Arbre Couvrant Minimal

- **Données**: Un graphe connexe **G = (V,E.w)** avec des arêtes (arcs) à des poids.
- **But**: Un arbre couvrant **G = (V,E'.w)** dont le pois total des arêtes (arcs) soit minimal.

Complexité:**O((V+E)log V)**
O(ElogV) //

Prim(G.w)

```
1.  Dist = [∞ for u in G]
2.  M = [0 for u in G]
3.  P = [0 for u in G]
4.  Dist[s] = 0 /#s is an arbitrary vertex of V
5.  P[s] = s
6.  Q = Makeheap(V) /#(using dist-values as keys)
7.  while Q:
8.      z = Q.deletemin_key() /# extract min operation
9.      M[z] = 1
10.     for x in Adj[z]:
11.         If  M[x] = 0 and Dist[x] > w(z, x)
12.             Q.decrease_key(x) /#Dist[x] = w(z, x),  P[x] = z
```

Kruskal(G.w)

```
1.    T = []
2.    A = sort(E)
3.    L = make_set(G)/ [-1 for u in G]
4.    While A:
5.          (z,x) = A.pop()
6.          a= find(L, z)
7.          b= find(L, x)
8.          if  a ! = b:
9.              Union(L, a, b)
10.         T = T + (z,x)
11.   return T
```

VERTEX COVER D'UN GRAPHE

- **Données**: Un graphe **G=(V,E)** en liste d'adjacence.
- **But**: Chercher un ***vertex cover*** du **G** . C-à-d un ***plus petit*** sous-ensemble **V'** de **V** tel que toute arête de **E** ait au moins une extremité dans **V'**.

Complexité: O((V+E)logV)

VERTEX COVER(G)

```
1.  S= []
2.  While E:
3.      (z,x)=E.top()
4.      S.append(z)
5.      S.append(x)
6.      Fox w in adj[z]:
7.          Adj[z].delete(w)
8.      For y in adj[x]:
9.          Adj[x].delete(y)
10  Return S
```

VERTEX_COVER HEURISTIC (G)

```
1.  D = [degree for u in G]
2.  Q = Makeheap(V) / #( degree-values as keys
3.  while Q:
4.    z = Q.delete_max_key() / # extract max
5.        If  D[z] > 0 :
6.            D[z]  = -1
7.        else:  Print D_negatif
8.                break
9.        for x in Adj[z]:
10.               If  D[x] > 0
11.                   Q.decrease_key(x) / #D[x]- =1
```

DOMINATING SET D'UN GRAPHE

- **Données**: Un graphe **G=(V,E)** en liste d'adjacence.
- **But**: Chercher un ***dominating set*** du **G** . C-à-d un ***plus petit*** sous-ensemble **V'** de **V** tel que tout autre sommet de **V** (c-à-d un sommet de **V-V'**) soit voisin d'au moins d'un sommet de **V'**.

Complexité: O((V+E)logV)

DOMINATING_SET_ = HEURISTIC (G)

```
1.  D = [degree for u in G]
2.  Q= Initialize(V:D) / #( degree-values as keys)
3.  while Q:
4.      z = Q.delete_max_key() / # extract max
5.          If D[z] >= 0 :
6.              Q.update_key(z:D[z]  = -2)
7.          else: Print D: -2
8.                  break
9.          for x in Adj[z]:
10.                 Q.update_key(x:D[x]=-1)
11.                 for y  in Adj[x]:
12.                     If   D[y] > 0
13.                         Q.update_key(y:D[y]-=1)
```