ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι και Πολυπλοκότητα

2η Σειρά Γραπτών Ασκήσεων

Ανδρέας Χατζησάββας 03118701

Άσκηση 1: Πολύχρωμος Πεζόδρομος

Για αυτό το πρόβλημα θα χρησιμοποιήσουμε δυναμικό προγραμματισμό. Θα χρησιμοποιήσουμε τον πίνακα dp(i,j) όπου η τιμή της κάθε κατάστασης του είναι ο ελάχιστος αριθμός ημερών που χρειάζεται για να χρωματιστεί το διάστημα (i,j). Για κάθε διάστημα μήκους 1(δηλαδή i=j) αρχικοποιούμε με 0 και για κάθε διάστημα όπου i>j η τιμή είναι 0. Σε κάθε κατάσταση είτε θα βάψουμε το διάστημα με ένα χρώμα(όπως πρέπει να βαφτεί η πλάκα) και να προσθέσουμε 1 στο σύνολο των ημερών είτε θα χωρίσουμε το διάστημα σε 2 υποδιαστήματα και θα υπολογίσουμε ξεχωριστα. Συμπληρώνουμε τον πίνακα ως εξής:

dp(i,j) : ελάχιστος αριθμός ημερών για να βαφτεί το διάστημα (i,j)

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

Πολυπλοκότητα. Ο πίνακας είναι n^2 διαστάσεων. Τα i και j παίρνουν τιμές από [0,n-1] και κάθε κατάσταση χρειάζεται O(n) πράξεις για να υπολογιστεί αφού υπολογίζουμε καθε φορα που πρέπει αν χωριστεί το διάστημα. Άρα η πολυπλοκότητα ειναι O(n^3)

Άσκηση 2: String matching

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

1. **κ=1**

   Το πρόβλημα επιλύεται εφαρμόζοντας τον αλγόριθμο Dijkstra 2 φορές:
   Αρχικά, ξεκινώντας από τον κόμβο s, βρίσκουμε όλες τις αποστάσεις προς όλους τους κόμβους χρησιμοποιώντας dijkstra dist_1(n).
   Δημιουργούμε ένα δεύτερο γράφημα αντιστρέφοντας την φορά σε όλες τις ακμές κρατόντας σταθερά τα μήκη. Εφαρμόζουμε ξανά dijkstra και με αρχή τον κόμβο t βρίσκουμε όλες τις αποστάσεις προς όλους τους κόμβους dist_2(n).
   Για κάθε ακμή (u,v) του αρχικού γραφήματος υπολογίζουμε το άθροισμα dist_1(u) + dist_2(v) και παίρνουμε την ελάχιστη απόσταση.
   Ο αλγόριθμος είναι **ορθός** γιατι μια απο τις ακμές μηδενίζεται και αν υπολογίσουμε όλες τις αποστάσεις θα βρεθεί η βέλτιστη λύση
   Στον αλγόριθμο που περιγράφηκε ο αλγόριθμος Dijkstra έχει πολυπλοκότητα O(mlogn)(ή O(m+nlogn) με Fibonacci heap). Η δημιουργία του ανάποδου γραφήματος έχει πολυπλοκότητα O(n) και ο υπολογισμός των αποστάσεων O(m). Άρα η συνολική **πολυπλοκότητα** είναι O(mlogn) ή O(m+nlogn).

2. **κ>=1**

   Για την λύση του προβληματος δεν μπορούμε να επεκτείνουμε τον προηγούμενο αλγόριθμο.
   Θα αντιγράψουμε τον αρχικό γράφο κ φορές και κάθε αντίγραφο θεωρείται σαν ένα επίπεδο(κ+1 επίπεδα). Για κάθε ακμή του πρώτου γράφου (u,v,w) ενώνουμε με κατευθυνόμενη γραμμή μηδενικού μήκους από τον κόμβο u του πρώτου επιπέδου στον κόμβο v του επόμενου επιπέδου. Ενώνουμε επίσης όλους τους κόμβους t μεταξύ τους με μηδενικά μήκη. Το κάνουμε αυτό σε όλα τα επίπεδα δημιουργώντας έτσι ένα "υπεργράφημα".

Έπειτα εφαρμόζουμε Dijkstra απο τον κόμβο s του πρώτου επιπέδου. Η λύση είναι η απόσταση του συντομότερου μονοπατιού από τον s στο t του τελευταίου επιπέδου. Κάθε φορά που αλλάζει ένα επίπεδο ουσιαστικά μηδενίζεται μια ακμή. και αφού δεν μπορεί ο αλγόριθμος να επιστρέψει απο ένα επίπεδο i στο επίπεδο i-1 και όλα τα επίπεδα είναι κ+1 εγγυάται ότι θα γίνουν το πολύ k μηδενισμοί.

Άσκηση 4: Ταξίδι σε Περίοδο Ενεργειακής Κρίσης

1)

Αρχικά αν $\exists e \in E: b(e) > B$, δηλαδή αν υπάρχει ακμή που απαιτεί βενζίνη μεγαλύτερη απο αυτη που μπορούμε να αποθηκεύσουμε στο ντεπόζιτο δεν μπορούμε να φτάσουμε στην πόλη t. Αυτός ο έλεγχος γίνεται σε χρόνο O(n).

Αν θεωρήσουμε ότι είναι δυνατόν να μεταβούμε στην τελική πολη t. Εφαρμόζουμε τον παρακάτω αλγόριθμο στον πίνακα $c[1..n]$ (όπου χρησιμοποιούμε την σύντμηση $c[i] = c[u_i]$ και $u_1 := s$, $u_n = t$) και αποθηκεύουμε τα αποτελέσματα στον $d[1..n]$:

ClosestRightCheaper($c[1..n]$):
↪ $d[1..n-1] \leftarrow 1, 2, ..., n-1$, $s \leftarrow stack(NULL)$

▷ Για i από $n-1$ έως 1:
- $s.push(i+1)$
- Όσο $s \neq$ κενή:
  - Αν $c[u_i] \leq c[u_{s.top()}]$, τότε: $s.pop()$
  - Αλλιώς: *break*
- $d[i] \leftarrow s.top()$

return $d[1..n]$

Μετά το πέρας του αλγορίθμου ισχυέι:

$$d[i] = arg\{min_{c[j]<c[i],\, j>i}(j-i)\}$$

Εν συντομία στον πίνακα $d[i]$ βρίσκεται η κοντινότερη θέση j "δεξιότερα" της i ($j > i$ δηλαδή) για την οποία ισχύει $c[j] < c[i]$. Εκτός από τις θέσεις i οι οποίες "κυριαρχούν" όλων τον επόμενων, τότε $c[i] = i$, αφού έτσι αρχικοποιήσαμε τον πίνακα (για ευκολία αργότερα). Η πολυπλοκότητα του αλγορίθμου είναι $O(n)$ και αποδείχθηκε στην πρώτη σειρά ασκήσεων (όπως και η ορθότητα του). Ο αλγόριθμος του προβλήματος παρουσιάζεται παρακάτω. Πιο συγκεκριμένα σαν μια ρουτίνα (η απάντηση είναι *MinCostToTravel*(1, 0)).

MinCostToTravel(i,tank):
↪ *cost* ← 0
Αν μπορούμε να φθάσουμε στον προορισμό τελειώσαμε. Ο πίνακας *CostFromStart* μπορεί να υπολογιστεί σε χρόνο $\Theta(n)$ ως ένα κινούμενο άθροισμα.
- Αν *tank* ≥ *CostFromStart*[t] − *CostFromStart*[*ui*], τότε: Επέστρεψε 0

Αν δεν υπάρχουν φθηνότερα βενζινάδικα στην πορεία ή το επόμενο φθηνότερο είναι πιο μακριά από την αυτονομία μας, τότε γεμίζουμε το ντεπόζιτο και εκτελούμε τον ίδιο αλγόριθμο στην αμέσως επόμενη πόλη.
- Αν $d[i] = i$ ή $tank < CostFromStart[i] - CostFromStart[d[i]]$, τότε:
    - $cost \leftarrow cost + (B - tank) \cdot c[i]$
    - $tank \leftarrow B$
    - Επέστρεψε $cost + MinCostToTravel(i + 1, B - b(i, i + 1))$

Αλλιώς γεμίζουμε το ντεπόζιτο (αν χρειάζεται) ώστε να αρκεί ακριβώς για να φθάσουμε στην επόμενη φθηνότερη πόλη και συνεχίζουμε από αυτήν.
▷ Αν $i < n$:
    - Για j από i μέχρι $d[i]$:
        – Αν $tank > b(ui , ui+1)$, τότε: $tank \leftarrow fuel - b(uj , uj+1)$
        – Αλλιώς:
            ★ $buy \leftarrow b(uj , uj+1) - tank$
            ★ $tank \leftarrow 0$
            ★ $cost \leftarrow cost + c[i] \cdot buy$
    - Επέστρεψε $cost + MinCostToTravel(d[i], 0)$
↩ Επέστρεψε 0.

Η πολυπλοκότητα του αλγορίθμου είναι γραμμική: $\Theta(n)$.
Η ορθότητα του βασίζεται στα εξής επιχειρήματα:

- Αν μπορούμε να φθάσουμε στον προορισμό με την υπολειπόμενη βενζίνη τελειώσαμε.
- Αν βρισκόμαστε σε μια πόλη όπου η αμέσως επόμενη φθηνότερη είναι η uj τότε γεμίζουμε το ντεπόζιτο ακριβώς ώστε να φθάσουμε εκεί.
- Αν δεν επαρκεί το ντεπόζιτο για να φθάσουμε εκεί ή δεν έπονται φθηνότερες πόλεις Σελίδα 5 γεμίζουμε το ντεπόζιτο και επαναλαμβάνουμε την διαδικασία στην επόμενη πόλη. Θα αποδείξουμε ότι αυτός ο άπληστος αλγόριθμος παράγει την βέλτιστη λύση. Το άπληστο κριτήριο αποτελούν τα τρία παραπάνω βήματα. Είναι προφανές ότι αν μια λύση δεν ακολουθεί το πρώτο δεν μπορεί να είναι βέλτιστη, οπότε προχωράμε στο δεύτερο. Αν ισχύει η συνθήκη του δεύτερου βήματος, τότε μια λύση που δεν το ακολουθεί θα πρέπει να αγοράζει βενζίνη σε κάποια ενδιάμεση πόλη (πριν την uj και μετά την αρχική), ώστε να φθάσει εκει (αν δεν υπάρχουν ενδιάμεσες τότε και οι δύο λύσεις απαιτούνται να αγοράσουν όση χρειάζεται για να πάνε από την αρχική στην επόμενη). Αν σε αυτή την λύση "αφαιρέσουμε" κάποια ποσότητα βενζίνης που αγοράσθηκε στην ενδιάμεση πόλη και την αγοράσουμε στην αρχική, τότε έχουμε μικρότερο κόστος. Η ίδια τεχνική ανταλλαγής μπορεί να χρησιμοποιηθεί και και για την ορθότητα του τρίτου βήματος. Άρα ο αλγόριθμος μας παράγει την βέλτιστη λύση.

2)
Για την γενική περίπτωση όπου το G είναι γράφος θα εφαρμόσουμε δυναμικό προγραμματισμό. Θέτουμε C(u,b) το ελάχιστο κόστος για να φτάσουμε από την πόλη u στην t έχοντας αρχικά b λίτρα βενζίνης. Τότε

$$C(u, b) = \min_{v \in OutAdjList(u)} \{C(v, 0) + c(u)[b(v, u) - b],\ c(v) < c(u) \text{ και } b \leq b(v, u)$$
$$C(v, B - b(u, v)) + c(u)(B - b)\ , c(v) > c(u)$$

$0,\ u = t$
$\infty,\ \text{αλλιώς}\}$

Μπορούμε να αποδείξουμε ότι σε μια διαδρομή βέλτιστου κόστους όπου οι στάσεις σε βενζινάδικα γίνονται στις πόλεις i, i + 1, ..., k τότε πρέπει:
- Αν $c(i) \geq c(j)$ τότε "γεμίζουμε" μέχρι να φτάσουμε με ακριβώς 0 βενζίνη στην j.
- Αν ισχύει το αντίθετο, τότε γεμίζουμε το ντεπόζιτο

Απόδειξη: Αν "αγοράσουμε" παραπάνω στην πόλη i ενώ ισχύει $c(i) \geq c(j)$, τότε θα μπορούσαμε να αφαιρέσουμε από αυτό που αγοράσαμε "περαιτέρω" στην πρώτη πόλη και να αγοράσουμε το ίδιο ποσό βενζίνης στην επόμενη με ίσο ή μικρότερο κόστος. Παρόμοια απόδειξη και για τον δεύτερο ισχυρισμό.

Πολυπλοκότητα. Αρχικά θα αποδείξουμε ότι χρειαζόμαστε μόνο συγκεκριμένες τιμές του b. Έστω $z \neq s$ μια κορυφή όπου αγοράζουμε βενζίνη στη βέλτιστη λύση και u η προηγούμενη της. Τότε με βάση τον αλγόριθμο θα φτάσουμε στην z με βενζίνη στο σύνολο: $\{B - b(u, z) | u \in IncAdjList(z)\}\{0\}$, δηλαδή θα μπορούσε να πάρει το πολύ $|V| - 1 + 1 = |V|$ τιμές. Άρα εφόσον χρειαζόμαστε το $D(u, g)$ για κάθε πιθανό u ($|V|$ κορυφές) και πιθανό g (το πολύ $|V|$ τιμές για κάθε κορυφή), αυτό εξαρτάται από το πολύ $|V|$ γείτονες της u. Επομένως η πολυπλοκότητα είναι $O(|V|^3)$.

Άσκηση 5: Παιχνίδια Εξουσίας

(α) Η σχέση μεταξύ ιππότη και κάστρου είναι ένα bipartile graph. Για την επίλυση του προβλήματος θα φτιάξουμε ένα γράφημα ως εξής:

i. Θέτουμε σαν πηγή s τον βασιλιά.

ii. Συνδέουμε την πηγή με τους ιππότες με ροή χωρητικότητας c_i.

iii. Συνδέουμε τους ιππότες με όλα τα κάστρα με ακμές χωρητικότητας 1, όπου το κάστρο δεχεται να τεθεί υπό την εποπτεία του εκάστοτε ιππότη ή 0 αν δεν δέχονται.

iv. Συνδέουμε τα κάστρα με τερματικό κόμβο t χωρητικότητας 1

Με τον σχεδιασμό έχουμε τηρήσει τους περιορισμούς του προβλήματος. Σιγουρεύτηκα ότι κάθε ιππότης μπορεί να έχει υπό την επίβλεψη του μόνο c_i κάστρα αφού συνδέεται με c_i απο την πηγή. Κάθε κάστρο θα τεθεί υπό την επίβλεψη ιππότη που δέχεται γιατι συνδέονται με χωρητικότητα 1 με αυτούς που δέχονται και 0 με αυτούς που δεν δέχονται.
Επίσης συνδέοντας τα κάστρα και τον τερματικό κομβο t με χωρητικότητα 1 εγγυόμαστε ότι κάθε κάστρο θα επιβλεπεται απο μόνο έναν ιππότη.
Έπειτα αφού ο το γράφημα είναι bipartile μπορεί να εφαρμοστεί ο Hopcroft-Karp αλγόριθμος ώστε να βρεθεί ροή μέγιστου ταιριάσματος καρδιναλιότητας. Αν με το τέλος του αλγορίθμου βρεθεί κάστρο που να μην συνδέεται με ακμή με κάποιον ιππότη, τότε δεν υπάρχει ανάθεσει που να ικανοποιεί όλους τους περιορισμούς.

Ο αλγόριθμος είναι ορθός γιατι με τον αλγόριθμο Hopcroft-Karp θα βρεθεί το maximum cardinality που είναι αυτό που ψάχνουμε. Κάθε ακμή απο ιππότη σε κάστρο με ροή ίση με 1 συμβολίζει πως το κάστρο έχει τεθεί υπό την εποπτεία του ιππότη. Επίσης ο αλγόριθμος ελέγχει πως υπάρχει μια τέτοια ανάθεση αφού αν κάποια ροή από κάστρο προς τον τελικό

κόμβο είναι μηδενική σημαίνει πως δεν έχει τεθεί το κάστρο απο εποπτεία και άρα δεν υπάρχει λύση.

Η πολυπλοκότητα του Hopcroft-Karp είναι O(E * sqrt(V)).

(β) Εφόσον ο Βασιλιάς θέλει να θέσει κάθε κάστρο υπο την εποπτεία ενός ιππότη, αναμένουμε στο τέλος του αλγορίθμου ότι το max flow θα είναι ίσο με το σύνολο των κάστρων. Δηλαδή κάθε κάστρο εποπτεύεται από κάποιον ιππότη. Αν τα κάστρα είναι m και ισχύει ότι max_flow<m τότε διαπιστώνουμε με σιγουριά ότι δεν υπάρχει ανάθεση που να ικανοποιεί όλους τους περιορισμούς.

Άσκηση 6: Ενοικίαση Αυτοκινήτων

Για την λύση του προβλήματος θέλουμε να βρούμε το μέγιστο δυνατό συνολικό ποσό που θα εισπράξουμε ως αντίτιμο(pi). Το πρόβλημα ανάγεται στην εύρεση ροής ελάχιστου κόστους σε ένα κατάλληλα διαμορφωμένο γράφημα G(V,E) που θα κατασκευάσουμε με βάση τους περιορισμούς τους προβλήματος. Για την κατασκευή εκτελούμε τα εξής:

1. Θέτουμε αρχική κορυφή s, τελική t και ενδιάμεσες κορυφές ui, μια για κάθε προσφορά που λαμβάνουμε από τους πελάτες. Το δίκτυο έχει σύνολικά V = n + 2 κορυφές.
2. Η κορυφή s συνδέεται με τις ενδιάμεσες κορυφές j με ακμές χωρητικότητας 1.
3. Κάθε ενδιάμεση κορυφή j συνδέεται με την τελική κορυφή t με ακμή μοναδιαίας μοναδιαίας χωρητικότητας και κόστος pj.
4. Συνδέουμε κάθε ενδιάμεση κορυφή j με άλλη ενδιάμεση κορυφή i ώστε να ισχύει $tj \leq si$ και $j \neq i$.
5. Συνδέουμε επίσης την αρχική κορυφη s και την τελική κορυφή t με ακμή χωρητικότητας k, που αναπαριστά το μέγιστο αριθμό αυτοκινήτων που μπορούν να ενοικιαστούν ταυτόχρονα αφού είναι ο αριθμός των αυτοκινήτων που διαθέτουμε.

Έχουμε πλέον δημιουργήσει γράφο με μέγιστη ροη s-t τηρώντας τους περιορισμούς του προβλήματος. Τα μη επικαλυπτομενα χρονικα διαστηματα των ενοικιάσεων αναπαρίστανται με τις κατευθυνομενες ακμές j - i κόστους pi. Επίσης αν κάποια ενοικίαση j τερματίσει πριν ξεκινήσει η επόμενη i μπορεί να χρησιμοποιηθεί το ίδιο αμάξι με την προηγούμενη j. Με την εφαρμογή του αλγόριθμου κόστους θα επιλεγούν τα k φθηνότερα μονοπάτια του κατευθυνόμενου ακυκλικού γράφου που δημιουργήθηκε.
Για την δημιουργία του γράφου η πολυπλοκότητα είναι O(nlogn) για να αποφασιστεί ποιές προσφορές δεν έχουν επικαλυπτόμενα χρονικά διαστήματα και O(n+m) για την κατασκευή του γράφου. Άρα πολυπλοκότητα O(nlogn).
Για τον αλγόριθμο Min Cost Flow η πολυπλοκότητα θα είναι $O(n^2 * m * log(n * C))$, C είναι η τιμή του μικρότερου κόστους μονοπατιού στο γράφημα.