M S T

```
PRIM(G, w, r)
  Q = ∅
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)      // r.key = 0
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              DECREASE-KEY(Q, v, w(u, v))
```

attention, on calcule la distance par rapport aux noeuds déjà dans le graphe, pas à la source.

- Initialize Q and first **for** loop: $O(V \lg V)$
- Decrease key of r: $O(\lg V)$
- **while** loop: V EXTRACT-MIN calls $\Rightarrow O(V \lg V)$
  $\leq E$ DECREASE-KEY calls $\Rightarrow O(E \lg V)$
- Total: $O(E \lg V)$ (can be made $O(E + V \lg V)$ with careful queue implementation)

```
KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into nondecreasing order by weight w
  for each (u, v) taken from the sorted list
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

- Initialize A: $O(1)$
- First **for** loop: V MAKE-SETS
- Sort E : $O(E \lg E)$
- Second **for** loop: $O(E)$ FIND-SETS and UNIONS
- Total time: $O((V + E)\alpha(V)) + O(E \lg E) = O(E \lg E) = O(E \lg V)$

If edges already sorted time is $O(E\alpha(V))$ which is almost linear

parcours graphe

```
BFS(V, E, s)
  for each u ∈ V − {s}
      u.d = ∞
  s.d = 0
  Q = ∅
  ENQUEUE(Q, s)
  while Q ≠ ∅
      u = DEQUEUE(Q)
      for each v ∈ G.Adj[u]
          if v.d == ∞
              v.d = u.d + 1
              ENQUEUE(Q, v)
```

O(V + E)

```
DFS(G)
  for each u ∈ G.V
      u.color = WHITE
  time = 0
  for each u ∈ G.V
      if u.color == WHITE
          DFS-VISIT(G, u)
```

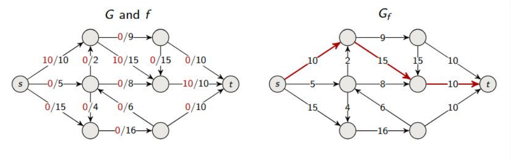
```
DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          DFS-VISIT(v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

O(V + E)

calcul du max-flow ou min-cut (bottleneck)

O(V * E^2)

```
FORD-FULKERSON-METHOD(G, s, t):
  1. Initialize flow f to 0
  2.  while exists an augmenting path p in the residual network Gf
  3.      augment flow f along p
  4. return f
```



shortest path

```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

- INIT-SINGLE-SOURCE updates ℓ, π for each vertex in time $\Theta(V)$
- Nested **for** loops runs RELAX $V - 1$ times for each edge. Hence total time for these loops is $\Theta(E \cdot V)$
- Final **for** loop runs once for each edge. Time is $\Theta(E)$

Total time: $\Theta(E \cdot V)$

Dijkstra : fonctionne seulement w > 0

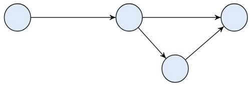
```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Running time Like Prim's dominated by operations on priority queue:

- If binary heap, each operation takes $O(\lg V)$ time $\Rightarrow O(E \lg V)$
- More careful implementation time is $O(V \lg V + E)$

SCC (graphes dirigés)

1er DFS, obtenir les finishing times



2ème DFS sur G^T, obtenir les SCC (trier les vertices par ordre de finishing **décroissants**).

connected components

run un DFS et changer le label à chaque fois qu'on passe dans la main loop

$$1/1 + 1/2 + 1/3 + 1/4 + 1/5 + \ldots + 1/n = H_n = \ln n + O(1)$$

n:th harmonic number

tree (u→**v**). *d[u] < d[v] < f[v] < f[u]*
première découverte de **v** lors du DFS.

back edge (u→**v**) *d[v] < d[u] < f[u] < f[v]*
v est gris — retour vers un ancêtre pas fini (peut former un cycle).

forward edge (u→**v**) *d[u] < d[v] < f[v] < f[u]*
v est noir et descendant — déjà entièrement visité.

cross edge (u→**v**) *d[v] < f[v] < d[u] < f[u]*
v est noir, ni ancêtre ni descendant — autre branche ou composante.

Theorem (Master Theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n).$$

Then, $T(n)$ has the following asymptotic bounds

- *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*
- *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$*
- *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$*

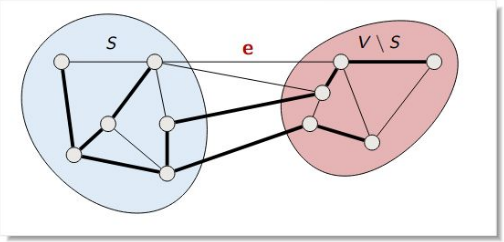
T(n) = T(n/5) + 2T(2n/5) + Θ(n). Educated guess O(n^a).
1 = (⅕)^a + 2(⅖)^a => a = 1, et comme on a f(n) = O(n^1) alors O(n log n).

PRE-ORDER →	Root	Left	Right
IN-ORDER →	Left	Root	Right
POST-ORDER →	Left	Right	Root

Consider a cut $(S, V \setminus S)$ and let

- T be a tree on S which is part of a MST
- e be a crossing edge of minimum weight

Then there is MST of G containing e and T



COUNTING-SORT(A, B, n, k)
let $C[0..k]$ be a new array
for $i = 0$ **to** k
  $C[i] = 0$
for $j = 1$ **to** n
  $C[A[j]] = C[A[j]] + 1$
for $i = 1$ **to** k
  $C[i] = C[i] + C[i-1]$
for $j = n$ **downto** 1
  $B[C[A[j]]] = A[j]$
  $C[A[j]] = C[A[j]] - 1$

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n+k)$

How big k is practical?

- 32-bit values? No
- 16-bit values? Probably not
- 8-bit values? Maybe depending on n
- 4-bit values? Probably unless n is very small

```
// on veut partitionner le tableau en deux parties
// cette fonction réorganise le tableau de telle sorte à ce que les éléments
// inférieurs ou égaux au pivot soient à gauche et les éléments supérieurs soient à droite
func partition(array []int, low, high int) int {

    // on choisit le dernier élément comme pivot (arbitrairement, plus tard ce sera random!)
    pivot := array[high]

    // i représentera le dernier élément de la zone contenant des éléments ≤ pivot
    i := low - 1

    // on parcourt le tableau et on compare chaque élément avec le pivot
    for j := low; j < high; j++ {
        // l'élément courant est inférieur ou égal au pivot!
        if array[j] <= pivot {
            // on agrandit la zone contenant des éléments ≤ pivot
            // donc i pointe maintenant vers l'élément juste en dehors de la zone
            i++
            // on échange l'élément courant avec cet élément juste en dehors de la zone
            // donc i pointe maintenenant correctement vers le dernier élément de la zone! (array[j]
            array[i], array[j] = array[j], array[i]
        }
    }

    array[i+1], array[high] = array[high], array[i+1] // on insert le pivot à sa place finale
    return i + 1                                       // la position du pivot
}

func quicksortRecursive(array []int, low, high int) {
    if low >= high {
        return
    }
    pivotIndex := betterPartitionRandomPivot(array, low, high)
    quicksortRecursive(array, low, pivotIndex-1)
    quicksortRecursive(array, pivotIndex+1, high)
}
```

Definition 2 *Assume the same as above, then we also call r the* competitive ratio *if*

$$ALG(I) \leq r \cdot OPT(I) + c$$

holds for all I and some constant c independent of the length of the sequence I. When c is 0, r is the strong competitive ratio.

$$T(n) = \begin{cases} \Theta(1) & \text{for } n < 50 \\ T(\frac{n}{4}) + T(\frac{3n}{4}) + O(\sqrt{n}) & \text{for } n \geq 50 \end{cases}$$

2b *(10 points)* **Prove** that $T(n) = O(n)$. You may simplify your calculations by assuming that $n/4, 3n/4$ and $\sqrt{n}$ always evaluate to integers.

In this problem you are asked to give a proof of $T(n) = O(n)$. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and next page.

We will prove that $T(n) \leq an - c\sqrt{n}$, for some constants a and c (which we will choose). Also, there exists a constant b such that $O(\sqrt{n}) \leq b\sqrt{n}$. We prove our claim using strong induction. The base case is clear by the formula for $n < 50$. Now, suppose that for all $k < n$, $T(k) \leq ak - c\sqrt{k}$. Now, we prove that $T(n) \leq an - c\sqrt{n}$. Note that

$$\begin{aligned} T(n) &= T(\tfrac{n}{4}) + T(\tfrac{3n}{4}) + O(\sqrt{n}) \\ &\leq \left(a\frac{n}{4} - c\sqrt{\frac{n}{4}}\right) + \left(a\frac{3n}{4} - c\sqrt{\frac{3n}{4}}\right) + b\sqrt{n} \\ &= an - c\sqrt{n}\left(\frac{1}{2} + \frac{\sqrt{3}}{2}\right) + b\sqrt{n}. \end{aligned}$$

Now, if we choose

$$c \geq \frac{b}{\frac{\sqrt{3}}{2} - \frac{1}{2}},$$

crazy complexity → dp ?