Algorithmique et Structures de Données 2

Alain Griffault
Université Bordeaux I
Version $Id: asd.tex,v 1.2 2010/01/05 11:36:17 griffaul Exp $

11 février 2011

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Les graphes

1.1 Préliminaires

1.1.1 Définitions et notations

Définition 1.1 (Graphe non orienté) – *S un ensemble de sommets*
– *A un ensemble d'aretes inclu dans SxS tel que $(s,t) \in A \Rightarrow (t,s) \in A$*
définissent un graphe non orienté $G(S,A)$

Définition 1.2 (Graphe orienté) – *S un ensemble de sommets*
– *A un ensemble d'arcs inclu dans SxS.*
définissent un graphe orienté $G(S,A)$

Définition 1.3 (Chemin) *Soit $G(S,A)$ un graphe (orienté ou non). Soit $(s,t) \in S^2$, un chemin $s \rightsquigarrow t$ est une suite de sommets $u_0, \ldots, u_n$ telle que :*
– $u_0 = s$
– $u_n = t$
– $\forall i (u_i, u_{i+1}) \in A$

1.1.2 Représentations des graphes

Représentation par matrice d'adjacence

Représentation par listes d'adjacence

Exemples

Besoin de coloriage pour parcourir

1.2 Parcours en largeur

1.2.1 L'algorithme

1.2.2 Plus courts chemins d'origine s

1.2.3 Arborescence en largeur d'origine s

1.2.4 Impression d'un chemin de s à t

1.3 Parcours en profondeur

1.3.1 L'algorithme

l'algorithme récursif `Visiter-PP(u)`

1. propriété des couleurs → complexite, parenthésage

l'algorithme `PP(G)`

1. propriété des couleurs → complexite, parenthésage

1.3.2 Classification des arcs

- liaison : $(\pi(v), v)$ (couleur(v) = blanc)
- retour : $(u, v), v$`ancetrede`u (couleur(v) = gris)
- avant : $(u, v), u$`ancetrede`v (couleur(v) = noir)
- couvrant : les autres (arborescences différentes mais aussi meme arborescence (couleur(v) = noir)
- avant : $(u, v) d(u) < d(v) < f(v) < f(u)$, et (couleur(v) = noir)
- couvrant : $(u, v) d(v) < f(v) < d(u) < f(u)$, et (couleur(v) = noir)

L'algorithme

1.3.3 Tri topologique

Propriété 1.1 (oriente acyclique) *orienté acyclique ssi pas d'arc retour*

L'algorithme

```
dans Visiter-PP(u)
si (arc retour) alors Impossible
...
f(u) := date;
date := date +1;
inserer_tete(L,u);

dans PP(G)
declarer L := CreerListeVide();

retourner L
```

exemple

1.3.4 Calcul des composantes fortement connexes d'un graphe orienté

Définition 1.4 (CFC) *Ensemble maximal d'états.*

Définition 1.5 (Transposé d'un graphe) *Matrice : rien à faire Liste de pointeurs :* $O(A)$

Propriété 1.2 *Un graphe et son transposé possédent les memes CFC.*

Un exemple

page 480

Définition 1.6 (aieul) *aieul(u) noté aieul*(u) *le sommet accessible depuis u ayant la valeur* $f(u)$ *maximale.*

Propriété 1.3 *Dans un graphe, tous les sommets ayant le meme aieul forment une CFC.*

L'algorithme

Il faut une variable globale aieul modifiée avant chaque appel à `Visiter_PP` dans `PP`.
Complexité $0(A + S)$

Chapitre 2

Arbres pondérés

Définition 2.1 (Graphe pondéré) *Un graphe (orienté ou non) pondéré $G(S, A, w)$ est un graphe $G(S, A)$ muni d'une fonction de pondération $w : A \to R$.*

2.1 Arbres couvrants de poids minimal

2.1.1 Le problème

Le problème des "Arbres couvrants minimaux" consiste à trouver un arbre couvrant d'un graphe pondéré non orienrté tel que la somme des poids des (arcs, aretes) soit minimal.

Problème lié à la construction du cablage d'un circuit,

2.1.2 Un algorithme générique

- A priori : problème d'optimisation "globale" de grande complexité (la première idée est de calculer tous les arbres, puis de comparer les poids). - Une approche gloutonne consiste à faire des choix (heuristique) locaux pour améliorer une solution existante pour en obtenir une meilleure. On obtient ansi une solution "maximale" localement mais pas forcément "optimale". - Pour certains problèmes (dont celui-ci), la solution "maximale" obtenue est "optimale".

Définition 2.2 (arete sure) *Soit $G(S, A, w)$, soit $E \subset ACM$. Une arete (u, v) est sure si $E \cup \{(u, v)\} \subseteq ACM$.*

Propriété 2.1 *Soit $G(S, A, w)$, soit $E \subset ACM$. il existe une arete sure (u, v).*

Définition 2.3 (coupure) *Soit $G(S, A, w)$, soit $P \subset S$. $(P, S - P$ est une coupure.*

Définition 2.4 (coupure pour un ensemble d'aretes) *Soit $G(S, A, w)$, soit $E \subseteq A$, soit $P \subset S$. la coupure $(P, S - P$ est une coupure qui respecte E si aucune arete de E traverse la coupure. ($E \subseteq P \times P$ ou $E \subseteq (S - P) \times (S - P)$*

Propriété 2.2 *Soit $G(S, A, w)$, soit $E \subseteq ACM$, soit une coupure $(P, S - P$ qui respecte E. Une arete de poids minimal qui tracverse la coupure est une arete sure.*

2.1.3 L'algorithme de Kruskal

2.1.4 L'algorithme de Prim

2.2 Plus courts chemins à origine unique

Le problème consiste dans un graphe $G(S, A, w)$, à trouver pour chaque sommet $s \in S$, le chemin de poids minimal entre un sommet $r \in S$ et x.

Problème similaire au précédent.

Il existe d'autres problèmes similaires :

– Plus courts chemins à destination unique.
– Plus court chemin pour un couple de sommets.
– Plus courts chemins pour tout couple de sommets.

Le problème se pose aussi avec des arcs de poids négatif sans circuit de poids négatif.

2.2.1 Plus courts chemins et relachement

Propriété 2.3 *Les sous-chemins des plus courts chemins sont des plus courts chemmins. Soit* $G(S, A, w)$*, soit* $(v_1, \ldots, v_n)$ *un plus court chemin de* v_1 *à* v_n*, alors* $\forall 1 \leq i < n, \forall i \leq j < n, (v_i, \ldots, v_j)$ *est un plus court chemin de* v_i *à* v_j*.*

Propriété 2.4 *Les plus courts chemins à origine unique forment une arborescence.*

Propriété 2.5 *Les plus courts chemins sont représentables par une structure* $\{(\pi(v), v)\}$*.*

Définition 2.5 *La technique de relachement consiste à faire décroitre une borne supérieure jusqu'à ce qu'elle soit égale au résultat attendu. Pour notre problème, cette quantité* $d(s)$ *sera le poids du plus court chemin entre un sommet* s *et le sommet racine* r*.*

L'algorithme de Dijkstra

Très similaire à l'algorithme de Prim pour les arbres couvrants.

Il part du sommet racine, et construit une arborescence unique de plus en plus grande en ajoutant le sommet ayant la distance de relachement la plus petite, puis en mettant à jour les informations sur ses adjacents.

Chapitre 3

Recherche de motifs

3.1 Le problème et les applications

Mots dans un texte, sequence ADN,...

3.2 L'algorithme naif

```
Motif_naif(T,P) {
  pour i de 0 a T.longueur - P.longueur faire {
    trouve <- vrai;
    j <- 0;
    tant que (trouve & j<P.longueur) faire {
      trouve <- T[i] = P[j];
      j <- j+1;
    }
    si trouve
    alors Ecrire "Le motif apparait au rang "+i;
  }
}
```

3.3 Les algorithmes basés automates

3.3.1 Automates finis et langages

definition et exemples

3.3.2 Automates de recherche de motif

exemples

3.3.3 L'algorithme de recherche

```
Motif_automate(T,P) {
  delta[0..m][Sigma] <-Motif_Construction_Automate(P);
  q <- 0;
  pour i de 0 a T.longueur-1 faire {
    q <- delta[q, T[i]];
    si (q=m)
    alors Ecrire "Le motif apparait au rang "+(i-m);
  }
}
```

3.3.4 La construction de l'automate

```
Motif_Construction_Automate(P) {
  m <- P.longueur;
  pour q de 0 à m faire {
    pour x dans Sigma faire {
      k <- min(m+1, q+2);
      repeter k <- k-1;
      jusqu'a Pk suffixe de Pq.x;
      delta[q,x] <- k;
    }
  }
  retourner delta;
}
```

3.4 L'algotithme de Knuth, Morris et Pratt

3.4.1 L'idée

Calculer directement l'effet de l'automate de recherche.

3.4.2 L'algotithme

```
Motif_KMP(T,P) {
  Pi[0..P.longueur-1] <- Calcul_Fonction_Prefixe(P);
  q <- 0;
  pour i de 0 a T.longueur - 1 faire {
    tant que (q>0 et P[q] <> T[i]) faire {
      q <- Pi[q];
    }
    si (P[q] = T[i])
    alors q <- q+1;
    si (q = P.longueur)
    alors {
      Ecrire ""+(i-m+1)
      q <- Pi[q-1];
    }
  }
}
```