Αλγόριθμοι
3η Σειρά Γραπτών Ασκήσεων

Ιωάννης Δάρας (03115018, el15018@central.ntua.gr, daras.giannhs@gmail.com)

"Perhaps the most important principle for the good algorithm designer is to refuse to be content."
—Alfred V. Aho

---

1 Άσκηση 1

1.1 Διατύπωση αλγορίθμου

Ο αλγόριθμος για την εύρεση του μακρύτερου μονοπατιού σε ένα δέντρο βασίζεται σε μια postorder διάσχιση του δέντρου. Για την άσκηση μας, postorder διάσχιση ενός δέντρου θεωρούμε ότι όταν θέλουμε να εξερευνήσουμε ένα υπόδεντρο, εξερευνούμε τελευταία τη ρίζα του υποδέντρου, δηλαδή πρώτα εξερευνούμε όλα τα παιδιά και στη συνέχεια τη ρίζα. Ξεκινάμε την ανάλυση του αλγορίθμου με ορισμένες παρατηρήσεις.

Ορίζουμε ως $P(n)$ το κατηγόρημα να περνάει το μονοπάτι p, όπου p ένα τυχαίο μονοπάτι του δέντρου, από τον κόμβο n και $R(u, v)$ το κατηγόρημα που μας λέει ότι η κορυφή v είναι παιδί της κορυφής u. Τότε, για οποιοδήποτε μονοπάτι p σε ένα δέντρο ισχύει ένα από τα ακόλουθα:

1. $$\nexists n \in V : P(n) \wedge P(u) \wedge P(v) \wedge R(n, u) \wedge R(n, v)$$
   Δηλαδή δεν υπάρχει κορυφή που το μονοπάτι να περνάει από παραπάνω από ένα υπόδεντρα της.

2. Ορίζουμε ως T(n) το κατηγόρημα:
   $$T(n) =:= P(n) \wedge P(u) \wedge P(v) \wedge R(n, u) \wedge R(n, v)$$
   Τότε:
   $$\exists n \in V : T(n) \wedge \exists w \in V : T(w) \implies n = w$$
   Αυτό σημαίνει ότι υπάρχει μόνο μία κορυφή που ανήκει στο μονοπάτι και το μονοπάτι περνάει και από το αριστερό και από το δεξί της υπόδεντρο.

Συνδυάζοντας τα 1. , 2. παίρνουμε ότι:

Θεώρημα 1 *Υπάρχει το πολύ 1 κορυφή που ένα μονοπάτι του δέντρου περνάει και από το αριστερό και από το δεξί της υπόδεντρο*

Αφού και το μακρύτερο μονοπάτι του δέντρου είναι και αυτό ένα μονοπάτι τότε και για αυτό θα ισχύει αυτή η ιδιότητα. Με βάση αυτή την παρατήρηση μπορούμε να υπολογίσουμε για κάθε κόμβο ποιό είναι το μεγαλύτερο προκύπτον μονοπάτι για τον κόμβο αυτό αν το μονοπάτι ξεκίναει (ή τελείωνει, όπως το δει κανείς) για τις ακόλουθες δύο επιλογές:

1. Το μονοπάτι που ξεκινάει από τον κόμβο δεν περνά από δύο υπόδεντρα του.

   Τότε, το καλύτερο μονοπάτι για τον κόμβο είναι είτε το max από τα καλύτερα μονοπάτια των παιδιών του + την αξία του κόμβου, αν υπάρχει παιδί με θετικής αξίας καλύτερο μονοπάτι, είτε η αξία του κόμβου.

2. Το μονοπάτι περνά που ξεκινάει από τον κόμβο περνάει από δύο υπόδεντρα του.

   Τότε, το καλύτερο μονοπάτι για τον κόμβο είναι το καλύτερο μονοπάτι του καλύτερου υποδέντρου + την αξία του κόμβου + το καλύτερο μονοπάτι του δεύτερου καλύτερου υποδέντρου.

Εφαρμόζοντας postorder διάσχιση, μπορούμε να υπολογίσουμε με προφανή τρόπο τα καλύτερα μονοπάτια για τα φύλλα και στη συνέχεια να υπολογίσουμε τα καλύτερα μονοπάτια των προγόνων παίρνοντας τα καλύτερα μονοπάτια των απογόνων που δεν έχουν διπλώσει (δηλαδή δεν περνάνε για κανέναν απόγονο από παραπάνω από ένα υπόδεντρο).

Κατά τη διάρκεια της preorder διάσχισης μπορούμε να κρατάμε τις ακόλουθες πληροφορίες:

- Για κάθε κόμβο έναν πίνακα δύο θέσεων με τους απογόνους για κάθε μία από τις περιπτώσεις, δηλαδή για την περίπτωση που το μονοπάτι περνάει από παραπάνω από έναν άμεσο απόγονο και για την περίπτωση που το μονοπάτι περνάει μόνο από έναν άμεσο απόγονο.
- Για κάθε κόμβο έναν πίνακα δύο θέσεων με τις τιμές του καλύτερου μονοπατιού για κάθε μία από τις περιπτώσεις.
- Συνολικά, μια τιμή που αναπαριστά την αξία του καλύτερου μονοπατιού που έχουμε συναντήσει.
- Συνολικά, έναν δείκτη στον κόμβο από τον οποίο "ξεκινά" το καλύτερο μονοπάτι που έχουμε συναντήσει.

Στο τέλος της postorder διάσχισης, μπορούμε μέσω του δείκτη στον κόμβο που έχει το καλύτερο μονοπάτι και ακολουθώντας τους πίνακες απογόνων των κόμβων να ανακτήσουμε το καλύτερο μονοπάτι.

1.2 Ορθότητα αλγορίθμου

Θα αποδείξουμε την ορθότητα του αλγορίθμου με επαγωγή. Υποθέτουμε ότι για έναν τυχαίο κόμβο του δέντρου έχουμε υπολογίσει ορθά το μακρύτερο μονοπάτι που μπορεί να ξεκινά για καθέναν από τους κόμβους παιδιά του (επαγωγική υπόθεση). Θα δείξουμε ότι και για αυτόν τον κόμβο μπορούμε να υπολογίσουμε το μακρύτερο μονοπάτι που ξεκινά από αυτόν.

Αρχικά, αφού το μονοπάτι ξεκινάει από τον κόμβο προς εξέταση δεν μας ενδιαφέρει ο γονέας του συγκεκριμένου κόμβου. Οι κόμβοι που εξετάζουμε είναι οι κόμβοι παιδιά που έχουν ήδη υπολογιστεί από επαγωγική υπόθεση (optimal substructure property).

Όπως αναλύσαμε και στην ενότητα διατύπωσης του αλγορίθμου, το μονοπάτι μπορεί είτε να περνάει από δύο υπόδεντρα του προς εξέταση κόμβου, είτε από ένα είτε από κανένα. Έστω v_1, v_2 οι δύο μεγαλύτερες αξίες των μονοπατιών των απογόνων που δεν έχουν διπλώσει και w η αξία του προς εξέταση κόμβου. Η αξία V του μακρύτερου μονοπατιού p που ξεκινάει από τον κόμβο αξίας w θα είναι:

$$max(v_1 + w, v_2 + w, v_1 + v_2 + w, w)$$

εφόσον άμα πάμε σε κάποιο από τα μονοπάτια θα περάσουμε από το προϋπολογισμένο μακρύτερο μονοπάτι + την αξία του κόμβου και αν το μονοπάτι "διπλώσει" θα περάσουμε από όλα.

Ο αλγόριθμος μας ακολουθεί ακριβώς αυτό το κριτήριο και συνεπώς είναι ορθός.

1.3 Πολυπλοκότητα λύσης

Η πολυπλοκότητα της λύσης μας είναι η πολυπλοκότητα της αναζήτησης σε ένα δέντρο, δηλαδή: $O(V+E)$

2 Άσκηση 2

2.1 (α)

2.1.1 Διατύπωση αλγορίθμου

Θα εκμεταλλευτούμε την ιδιότητα της τοπογραφικής ταξινόμησης που έχουν τα DAG γραφήματα.

Αρχικά, ταξινομούμε τοπολογικά το γράφο. Για να ταξινομήσουμε τοπολογικά το γράφο κάνουμε DFS με μία στοίβα. Συγκεκριμένα, κατά την DFS διάσχιση χωρίζουμε τις κορυφές σε τρεις κατηγορίες: σε αυτές που δεν έχουμε ακόμα εξερευνήσει, αυτές που έχουμε μερικώς εξερευνήσει και αυτές που έχουμε εξερευνήσει τελείως. Αρχικά όλες οι κορυφές είναι ανεξερεύνητες. Διαλέγουμε μια τυχαία και κάνουμε DFS διάσχιση. Όταν συναντάμε κορυφές που δεν έχουμε ξαναδεί τις μεταφέρουμε στις μερικώς εξερευνημένες και πάμε να εξερευνήσουμε ένα από τα ανεξερεύνητα παιδιά της. Αν όλα τα παιδιά της είναι εξερευνημένα, τότε η κορυφή μεταφέρεται στις πλήρως εξερευνημένες και την εισάγουμε στη στοίβα. Συνεχίζουμε επαναληπτικά μέχρι να εξερευνηθούν όλες οι κορυφές. Στο τελος, εξάγουμε ένα ένα τα στοιχεία από τη στοίβα και παίρνουμε την τοπολογική ταξινόμηση του γράφου.

Αφού ολοκληρώσουμε την τοπολογική ταξινόμηση, βάζουμε αρχικά σε όλες τις κορυφές ως κόστος την τιμή της κορυφής τους. Στη συνέχεια, ο αλγόριθμος μας επισκέπτεται μία μία τις κορυφές με την ανάποδη σειρά στην τοπολογική τους ταξινόμηση και κάνει τα αντίστοιχα updates στις κορυφές που έχουν ακμές προς αυτή. Τα updates γίνονται ως εξής: Έστω p(u) η τιμή μιας κορυφής u και κορυφές v, w που έχουν ακμές προς την κορυφή u και ως τώρα υπολογισμένα κόστη c(v), c(w) αντίστοιχα. Τότε:

$$c(v) \leftarrow min\left(c(v), c(u)\right), \quad c(w) \leftarrow min\left(c(w), c(u)\right)$$

Σε μορφή ψευδοκώδικα:

```
V ← topological_sort(G)
for v ∈ reverse(V) do
    c(v) ← p(v)
    for u ∈ V : (u, v) ∈ E do
        if c(v) < c(u) then
            c(u) ← c(v)
        end
    end
end
return c
```

Algorithm 1: Cost function on DAG

2.1.2 Απόδειξη ορθότητας

Ο λόγος που ο αλγόριθμος μας δουλεύει είναι ότι υπάρχει το optimal substructure property. Διαισθητικά αυτό σημαίνει για την περίπτωση μας ότι κάθε φορά που ανανεώνουμε το κόστος μιας κορυφής u με βάση μια ακμή της προς μια κορυφή v, το κόστος της κορυφής v έχει ήδη υπολογιστεί σωστά.

Αποδεικνύουμε την ορθότητα του αλγορίθμου μας αυστηρά μαθηματικά με επαγωγή. Έστω ότι έχουμε υπολογίσει σωστά τα κόστη των τελευταίων n κορυφών της τοπολογικής διάταξης. Θα δείξουμε ότι ο αλγόριθμος μας δίνει σωστό κόστος στην κορυφή n+1 από το τέλος της τοπολογικής διάταξης.

Αρχικά, για τη συγκεκριμένη κορυφή γνωρίζουμε ότι έχει ακμές μόνο προς κορυφές που έχουν ήδη υπολογιστεί σωστά οι τιμές τους από επαγωγική υπόθεση, λόγω της τοπολογικής διάταξης. Αυτό μας δίνει το optimal substructure του προβλήματος. Ο αλγόριθμος μας εξέτασε αυτές τις ακμές όταν εξέτασε τις κορυφές στις οποίες οι ακμές καταλήγουν. Δεδομένου ότι όταν εξέταζε αυτές τις ακμές τα κόστη των κορυφών αυτών είχαν πάρει τη σωστή τιμή (από επαγωγική υπόθεση) παίρνοντας το min των κορυφών αυτών και ο προς εξέταση κόμβος θα πάρει τη σωστή τιμή. Συνεπώς, ο αλγόριθμος μας είναι ορθός.

2.1.3 Πολυπλοκότητα

Ο αλγόριθμος μας αρχικά φτιάχνει μια τοπολογική διάταξη του γράφου. Η τοπολογική διάταξη φτιάχνεται με DFS, δηλαδή έχει πολυπλοκότητα $O(V + E)$. Στη συνέχεια, εξετάζει όλες τις ακμές που εμπίπτουν σε μια κορυφή εξετάζοντας όλες τις κορυφές. Αυτή η πράξη έχει επίσης πολυπλοκότητα $O(V + E)$. Έτσι, η συνολική πολυπλοκότητα του αλγορίθμου μας είναι: $O(V + E)$.

(β)

2.1.4 Διατύπωση αλγορίθμου

Ο αλγόριθμος μας βασίζεται στις ισχυρά συνεκτικές συνιστώσες του γράφου. Το πρώτο βήμα είναι να βρούμε τις ισχυρά συνεκτικές συνιστώσες. Για να το κάνουμε αυτό, χρησιμοποιούμε τον αλγόριθμο του Kosaraju's. Συγκεκριμένα, αρχικά εκτελούμε DFS μέχρι να ανακαλύψουμε όλες τις ακμές και χρησιμοποιούμε μια στοίβα για να αποθηκεύσουμε τη σειρά ανακάλυψης των ακμών, όπως ακριβώς κάναμε και στην τοπολογική ταξινόμηση. Στη συνέχεια, αντιστρέφουμε τη φορά όλων των ακμών στο γράφο και εκτελούμε στον καινούργιο γράφο πάλι DFS ξεκινώντας από την κορυφή που ανακαλύφθηκε τελευταία στην πρώτη DFS και συνεχίζοντας με τη σειρά που υπαγορευει η στοίβα ανακάλυψης ακμών. Για κάθε κορυφή από την οποία ξεκινάμε DFS στον καινούργιο γράφο, οι κορυφές που ανακαλύπτει είναι οι κορυφές του strongly connected component στο οποίο ανήκει. Κάνουμε τις κορυφές που ανακαλύφθηκαν visited και συνεχίζουμε με την επόμενη κορυφή της στοίβας για να βρούμε το επόμενο strongly connected component.

Ο λόγος που χρησιμοποιομε τα strongly connected components είναι ότι κάθε κατευθυνόμενος γράφος είναι ένα DAG από τα strongly connected components του. Έτσι, μπορούμε αρχικά να φτιάξουμε μια κορυφή για κάθε strongly connected component και να φτιάξουμε από τις ακμές E τις ακμές E' για το νέο γράφο. Στο νεό γράφο, ο οποίος είναι dag, μπορούμε να τρέξουμε τον αλγόριθμο που περιγράψαμε στο βήμα (α). Τέλος, το μόνο που μένει να κάνουμε είναι να βάλουμε σε κάθε κορυφή την τιμή του strongly connected component στο οποίο ανήκει, η οποία υπολογιστήκε από τον αλγόριθμο του (α).

Σε μορφή ψευδοκώδικα:

```
/* get strongly connected components                          */
G_1, G_2, ..., G_n ← scc(G)
/* create graph of strongly connected components              */
G' ← create_scc_graph(G_1, G_2, ..., G_n, E)
/* run algorithm of (a)                                       */
c ← run_a_algorithm(G')
/* set cost of each vertex to the cost of the scc it belongs  */
c ← set_cost(V, G', c)
return c
```

Algorithm 2: Cost function on G

2.1.5 Απόδειξη ορθότητας

Για την απόδειξη ορθότητας πρέπει να αποδείξουμε τα ακόλουθα θεωρήματα:

Θεώρημα 2 *Κάθε γράφος μπορεί να μετατραπεί σε ένα dag από τα strongly connected components του.*

Θα δουλέψουμε με άτοπο. Έστω ότι ο γράφος των strongly connected components δεν είναι dag, δηλαδή υπάρχει κύκλος. Αφού υπάρχει κύκλος μπορώ να πάω από το strongly component u σε ένα strongly component v και στη συνέχεια να ξαναγυρίσω στο strongly component u (πιθανώς και μέσω από άλλα

strongly components w, t, p). Αφού μπορώ να το κάνω αυτό θα έπρεπε τα u,v να ανήκουν στο ίδιο strongly connected component αφού μπορώ από κάθε κορυφή του u να φτάσω σε κάθε κορυφή του v και αντίστροφα, άρα άτοπο.

Συνεπώς, ο γράφος των strongly connected components είναι DAG.

Θεώρημα 3 *Κάθε κορυφή σε ένα strongly connected component έχει το κόστος του strongly connected component.*

Η απόδειξη της πρότασης αυτής είναι ευθεία. Το strongly connected component έχει το κόστος της μικρότερης κορυφής του καθώς αυτή συνδέεται με όλες τις άλλες κορυφές και άρα με όλες τις κορυφές που αυτές συνδέονται. Όμως αφού το component είναι strongly connected και όλες οι άλλες κορυφές συνδέονται με αυτή και αφού αυτή έχει το ελάχιστο κόστος στο component αυτές έχουν το κόστος αυτής. Άρα, όλες οι κορυφές σε ένα strongly connected component έχουν το ίδιο κόστος, το κόστος του component.

Αφού αποδείξαμε τα παραπάνω δύο θεωρήματα, η ορθότητα του αλγορίθμου μας προκύπτει άμεσα από την ορθότητα του αλγορίθμου του (α) που έχουμε ήδη αποδείξει.

2.1.6 Πολυπλοκότητα

Αρχικά χωρίζουμε σε συνεκτικές συνιστώσες με τον αλγόριθμο του Kosaraju's. Για να το κάνουμε αυτό τρέχουμε δύο φορές DFS συνεπώς η πολυπλοκότητα της πράξης είναι: $O(V+E)$. Στη συνέχεια, φτιάχνουμε τον γράφο των strongly connected components. Ο γράφος αυτός έχει το πολύ V κορυφές και για να τον φτιάξουμε εξετάζουμε όλες τις ακμές, συνεπώς η πολυπλοκότητα κατασκευής του γράφου είναι: $O(V+E)$. Στη συνέχεια, τρέχουμε τον αλγόριθμο του ερωτήματος (α) με πολυπλοκότητα $O(V+E)$. Τέλος, για κάθε κορυφή δίνουμε το κόστος του strongly connected component στο οποίο ανήκει με πολυπλοκότητα $O(V)$. Έτσι, η συνολική πολυπλοκότητα του αλγορίθμου είναι:

$$O(V+E)$$

3 Άσκηση 3

3.1 Minimax algorithm

Προαπαιτούμενο για τη διατύπωση αλγορίθμου για τη συγκεκριμένη άσκηση είναι να διατυπώσουμε τον αλγόριθμο Minimax. Ο αλγόριθμος Minimax είναι ένας αλγόριθμος εύρεσης βέλτιστης ακολουθίας κινήσεων σε ένα παιχνίδι δύο παικτών που παίζεται σε γύρους. Κάθε κατάσταση του παιχνιδιού μοντελοποιείται ως ένας κόμβος ενός γράφου με ακμές προς όλες τις επόμενες δυνατές καταστάσεις του παιχνιδιού. Σε κάθε κόμβο του γράφου δίνεται μια τιμή από μια ευριστική συνάρτηση που δείχνει το "πόσο" νικάει ο παίκτης που παίζει πρώτος. Ο παίκτης που παίζει πρώτος προσπαθεί να μεγιστοποιήσει αυτή την τιμή ενώ ο παίκτης που παίζει δεύτερος προσπαθεί να την ελαχιστοποιήσει. Ο παίκτης που παίζει πρώτος διαλέγει να κάνει την κίνηση που θα τον οδηγήσει σε επόμενη κατάσταση που θα νικάει όσο το δυνατόν περισσότερο, δηλαδή διαλέγει πάντα από τις επόμενες καταστάσεις αυτή με το μεγαλύτερο σκορ. Αντίθετα, ο παίκτης που παίζει δεύτερος διαλέγει από τις επόμενες καταστάσεις αυτή με το λιγότερο σκορ προκειμένου να πετύχει το αντίθετο αποτέλεσμα.

3.2 AB Pruning

Μια γρηγορότερη παραλλαγή του αλγορίθμου Minimax είναι ο αλγόριθμος AB Pruning. Ο αλγόριθμος αυτός έχει πολυπλοκότητα χειρότερης περίπτωσης ίδια με τον αλγόριθμο Minimax και συνεπώς δεν τον εξετάζουμε στα πλαίσια αυτής της άσκησης. Τον αναφέρουμε μόνο για λόγους πληρότητας.

3.3 Διατύπωση αλγορίθμου

Το πρώτο πράγμα που κάνουμε είναι να φτιάξουμε ένα νέο γράφο $G^{'}(V^{'}, E^{'})$ ο οποίος να περιέχει ως καταστάσεις $V^{'}$ όλες τις τριάδες $i, j, k : i, j \in V,\ k \in [0, 1]$ όπου το i είναι η κορυφή στην οποία βρίσκεται ο Κώστας, j η κορυφή στην οποία βρίσκεται ο Αντρέας και k δείχνει ποιος παίκτης παίζει, με k=0 να παίζει ο Κώστας. Αυτός ο γράφος έχει ως ακμές όλες τις επιτρεπτές μεταβάσεις που υπακούουν στους περιορισμούς

του παιχνιδιού, δηλαδή όταν k=0 μετακινούμε το i σε γειτονική κορυφή από το γράφο G και όταν k=1 μετακινούμε το j σε γειτονική κορυφή από το γράφο G με την προϋπόθεση $j \neq d$. Αφού φτιάξουμε τις ακμές με τον τρόπο που αναφέραμε τρέχουμε αναζήτηση bfs από τον κόμβο που αρχίζει ο Κώστας και για κάθε κόμβο που συναντάμε σημειώνουμε τις τιμές (a, b, c):

- a: μας δείχνει πόσο απέχει ο κόμβος από την κοντινότερη νίκη για το συγκεκριμένο παίκτη.

  Για τον Κώστα νίκη είναι να βρεθεί στο καταφύγιο ενώ για τον Αντρέα νίκη είναι να βρεθεί στην ίδια θέση με τον Κώστα.

- b: μας δείχνει πόσο απέχει ο κόμβος από την κοντινότερη ήττα για το συγκεκριμένο παίκτη.

  Για τον Κώστα ήττα είναι να βρεθεί σε ίδια θέση με τον Αντρέα ενώ για τον Αντρέα ήττα είναι να βρεθεί ο Κώστας στο καταφύγιο.

- c: μας δείχνει πόσο απέχει ο κόμβος από την κοντινότερη ισοπαλία

  Η ισοπαλία ισοδυναμεί με εύρεση κύκλου στο γράφο, δηλαδή να βρούμε ακμή που να μας ξαναγυρίζει σε κάποια visited.

Αφού σημειώσουμε αυτές τις τιμές για τον κάθε κόμβο, κάνουμε άλλο ένα πέρασμα του γράφου και αυτή τη φορά επισημείωνουμε άλλες δύο τιμές t, p για τον κάθε κόμβο του γράφου ως εξής:

$$t = b - a, \quad p = b - c$$

Αφού κάνουμε αυτό εφαρμόζουμε τον αλγόριθμο του Minimax δύο φορές, μία για τον γράφο με τις επισημειώσεις t και μία με τον γράφο με τις επισημειώσεις p. Ο παίκτης Max είναι ο Κώστας που προσπαθεί να μεγιστοποιήσει το t, δηλαδή να απομακρύνει την ήττα του και να φέρει κοντά την νίκη του ενώ ο παίκτης Min είναι ο Αντρέας που προσπαθεί να ελαχιστοποιήσει το t, δηλαδή να πετύχει το αντίστροφο αποτέλεσμα.

Στο τέλος, κάνουμε trace αρχικά τον Minimax για τον γράφο με τις επισημειώσεις t και βλέπουμε ποιά ακολουθία κινήσεων θα γίνει και ποιός τελικά θα κερδίσει. Αν δούμε ότι κερδίζει ο Αντρέας, τότε κάνουμε trace και τον Minimax στον γράφο με τις επισημειώσεις p προκειμένου να δούμε αν ο Κώστας με διαφορετική ακολουθία κινήσεων μπορεί να πετύχει ισοπαλία.

3.4 Πολυπλοκότητα

3.4.1 Κατασκευή γράφου

Ο γράφος έχει $2V^2$ καταστάσεις και οι μεταβάσεις για κάθε κατάσταση υπολογίζονται σε $O(1)$. Έτσι, η συνολική πολυπλοκότητα για την κατασκευή του γράφου είναι: $O(V^2)$

3.4.2 Επισημειώσεις a, b, c

Για να επισημειώσουμε κάθε κόμβο του γράφου με a, b, c πρέπει να τρέξουμε μια BFS τον κόμβο που ξεκινά ο Κώστας. Η πολυπλοκότητα της BFS για γράφο με V^2 κορυφές και $\approx E^2$ ακμές είναι:

$$O(V^2 + E^2)$$

.

3.4.3 Minimax

Ο minimax κάνει μια διάσχιση του γράφου, συνεπώς έχει πολυπλοκότητα: $O(V^2 + E^2)$.

Από τα παραπάνω, προκύπτει ότι η συνολική πολυπλοκότητα του αλγορίθμου μας είναι:

$$O(V^2 + E^2)$$

3.5 Ορθότητα αλγορίθμου

Η ορθότητα του συγκεκριμένου αλγορίθμου προκύπτει από τα ακόλουθα :

1. Συνέπεια ευριστικής συνάρτησης για τους δύο παίκτες.

   Δεδομένου ότι η ευριστική συνάρτηση είναι η διαφορά μεταξύ της απόστασης νίκης του ενός και της απόστασης νίκης του άλλου, υπάρχει συνέπεια στην τιμή της ευριστικής σχετικά με την κατάσταση του παιχνιδιού.

2. Εξαντλητικός χώρος αναζήτησης

   Ο χώρος αναζήτησης είναι εξαντλητικός καθώς ο γράφος περιέχει όλες τις πιθανές καταστάσεις και η αναζήτηση είναι αναζήτηση bfs που σημαίνει ότι όλοι οι προσπελάσιμοι κόμβοι θα εξερευνηθούν.

3. Στρατηγική ισοπαλίας

   Ο αλγόριθμος μας λαμβάνει υπόψιν το ότι άμα ένας παίκτης δεν μπορεί να κερδίσει πρέπει να θέσει ως στόχο την ισοπαλία και όχι τη νίκη (Minimax 2).

Από τα παραπάνω και από την ορθότητα του αλγορίθμου Minimax προκύπτει άμεσα ότι και ο δικός μας αλγόριθμος οδηγεί στη βέλτιστη λύση.

4 Άσκηση 4

4.1 (α)

4.1.1 Απόδειξη ύπαρξης ακμής

Αρχικά, θα αποδείξουμε την ύπαρξη της ακμής $e^{'}$. Έστω ότι η ακμή e συνδέει δύο κορυφές a, b. Αν βγάλουμε την ακμή e από το spanning tree T_1 τότε τα a, b ανήκουν πλέον σε δύο ξένες συνεκτικές συνιστώσες, ας τις ονομάσουμε G_1, G_2. Στο spanning tree T_2 οι συνεκτικές συνιστώσες αυτές συνδέονται με μια ακμή $e^{'}$. Αν $e^{'} \in T_1$ τότε θα υπάρχουν δύο μονοπάτια από το a στο b, ένα απευθείας μέσω της ακμής e και ένα μέσω ενδιάμεσων κορυφών που συνδέουν τελικά τις δύο συνεκτικές συνιστώσες G_1, G_2 μέσω της ακμής $e^{'}$. Όμως, σε κάθε δέντρο, δύο οποιεσδήποτε κορυφές συνδέονται με μοναδικό μονοπάτι. Άρα, οδηγούμαστε σε άτοπο, δηλαδή $e^{'} \notin T_1$.

4.1.2 Απόδειξη συνεκτικότητας

Αφού αποδείξαμε την ύπαρξη της ακμής $e^{'}$ μπορούμε με ευθεία απόδειξη να δείξουμε ότι το νέο δέντρο $T_1 \setminus \{e\} \cup \{e^{'}\}$ είναι συνεκτικό. Πιο ειδικά, η ακμή $e^{'}$ όπως εξηγήσαμε συνδέει τις συνεκτικές συνιστώσες G_1, G_2 που αποσυνδέθηκαν λόγω της αφαίρεσης της ακμής e. Η προσθήκη μιας ακμής που συνδέει δύο ασύνδετες συνεκτικές συνιστώσες δεν μπορεί να προκαλέσει κύκλο καθώς δεν υπάρχει τρόπος επιστροφής από την G_2 στην G_1 πέρα από την ακμη που προστέθηκε. Άρα το προκύπτον γράφημα είναι και συνεκτικό και ακυκλικό (και προφανώς μένει δέντρο), άρα έχουμε ένα συνδετικό δέντρο.

4.1.3 Αλγόριθμος εύρεσης μιας τέτοιας ακμής

Ο τρόπος να βρούμε μια τέτοια ακμή είναι αρκετά προφανής. Αρχικά, αφαιρόντας την ακμή e εξηγήσαμε ότι το έχουμε δύο ξένες συνεκτικές συνιστώσες G_1, G_2. Για κάθε μία από τις κορυφές του G_1 εξετάζουμε αν υπάρχει ακμή προς το G_2 στο γράφο G. Όταν βρούμε μια τέτοια ακμή διαφορετική της e τότε βρήκαμε την $e^{'}$ που ψάχναμε.

4.1.4 Πολυπλοκότητα αλγορίθμου

Ο αλγόριθμος που περιγράψαμε απλά προσπελαύνει ακμές και κορυφές, που σημαίνει ότι είναι γραμμικός, δηλαδή έχει πολυπλοκότητα: $O(V + E)$

4.2 (β)

4.2.1 Συνεκτικότητα γραφήματος H

Το ερώτημα (α) μας έδωσε έναν μηχανισμό παραγωγής συνδετικών δέντρων. Συγκεκριμένα, μας έδωσε έναν τρόπο να ξεκινήσουμε από ένα συνδετικό δέντρο T_1 και αλλάζοντας μία ακμή τη φορά να πάμε σε ένα συνδετικό δέντρο T_2.

Θεώρημα 4 *Όλα τα συνδετικά δέντρα έχουν τον ίδιο αριθμό ακμών*

Απόδειξη θεωρήματος
Κάθε δέντρο $|V|$ κόρυφών έχει $|V-1|$ ακμές. Κάθε συνδετικό δέντρο έχει όλες τις κορυφές (αφού είναι συνδετικό), άρα έχει V κορυφές και συνεπώς $|V-1|$ ακμές.

Αφού όλα τα συνδετικά δέντρα έχουν τον ίδιο αριθμό ακμών, χρησιμοποιώντας τον μηχανισμό του (α) μπορούμε ανταλλάσσοντας ακμές να πάμε από οποιοδήποτε συνδετικό δέντρο σε οποιοσδήποτε συνδετικό δέντρο. Κάθε φορά ανταλλάσσουμε μια ακμή και συνεπώς πάμε σε κάποιον γείτονα μας στο γράφημα H. Άρα, το γράφημα H είναι συνεκτικό.

4.2.2 Μήκος συντομότερου μονοπατιού στο H

Εξηγήσαμε ότι είναι εφικτό να πάμε από ένα συνδετικό δέντρο T_1 σε ένα οποιοδήποτε συνδετικό δέντρο T_2 αλλάζοντας 1 ακμή τη φορά. Στο γράφημα H οι μεταβάσεις είναι όλες μεταβάσεις 1 ακμής διαφοράς. Άρα, το να καταφέρουμε να πάμε από ένα συνδετικό δέντρο T_1 στο T_2, αν το T_1 έχει k διαφορετικές ακμές από το συνδετικό δέντρο T_2, αλλάζοντας 1 ακμή τη φορά είναι το συντομότερο μονοπάτι που μπορούμε να βρούμε στο γράφο H και δείξαμε ότι με το μηχανισμό του (α) ένα τέτοιο μονοπάτι υπάρχει. Άρα, το συντομότερο μονοπάτι μεταξύ T_1, T_2 έχει μήκος:

$$|T_1 \setminus T_2|$$

4.2.3 Αλγόριθμος εύρεσης συντομότερου μονοπατιού στο H

Για να βρούμε τον συντομότερο μονοπάτι στο H χρησιμοποιούμε τον αλγόριθμο που μας δίνει το βήμα (α). Πιο συγκεκριμένα, βρίσκουμε k του T_1 που δεν υπάρχουν στο T_2 και αντίστοιχα k ακμές του T_2 που δεν υπάρχουν στο T_1. Σε χρόνο $O(E+V)$ φτιάχνουμε το συνδετικό δέντρο του (α), δηλαδή αφαιρούμε μια ακμή από τις k που δεν υπάρχει στο T_2 και υπάρχει στο T_1 και την αντικαθιστούμε με μια ακμή από τις άλλες k που υπάρχει στο T_2 και δεν υπάρχει στο T_1. Έτσι, προκύπτει ένα δέντρο $T_1^{'}$. Συνεχίζουμε με τις υπόλοιπες $k-1$ ακμές την ίδια διαδικασία για το προκύπτον δέντρο επαναληπτικά μέχρι να φτάσουμε στο T_2. Σε κάθε βήμα, μεταβαίνουμε στο δέντρο του H που ταιριάζει με το προκύπτον συνεκτικό δέντρο.

4.3 (γ)

4.4 Διατύπωση αλγορίθμου

Η λύση μας είναι απλή: φτιάχνουμε δύο συνεκτικά δέντρα T_1, T_2 όπου το T_1 έχει όσο περισσότερες ακμές γίνεται από το E_1 και το T_2 έχει όσες περισσότερες ακμές γίνεται από το E_2. Η λύση μας βασίζεται στον αλγόριθμο union-find.

Πιο ειδικά, για το πρώτο δέντρο ξεκινάμε από έναν γράφο χωρίς ακμές και με κορυφές τις V. Στη συνέχεια, ιεραρχούμε τις ακμές, βάζοντας πρώτα τις ακμές του συνόλου E_1 και στη συνέχεια τις ακμές του συνόλου E_2. Για κάθε μία ακμή του συνόλου εξετάζουμε (operation find) αν σχηματίζει κύκλο. Αν δεν σχηματίζει κύκλο, την προσθέτουμε στο γράφημα μας (operation union). Επαναλαμβάνουμε μέχρι να σχηματίσουμε συνεκτικό δέντρο T_1.

Στη συνέχεια, ιεραρχούμε τις ακμές με ανάποδη σειρά και τρέχουμε πάλι τον ίδιο αλγόριθμο για να σχηματίσουμε δέντρο T_2.

Είναι φανερό ότι το T_1 έχει όσο πιο πολλές E_1 ακμές γίνεται ενώ το T_2 όσο πιο πολλές E_2 ακμές γίνεται. Έστω ότι το T_1 έχει k_1 ακμές του συνόλου E_1 και το T_2 έχει k_2 ακμές του συνόλου E_1. Διακρίνουμε τις ακόλουθες περιπτώσεις:

1. $$k_2 > k$$

   Τότε το πρόβλημα μας δεν λύνεται.

2. $$k_1 < k$$

   Επίσης το πρόβλημα μας δεν λύνεται.

3. $$k_2 < k < k_1$$

   Στην περίπτωση αυτή το πρόβλημα μας λύνεται προσπαθώντας να προσεγγίσουμε το δέντρο T_2 από το δέντρο T_1 ανταλλάσοντας 1 ακμή τη φορά μέχρι να φτάσουμε σε συνεκτικό δέντρο k ακμών στο σύνολο E_1, με τον αλγόριθμο που είδαμε στο ερώτημα (α).

4.4.1 Πολυπλοκότητα

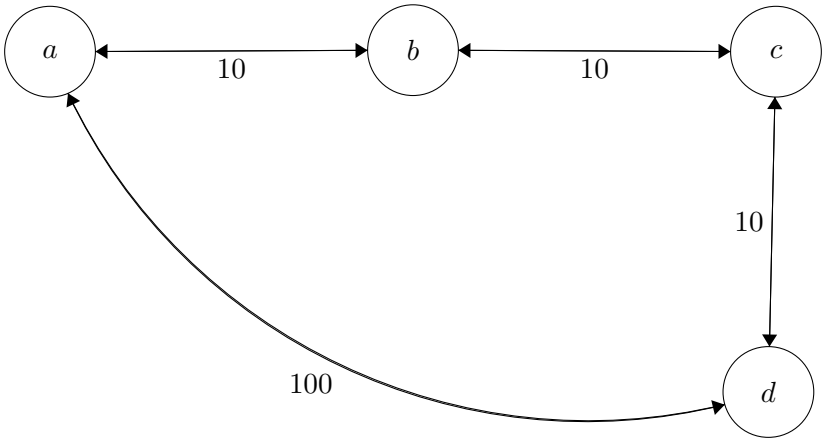
Αν υλοποιήσουμε το union find με weighted quick union και path compression όπως περιγράφεται στο [6] για την κατασκευή των δύο δέντρων θέλουμε πολυπλοκότητα: $O(E + Vlog^*E)$. Στη συνέχεια, για να πάμε από το ένα δέντρο στο άλλο θέλουμε πολυπλοκότητα χειρότερης περίπτωσης $O\left(k \cdot (V + E)\right)$ αφού το πολύ k αλλαγές θα γίνουν μέχρι να φτάσουμε στο ιδανικό δέντρο. Έτσι, η συνολική πολυπλοκότητα είναι:

$$O\left(k \cdot (V + E) + E + Vlog^*E\right)$$

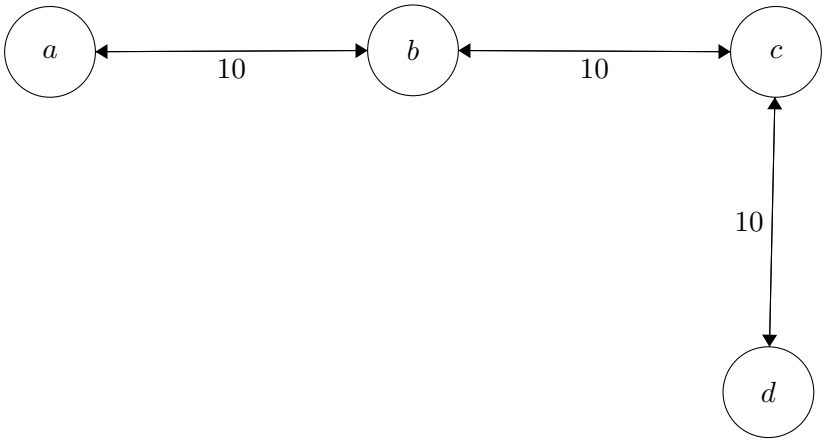
5 Άσκηση 5

5.1 (α)

Έστω το γράφημα:



Έχει μοναδικό ΕΣΔ:



Άρα, το αντίστροφο δεν ισχύει.

5.2 (β)

5.2.1 Απόδειξη

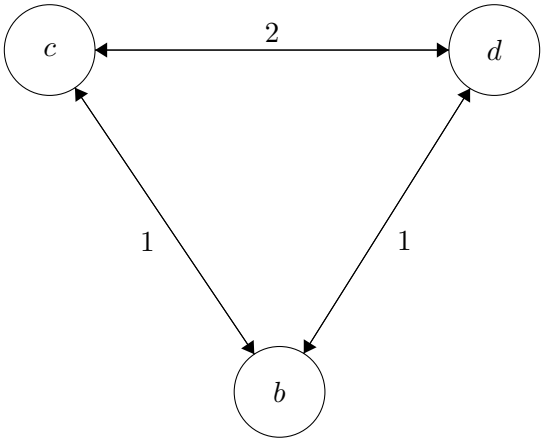
Θα δούλεψουμε με επαγωγή σε άτοπο. Έστω ότι για ένα γράφημα G που για κάθε τομή η ακμή ελάχιστου βάρους είναι μοναδική έχει δύο ελάχιστα συνδετικά δέντρα. Ας υποθέσουμε, χωρίς βλάβη της γενικότητας ότι τα συνδετικά αυτά δέντρα διαφέρουν μόνο σε μία ακμή, ας πούμε την ακμή $e = (u, v)$. Αν τα συνδετικά δέντρα απέχουν περισσότερες ακμές, μπορούμε να ανάξουμε το πρόβλημα στη διαφορά μίας ακμής χρησιμοποιώντας το μηχανισμό παραγωγής ελάχιστων συνδετικών δέντρων που εξηγήσαμε στην άσκηση 4 στα ερωτήματα (α), (β).

Αφαιρούμε από το T_1 την ακμή e. Έτσι, το συνδετικό δέντρο γίνεται δάσος και σπάει σε δύο συνιστώσες G_1, G_2. Θεωρούμε την τομή $(G_1, G - G_1)$ του γράφου. Οι ακμές που μας πηγαίνουν από τη μία συνιστώσα της τομής στην άλλη έχουν διάφορα βάρη w. Παρόλα αυτά, η ακμή ελάχιστου βάρους είναι μοναδική από την υπόθεση μας για το γράφο G. Άρα, οποιαδήποτε ακμή και αν επιλέξουμε εκτός της e θα οδηγηθούμε σε συνδετικό δέντρο μεγαλύτερου συνολικού κόστους, άρα άτοπο.

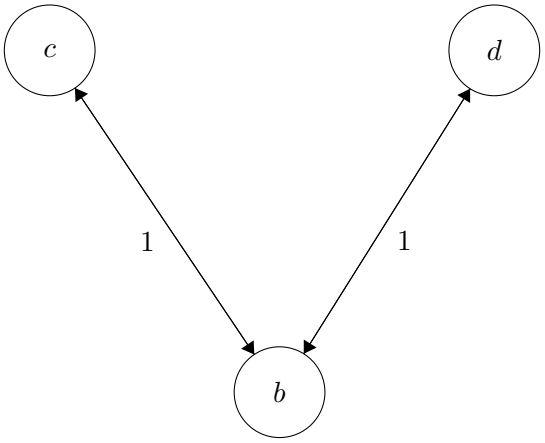
Από τα παραπάνω προκύπτει πως αν για κάθε τομή του γράφου G η ακμή ελάχιστου βάρους είναι μοναδική, τότε ο γράφος έχει μοναδικό ΕΣΔ.

5.2.2 Αντιπαράδειγμα

Έστω ο ακόλουθος γράφος:



Ο γράφος αυτός μπορεί να χωριστεί σε δύο ξένες συνεκτικές συνιστώσες: $\{c, d\}, \{b\}$ με τις ακμές της τομής να είναι οι : $(b, c), (b, d)$. Οι ακμές αυτές έχουν ίδιο βάρος, παρόλα αυτά το ελάχιστο συνδετικό δέντρο είναι μοναδικό:



5.3 (γ)

Η ικανή και αναγκαία συνθήκη που προτείνουμε είναι η ακόλουθη:

Θεώρημα 5 *Έστω ένας γράφος G και $T_1, T_2, ..., T_n$ το σύνολο όλων των δυνατών συνδετικών δέντρων του γράφου. Αν $\exists i : Cost(T_i) < Cost(T_j) \forall j \neq i$ τότε το ελάχιστο συνδετικό δέντρο είναι μοναδικό.*

5.3.1 Ικανή συνθήκη

Αν υπάρχει μοναδικό συνδετικό δέντρο με ελάχιστο κόστος τότε από ορισμό το ελάχιστο συνδετικό δέντρο είναι μοναδικό. Συνεπώς, η συνθήκη που διατυπώσαμε είναι ικανή.

5.3.2 Αναγκαία συνθήκη

Έστω ότι $\exists T_i, T_j : Cost(T_i) = Cost(T_j)$. Και τα δύο δέντρα είναι συνδετικά δέντρα του γράφου G και αφού έχουν ίδιο κόστος είναι και τα δύο ελάχιστα συνδετικά δέντρα, άτοπο. Συνεπώς, η συνθήκη που διατυπώσαμε είναι αναγκαία.

5.4 (δ)

5.5 Διατύπωση αλγορίθμου

Αρχικά, φτιάχνουμε ένα συνδετικό δέντρο με τον αλγόριθμο του Kruskal. Στη συνέχεια, τρέχουμε ξανά τον αλγόριθμο του Kruskal και στις ισοπαλίες δοκιμάζουμε τελευταίες τις ακμές που βάλαμε στο συνδετικό δέντρο που κατασκευάσαμε προηγμένως. Αν οδηγηθούμε στο ίδιο συνδετικό δέντρο τότε το συνδετικό δέντρο είναι μοναδικό.

5.5.1 Ορθότητα αλγορίθμου

Η ορθότητα του αλγορίθμου μας βασίζεται στην ικανότητα του αλγορίθμου Kruskal να βρεί όλα τα συνδετικά δέντρα ανάλογα με το πως χειρίζεται τις ισοπαλίες. Βάζοντας τελευταίες ανάμεσα στις ισοδύναμες ακμές τη δεύτερη φορά τις ακμές που βάλαμε στο συνδετικό δέντρο της πρώτης φοράς, πρακτικά, δοκιμάζουμε να φτιάξουμε κάθε άλλο δυνατό ελάχιστο συνδετικό δέντρο πριν φτιάξουμε το ελάχιστο συνδετικό δέντρο της πρώτης φοράς. Αν δεν υπάρχει άλλο δυνατό συνδετικό δέντρο, τότε φτιάχνουμε τελικά το συνδετικό δέντρο της πρώτης φοράς.

Αυτή η ιδιότητα του αλγορίθμου του Kruskal μας εξασφαλίζει με ευθεία απόδειξη την ορθότητα του αλγορίθμου μας.

5.6 Πολυπλοκότητα

Η πολυπλοκότητα του αλγορίθμου μας είναι η πολυπλοκότητα του αλγορίθμου Kruskal, δηλαδή $O(ElogE)$.

6 Βιβλιογραφία

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), Introduction to Algorithms
[2] Dial, Robert B. (1969). "Algorithm 360: Shortest-path forest with topological ordering", Communications of the ACM.
[3] Beineke, Lowell W.; Wilson, Robin J. (2009), Topics in topological graph theory, Encyclopedia of Mathematics and its Applications, 128, Cambridge University Press, Cambridge, p. 36
[4] Gross, Jonathan L.; Yellen, Jay (2005), Graph Theory and Its Applications (2nd ed.)
[5] Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall
[6] Princeton Union Find Presentation