Διάλεξη 34: 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: Παναγιώτης Ανδρέου

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 A προς οποιονδήποτε κόμβο X.
- **<u>Ορισμός: *Βραχύτερο μονοπάτι*</u>** μεταξύ ενός συνόλου από μονοπάτια είναι το μονοπάτι με το ελάχιστο βάρος.
- Υπενθύμιση: Το βάρος w(p) ενός μονοπατιού p δίνεται ως εξής:

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

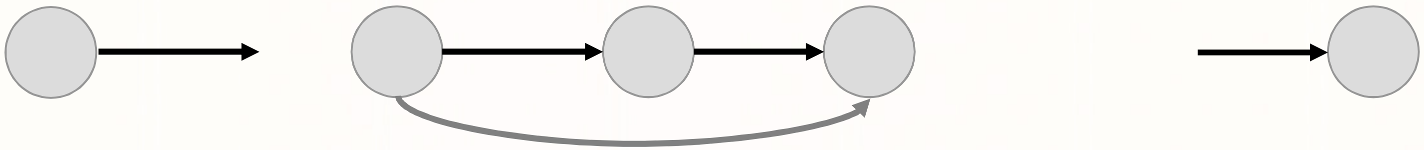
- Σε αυτή την διάλεξη θα δούμε ακόμα ένα άπληστο αλγόριθμο (greedy algorithm) του Δανού **Edsger Dijkstra** (1930-2002) για την επίλυση αυτού του προβλήματος.
- Ένας τέτοιος αλγόριθμος έχει πολλές εφαρμογές (π.χ. εύρεση μικρότερης διαδρομής σε ένα οδικό δίκτυο, σε δίκτυα υπολογιστών κτλ).

Η δομή της βέλτιστης λύσης

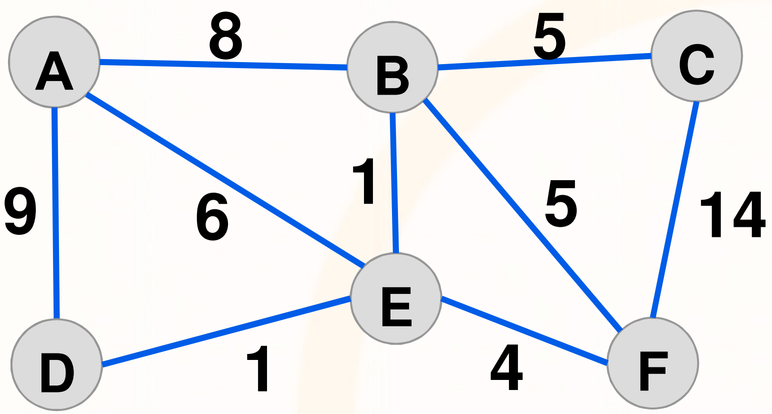
- **Λήμμα 1**
  Έστω κατευθυνόμενος γράφος με βάρη G=(V,E) και έστω

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

- το βραχύτερο μονοπάτι μεταξύ των κόμβων u και v.
  Τότε κάθε υπο-μονοπάτι του p είναι βραχύτερο.

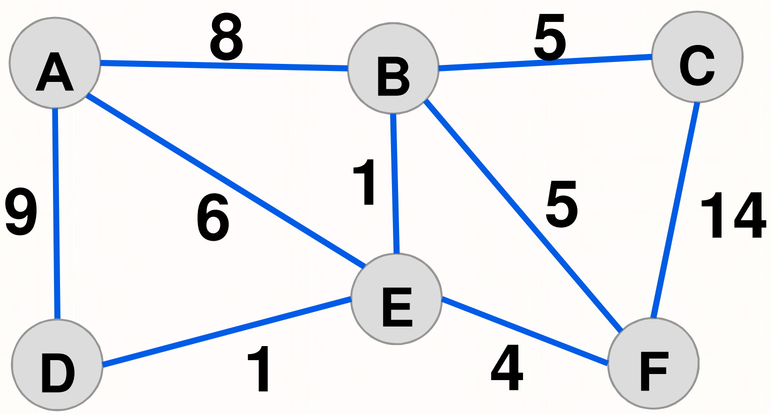


- Ας δούμε την λογική πίσω από το Λήμμα.



Ανασκόπηση του Αλγόριθμου Dijkstra

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

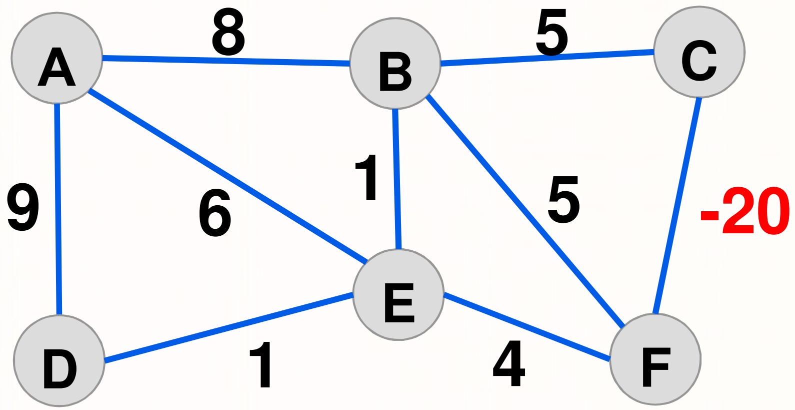


1. Ποιο είναι το κόστος του βραχύτερου μονοπατιού από **D** προς C;
2. Ποιο είναι το ακριβές μονοπάτι;

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

Αρνητικά Βάρη στον Αλγόριθμο του Dijkstra

- Σημειώστε ότι όταν υπάρχουν αρνητικά βάρη, τότε είναι δυνατό αυτά να μας θέσουν την μικρότερη απόσταση, **αναδρομικά, ίση με -∞.**



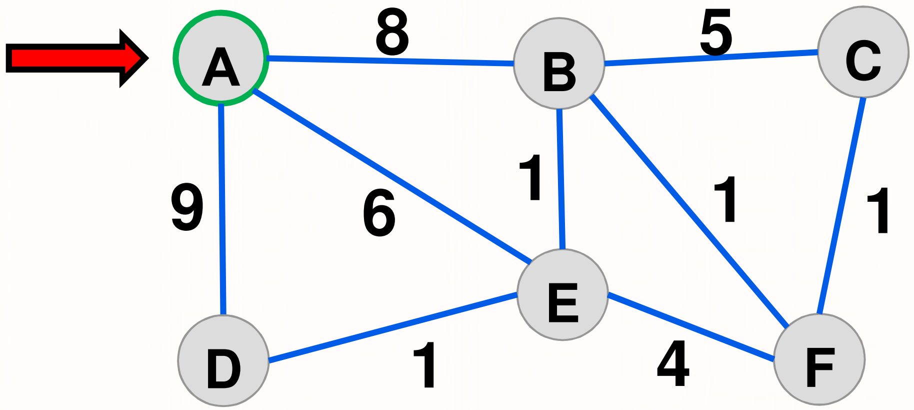
Παράδειγμα:

Ποια είναι η βραχύτερη απόσταση μεταξύ (Β,Γ);

- Πρώτη Εκτέλεση 5: B=>C
- Δεύτερη Εκτέλεση -5: B=>C=>F=>B=>C
- Τρίτη Εκτέλεση -15: B=>C=>F=>B=>C=>F=>B=>C
- .....

Παράδειγμα Εκτέλεσης Αλγορίθμου Dijkstra 1/4

- Πάρε σαν παράμετρο κάποια κορυφή, έστω η κορυφή Α



- Αρχικοποίησε ένα **πίνακα distance** και θέσε την απόσταση όλων των κορυφών από την κορυφή Α είναι ίση με -∞.

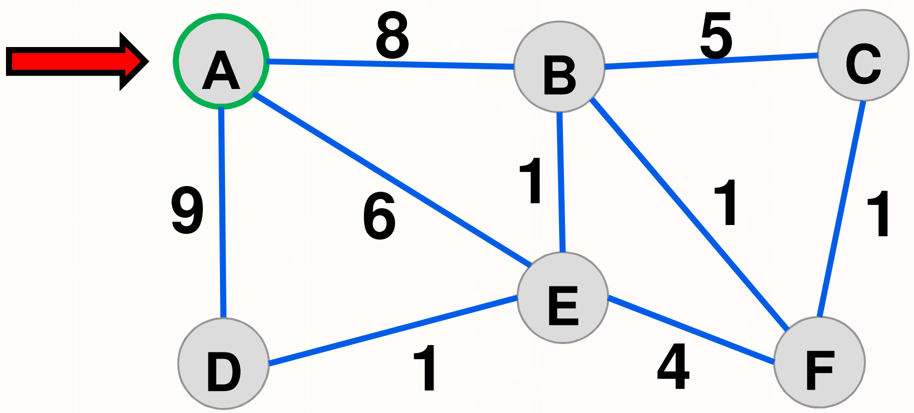
Εκτέλεση	S	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
1	S=∅	0	∞	∞	∞	∞	∞

- Αρχικοποίησε ένα **πίνακα visited** όπου θα σημειώνει **από ποιες κορυφές έχουμε περάσει ήδη**

Εκτέλεση	S	v(A)	v(B)	v(C)	v(D)	v(E)	v(F)
1	S=∅	1	0	0	0	0	0

Παράδειγμα Εκτέλεσης Αλγορίθμου Dijkstra 2/4

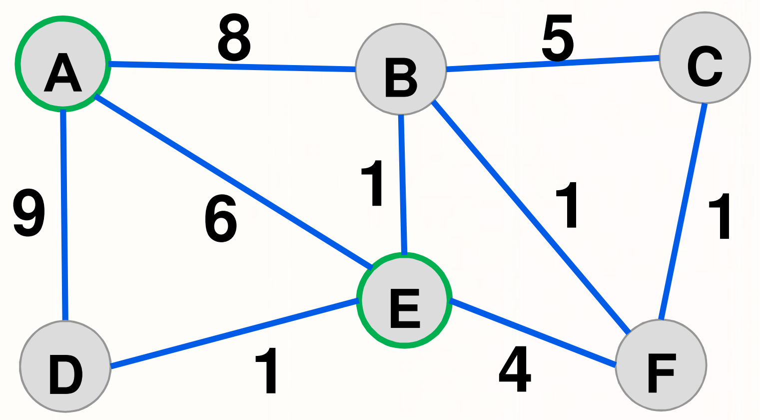
- Αρχικοποίησε τον πίνακα distance με τις ακμές της κορυφής εισόδου.

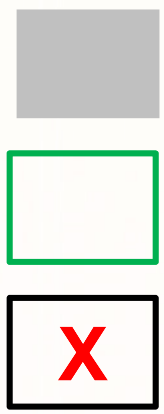


Εκτέλεση	S	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
1	S=∅	0	∞	∞	∞	∞	∞
2	S={A}	0	8	∞	9	6	∞

- Σε κάθε βήμα του αλγόριθμου, **επίλεξε** «άπληστα» την **κορυφή με την ελάχιστη απόσταση που δεν έχεις επισκεφθεί**

Παράδειγμα Εκτέλεσης Αλγορίθμου Dijkstra 3/4

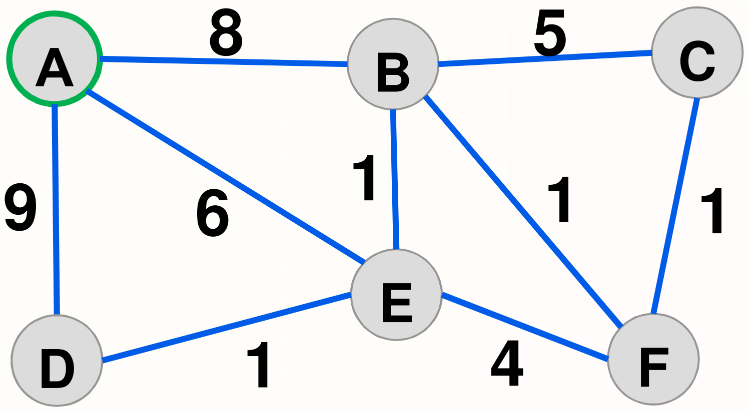




Εκτέλεση	S	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
1	S=∅	0	∞	∞	∞	∞	∞
2	S={**A**}	0	8	∞	9	6	∞
3	S={A,**E**}	0	7	∞	7	6	10

- Η E (απόσταση=6) έχει ακμές με **B,D**(6+1=7) και **F**(6+4=10)
- **Αφού οι αποστάσεις αυτές είναι μικρότερες από τις υπάρχουσες τότε ενημερώνεται ο πίνακας ανάλογα.**

Παράδειγμα Εκτέλεσης Αλγορίθμου Dijkstra 4/4



Κορυφές που έχουμε επισκεφθεί (visited=1)

Επιλογή της ελάχιστης απόστασης

X Μείωση από την προηγούμενη απόσταση

Εκτέλεση	S	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
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,D}	0	7	12	7	6	8
6	S={A,E,B,D,F}	0	7	**9**	7	6	8
7	S={A,E,B,D,F, C}	0	7	9	7	6	8

Υλοποίηση του Αλγόριθμου Dijkstra

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

   distance[|V|]={∞}; // Απόσταση κορυφής i από κορυφή A
   visited[|V|]={};   // Σημειώνει αν περάσαμε η όχι από κορυφή
   count = 0;         // Μετρητής κορυφών που επισκεφθήκαμε

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

   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: $O(|V|^2)$

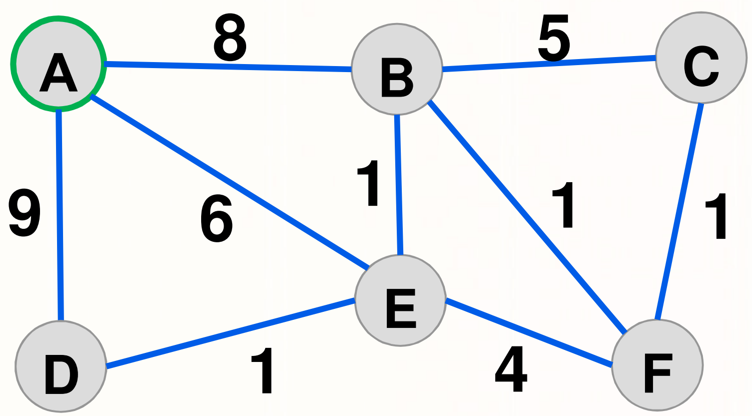
Dijkstra για εύρεση του Βραχύτερου Μονοπατιού

- 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 X

Αλγόριθμος Dijkstra για εύρεση Βραχ. Μονοπατιού



Κορυφές που έχουμε επισκεφθεί (visited=1)

Επιλογή της ελάχιστης απόστασης

X Μείωση από την προηγούμενη απόσταση

Εκτέλεση	S	d(A)	d(B)	d(C)	d(D)	d(E)	d(F)
1	S=∅	0,-	∞,-	∞,-	∞,-	∞,-	∞,-
2	S={**A**}	0,-	8,A	∞	9,A	6,A	∞
3	S={A,**E**}	0,-	**7,E**	∞	**7,E**	6,A	10,E
4	S={A,E,**B**}	0,-	7,E	12,B	7,E	6,A	**8,B**
5	S={A,E,B,D}	0,-	7,E	12,B	7,E	6,A	8,B
6	S={A,E,B,D,F}	0,-	7,E	**9,F**	7,E	6,A	8,B
7	S={A,E,B,D,F, C}	0,-	7,E	9,F	7,E	6,A	8,B

Υλοποίηση του Αλγόριθμου Dijkstra

```
// A: Σημείο Εκκίνησης. G(V,E): O Γράφος
dijkstra( G(V,E), vertex A){
   distance[|V|]={∞}; // Απόσταση κορυφής i από κορυφή A
   visited[|V|]={};   // Σημειώνει αν περάσαμε η όχι από κορυφή
   shortest[|V|]={};   // Σημειώνει το βραχύτερο μονοπάτι
   count = 0;         // Μετρητής κορυφών που επισκεφθήκαμε
  // Αρχικοποίηση σημείου εκκίνησης
  distance[A]=0; visited[A]=1;

   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);
                shortest[v] = u; }
        }
        count++;
   }
}
```