ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΠΛ231: Δομές Δεδομένων και Αλγόριθμοι

Εαρινό Εξάμηνο 2013

Φροντιστήριο 8 - ΛΥΣΕΙΣ

Άσκηση 1

Ένας γράφος ονομάζεται k-χρωματίσιμος αν είναι δυνατό να χρωματίσουμε τους κόμβους του με k διαφορετικά χρώματα έτσι ώστε κανένα ζεύγος από γειτονικούς κόμβους να μην έχει το ίδιο χρώμα. Να προτείνετε αλγόριθμο ο οποίος με δεδομένο εισόδου κάποιο γράφο G = (V,E) αποφασίζει κατά πόσο ο γράφος είναι 2-χρωματίσιμος σε χρόνο O(|V| + |E|).

Μας δίνεται ένας γράφος G και θέλουμε να αποφασίσουμε κατά πόσο είναι 2-χρωματίσιμος. Για να το πετύχουμε, επιχειρούμε χρωματισμό του γράφου ως εξής: Ξεκινούμε με κάποιο τυχαίο κόμβο και τον χρωματίζουμε με το χρώμα 1. Στη συνέχεια παίρνουμε όλους τους γείτονες του και τους χρωματίζουμε με το χρώμα 2. Συνεχίζουμε με τους γείτονες αυτών και τους χρωματίζουμε με το χρώμα 1, και ούτω καθεξής.
Αν συναντήσουμε κάποιο κόμβο ο οποίος είναι ήδη χρωματισμένος υπάρχουν δύο περιπτώσεις: αν είναι χρωματισμένος με το χρώμα που θα θέλαμε να τον χρωματίσουμε, τότε προχωρούμε κανονικά, αν όμως είναι χρωματισμένος με το άλλο χρώμα, συμπεραίνουμε ότι ο γράφος μας δεν είναι 2-χρωματίσιμος και τερματίζουμε την εκτέλεση της διαδικασίας.
Η διαδικασία αυτή είναι επέκταση της διαδικασίας κατά-πλάτος διερεύνησης σε γράφους:

```
2-colorable(graph G, vertex v){
      Queue Q=new Queue();
      for each w in G
            color[w] = 0;
      color[v] = 1;
      Q.enqueue(v);
      while (!Q.isEmpty()){
            w = Q.dequeue();
            for each u adjacent to w
                  if (color[u] != 0)
                        if (color[w] == color[u])
                              print “THE GRAPH IS NOT 2-COLORABLE
                  else
                        if (color[w] == 1)
                              color[u] = 2;
                        else
                              color[u] = 1;
                        Q.enqueue(u);
}
```

Άσκηση 2

Ο Φαραώ της Αιγύπτου Alghamon IV σας έχει προσλάβει για να δουλέψετε στο κτίσιμο της πυραμίδας που θα χρησιμοποιηθεί για την ταφή του. Η πυραμίδα θα αποτελείται από n μεγάλες πέτρες s_1, s_2, ..., s_n. Οι αρχιτέκτονες της πυραμίδας έχουν υπολογίσει μία σχέση $\angle$ όπου, για δύο πέτρες s_i, s_j ισχύει $s_i \angle s_j$ αν και μόνο αν η πέτρα s_i πρέπει να τοποθετηθεί πριν από την πέτρα s_j. Επίσης, για κάθε πέτρα s γνωρίζουμε τον χρόνο t(s) που απαιτείται για τοποθέτηση της στη σωστή θέση στην πυραμίδα.

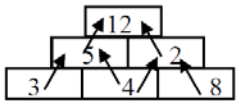
A. Να σχεδιάσετε αποδοτικό αλγόριθμο ο οποίος με δεδομένο εισόδου μια πέτρα, να υπολογίζει τον ελάχιστο χρόνο που χρειάζεται για την τοποθέτησή της, υποθέτοντας ότι πέτρες μπορούν να τοποθετηθούν στη θέση τους παράλληλα, εφόσον οι περιορισμοί της σχέσης $\angle$ δεν παραβιάζονται.
(**Υπόδειξη**: Ο ελάχιστος χρόνος τοποθέτησης για την πέτρα i είναι το μήκος του μεγαλύτερου κατευθυνόμενου μονοπατιού από την αρχή μέχρι το i.)

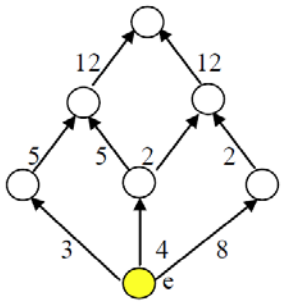
B. Να σχεδιάσετε αποδοτικό αλγόριθμο ο οποίος με δεδομένο εισόδου μια πέτρα να υπολογίζει τον μέγιστο χρόνο στον οποίο πρέπει να ολοκληρωθεί η τοποθέτηση της πέτρας έτσι ώστε η ολοκλήρωση της πυραμίδας να μην καθυστερήσει. Υποθέτουμε ότι οι πέτρες μπορούν να τοποθετηθούν στη θέση τους παράλληλα, εφόσον οι περιορισμοί της σχέσης $\angle$ δεν παραβιάζονται.
(**Υπόδειξη**: Ο μέγιστος χρόνος τοποθέτησης για την πέτρα i + το μήκος του μεγαλύτερου κατευθυνόμενου μονοπατιού από το i μέχρι το τέλος δεν πρέπει να ξεπερνά το συνολικό χρόνο αποπεράτωσης του έργου. Ο μέγιστος χρόνος τοποθέτησης για την τελευταία πέτρα με τι πρέπει να ισούται;)

Α.

Το πρόβλημα μπορεί να μεταφραστεί σε πρόβλημα εύρεσης του μονοπατιού με το μέγιστο κόστος σε γράφο: Θεωρήστε τον γράφο με βάρη G=(V,E) όπου $V=\{s_1,s_2,...,s_n\}$, δηλαδή υπάρχει μία κορυφή για κάθε πέτρα και η επιπλέον κορυφή e (η γή), $E=\{(s_i,s_j)|$ αν $s_i \angle s_j\} \cup \{(e,s_i) |$ αν δεν ισχύει $s_i \angle s_j$ για κανένα j}, και βάρη w(u,v) = t(v).

Για παράδειγμα,





Ο ελάχιστος χρόνος που χρειάζεται για την ολοκλήρωση της πυραμίδας, (υποθέτοντας ότι πέτρες μπορούν να τοποθετηθούν στη θέση τους παράλληλα) είναι ίσος με το μέγιστο κόστος μονοπατιού από την κορυφή e στον πιο πάνω γράφο.

Για το i ισχύει τα ακόλουθα:

- earliestEventTime(0) = 0
- earliestEventTime(i) = max {earliestEventTime(j) + length(j,i)}, (j,i) ϵ E,
    - E είναι το σύνολο των ακμών
    - length(j,i) είναι ο χρόνος που χρειάζεται για την δραστηριότητα (j,i)

Για να το υπολογίσουμε, χρησιμοποιούμε την πιο κάτω αναδρομική συνάρτηση:

```
int earliestEventTime(graph GRAPH [][], int vertex) {
      int i;
      int max=0, val;
      for (i=0; i<N; i++) {
            if (GRAPH[i][vertex] != 0) {
                  // filter out the max
                  val = earliestEventTime(i) + GRAPH[i][vertex];
                  if (val>max)
                        max = val;
            }
      }
      return max;
}
```

Β.

Έστω ότι latestEventTime(i) είναι ο πιο αργός χρόνος για τον οποίο το γεγονός i μπορεί να συμβεί. Εάν το γεγονός i δεν συμβεί μέχρι τον χρόνο αυτό, τότε το έργο δεν μπορεί να ολοκληρωθεί σε χρόνο ίσο με το μήκος έργου. Άρα το latestEventTime του τελικού γεγονότος είναι ίσο με το earliestEventTime του τελικού γεγονότος. Για το i ισχύει το ακόλουθο:

- latestEventTime(n-1) = earliestEventTime(n-1)
- latestEventTime(i) = min {latestEventTime(j) - length(i,j)}, (i,j) ϵ E,
    - E είναι το σύνολο των ακμών
    - length(j,i) είναι ο χρόνος που χρειάζεται για την δραστηριότητα (j,i)

Για να το υπολογίσουμε, χρησιμοποιούμε την πιο κάτω αναδρομική συνάρτηση:

```
int latestEventTime(graph GRAPH [][], int vertex) {
      int j;
      // in order to recognize the final vertex
      int min = INT_MAX;
      int val;
      for (j=0; j<N; j++) {
            if (GRAPH[vertex][j] != 0) {
                  val = latestEventTime(j) - GRAPH[vertex][j];
                  if (val<min)
                        min = val;
            }
      }
      if (min == INT_MAX)
            return earliestEventTime(vertex);
      return min;
}
```