CHEMINS DANS LES GRAPHES

- Le PCC entre 2 sommets dans un graphe sans poids
- Le PCC entre un sommet et tous les autres sommets dans un graphe à des poids positifs
- Le PCC entre un sommet et tous les autres sommets dans un graphe pondéré.
- Le PCC entre un sommet et tous les autres sommets dans un graphe orienté et sans cycle

Generic Shortest Path (G, s)

```
Dist = [∞ for u in G]
Dist[s] = 0
P = []
P[s] = s
Q = []
Q.append(s)
while Q:
    z = Q.get_and_delete()
    for x in Adj[z]:
        If   Dist[x] > Dist[z]+w(z, x): /edge (z, x) is « tense »
             Dist[x] = Dist[z]+w(z, x)
             P[x] = z
             Q.append(x)
Return < P, D >
```

Le plus court chemin entre 2 sommets

- **Données**: Un graphe **G** avec des arêtes (arcs) sans poids et 2 sommets **s** et **t** de de **G**.
- **But**: Le plus court chemin entre **s** et **t**.

Complexité: **O(V+E)**

Shortest Path (G, s)

```
Dist = [∞ for u in G]
Dist[s] = 0
P = []
P[s] = s
Q = []
Q.append(s)
while Q:
    z = Q.pop()
    for x in Adj[z]:
        If  P[x] == 0: /edge (z, x) is « tense »!!!!
            Dist[x] =  Dist[z]+1
            P[x] = z
            Q.append(x)
 return path(P,s,t)
```

Le plus court chemin: Dijkstra

- **Données**: Un graphe **G** avec des arêtes (arcs) à des poids positifs et un sommet **s**.
- **But**: Le plus court chemin entre **s** et les autres sommets du graphe.

Complexité: **$O(V^2+E)$ // $O((V+E)\log V)$ // $O(E+V\log V)$**

The algorithm DIJKSTRA(G,s)

```
Dist = [infinity for u in G]
Dist[s]=0
P[s]=s
Q = V
while Q:
    z = Q.delete_min()
        for x in Adj[z]:
            Relax(z, x)
...
Return <P, Dist>
```

Dijkstra1 (G.w, s)

```
Dist = [∞ for u in G]
Dist[s] = 0
P[s] = s
Q = Makeheap(V)  (using dist-values as keys)
while Q:
    z = Q.deletemin_key()
    for x in Adj[z]:
        If  Dist[x] > Dist[z]+w(z, x):
            Dist[x] = Dist[z]+w(z, x)
            P[x] =  z
            Q.decrease_key(x)
```

Dijkstra2 (G.w, s)

```
Dist = [∞ for u in G]
M = [0 for u in G]
P= []
Dist[s] = 0
M[s]=1
P[s] = s
Q.add(s)
while Q:
    z = Q.delete_min()
    for x in Adj[z]:
        If  Dist[x] > Dist[z]+c(z, x)
            Dist[x] = Dist[z]+c(z, x)
            P[x] = z
            If   M[x]==0:
                 M[x]=1
                 Q.add(x)
            elif Q.decrease_key(x)
```

Le plus court chemin: Bellman-Ford

Données: Un graphe valué **G** et un sommet **s** sans cycle négatif.

But: Le plus court chemin entre **s** et les autres sommets du graphe.

Complexité: O(VE)

Bellman-Ford (G.w, s)

```
Dist = [∞ for u in G]
M = [0 for u in G]
P = []
Dist[s]=0
M[s]=1
P[s]=s
Q.enqueue(s)
while Q:
    z = Q.dequeue()
    M[z]=0
    for x in Adj[z]:
        If  Dist[x] > Dist[z]+w(z, x)
            Dist[x] = Dist[z]+w(z,x)
            P[x] = z
            if  M[x]==0:
                M[x]=1
                Q.enqueue(x)
```

Le chemin le plus court: DAGs

- **Données**: Un graphe orienté et sans cycle **G,** et un sommet **s** de **G.**
- **But**: Le plus court chemin entre **s** et les autres sommets du graphe.

Complexité: O(V+E)

SHORTEST_PATHS_DAGs (G.w, s)

```
Dist = [∞ for u in G]
Dist[s] =0
P[s] = s
Q = Topological_sort(G)
while Q:
    z = Q.pop()
    for x in Adj[z]:
        If  Dist[x] > Dist[z]+w(z, x)
            Dist[x] = Dist[z]+w(z, x)
```