# Δομές Δεδομένων και Αλγόριθμοι

Χρήστος Γκόγκος

ΤΕΙ Ηπείρου

Χειμερινό Εξάμηνο 2014-2015
Παρουσίαση 18. Dijkstra's Shortest Path Algorithm

# Ο αλγόριθμος εύρεσης της συντομότερης διαδρομής του Dijkstra

- Ο αλγόριθμος δέχεται ως είσοδο ένα γράφημα $G = (V, E)$ (κατευθυνόμενο ή μη) και μια κορυφή του γραφήματος $s$ η οποία αποτελεί την αφετηρία.
- Υπολογίζει για όλες τις κορυφές $v \in V$ το μήκος του συντομότερου μονοπατιού από το $s$ στο $v$.

**Προϋπόθεση λειτουργίας του αλγορίθμου**

Κάθε ακμή θα πρέπει να έχει μη αρνητικό βάρος. Αν το γράφημα περιέχει ακμές με αρνητικό βάρος τότε μπορεί να χρησιμοποιηθεί ο αλγόριθμος των Bellman-Ford.

Αρχικά βρίσκεται η πλησιέστερη κορυφή προς την αφετηρία $s$ και στη συνέχεια η δεύτερη πλησιέστερη, η τρίτη πλησιέστερη κ.ο.κ.

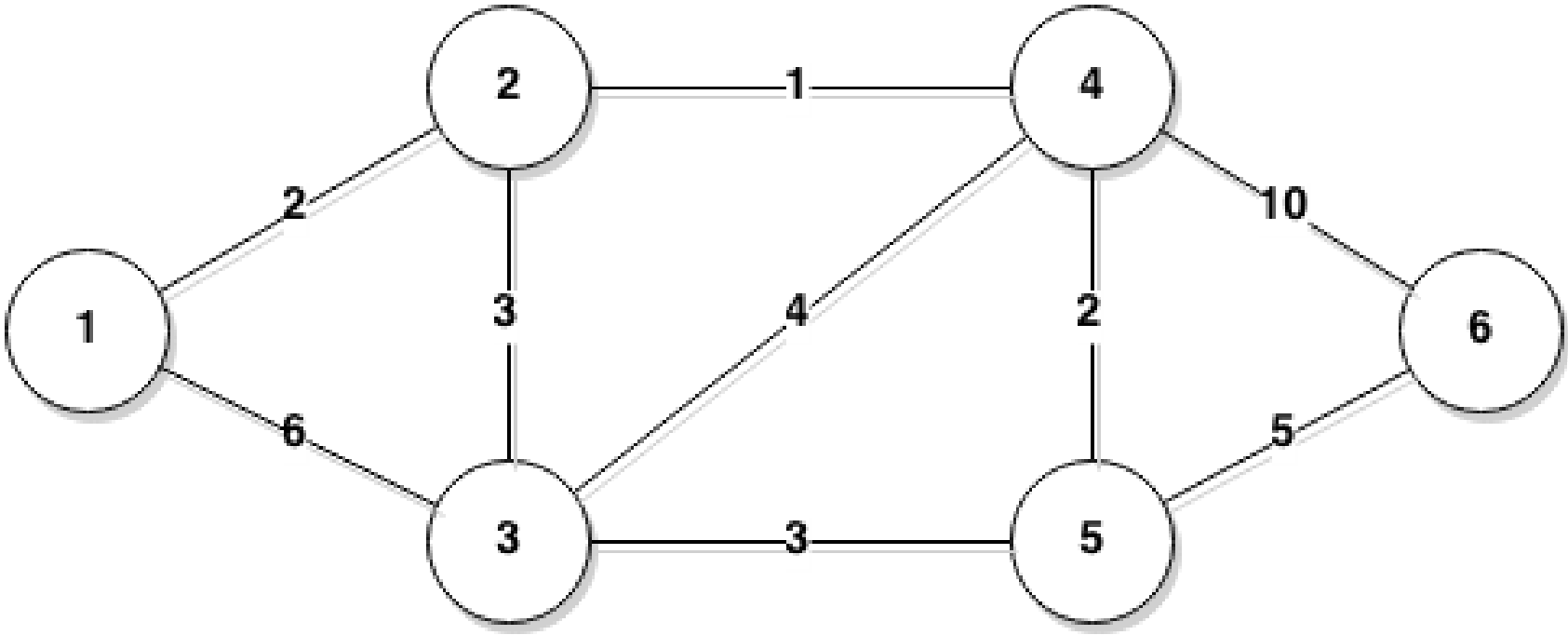
# Περιγραφή του αλγορίθμου

1. Ο αλγόριθμος εντοπίζει τις συντομότερες διαδρομές προς τις κορυφές του γραφήματος σε σειρά απόστασης από την κορυφή αφετηρία.
2. Σε κάθε βήμα του αλγορίθμου η αφετηρία και οι ακμές προς τις κορυφές για τις οποίες έχει ήδη βρεθεί συντομότερο μονοπάτι σχηματίζουν το υποδένδρο $S$ του γραφήματος.
3. Οι κορυφές που είναι προσπελάσιμες με 1 ακμή από το υποδένδρο $S$ είναι υποψήφιες να αποτελέσουν την επόμενη κορυφή που θα εισέλθει στο υποδένδρο. Επιλέγεται μεταξύ τους η κορυφή που βρίσκεται στη μικρότερη απόσταση από την αφετηρία.
4. Για κάθε υποψήφια κορυφή $u$ υπολογίζεται το άθροισμα της απόστασής της από την πλησιέστερη κορυφή $v$ του δένδρου συν το μήκος της συντομότερης διαδρομής από την αφετηρία $s$ προς την κορυφή $v$. Στη συνέχεια επιλέγεται η κορυφή με το μικρότερο άθροισμα.
5. Όταν επιλεγεί η κορυφή που πρόκειται να προστεθεί στο δένδρο τότε προστίθεται στο σύνολο των κορυφών που απαρτίζουν το υποδένδρο $S$ και για κάθε μία από τις υποψήφιες κορυφές που συνδέονται με μια ακμή με την κορυφή που επιλέχθηκε ενημερώνεται η απόστασή της από το υποδένδρο εφόσον προκύψει μικρότερη τιμή.

# Ψευδοκώδικας

- Το σύνολο $S$ περιέχει τις κορυφές για τις οποίες έχει προσδιοριστεί η συντομότερη διαδρομή από την κορυφή $s$ ενώ το διάνυσμα $d$ περιέχει τις αποστάσεις από την κορυφή $s$.
- Αρχικά $S = \{s\}, d_s = 0$ και για όλες τις κορυφές $i \neq s, d_i = +\infty$
- Μέχρι να γίνει $S = V$
  - Εντοπισμός του στοιχείου $v \notin S$ με τη **μικρότερη τιμή** $d_v$ και προσθήκη του στο $S$
  - Για κάθε ακμή από την κορυφή $v$ στη κορυφή $u$ με βάρος $w$ ενημερώνεται η τιμή $d_u$ έτσι ώστε $d_u = min(d_u, d_v + w)$

# Παράδειγμα εκτέλεσης του αλγορίθμου




## Δεδομένα γραφήματος

1 2,2 6,3
2 2,1 3,3 1,4
3 6,1 3,2 4,4 3,5
4 1,2 4,3 2,5 10,6
5 3,3 2,4 5,6
6 10,4 5,5

# Μια απλή υλοποίηση του αλγορίθμου σε C++

```
void compute_shortest_paths_to_all_vertices(list<pair<int, int>>*
      g, int V,
        int source, int* distance) {
     list <int> S; set<int> V_S;
    for (int i = 1; i <= V; i++)
        if (i == source)
            {S.push_back(i); distance[i] = 0;}
        else
            {V_S.insert(i); distance[i] = INT_MAX;}
    while (!V_S.empty()) {
        int min = INT_MAX, pmin = -1;
        for (int v1 : S) {
            for (pair<int, int> w_v : g[v1]) {
                int weight = w_v.first, v2 = w_v.second;
                bool is_in_V_S = V_S.find(v2) != V_S.end();
                if ((is_in_V_S) && (distance[v1] + weight < min)) {
                    min = distance[v1] + weight; pmin = v2;
                }
            }
        }
        distance[pmin]=min; S.push_back(pmin); V_S.erase(pmin);
    }
}
```

# Η απόδοση του αλγορίθμου

Η ταχύτητα εκτέλεσης του αλγορίθμου εξαρτάται από τις δομές δεδομένων που χρησιμοποιούνται για να αναπαρασταθεί το γράφημα και από τη χρήση ουράς προτεραιότητας έτσι ώστε να επιταχυνθούν οι διαγραφές στοιχείων από τις κορυφές που δεν έχουν εισαχθεί στο δένδρο ακόμη
Πρόκειται για ένα εξαιρετικά γρήγορο αλγόριθμο (εφόσον υλοποιηθεί κατάλληλα) με πολυπλοκότητα χειρότερης περίπτωσης $O(|E| \log |V|)$.