Gestion dynamique d'une partition

Résumé

Nous nous intéressons au problème suivant : un ensemble E fini étant fixé, comment gérer une partition de E qui évolue au cours du temps ? On veut à chaque instant savoir si deux éléments a et b de E sont dans la même classe ; et pouvoir fusionner les classes de deux éléments a et b de E, si elles sont distinctes.

Table des matières

1 Un exemple introductif **2**

2 Arbre recouvrant minimal **3**

3 Une implémentation naïve **4**

4 Première amélioration **5**

5 Utilisation d'arbres **5**

6 Fusion avec pondération **6**

7 Fusion avec compression de chemins **6**

1 Un exemple introductif

Considérons un archipel dont les îles sont, initialement, isolées. Petit à petit, des liaisons maritimes sont ouvertes. À chaque moment, les liaisons maritimes définissent une partition de l'archipel : chaque classe contient un ensemble d'îles deux à deux reliées, directement ou indirectement. Avec le point de vue de la théorie des graphes, ce sont les composantes connexes du graphe dont les sommets sont les îles et dont les arêtes sont les liaisons maritimes.

L'ouverture d'une nouvelle liaison entre deux îles peut fusionner deux classes, dans le cas où ces deux îles n'étaient pas dans la même composante connexe. Notons que nous ne considérerons pas d'éventuelles fermetures de liaisons ; par conséquent, le nombre de classes ne peut que diminuer avec le temps.

Les figures 1 à 6 décrivent la suite des ouvertures de liaisons. Le tableau 1 résume l'évolution de la partition correspondante. Remarquons que le nombre de fusions ne peut dépasser $|E| - 1$, où E est l'archipel (ensemble des îles).

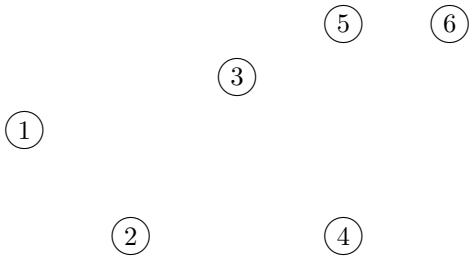

Figure 1: l'archipel, avant l'invention de la navigation

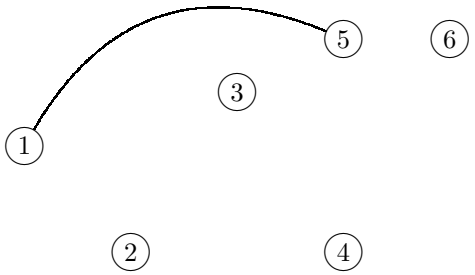

Figure 2: après ouverture d'une liaison entre 1 et 5

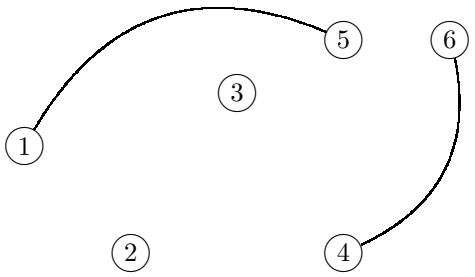

Figure 3: ouverture d'une liaison entre 4 et 6

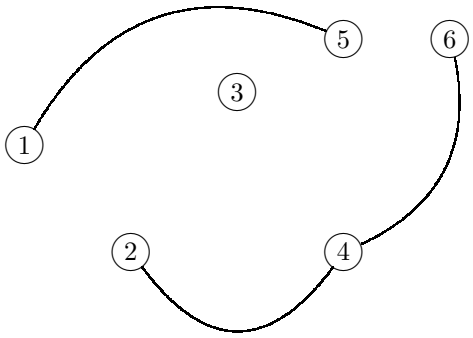

Figure 4: ouverture d'une liaison entre 2 et 4

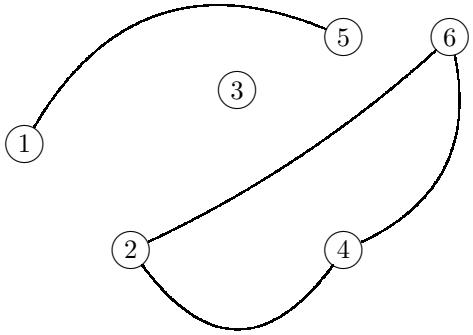

Figure 5: ouverture d'une liaison entre 2 et 6

Appelons *consultation* l'opération consistant à déterminer si deux éléments sont dans la même classe, et *fusion* l'opération qui réunit deux classes en une seule. Dans toute la suite, on note $n = |E|$; on effectue (dans un ordre quelconque) une suite de q consultations et au plus $n - 1$ fusions. On s'intéresse au coût total de cette suite d'opérations.

au départ	$\{\{1\};\{2\};\{3\};\{4\};\{5\};\{6\}\}$
après ouverture de 1—5	$\{\{1,5\};\{2\};\{3\};\{4\};\{6\}\}$
après ouverture de 4—6	$\{\{1,5\};\{2\};\{3\};\{4,6\}\}$
après ouverture de 2—4	$\{\{1,5\};\{2,4,6\};\{3\}\}$
après ouverture de 2—6	$\{\{1,5\};\{2,4,6\};\{3\}\}$
après ouverture de 5—6	$\{\{1,5,2,4,6\};\{3\}\}$

Tableau 1: évolution de la partition

2 Arbre recouvrant minimal

Un *arbre* est un graphe non orienté connexe et sans cycle : deux sommets quelconques sont reliés par un chemin et un seul. Le choix d'un sommet permet d'*enraciner* l'arbre, en orientant les chemins à partir de la racine ; en général, on dessine un tel arbre avec la racine en haut. La *hauteur* d'un arbre t, notée $h(t)$, est la longueur maximale d'un chemin issu de la racine ; la *taille* de l'arbre, notée $s(t)$, est le nombre de sommets de t. Une *forêt* est un ensemble d'arbres deux à deux disjoints.

La recherche d'un arbre recouvrant minimal est un problème classique : on se donne un graphe (S, A) connexe non orienté, dont les arêtes sont *valuées*, c'est-à-dire étiquetées par des nombres positifs. On voudrait choisir un sous-ensemble d'arêtes conservant la connexité, et minimisant la somme des valuations.

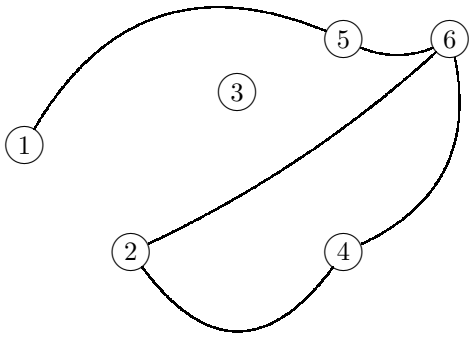

Figure 6: ouverture d'une liaison entre 5 et 6

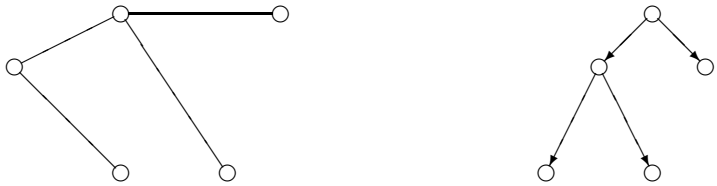

Figure 7: un arbre non enraciné, un arbre enraciné

L'algorithme de KRUSKAL consiste à construire une suite de forêts ; initialement, chaque arbre se réduit à un sommet. À chaque étape, on choisit l'arête de valuation minimale : si elle n'introduit pas de cycle (ce qui revient à dire qu'elle joint deux sommets qui ne sont pas dans le même arbre), on ajoute cette arête, ce qui revient à fusionner deux des arbres de la forêt. Au bout de $|S| - 1$ étapes, on obtient un arbre recouvrant, et il est facile de montrer qu'il est minimal.

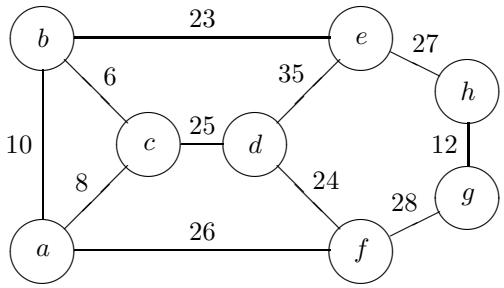

Figure 8: un graphe valué

Suggestion : déroulez l'alghorithme de KRUSKAL sur l'exemple de la figure 8. Montrez que, lorsque les valuations des arêtes sont deux à deux distinctes, il n'existe qu'un arbre recouvrant minimal. Construisez un exemple où il existe plusieurs arbres recouvrants minimaux.

3 Une implémentation naïve

Une relation d'équivalence sur un ensemble E de cardinal n (dans la suite, $E = [\![1, n]\!]$) est un cas particulier de relation ; elle peut donc être représentée par une matrice carrée R d'ordre n, avec la convention $R_{i,j} = 1$ ssi i et j sont équivalents. Cette structure de données requiert n^2 cellules.

Avec cette structure de données, la consultation se fera en un temps $\mathcal{O}(1)$. En revanche, lorsque l'on demande de fusionner les classes de i et j, il faudra prendre en compte la transitivité, ce qui amène à balayer les colonnes d'indices i et j : par exemple, il faut mettre mettre à 1 les $R_{j,k}$ tels que $R_{i,k} = 1$; le coût de la fusion de deux classes est donc de l'ordre de $4n$.

La figure 9 montre la suite des matrices décrivant l'évolution de la partition.

$$\begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0&1&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 1&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0&1&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&1\\ 1&0&0&0&1&0\\ 0&0&0&1&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0&1&0\\ 0&1&0&1&0&1\\ 0&0&1&0&0&0\\ 0&1&0&1&0&1\\ 1&0&0&0&1&0\\ 0&1&0&1&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0&1&0\\ 0&1&0&1&0&1\\ 0&0&1&0&0&0\\ 0&1&0&1&0&1\\ 1&0&0&0&1&0\\ 0&1&0&1&0&1 \end{pmatrix} \begin{pmatrix} 1&1&0&1&1&1\\ 1&1&0&1&1&1\\ 0&0&1&0&0&0\\ 1&1&0&1&1&1\\ 1&1&0&1&1&1\\ 1&1&0&1&1&1 \end{pmatrix}$$

Figure 9: suite des matrices décrivant l'évolution de la partition

Suggestion: : quelle modification simple divise par 2 l'espace mémoire requis et les coûts des calculs ?

4 Première amélioration

Supposons que l'on décide de choisir un représentant dans chaque classe. On peut utiliser un vecteur v de n cellules, v_i étant le représentant de la classe de i. Initialement, $v_i = i$ pour tout $i \in [\![1, n]\!]$.

Chaque consultation a un coût $\mathcal{O}(1)$ puisqu'elle se réduit à la comparaison de v_i et v_j.

Pour fusionner les classes de i et j, on commence par vérifier que $v_i \neq v_j$ (en cas d'égalité, il n'y a aucun travail à effectuer) ; ensuite, pour chaque $k \in [\![1, n]\!]$, on donne à v_k la valeur de v_i, si $v_k = v_j$. Le coût d'une fusion est donc de l'ordre de n.

au départ	1	2	3	4	5	6
après ouverture de 1—5	1	2	3	4	1	6
après ouverture de 4—6	1	2	3	4	1	4
après ouverture de 2—4	1	2	3	2	1	2
après ouverture de 2—6	1	2	3	2	1	2
après ouverture de 5—6	1	1	3	1	1	1

Tableau 2: suite des vecteurs décrivant l'évolution de la partition

5 Utilisation d'arbres

On décide de représenter chaque classe par un arbre, le représentant étant la racine. Notons r_i la racine de l'arbre contenant i. Pour savoir si i et j sont dans la même classe, il suffit de comparer r_i et r_j. Pour fusionner les classes de i et j (lors de l'ouverture de la liaison i—j), on rattachera r_j à r_i.

La taille de la structure de données est un $\mathcal{O}(n)$. On peut par exemple la réaliser avec un vecteur p : p_i est le père de i. La racine d'un arbre est son propre père. Le tableau 3 montre la suite des forêts décrivant l'évolution de la partition.

Le coût des deux opérations (fusion et test) est un $\mathcal{O}(n)$.

au départ	1	2	3	4	5	6
après ouverture de 1—5	1	2	3	4	1	6
après ouverture de 4—6	1	2	3	4	1	4
après ouverture de 2—4	1	2	3	2	1	4
après ouverture de 2—6	1	2	3	2	1	4
après ouverture de 5—6	1	1	3	2	1	4

Tableau 3: suite des forêts décrivant l'évolution de la partition

Suggestion: : donnez un exemple de suite de $n - 1$ fusions, dont le coût total est un $\Omega(n^2)$.

6 Fusion avec pondération

On améliore la performance en décidant, à chaque fusion, de rattacher l'arbre de plus petite taille à la racine de l'autre. Pour ce faire, on associe à chaque arbre t sa taille $s(t)$; lors de la fusion de deux arbres t et t', la taille du nouvel arbre est $s(t) + s(t')$. La figure 10 illustre ceci.

On montre facilement qu'au cours d'une suite de fusions, chaque arbre t vérifie la relation $h(t) \leqslant \lg\big(s(t)\big)$. Le coût d'une suite de q consultations est donc un $\mathcal{O}(n + q \lg n)$.

Suggestion : montrez cette majoration.

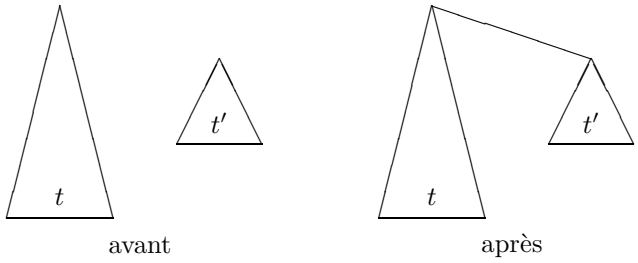

Figure 10: fusion avec pondération

7 Fusion avec compression de chemins

Toute fusion nécessite une consultation préalable, pour savoir si les deux classes à fusionner sont effectivement différentes. L'idée de la compression de chemins est la suivante : lors de chaque consultation, on rattache tous les sommets visités à la racine ; ceci peut diminuer la hauteur de l'arbre, donc diminuer les coûts des consultations ultérieures. La figure 11 illustre cette idée.

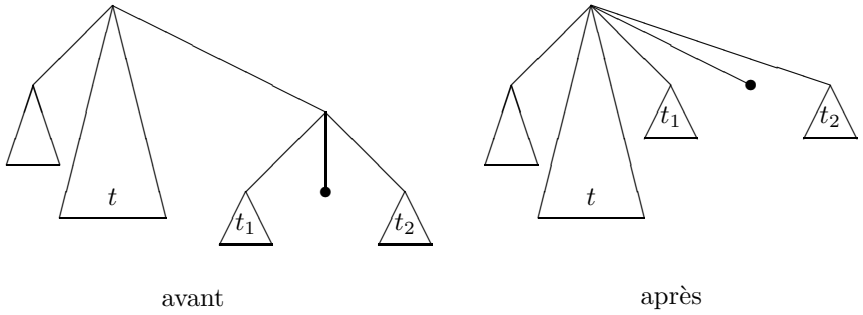

Figure 11: fusion avec compression

L'analyse de cet algorithme a été réalisée par Tarjan ; il a montré que le coût de q consultations était un $\mathcal{O}\big((q+n) \lg^* n\big)$, où $\lg^*$ est l'inverse fonctionnelle de lg, définie comme suit : $\lg^* k$ est le plus petit nombre de 2 qu'il faut ≪empiler≫ pour atteindre ou dépasser k. Par exemple, $\lg^* 65536 = 4$ car $2^{2^2} = 2^4 = 16$ et $2^{2^{2^2}} = 2^{2^4} = 2^{16} = 65536$.

FIN