(i,\kappa) + \text{min\_days}(\kappa+1,j)\} & i < j \\ 1 & i = j \\ 0 & i > j \end{cases}$$

Σχετικά με την πολυπλοκότητα του αλγορίθμου αυτού αρκεί να δούμε το state space των δεικτών διαστήματος και του δείκτη διαχωρισμού για να αντιληφθούμε ότι κάθε ένα από αυτά είναι της τάξης του n αφού έχουμε n πιθανά i,n πιθανά j, και η min θα χρειαστεί n δείκτες διαχωρισμού για την εύρεση ελαχίστου. Άρα ο αλγόριθμος είναι $O(n^3)$.

2 String Matching

2.1 1ο Ερώτημα

Για την εύρεση των προθεμάτων της συμβολοσειράς pt, στην πραγματικότητα θα δημιουργήσουμε όλα τα δυνατά substrings τα οποία ξεκινούν από την αρχή του pt και λήγουν σε κάποιο i, δηλαδή τα substrings εκείνα $pt[:i] \forall i$. Κατά την δημιουργία των substrings αυτών προκύπτουν πυρήνες $c(pt[:i]) \forall i$ οι οποίοι όταν φτάσουμε στο μήκος εκείνο του pt substring το οποίο είναι διπλάσιο από το p, θα προκύψουν πυρήνες ενδιαφέροντος. Κατά την διάρκεια των πυρήνων αυτών θα προκύψει επίσης η συμβολοσειρά p μέσα στο t και θα είναι και πυρήνας, καθώς περνούμε διαδοχικά από όλα τα πιθανά σειριακά patterns μέσα στο t και το pattern p είναι στην αρχή, θα εφαρμόζει μέρος του p πάντα στον πυρήνα, αν αυτός υπάρχει.

Παρακάτω φαίνεται ένα σύντομο παράδειγμα σε python:

In [1]:
```
text = "AABAACAADAABAABA"
pattern = "AABA"
```

In [2]:
```
pt = pattern + text
```

In [3]:
```
def kernel(string):
    n = len(string)
    lps = [0] * n

    i = 1
    length = 0
    while i < n:
        if string[i] == string[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1

    return string[:lps[-1]] if lps[-1] > 0 else "ε"
```

In [4]:
```
def prefixes(string):
    for i in range(len(string)):
        if i == 0:
            continue
        out = ""
        for char in string[:i]:
            out = out + char
        out = out + "=>" + kernel(string[:i])
        print(out)
```

In [5]:
```
prefixes(pt)
```

```
A=>ε
AA=>A
AAB=>ε
AABA=>A
AABAA=>AA
AABAAA=>AA
AABAAAB=>AAB
AABAAABA=>AABA
AABAAABAA=>AABAA
AABAAABAAC=>ε
AABAAABAACA=>A
AABAAABAACAA=>AA
AABAAABAACAAD=>ε
AABAAABAACAADA=>A
AABAAABAACAADAA=>AA
AABAAABAACAADAAB=>AAB
AABAAABAACAADAABA=>AABA
AABAAABAACAADAABAA=>AABAA
AABAAABAACAADAABAAB=>AAB
```

2.2 2ο Ερώτημα

Γνωρίζουμε ότι ο πυρήνας u μίας συμβόλοσειράς v είναι η μεγαλύτερου μήκους συμβολοσειρά για την οποία ισχύει $u \prec v$. Επομένως είναι σαφές πως ο πυρήνας αυτός είναι άνω φραγμένος στο μέγιστο της δυνατής αυτής συμβολοσειράς, έστω άνω φράγμα το μήκος του πυρήνα u ίσο με U.

Ο k-πυρήνας είναι καλά ορισμένος αφού:

1. Το μήκος μιας συμβολοσειράς είναι μη αρνητικός αριθμός.
2. Στην περίπτωση όπου δεν ορίζεται συμβολοσειρά, δεν μπορεί να οριστεί και πυρήνας.
3. Για να οριστεί πυρήνας θα πρέπει να ισχύει $u \prec v$.
4. Θα πρέπει το μήκος της u να είναι το πολύ k, αφού ο πυρήνας είναι άνω φραγμένος.
5. Θα πρέπει η u να είναι η μεγαλύτερη δυνατή συμβολοσειρά γιατί αλλιώς δεν ορίζεται πυρήνας.

Για την εύρεση ενός k-πυρήνα μπορούμε να λάβουμε τα k πρώτα στοιχεία της συμβολοσειράς και τα k τελευταία, έστω p και t, και να τα παραθέσουμε σε μια νέα συμβολοσειρά pt. Θα ορίσουμε δύο δείκτες i, j και έναν πίνακα LPS ο οποίος περιέχει την μεγαλύτερη υποσυμβολοσειρά μέσα στη pt η οποία είναι και πρόθεμα και επίθεμα της pt.

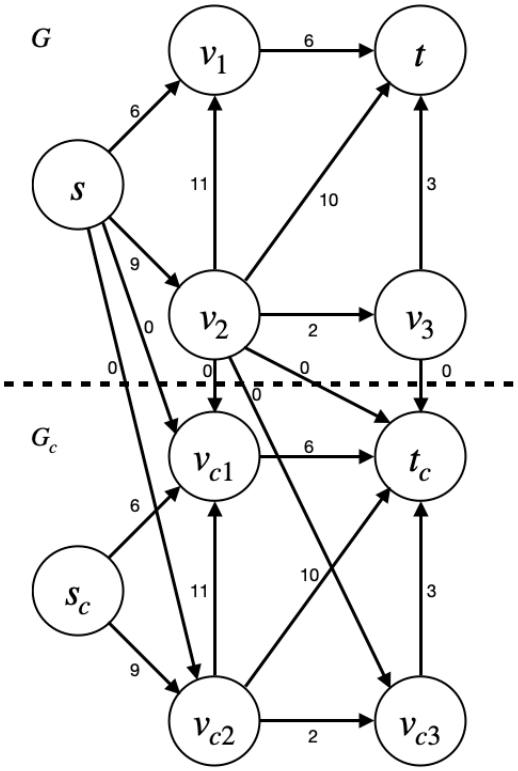
Algorithm 1 Kernel(string,k), Pythonic Notation

```
if k < string.length//2 then
    string = concat(string[: k], string[string.length − k − 1 :])
i = 1
length = 0
LPS[string.length] = 0
while i < string.length do                                   ▷ O(n)
    if string[i] == string[length] then
        length + +
        LPS[i] = length
        i + +
    else
        if length! = 0 then
            length = LPS[i − 1]
        else
            LPS[j] = 0
            j + +
if LPS[−1]! = 0 then return string[: LPS[−1]]
return "ϵ".
```

3 Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

3.1 1ο Ερώτημα

Για την επιλύση του παρόντος προβλήματος θα χρειαστεί η δημιουργία ενός πανομοιότυπου γράφου με τον G, τον οποίο θα ονομάσουμε G_c και η ένωση των δύο αυτών γράφων σε ένα G^*. Για κάθε ακμή στον γράφο G μεταξύ κόμβων u και v (κατευθυνόμενη ακμή), θα ορίσουμε μια νέα ακμή η οποία συνδέει τον γράφο G με τον G_c, πιο συγκεκριμένα τον κόμβο u και v_c, ο οποίος v_c είναι ο 'κλώνος' στον γράφο G_c του κόμβου v στον γράφο G. Οι ακμές αυτές διασύνδεσης μεταξύ των δύο γράφων θα έχουν μηδενικό κόστος διάσχυσης. Επιπρόσθετα η επιστροφή από τον γράφο G_c στον G θα είναι αδύνατη καθώς δεν υπάρχουν ακμές οι οποίες συνδέουν τους κόμβους u_c και v. Αυτό συμβαίνει γιατί δεν επιθυμούμε περισσότερες μηδενικές ακμές από μία. Πλέον αναζητούμε το ελάχιστο μονοπάτι από το s στο t_c με χρήση του αλγορίθμου Dijkstra. Για την εύρεση ελάχιστου μονοπατιού με Dijkstra στο γράφο $G^*(2V, 2E)$ η πολυπλοκότητα ανέρχεται στην τάξη του $O(2E+2Vlog2V) = O(E+VlogV)$.



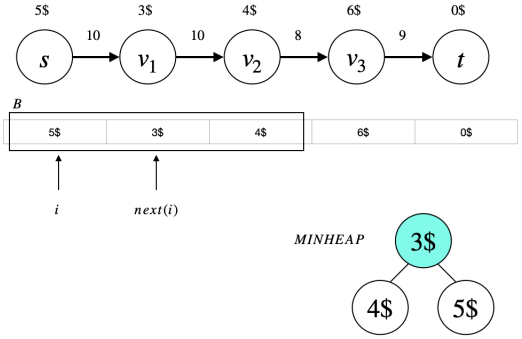
3.2 2ο Ερώτημα

Πλέον είναι σαφές πως η γενίκευση για k μηδενικές ακμές θα προκύψει δημιουργώντας πολλούς G_c γράφους πανομοιότυπους του G έστω $G_{c1}, G_{c2}, G_{c3}, \ldots, G_{ck}$ οι οποίοι συνδέονται μεταξύ τους σύμφωνα με τη σχέση ένωσης ακμών u_{ci} και $v_{c(i+1)} \forall i \leq k$. Ο νέος γράφος G^* θα είναι ίσος με την ένωση όλων των γράφων $G_{c1} \cup G_{c2} \cup \cdots \cup G_{ck}$. Για την αποφυγή δημιουργίας αυτού του γράφου στην περίπτωση όπου δεν απαιτείται, μπορούμε να κάνουμε μια BFS αναζήτηση στο γράφο για μονοπάτι με σύνολο ακμών $\leq k$ και έτσι απλά να μηδενίσουμε το κόστος των ακμών αυτών λύνοντας έτσι το πρόβλημα. Σχετικά με την πολυπλοκότητα του αλγορίθμου προκύπτει ίση με $O(kE + kVlogkV)$.

4 Ταξίδι σε Περίοδο Ενεργειακής Κρίσης

4.1 1ο Ερώτημα

Το παρόν πρόβλημα θα λυθεί ως εξής, αρχικά για κάθε κόμβο i θα πρέπει να βρίσκουμε τον κόμβο εκείνο j ο οποίος έχει μικρότερη τιμή καυσίμου και μπορούμε να φτάσουμε από τον i με $b(i,j) = b(i) + \cdots + b(k) + \cdots + b(j) \leq B$, τον οποίο θα ορίσουμε ως $next(i) = j$. Για να γίνει αυτό θα τοποθετήσουμε όλες τις τιμές καυσίμου σε κάθε κόμβο σειριακά όπως εμφανίζονται στο διαδοχικό γράφημα, και θα δημιουργήσουμε ένα πάραθυρο το οποίο κάθε φορά θα έχει μήκος B και θα περιέχει εκείνους τους κόμβους j για κάθε i όπως είδαμε παραπάνω. Για την υλοποίηση του παραθύρου θα κάνουμε χρήση σωρού ελαχίστου, με χρήση priority queue, όπου σε κάθε επόμενο κόμβο i θα διώχνουμε τον προηγούμενο κόμβο και θα τοποθετούμε τον επόμενο. Κάθε φορά με $O(1)$ για έναν κόμβο i μπορούμε να βρίσκουμε τον ελάχιστο σε αυτό το διάστημα. Με την υλοποίηση αυτή μπορούμε με $O(VlogV)$ να βρούμε όλους τους επόμενους $next(i) \forall i$. Έτσι κάθε φορά θα γεμίζουμε την απαιτούμενη ποσότητα καυσίμου για να φτάσουμε στον $next(i)$ και θα πληρώνουμε κόστος $b(i, next(i)) \cdot c(i)$. Επομένως το συνολικό κόστος θα είναι $C = \sum_{i=s}^{i=prev(t)} b(i, next(i))c(i)$ και η πολυπλοκότητα $O(nlogn)$.



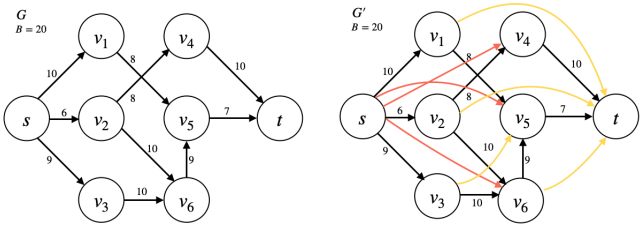
4.2 2ο Ερώτημα

Για την γενίκευση του προβλήματος θα δημιουργήσουμε έναν νέο γράφο G' ο οποίος θα περιέχει όλους τους κόμβους του G αλλά νέες ακμές οι οποίες θα περιέχουν την πληροφορία του κόστος επί την ποσότητα λίτρων για την διάνυση μίας διαδρομής από κάθε κόμβο σε κάθε άλλο σύμφωνα με τις κατευθύνσης των ακμών του γράφου G και λαμβάνοντας υπόψιν τον περιορισμό τον οποίο θέτει η συνολική χωρητικότητα του ντεπόζιτου B.

Ο τρόπος με τον οποίο θα συμβεί αυτό είναι ο εξής:

1. Για κάθε κόμβο στο γράφημα G, θα τρέξουμε τον Dijkstra με περιορισμό να σταματάμε να εξερευνούμε ένα μονοπάτι το οποίο είναι μεγαλύτερης ανάγκης σε καύσιμα από την χωρητικότητα B. Επομένως έτσι, για κάθε κόμβο j που συνδέεται με τον αρχικό μας κόμβο, έστω i, και έχει απόσταση $b(i,j) < B$ θα μπορούμε να βρούμε τι διαδρομή χρειάζεται να διανύσουμε με κόστος καυσίμου c_i.
2. Για τις παραπάνω διαδρομές που βρήκαμε θα ορίσουμε νέες ακμές με κόστος $c_i \cdot \sum_j b(i,j)$
3. Τοποθετούμε τις νέες ακμές στο γράφο G, αντικαθιστώντας τις παλιές με $c_i \cdot b(i, \text{adjacent})$ και τώρα είμαστε έτοιμοι να κάνουμε έναν τελικό Dijkstra για την εύρεση του ελάχιστου μονοπατιού στον γράφο από το s στο t.

Σχετικά με την πολυπλοκότητα της λύσης μας, αρχικά για την δημιουργία του νέου γράφου G', θα προκύψει $O(V \cdot (E + VlogV)) = O(VE + V^2logV)$, ενώ για τον τελικό Dijkstra, στην χειρότερη περίπτωση όπου έχουμε ένα πλήρη γράφο και όλα τα μονοπάτια έχουν απόσταση από το s στο t μικρότερη από B, προκύπτει $O(V^2 + VlogV)$ Άρα συνολικά η πολυπλοκότητα του αλγορίθμου μας είναι $O(VE + V^2logV)$.



5 Παιχνίδια Εξουσίας

5.1 1ο Ερώτημα

Πρόκειται για ένα πρόβλημα Birtate Matching και μάλιστα Full Matching για το σύνολο των ιπποτών όπου $I = |M| = |I| \leq |K|$ όπου I το σύνολο των ιπποτών και K το σύνολο των κάστρων. Θα ορίσουμε ένα γράφο $G = I \cup K \cup \{s, t\}$.

1. $\forall \text{ιππότη}_i \Rightarrow e = s \rightarrow \text{ιππότης}_i \in E : (f_e, b_e) = (f_e, c_i)$.
2. Για κάθε κάστρο το οποίο δέχεται να ηγηθεί από έναν ιππότη από τη λίστα K_j δημιούργουνται οι εξής ακμές $e = \text{ιππότης}_i \rightarrow \text{κάστρο}_j \in E : (f_e, b_e) = (\{0, 1\}, 1)$.
3. $\forall \text{κάστρο}_i \Rightarrow e = \text{κάστρο}_j \rightarrow t \in E : (f_e, b_e) = (\{0, 1\}, 1)$.

Πλέον μπορούμε να λύσουμε το πρόβλημα του matching των ιπποτών και των κάστρων με τον αλγόριθμο max flow με χρήση επαυξητικών μονοπατιών με μέγιστη χωρητικότητα των Edmonds-Karp στο γράφημα G. Αν το flow σε οποιαδήποτε ακμή των $\text{κάστρο}_j \rightarrow t$ προκύψει μηδενικό, αυτό σημαίνει πως κάποιο κάστρο δεν έχει ιππότη από τον οποίο διοικείται. Η πολυπλοκότητα είναι $O(m^2 logU)$ όπου U άνω φράγμα των χωρητικοτήτων του γράφου.

5.2 2ο Ερώτημα

Έστω ότι ο αλγόριθμος δεν βρίσκει το μέγιστο δυνατό flow, το οποίο σημαίνει ότι στο υπολειμματικό δίκτυο υπάρχει μονοπάτι από το s στο t, το οποίο βέβαια σημαίνει ότι μπορεί να γίνει περαιτέρω επαύξηση επομένως γνωρίζουμε ότι θα αλγόριθμος θα τερματίσει με βέλτιστη λύση.

5.3 3ο Ερώτημα

Το πρόβλημα αυτό μπορεί να λυθεί με χρήση του αλγορίθμου minimum vertex cover. Θα δημιουργήσουμε ένα γράφο G ο οποίος θα περιέχει όλους τους ιππότες ως κόμβους του γράφου και οι σχέσεις σύγκρουσης μεταξύ τους θα οριστούν ως οι ακμές που τους ενώνουν. Το minimum vertex cover του γράφου αυτού θα είναι το ελάχιστο σύνολο ιπποτών που θα πρέπει να εξοριστούν για να διακοπούν οι συγκρούσεις, ενώ το compliment του γράφου αυτού, δηλαδή το maximum independent set θα είναι το σύνολο των ιπποτών που θα μείνουν στο γράφο, δηλαδή δεν θα εξοριστούν από το βασιλιά. Από [1] γνωρίζουμε ότι υπάρχει προσεγγιστικός αλγόριθμος σε χρόνο $O(V+E)$ με χρήση λιστών γειτνίασης. Ο αλγόριθμος του βιβλίου έχει ως εξής:

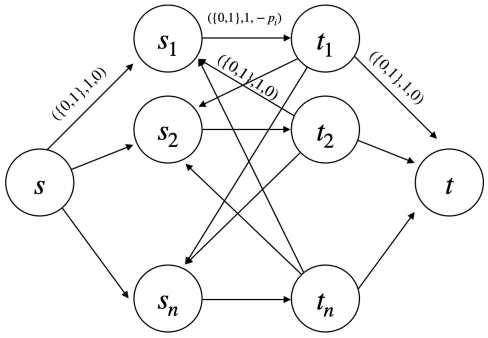
Algorithm 2 Approx Vertex Cover

```
1: C = 0
2: E' = G.E
3: while E'! = 0 do
4:     let (u,v) be an arbitrary edge of E'
5:     C = C ∪ u, v
6:     remove from E' edge (u,v) and every edge incident on either u or v
   return C
```

6 Ενοικίαση Αυτοκινήτων

Για την επίλυση του παρόντος προβλήματος θα αναγάγουμε το πρόβλημα στο πρόβλημα του **min cost flow**. Για την αναγωγή αυτή θα πρέπει να δημιουργήσουμε ένα γράφο $G(V, E)$ για το matching των δυνατών διαστημάτων μεταξύ προσφορών της επιχειρήσης.

1. Αρχικά ο γράφος αυτός θα περιέχει δύο διακριτά nodes πηγής **s** και καταβόθρας **t**.
2. $\forall$προσφορά$_i \Rightarrow e = s_i \rightarrow t_i \in E : (f_e, b_e, c_e) = (\{0, 1\}, 1, -p_i)$.
3. $\forall$προσφορά$_i$, προσφορά$_j : t_i < s_j \Rightarrow e = t_i \rightarrow s_j \in E : (f_e, b_e, c_e) = (\{0, 1\}, 1, 0)$.
4. $\forall$προσφορά$_i \Rightarrow e = s \rightarrow s_i \in E : (f_e, b_e, c_e) = (\{0, 1\}, 1, 0)$.
5. $\forall$προσφορά$_i \Rightarrow e = t_i \rightarrow t \in E : (f_e, b_e, c_e) = (\{0, 1\}, 1, 0)$.



Αναζητώ ελάχιστο κόστος γράφου με χρήση αλγορίθμου **min cost flow** και flow ίσο με κ από **s** στο **t**. Κάθε προσφορά για ένα αυτοκίνητο αναπαριστά μία μονάδα των κ μονάδων ροής. Με την εισαγωγή ενός αυτοκινητού, λόγω του capacity της ακμής το οποίο είναι ίσο με τη μονάδα, κατά τη διάρκεια ισχύος της δοσμένης προσφοράς δεν θα απασχοληθεί άλλη προσφορά. Με τη λήξη της προσφοράς αυτής θα υπάρχει η δυνατότητα για λήξη του συγκεκριμένου μονοπατιού ή επιλογή επόμενης προσφοράς.

Σχετικά με την πολυπλοκότητα του αλγορίθμου γνωρίζουμε από Erickson [2] ότι για το παρόν πρόβλημα μπορούμε να πετύχουμε πολυπλοκότητα της τάξης $O(E^2 log^2 V)$ με $V = 2n$ και $E = n$ άρα $O(n^2 log^2(2n))$.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.

[2] Jeff Erickson. "Minimum-Cost Flow". In: (). URL: https://jeffe.cs.illinois.edu/teaching/algorithms/notes/G-mincostflow.pdf.