Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή ΗΜ&ΜΥ
Αλγόριθμοι και Πολυπλοκότητα
7ο εξάμηνο, Ροή Λ
Ακαδημαϊκή περίοδος: 2011-2012

4η Σειρά Γραπτών Ασκήσεων

Γερακάρης Βασίλης
<vgerak@gmail.com>
Α.Μ.: 03108092

20 Φεβρουαρίου 2012

1 Παιχνίδι Επιλογής Ακμών σε Κατευθυνόμενο Ακυκλικό Γράφημα

Αρχικά, φτιάχνουμε την τοπολογική διάταξη του DAG με χρήση DFS και δίνουμε στους κόμβους αρχική τιμή 0, σε χρόνο $O(|V| + |E|)$. *(0 = Lose condition για τον A, 1 = win condition)*. Αρχίζοντας από το τέλος του DAG και πηγαίνοντας προς τα πίσω ελέγχουμε τους κόμβους.

- Αν όλες οι εξερχόμενες ακμές τους είναι προς κόμβους με τιμή 1, τότε ο κόμβος είναι lose condition για τον A, οπότε σημειώνεται με 0.
- Αλλιώς, υπάρχει κόμβος που οδηγεί σε lose condition (0) για τον B, άρα ο κόμβος αυτός είναι νικηφόρος για τον A.

Υπάρχει νικηφόρα στρατηγική για τον A, αν ο κόμβος έναρξης είναι νικηφόρος μετά την εκτέλεση του αλγορίθμου.

Algorithm 1 DAG traversing game

```
1: procedure Traverse((G(V, E), s))
2:     Make topological sorting of G
3:     for all u ∈ G do
4:         C[u] ← 0.
5:     for all u ∈ G, starting from the end -> start do
6:         for all e[u, v] ∈ AdjList(u) do
7:             if C[v] = 0 then
8:                 C[u] ← 1
9:     if C[s] = 1 then
10:        return Win
11:    else
12:        return Lose
```

Η πολυπλοκότητα του αλγορίθμου είναι $O(|V| + |E|)$ αφού τόσο το DFS όσο και το σώμα του αλγορίθμου εξετάζουν από μία φορά κάθε κόμβο και ακμή.

2 Σχεδιασμός Ταξιδιού (DPV 4.13)

i) Μία τέτοια διαδρομή είναι εφικτό να βρεθεί (ή να αποδειχτεί η μη ύπαρξή της) σε γραμμικό χρόνο. Κάνοντας ένα BFS στο γράφημα απορρίπτωντας κάθε ακμή η οποία έχει βάρος $w > L$ βρίσκουμε μία τέτοια διαδρομή, αν υπάρχει σε χρόνο $O(|E| + |V|)$.

ii) Για να υπολογίσουμε την ελάχιστη αυτονομία θα χρησιμοποιήσουμε μια τροποποιημένη μορφή του αλγορίθμου του Dikjstra. Όπως ο Dijkstra κρατάει ώς δεδομένο την ελάχιστη απόσταση ώς ένα κόμβο και "αναπτύσσει" τον κόμβο με τη λιγότερη απόσταση, εδώ θα κρατάμε ώς δεδομένο την ελάχιστη αυτονομία καυσίμου (το ελάχιστο από τα μέγιστα βάρη ακμών) για να φτάσουμε σε ένα κόμβο και θα συνεχίζουμε την αναζήτηση από τον κόμβο με τη μικρότερη τιμή. Δηλαδή:

Algorithm 2 Minimum tank capacity

```
1: procedure minCapacity((G(V, E, w)s, t))
2:     for all u ∈ V do
3:         D[u] ← ∞
4:     D[s] ← 0; S ← ∅
5:     while |S| < |V| do
6:         u ∉ S : D[u] = min_{v∉S}{D[v]}
7:         S ← S ∪ {u}
8:     for all v ∈ AdjList[u] do
9:         if D[v] > max(D[u], w[u, v]) then
10:            D[v] ← max(D[u], w[u, v])
11:    return D[t]
```

Ο αλγόριθμος αυτός έχει πολυπλοκότητα $O(|E|log|V|)$ (με binary heaps) ή $O(|E| + |V|log|V|)$ (με fibonacci heaps), όσο η εκτέλεση του αλγορίθμου του Dijkstra

3 Διαχωρισμός Γραφήματος

i) Έστω ότι δεν είναι το συντομότερο μονοπάτι. Θα υπάρχει δηλαδή μια ακμή $e' \notin T$ με $w(e') < w(e)$, που ακουμπάει στον κόμβο u. Αυτό είναι άτοπο, αφού ο αλγόριθμος του Kruskal (που ταξινομεί τις ακμές) θα την είχε επιλέξει για το MST προτού επεξεργαστεί την e που θεωρούμε πως ανήκει στο MST.

ii) Υπολογίζουμε το MST T με έναν από τους γνωστούς αλγορίθμους (σε ελάχιστο χρόνο $O(|E|+|V|log|V|)$. Στη συνέχεια βρίσκουμε σε γραμμικό χρόνο τη μεγαλύτερου βάρους ακμή και την αφαιρούμε από το T. Οι κόμβοι στις 2 συνεκτικές συνιστώσες που προκύπτουν ανήκουν στα υποσύνολα S_1, S_2 της επιθυμητής διαμέρισης.

Το βέλτιστο της επιλογής μας προκύπτει λογικα: Οι μεγαλύτερου βάρους ακμές που δεν είναι αναγκαίες για τη συνεκτικότητα του δέντρου αφαιρούνται από τη διαδικασία εύρεσης του MST. Σκοπός μας είναι να χωρίσουμε το γράφο σε 2 συνιστώσες, μεγιστοποιώντας την απόσταση μεταξύ των συνιστωσών. Το ελάχιστο κόστος που μπορούμε λοιπόν να "πληρώσουμε" είναι όσο μια ακμή που ανήκει στο MST, και για να μεγιστοποιήσουμε αυτή την ποσότητα, επιλέγουμε τη μέγιστη.

4 Παιχνίδια εξουσίας

FILL ME 4!

5 Αναγωγές και NP-Πληρότητα

5.1 Dense subgraph

Dense fill!

5.2 Μακρύ μονοπάτι

Μακρύ indeed!

5.3 Feedback Vertex Set - Directed

directed!

5.4 Feedback Vertex Set - Undirected

undirected!