ΠΕΡΙΕΧΟΜΕΝΑ:

Γιώργος Μ.
Σμαραγδένιος Χορηγός Μαθήματος

Γεράσιμος Μπ.
Ασημένιος Χορηγός Μαθήματος

- **Ελάχιστο Συνδετικό Δένδρο:**
  - Δίνεται απλός, μη κατευθυνόμενος γράφος με βάρη G=(V,E,W),
  - Ζητείται ένα ελάχιστο συνδετικό δένδρο του γραφήματος
  - (Αναλυτικά: Βλέπε Θεωρία Γράφων: Μάθημα 6.2)

Παρατηρήσεις για την υλοποίηση:

- Επιλέγουμε την αποτύπωση του γράφου με λίστες γειτνίασης (λόγω πιο γρήγορης πρόσβασης στους επόμενους μιας κορυφής)

Αλγόριθμος του Prim

Αρχικοποίηση:

- Τοποθετούμε αυθαίρετα μια κορυφή στο συνδετικό δένδρο

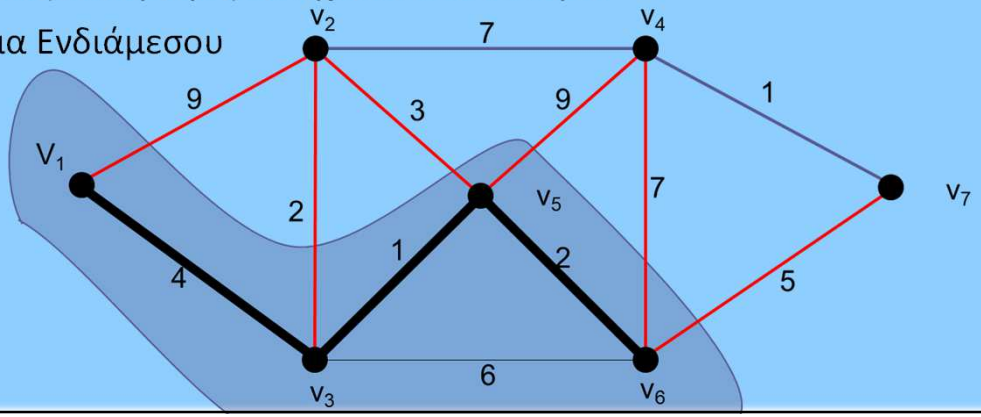
Σε κάθε βήμα:

- Υποψήφιες ακμές για να μπουν στο συνδετικό δένδρο είναι εκείνες οι ακμές που έχουν το ένα τους άκρο στο υπο κατασκευή συνδετικό δένδρο και το άλλο τους άκρο εκτός του συνδετικού δένδρου.
- Επιλέγεται η ακμή με το ελάχιστο βάρος από τις υποψήφιες
- Η ακμή εισάγεται στο δένδρο καθώς και το άκρο της που δεν ανήκε στο δένδρο.

Τερματισμός:

- Όταν όλες οι κορυφές εισαχθούν στο δένδρο.

Παράδειγμα Ενδιάμεσου Βήματος:



Αλγόριθμος σε Ψευδογλώσσα (~ από Wikipedia):

```
procedure Prim(G=(V,E,W)):
  Q: σύνολο όλων των κορυφών, T=κενό δένδρο
  Αρχικοποίησε: C[v]=+∞ για κάθε κορυφή v
  Διάλεξε αυθαίρετα κάποια αφετηρία s, αφαίρεσε s από Q
  Για κάθε γείτονα x της s:
        C[x] = W[s,x], E[x] = (s,x),
  while (Q δεν είναι άδειο)
     v = κορυφή του Q με ελάχιστο C
     Αφαίρεσε τη v από το Q
     Θέσε E[x] στο συνδετικό δένδρο.
     Για κάθε γείτονα x της v (που ανήκει στο Q):
          Αν W[v,x] < C[x]:
               C[x] = W[v,x],  E[x] = (v,x)
   Επέστρεψε το T
```

Πολυπλοκότητα:

- Η πολυπλοκότητα του αλγορίθμου είναι O(n*A+m*B), A: χρόνος εντοπισμού κορυφής και B: διόρθωση στο Q
- Αν Q είναι απλός πίνακας: $O(n^2+m) = O(n^2)$
- Αν Q είναι σωρός Fibonacci: O(m+nlogn)

Υλοποίηση σε C (βλέπε project Prim):

```
GRAPH Prim(GRAPH g)
{
   struct edge {
      int vertex1;
      int vertex2;
   };
   struct edge *E;
   int *Q, *C;
   int i, x, v, cnt, vertices, neighbors_length;
   int min, infinity;
   int start = 0;
   elem *neighbors;
   GRAPH st;

   vertices = GR_vertices_count(g);
   GR_init(&st, vertices);

   /* C[*]: Varos elaxistis akmis tis korifis v */
   C = (int *)malloc(sizeof(int)*vertices);
   if (!C)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   infinity = 0;
   for (v=0; v<vertices; v++)
   {
      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (i=0; i<neighbors_length; i++)
         if (infinity < neighbors[i].weight)
            infinity = neighbors[i].weight;
      free(neighbors);
   }
   infinity = infinity + 1;
   for (i=0; i<vertices; i++)
      C[i] = infinity;
   C[start] = 0;

   /* E[*]: kaliteri akmi gia na sindethei o komvos v */
   E = (struct edge *)malloc(sizeof(struct edge)*vertices);
   if (!E)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   /* Q[i]=0 an i korufi den exei oristikopoiithei, alliws 1 */
   Q = (int *)malloc(sizeof(int)*vertices);
   if (!Q)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   for (i=0; i<vertices; i++)
      Q[i] = 0;
```

```
/* Prim */
GR_neighbors(g, start, &neighbors_length, &neighbors);
for (x=0; x<neighbors_length; x++)
{
   C[neighbors[x].id] = neighbors[x].weight;
   E[neighbors[x].id].vertex1 = start;
   E[neighbors[x].id].vertex2 = neighbors[x].id;
}
free(neighbors);
Q[start]=1;

cnt=1;
while(cnt<vertices)
{
   min=infinity;
   for (i=0; i<vertices; i++)
      if (C[i]<min && Q[i]==0)
      {
         min = C[i];
         v=i;
      }
   cnt++;
   Q[v]=1;
   GR_add_edge(st, E[v].vertex1, E[v].vertex2, C[v]);
```

```
   printf("\nProstithetai: (%d, %d) me varos %d", E[v].vertex1,
            E[v].vertex2, C[v]);

      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (x=0; x<neighbors_length; x++)
      {
         if (Q[neighbors[x].id]==0 &&
             neighbors[x].weight<C[neighbors[x].id])
         {
            C[neighbors[x].id] = neighbors[x].weight;
            E[neighbors[x].id].vertex1 = v;
            E[neighbors[x].id].vertex2 = neighbors[x].id;
         }
      }
      free(neighbors);
   }

   free(C);
   free(E);
   free(Q);

   return st;
}
```

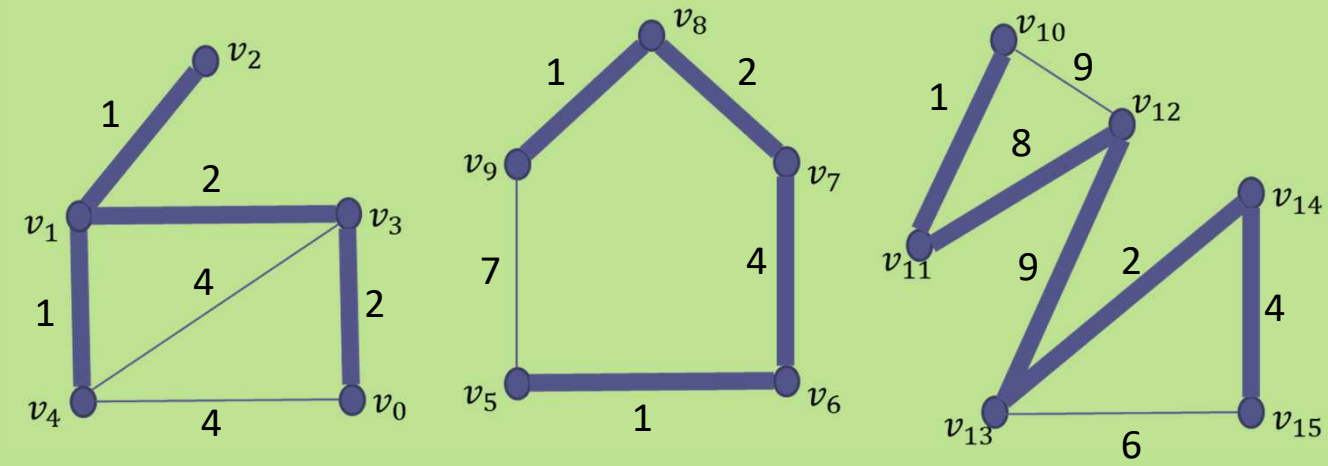
Άσκηση 1: Μη συνδεόμενο γράφημα

Κατασκευάστε τη συνάρτηση PrimSpanningForest η οποία με όρισμα έναν απλό μη κατευθυνόμενο γράφο (που ενδέχεται να μην είναι συνδεόμενος) :

- Να επιστρέφει ένα δάσος από ελάχιστα συνδετικά δένδρα.

Παράδειγμα:

- Δοκιμάστε τον κώδικα σας, στο παρακάτω παράδειγμα:



Άσκηση 2: Με ουρά προτεραιότητας

Θα υλοποιήσουμε το Q ως μια ουρά προτεραιότητας.

- Η προτεραιότητα θα είναι η ετικέτα μήκους C (όσο μικρότερο το C, τόσο μεγαλύτερη η προτεραιότητα)
- Όταν γίνεται διόρθωση του L κάποιας κορυφής, θα πρέπει να γίνεται διόρθωση της ουράς.

Ενσωματώστε την ουρά προτεραιότητας στον αλγόριθμο.

Παρατήρηση:

- Η χρήση μίας πιο αποδοτικής ουράς προτεραιότητας (σωρός Fibonacci) ρίχνει την πολυπλοκότητα έως και O((m+n)logn)

- Ο **αλγόριθμος του Kruskal** υπολογίζει το ελάχιστο συνδετικό δένδρο ενός απλού, συνδεόμενου, μη κατευθυνόμενου γραφήματος.
- Η ιδέα του είναι πιο απλή από αυτήν του Prim: "Πρόσθεσε την ακμή ελαχίστου βάρους που δεν δημιουργεί κύκλο"

Αλγόριθμος του Kruskal

Αρχικοποίηση:

- Κατασκεύασε ένα δάσος Τ απομονωμένων κορυφών.
- Κατασκεύασε ένα σύνολο S με όλες τις ακμές του γραφήματος.

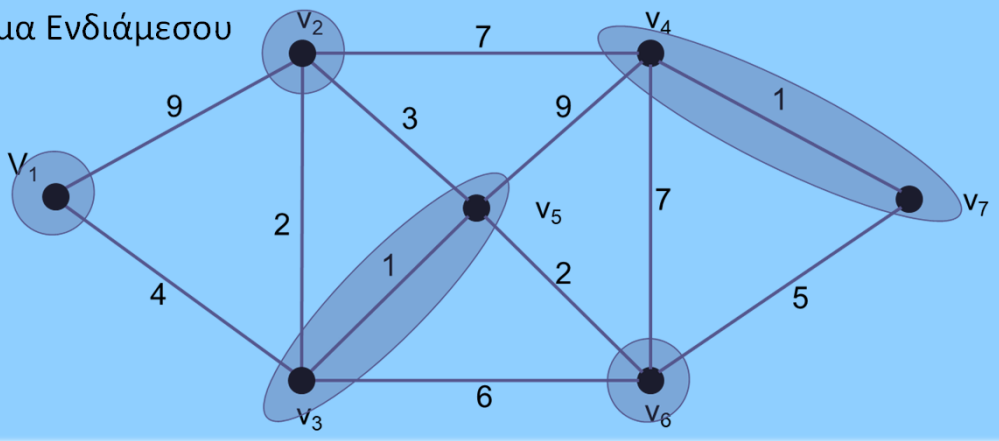
Σε κάθε βήμα:

- Διάλεξε την ακμή e ελαχίστου βάρους του S, αφαίρεσε την απο το S.
- Εξέτασε αν η προσθήκη της e δημιουργεί κύκλο στο Τ. Αν ναι, τότε απορρίπτεται, αλλιώς την προσθέτουμε στο Τ.

Τερματισμός:

- Όταν προστεθούν n-1 ακμές στο Τ.

Παράδειγμα Ενδιάμεσου Βήματος:



Παρατηρήσεις για την υλοποίηση:

- Επιλέγουμε την αποτύπωση του γράφου με λίστες γειτνίασης (για να κατασκευάζουμε γρήγορα το σύνολο ακμών S)
- Χρησιμοποιείται η δομή δεδομένων Ξένα Σύνολα (Disjoint Sets) – (Βλ. Δομές Δεδομένων – Μάθημα 11)

Αλγόριθμος σε Ψευδογλώσσα (~ από Wikipedia):

```
procedure Kruskal(G=(V,E,W)):
    Θέσε Τ να περιέχει μόνο το V
    Θέσε Ε': οι ακμές του Ε ταξινομημένες κατά το βάρος

    Για κάθε κορυφή του V: MAKE-SET(v)

    Για κάθε ακμή e=(u,v) του Ε'
        Αν FIND-SET(u)≠FIND-SET(v):
            Θέσε την e στο Τ
            UNION(FIND-SET(u), FIND-SET(v))
        Αν |E(T)|=n-1 : break

    Επέστρεψε το Τ
```

Πολυπλοκότητα:

- Η πολυπλοκότητα του αλγορίθμου είναι O(mlogm) (ο χρόνος της ταξινόμησης των ακμών) άρα ισοδύναμα (αφού $m=O(n^2)$) είναι O(mlogn)

Υλοποίηση σε C (βλέπε project Kruskal):

```
GRAPH Kruskal(GRAPH g)
{
   struct edge *sorted_edges;
   int edge_count, i, x, v, cnt, vertices, neighbors_length;
   elem *neighbors;
   GRAPH st;  DISJOINT d;

   vertices = GR_vertices_count(g);
   GR_init(&st, vertices);
   DS_init(&d, vertices);

   /* Sort Edges */
   edge_count = GR_edges_count(g);
   sorted_edges = (struct edge *)malloc(sizeof(struct
                        edge)*edge_count);
   if(!sorted_edges)
   {
      printf("Error Allocating Memory");
      exit(0);
   }
   i=0;
   for (v=0; v<vertices; v++)
   {
      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (x=0; x<neighbors_length; x++)
      {
         if (v<neighbors[x].id)
         {
            sorted_edges[i].vertex1=v;
            sorted_edges[i].vertex2=neighbors[x].id;
            sorted_edges[i].weight=neighbors[x].weight;
            i++;
         }
      }
   }
   quicksort(sorted_edges, 0, edge_count-1);
   /* Kruskal */
   i=0; cnt=0;
   while(cnt<vertices-1)
   {
      if(DS_find_set(d,sorted_edges[i].vertex1)!=
         DS_find_set(d,sorted_edges[i].vertex2))
      {
         GR_add_edge(st, sorted_edges[i].vertex1,
                     sorted_edges[i].vertex2, sorted_edges[i].weight);
         DS_union(&d, sorted_edges[i].vertex1,
                      sorted_edges[i].vertex2);
         cnt++;
      }
      i++;
   }
```

Άσκηση 3: Τροποποιήση Ελάχιστου Συνδετικού Δένδρου

Έστω ότι έχουμε υπολογίσει ένα ελάχιστο συνδετικό δένδρο και έπειτα προστίθεται μία ακμή στον αρχικό γράφο.

- Δώστε έναν αποδοτικό αλγόριθμο (που δεν χρησιμοποιεί ούτε τον αλγόριθμο του Prim, ούτε του Kruskal) ώστε να υπολογίζεται ένα νέο ελάχιστο συνδετικό δένδρο.