Mon projet

Dos Santos David

October 11, 2008

Introduction

Un problème comme la recherche d'occurrence dans un texte peut sembler simple au premier abord, mais il devient critique lorsque l'on a à traiter une grande quantité de données (bibliothèques électroniques, Base de données biologiques, etc.) ainsi qu'une grande quantité de requêtes (moteurs de recherches). Pour résoudre ce problème il semble naturel de se tourner vers la puissance offerte par les machines parallèles. Néanmoins les algorithmes ne sont pas les mêmes que ceux applicables sur des machines séquentiels.

On se propose ici de présenter un algorithme qui recherchera une occurrence d'une chaîne dans un ou plusieurs textes en utilisant les capacités des machines parallèles. Notre but est d'arriver à répartir équitablement la charge de travail entre p processeurs ainsi que d'équilibrer l'occupation mémoires. De plus on prendra en considération le coût des communications.

Dans un premier temps nous reviendrons sur des notions nécessaires à la compréhension de l'algorithme présen'té. Nous aborderons les structures de données " Patricia Tries " qui permettent une recherche efficace sur un ensemble tout en minimisant l'espace mémoire nécessaire. Dans l'optique de réaliser une analyse de complexité, nous présenterons le modèle BSP.

Une fois ces quelques rappels effectué nous pourrons alors nous lancé dans la description de l'algorithme. Nous commencerons par un algorithme qui recherchera les occurrences d'une multitude de chaînes dans un texte (de longues chaînes de même taille). Nous introduirons la structure de donnée nécessaire avant de nous lancer dans un descriptif détaillé de l'algorithme, grâce auquel nous pourrons effectuer une analyse de complexité. Pour finir on donnera un exemple d'éxécution.

On finira par l'étude d'un algorithme de recherche d'occurrence pour une seule chaîne. De la même façon que pour l'algorithme précédant on introduira les structures nécessaires avant de donner une description détaillée de l'exécution. On finira par un exemple d'exécution.

Chapter 1

Travaux relatifs

Chapter 2

BSML

Chapter 3

Patricia tries

3.1 Définitions des Patricia Tries

Les 'Patricia Trie' (Practical Algorithm to Retrieve Information Coded in Alphanumeric) sont des structures qui permettent une recherche rapide sur un ensemble de chaîne tout en minimisant la taille de l'arbre de recherche. Cette structure a été introduite par Morrison en 1968. On utilise ici une version légèrement différente qui permet de travailler sur un alphabet arbitraire. Cette version est définit dans l'article de Paolo Ferragina et Roberto Grossi [2]. Nous en rappelons ici les principales caractéristiques.

Soit une chaîne de n caractéres $X[1,n]$ et une chaîne de m caractéres $Y[1,m]$ (aucune n'étant préfixe de l'autre). On définit $lcp(X,Y)$ (longest common prefix) comme étant la taille de leur plus grand préfixe commun . On a donc $lcp(X,Y) = k$ si et seulement $X[1,k] = Y[1,k]$ et $X[1,k+1] \neq Y[1,k+1]$ La proposition suivante met en évidence la relation entre l'ordre lexicographique $\leq_L$ et la valeur de lcp :

Prop 1 $\forall X1, X2, Y$ *tel que* $X1 \leq_L X2 \leq_L Y$ ***ou*** $Y \leq_L X2 \leq_L X1$ *alors :* $Lcp(x1,Y) \leq lcp(X2,Y)$

Soit $\mathcal{S} = \{X_1, \ldots, X_d\}$ un ensemble de chaînes ordonnées (on suppose que deux chaînes de $\mathcal{S}$ ne sont pas préfixe l'une de l'autre). On note $max_{lcp}(Y,\mathcal{S})$ la plus grande valeur lcp entre Y et les chaînes de $\mathcal{S}$: $max_{lcp}(Y,\mathcal{S}) = max_{x\epsilon\mathcal{S}} lcp(Y,X)$. On dit que j est la position de Y dans l'ensemble $\mathcal{S}$ si $(j-1)$ chaînes appartenant à S sont plus petites que Y. D'où $1 \leq j \leq d+1$. On a :

Prop 2 *Si la position de* Y *dans* S *est* j *alors :*

$$max_{lcp}(Y,\mathcal{S}) = \begin{cases} lcp = (Y, x_1) & si \quad j = 1 \\ max(\, lcp(X_{j-1}, Y),\, lcp(Y, X_j)\,) & si \quad 2 \leq j \leq d \\ lcp(X_d, Y) & si \quad j = d+1 \end{cases}$$

Nous pouvons maintenant introduire une définitions des 'Patricia Trie'. Un arbre 'PATRICIA' construit sur l'ensemble ordonné $\mathcal{S}$ satisfait les conditions suivantes :

1. Chaque arc possède un caractère de branchement pris de Σ (Σ étant l'alphabet sur lequel est défini $\mathcal{S}$) et chaque Noeud interne possède au minimum deux arcs sortants (avec un caractère de branchement différent). Les arcs sortants sont ordonnés en fonctions des caractères de branchement. Seul la racine peut ne posséder qu'un seul arc sortant;

2. Une feuille différente v est associée à chaque ch'îne de $\mathcal{S}$. On note cette chaîne $W(v)$;

3. Si le nœud u est le dernier ancêtre commun entre deux feuilles l et f, alors il aura pour valeur l'entier $len(u) = lcp(W(l), W(f))$ (la valeur valeur du noeud racine sera $len(r) = 0$). La feuille l (respectivement f) descend de l'arc sortant possèdant le caractère de branchement egal au (lcp+1)-ème caractère de la chaîne $W(l)$ (respectivement $W(f)$) .

Prenons en exemple l'ensemble $S = \{aaabcb, babbcb, babbcc, babcbc, babccc, caabbc, caabbd, cbabbd\}$. Voici le Patricia Trie associé à cet ensemble.

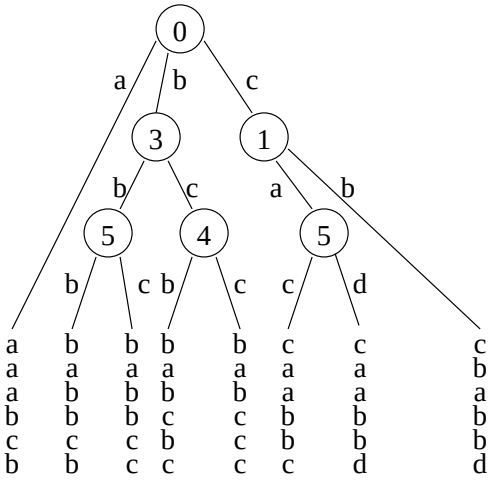

Figure 3.1: exemple de Patricia Tree

Soit u un nœud interne de $PT_{\mathcal{S}}$ on note $parent(u)$ le parent de u. Soit f est une feuille descendante de u. On note $W(u)$ la chaîne implicitement contenue dans u qui d'après la conditions (3) est égal aux $len(u)$ premiers caractères de la chaîne $W(f)$. $Arc(parent(u), u)$ correspont donc à une sous-chaîne de taille $(len(u) - len(parent(u))$ dont le premier caractère est $W(f)[len(parent(u)) + 1]$ et les autres caractères et les caractères suivant égales aux caractères de $W(f)$ de la position de $len(parent(u)) + 2$ à $len(u)$.

Définition 1 *le Noeud de 'contact'(Hit Node dans la terminologie anglaise) pour une paire (f, k) tel que f est une feuille et $0 < k \leq len(f)$, est l'ancètre u de f tels que : $len(u) \geq k > len(parent(u))$. Si $k = 0$ alors le 'Hit Node' est la racine.*

3.2 Algorithme de recherche dans une PT

On se propose maintenant de fournir un algorithme permettant de retrouver rapidement la position lexicographique d'une chaîne dans un ensemble ordonné grâce aux PT.

Prop 3 *Soit P une chaîne différente de toutes les autres chaînes, si une chaîne appartenant à $\mathcal{S}$ a le préfixe $p[1, |P|]$ alors la position de P dans F est à gauche de cette chaîne.*

Première phase:

On localise une Feuille l en parcourant le graphe $PT_{\mathcal{S}}$. On commence de la racine et on compare le caractère de branchement de l'arc traversé avec le caractère de P. Soit u le Nœud traversé, si u possède un $arc(u, v)$ dont le caractère de branchement est égal à $P[len(u)+1]$, alors on passe de u à son fils v. On continue jusqu'à arriver à une feuille ou jusqu'à ce que aucun arcs de u ne possède de caractère de branchement égal à $P[len(u) + 1]$. Dans ce cas on choisit l comme étant une feuille descendante de u. La feuille l trouvée vérifie la propriété suivante : $Lcp(W(l), P) = max_{lcp}(P, \mathcal{S})$

En reprenant l'exemple précédemment on montre en gras sur la figure le parcours éffectué durant la première phase pour rechercher la chaîne $X = caaabdc$.

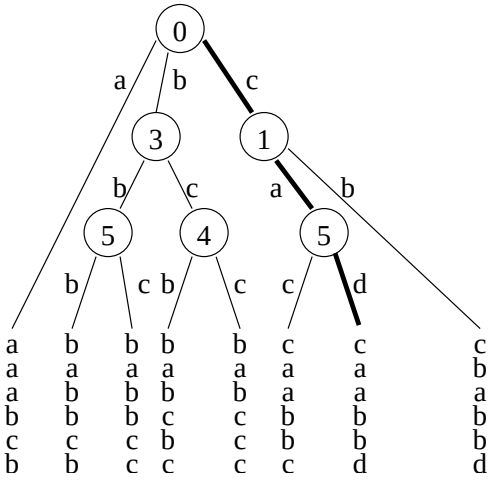

Figure 3.2: Phase 1 de la recherche de position

Deuxième phase, déterminons la position j de P : On trouve lcp = lcp(W(l),P) et les deux caractéres différents : $c = P[lcp+1] et c' = W(l)[lcp+1]$. On détermine le " Hit Node " u du couple (l, lcp). Toute les chaînes stockées dans les descendants de u ont le même préfixe de taille $len(u)$. On a $len(u) > lcp$ d'où le $(lcp+1)$-ème caractére de toutes les chaînes descendante de u est le même. Donc si $c <_L c'$ on se place dans la feuille descendante de u la plus á gauche que l'on note z . On a $j-1$ comme étant le nombre de feuille á gauche de z (z exclus). Si $c >_L c'$ alors on descendant jusqu'á la feuille z descendante de u la plus á droite. On a $j-1$ comme étant le nombre de feuille á gauche de z (z incluse).

Exemple on voit en gras sur la figure le Noeud de contact ainsi que la position finale de X=caaabdc.

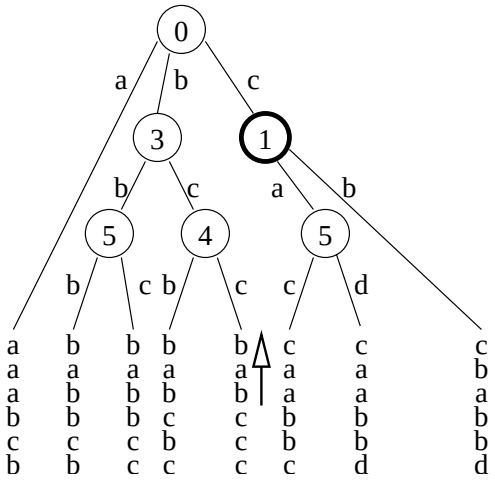

Figure 3.3: Phase 2 de la recherche de position

3.3 Construction d'un 'PATRICIA trie' :

Nous présentons ici un algorithme qui nous sert á construire $PT_{\mathcal{S}}$ á partir d'un ensemble ordonné $\mathcal{S}$. Nous allons construire notre arbre par insertion successive des différentes chaînes de l'ensemble $\mathcal{S}$.

On parcours notre arbre comme dans la première phase de l'algorithme de recherche.

1. Si on arrive à un nœud u pour lequel aucun branchement n'est possible. Alors on lui attache une feuille l avec $W(l) = P$. Le caractère de branchement de l'$arc(u, l)$ sera $P[len(u)+1]$ ($arc(u, l)$ sera bien entendu placé de façon à respecter l'ordre de lexicographique)

2. Si on arrive à une feuille l. On détermine $lcp(W(l), P)$ ainsi que$c = P[lcp+1]$, $c' = W(l)[lcp+1]$. Soit u le 'Hit Node' obtenue à partir du couple $(W(l), lcp)$. On prend $parent(u)$, on note a l'arc$(parent(u), u)$. On créer un nouveau nœud v ayant pour valeur lcp . v aura deux arcs sortant un arc(v, u) avec comme caractère de branchement c' et un autre arc(v, f) ayant comme caractère de branchement c et où f est la feuille tel que $W(f) = P$. On rattachera ensuite ce nouveau nœud v à parent(u) via l'arc a (qui deviendra donc arc(parent$(u), v)$))

Par exemple pour en insérant la chaîne $X = cabddd$ on obtiendra le PT de la figure suivante.

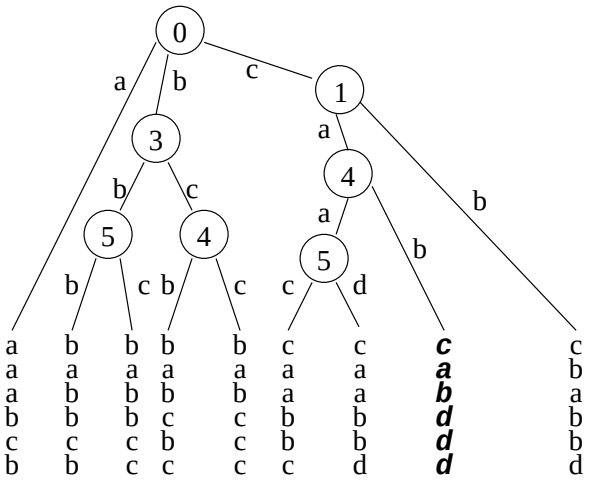

Figure 3.4: Nouveau PT aprés insertion de $X = cabdd$

Chapter 4

Algorithme

Considérons $T = T[1, n]$ comme une chaîne de caractères, alors on peut définir $T[1, i]$ (i appartenant à $[1, \ldots, n]$) comme un préfixe de T. Et $T[j, n]$ (j appartenant à $[1, \ldots, n]$) comme un suffixe de T. $T[i, j]$ est une sous chaîne de T. Soit$X[1, m]$ un pattern, on dit qu'il y a une occurrence de X à la postion i dans T si $X = [i, i + m - 1]$. Ou de façon équivalente si X est le préfixe d'un suffixe de $T[i, n]$.

On note $SUF(T) = \{T_1, \ldots, T_n\}$ l'ensemble ordonné de tout les suffixes de T, où $T_i = T[j_i, m]$ pour $1 \leq j_i \leq n$, $Ti \leq_L T_i + 1$, $\forall i = 1, \ldots, n - 1$ ($\leq_L$ étant l'ordre lexicographique). Rechercher des occurences du pattern $X[1, m]$ dans T revient à retrouver toutes les chaînes de $SUF(T)$ ayant pour préfixe X. Une propriété de Manber et Myers [3] nous dit que : les suffixes ayant pour préfixes X occupe une partie contigüe dans SUF(T). De plus, la plus à gauche de ces chaînes peut être identifiée par le fait qu'elle est adjacente à la position de X dans SUF(T). Rechercher une occurence de X dans T revient donc à retrouver sa position lexicographique dans SUF(T).On utilisera un PT pour déterminer une position dans SUF(T). La construction d'un PT sur un ensemble ordonné suppose qu'aucune chaîne n'est préfixe d'une autre. On ne peux pas faire ce postulat pour un ensemble qui représente les suffixes d'un texte. Les auteurs de l'article ne parlent pas du cas ou un suffixe est préfixe d'un autre. Pour être sur de ne pas tombé sur ce cas on rajoutera à la fin du texte un caractère qu'on ne retrouvera nul part ailleurs (on le supposera plus petit que tout les autres dans l'ordre lexicographique).

Exemple : $T = aabbcabccabbc\#$ l'ensemble ordonné sera alors $SUF(T) = \{1, 10, 2, 6, 11, 3, 12, 4, 7, 13, 9, 5, 8, 14\}$ (on désigne ici les chaînes par leur preimer caractères dans le texte T).Prenons $X = cab$, la position lexicographique de X dans $SUF(T)$ est entre $T[13, 13]$ et $T[9, 13]$. On voit que X est préfixe de $T[9, 13]$ on en conclut donc que T apparait dans T (on peut de plus en déduire facilement la position de l'occurence, ici 9).De plus X est également préfixe du suffixe $T[5, 13]$. T possède donc deux occurences de X,la premiere à la position 5 et la seconde à la position 9. Si on prend X=aba, on place cette chaîne entre T[1,13] et T[10,13]=abbc#. X n'étant pas préfixe de $T[10, 13]$ on en déduit que

T ne possède pas d'occurence de T.

4.1 Algorithme de recherche pour plusieurs longues chaînes

Nous allons ici décrire la façon de déterminer l'occurence de plusieurs longues chaînes dans un textes. Soit $X_1[1,m], X_2[1,m], \ldots, X_k[1,m]$ les chaînes à rechercher dans le texte T. L'algorithme présenté est optimal pour $m \geq p$ et $k \geq p$ (p étant le nombre de processeurs).Le problème principale est que l'on ne peut pas stocké un PT construit sur l'ensemble SUF(T) car on aurait alors une compléxité mémoire de $O(n)$. On doit distribuer ce PT entre les processeurs, tout en veillant à minimiser les communications nécéssaires.

4.1.1 Structure de données

On distribue les caractères de T[1,n] aux différents processeurs de façon a ce que le premier processeur contienne $T[1], T[p+1], \ldots, T[kp+1]$, le deuxième $T[2], \ldots, T[kp+2]$, et ainsi de suite.On construit de plus un BT sur l'ensemble des suffixes $S = \{T_1, T_{p+1}, T_{2p+1}, T_{3p+1}, \ldots\}$ que l'on nommera PT_S. En ayant un caractères par arcs, un entier par noeud et un entier par feuille (pointeur sur le premier caractère de la chaîne dans T) on peut considérer que PT_S occupe $O(n/p)$. On peut alors en stocker une copie dans chaque processeur. On prend les ensembles $S^j = \{T_{jp+1}, T_{jp+2}, \ldots, T_{(j+1)p}\}$ avec $j = 0, 1, \ldots, \lfloor n/p \rfloor$ et on construit un PT^j sur chacun de ces S^j. On observe que chaque PT^j utilise $O(n)$ espace puisque il est construit sur p suffixes. Chaque PT^j est alors divisé en p parties à peu prés équilibrées, et on distribue ces parties aux processeurs.On note PT_i^j le morceau de PT^j contenue dans le i-ème processeur.

En reprenant notre premier exemple et en supposant p=3 on aura: $S = \{1, 6, 12, 13, 8\}$, une copie de PT_S sera contenue dans chaque processeur.De plus on aura $S^0 = \{1, 10, 2\}$, $S^1 = \{6, 11, 3\}$, $S^2 = \{12, 4, 7\}$, $S^3 = \{13, 9, 5\}$, $S^4 = \{8\}$ et dans chaque processeur une partie de $PT^0, PT^1, PT^2, PT^3, PT^4$.

Nous allons maintenant décrire les différentes étapes de l'algorithme. On considère k chaînes $X_1, X_2, \ldots, X_k$ à chercher dans T. Elle sont distribuées parmis les processeurs de façon équilibrée. On décrit de façon détaillé le travail d'un processeur en sachant que tous auront à faire le même travail.

Super-étape 1: Le processeur exécute la première phase de la recherche de position sur sa copie de PT_S. On obtient $l_1, l_2, \ldots, l_{k/p}$ correspondant au feuille contenant les chaînes qui possède le plus long préfixe commun avec (respectivement) $X_1, X_2, \ldots, X_{k/p}$.La compléxité en calcul de cette phase est de O(km/p).Une fois ceci fait le processeur broadcast $(l_1, l_2, \ldots, l_{k/p})$ à tous les processeurs.Ceci coutera une (k/p)-relation.

Super-étape 2: Le processeur collecte les requêtes $l_1, ..., l_k$ envoyées par les autresp processeurs. Pour chaque requête le processeurs récupère les premiers (m/p) caractères stocker dans sa mémoire local (par exemple pour

une requête à 7 le processeur récupére (m/p) caractères qu'il possède à partir 7). Il renvoit les données récupérées aux différents processeurs. Ainsi chaque processeur possèdera l'intégralité des chaînes contenue dans les feuilles $l_1, ..., l_k$.Cette étape nécéssite une (km/p)-relation.

Super-étape 3: le processeur reçoit les préfixes de taille m associés à $l_1, l_2, \ldots, l_{k/p}$, il les compare aux chaînes $X_1, X_2, \ldots, X_{k/p}$. On détermine la position des chaînes $X_1, X_2, \ldots, X_{k/p}$ (en exécutant la deuxième phase), grâce auxquelles on trouve dans qu'elle PT^j rechercher la position de nos chaînes.Alors le processeur demande aux autres processeurs les parties de PT dont il a besoin.Ceci nessecite une (k/p)-relation.

Super-étape 4: Le processeur collecte les k requêtes des autres processeurs et renvoit les PT_0^i correspondant.Cette étape demande une k-relation.

Super-étape 5-7: le processeur reconstruit dans sa mémoire local les PT^j et recherche les positions des chaînes $X_1, X_2, \ldots, X_{k/p}$ grace a ces PT. Cette étape coûte $O(km/p)$ temps de calcul et demande une (km/p)-relation (car il faudra récupérer les caractères de T dans les autres processeurs).

Super-étape 5-7: grâce aux positions léxicographiques retrouvées on détermine si X_j apparait dans T en vérifiant si il est préfixe $T_{pos(j)}$. Ceci demande une (k/p)-relation.

La procédure totales a une compléxité en temps de $O((km/p) + k)$ et une coût en communication de $O(kg + (km/p)g)$.

Bibliography

[1] P. Ferragina et F. Lucio,"'String Search in Coarse-Grained Parallel Computers"', Algorithmica (24: 177-194), 1999

[2] P. Ferragina et R. Grossi, "'A fully-dynamic structure for external substring search"'. In Proc. of the 27h ACM-SIAM Symposium on Discrete Algorithms, pp. 373-382, 1996

[3] U. Mamber et G.Myers. Suffix arrays: a new method for on-line string searches. SIAM Journal on Computing, 22(5):935-948, 1993