Picot
Rodolphe

Rapport de stage

Sujet : Enumération parallèle d'un monoïde généralisant le tri à bulle

Sommaire

Introduction

J'ai effectué mon stage au Laboratoire de Recherche en Informatique situé sur le plateau de Saclay. Il fait partie de l'université Paris-Sud et du CNRS. Ce laboratoire a été fondé il y a 35 ans et est composé de 9 équipes totalisant plus de 250 personnes dont 133 permanents et 90 doctorants.

On y effectue des recherches dans de nombreux domaines non seulement pratiques mais théoriques tel l'algorithmique, la vérification de preuves ou le réseau.

J'ai personnellement fait mon stage dans l'équipe GALaC qui est spécialisée dans tout ce qui a attrait à la combinatoire, à l'algorithmique, à la théorie des graphes et aux systèmes en réseaux et distribués.

Cette équipe est composée de 18 membres permanents ainsi qu’une dizaine de doctorants.

Problèmes d'énumération

Introduction aux problèmes d'énumération :

Mon stage étant orienté combinatoire, en voici une définition :

« La combinatoire est un domaine des mathématiques qui étudie les configurations d'ensembles finis d'objets ainsi que les dénombrements ». Un des paramètres à prendre en compte en combinatoire est la taille des objets rencontrés. Par exemple, si on prend l'ensemble des permutations des nombres entiers entre 1 et n , on obtient un ensemble constitué de $n!$ éléments. On considère souvent des ensembles infinis dont les objets ont une notion de taille.

Définition : on appelle classe combinatoire un ensemble X fini ou dénombrable sur lequel nous avons une fonction de taille tel que :

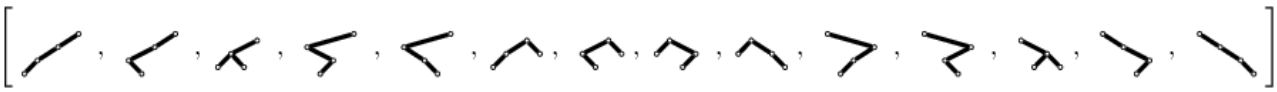
1. taille(x) est un entier pour un élément x de X
2. le nombre d'éléments de taille n est fini

Nous avons besoin, pour pouvoir travailler sur des objets de tailles conséquentes, de « ranger » les éléments d'un ensemble, de leur donner un ordre. Pendant ce stage, nous avons donc donné un ordre pour chaque ensemble *e* étudié pour pouvoir faire des méthodes telles que :

- count, qui doit retourner la cardinalité de *e*
- list, quit retourner la liste de tous les objets de *e* selon l'ordre prédéfini
- rank(*o*), qui doit retourner la position dans la liste d'un objet *o* appartenant à *e* selon l'ordre prédéfini
- unrank(*i)* qui doit retourner le $i^{ème}$ objet de l'ensemble *e* selon l'ordre prédéfini
  Note: Avec unrank et count, il est facile de faire list, il suffit d'appeler unrank pour tout i allant de 0 à count(e).

Voyons cela avec un exemple concret. Prenons le treillis de Tamari qui est l'ensemble des arbres de taille n . Pour le reste de l'exemple, nous choisissons l'ordre lexicographique :

- On range les arbres en fonction de la taille de leur sous arbre gauche, pour les arbres ayant leur sous arbre gauche de même taille, on applique récursivement cette définition.



Ensemble des arbres de taille 4 ordonnés dans l'ordre lexicographique

Il est bien connu qu'il y a 14 éléments puisque son nombre d'éléments suit la suite que l'on nomme les nombres de Catalan. Cette suite est définie de la façon suivante :

$$C_n = \frac{(2n)!}{(n+1)!\,n} = \sum_{i=0}^{n-1} C_i * C_{n-1}$$

Nous avons donc ici la cardinalité du treillis de Tamari c'est à dire la méthode « count », mais pour pouvoir travailler desssus nous avons besoin de trier ses éléments, de leur donner un rang.

Rank/Unrank :

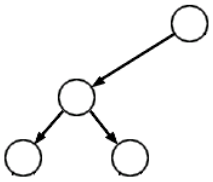
Nous avons calculé le rang d'un arbre de taille n de la manière suivante : on regarde le nombre *g* de nœuds se trouvant à gauche. Le **rang minimum d'un arbre *a*** est le nombre d'arbres ayant un sous arbre gauche plus grand que celui de *a*. On trouve alors son rang minimum par le calcul suivant :

$$\sum_{i=g}^{n-1} C_i * C_{k-i}$$

On a donc le rang d'un arbre en appelant récursivement cette méthode sur son sous arbre gauche et droit en faisant

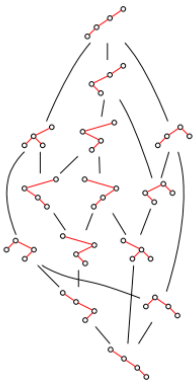
$$rang = min + rank(fils\ droit) + C_d * rank(fils\ gauche)$$

Par exemple :



Prenons le premier nœud, il possède 3 nœuds à gauche, il n'a donc pas de nœud à doite puisque nous sommes dans le treillis de Tamari d'ordre 4. Son rang minimum est 0 pour l'instant car aucun arbre ne peut posséder un sous arbre gauche plus grand que lui. On rappelle cette méthode sur son fils gauche qui a 1 nœud à gauche, il en a donc 1 à droite, son rang minimum est de 2 car $C_2 = 2$. Les fils suivant étant des nœuds finaux, on retourne le minimum c'est à dire 2, cet arbre a donc le rang 2 dans notre treillis de Tamari. On remarque que chaque arbre a un rang compris entre 0 et $C_n - 1$.

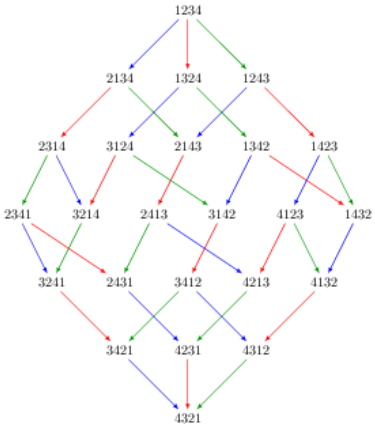
Pour calculer la méthode d'unranking il suffit d'appliquer l'algorithme inverse. Nous avons aussi codé les rotations gauche et droite pour pouvoir passer d'un arbre à un autre qui sont en relation. Voici une réprésentation des arbres de taille 4 reliés entre eux :



Permutohèdre

Définition et première approche :

Après avoir travaillé sur le treillis de Tamari, mon stage a consisté à étudier le permutohèdre d'ordre n. Le permutohèdre d'ordre n est un graphe dont les sommets sont les permutations de 1 à n. Les arrêtes de ce graphe sont données par les transpositions élémentaires. Une **transposition élémentaire** est le fait d'échanger deux entiers adjacents. Voici le permutohèdre d'ordre 4 :



Contrairement au treillis de Tamari, nous voyons que ce graphe est étagé. Chaque étage i est composé par les permutations ayant subi i transpositions élémentaires. Nous nous sommes intéressés à des méthodes de ranking/d'unranking.

Tout d'abord, nous avons cherché une méthode pour écrire les permutations permettant de mieux concevoir les algorithmes et nous avons décidé d'utiliser le code de Lehmer. Pour une permutation p de taille n, le code de Lehmer de p est le tableau de taille n dont la $i^{ème}$ case contient le nombre d'entiers à droite inférieurs au $i^{ème}$ entier de p.

Par exemple pour la permutation *3412*, son code de Lehmer sera *2200*. On remarquera que ce code nous donne l'étage de la permutation par la somme de chaque entier qui le compose, en effet $2+2+0+0=4$, ce qui correspond à l'étage de la permutation *3412*.

Avec le code de Lehmer, nous avons utilisé le triangle de Mac Mahon, qui se réfère à la fonction $M(r,c)$ où r et c sont des entiers, r représentant la ligne et c la colonne, pour nos méthodes de ranking/unranking, voici la définition de ce triangle :

M(0, 0) = 1,
M(0, c) = 0 pour $c \neq 0$
M(r, c) = $\sum_{k=0}^{r} M(r-1, c-k)$ pour r >0

Ce qui donne ce triangle :

r\c	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	2	1							
3	1	3	5	6	5	3	1				
4	1	4	9	15	20	22	20	15	9	4	1

$M(r,c)$ compte le nombre de permutations de taille $r-1$ dans l'étage c.

Rank/Unrank :

Avec le triangle de Mac Mahon et le code de Lehmer, la méthode de ranking devient alors naturelle avec la formule suivante :

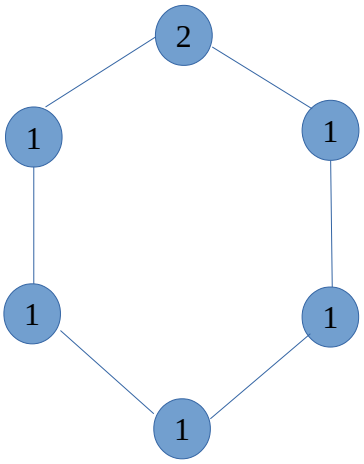
$$rang = \sum_{i=1}^{n-1} M\left(i-1, \sum_{j=0}^{i-1} l_{n-1-j}\right)$$

avec l_j le $j^{ème}$ élément du code de Lehmer. Comme pour le treillis de Tamari, il suffit d'appliquer la méthode inverse pour avoir celle d'unranking.

Enumération de chemins :

Nous nous sommes aussi intéressés au nombre de chemins entre la permutation identité et la permutation finale, pour ce faire nous avons fait un algorithme parcourant le graphe du permutohèdre étage par étage.

L'algorithme démarre de la permutation par une des deux extrémités du graphe et lui attribue une valeur de 1. Puis, pour accéder à l'étage suivant, il suffit de le calculer (nous savons le faire grâce au ranking) et d'additionner les valeurs des pères aux fils. Récursivement, nous arrivons à l'autre extrémité du graphe. Si on prend le permutohèdre d'ordre 3, nous obtenons ce graphe :



Il y a donc 2 chemins pour aller d'une extrémité à l'autre dans le permutohèdre d'ordre 3, les résultats pour ce calcul pour n allant de 1 à 6 sont :

1,2,16,768,292864,1100742656

Cette suite est la suite dont le $i^{ème}$ *terme est le nombre de chaînes maximales dans le poset des chemins de Dyck. (Suite* A005118 *de l'OEIS).*

Dans cette partie, nous avons décidé d'implémenter un algorithme issu de la programmation dynamique pour avoir seulement deux étages du graphe en mémoire, nous entraînant pour l'étude des monoïdes puisque ces objets sont trop volumineux pour être entièrement en mémoire. Nous avons aussi parallèlisé l'algorithme en faisant attention à ne pas avoir d'écritures concurrentes.

Monoïde de fonctions

Définition :

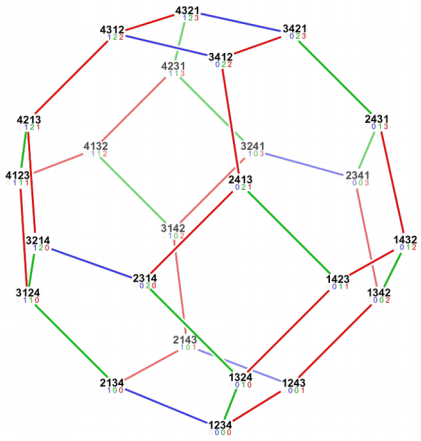
Nous nous sommes ensuite intéressés au monoïde des partitions ordonnées. Tout d'abord, definissons ce qu'est un **monoïde**: « C'est un ensemble d'objets qui admet un élément neutre et une loi associative ».

Une **partition** est un ensemble de parties non vide d'un ensemble e deux à deux disjoints et qui recouvrent e . On dit d'une partition qu'elle est ordonnée lorsque l'ordre de ses parties importe.

Par exemple, les partitions *1|3|2* et *1|2|3* sont équivalentes mais les partitions ordonnées *1|3|2* et *1|2|3* ne le sont pas.

Les partitions ordonnées contenant les entiers de 1 à 4, sont en bijection avec les facettes du permutohèdre d'ordre 4. Nous retrouvons 24 partitions avec 4 parties c'est à dire les sommets du permutohèdre, nous avons aussi 36 partitions avec 3 parties répresentant les arrêtes, 14 partitions à 2 parties représentant les faces et 1 partition à 1 partie représentant l'intérieur du permutohèdre.

Ceci est vrai pour tout n avec toutes dimensions.



Rank :

Nous avons encore une fois cherché une méthode de ranking/unranking dans cet ensemble, tout d'abord, nous avons remarqué qu'il y avait une bijection entre ce qu'on appelle les « formes » et les mots binaires de taille $n-1$.

La **Forme d'une partition** est une suite d'entiers dont la somme vaut n et dont les entiers représentent la répartition des entiers dans la partition.

Explications : si nous prenons toutes les formes possibles du pseudo-permutohèdre d'ordre 3, nous avons les formes suivantes, *(3), (2,1), (1,2), (1,1,1)*. On remarque bien ici qu'il y a 2^{n-1} formes c'est à dire 4 pour $n=3$. La bijection est la suivante, on parcourt le mot binaire et pour chaque ensemble de k 0 consécutifs suivis d'un 1, il y aura $k+1$ éléments dans la partie, pour que la bijection marche, on considère que chaque mot binaire se termine par un 1.

Par exemple, le mot binaire *0100* donnera la forme *(2,3)* pour $n=5$.

Pour une partition p , nous devons connaître le nombre de partitions correspondant aux formes précédant sa propre forme, nous avons pour cela utilisé les coefficients multimoniaux donnant un rang minimum à la partition. Puis nous avons incrémenté ce rang minimum en parcourant la partition toujours en nous aidant des coefficients multimoniaux.

Pour faire un exemple de ranking, prenons la partition *23|14*.

Cette partition est de forme *(2,2)*, nous devons donc connaître le nombre de partitions qui correpondent aux formes *(4)* et *(3,1)*. On utilise un premier coefficient multimonial pour la forme *(4)*. Ce qui nous donne le calcul suivant :

$$\binom{n}{4}$$

N valant 4, ce coefficient multimonial est au final un coefficient binomial valant 1. Il y a donc une seule partition qui respecte la forme *(4)*. Pour la forme *(3,1)*, nous avons

$$\binom{n}{3,1}$$

Ce qui nous donne 4. Nous avons comme rang minimum 4 (nous commencons à compter à partir de 0). Puis on réapplique un coefficient multimonial sur chaque élément de la partition de la facon suivante :

$$rang\,min = rang\,min + \binom{n_{restant}}{forme_{restante}}$$

$n_{restant}$ est le nombre d'éléments restants dans la partition et $forme_{restante}$ la forme issue des éléments restants dans la partition. Pour notre partition, cela nous donnera

$$rang\,min = 4 + \binom{3}{1,2} + \binom{2}{2} + \binom{1}{1} = 4+2+1+1 = 8$$

Le rang de la partition *23|14* est donc 8.

Enumération de fonctions :

Maintenant que nous avons ordonné nos partitions, nous pouvons regarder les relations qui existent entre elles. Nous avons décidé de regarder les fonction suivantes :

S_i : Sépare une partie en deux au $i^{ème}$ élément, mettant les plus grands éléments de la partie d'origine à gauche.

$Ex : S_4$ (35|2467|1)=35|67|24|1

F_i : Fusionne deux parties dont font partis le $i^{ème}$ et le $i+1^{ème}$ entier si tous les entiers de la partie droite sont plus grand que les entiers de la partie gauche.

$Ex : F_3$ (1|34|567|2)=1|34567|2

π_i : Applique $S_i et F_i$ sur la partition quand cela est possible, ne fait rien sinon.

Le but recherché était de savoir combien de fonctions pouvait-on obtenir en composant les opérateurs π_i . L'ensemble de ces fonctions est le sous-monoïde du monoïde engendré par les opérateurs π_i . Pour cela, nous avons regardé quelles partitions nous obtenons en appliquant chaque S_i avec i allant de 1 à $n-1$ sur toutes partitions, puis chaque F_i pour enfin avoir π_i . Bien sûr, nous avons utilisé notre méthode de ranking pour enregistré les résultats des applications de π_i pour ne pas être obligé d'enregistré les partitions elles-mêmes ce qui aurait été un gaspillage de mémoire.

Le nombre de fonctions selon n décrit cette suite pour n allant de 1 à 6 :

$$1, 2, 6, 48, 994, 56883$$

Conclusion

Lors de mon stage, j'ai été confronté à des exercices de combinatoire qui m'ont montré que des problèmes qui demandent au premier abord beaucoup de ressources peuvent être simplifiés par des représentations plus théoriques aidant ainsi la manipulation d'objets.

J'ai aussi appris pendant ce stage certains algorithmes utilisant la mémoïsation ou la programmation dynamique dans un graphe. Le coté implémentation n'ayant pas été beaucoup abordé lors de ce rapport, je tiens a dire que nous avons effectué de l'optimisation de code grâce à des profilers comme *gprof* ou *perf* ce qui a été très intéressant puisqu'ils m'ont permis de me rendre compte que certaines optimisations se faisaient à des endroits innatendus.

Je tiens à remercier M. Florent Hivert, professeur à l'Université Paris-Sud 11 de m'avoir accepté et accompagné tout au long de ce stage me faisant ainsi découvrir la combinatoire. Je suis aussi reconnaissant envers le projet Opendreamkit (#676541) qui est un projet de développement européen inscrit dans le programme de recherche européen horizon2020 pour le financement de mon stage.