Union-Find : une gestion dynamique des composantes connexes

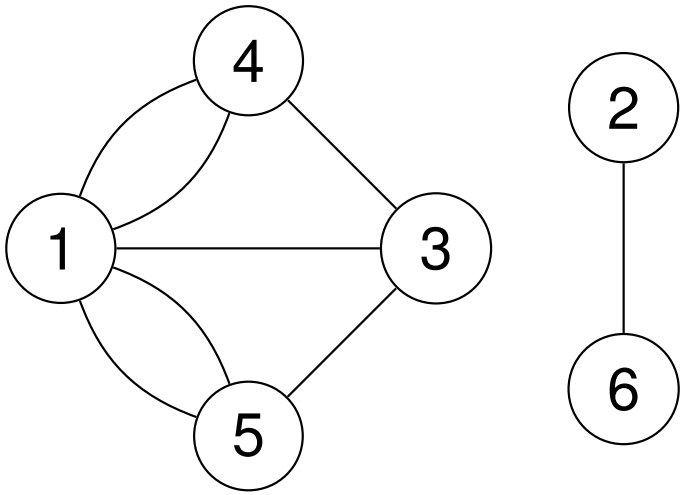
stephane@gonnord.org

Luminy - le 2 mai 2016

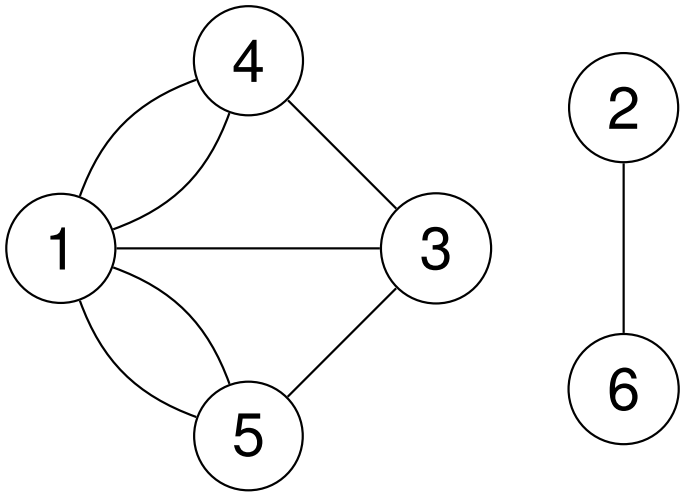
Plan

1. Connexité... et problèmes connexes
2. Des solutions classiques
3. Graphes aléatoires
4. Union-find

Questions autour de la connexité

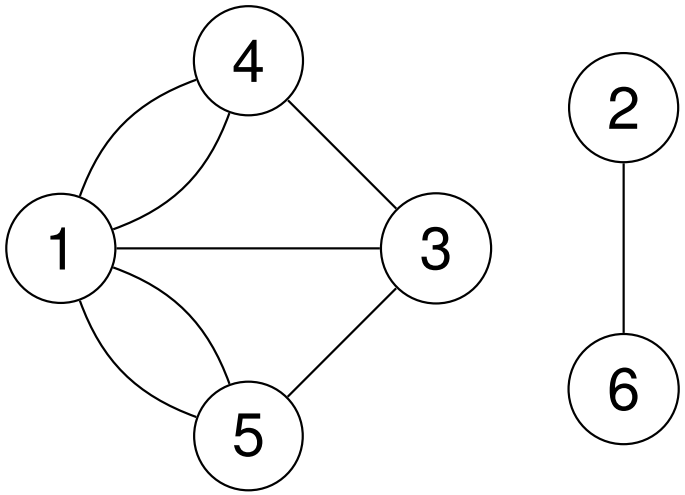


Questions autour de la connexité



- Classique :
  - $G = (\mathcal{S}, \mathcal{A})$ est-il connexe ?
  - Quelle est la composante de s_0 ?
  - Combien de composantes ?
  - (Distance entre deux sommets ?)

Questions autour de la connexité



- Classique :
  - $G = (\mathcal{S}, \mathcal{A})$ est-il connexe ?
  - Quelle est la composante de s_0 ?
  - Combien de composantes ?
  - (Distance entre deux sommets ?)
- Mais aussi :
  - Et si on place de nouvelles arêtes ?
  - Un graphe aléatoire est-il connexe ?

Diverses solutions

- Complexités... en fonction de quoi ?

Diverses solutions

- Complexités... en fonction de quoi ?
- Parcours en profondeur : $O(|\mathcal{A}|)$.
- Floyd-Warshall : $O(|\mathcal{S}|^3)$... pour toutes les distances !
- Dijkstra : $O(|\mathcal{A}|\ln(|\mathcal{A}|))$ pour les distances à un sommet.

Diverses solutions

- Complexités... en fonction de quoi ?
- Parcours en profondeur : $O(|\mathcal{A}|)$.
- Floyd-Warshall : $O(|\mathcal{S}|^3)$... pour toutes les distances !
- Dijkstra : $O(|\mathcal{A}|\ln(|\mathcal{A}|))$ pour les distances à un sommet.

MAIS...

Qu'est-ce qui se passe si Alice devient friend de Bob ?

Petite digression

- Un graphe aléatoire est-il connexe ?

Petite digression

- Un graphe aléatoire est-il connexe ?
- Heu... c'est quoi un graphe aléatoire ?

Petite digression

- Un graphe aléatoire est-il connexe ?
- Heu... c'est quoi un graphe aléatoire ?
- Seuil ?
  - Nécessaire : $\mathcal{A} \geq |\mathcal{S}| - 1$
  - Suffisant : $\mathcal{A} \geq \frac{(|\mathcal{S}|-1)(|\mathcal{S}|-2)}{2} + 1$

  Mais plus précisément ?

Petite digression

- Un graphe aléatoire est-il connexe ?
- Heu... c'est quoi un graphe aléatoire ?
- Seuil ?
  - Nécessaire : $\mathcal{A} \geq |\mathcal{S}| - 1$
  - Suffisant : $\mathcal{A} \geq \frac{(|\mathcal{S}|-1)(|\mathcal{S}|-2)}{2} + 1$

  Mais plus précisément ?
- Première approximation : $\mathcal{A} \simeq |\mathcal{S}| \ln |\mathcal{S}|$

Petite digression

- Un graphe aléatoire est-il connexe ?
- Heu... c'est quoi un graphe aléatoire ?
- Seuil ?
  - Nécessaire : $\mathcal{A} \geq |\mathcal{S}| - 1$
  - Suffisant : $\mathcal{A} \geq \frac{(|\mathcal{S}|-1)(|\mathcal{S}|-2)}{2} + 1$

  Mais plus précisément ?
- Première approximation : $\mathcal{A} \simeq |\mathcal{S}| \ln |\mathcal{S}|$
- Deuxième : $\mathcal{A} \simeq \frac{|\mathcal{S}| \ln |\mathcal{S}|}{2}$

Petite digression

- Un graphe aléatoire est-il connexe ?
- Heu... c'est quoi un graphe aléatoire ?
- Seuil ?
  - Nécessaire : $\mathcal{A} \geq |\mathcal{S}| - 1$
  - Suffisant : $\mathcal{A} \geq \frac{(|\mathcal{S}|-1)(|\mathcal{S}|-2)}{2} + 1$

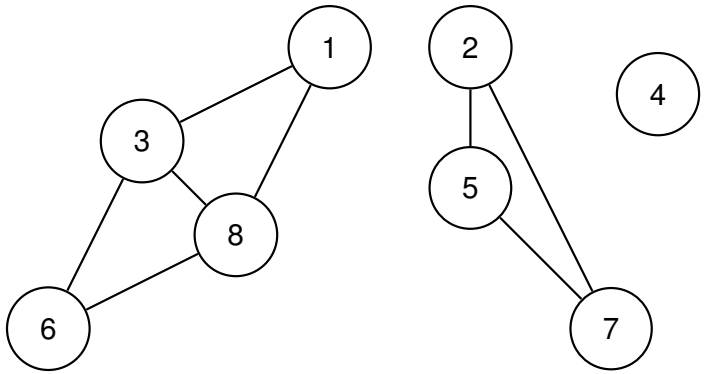
  Mais plus précisément ?
- Première approximation : $\mathcal{A} \simeq |\mathcal{S}| \ln |\mathcal{S}|$
- Deuxième : $\mathcal{A} \simeq \frac{|\mathcal{S}| \ln |\mathcal{S}|}{2}$
- Troisième :

$$\mathcal{A} = \frac{|\mathcal{S}| (\ln |\mathcal{S}| + C)}{2} \quad \Longrightarrow \quad \mathbb{P}(\textit{Connexe}) \simeq \mathrm{e}^{-\mathrm{e}^{-C}}$$

Union-Find : la structure

Grande idée : désigner un délégué de classe !

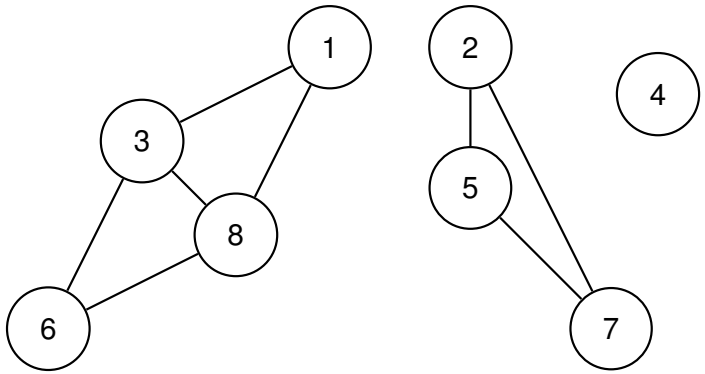
- Le graphe :



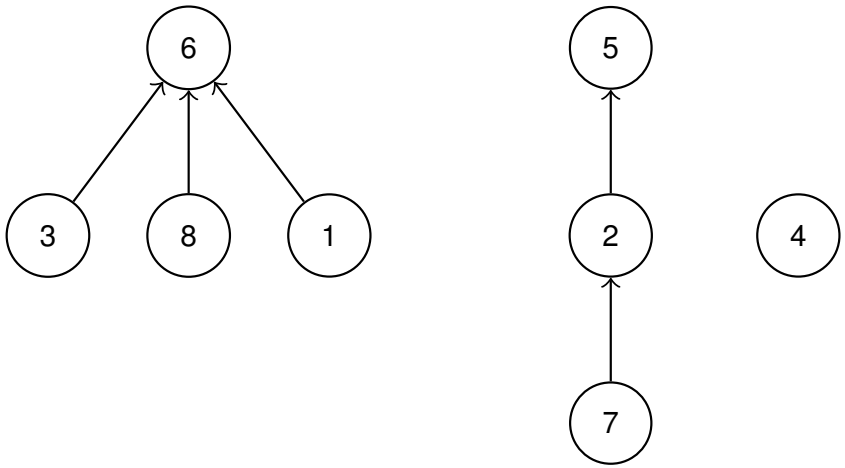
Union-Find : la structure

Grande idée : désigner un délégué de classe !

- Le graphe :



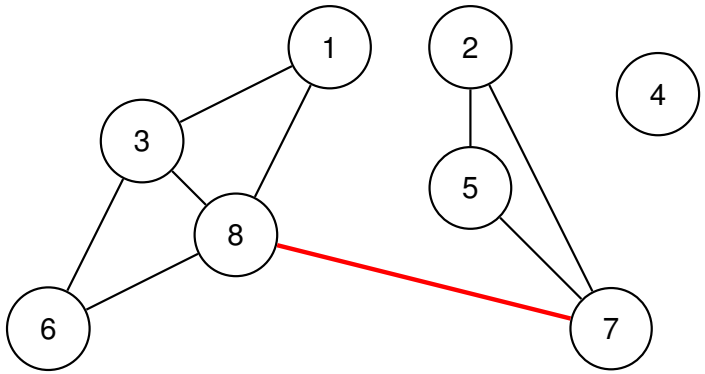
- Une forêt associée :



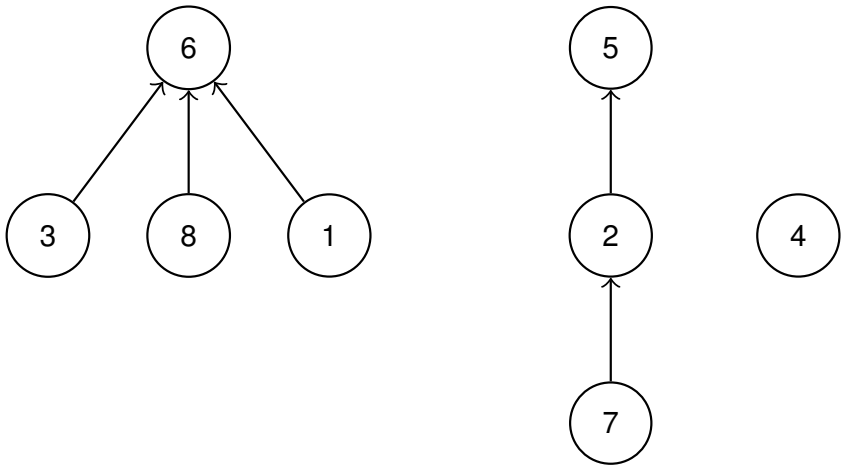
Union-Find : la structure

Grande idée : désigner un délégué de classe !

- Le graphe :



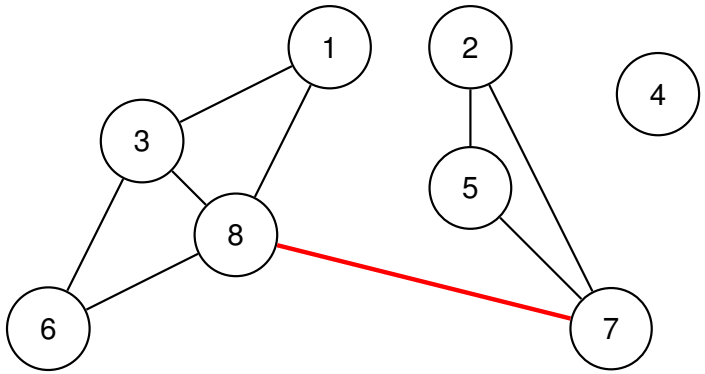
- Une forêt associée :



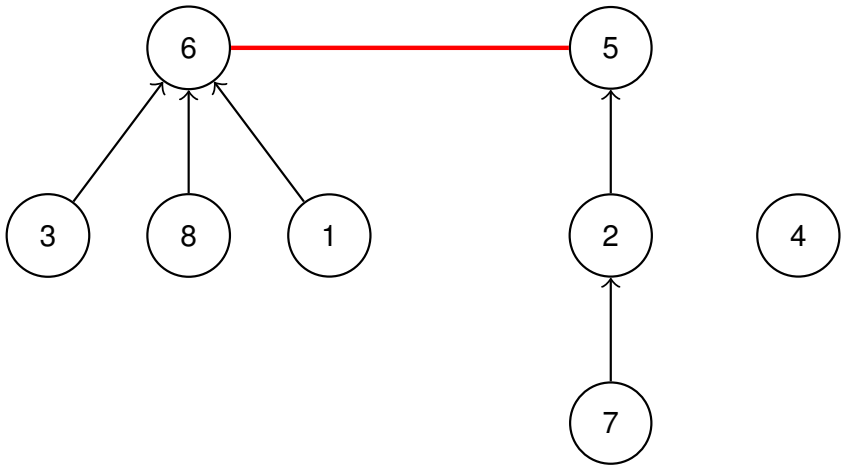
Union-Find : la structure

Grande idée : désigner un délégué de classe !

- Le graphe :



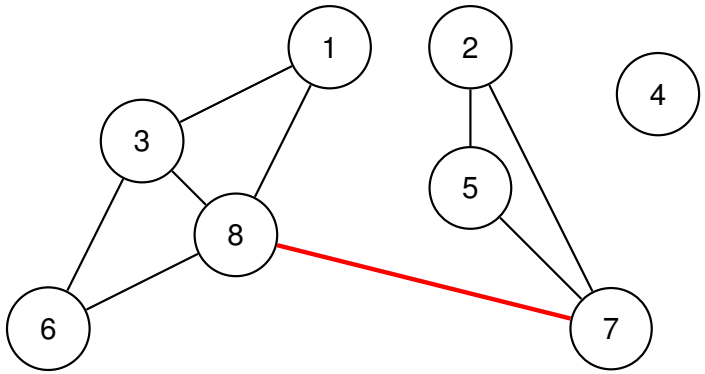
- Une forêt associée :



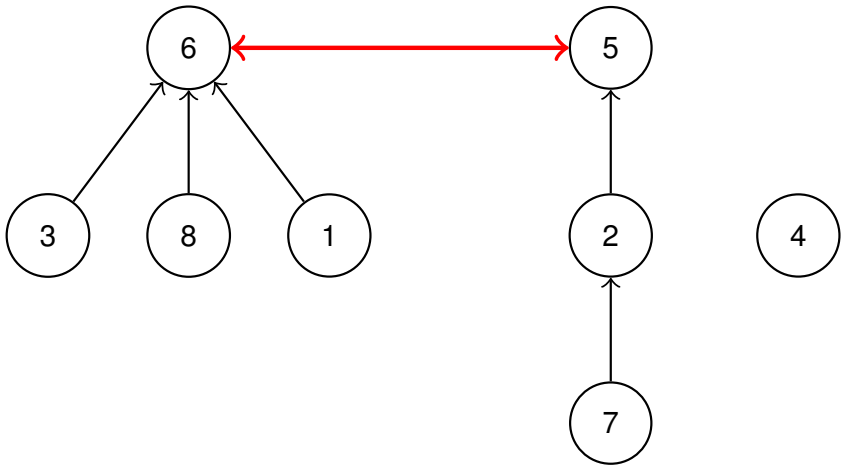
Union-Find : la structure

Grande idée : désigner un délégué de classe !

- Le graphe :



- Une forêt associée :



Union-Find : les deux étapes

- Find : retrouver l'ancêtre.

$$O(h) = O(|\mathcal{S}|)$$

Union-Find : les deux étapes

- Find : retrouver l'ancêtre.

$$O(h) = O(|\mathcal{S}|)$$

- Union : relier un ancêtre à l'autre.

Union-Find : les deux étapes

- Find : retrouver l'ancêtre.

$$O(h) = O(|\mathcal{S}|)$$

- Union : relier un ancêtre à l'autre.
- DRAME : pour n sommets successifs, $O(n^2)$.

Union-Find : de n^2 à $n \ln n$... à $n\alpha(n)$

- Première amélioration : petite classe greffée sur la grosse.
  - $h = O(\ln n)$
  - Besoin d'une information supplémentaire.
  - Hauteur ? Poids ?

Union-Find : de n^2 à $n \ln n$... à $n\alpha(n)$

- Première amélioration : petite classe greffée sur la grosse.
  - $h = O(\ln n)$
  - Besoin d'une information supplémentaire.
  - Hauteur ? Poids ?
- Deuxième amélioration : tronquer les chemins !
  - $h = O(\alpha_n)$.
  - en pratique : $h \leq 5$ (largement !)

Union-Find : le code

```
def foret_vierge(n): # nombre de sommets, puis [père, poids]
    return [n] + ([[-1, 1]] * n)
```

Union-Find : le code

```
def foret_vierge(n): # nombre de sommets, puis [père, poids]
    return [n] + ([[-1, 1]] * n)


def find(s, f): # donne l'ancêtre du sommet s dans la foret f
    courant = s
    while f[courant][0] != -1:
        courant = f[courant][0]
    return courant
```

Union-Find : le code

```
def foret_vierge(n): # nombre de sommets, puis [père, poids]
    return [n] + ([[-1, 1]] * n)

def find(s, f): # donne l'ancêtre du sommet s dans la foret f
    courant = s
    while f[courant][0] != -1:
        courant = f[courant][0]
    return courant

def union(s1, s2, f):
    a1, a2 = find(s1, f), find(s2, f)
    if a1 != a2:
        f[0] -= 1
        p1, p2 = f[a1][1], f[a2][1] # les poids
        if (p1 < p2) or (p1 == p2 and randint(0, 1) == 0):
            gros, petit = a2, a1
        else:
            gros, petit = a1, a2
        f[gros] = [-1, f[a1][1] + f[a2][1]]
        f[petit] = [gros, 0] # le poids n'a plus d'importance
```

Voila, c'est fini

Merci de votre attention !

Voila, c'est fini

Merci de votre attention !