ARISTOTLE UNIVERSITY OF THESSALONIKI

Aristotle University of Thessaloniki
Department of Electrical and Computer Engineering

Καράτης Δημήτριος 10775
Βαπόρης Δημήτριος 10625
(Team 10)

Ανάλυση και σχεδιασμός Αλγορίθμων

6ο εξάμηνο

Contents

1 Πρόβλημα 1

1.1 Ερώτημα 1

Για τον σχεδιασμό ενός αποδοτικού αλγορίθμου ο οποίος θα βρίσκει εάν υπάρχει εφικτό δρομολόγιο από την πόλη s στην πόλη t, η προσέγγισή μας μετέρχεται μια παραλλαγή του ήδη γνωστού από το μάθημα αλγορίθμου BFS ($Breadth-FirstSearch$). Πιο συγκεκριμένα, για να καταστήσουμε την λύση μας όσο το δυνατόν πιο απλοϊκή και εναργής γίνεται αποφασίσαμε πριν την χρήση του BFS να διαγράψουμε όλες τις οδικές αρτηρίες e, ή αλλιώς τις ακμές του δοθέντος γράφου $G=(V,E)$, των οποίων το μήκος l_e δεν μας επιτρέπει να τις διασχίσουμε. Δηλαδή $l_e > L$, και άρα δεν διαθέτουμε την κατάλληλη χωρητικότητα καυσίμων με γεμάτο ρεζερβουάρ L για να ταξιδέψουμε κατά μήκος του εκάστοτε l_e. Επομένως, σε παρόμοιες περιπτώσεις διαγράφουμε από τον γράφο την ακμή e και άρα αφού εξετάσουμε με τον τρόπο αυτόν όλες τις ακμές e του γράφου, (οδικές αρτηρίες) στο τέλος θα έχουμε έναν γράφο $G=(V,E')$. Ο νέος γράφος θα έχει τους ίδιους κόμβους (ίδιες πόλεις) με τον αρχικό, αλλά θα διαφέρουν στις συνδέσεις/διαδρομές/ακμές τους, και άρα κι ο πίνακας Adj θα είναι επίσης αλλαγμένος. Στον νέο γράφο θα υπάρχουν πλέον μονάχα οι οδικές αρτηρίες μεταξύ πόλεων στις οποίες μπορούμε να ταξιδέψουμε δεδομένης της χωρητικότητας καυσίμων ενός γεμάτου ρεζερβουάρ L.

Επόμενό μας βήμα είναι να σχεδιάσουμε έναν αλγόριθμο ο οποίος θα επισκέπτεται όλους τους κόμβους και όλες τις ακμές. Για τον σκοπό αυτόν επιλέξαμε να χρησιμοποιήσουμε τον ήδη γνωστό από την βιβλιογραφία αλγόριθμο του BFS ($Breadth-FirstSearch$). Έτσι, θα υπάρχει διαθέσιμο δρομολόγιο από την πολη s στην πόλη t, εάν μετά το πέρας των επαναλήψεων του αλγορίθμου BFS ο κομβος/πόλη t δεν είναι άσπρος.

1.2 Ανάλυση ψευδοκώδικα και χρονική πολυπλοκότητα

1. Αρχικά, ο αλγόριθμος εξετάζει κάθε ακμή (δρόμο) στο γράφο. Αν το μήκος της ακμής είναι μεγαλύτερο από τη μέγιστη απόσταση που μπορούμε να καλύψουμε με ένα γεμάτο ρεζερβουάρ (συμβολίζεται με το L), τότε αφαιρούμε αυτή την ακμή από το σύνολο των ακμών του γράφου. **Η ενέργεια αύτη απαιτέι συνολικά O(|E|) χρόνο εκτέλεσης, αφού η επανάληψη εξετάζει κάθε ακμή μια φορά και η διαδικασία εξέτασης γίνεται σε σταθερό χρόνο.**
2. Στη συνέχεια, αρχικοποιούνται οι κόμβοι του γράφου. Κάθε κόμβος έχει ένα χρώμα (που αρχικά είναι λευκό), μια απόσταση (που αρχικά ορίζεται ως άπειρο) και έναν πατέρα (που αρχικά ορίζεται ως κενός). **Η ενέργεια αυτή απαιτεί συνολικά O(|V|) χρόνο εκτέλεσης εφόσον η αρχικοποίηση κάθε κόμβου απαιτεί σταθερό χρόνο για κάθε κόμβο και πρέπει να αρχικοποιηθούν όλοι.**
3. Ο κόμβος εκκίνησης s ορίζεται ως γκρι και ο πατέρας του ως κενός. Αυτός ο κόμβος προστίθεται στην ουρά Q. **Οι ενέργειες αυτές απαιτούν σταθερό χρόνο η κάθε μία.**
4. Επαναληπτικά, μέχρι η ουρά Q να μην είναι άδεια, εξετάζουμε κάθε κόμβο u που αφαιρείται από την ουρά Q. **Η ενέρεγεια αυτή απαιτεί χρόνο O(|V|).**
5. Για κάθε γείτονα v του κόμβου u (δηλαδή για κάθε κόμβο που συνδέεται με τον κόμβο u με μια ακμή), αν ο γείτονας v είναι λευκός, τότε τον επισημαίνουμε ως γκρι, και αυξάνουμε κατά 1 την απόστασή του από τον κόμβο εκκίνησης ενώ παράλληλα τον ορίζουμε ως γονέα του κόμβου u. Στη συνέχεια, προσθέτουμε τον γείτονα στην ουρά Q. **Η επανάληψη αυτή απαιτεί χρόνο O(|E|). (Κάθε ακμή εξετάζεται το πολύ 2 φορές σε μη προσανατολισμένους γράφους).**
6. Όταν εξετάζεται ένας κόμβος u, ο ίδιος χρωματίζεται μαύρος, υποδεικνύοντας ότι έχει ολοκληρωθεί η εξέτασή του και ότι έχουν εξεταστεί όλοι του οι γείτονές. **Αυτό γίνεται σε σταθερό χρόνο.**
7. Τέλος, ελέγχεται αν ο κόμβος-στόχος t έχει παραμείνει λευκός. Αν ναι, τότε δεν υπάρχει εφικτό μονοπάτι από τον κόμβο εκκίνησης στον κόμβο-στόχο. Αν όχι, τότε υπάρχει. **Οι ενέργειες**

αυτές εκτελούνται επίσης σε σταθερό χρόνο.
Συνολικά λοιπόν, ο χρόνος εκτέλεσης του αλγορίθμου είναι $\mathbf{O(|V| + |E|)}$, *ίδιος με τον χρόνο εκτέλεσης του* BFS.

1.3 Ερώτημα 2

Παρατηρούμε αρχικά ότι το ζητούμενο του συγκεκριμένου προβλήματος αφορά την ελάχιστη χωρητικότητα του ρεζερβουάρ ώστε να είναι δυνατή η μετάβαση από την πόλη s στην πόλη t. Αυτό σημαίνει ότι δε μας ενδιαφέρει η συνολική απόσταση που διανύουμε ώστε να μεταβούμε από την αρχική πόλη στην τελική, αλλά η μέγιστη απόσταση των επιμέρους μεταβάσεων (από τη μία πόλη στην άλλη) που πραγματοποιούμε ώστε να εκπληρώσουμε τον στόχο μας. Το πρόβλημα είναι, λοιπόν, πρόβλημα ελαχιστοποίησης της μέγιστης απόστασης, πράγμα που μας θυμίζει αμέσως τη λογική των «άπληστων» αλγορίθμων. Μάλιστα, αφού το πρόβλημα ήδη έχει μοντελοποιηθεί με γράφους, γίνεται εμφανής η πιθανή εφαρμογή των ελάχιστων επικαλυπτόντων δένδρων ($MSTs$). Βασική ιδιότητα του MST ενός συνδεδεμένου γράφου είναι ότι αποτελεί τον υπογράφο που συνδέει κάθε κορυφή του αρχικού με οποιαδήποτε άλλη, ελαχιστοποιώντας το συνολικό βάρος των ακμών που επιλέγονται. Αρχικά, η μορφή αυτή φαίνεται να μην ταιριάζει με το πρόβλημά μας, αφού αναφέρεται στο συνολικό βάρος των ακμών. Ωστόσο, αναλογιζόμενοι τους αλγορίθμους παραγωγής $MSTs$ ($Kruskal's, Prim's$), συνειδητοποιούμε ότι κατά την κατασκευή του MST επιλέγεται κάθε φορά μία κατάλληλη (ασφαλή) ακμή με το ελάχιστο βάρος. Κατάλληλη είναι μία ακμή όταν συνδέει δύο ασύνδετα τμήματα του υπογράφου που αναπτύσσουμε μεταξύ τους. Δηλαδή, κάθε ακμή που επιλέγεται για ένα MST αποτελεί: α) απαραίτητο τμήμα του MST (σε κάθε MST του γράφου θα υπάρχει είτε η συγκεκριμένη ακμή είτε κάποια με ίδιο βάρος, πράγμα που δεν επηρεάζει το πρόβλημά μας) και β) ακμή που συνδέει δύο μέχρι τότε ασύνδετα τμήματα του γράφου. Αναδεικνύεται τώρα η καταλληλόλητα των $MSTs$ και των προαναφερθέντων αλγορίθμων που τα κατασκευάζουν για το πρόβλημά μας. Μάλιστα, εφόσον μας ενδιαφέρει συγκεκριμένα το ελάχιστο ρεζερβουάρ για να είναι εφικτό το ταξίδι από την πόλη s στην πόλη t, και όχι από οποιαδήποτε πόλη σε οποιαδήποτε άλλη, μπορούμε να κάνουμε τις εξής βελτιώσεις: 1) Χρησιμοποιούμε ως βάση τον αλγόριθμο του $Prim$ και θέτουμε ως κορυφή αφετηρίας την κορυφή της πόλης s από την οποία και ξεκινούμε. 2) Τρέχουμε τη βασική επανάληψη του αλγορίθμου όσο το $t.key$ είναι άπειρο (όσο, δηλαδή, δεν έχει προστεθεί η πόλη t στο MST). 3) Κρατούμε το μέγιστο βάρος των ακμών του MST σε μία μεταβλητή, έστω max_weight, και για κάθε ακμή που προσθέτουμε το ανανεώνουμε ανάλογα (αν το βάρος της ακμής που προσθέτουμε είναι μεγαλύτερο από το μέχρι τώρα μέγιστο, τότε το νέο μέγιστο γίνεται ίσο με το συγκεκριμένο βάρος). Η μεταβλητή αρχικοποιείται με τιμή μηδέν (μεταφράζεται στο ότι οι κορυφές συνδέονται άμεσα, ότι δηλαδή δεν υπάρχει βάρος στη διαδρομή που να ενώνει τις πόλεις). 4) Επιστρέφουμε το max_weight στο τέλος του αλγορίθμου. Να σημειωθεί ότι ο αλγόριθμος λειτουργεί μόνο εφόσον γνωρίζουμε ότι οι πόλεις s και t συνδέονται (αν αυτό δε συμβαίνει, δημιουργείται ένας ατέρμων βρόχος, καθώς το $t.key$ παραμένει για πάντα άπειρο). Για να είμαστε βέβαιοι ότι αυτό ισχύει, μπορούμε να αξιοποιήσουμε τον αλγόριθμο $Dijkstra$ με αφετηρία την πόλη s. Εάν στο τέλος του αλγορίθμου αυτού η απόσταση μεταξύ s και t είναι άπειρη, τότε οι πόλεις δε συνδέονται και άρα το πρόβλημα δε λύνεται. Αν η απόσταση δεν είναι άπειρη, ο συγκεκριμένος αλγόριθμος υπολογίζει την ελάχιστη χωρητικότητα καυσίμων, όπως ζητείται.

1.4 Ανάλυση ψευδοκώδικα και χρονική πολυπλοκότητα

1. Αρχικά εκτελείται ο αλγόριθμος $Dijkstra$ στον γράφο G με αφετηρία την κορυφή s. Ο χρόνος εκτέλεσης του αλγορίθμου αυτού, όπως είναι γνωστό από το μάθημα, εξαρτάται από την υλοποίηση που επιλέγεται για τον γράφο. Χρησιμοποιώντας $binaryheap$, ο χρόνος εκτέλεσης είναι $\mathbf{O((|V| + |E|)log|V|)}$.

2. Ανάλογα με το αποτέλεσμα της εκτέλεσης του $Dijkstra$ (αν το $t.d$ δεν είναι άπειρο, δηλαδή αν υπάρχει διαδρομή που να συνδέει την πόλη s και την πόλη t), ο αλγόριθμος συνεχίζεται με τη διαδικασία $MODIFIED-MST-PRIM$. Η διαδικασία αυτή απαιτεί πρώτα την αρχικοποίηση των μεταβλητών $u.key$ και $u.\pi$ μέσω μίας επανάληψης με χρόνο εκτέλεσης $\mathbf{O(|V|)}$. Ακολουθούν αρχικοποιήσεις των μεταβλητών $s.key$ και max_weight, με χρόνο εκτέλεσης $\mathbf{O(1)}$ και της δομής Q.
3. Ακολουθεί η βασική επανάληψη της διαδικασίας $Prim$, με διαφορά ότι η επανάληψη λήγει όταν το $t.key$ δεν είναι πια άπειρο, όταν δηλαδή υπάρξει για πρώτη φορά μονοπάτι από την κορυφή s στην κορυφή t. Η βελτίωση αυτή στις περισσότερες περιπτώσεις ελαχιστοποιεί τον χρόνο εκτέλεσης του αλγορίθμου, ενώ στη χειρότερη περίπτωση ο χρόνος εκτέλεσης είναι ίδιος.
4. Η υποδιαδικασία $EXTRACT-MIN$ (κοινή για τους $MODIFIED-MST-PRIM$ και $MST-PRIM$) επιστρέφει σε κάθε εκτέλεση της επανάληψης την κορυφή με το ελάχιστο key και ο χρόνος εκτέλεσής της εξαρτάται από την υλοποίηση που επιλέγεται για τη δομή Q. Οι υπόλοιπες εντολές του $MODIFIED-MST-PRIM$ είναι ίδιες εκτός από έναν έλεγχο για το max_weight και μία πιθανή ενημέρωση, ανάλογα με το αποτέλεσμα του ελέγχου. Σε κάθε περίπτωση, οι κινήσεις αυτές (ο έλεγχος $l(u,v) > max\_weight$ και η ενημέρωση του max_weight) έχουν χρόνο εκτέλεσης $\mathbf{O(1)}$ και δεν επηρεάζουν τον συνολικό χρόνο εκτέλεσης της διαδικασίας.
5. Επομένως, όπως είναι εμφανές, οι χρόνοι εκτέλεσης των $MODIFIED-MST-PRIM$ και $MST-PRIM$ είναι ίδιοι. Ο χρόνος εκτέλεσης του $MST-PRIM$, με επιλογή χρήσης $binaryheap$, είναι $\mathbf{O((|V|+|E|)logV)}$.
6. Τελικά, ο συνολικός χρόνος εκτέλεσης του αλγορίθμου για το συγκεκριμένο πρόβλημα είναι $\mathbf{O((|V|+|E|)logV)}$ και άρα όντως υπάρχει αλγόριθμος που υλοποιεί το *Πρόβήμα 1: Ερώτημα 2* σε χρόνο $\mathbf{O((|V|+|E|)logV)}$.

2 Πρόβλημα 2

2.1 Ερώτημα 1

Για να βρούμε μία αποδοτική λύση για το δοσμένο πρόβλημα, σκεφτόμαστε αρχικά τα προβλήματα που προκύπτουν από τη συνήθη λύση, δηλαδή την τήρηση σειράς προτεραιότητας με βάση τον χρόνο προσέλευσης του κάθε πολίτη. Συγκεκριμένα, ο αλγόριθμος αυτός, αν και είναι ο πιο δίκαιος, δε θέτει ως κριτήριο τον συνολικό χρόνο αναμονής μεταξύ όλων των πολιτών, αλλά τον εκτιμώμενο χρόνο αναμονής του κάθε πολίτη ξεχωριστά. Επομένως, ο αλγόριθμος που θα επιλεχθεί από εμάς οφείλει να αποφύγει την προτεραιότητα με βάση τη σειρά άφιξης. Έστω ότι όλοι οι n πολίτες που θα εξυπηρετηθούν σε μία συγκεκριμένη ημέρα βρίσκονται στην ουρά από την αρχή της ημέρας. Για να ελαχιστοποιήσουμε τον συνολικό χρόνο αναμονής, θα ήταν λογικό να εξυπηρετηθούν πρώτα οι πολίτες των οποίων οι χρόνοι εξυπηρέτησης είναι μικρότεροι. Για παράδειγμα, αν υπάρχουν 20 πολίτες στην ουρά και επιλεχθεί ως πρώτος εξυπηρετούμενος κάποιος με χρόνο εξυπηρέτησης 2 λεπτών, ο συνολικός χρόνος αναμονής των 19 υπολοίπων για τον πρώτο πολίτη θα είναι ίσος με 38 λεπτά, ενώ αν επιλεχθεί πρώτος κάποιος με χρόνο εξυπηρέτησης 20 λεπτών, ο συνολικός χρόνος αναμονής θα ανερχόταν σε 380 λεπτά. Έτσι, σκεπτόμενοι και με τη λογική των «άπληστων» αλγορίθμων, εικάζουμε πως η στιγμιαία καλύτερη επιλογή κάθε φορά ξεχωριστά οδηγεί και στην καλύτερη δυνατή συνολική επίλυση του προβλήματος. Δηλαδή, θεωρούμε ότι η πιο αποδοτική λύση είναι να εξυπηρετηθούν οι πολίτες με σειρά προτεραιότητας που έχει ως κριτήριο τον χρόνο εξυπηρέτησης του καθενός, σε αύξουσα σειρά. Παρακάτω προχωρούμε στην απόδειξη του παραπάνω ισχυρισμού.

Η λύση που προτείνουμε οδηγεί σε συνολικό χρόνο αναμονής (για n πολίτες συνολικά):
$T_1 = t_1(n-1) + t_2(n-2) + t_3(n-3) + ... + t_i(n-i) + t_{i+1}(n-i-1) + ...$
Όπου $t_1, t_2, t_3...$ οι χρόνοι για την εξυπηρέτηση του κάθε πολίτη σε άυξουσα σειρά.

Έστω τώρα ότι υπάρχει καλύτερη λύση από την προτεινόμενη. Αυτή θα διαφέρει από την προτεινόμενη σε τουλάχιστον ένα σημείο και θα ισχύει: $T_2 < T_1$
Χωρίς βλάβη της γενικότητας, έστω ότι διαφέρει μόνο στο *i-οστό* και *i+1-οστό στοιχείο* (απλή αντιμετάθεση δύο πολιτών). Τότε:
$T_2 = t_1(n-1) + t_2(n-2) + t_3(n-3) + ... + t_{i+1}(n-i) + t_i(n-i-1) + ...$ και άρα:
$T_2 - T_1 = t_{i+1}(n-i) + t_i(n-i-1) - t_i(n-i) - t_{i+1}(n-i-1)$
εξ ορισμού, αφού οι χρόνοι βρίσκονται σε άυξουσα σειρά.
Αυτό σημαίνει όμως ότι $T_2 - T_1 = t_{i+1} - t_i \geq 0$ δηλαδή $T_2 \geq T_1$. Οδηγηθήκαμε σε άτοπο, άρα δεν υπάρχει καλύτερη λύση από την προτεινόμενη.
Προφανώς ο παραπάνω ισχυρισμός ισχύει και για περισσότερες από μία αντιμεταθέσεις, σε αποστάσεις ακόμη και μεγαλύτερες από 1 μεταξύ των δύο πολιτών που ¨αλλάζουν σειρά¨. Σε περίπτωση που δεν είναι προφανείς αυτές οι αντιμεταθέσεις, το πρόβλημα μπορεί να αναχθεί στη σύγκριση της δοσμένης σειράς των πολιτών με το αποτέλεσμα αντιμεταθέσεων πολιτών της λύσης που προτείνουμε, το οποίο, όπως αποδείξαμε, είναι σε κάθε περίπτωση εξίσου γρήγορο ή αργότερο από την προτεινόμενη λύση.
Ο προτεινόμενος αλγόριθμος έχει υλοποιηθεί σε μορφή ψευδοκώδικα στο αντίστοιχο παράρτημα. Αρχικά, οι πολίτες τοποθετούνται σε έναν πίνακα Α, ο οποίος ταξινομείται με βάση τον χρόνο εξυπηρέτησης του καθενός. Η ακριβής περιγραφή του αλγορίθμου εξαρτάται από την υλοποίηση που θα επιλεχθεί, καθώς επηρεάζεται ελαφρώς η ταξινόμηση (π.χ. θεωρούμε τους πολίτες αντικείμενα μίας κλάσης). Για τη διαδικασία της ταξινόμησης μπορεί να αξιοποιηθεί η διαδικασία $mergesort()$, στην οποία έχει γίνει αναφορά στο μάθημα. Από τη διαδικασία της ταξινόμησης, προκύπτει ο πίνακας Α, στον οποίο οι πολίτες βρίσκονται πλέον στη σειρά, με αυξανόμενο χρόνο εξυπηρέτησης. Από αυτόν τον πίνακα, οι πολίτες ενσωματώνονται σε μία ουρά Q, η οποία αποτελεί και την ουρά βέλτιστης εξυπηρέτησης ώστε να ελαχιστοποιείται ο συνολικός χρόνος αναμονής των πολιτών.

3 Πρόβλημα 3

3.1 Ερώτημα 1

Για τον σχεδιασμό ενός αποδοτικού αλγορίθμου εύρεσης του ελάχιστου υπολογιστικού κόστους για το πρόβλημα τομής της συμβολοσειράς σε $m+1$ τμήματα, σκεφτόμαστε αρχικά τη βασική ερώτηση του δυναμικού προγραμματισμού: σε ποια υποπροβλήματα μπορούμε να χωρίσουμε το πρόβλημά μας... Η λογική απάντηση είναι να βρούμε το ελάχιστο κόστος τομής της συμβολοσειράς σε λιγότερα από $m+1$ τμήματα, δηλαδή το κόστος K[u], με $0 \leq u \leq m$. Τότε, το ζητούμενο κόστος θα ήταν το K[u] για $u = m$. Για να το πετύχουμε αυτό, ακολουθούμε στρατηγική παρόμοια με αυτήν που παρουσιάστηκε στο μάθημα για το πρόβλημα πολλαπλασιασμού πινάκων, στο οποίο αναζητούσαμε το ελάχιστο υπολογιστικό κόστος για τον διαδοχικό πολλαπλασιασμό πινάκων. Ορίζουμε, λοιπόν, έναν μονοδιάστατο πίνακα P (διάστασης m) στον οποίον αποθηκεύουμε τις δεδομένες θέσεις των τομών. Στον πίνακα αυτόν, προσθέτουμε και τις ακραίες τιμές *0* και n, για λόγο που θα γίνει φανερός αργότερα. Επομένως, ο πίνακας P έχει μέγεθος $m+2$ (ξεκινώντας από στοιχείο με $index$ 0). Για ευκολία θεωρούμε ότι το m (ο αριθμός των σημείων τομών) αυξάνεται κατά 2 και ότι ο πίνακας P έχει τώρα m στοιχεία και μέγεθος επίσης m. (Στην ουσία για χάρην απλότητας θεωρούμε ότι το m είναι ο αριθμός των τομών που μας δίνεται από την εκφώνηση του προβλήματος + 2 που προσθέσαμε αυθαίρετα εμείς). Το κύριο εργαλείο του προτεινόμενου αλγορίθμου είναι ο δισδιάστατος πίνακας $C_{i,j}$ μεγέθους m x m, του οποίου τα στοιχεία εκφράζουν το κόστος της τομής μεταξύ των θέσεων τομών P[i] και P[j] του πίνακα P. Προφανώς, τα στοιχεία $C_{i,i}$ είναι *0*, διότι δεν υπάρχει δυνατή τομή ανάμεσα στον ίδιο χαρακτήρα μιας συμβολοσειράς. Τα στοιχεία $C_{i,i+1}$ είναι επίσης *0*, διότι δεν υπάρχει δυνατή τομή ανάμεσα στις θέσεις τομών i και $i+1$, αφού οι θέσεις αυτές είναι διαδοχικές. Να σημειωθεί ότι τα στοιχεία i και j του πίνακα αναφέρονται στις θέσεις i και j του πίνακα P και όχι στις θέσεις της συμβολοσειράς. Τα στοιχεία i και j του πίνακα P αναφέρονται στις αντίστοιχες θέσεις της γραμματοσειράς στις οποίες γίνονται οι τομές. Με τη βοήθεια των παραπάνω γνωστών τιμών, υπολογίζουμε μέσω μίας επανάληψης και τις υπόλοιπες τιμές του πίνακα C που μας ενδιαφέρουν. Στην επανάληψη, φροντίζουμε κάθε φορά να εξετάζουμε κάθε πιθανό ενδεχόμενο και να τοποθετούμε το ελάχιστο αποτέλεσμα στη θέση $C_{i,j}$. Τέλος, το ελάχιστο κόστος που ψάχνουμε βρίσκεται στο στοιχείο $C_{0,m-1}$, το στοιχείο δηλαδή που σύμφωνα με τον συμβολισμό μας εκφράζει το κόστος τομής από το πρώτο μέχρι το τελευταίο στοιχείο του πίνακα P, δηλαδή από τη θέση *0* έως τη θέση n της συμβολοσειράς, άρα και το ελάχιστο κόστος της τομής της συμβολοσειράς που ψάχναμε.
Για καλύτερη κατανόηση του τρόπου με τον οποίο λειτουργεί ο αλγοριθμός μας, έχουμε συμπεριλάβει ένα παράδειγμα εκτέλεσής του σε αντίστοιχο παράρτημα, μαζί και με τον ψευδοκώδικα που τον υλοποιεί.

3.2 Ανάλυση ψευδοκώδικα και χρονική πολυπλοκότητα

1. Αρχικά, προσθέτουμε την θέση κοπής *0* στην αρχή του πίνακα P, που περιλαμβάνει τις θέσεις των απαιτούμενων τομών, καθώς επίσης και τη θέση κοπής n στο τέλος αυτού. Με τον τρόπο αυτόν εξασφαλίζουμε ότι η λίστα των θέσεων κοπής P περιλαμβάνει και τις δύο άκρες του συμβολοσειράς, γεγονός που θα μας χρειαστεί ώστε να εφαρμόσουμε τεχνικές δυναμικού προγραμματισμού. **Οι ενέργειες αυτές γίνονται σε σταθερό χρόνο.**
2. Έπειτα θα πρέπει να ενημερώσουμε τη μεταβλητή m ώστε να λάβει υπόψη της τις δύο θέσεις κοπής που προσθέσαμε προηγουμένως, **γεγονός που επιτυγχάνεται επίσης σε σταθερό χρόνο.**
3. Σε επόμενο βήμα, αρχικοποιούμε τη διαγώνιο του πίνακα με τα κόστη C καθώς και τα στοιχεία που είναι δίπλα σε αυτή με *0*. Αυτό αντιστοιχεί στη βασική περίπτωση όπου το μήκος του τμήματος είναι μικρότερο ή ίσο με *2*, οπότε δεν μπορεί να υπάρξει τομή μεταξύ των στοιχείων/συμβόλων i, i και $i, i+1$ του πίνακα P και άρα το κόστος τομής θα είναι *0* σε εκείνες τις περιπτώσεις. **Η αρχικοποίηση αυτή γίνεται σε χρόνο $\mathbf{O(m^2)}$.**

4. Έπειτα, με τη χρήση των δυο επαναλήψεων for καταφέρνουμε να κινηθούμε διαγωνίως του πίνακα κόστους C, υπολογίζοντας κάθε φορά το κόστος C[i,j], με $j = i + s$, όπου s το μέγεθος της κάθε συμβολοσειράς που εξετάζουμε σε κάθε υποπρόβλημα. **Για τον υπολογισμό του C[i,j] χρειαζόμαστε χρόνο $\mathbf{O(m^3)}$**, καθώς πρέπει να υπολογίσουμε το κόστος κοπής της κάθε συμβολοσειράς με τομές από την θέση i έως τη θέση j. Αυτό απαιτεί m^3 επαναλήψεις, καθώς πρέπει να εξεταστούν όλες οι δυνατές θέσεις κοπής k για κάθε ζευγάρι i, j.
Συνολικά λοιπόν, ο χρόνος εκτέλεσης του αλγορίθμου είναι $\mathbf{O(m^3)}$, *ίδιος με τον χρόνο εκτέλεσης του αλγορίθμου πολλαπλασιασμού πινάκων από το μάθημα.*

A Παράρτημα

A.1 Ψευδοκώδικας για το Πρόβλημα 1: Ερώτημα 1

Algorithm 1 Αλγόριθμος για το Πρόβλημα 1: Ερώτημα 1

```
for each edge e ∈ G.E do
    if l_e > L then
        G.E = G.E − {e}
    end if
end for
BFS(G,s)
if t.color == WHITE then return false
else return true
end if

procedure BFS(G,s)
    for each vertex u ∈ G.V − {s} do
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    end for
    s.color = GREY
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅ do
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u] do
            if v.color == WHITE then
                v.color = GRAY
                v.d = v.d + 1
                v.π = u
                ENQUEUE(Q, v)
            end if
        end for
        u.color = BLACK
    end while
end procedure
```

A.2 Ψευδοκώδικας για το Πρόβλημα 1: Ερώτημα 2

Algorithm 2 Αλγόριθμος για το Πρόβλημα 1: Ερώτημα 2

```
DIJKSTRA(G,l,s)
if t.d == ∞ then return ∞
else MODIFIED-MST-PRIM(G,l,s)
end if

procedure MODIFIED-MST-PRIM(G,l,s)
    for each u ∈ G.V do
        u.key = ∞
        u.π = NIL
    end for
    s.key = 0
    max_weight = 0
    Q = G.V
    while t.key == ∞ do
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u] do
            if v ∈ Q and l(u,v) < v.key then
                v.π = u
                s.key = l(u,v)
                if l(u,v) > max_weight then
                    max_weight = l(u,v)
                end if
            end if
        end for
    end while
return max_weight
end procedure
```

A.3 Ψευδοκώδικας για το Πρόβλημα 2: Ερώτημα 1

Algorithm 3 Αλγόριθμος για το Πρόβλημα 2: Ερώτημα 1

```
A = ∅
for each citizen c do
    add c to A
end for
sort citizens based on time
Q = ∅
for each citizen c ∈ A do
    ENQUEUE(Q,c)
end for
```

A.4 Ψευδοκώδικας για το Πρόβλημα 3: Ερώτημα 1

Algorithm 4 Αλγόριθμος για το Πρόβλημα 3: Ερώτημα 1

```
add cut position 0 at the start of P and
cut position n at the end of P
m = m + 2
for i = 0 to m do
    C_{i,i} = 0 and C_{i,i+1} = 0
end for
for s = 2 to m do
    for i = 0 to m − s do
        j = i + s
        C_{i,j} = min_{i≤k≤j} {P_j − P_i + C_{i,k} + C_{k,j}}
    end for
end for
return C_{0,m−1}
```

A.5 Παράδειγμα υλοποίησης/ Τρόπου σκέψης του αλγορίθμου

Έστω ότι έχουμε την συμβολοσειρά $ABCDEFG$ με $n = 7$, και θα θέλαμε να την διασπάσουμε στις θέσεις 1 και 5 (δηλαδή: $A|BCDE|FG$), P = [1,5] και $m = 2$, μη γνωρίζοντας με ποια σειρά, έτσι ώστε στο τέλος να έχουμε το ελάχιστο δυνατό κόστος διάσπασης. Τότε:

Αρχικά, στον πίνακα P προσθέτουμε τις εικονικές θέσεις τομών *0* και *7* (n δηλαδή), ώστε ο πίνακας P να περιλαμβάνει ολόκληρη τη συμβολοσειρά. Έτσι θα έχουμε: P = [0,1,5,7] (δηλαδή: $|A|BCDE|FG|$) και άρα το μέγεθος πλεόν του πίνακα P θα είναι $m = 4$.

Σε επόμενο στάδιο θα σχηματίσουμε έναν νέο πίνακα C διαστάσεων m x m (4 x 4) με τον οποίον θα υπολογίζουμε το κόστος τομών για κάθε υποπρόβλημα. Έτσι, τα στοιχεία $C_{i,j}$ του πίνακα C θα εκφράζουν το κόστος της τομής μεταξύ των θέσεων τομών P[i] και P[j] του πίνακα P. Προφανώς, τα στοιχεία $C_{i,i}$ είναι *0*, διότι δεν υπάρχει δυνατή τομή ανάμεσα στον ίδιο χαρακτήρα μιας συμβολοσειράς. Τα στοιχεία $C_{i,i+1}$ είναι επίσης *0*, διότι δεν υπάρχει δυνατή τομή ανάμεσα στις θέσεις τομών i και $i+1$, αφού οι θέσεις αυτές είναι διαδοχικές. Να σημειωθεί ότι τα στοιχεία i και j του πίνακα C αναφέρονται στις θέσεις i και j του πίνακα P και όχι στις θέσεις της συμβολοσειράς. Τα στοιχεία i και j του πίνακα P αναφέρονται στις αντίστοιχες θέσεις της γραμματοσειράς στις οποίες γίνονται οι τομές. Έτσι, με βάση αυτά έχουμε τον παρακάτω πίνακα.

	$j=0$	$j=1$	$j=2$	$j=3$
$i=0$	$C_{00}=0$	$C_{01}=0$	C_{02}	C_{03}
$i=1$	$C_{10}=NIL$	$C_{11}=0$	$C_{12}=0$	C_{13}
$i=2$	$C_{20}=NIL$	$C_{21}=NIL$	$C_{22}=0$	$C_{23}=0$
$i=3$	$C_{30}=NIL$	$C_{31}=NIL$	$C_{32}=NIL$	$C_{33}=0$

Τώρα πρέπει να υπολογίσουμε τις τιμές του πίνακα στις θέσεις $C_{0,2}$, $C_{1,3}$ και τέλος $C_{0,3}$, που θα δώσει και την λύση στο αρχικό μας πρόβλημα. Για τους υπολογισμούς αυτούς έχουμε:

- Για το $C_{0,2}$: Συμβολίζει το κόστος των τομών που μπορώ να κάνω στη συμβολοσειρά μεταξύ P_0 και P_2. Βλέποντας τον πίνακα P, παρατηρώ ότι από P_0 μέχρι P_2. μπορώ να κάνω ακριβώς 1 κόψιμο, το $P_1(= 1$, δηλαδή κόψιμο στη θέση 1 της αρχικής συμβολοσειράς για υποπρόβλημα που αντιστοιχεί σε συμβολοσειρά μήκους $P_2 - P_0)$. Για τον λόγο αυτόν, θα έιναι: $C_{0,2} = P_2 - P_0(= 5)$, όσο και το μήκος της συμβολοσειράς που θέλω να κόψω σε αυτό το υποπρόβλημα (από P_0 μέχρι P_2).

- Για το $C_{1,3}$: Με εντελώς όμοιο τρόπο το $C_{1,3}$ συμβολίζει το κόστος των τομών που μπορώ να κάνω στη συμβολοσειρά μεταξύ P_1 και P_3. Βλέποντας τον πίνακα P, παρατηρώ ότι από P_1 μέχρι P_3. μπορώ να κάνω ακριβώς 1 κόψιμο, το $P_2(= 5$, δηλαδή κόψιμο στη θέση 5 της αρχικής συμβολοσειράς για υποπρόβλημα που αντιστοιχεί σε συμβολοσειρά μήκους $P_3 - P_1)$. Για τον λόγο αυτόν, θα έιναι: $C_{1,3} = P_3 - P_1(= 6)$, όσο και το μήκος της συμβολοσειράς που θέλω να κόψω σε αυτό το υποπρόβλημα (από P_1 μέχρι P_3).

- Για το $C_{0,3}$: Συμβολίζει το κόστος των τομών που μπορώ να κάνω στη συμβολοσειρά μεταξύ P_0 και P_3 και κατ επέκταση και το κόστος που θέλω να υπολογίσω για να λύσω το πρόβλημα. Βλέποντας τον πίνακα P, παρατηρώ ότι από P_0 μέχρι P_3. μπορώ να κάνω 2 κοψίματα. Πρώτα μπορώ να ξεκινήσω με το κόψιμο της συμβολοσειράς στη θέση P_1 και έπειτα στη θέση P_2 ή το ανάποδο.

  Εαν ξεκινήσω πρώτα με την τομή στη θέση P_1, τότε το ζητούμενο κόστος θα είναι:
  $C_{0,3} = (P_3 - P_0) + C_{0,1} + C_{1,3} = (7 - 0) + 0 + 6 = 13$

  Αντίστοιχα, εαν ξεκινήσω πρώτα με την τομή στη θέση P_2, τότε το ζητούμενο κόστος θα είναι:
  $C_{0,3} = (P_3 - P_0) + C_{0,2} + C_{2,3} = (7 - 0) + 5 + 0 = 12$

  Για να βρω λοιπόν το ελάχιστο υπολογιστικό κόστος για την διάσπαση της συμβολοσειράς για το συγκεκριμένο παράδειγμα, αρκεί να επιλέξω το:
  $\min\{(P_3 - P_0) + C_{0,1} + C_{1,3}, (P_3 - P_0) + C_{0,2} + C_{2,3}\} = \min\{13, 12\} = 12$

Η λογική του παραδείγματος αυτού μπορεί να γενικευτεί ώστε εν τέλει να παραχθεί ο *Αλγόριθμος για το Πρόβλημα 3: Ερώτημα 1.*