Tri topologique, méthode P.E.R.T.

L'objet de la méthode P.E.R.T. est la planification dans le temps d'un certain nombre de tâches (par exemple, les interventions des divers corps de métier participant dans un chantier) liées entre elles par des contraintes de précédence du genre : « telle tâche ne peut être commencée que si telle autre tâche a été entièrement exécutée ». Les données sont

- l'ensemble des tâches auxquelles on s'intéresse. Nous supposerons les avoir numérotées et, dans les explications suivantes, nous écrirons T_i à la place de « la tâche numéro i » ;
- la durée D_i de chaque tâche T_i ;
- un ensemble de couples $T_i \triangleright T_j$ (lisez « la tâche i doit être terminée pour que la tâche j puisse être commencée ») définissant la relation de précédence.

Nous cherchons deux choses :

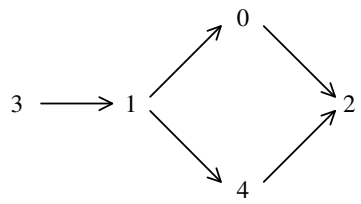
A. Un arrangement séquentiel des tâches qui respecte les contraintes de précédence.

B. Pour chaque tâche, une date de commencement au plus tôt, tenant compte des durées de toutes les tâches qui doivent être achevées avant que la tâche en question puisse commencer.

Notez bien que *A* et *B* sont les réponses à deux questions très différentes. Dans *A*, on vous demande de trouver une des dispositions séquentielles possibles (il y en a généralement plusieurs), comme si une seule personne ou équipe devait réaliser toutes les tâches. On appelle cela le *tri topologique* des tâches selon la relation de précédence, qui est une relation d'ordre partiel. Dans *B*, il s'agit au contraire de trouver à quelle date chaque tâche pourrait commencer si celles qui doivent la précéder étaient exécutées sans prendre aucun retard et en exploitant tous les parallélismes possibles.

On peut représenter la relation $\triangleright$ par un graphe. Les sommets correspondent aux tâches ; chaque relation $T_i \triangleright T_j$ donne lieu à un arc reliant le sommet i au sommet j.

Par exemple, le graphe ci-dessuos exprime les contraintes $T_3 \triangleright T_1$, $T_1 \triangleright T_0$, $T_1 \triangleright T_4$, $T_0 \triangleright T_2$ et $T_4 \triangleright T_2$:



DONNEES INITIALES. La lecture des données permet d'associer *immédiatement* à chaque tâche i :

- sa durée D_i ;
- le *nombre* P_i des prédécesseurs immédiats de T_i (les tâches T_k qui figurent dans des couples $T_k \triangleright T_i$) ;
- l'*ensemble* S_i des successeurs immédiats de T_i (les tâches T_j figurant dans des couples $T_i \triangleright T_j$). Pour la représentation de cet ensemble, voir la section *Ensembles* à la fin du sujet.

D, P et S sont des tableaux[1] à n éléments (les éléments de D et P sont des nombres, ceux de S sont des ensembles de nombres). On introduira en outre un tableau C défini par :

- C_i : date de commencement au plus tôt de T_i.

Ce tableau sera initialisé par des zéros. Modifié à chaque tour de l'algorithme, il contiendra en fin d'exécution le résultat cherché.

ALGORITHME. On gère un *ensemble de Tâches Sans Prédécesseur* (*TSP*, pour les intimes). Pour sa représentation, voir la section *Ensembles* à la fin du sujet.

Initialisation.

Placer dans *TSP* toutes les tâches sans prédécesseur.

---

[1] Notez bien qu'il n'y a pas besoin ici d'un tableau nommé T. Comme on l'a dit, « T_i » n'est qu'un raccourci commode pour dire « la tâche numéro i ».

Itérations.

Tant que *TSP* n'est pas vide :

- choisir une tâche T_i de *TSP* (peu importe laquelle), indiquer qu'elle peut être exécutée et l'enlever de *TSP*. Sa date de commencement au plus tôt n'est autre que la valeur actuelle de C_i.
- parcourir l'ensemble S_i des successeurs de T_i. Pour chaque élément T_j de S_i faire
  - $C_j \leftarrow$ max { C_j , $C_i + D_i$ }
  - $P_j \leftarrow P_j - 1$
  - si $P_j = 0$, faire entrer T_j dans *TSP*

Conclusion.

Les itérations s'arrêtent lorsque *TSP* est vide (cela finit toujours par arriver). Alors :

- si toutes les tâches ont été exécutées, le programme a fini son travail ;
- sinon, c'est qu'il y a une boucle dans les relations de précédence ; il faut annoncer cette anomalie.

PROGRAMMATION. Votre programme devra effectuer les opérations suivantes :

1. Lire le nombre *n* de tâches, puis la durée D_i de chaque tâche T_i.

2. Lire des couples de nombres i j jusqu'à la rencontre d'une valeur conventionnelle indiquant la fin. Chacun de ces couples exprime une contrainte $T_i \triangleright T_j$. En même temps, construire de proche en proche les tableaux *S* et *P* (la lecture de la relation $T_i \triangleright T_j$ doit modifier S_i et P_j).

3. Exécuter l'algorithme décrit ci-dessus, afin de trouver la date de commencement au plus tôt des différentes tâches. Affichez des couples i C_i où *i* est un numéro de tâche et C_i sa date de commencement au plus tôt. Cet affichage se fera au moment où la tâche *i* est extraite de *TSP*, ce qui affichera les tâches dans l'un des arrangements séquentiels possibles.

Pour terminer, affichez la date d'achèvement (au plus tôt) du projet tout entier.

EXEMPLE D'EXECUTION. Les interventions de l'utilisateur sont soulignées :

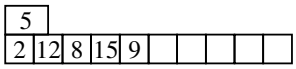
```
Nombre de taches : 5
durée de la tache 0 : 20
durée de la tache 1 : 15
durée de la tache 2 : 30
durée de la tache 3 : 10
durée de la tache 4 : 25
donne des couples i j (-1 pour terminer) :
3  1    1  0    1  4    0  2    4  2    -1

Tri et dates des taches :
   tache 3 (date : 0)
   tache 1 (date : 10)
   tache 4 (date : 25)
   tache 0 (date : 25)
   tache 2 (date : 50)
Date d'achèvement du projet : 80
```

ENSEMBLES. La méthode expliquée ci-dessus comporte un sous-problème clairement identifié : comment représenter des *ensembles* de nombres, et leurs opérations associées :

- faire entrer un nombre dans un ensemble ;
- faire sortir un nombre d'un ensemble ;
- détecter qu'un ensemble est vide ou rendre vide un ensemble ;
- effectuer une certaine action pour chaque membre d'un ensemble.

Puisque nous connaissons un majorant *M* du nombre possible d'éléments, nous pouvons représenter ici un ensemble par un tableau déclaré de taille *M*, contenant les valeurs de l'ensemble, associé à une variable entière indiquant le nombre effectif de ses éléments.

Par exemple, avec $M = 10$, l'ensemble { 2, 12, 8, 15, 9 } sera représenté par la structure :



☺ ☹