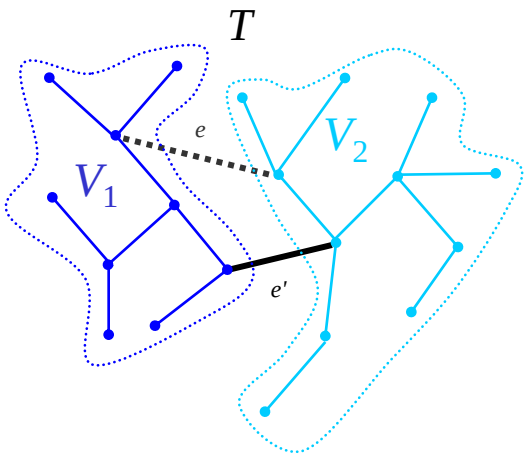
Ελάχιστα Επικαλύπτοντα Δένδρα

- Ορισμός
  - Εύρεση Επικαλύπτοντος Δένδρου με Ελάχιστο Βάρος, δηλαδή ελάχιστο άθροισμα βαρών ακμών
- Αλγόριθμοι Prim, Kruskal, Baruvka
  - Βασίζονται στην τεχνική της Απληστίας
  - Η ορθότητα τους στηρίζεται στο ίδιο θεώρημα

Θεώρημα Ορθότητας ΕΕΔ

- Δοθέντων ενός συνεκτικού βεβαρημένου γραφήματος G και μίας διαμερίσεως (V_1,V_2) του συνόλου των κορυφών, υπάρχει ένα ΕΕΔ T που περιλαμβάνει την ακμή e ελαχίστου βάρους, συνδέουσα κορυφή του V_1 με κορυφή του V_2

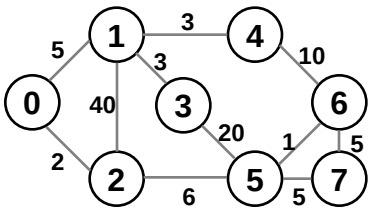


Αλγόριθμος Kruskal

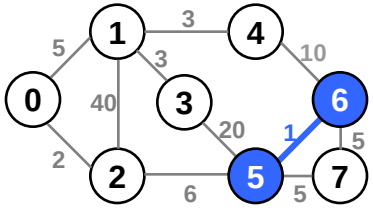
- Εξετάζει μία προς μία τις ακμές, *κατά αύξουσα τιμή βάρους*, και τις προσθέτει *μόνον εφόσον δεν δημιουργείται κύκλος*
  - Ξεκινά από V στοιχειώδη δένδρα, τα οποία, εν τέλει, με τις διαδοχικές προσαρτήσεις ακμών, ενοποιούνται στο τελικό ΕΕΔ
  - Ορθότητα:
    - Σε κάθε βήμα, έστω $e=(a,b)$ η ακμή που επιλέγεται, με $a<b$
    - V_1= όλες οι προσπελάσιμες κορυφές, μέσω μονοπατιών από ακμές δένδρων, τής a
    - $V_2 = V-V_1$

Παράδειγμα

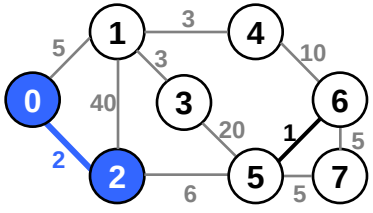
{0}, {1}, {2},{3},{4}, {5},{6},{7}



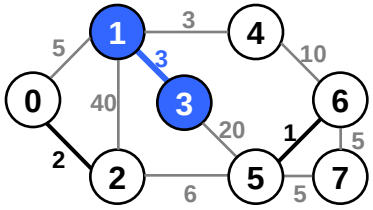
{0}, {1}, {2},{3},{4}, {5,6},{7}



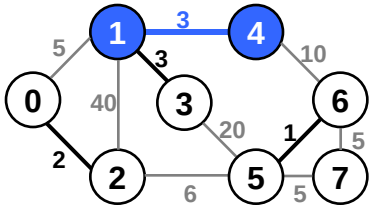
{0,2}, {1}, {3},{4}, {5,6},{7}



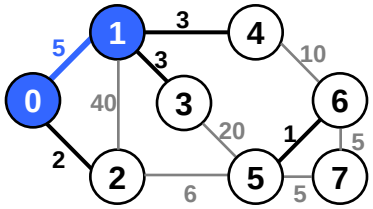
{0,2}, {1,3},{4}, {5,6},{7}



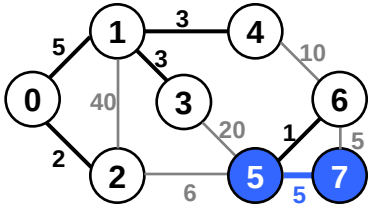
{0,2}, {1,3,4}, {5,6},{7}



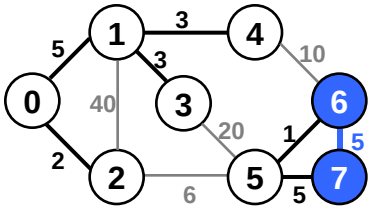
{0,2,1,3,4}, {5,6},{7}



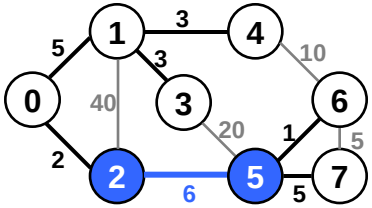
{0,2,1,3,4}, {5,6,7}

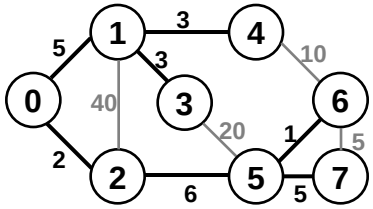


{0,2,1,3,4}, {5,6,7}



{0,2,1,3,4,5,6,7}





Ψευδοκώδικας

Algorithm kruskalMST(graph g)

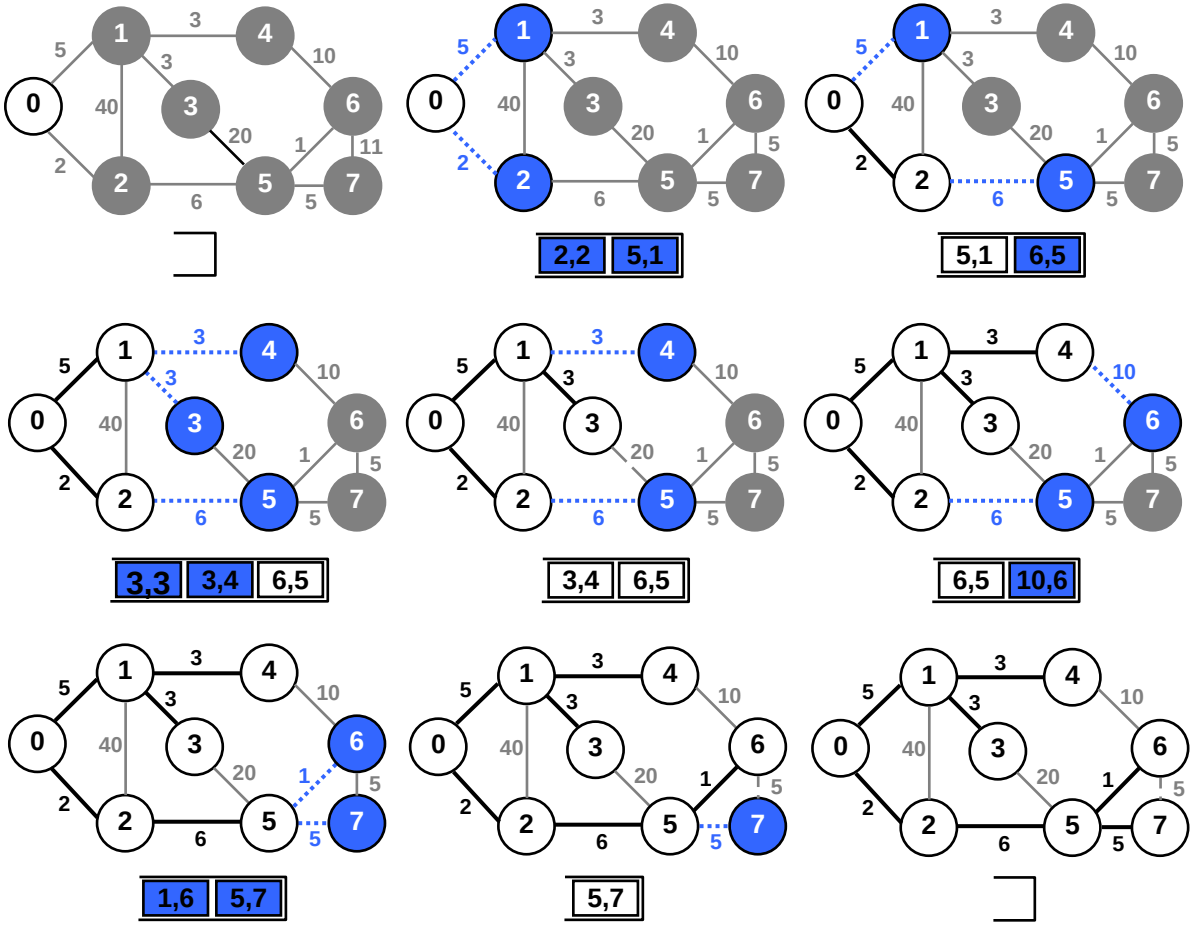
1. list mstList = **new** list();
2. unionfind u = **new** unionfind(g.V); *// δομή union find, αρχικοποιημένη με τα V μονοσύνολα-δένδρα*
3. sort(g.Edges); **int** i=0; *// ταξινόμηση των ακμών σε χρόνο O(E*log*E)*
4. **while** ((i < g.E) && (mstList.nofElements() < g.V)) *// O(E*log**E*) *ή O(Va(E,V))*
5. **if** (!u.find(g.Edges[i].v, g.Edges[i].w)){ *// διαφορετικά δένδρα - συνολικά, # E finds*
6. u.union(g.Edges[i].v, g.Edges[i].w); *// συνένωση των δύο δένδρων - συνολικά, # V-1 unions*
7. mstList.addLast(g.Edges[i]);
8. i++;
9. }
10. **return** mstList;

- Πολυπλοκότητα: *O*(*E*log*E*) *εξ αιτίας του sorting*
- Εναλλακτικά, α) radix sort ⇒ *O*(*E*log**E*) ή *O*(*Va*(*E*,*V*)), εάν χρησιμοποιηθούν κατάλληλες δομές union find, β) ουρά προτεραιότητας ⇒ *O*(*E*+*E'*log*V*), *E'* πλήθος ακμών με βάρος μικρότερου της βαρύτερης ακμής του ΕΕΔ

Αλγόριθμος Prim

- Ξεκινώντας από ένα στοιχειώδες δένδρο-κορυφή, το επεκτείνουμε, ακμή προς ακμή, επιλέγοντας κάθε φορά την ελαφρύτερη ακμή που συνδέει κορυφή του τρέχοντος δένδρου με κορυφή εκτός (ανήκουσα, δηλαδή, στο σύνολο παρυφής)
- Ορθότητα: V_1 οι κορυφές του υπό διαμόρφωση ΕΕΔ, $V_2=V- V_1$

Παράδειγμα


0 1 2 3 4 5 6 7
5 3 10 3 40 20 1 11 2 6 5
2,2 5,1
5,1 6,5
3,3 3,4 6,5
3,4 6,5
6,5 10,6
1,6 5,7
5,7

Λύση Πολυπλοκότητας $O(E\log V)$

Algorithm primMST(**graph** g)

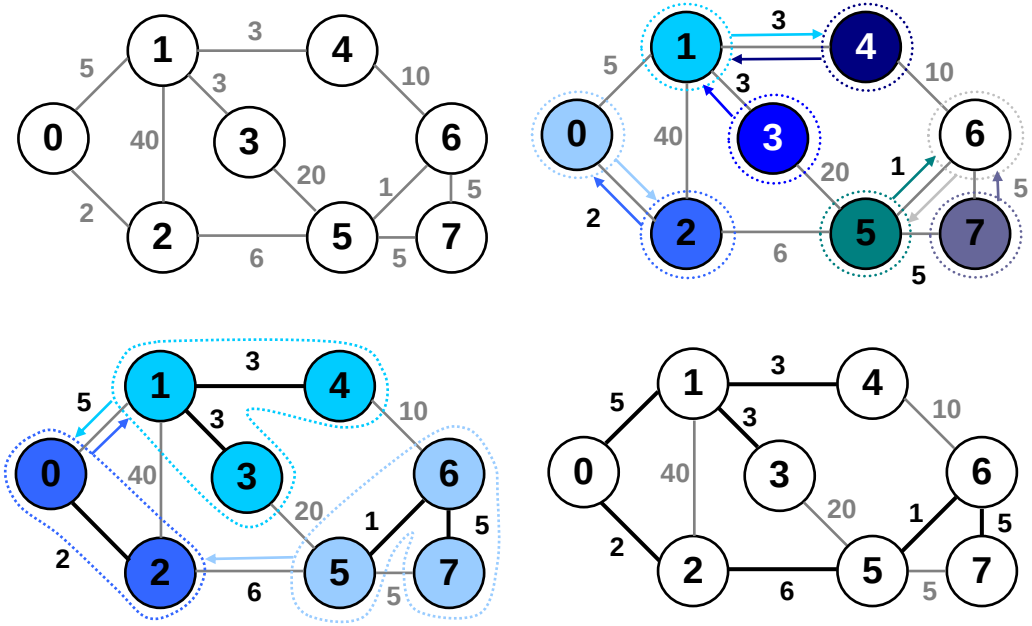
```
1.  pqueue pq  = new pqueue(0,0);          // βοηθητική ουρά ανεξερεύνητων κορυφών
2.  boolean[] isIn = new boolean[g.V];  // βοηθητικός πίνακας κορυφών υπό δημιουργία ΕΕΔ
3.  double[] key = new double[g.V];      // βοηθητικός πίνακας προτεραιοτήτων , μία για κάθε κορυφή
4.  for (v = 0; v < g.V; v++){               // αρχικοποίηση
5.    isIn[v] = false;
6.    key[v] = INFTY;
7.    pq.insKey(v, key[v]);                    //  V φορές
8.  }
9.   while (!pq.isEmpty()){   // όσο υπάρχουν ανένταχτες κορυφές
10.    v = pq.delMin();          // ελαφρύτερη ακμή με απόληξη v - εκτελείται V φορές
11.    for (x = G.List[v]; x != null; x = x.getNext()){
12.      if ((!isIn[w = x.v]) && (x.weight < key[w])){
13.        key[w]=x.weight;   // η (v,w) δεν ανήκει στο δένδρο και είναι ελαφρύτερη από την παλιά
14.        pq.decKey(w, x.weight);       // διόρθωση προτεραιότητας – εκτελείται Ε φορές
15.        g.T[w]=v;             // Τρέχουσα σύνδεση της w μέσω (v,w)
16.      }
17.    }
18.    isIn[v] = true;           // η v ανήκει πλέον στο δένδρο
19. }
```

- Εναλλακτικά: (α) Fibonacci heaps με λογαριθμικό delMin και $O(1)$ αλλαγή προτεραιότητας $\Rightarrow$ $O(V\log V+E)$, (β) d-heap, με $d\log_d V$ delmin/αλλαγή προτεραιότητας $O(Vd\log_d V+E\log_d V)$, οπότε για αραιά γραφήματα βάζουμε $d=E/V$ και $O(V\log V)$

Αλγόριθμος Baruvka

- Αποτελεί τον παλαιότερο αλγόριθμο
- Σε σχέση με τον αλγόριθμο του Kruskal
  - Ομοιάζει, γιατί και αυτός διατηρεί σύνολο από δένδρα
  - Διαφέρει, γιατί πράττει πολλαπλές συνενώσεις κάθε φορά και όχι μόνο μία:
    - κάθε δένδρο της συλλογής συνδέεται με το κοντινότερο προς αυτό (ελαφρύτερη ακμή)
    - εάν υπάρχουν περισσότερες από μία επιλογές ελαφρύτερης ακμής, πρέπει να βρεθεί τρόπος σπασίματος των ισοπαλιών, π.χ., επιλογή μικρότερου ή μεγαλύτερου ονόματος, ώστε να αποφευχθεί ο σχηματισμός κύκλων (γιατί;)
- ΕΥΝΟΕΙ ΤΟΝ ΠΑΡΑΛΛΗΛΙΣΜΟ
- Ορθότητα:
  - Ανάλογη με του Prim

Παράδειγμα


0
1
2
3
4
5
6
7
5
3
40
3
10
20
1
5
2
6
5
3
40
3
10
20
1
5
2
6
5
3
40
3
10
20
1
5
2
6
5
3
40
3
10
20
1
5
2
6
5

Λύση πολυπλοκότητας O((V+E)logV)

Algorithm BaruvkaMST(graph g)

```
1.  list mstEdges = new list();     // λίστα ακμών ΕΕΔ
2.  int[] CC = new int[g.V];        // πίνακας κορυφής-συνιστωσών
3.  int[] NCC = new int[g.V];       // πίνακας ελαφρύτερης ακμής προς συνιστώσα
4.  int comnumber = g.V;
5.  for (k = 0; k < g.V; k++)
6.    CC[k] = k;
7.  do {                            // όσο υπάρχουν περισσότερες της μιας συνιστώσες
8.     for (x = 0; x < E; x++){     // εξέταση ακμών - χρόνος O(V+E)
9.       if ((i = g.CC[g.Edge[x].v])) == (j = g.CC[g.Edge[x].w]))
10.          continue;              // τα άκρα ανήκουν στην ίδια συνιστώσα
11.    if (g.Edge[x].weight < NCC[i].weight) // i≠j
12.          NCC[i] = g.Edge[x];    // νέα ελαφρύτερη για την συνιστώσα i
13.    if (g.Edge[x].weight < NCC[j].weight)
14.          NCC[j] = g.Edge[x];    // νέα ελαφρύτερη για την συνιστώσα j
15.   }
16.   for (c = 0; c < g.comnumber; c++)          // προσθήκη ακμών
17.         mstEdges.add(NCC[c]);
18.   comnumber = dfsEdges(mstEdges, CC);  // dfs για τον προσδιορισμό των συνιστωσών – χρόνος O(V+E)
19. }
20. while (comnumber > 1); //#logV σε κάθε στάδιο, τουλάχιστον υποδιπλασιάζεται το # συνιστωσών
21. return mstEdges;
```

- Η εκτίμηση είναι ΑΠΑΙΣΙΟΔΟΞΗ…