Διάλεξη 25: Βραχύτερα Μονοπάτια σε Γράφους

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Βραχύτερα Μονοπάτια σε γράφους

Ο αλγόριθμος ***Dijkstra*** *για εύρεση της* ***βραχύτερης απόστασης***

Ο αλγόριθμος ***Dijkstra*** *για εύρεση του* ***βραχύτερου μονοπατιού & απόστασης***

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

Βραχύτερα Μονοπάτια σε Γράφους

- Με δεδομένο ένα κατευθυνόμενο γράφο με βάρη G=(V,E), θέλουμε να βρούμε τα μονοπάτια με το ελάχιστο δυνατό βάρος από κάποιο κόμβο Α προς οποιονδήποτε κόμβο Χ.
- **Ορισμός:** ***Βραχύτερο μονοπάτι*** μεταξύ ενός συνόλου από μονοπάτια είναι το μονοπάτι με το ελάχιστο βάρος.
- Υπενθύμιση: Το βάρος w(p) ενός μονοπατιού p δίνεται ως εξής:

$$\begin{array}{ll} \text{Αν} & p = v_o \rightarrow v_1 \rightarrow \ldots \rightarrow v_k \\ \text{τότε} & w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \end{array}$$

- Σε αυτή την διάλεξη θα δούμε ακόμα ένα άπληστο αλγόριθμο (greedy algorithm) του Δανού **Edsger Dijkstra** (1930-2002) για την επίλυση αυτού του προβλήματος.
- Ένας τέτοιος αλγόριθμος έχει πολλές εφαρμογές (π.χ. εύρεση μικρότερης διαδρομής σε ένα οδικό δίκτυο, σε δίκτυα υπολογιστών κτλ).

Η δομή της βέλτιστης λύσης

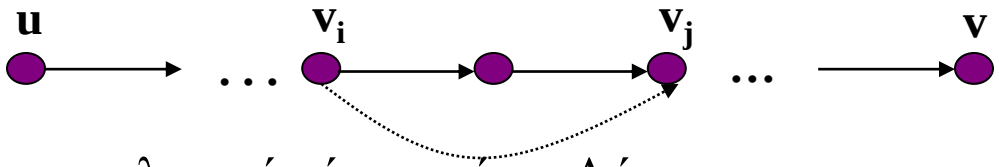
Λήμμα 1

Έστω κατευθυνόμενος γράφος με βάρη G=(V,E) και έστω

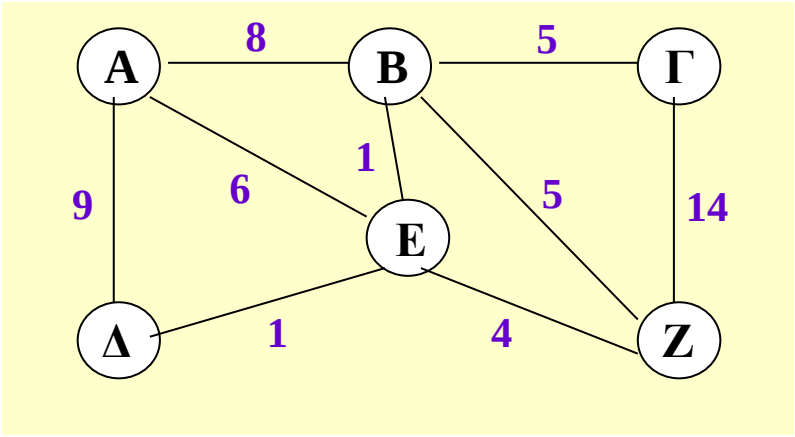
$$p = u \to v_1 \to \ldots \to v_k \to v$$

το βραχύτερο μονοπάτι μεταξύ των κόμβων u και v.

Τότε κάθε υπο-μονοπάτι του p είναι βραχύτερο.

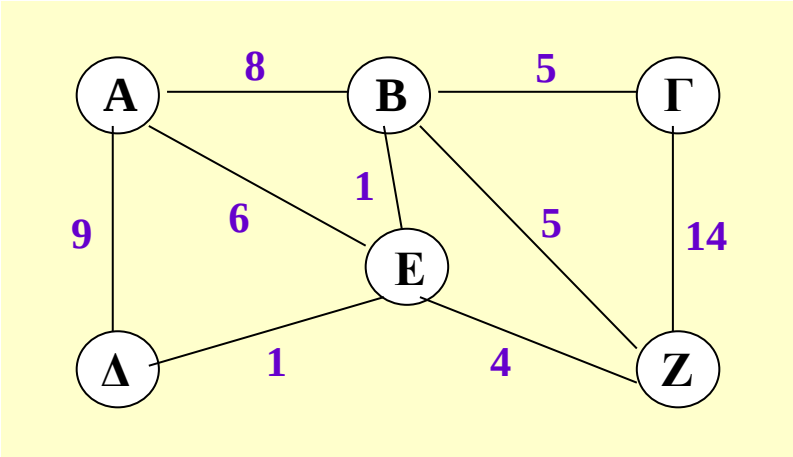


- Ας δούμε την λογική πίσω από το Λήμμα.



Ανασκόπηση του Αλγόριθμου Dijkstra

- Έστω ότι θέλουμε να βρούμε το/α **βραχύτερα μονοπάτια** από κάποιο κόμβο **Α** προς όλους τους υπόλοιπους κόμβους σε κάποιο γράφο **G(V,E)**

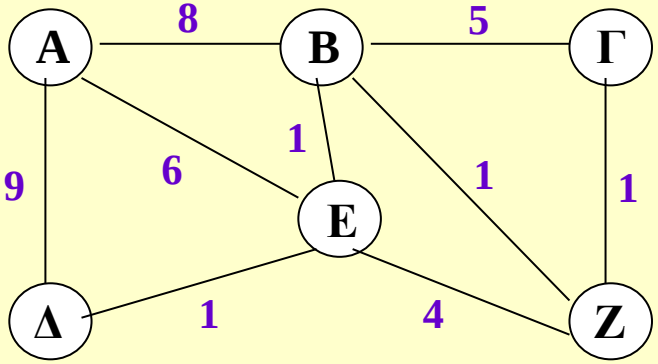


- Ποιο είναι το κόστος του βραχύτερου μονοπατιού από το **Δ** προς τον **Γ**;
- Ποιο είναι το ακριβές μονοπάτι;

- Ο αλγόριθμος του Dijkstra είναι ένας **single source shortest path algorithm** (δηλαδή αναφέρεται συγκεκριμένα σε μια κορυφή εκκίνησης) για γράφους με **μη-αρνητικά βάρη**.

Παράδειγμα Εκτέλεσης Αλγορίθμου Dijkstra

Κορυφή Εκκίνησης: Α



dist(A,E) + dist(E,B)

Πίνακας ο οποίος σημειώνει για κάθε κορυφή την πιο κοντινή (συνολική) απόσταση από τον Α

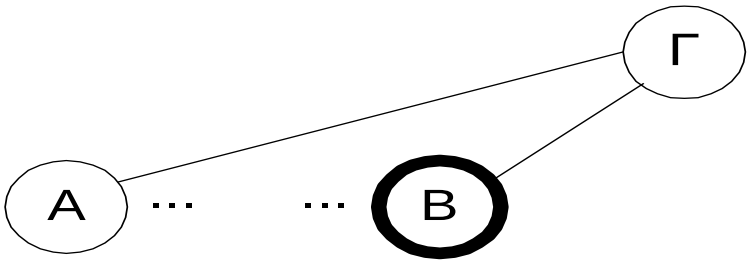
	S	d(A)	d(B)	d(Γ)	d(Δ)	d(E)	d(Z)
1	S=∅	0	∞	∞	∞	∞	∞
2	S={**A**}	0	8	∞	9	6	∞
3	S={A,**E**}	0	7	∞	7	6	10
4	S={A,E,**B**}	0	7	12	7	6	8
5	S={A,E,B,**Δ**}	0	7	12	7	6	8
6	S={A,E,B,Δ,**Z**}	0	7	9	7	6	8
7	S={A,E,B,Δ,Z, **Γ**}	0	7	9	7	6	8

μείωση

Αυτό το χρώμα δηλώνει ποιες κορυφές χρησιμοποιήθηκαν στο παρόν στάδιο.

Ανασκόπηση του Αλγόριθμου Dijkstra

- Δηλώνουμε τον **πίνακα visited[|V|],** ο οποίος αποθηκεύει τις κορυφές του G, για τις οποίες το μήκος του βραχύτερου μονοπατιού έχει υπολογισθεί (δηλαδή από τις οποίες περάσαμε ήδη).
- Επίσης διατηρεί τον **πίνακα distance[|V|]**, ο οποίος φυλάει την ανά πάσα στιγμή, την μικρότερη απόσταση οποιουδήποτε κόμβου **X**, από τον κόμβο **A** (από την οποία ξεκινήσαμε).
- Αρχικά **visited[|V|]=∅** και **distance[|V|] = ∞ .**
- Ο αλγόριθμος διαλέγει "άπληστα" την κοντινότερη κορυφή **B** (ως προς την κορυφή **A)**, που δεν έχει μέχρι στιγμής τύχει επεξεργασίας.
- Μετά ελέγχει αν για οποιοδήποτε γείτονα **Γ** της **B**, αν η χρήση της ακμής **(B,Γ)** μπορεί να δημιουργήσει βραχύτερο μονοπάτι **A-Γ**.



Υλοποίηση του Αλγόριθμου Dijkstra

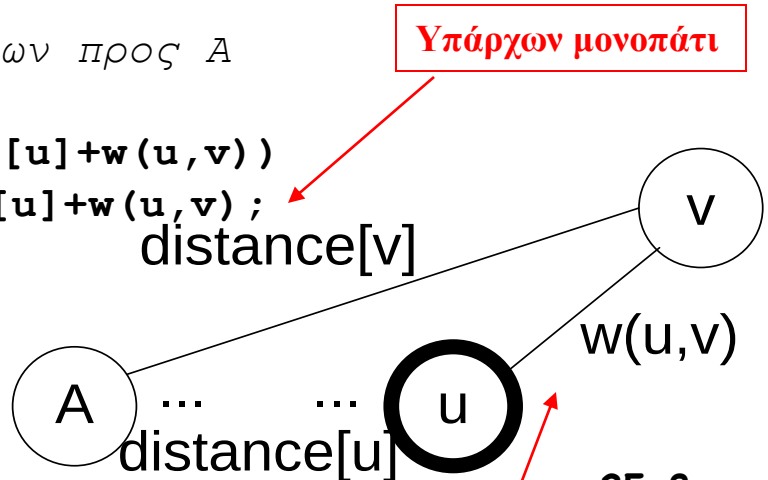
```
// A: Σημείο Εκκίνησης. G(V,E): Ο Γράφος
dijkstra( G(V,E), vertex A){

   distance[|V|]={∞}; // Απόσταση κορυφής i από κορυφή A
   visited[|V|]={};  // Πίνακας που σημειώνει από πού περάσαμε
   count = 0;  // Μεταβλητή που σημειώνει από πόσους κόμβους
   περάσαμε

   distance[A]=0; visited[A]=1; // Αρχικοποίηση σημείου εκκίνησης

   while (count < |V|){      // σε χρόνο O(|V|)
        // Προχωρούμε στον επόμενο κόμβο
        u=minVertex(V);      // Άπληστη επιλογή σε χρόνο O(|V|)

        // Ενημέρωση αποστάσεων γειτόνων προς A
        για κάθε γείτονα v του u {
            if (distance[v] > distance[u]+w(u,v))
                distance[v] = distance[u]+w(u,v);
        }
        count++;
   }
}
Συνολικός Χρόνος Εκτέλεσης: O(|V|²)
```



Η βοηθητική συνάρτηση **minVertex**

- Η βοηθητική διαδικασία minVertex επιστρέφει την κορυφή j η οποία είναι η πιο κοντινή (μεταξύ των κορυφών που δεν ανήκουν ακόμα στο μονοπάτι). Δηλαδή όμοια με την minVertex του αλγορίθμου Prim:

// visited[|V|] : Σημειώνει από ποιες κορυφές περάσαμε

// distance[|V|] : // Απόσταση κορυφής i από κορυφή A

```
vertex minVertex(int visited[], int distance[]){
   min = ∞;
   for (i=0; i<|V|; i++) {
      if (visited[i] == 1) continue;
      if (distance[i] < distance[min])
         min = i;
    }
    return min;
}
```

S	d(A)	d(B)	d(Γ)	d(Δ)	d(E)	d(Z)
S=∅	0	∞	∞	∞	∞	∞
S={A}	0	8	∞	9	6	∞
S={A,E}	0	7	∞	7	6	10
S={A,E,B}	0	7	12	7	6	8
S={A,E,B,Δ}	0	7	12	7	6	8
S={A,E,B,Δ,Z}	0	7	9	7	6	8
S={A,E,B,Δ,Z, Γ}	0	7	9	7	6	8

Η minvertex επιστρέφει το Β (θα μπορούσε να επέστρεφε και το Δ)

Με τη χρήση σωρού μπορούσε να υλοποιηθεί σε O(1)-χρόνο η ανάκτηση

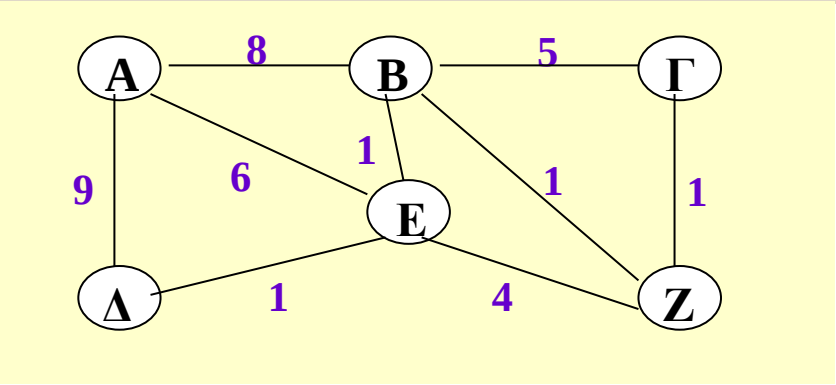
Αλγόριθμος του Dijkstra **με Εύρεση του Βραχύτερου Μονοπατιού ΒΜ (όχι μονο της απόστασης)**

- Αν εκτός από το μήκος του μονοπατιού μας ενδιαφέρει και το **ακριβές μονοπάτι (οι κόμβοι του)** τότε μπορούμε να ακολουθήσουμε την ίδια βασική ιδέα με την εξής προσθήκη

- Για κάθε κορυφή αποθηκεύουμε σε ένα **πίνακα P,** τον γείτονα που θα μας δώσει το βραχύτερο μονοπάτι προς τον κόμβο εκκίνησης.

- Σε αυτή την περίπτωση μπορούμε με τον τερματισμό του αλγόριθμου να κατασκευάσουμε από **τον πίνακα P** το **μέγιστο μονοπάτι** από τον κόμβο εκκίνησης προς κάποιον άλλο κόμβο Χ

Παράδειγμα Κτισίματος του Μονοπατιού



S	d,P(A)	d,P(B)	d,P(Γ)	d,P(Δ)	d,P(E)	d,P(Z)
S=∅	0, -	∞, -	∞, -	∞, -	∞, -	∞, -
S={A}	0, -	8, **A**	∞, -	9, **A**	6, **A**	∞, -
S={A,E}	0, -	7, E	∞, -	7, E	6, A	10,E
S={A,E,B}	0, -	7, E	12,B	7, E	6, A	8, B
S={A,E,B,Δ}	0, -	7, E	12,B	7, E	6, A	8, B
S={A,E,B,Δ,Z}	0, -	7, E	9, Z	7, E	6, A	8, B
S={A,E,B,Δ,Z, Γ}	0, -	7, **E**	9, Z	7, E	6, **A**	8, **B**

Πως μπορούμε τώρα να βρούμε το μικρότερο μονοπάτι από το Α προς Z;

Υλοποίηση του Αλγόριθμου Dijkstra

```
dijkstra( G(V,E), vertex A){

    distance[|V|]={∞}; // Απόσταση κορυφής i από κορυφή A
    visited[|V|]={};  // Πίνακας που σημειώνει από πού περάσαμε
    P[|V|]= {'-'}; // Πίνακας που σημειώνει τον γείτονα που μας προσφέρει το
                   // βραχύτερο μονοπάτι προς τον A
    count = 0;

    d[A]=0; S[A]=1; // Αρχικοποίηση σημείου εκκίνησης

    while (count < |V|){
          // Προχωρούμε στον επόμενο κόμβο
          u=minVertex(V);

          // Ενημέρωση αποστάσεων γειτόνων προς A
          για κάθε γείτονα v του u {
              if (distance[v] > distance[u]+w(u,v))
                   distance[v]=distance[u]+w(u,v);
                   P[v] = u; // Ενημέρωση πεδίου προηγούμενου που θα μας
                             // οδηγήσει προς το σημείο εκκίνησης A
          }
          count++;
    }
}
Συνολικός Χρόνος Εκτέλεσης: O(|V|²)
```