Ch. 6 Ordonnancement et graphes orientés sans circuits

1. Ordonnancement et graphe de dépendance
2. Propriétés des graphes sans circuit
3. Tri topologique et ordonnancement séquentiel
4. Tri par niveaux et ordonnancement parallèle

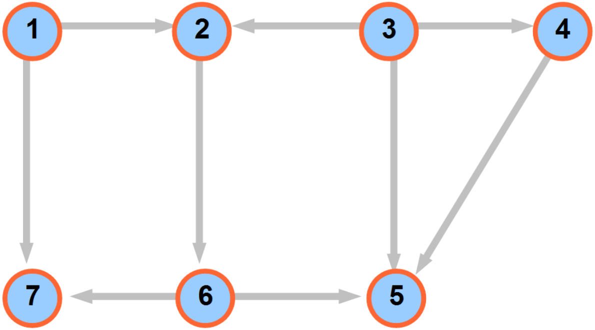
Ordonnancement et graphe de dépendance

Tant d'un point de vue théorique que pratique, les graphes sans circuit jouent un rôle important dans l'étude des ordonnancement de tâches et de l'optimisation. Les applications sont très nombreuses.

Le problème de l'ordonnancement (séquentiel ou parallèle) d'un ensemble de tâches est le suivant :

- La réalisation globale d'un projet à été décomposée en un certain nombre de tâches élémentaires à réaliser : $t_1, t_2, ...t_n$.
- Ces tâches ne peuvent être exécutées dans n'importe quel ordre et doivent respecter un certain nombre de dépendances.
- Ce problème peut être modélisé par un graphe orienté appelé graphe de dépendances dont les sommets sont les tâches $X = (t_1, t_2, ...t_n)$ à réaliser et tel qu'il existe un arc entre t et t' si la tâche t doit être terminée avant de pouvoir commencer la tâche t'.

Le graphe G_5, un exemple de graphe de dépendance



Construction d'un tri par niveaux

Par exemple : On ne peut commencer la tâche 2 tant que les tâches 1 et 3 ne sont pas terminées.

En fonction des ressources (des personnes par exemples) dont on dispose on va alors chercher à classer (ordonnancer) ces tâches pour permettre leur exécution.

- Si on ne dispose que d'une personne, elle va exécuter les tâches l'une après l'autre en respectant les dépendances. On parle d'ordonnancement séquentiel.
- Si on dispose d'un nombre de personnes aussi grand que nécessaire on va chercher à exécuter les tâches en parallèle tout en respectant les dépendances. On parle d'ordonnancement parallèle.
- Si on dispose d'un nombre borné de personnes, on produira un ordonnancement parallèle contraint.

Ce problème est bien sûr lié à l'existence de cycles dans le graphe G car si G possède un cycle $(c_1, c_2, ..c_p, c_1)$ la tâche c_1 ne peut être réalisée avant c_2 qui ne peut être réalisée avant c_p qui ne peut être réalisée avant c_1. Il est bien clair que le projet est alors irréalisable.

Propriétés des graphes sans circuit

Proposition

1. *Si G est sans circuit, tous ses sous-graphes sont sans circuit.*
2. *Si G est sans circuit, le graphe inverse de G, obtenu en inversant l'orientation de tous les arcs de G est encore sans circuit.*
3. *Un graphe est sans circuit ssi tous ses chemins sont élémentaires.*

Propriétés des graphes sans circuit

Un graphe sans circuit possède des sommets remarquables, on rappelle que dans un graphe orienté

Définition

- *Une source est un sommet dont le degré entrant est nul.*
- *Un puits est un sommet dont le degré sortant est nul.*

Théorème

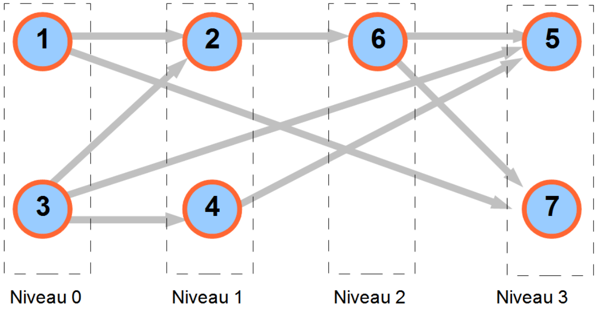
Tout graphe sans circuit possède une source et un puits.

Tri topologique et ordonnancement séquentiel

Définition

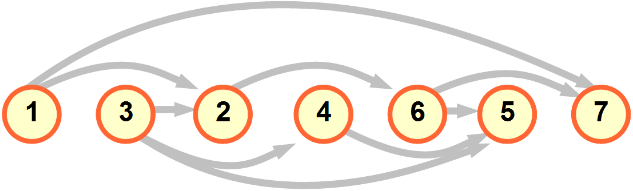
Mathématiquement, on appelle tri topologique d'un graphe orienté $G = (X, E)$ toute numérotation des sommets respectant l'ordre des arcs :
En informatique, on considère souvent un tri topologique comme une fonction qui à un graphe associe une liste de ses sommets triés de manière à ce que si G contient un arc (x, y) alors x soit avant y dans la liste.

Tri par niveau



Le tri par niveaux de G

Tri topologique



Tri topologique de G = [1,3,2,4,6,5,7]

Théorème

Un graphe orienté admet un tri topologique ssi il est sans circuit.

Si G n'a pas de circuit, c'est aussi le cas de tous ses sous-graphes. Tant qu'on n'a pas traité tous les sommets de G, on sait que G possède une source x (donc un sommet sans prédécesseur). x peut donc être traité. On remplace alors G par le sous-graphe obtenu en enlevant x et ses arêtes incidentes. Il n'y a plus qu'à itérer.

Réciproquement si on ne trouve pas de source alors que l'on n'a pas traité tous les sommets de G c'est que G possède un cycle.
La mise en oeuvre de cet algorithme ne nécessite que de déterminer une source à chaque étape.
Pour cela on calcule le vecteur $Degre$ tel que $Degre[x]$ soit le degré entrant de x dans G. et on définit une liste S des sources trouvées.
L'initialisation ne pose pas de problème.
La mise à jour est la suivante : après avoir traité une source x, on décrémente le degré entrant de tous ses successeurs dans G. si un de ces degré devient nul, c'est qu'on a trouvé une nouvelle source que l'on peut ajouter à S. On itère tant qu'on a des sources. Si on n'a plus de sources avant d'avoir traiter tous les sommets c'est que G possède un cycle, sinon l'ordre de traitement des sommets donne un tri topologique donc un ordonnancement séquentiel.

```
Fonction tri topologique G:graphe -> T liste des sommets t
Début
S:=[];T:=[];
Pour x de 1 à n faire
  début
  Degre[x]:=d^-(x) dans G;
  Si Degre[x]=0 alors ajouter x à S;
  fin;
{S contient alors toutes les sources de G}
```

```
tant que S<>[] faire
  début
  x:=enleverTête(S);
  ajouterFin(x,T)

  {Mettre à jour les degrés}
  Pour chaque successeur y de x dans G faire
      début
      Degre[y]:=Degre[y]-1;
      si Degre[y] = 0 alors ajouterFin(y,S)
      fin
  fin
fin.
```

Propriété

- *A chaque étape on traite le premier élément de la liste des sources mais on pourrait traiter n'importe lequel des éléments de S. Il n'existe donc pas un unique tri topologique.*
- *Le tri topologique du graphe de dépendance fournit un ordonnancement séquentiel du problème initial.*

Tri par niveaux et ordonnancement parallèle

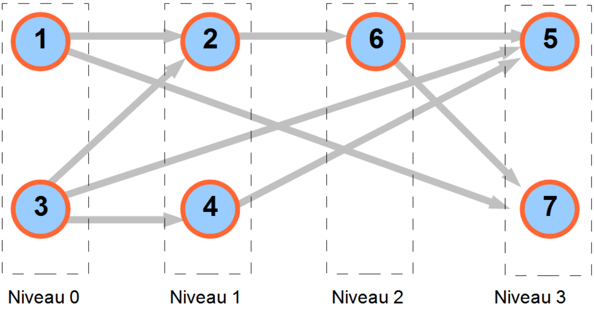
On souhaite cette fois-ci paralléliser au maximum l'exécution des tâches dans le but de réduire la durée du projet.

Théorème

"Partition en niveaux" :
$G = (X, E)$ *est un graphe sans circuit ssi* X *admet une partition en niveaux i.e. une partition de l'ensemble des sommets* X *en* $X = N0 \cup ... \cup N_p$ *telle que* $\forall i \in [1, p]$, *les sommets du niveau* N_i *peuvent être exécutés en parallèle à l'étape* i *(n'ont plus de prédécesseurs non réalisés à l'étape* i.*)*

Il suffit pour cela d'adapter l'algorithme de tri topologique en traitant simultanément toutes les sources de G. On définit deux listes de sources $N1$ et $N2$ correspondant à deux niveaux successifs. Le traitement des sources de $N1$ permettant de déterminer les sources du niveau suivant $N2$.

Le graphe G_5 et son tri par niveaux



Le tri par niveaux de G

```
Fonction tri par niveau G:graphe -> T liste des niveaux T=
Début
N1:=[];T:=[];
Pour x de 1 à n faire
  début
  Degre[x]:=d^-(x) dans G;
  Si Degre[x]=0 alors ajouter x à N1;
  fin;
{S contient alors toutes les sources de G}
```

```
tant que N1<>[] faire
  ajouterFin(N1,T);
  N2:=[];
  pour tout x dans N1 faire
  {Mettre à jour les degrés des successeurs de x et calcul
  Pour chaque successeur y de x dans G faire
      début
      Degre[y]:=Degre[y]-1;
      si Degre[y] = 0 alors ajouterFin(y,N2)
      fin
  fin;
  N1:=N2;
fin
fin.
```

Propriété

- *Si on traite les sommets dans l'ordre des niveaux, on obtient un tri topologique. Mais la réciproque n'est pas nécessaire, il existe des tris topologiques ne respectant pas les niveaux.*
- *Le tri par niveau du graphe de dépendance fournit un ordonnancement parallèle optimal du problème initial. La taille des niveaux fournit le nombre de ressources nécessaires à chaque étape.*