Εθνικο Μετσοβιο Πολυτεχνειο

3η Γραπτή Άσκηση

Αλγόριθμοι & Πολυπλοκότητα

Σπουδαστής:
Διονύσης Ζηνδρος (06601)
<dionyziz@gmail.com>

Διδάσκοντες:
Στάθης Ζαχος
Δημήτρης Φωτακης

24 Ιανουαρίου 2011

Άσκηση 1

Το πρόβλημα μοντελοποιείται αν το αντιπροσωπεύσουμε με έναν γράφο στον οποίο κάθε ταινία είναι ένα κόμβος και κάθε συνδρομητής είναι μία ακμή ανάμεσα στις δύο ταινίες που θέλει να δει. Στη συνέχεια αρκεί να χρωματίσουμε το γράφο με δύο χρώματα έτσι ώστε καμία ακμή να μην πατάει στο ίδιο χρώμα δύο φορές· δηλαδή μένει να ελέγξουμε αν ο γράφος μας είναι διμερής.

Για να ελέγξουμε αν ο γράφος είναι διμερής, χρωματίζουμε μία κορυβή αυθαίρετα και στη συνέχεια για κάθε χρωματισμένη κορυφή χρωματίζουμε τους γείτονές της με το αντίθετο χρώμα έως ότου ολοκληρωθεί η διαδικασία ή καταλήξουμε σε αντίφαση. Αυτό πρέπει να το επαναλάβουμε για κάθε συνεκτική συνιστώσα του γράφου μας.

Λήμμα. *Ο αλγόριθμος ελέγχου διμερούς γράφου είναι ορθός.*

Απόδειξη. Η ορθότητα του αλγορίθμου προκύπτει ως εξής. Ο αρχικός κόμβος κάθε συνιστώσας σε έναν έγκυρο διμερή διαχωρισμό, αν υπάρχει, μπορεί να χρωματιστεί αυθαίρετα. Στη συνέχεια, αν ο γράφος μας είναι πραγματικά διμερής, η αναζήτηση στη συγκεκριμένη συνιστώσα θα πρέπει να χρωματίσει με σωστό τρόπο όλους τους κόμβους χωρίς να αποτύχει. Άρα αν ο γράφος είναι πραγματικά διμερής, τότε ο χρωματισμός θα πρέπει να είναι επιτυχής. Αντίστροφα, αν έχουμε επιτυχή χρωματισμό, και επειδή κάθε χρώμα είναι αμετάκλητο, τότε θα έχουμε ελέγξει όλες τις ακμές για αντιφάσεις και καμία δεν θα έχει βρεθεί. Συνεπώς ο γράφος θα είναι διμερής. Αυτό ισχύει για κάθε συνεκτική συνιστώστα του γράφου και έτσι ολοκληρώνει την απόδειξη. □

Algorithm 1 Άσκηση 1

```
1:  procedure BIPARTITE(V, E)
2:      unexplored ← V
3:      for v ∈ V do
4:          color[v] ← ∅
5:      while unexplored ≠ ∅ do
6:          u ← an item of unexplored
7:          unexplored ← unexplored \ {u}
8:          q ← EMPTYSTACK()
9:          PUSH(q, u)
10:         color[u] ← ⊤
11:         while q ≠ ∅ do
12:             s ← POP(q)
13:             unexplored ← unexplored \ {s}
14:             paint ← ¬color[s]
15:             for neighbour t of s do
16:                 if color[t] ≠ ∅ then
17:                     if color[t] ≠ paint then
18:                         return ⊥
19:                 else
20:                     color[t] ← paint
21:                     PUSH(q, t)
22:     return ⊤
```

Ο χρόνος εκτέλεσης του αλγορίθμου θα είναι $O(|V|+|E|)$. Αυτό προκύπτει από το γεγονός ότι διασχίζουμε όλες τις ακμές του γράφου από το πολύ δύο φορές, μία για κάθε άκρη τους και άρα έχουμε $O(|E|)$, ενώ για κάθε συνιστώσα θα πρέπει να ξεκινήσουμε από κάποιον κόμβο της το οποίο κοστίζει $O(|V|)$.

Άσκηση 2

Διασχίζουμε το γράφο χρησιμοποιώντας αναζήτηση κατά πλάτος και διατηρούμε κάθε φορά το πλήθος των τρόπων με τους οποίους φτάσαμε σε κάθε κόμβο.

Algorithm 2 Άσκηση 2

```
1: procedure ShortestCount(V, E, s, t)
2:     for v ∈ V do
3:         visited[v] ← 0
4:         distance[v] ← ∞
5:     visited[s] ← 1
6:     distance[s] ← 0
7:     q ← EmptyQueue()
8:     Push(q, s)
9:     while ¬Empty(q) do
10:        u ← Pop(q)
11:        if u = t then
12:            return visited[t]
13:        for neighbour v of u do
14:            if distance[v] = ∞ then
15:                Push(q, u)
16:            if distance[u] + 1 ≤ distance[v] then
17:                distance[v] ← distance[u] + 1
18:                visited[v] ← visited[v] + visited[u]
```

Η πολυπλοκότητα του αλγορίθμου είναι η ίδια με αυτή της αναζήτησης κατά πλάτος, δηλαδή $O(|E|)$.

Άσκηση 3

(α)

Ο γράφος της άσκησης 4 με $L = \{C\}$ είναι ένα παράδειγμα γράφου που έχει διαφορετικό ΕΣΔ υπό περιορισμούς.

(β)

Για κάθε κόμβο για τον οποίο υπάρχει απαίτηση να είναι φύλλο κρατάμε την φθηνότερη ακμή που το συνδέει με κάποιον κόμβο που δεν υπάρχει απαίτηση να είναι φύλλο. Στη συνέχεια βρίσκουμε το ΕΣΔ κατά τα γνωστά.

Algorithm 3 Άσκηση 3

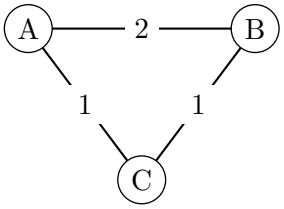
```
1: procedure ConstraintedMST(V, E, L)
2:     if |V| ≤ 2 then
3:         return E
4:     for v ∈ V do
5:         adj[v] ← ∅
6:         best[v] ← ∞
7:     for e = (u, v, w) ∈ E do
8:         if u ∈ L ⊕ v ∈ L then
9:             for s ∈ {u, v} do
10:                if s ∈ L then
11:                    if w < best[s] then
12:                        best[s] ← w
13:                        adj[s] ← {e}
14:        else
15:            adj[v] ← adj[v] ∪ {e}
16:    return Prim(V, adj)
```

Η πολυπλοκότητα του αλγορίθμου προκύπτει ως εξής. Για κάθε ακμή ενημερώνουμε την βέλτιστη ακμή που είναι γνωστή για κάποιο φύλλο, αν συνδέεται με κάποιο, σε σταθερό χρόνο. Ο συνολικός χρόνο γι΄ αυτό απαιτεί να περάσουμε από όλες τις ακμές σε συνολικό χρόνο $O(|E|)$. Ο αλγόριθμος του Πριμ τρέχει σε $O(|E| + |V| \log |V|)$ που είναι και άνω φράγμα για όλο τον αλγόριθμο.

Άσκηση 4

(α)

Ο ακόλουθος γράφος έχει μοναδικό ΕΣΔ, αλλά περιέχει ακμές ίδιου βάρους:



(β)

Ο γράφος του υποερωτήματος (α) έχει μοναδικό ΕΣΔ αλλά περιέχει ακμές ίδιου βάρους που είναι οι ελάχιστες που διασχίζουν την ίδια τομή.

Λήμμα. *Κάθε ακμή ενός ΕΣΔ ενός συνεκτικού μη κατευθυνόμενου ζυγισμένου γράφου είναι η ελάχιστη που διασχίζει κάποιο κόψιμο.*

Απόδειξη. Έστω τέτοιος γράφος $G = (V, E, w)$ με ένα ΕΣΔ του $T \subseteq E$ και έστω ότι υπάρχει κάποια ακμή $e = (u, v) \in T$ τέτοια ώστε να μην υπάρχει κόψιμο του Γ το οποίο να είναι η ελάχιστη που το διασχίζει. Τότε έστω K το ΕΣΔ που κατασκευάζει ο αλγόριθμος του Kruskal. Κατά την εκτέλεση του αλγορίθμου, υπάρχει κάποιο βήμα κατά το οποίο γίνεται ένωση των ανεξάρτητων συνόλων που περιέχουν τους κόμβους u και v μέσω κάποιας ακμής f η οποία είναι ελάχιστη σε ένα κόψιμο το οποίο διασχίζει και η e και άρα $w(f) < w(e)$. Όμως στο T η ακμή e μπορεί να αντικατασταθεί από την f και το δέντρο να παραμείνει συνεκτικό και με βάρος $w(T \cup \{f\} \setminus \{e\}) < w(T)$. Άρα το T δεν είναι ΕΣΔ, κάτι το οποίο αποτελεί αντίφαση. □

Λήμμα. *Κάθε συνεκτικός μη κατευθυνόμενος ζυγισμένος γράφος στον οποίο για κάθε τομή η ακμή ελάχιστου βάρους που τη διασχίζει είναι μοναδική έχει μοναδικό ΕΣΔ.*

Απόδειξη. Έστω τέτοιος γράφος $G = (V, E, w)$ και δύο ΕΣΔ του, $T \subseteq E$ και $T' \subseteq E$, με $T \neq T'$ και $w(T) = w(T')$.

Τότε, επειδή $T \neq T$, θα υπάρχει κάποια ακμή $e = (u, v)$ με $e \in T \wedge e \notin T'$. Από το παραπάνω λήμμα, η e θα είναι η ελάχιστη που διασχίζει κάποιο κόψιμο του T, έστω S.

Τώρα, επειδή το T' είναι συνεκτικό δέντρο, θα υπάρχει μοναδικό μονοπάτι από τον κόμβο u στον κόμβο v που διασχίζει το S με κάποια ακμή, έστω $e' \neq e$. Αφού $w(e) < w(e')$, αντικαθιστώντας την ακμή e' με την e στο T' παίρνουμε ένα συνεκτικό δέντρο με βάρος $w(T' \cup \{e\} \setminus \{e'\}) < w(T')$. Άρα το T' δεν είναι ελάχιστο συνεκτικό δέντρο, κάτι το οποίο αποτελεί αντίφαση.

Άρα το ΕΣΔ είναι μοναδικό. □

(γ)

Λήμμα. *Το ΕΣΔ ενός συνεκτικού ζυγισμένου μη κατευθυνόμενου γράφου είναι μοναδικό αν κάθε ακμή του είναι η μοναδική ελάχιστη σε κάποιο κόψιμο του γράφου.*

Απόδειξη. Έστω $G = (V, E, w)$ ένας τέτοιος γράφος και έστω K ένα ΕΣΔ του.

Θα δείξουμε ότι αν κάθε ακμή του K είναι η μοναδική ελάχιστη σε κάποιο κόψιμο, τότε το δέντρο είναι μοναδικό. Πράγματι, έστω ότι υπάρχει ΕΣΔ $T \neq K$. Τότε θα υπάρχει ακμή $e = (u, v) \in K \wedge e \notin T$. Από την υπόθεση, υπάρχει κόψιμο S το οποίο η e διασχίζει ως μοναδική ελάχιστη. Επειδή το T είναι συνδετικό, θα υπάρχει μοναδικό μονοπάτι που συνδέει τους κόμβους u και v το οποίο διασχίζει το κόψιμο S χρησιμοποιώντας κάποια ακμή $f \neq e$. Επειδή η e είναι ελάχιστη, θα είναι $w(e) < w(f)$. Αντικαθιστώντας την f με την e στο T προκύπτει ΕΣΔ με $w(T \cup \{e\} \setminus \{f\}) < w(T)$ κάτι το οποίο αποτελεί αντίφαση. Άρα το K είναι μοναδικό.

Αντίστροφα, έστω ότι το K είναι μοναδικό. Τότε ο αλγόριθμος του Kruskal θα δώσει το K. Έστω, τώρα, ότι υπάρχει κάποια ακμή $e \in K$ που δεν είναι ελάχιστη σε κανένα κόψιμο του G. Τότε ο αλγόριθμος δεν θα μπορούσε να την έχει επιλέξει ποτέ, και άρα $e \notin K$. Αυτό αποτελεί αντίφαση, και άρα όλες οι ακμές είναι ελάχιστες σε κάποιο κόψιμο. □

(δ)

Ξεκινάμε υπολογίζοντας ένα ΕΣΔ του γράφου. Στη συνέχεια ελέγχουμε αν υπάρχει εναλλακτικό ΕΣΔ εργαζομενοι ως εξής. Κατασκευάζουμε έναν πίνακα που δείχνει τη μέγιστη ακμή του μονοπατιού ανάμεσα σε οποιουσδήποτε δύο κόμβους του ΕΣΔ. Τέλος, ελέγχουμε για κάθε ακμή του αρχικού γράφου (u, v) αν μπορεί να αντικαταστήσει κάποια ακμή του ΕΣΔ και αυτό να παραμείνει ελάχιστο. Αυτό γίνεται πλέον εύκολα ελέγχοντας την τιμή της μέγιστης ακμής στο μονοπάτι από το u στο v πάνω στον αρχικό γράφο.

Η ορθότητα του αλγορίθμου προκύπτει από το παρακάτω λήμμα.

Λήμμα. *Ένα ΕΣΔ T είναι μοναδικό αν $\forall (u, v) \in E \setminus T : w(u, v) > \max(p(u, v))$ όπου $p(u, v)$ το μονοπάτι από την κορυφή u στην κορυφή v στο δέντρο T.*

Algorithm 4 Άσκηση 4

```
1:  procedure UniqueMST(V, E, w)
2:      T ← Prim(V, E, w)
3:      for (u, v) ∈ V² do
4:          dist[u][v] ← ∞
5:      for v ∈ V do
6:          q ← EmptyStack()
7:          Push(q, v)
8:          while ¬Empty(q) do
9:              u ← Pop(q)
10:             for neighbour r of u in T do
11:                 if dist[v][u] = ∞ then
12:                     dist[v][u] ← max(dist[v][r], w(r, u))
13:                     Push(q, u)
14:     for (u, v) ∈ E \ T do
15:         if w(u, v) = dist[u][v] then
16:             return ⊥
17:     return ⊤
```

Ο αλγόριθμος του Prim τρέχει σε $O(|V|^2)$. Η κατασκευή του πίνακα μέγιστων ακμών ανά μονοπάτι μπορεί να γίνει σε χρόνο $O(|V|^2)$ χρησιμοποιώντας

αναζήτηση κατά πλάτος πάνω στο δέντρο. Η αναζήτηση ακμής αντικατάστασης γίνεται στη συνέχεια σε $O(|E|)$. Έτσι ο αλγόριθμος που προκύπτει είναι $O(|V|^2)$.

Άσκηση 5

(α)

Λήμμα. *Σε ένα συνεκτικό κατευθυνόμενο ζυγισμένο γράφο, για κάθε κύκλο υπάρχει ΕΣΔ που δεν περιέχει την ακμή μέγιστου βάρους του κύκλου αυτού.*

Απόδειξη. Έστω τέτοιος γράφος $G = (V, E, w)$ με κάποιον κύκλο C και έστω ένα ΕΣΔ $T \subseteq E$ που περιέχει την ακμή e μέγιστου βάρους του C. Τότε έστω το δάσος $T \setminus \{e\}$ που θα αποτελείται από δύο συνδεδεμένους υπογράφους. Θα υπάρχει μία ακμή $f \in E$ που θα συνδέει αυτούς τους δύο συνδεδεμένους υπογράφους με $f \neq e$ και άρα $w(f) \leq w(e)$. Αντικαθιστώντας την e με την f στο T προκύπτει ένα συνεκτικό δέντρο με βάρος $w(T \cup \{f\} \setminus \{e\}) \leq w(T)$. Άρα το $T \cup \{f\} \setminus \{e\}$ είναι ΕΣΔ. □

(β)

Λήμμα. *Ο αντίστροφος αλγόριθμος του Kruskal είναι ορθός.*

Απόδειξη. Σε κάθε βήμα i, ο αλγόριθμος διατηρεί τον ελάχιστου κόστους συνδετικό υπόγραφο G_i με ακριβώς $j(i) = m - i + 1$ ακμές. Πράγματι, στο 1ο βήμα, ο υπόγραφος είναι ίσος με τον αρχικό γράφο και άρα συνδετικός και ελάχιστος. Έστω ότι ο υπόγραφος είναι συνδετικός και ελάχιστος στο i-οστό βήμα. Τότε έστω $e = (u, v)$ η επόμενη ακμή που θα επιλεγεί.

Στην περίπτωση που η e είναι γέφυρα του G_i ανάμεσα σε δύο συνδεδεμένα τμήματα C_1 και C_2, δηλαδή διασχίζει το κόψιμο $S = (C_1, C_2)$ του αρχικού γράφου, η ακμή αυτή δε θα λείπει από κανέναν ελάχιστο συνδετικό υπογράφο G_k με $k > i$. Αυτό φαίνεται ως εξής. Έστω ότι η e έλειπε από τον G_k· τότε αυτός θα περιείχε κάποια ακμή $f \neq e$ που διασχίζει το S για να είναι συνδεδεμένος. Όμως θα είναι $w(f) > w(e)$ διότι, λόγω άπληστης ιδιότητας, η f είχε αφαιρεθεί πριν το i-οστό βήμα αφού η e ήταν γέφυρα του G_i. Αντικαθιστώντας την f με την e λαμβάνουμε ένα συνδετικό υπόγραφο ίδιου πλήθος ακμών με βάρος $w(G_k \cup \{e\} \setminus \{f\}) < w(G_k)$. Άρα το G_k δεν θα ήταν ελάχιστο χωρίς την ύπαρξη της e. Συνεπώς μπορούμε με ασφάλεια να διατηρήσουμε την e στον υπογράφο και να προχωρήσουμε στην επόμενη ακμή στο ίδιο βήμα.

Στην περίπτωση που η e δεν είναι γέφυρα του G_i τότε ανήκει σε ένα κύκλο του G_i. Εργαζόμενοι παρόμοια όπως στο παραπάνω θεώρημα, είναι εμφανές ότι η e δεν ανήκει στον υπόγραφο G_{i+1}. Την αφαιρούμε, λοιπόν, και προχωράμε στο επόμενο βήμα.

Χρησιμοποιώντας, τώρα, τη μαθηματική επαγωγή, έχουμε αποδείξει την αμετάβλητη ιδιότητα της επανάληψης. Τέλος, ο ελάχιστος συνδετικός υπόγραφος με ακριβώς $n-1$ είναι ΕΣΔ. □

(γ)

Algorithm 5 Άσκηση 5

```
1: procedure ReverseKruskal(V, E, w)
2:     E ← sort E decreasingly by weight
3:     S ← E
4:     for e ∈ E do
5:         if ¬Bridge(S, e) then
6:             S ← S \ {e}
7:     return S
```

Η υλοποίηση του $\textsc{Bridge}(S, e)$ ελέγχει αν η e είναι γέφυρα του S και μπορεί να γίνει με διάφορους τρόπους. Ο απλούστερος τρόπος θα ήταν ένας αλγόριθμος αναζήτησης κατά βάθος ή κατά πλάτος που ελέγχει αν υπάρχει και άλλο μονοπάτι ανάμεσα στις δύο κορυφές που συνδέει η ακμή. Τότε η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(E^2)$. Αντικαθιστώντας τον έλεγχο για γέφυρες με κάποια πιο αποδοτική υλοποίηση (βλ. [1]) μπορούμε να πετύχουμε την αισθητά καλύτερη πολυπλοκότητα $O(E \log E (\log \log E)^3)$.

Βιβλιογραφία

[1] Mikkel Thorup, *Near-optimal fully-dynamic graph connectivity.* Addison Wesley, Massachusetts, Proc. 32nd ACM Symposium on Theory of Computing, 2000.