Rapport intermédiaire - Projet GC

Mathis Verstrepen Remi Clement

5 Mai 2023

1 Précisions du cahier des charges

1.1 Objectifs du projet

- Créer une application pour résoudre des problèmes d'ordonnancement.
- Calculer les dates de début au plus tôt et au plus tard pour chaque tâche.
- Calculer la marge de chacune des tâches.
- Identifier la durée minimale du projet et les tâches critiques.

1.2 Exigences fonctionnelles

- Le programme doit être capable de lire un fichier d'entrée avec un format spécifique (inspiré de DIMACS) contenant les informations sur les sommets, les arêtes, les poids et les contraintes de précédence.
- Le programme doit être capable de calculer les informations suivantes pour chaque tâche : date de début au plus tôt, date de début au plus tard, marge et si la tâche est critique.
- Le programme doit être capable de traiter de grands ensembles de données (jusqu'à plus de 300 000 sommets).

1.3 Exigences non fonctionnelles

- Performance: Le programme doit être efficace en termes de coût mémoire et de coût CPU pour traiter de grands ensembles de données.
- Lisibilité: Le code doit être bien structuré, commenté et facile à comprendre.

2 Analyse et conception du problème

L'objectif de ce projet est de créer un programme pour résoudre des problèmes d'ordonnancement de tâches simples en tenant compte des contraintes de précédence et de la durée des tâches individuelles.

2.1 Modélisation du problème

Le problème d'ordonnancement peut être modélisé à l'aide d'un graphe de potentiel tâche, où les sommets représentent les tâches à réaliser et les arêtes représentent les contraintes de précédence entre les tâches. Les poids des arêtes correspondent à la durée des tâches. Le graphe peut être représenté en mémoire à l'aide de diverses structures de données, telles que des listes d'adjacence, des matrices d'incidence ou des matrices d'adjacence. Les différentes structures de données seront comparées et choisies dans la section 4.

2.2 Méthode de résolution

Pour résoudre ce problème d'ordonnancement, nous devons déterminer, comme indiqué dans le cahier des charges, les dates de début au plus tôt et au plus tard pour chaque tâche, ainsi que la durée minimale du projet ,les tâches critiques et la marge de chacune des tâches. Nous utiliserons la méthode de Potentiel-Tâche pour calculer ces informations en utilisant l'algorithme de Bellman sur le graphe.

Cette méthode se déroule en plusieurs étapes :

2.2.1 Calcul des potentiels maximaux

On applique l'algorithme de Bellman à partir du sommet de départ pour déterminer les potentiels maximaux de chaque sommet. Les potentiels maximaux correspondent aux dates de début au plus tôt pour chaque sommet.

2.2.2 Calcul des potentiels minimaux

On construit l'antiarborescence et on utilise l'algorithme de Bellman à partir du sommet de fin pour déterminer les potentiels minimaux de chaque sommet. Les potentiels minimaux correspondent aux dates de fin au plus tard pour chaque sommet.

2.2.3 Identification des tâches critiques et des marges

On calcule la marge de chaque tâche en calculant la différence entre le potentiel maximal et minimal. Les tâches critiques sont celles dont le potentiel maximal est égal au potentiel minimal, c'est à dire où la marge est nulle. Ces tâches sont critiques car elles ne peuvent pas être retardées sans affecter la durée totale du projet.

2.2.4 Calcul de la durée minimale du projet

La durée minimale du projet est déterminée en trouvant la date de fin au plus tard du sommet de fin du graphe.

3 Modélisation et résolution de l'exemple du sondage

On recherche d'abord les niveaux pour chaque sommets du graphe. Appliqué à l'exemple du sondage on trouve les niveaux suivants :

- N_0 = Debut
- N_1 = T1
- N_2 = T2, T3
- N_3 = T4, T5
- N_4 = T6, T7
- N_5 = T8
- N_6 = T9
- N_7 = Fin

On calcule ensuite les valeurs $\pi(x)$, en utilisant l'algorithme de Bellman à partir du sommet de début. Ici, on cherche le potentiel maximum de chaque tâche.

Enfin, on calcule les valeurs $\eta(x)$ en appliquant l'algorithme de Bellman sur l'antiarborescence issue du sommet de fin. Ici on cherche le potentiel minimum de chaque tâche par rapport à la durée minimale trouvée $\pi(F)$.

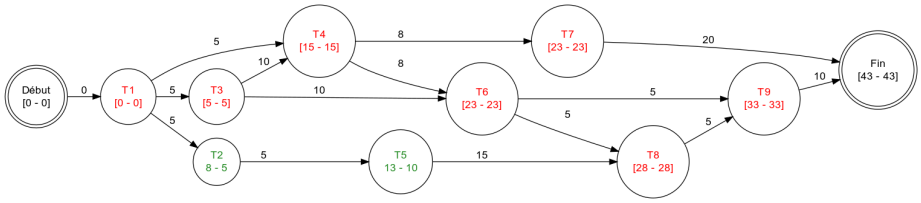

Figure 1: Modélisation de l'exemple du sondage, en rouge les tâches critiques, la première valeur représente la date au plus tard, la seconde la date au plus tôt.

On trouve bien la valeur optimale 43.
Les sommets en rouge sont les tâches critiques.

4 Choix et justification des structures de données utilisées

4.1 Comparaison

Comparaison des structures				
Structure	Coût mémoire	Coût accès sommet	Coût recherche prédécesseurs	Coût recherche succésseurs
Matrice adjacence	$O(n^2)$	O(1)	O(n)	O(n)
Table de successeurs	$O(n+m)$	O(n)	O(m)	O(d)
Table de listes chainées	$O(n+m)$	O(n)	O(m)	O(d)
Notre proposition	$O(n+2m+2n)$	O(n)	O(d)	O(d)

Où n est le nombre de sommets, m le nombre de d'arêtes et d le degré du sommet.

La matrice d'adjacence a un coût mémoire trop élévé pour supporter les problèmes de grande taille.

Ici le graphe ne sera pas amené à être souvent modifié, il n'est donc pas utile d'implémenter la structure les listes chainées.

La tables de successeurs est une bonne solution mais necessite une optimisation quand au coup d'accès des prédécesseurs.

Notre proposition est d'adapter le principe de la table de successeurs composé de la liste Head et la liste *Succ* en y ajoutant une liste *Pred* qui est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets prédécesseurs.

4.2 Schéma applicatif

NB : les indices de ce schéma commence à 1.

4.2.1 Tableau Head

Deb	T1	T2	T3	T4	T5	T6	T7	T8	T9	Fin
1	2	5	6	8	10	11	13	14	15	
1	2	3	4	5	7	8	10	11	13	15
0	5	5	10	8	15	5	20	5	10	0

Le tableau *Head* est une liste composée de structure *Sommet*. Ces structures seront composées de plusieurs champs :

- SuccIdx : Indice du tableau *Succ* à partir duquel sont rangés les successeurs du sommet.
- PredIdx : Indice du tableau *Pred* à partir duquel sont rangés les predecesseurs du sommet.
- Poids : Poids du sommet et donc des arêtes sortantes du sommet.

Nous ajouterons également les champs dateTard, dateTot, estCritique pour stocker les résultats des algorithmes.

Chaque sommets sera donc représenté comme :

```
struct sommet {
    entier succIdx
    entier predIdx
    entier poids
    entier dateTot
    entier dateTard
    booléen estCritique
}
```

4.2.2 Tableau Succ

2	3	4	5	6	5	7	7	8	9	9	10	11	10	11

Le tableau *Succ* est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets successeurs.

4.2.3 Tableau Pred

1	1	2	2	2	4	3	4	5	5	6	7	7	9	8	10

Le tableau *Pred* est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets predecesseurs. On fixe le predecesseur du sommet de départ à lui même pour le bon déroulement de l'algo de Bellman.

5 Algorithmes de résolution appliqués

Algorithm 1 Algorithme d'exploration en profondeur pour calculer l'ordre topologique

Require: $Head, Succ, n$ (nombre de sommets), r (sommet de départ)
1: Initialiser un tableau $Mark$ de taille n avec des valeurs FALSE
2: $Mark[r] \leftarrow$ TRUE
3: Initialiser une pile vide Z
4: PushZ, r
5: Initialiser un tableau $Next$ de taille n avec les valeurs de $Head$
6: Initialiser un tableau vide $Sorted$ de taille n
7: $idxSorted \leftarrow n$
8: **repeat**
9: $x \leftarrow$ Front(Z)
10: **if** $Next[x] > Head[x+1]$ **then**
11: Pop(Z, x)
12: $Sorted[idxSorted] \leftarrow x$
13: $idxSorted \leftarrow idxSorted - 1$
14: **else**
15: $y \leftarrow Succ[Next[x]]$
16: $Next[x] \leftarrow Next[x] + 1$
17: **if** NOT $Mark[y]$ **then**
18: $Mark[y] \leftarrow$ TRUE
19: Push(Z, y)
20: **end if**
21: **end if**
22: **until** SetIsEmpty(Z)
23: **return** $Sorted$

Algorithm 2 Algorithme de Bellman avec sommets triés dans l'ordre topologique

Require: $Head, Pred, n$ (nombre de sommets), $Sorted$ (liste des sommets triés par ordre topologique)

Initialiser $Head[sommet\_de\_depart].dateTard = 0$ et autres sommets avec des valeurs infinies
for $k = 2$ jusqu'à n **do**
  $y \leftarrow Sorted[k]$
  for $i = Head[y].PredIdx$ jusqu'à $Head[y+1].PredIdx - 1$ **do**
    $x \leftarrow Pred[i]$
    $temp \leftarrow Head[x].dateTard + Head[x].Poids$
    if $temp < Head[y].dateTard$ **then**
      $Head[y].dateTard \leftarrow temp$
    end if
  end for
end for
return $Head$

Algorithm 3 Algorithme de Bellman inversé avec sommets triés dans l'ordre topologique inversé

Require: $Head, Succ, n$ (nombre de sommets), $InvSorted$ (liste des sommets triés par ordre topologique inversé)

Initialiser $Head[sommet_de_fin].dateTard = Head[sommet\_de\_fin].dateTot$ et autres sommets avec des valeurs infinies
for $k = 2$ jusqu'à n **do**
  $x \leftarrow InvSorted[k]$
  for $i = Head[x].SuccIdx$ jusqu'à $Head[x+1].SuccIdx - 1$ **do**
    $y \leftarrow Succ[i]$
    $temp \leftarrow Head[y].dateTard - Head[x].Poids$
    if $temp < Head[x].dateTard$ **then**
      $Head[x].dateTard \leftarrow temp$
    end if
  end for
end for
return $Head$

6 Procédure globale d'enchaînement

6.1 Initialisation du graphe

```
Fonction initGraphe(p, Head, Pred, Succ)

        Determination des niveaux du graphe
```

```
        Creation du tableau contenant les niveaux du graphes

        Donnees:
                - Head, tableau contenant les informations sur les sommets
                - Pred, tableau contenant les predecesseurs des sommets

        Resultat:
                - N, tableau contenant les niveaux du graphes
```

6.2 Calcul des niveaux

```
Fonction niveau(p, Head, Pred, N)

        Tri des niveaux
        Tri les niveaux par ordre alphanumerique

        Donnees:
                entier p, nombre de sommets
                N, tableau contenant les niveaux du graphes

        Resultat:
                Ntri, liste de taille p contenant les sommets tries par
                niveau et ordre alphanumerique

tri(p, N, Ntri)
```

6.3 Calcul des dates au plus tard

```
Fonction Bellman(p, Head, N, Pred)

        Calcul des dates au plus tard
        Calcul le potentiel de chaque sommets, correspondant a la
        date au plus tard. Modification de l'information dans la
        structure de chaque sommets
        On utilise ici l'algorithme de Bellman

        Donnees:
                entier p, sombre de sommets
                Ntri, liste de taille p contenant les sommets tries
                        par niveau et ordre alphanumerique
                Pred, tableau conteant les predecesseurs des sommets

        Donnees/Resultat:
                Head, tableau contenant les sommets du graphe
```

```
Bellman(p, Head, N, Pred)
```

6.4 Calcul des dates au plus tot

```
Fonction BellmanInv(p, Head, Pred, N)

        Calcul des dates au plus tot
        Memes informations que precedemment.
        Ici le potentiel calcule correspond a la date au plus tot

BellmanInv(p, Head, Pred, N)
```

6.5 Mise à jour des infos sur les tâches critiques et les marges

```
Fonction critmarge(p, Head)

        Mise a jour des infos sur les taches critiques et les marges
        Determine la marge d'une tache et met a jour le booleen "critique"
        contenu dans la structure si besoin

        Donnees:
                entier p, nombre de sommets

        Donnees/Resultat:
                Head, tableau contenant les sommets du graphe

critmarge(p, Head)
```

6.6 Affichage des résultats

afficher "La durée minimale du projet est de: Head[p].datetard"
for $i \leftarrow 1, n$ **do**
  afficher "La tache i commencera au plus tot a la date: $Head[i].datetot$, se terminera au plus tard a la date $Head[i].datetard$"
  if $Head[i].estCritique$ **then**
    afficher "C'est une tache critique, il n'y a pas de marge"
  else
    afficher "La marge pour cette tâche est de:" + $Head[i].datetard - Head[i].datetot$"
  end if
end for