ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΣΥΝΔΥΑΣΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

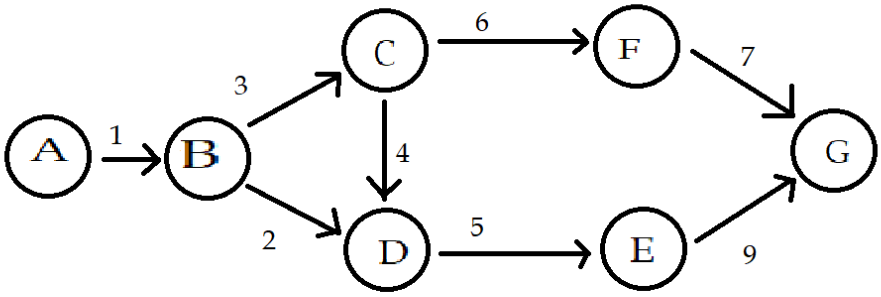
1η Ομάδα Ασκήσεων

Ονοματεπώνυμο: Δούρου Βασιλική Ευαγγελία

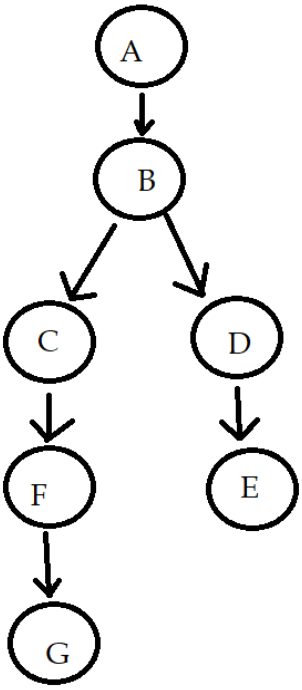
Α.Μ.: 1072633

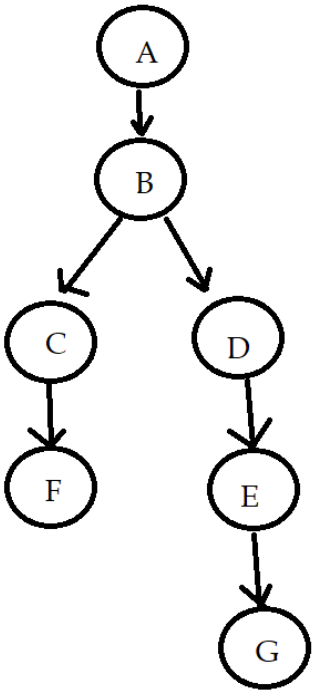
1. (α) Ο ισχυρισμός είναι ψευδής.

Αυτό αποδεικνύεται με το παρακάτω αντιπαράδειγμα. Έστω το ακόλουθο γράφημα, όπου κάθε ακμή έχει διαφορετικό κόστος:



Αν εκτελέσουμε κάποιον αλγόριθμο για την εύρεση του συντομότερου μονοπατιού από την κορυφή Α προς όλες τις υπόλοιπες κορυφές θα παρατηρήσουμε ότι η κορυφή G μπορεί να ανακαλυφθεί και από την κορυφή F και από την κορυφή E με κόστος d(G)=17 και στις δύο περιπτώσεις. Συνεπώς, μπορούν να προκύψουν τα ακόλουθα δύο δέντρα συντομότερων διαδρομών.





1. (β) Ο ισχυρισμός είναι αληθής.

Έστω το γράφημα G=(V,E). Αν το κόστος κάθε ακμής αυξάνεται κατά β μονάδες θα ισχύει $wt'(u_i,u_{i+1})=wt(u_i,u_{i+1})+\beta$, για κάθε $(u_i,u_i) \in E$, όπου Ε το σύνολο των ακμών του γραφήματος, και $u_i,u_{i+1} \in V$.

Γνωρίζουμε ότι για μία διαδρομή $P=(u_1,u_2,\ldots,u_\kappa)$ στο γράφημα G ισχύει $wt(P)=\sum_{i=1}^{\kappa-1} wt(u_i,u_{i+1})$.

Οπότε μετά την αλλαγή του κόστους όλων των ακμών το κόστος της διαδρομής θα ισούται με:

$wt'(P)=\sum_{i=1}^{\kappa-1} wt'(u_i,u_{i+1})=\sum_{i=1}^{\kappa-1}(wt(u_i,u_{i+1})+\beta)=(\kappa-1)\beta+\sum_{i=1}^{\kappa-1} wt(u_i,u_{i+1})=(\kappa-1)\beta+wt(P)$, όπου κ ο αριθμός των κορυφών.

Άρα, οι αποστάσεις από την αρχική κορυφή αυξάνονται κατά ένα πολλαπλάσιο του β για όλες τις κορυφές. Το αντίστοιχο ισχύει και για μείωση κατά μία σταθερά β.

2. Έστω Τ το αρχικό δέντρο συντομότερων διαδρομών από την κορυφή s προς όλες τις υπόλοιπες. Προσθέτουμε μία σταθερά α σε όλες τις εξερχόμενες ακμές της κορυφής s. Οπότε για κάθε κορυφή u ∈ V στην οποία εισέρχεται ακμή από την αρχική κορυφή s ισχύει: wt'(s,u)=wt(s,u)+α και επειδή τα αρχικά κόστη των ακμών είναι μη αρνητικά αν πριν ίσχυε $wt(s,u_1)<wt(s,u_2)$ μετά την προσθήκη του α θα ισχύει $wt(s,u_1)+α<wt(s,u_2)+α \Rightarrow wt'(s,u_1)<wt'(s,u_2)$. Άρα αν $δ(s,u_1)<δ(s,u_2)$ θα είναι και $δ'(s,u_1)<δ'(s,u_2)$(1) για κάθε κορυφή που συνδέεται απευθείας με την s.

Έστω Τ' το δέντρο συντομότερων διαδρομών που προκύπτει μετά την προσθήκη της σταθεράς α. Θα αποδείξουμε ότι το Τ ταυτίζεται με το Τ'. Έστω ότι είναι διαφορετικά και για μία κορυφή m για την οποία προηγουμένως ίσχυε δ(s,m)=δ(s,u)+wt(u,m), τώρα ισχύει δ'(s,m)=δ'(s,l)+wt(l,m), όπου l,u κορυφές που συνδέονται άμεσα με την αρχική κορυφή s, δηλαδή πλέον υπάρχει συντομότερη διαδρομή από την κορυφή l στην m, παρά από τη u στην m. Άρα πρέπει να ισχύει δ'(s,u)+wt(u,m)> δ'(s,l)+wt(l,m). Όμως τα wt(u,m) και wt(l,m) είναι αμετάβλητα και πριν την προσθήκη του α ίσχυε δ(s,u)+wt(u,m)<δ(s,l)+wt(l,m). Άρα από τη σχέση (1) προκύπτει ότι είναι άτοπο και τα δέντρα συντομότερων διαδρομών Τ και Τ' ταυτίζονται.

3. Για το ερώτημα αυτό τροποποιήθηκε ο αλγόριθμος SSSP-DAG της διαφάνειας 5 του σετ διαφανειών της ενότητας 2β, αφού γνωρίζουμε ότι το γράφημα είναι DAG με μη αρνητικά κόστη ακμών.

Μετά την τροποποίηση ο αλγόριθμός είναι ο ακόλουθος:

1. Εύρεση τοπολογικής διάταξης κορυφών v_1, v_2, v_k.
2. for i=1 to n {$d(v_i)=-\infty$;$pred(v_i)=0$;}

3. d(s)=0; //Υπόθεση ότι $s=v_k$ για κάποιο $1\leq k<n$
4. for i=k to n
   { for all $(v_i, z) \in E$
   { if $d(z)<d(v_i)+wt(v_i, z)$ then
   { $d(z)=d(v_i)+wt(v_i, z)$; $pred(z)=v_i$;}
   }
   }

Ο αλγόριθμος αυτός καθώς είναι μία μικρή παραλλαγή του SSSP-DAG έχει ίδια χρονική πολυπλοκότητα με αυτόν και επιλύει το πρόβλημα σε χρόνο O(n+m).

Αν το γράφημα G έχει κύκλους ο παραπάνω αλγόριθμος δεν μπορεί να λειτουργήσει αφού δεν θα μπορεί να ολοκληρωθεί το $1^ο$ βήμα του αλγορίθμου, δηλαδή να βρεθεί τοπολογική διάταξη. Αυτό συμβαίνει καθώς απαραίτητη προϋπόθεση για να προκύψει τοπολογική διάταξη από ένα γράφημα είναι να είναι ένα κατευθυνόμενο άκυκλο γράφημα.

4. Στο πρόβλημα της ταξινόμησης είχε αποδειχθεί ότι οι αλγόριθμοι Heap-Sort και Merge-Sort, δηλαδή οι αλγόριθμοι που εκμεταλλευόταν την ιδιότητα του σωρού, είχαν βέλτιστη πολυπλοκότητα χειρότερης περίπτωσης, όπως φαίνεται και από τον παρακάτω πίνακα:

Αλγόριθμος Ταξινόμησης	**Χρόνος Χειρότερης Περίπτωση**
Bubble Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Quick Sort	$O(n^2)$
Merge Sort	O(nlogn)
Heap Sort	O(nlogn)

Οι χρονικές πολυπλοκότητες που προκύπτουν από τη χρήση διαφορετικών σωρών είναι οι ακόλουθες (πηγή: https://en.wikipedia.org/wiki/Heap_(data_structure)):

Operation	find-min	delete-min	insert	decrease-key	meld
Binary[8]	Θ(1)	Θ(log *n*)	O(log *n*)	O(log *n*)	Θ(*n*)
Leftist	Θ(1)	Θ(log *n*)	Θ(log *n*)	O(log *n*)	Θ(log *n*)
Binomial[8][9]	Θ(1)	Θ(log *n*)	Θ(1)[a]	Θ(log *n*)	O(log *n*)[b]
Fibonacci[8][2]	Θ(1)	O(log *n*)[a]	Θ(1)	Θ(1)[a]	Θ(1)
Pairing[10]	Θ(1)	O(log *n*)[a]	Θ(1)	o(log *n*)[a][c]	Θ(1)
Brodal[13][d]	Θ(1)	O(log *n*)	Θ(1)	Θ(1)	Θ(1)
Rank-pairing[15]	Θ(1)	O(log *n*)[a]	Θ(1)	Θ(1)[a]	Θ(1)
Strict Fibonacci[16]	Θ(1)	O(log *n*)	Θ(1)	Θ(1)	Θ(1)
2–3 heap[17]	O(log *n*)	O(log *n*)[a]	O(log *n*)[a]	Θ(1)	?

Συνεπώς είναι φανερό ότι η ουρά προτεραιότητας Fibonacci έχει την καλύτερη χρονική πολυπλοκότητα από όλες τις διαφορετικές δομές σωρών.

Σε κάθε βήμα του αλγορίθμου του Dijkstra χρειάζεται να κάνουμε τα ακόλουθα:

1. Να βρούμε τον κόμβο με την ελάχιστη απόσταση από την κορυφή s (αρχική κορυφή).

2. Να ανανεώσουμε τις αποστάσεις προς όλες τις κορυφές από τον κόμβο εκείνο.

Το πρώτο βήμα είναι μία λειτουργία εύρεσης και αφαίρεσης του ελάχιστου στοιχείου που γίνεται σε χρόνο O(logn). Το δεύτερο βήμα γίνεται με χρήση του decrease-key που γίνεται σε χρόνο O(1).

Άρα, τελικά ο αλγόριθμος του Dijkstra μπορεί να υλοποιηθεί ως n meld, n delete-min και m decrease-key. Οπότε με χρήση

ουράς προτεραιότητας Fibonacci προκύπτει πολυπλοκότητα $O(n*1+n*\log n+m*1)=O(n+n\log n+m)=O(m+n\log n)$.

5. Αν σε ένα κατευθυνόμενο γράφημα n κορυφών, όλες οι κορυφές ενώνονται με όλες τις κορυφές, τότε στη χειρότερη περίπτωση μία κορυφή k θα έχει n-1 εισερχόμενες ακμές και n-1 εξερχόμενες ακμές και αν χρειαστεί να ελεγχθεί πάλι η ετικέτα της, τότε είναι πιθανό να πρέπει να ελεγχθεί n-1 φορές. Μετά και κάθε κορυφή που συνδέεται με την k με τις άλλες n-1 ακμές πρέπει να ελεγχθεί και αυτή και ούτω καθεξής. Αυτό έχει ως αποτέλεσμα η πολυπλοκότητα του αλγορίθμου να πάρει εκθετικές διαστάσεις και να γίνει ίση με $O(2^n)$, αφού και τα κόστη των ακμών ανήκουν στο διάστημα $[0,2^{n-1}]$.

6. Αφού το γράφημα G είναι διμερές ισχύει ότι $V_1 \cap V_2 = \emptyset$ και δεν υπάρχουν ακμές που να συνδέουν κορυφές του V_1 μεταξύ τους και κορυφές του V_2 μεταξύ τους. Άρα, οι κορυφές του V_1 μπορούν να ανακαλυφθούν μόνο από τις κορυφές του V_2 και αντίστροφα.

Αν $n_1=n_2$:

Τότε ως γνωστόν η πολυπλοκότητα του αλγορίθμου Bellman-Ford-Moore είναι ίση με $O(nm)$ και αφού $n=n_1+n_2=2n_1$, θα είναι ίση με $O(2n_1m)=O(n_1m)$.

Αν $n_1<n_2$:

Γνωρίζουμε πως μετά την i-οστή επανάληψη του βρόγχου while θα έχουν υπολογιστεί οι συντομότερες διαδρομές που περιλαμβάνουν i ακμές. Άρα, μετά από n_1 επαναλήψεις θα έχουν υπολογιστεί οι συντομότερες διαδρομές με n_1 ακμές.

Όμως, οι κορυφές του συνόλου V_1 μπορούν να συνδέονται με ακμές μόνο με τις κορυφές του συνόλου V_2 και αντίστροφα, όπως ειπώθηκε και προηγουμένως. Άρα, οι συντομότερες διαδρομές με n_1 ακμές θα έχουν τις μισές κορυφές να ανήκουν στο σύνολο V_1 και τις άλλες μισές να ανήκουν στο σύνολο V_2. Άρα, θα μένει να ανακαλυφθούν ακόμη $n_1/2$ κορυφές από το σύνολο V_1 και $n_2-n_1/2$ από το σύνολο V_2. Οπότε μετά από άλλες n_1 επαναλήψεις θα έχουν ανακαλυφθεί οι συντομότερες διαδρομές για όλες τις κορυφές του συνόλου V_1 και θα μένει να ανακαλυφθούν οι συντομότερες διαδρομές για τις υπόλοιπες n_2-n_1 κορυφές του συνόλου V_2. Αυτό, όμως, μπορεί να γίνει στην επόμενη αμέσως επανάληψη, αφού θα βρεθούν οι συντομότερες διαδρομές από τις κορυφές του V_1 που έχουν βρεθεί ήδη οι αντίστοιχες δικές τους. Άρα, συνολικά χρειάζονται $2n_1+1$ επαναλήψεις του βρόγχου while. Οπότε, η χρονική πολυπλοκότητα του αλγορίθμου είναι $O((2n_1+1)m)=O(n_1m)$.

Συνεπώς, σε κάθε περίπτωση η χρονική πολυπλοκότητα ισούται με $O(n_1m)$.

7. (α) Θα δείξουμε, αρχικά, ότι μετά την αλλαγή του κόστους της ακμής wt(u,v) σε wt’(u,v)<wt(u,v) αν δ(v,u)+wt’(u,v)<0 υπάρχει αρνητικός κύκλος.

Αν η ακμή (u,v) άνηκε στο δ(u,v), τότε θα ανήκει ακόμη αφού wt’(u,v)<wt(u,v). Άρα, αν δ(v,u)+wt’(u,v)<0⇒δ(v,v)<0, άρα υπάρχει αρνητικός κύκλος.

Αν η ακμή (u,v) δεν άνηκε στο δ(u,v), τότε υπάρχει συντομότερη διαδρομή από το u στο v με d(v)<wt’(u,v)⇒δ(v,u)+d(v)<δ(v,u)+wt’(u,v)⇒δ(v,v)<0, οπότε πάλι υπάρχει αρνητικός. Άρα, αναγνωρίζει με επιτυχία αν υπάρχει αρνητικός κύκλος.

Αν δεν υπάρχει αρνητικός κύκλος, τότε ο αλγόριθμος βρίσκει τη ελάχιστη διαδρομή από την κορυφή x στην κορυφή y, για κάθε x,y∈ V, υπολογίζοντας το min μεταξύ της υπάρχουσας διαδρομής και της εναλλακτικής αν χρησιμοποιηθεί η ακμή (u,v), σε περίπτωση που δεν υπήρχε ήδη στη συντομότερη διαδρομή, αφού μόνο αυτής το κόστος αλλάζει. Οπότε, ο αλγόριθμος διατηρεί σωστά τις συντομότερες διαδρομές μεταξύ όλων των κορυφών.

(β) Ο παραπάνω αλγόριθμος εκτελείται για κάθε ζεύγος κορυφών x,y∈ V σε ένα κατευθυνόμενο γράφημα. Οπότε για κάθε κορυφή εκτελείται n-1 φορές και έχουμε n κορυφές. Άρα, συνολικά εκτελείται $n(n-1)=n^2-n$ φορές και η πολυπλοκότητα του είναι ίση με $O(n^2)$.

(γ) Αν αυξηθεί το κόστος της ακμής wt(u,v) σε wt'(u,v)≥wt(u,v) τότε αν η ακμή wt(u,v) άνηκε πριν στη συντομότερη διαδρομή δ(x,y) από μία κορυφή x σε μία κορυφή y, τότε υπάρχει περίπτωση ή να μην ανήκει πλέον και να βρεθεί άλλη πιο σύντομη διαδρομή ανάμεσα σε αυτές τις δύο κορυφές ή να ανήκει πάλι και να μη μεταβάλλεται καθόλου η διαδρομή. Αν δεν άνηκε πριν στη δ(x,y), τότε ούτε μετά την αλλαγή θα ανήκει στη δ(x,y). Άρα, δεν υπάρχει περίπτωση η ακμή (u,v) να μην υπήρχε πριν στην συντομότερη διαδρομή και να υπάρξει μετά την αλλαγή. Οπότε, παρόλο που για την ανίχνευση ύπαρξης αρνητικού κύκλου ο αλγόριθμος της άσκησης θα λειτουργεί σωστά, μετά πρέπει να βρεθούν εξαρχής όλες οι συντομότερες διαδρομές στις οποίες άνηκε πριν η (u,v) με κάποιον από τους αλγορίθμους που λύνουν το πρόβλημα ΣΔ(*,*).

8.(α) Ο υπολογισμός της πιο αξιόπιστης διαδρομής ανάγεται σε υπολογισμό του μέγιστου γινομένου των επιμέρους r(u,v),

δηλαδή στον υπολογισμό του $max \prod_{i=1}^{k-1} r(v_i, v_{i+1})$. Οι αλγόριθμοι εύρεσης συντομότερων διαδρομών υπολογίζουν το min.

Επειδή στον υπολογισμό του $max \prod_{i=1}^{k-1} r(v_i, v_{i+1})$ δεν μας ενδιαφέρει η τιμή που προκύπτει αλλά η διαδρομή που προκύπτει, μπορεί ο υπολογισμός που θέλουμε να κάνουμε να μετατραπεί σε $maxlog(\prod_{i=1}^{k-1} r(v_i, v_{i+1}))$. Επειδή ο λογάριθμος είναι γνησίως αύξουσα συνάρτηση μπορούμε να την εφαρμόσουμε χωρίς αλλαγή του αποτελέσματος και να προκύψει $max \sum_{i=1}^{k-1} \log\big(r(v_i, v_{i+1})\big)$. Αν υπολογίσουμε το μείον του αθροίσματος τότε το πρόβλημα μετατρέπεται σε υπολογισμό του ελαχίστου, δηλαδή σε

$$\min\left(-\sum_{i=1}^{k-1} \log\big(r(v_i, v_{i+1})\big)\right).$$

Αν το μείον αντί να τοποθετηθεί έξω από το άθροισμα, τοποθετηθεί εντός, προκύπτει:

$$\min \sum_{i=1}^{k-1} -\log\big(r(v_i, v_{i+1})\big).$$

Έτσι, το πρόβλημα μετατράπηκε σε υπολογισμό της συντομότερης διαδρομής.

(β) Χρησιμοποιήθηκε ο ψευδοκώδικας της διαφάνειας 10 του σετ διαφανειών της ενότητας 2β. Η παραλλαγή του αλγορίθμου του Dijkstra για την επίλυση του προβλήματος ΔΜΑΑΚ είναι η ακόλουθη:

S= ∅; $\bar{S}$=V;

for every u∈V do d(u)=-∞;

d(s)=0; pred(s)=0;

while |S|<n do

let u∈$\bar{S}$ be the vertex for which d(u)=max$_{v\in\bar{S}}${d(v)};

S=S ∪{u}; $\bar{S}$=$\bar{S}$-{u};

for each (u,w)∈E do

if d(w)<d(u)+wt(u,w) then

d(w)=d(u)+wt(u,w); pred(w)=u;

od

od

Η πολυπλοκότητα αυτού του αλγορίθμου είναι ίδια με αυτή του κλασσικού Dijkstra και είναι ίση με O(n^2). Η απόδειξη της ορθότητας του αλγορίθμου θα βασιστεί στην απόδειξη της ορθότητας του Dijkstra που βρίσκεται στις διαφάνειες 22-24 του σετ διαφανειών της ενότητας 2β. Με δ(x,y) συμβολίζεται η μακρύτερη διαδρομή από τη κορυφή x στη κορυφή y.

Θα αποδείξουμε ότι για κάθε v∈V d(v)≤δ(s,v).

Έστω u η πρώτη κορυφή του S' που επιλέχθηκε αν και υπάρχει μεγαλύτερη διαδρομή από d(u), δηλαδή d(u)<δ(s,v) και θα καταλήξουμε σε αντίφαση.

Έστω y η πρώτη κορυφή στο S' στη ΜΔ(s,u). Τότε, d(y)=δ(s,y):

d(x)=δ(s,x)(x∈S η προηγούμενη της y) στη ΜΔ(s,u).

Όταν ο αλγόριθμος τοποθέτησε την x στο S, χαλάρωσε την ακμή (x,y), θέτοντας d(y)=d(x)+wt(x,y)=δ(s,x)+wt(x,y).

d(u)<δ(s,u)=δ(s,y)+δ(y,u)=d(y)+δ(y,u)≤d(y).

Αν, όμως, d(u)<d(y), τότε ο αλγόριθμος θα επέλεγε την y και όχι τη u ⇒ αντίφαση.

Άρα, d(u)=δ(s,u) όταν η u εισάγεται στο S, και άρα ο αλγόριθμος είναι ορθός.