Devoir maison n°3

Corrigé

⋆ ⋆ ⋆

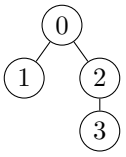
1 Algorithme de Kruskal

Question 5 On trouve respectivement 38 307 et 2 240 681.

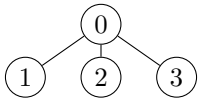
Question 9 On peut montrer ce résultat avec $n = 4$. On considère les opérations dans cet ordre :

- unir 0 et 1 ;
- unir 2 et 3 ;
- unir 0 et 2 ;
- trouver 3.

En supposant que lors d'un cas d'égalité de rangs lors d'une union, le représentant du premier devient parent du représentant de l'autre, après la troisième union, la partition est représentée par :



et est de rang 2 (ce qui correspond à sa hauteur). Après l'opération trouver, étant donné la compression des arbres, le rang ne change pas, mais l'arbre devient :



qui est de hauteur 1, mais de rang 2.

Question 10 Montrons par induction que la classe d'équivalence X d'un arbre de rang r vérifie $|X| \geqslant 2^r$:

- si $r = 0$, alors X est un singleton et vérifie $|X| = 1 = 2^r$;
- supposons le résultat établi pour X_1 et X_2 de rangs respectifs r_1 et r_2 et soit X la classe obtenue par union de X_1 et X_2. Sans perte de généralité, supposons $r_1 \geqslant r_2$. On distingue :
  - si $r_1 > r_2$, alors $rg(X) = r_1$ et $|X| = |X_1| + |X_2| \geqslant |X_1| \geqslant 2^{r_1} = 2^{rg(X)}$;
  - si $r_1 = r_2$, alors $rg(X) = r_1 + 1$ et $|X| = |X_1| + |X_2| \geqslant 2|X_2| \geqslant 2 \times 2^{r_2} = 2^{r_2+1} = 2^{rg(X)}$.

On conclut par induction.

Dès lors, il suffit de remarquer que le rang est un majorant de la hauteur : c'est le cas initialement, et cette propriété est conservée lors d'une union (par le principe de l'incrémentation lors de l'union de deux classes de même rang) et lors d'une recherche (car le rang n'est pas modifié et la hauteur peut diminuer). On en déduit le résultat attendu.

Question 13 On trouve :

- `g_1000_ad` : 18 935 ;
- `g_1000_sd` : 1 064 465 ;
- `g_50000_ad` : 1 840 669 ;
- `g_50000_sd` : 1 364 763 779.

2 Algorithme de Borůvka

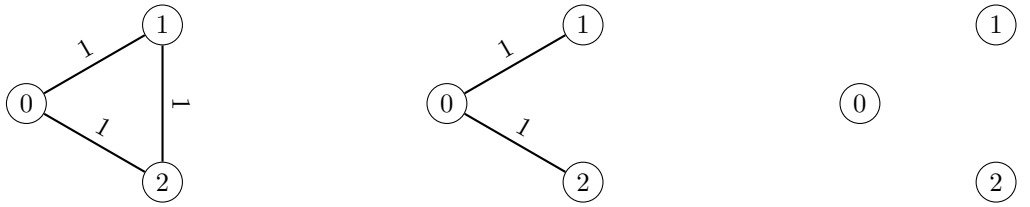
Question 18 On trouve bien les valeurs précédentes.

2.1 Arêtes inutiles et arêtes sûres

Question 19 Ajouter une arête inutile à B créerait un cycle dans T^*. H ne contient donc aucune arête inutile.

Question 20 Soit C une composante connexe de H. Soit $a = \{s, t\}$ l'arête de poids minimal parmi les arêtes qui ont une extrémité dans C. Supposons que l'unique arbre couvrant minimal $T^* = (S, B^*)$ ne contient pas a. T^* étant connexe, il contient un chemin de s à t. Ce chemin contient une arête a^* entre un sommet de C et un sommet qui n'est pas dans C. Nécessairement $f(a^*) > f(a)$ car a est une arête sûre. Dès lors $T' = (S, B^* \cup \{a\} \setminus \{a^*\})$ est un arbre couvrant (il est connexe, car il existe toujours un chemin entre deux sommets, quitte à passer par a, et il contient $|S| - 1$ arêtes) de G dont le poids est strictement inférieur à celui de T^*, ce qui contredit la minimalité de T^*.

Question 21 On considère les graphes G, T et H suivants, dans cet ordre :



Ici, toutes les arêtes de G sont des arêtes sûres pour H, mais de manière évidente, elles ne sont pas toutes dans T.

Question 24 On remarque que la fonction `aretes_sures` se contente de parcourir toutes les listes d'adjacence du graphe une seule fois, et fait des opérations en temps constant pour chaque arête ainsi étudiée (grâce au tableau `cc`, on peut tester efficacement si une arête est entre deux composantes connexes). La complexité totale est donc en $\mathcal{O}(|S| + |A|)$.

2.2 Algorithme de Borůvka

Question 25 L'algorithme termine bien, car l'ajout à B d'une arête sûre pour H diminue strictement le nombre de composantes connexes de H. Ce nombre est donc un variant de boucle, ce qui montre bien la terminaison.

De plus, la propriété « H est acyclique » est un invariant de boucle. En effet, l'ajout d'une arête sûre ne peut pas créer de cycle. Par ailleurs, l'ensemble des arêtes sûres (avec la deuxième définition) ne peut pas créer de cycle non plus. En effet, supposons que $C_1, ..., C_k$ sont des composantes connexes, et qu'en rajoutant $a_1, ..., a_k$ qui sont des arêtes sûres, chaque a_i étant une arête entre C_i et C_{i+1} (où on assimile C_{k+1} à C_1), on crée un cycle. Sans perte de généralité, supposons que a_1 est l'arête minimale pour $\prec$. Alors soit a_2 soit a_k n'aurait pas dû être une arête sûre.

Finalement, lorsque l'algorithme termine, le graphe H est connexe (sinon il possède des arêtes sûres). C'est donc un arbre couvrant de G.

Montrons que c'est bien un arbre couvrant de poids minimum. On a déjà montré en questions 19 et 20 que c'est un arbre couvrant minimum si la fonction de pondération est injective. Si elle ne l'est pas, on crée une nouvelle fonction de pondération.

On suppose que $A = \{a_1, \ldots, a_m\}$, où $a_1 \prec a_2 \prec \ldots \prec a_m$. On pose $g(a_i) = (f(a_i), i)$. Ainsi, on a, pour $a, a' \in A$, $a \prec a'$ si et seulement si $g(a) < g(a')$. La fonction g est donc une fonction de pondération injective.

On remarque que si T est un arbre couvrant de poids minimal pour (S, A, g), alors T est un arbre couvrant de poids minimal pour (S, A, f). En effet, s'il existe un arbre couvrant T^* tel que $f(T^*) < f(T)$, alors $g(T^*) < g(T)$ (on utilise l'ordre lexicographique pour comparer les poids de la fonction g).

Cela permet de conclure que l'algorithme de Borůvka (qui renvoie un ACM pour (S, A, g)) renvoie bien un arbre couvrant de poids minimum, même si f n'est pas injective.

Question 27 On retrouve bien les mêmes valeurs qu'avec Kruskal.

Question 28 On remarque que le nombre de composantes connexes est au moins divisé par deux à chaque passage dans la boucle **Tant que**. En effet, à chaque composante connexe est associée une arête sûre. Une arête sûre, elle, peut être associée à au plus deux composantes connexes. Il y a donc au moins $\frac{m}{2}$ arêtes sûres à chaque itération, m étant le nombre de composantes connexes.

On en déduit qu'il y a au plus $\log |S|$ passages dans la boucle. Chaque passage dans la boucle fait le calcul des composantes connexes, puis des arêtes sûres, puis de l'ajout des arêtes au graphe H. Toutes ces opérations se font en temps $\mathcal{O}(|S| + |A|)$. La complexité totale est donc en $\mathcal{O}((|S| + |A|) \log |S|) = \mathcal{O}(|A| \log |S|)$, comme l'algorithme de Kruskal.

En pratique, cette complexité n'est pas forcément atteinte, car il s'agit du pire cas, où il y a exactement $\frac{m}{2}$ arêtes sûres à chaque itération. Cependant, il est possible d'avoir plus que ce nombre, qui est le cas défavorable. En pratique, l'algorithme de Borůvka est donc plus rapide que l'algorithme de Kruskal (ça se confirme si on fait un grand nombre de tests).

⋆ ⋆ ⋆