Projet de Programmation Nécessaire

Méthode Potentiels-Tâches

- Pour mener à son terme un projet, certaines tâches sont requises.
- Ces différentes tâches présentent de fortes interdépendances qu'il faut gérer au mieux.
- Autrement dit, il faut éviter toute interférence de ces tâches entre elles.
- Il est donc d'usage d'adjoindre au cahier des charges un agenda des tâches.

La coordination des tâches conduit au respect de:

Contraintes de préséances Certaines tâches dépendent souvent de tâches devant être exécutées avant elles; seulement après la fin de la dernière des tâches qui les précèdent, leurs exécutions peuvent être envisagées;

Contraintes absolues Certaines tâches doivent être achevée avant une date précise;

Contraintes d'exclusion mutuelle Certaines tâches ne peuvent s'effectuer simultanément. Par exemple, elles partagent une même ressource.

Ce projet se concentre sur les contraintes de préséance.

Considérons la réalisation d'une salle de bain.

Les tâches sont:

- dépose des sanitaires (DS) et du carrelages existants (DC)
- pose du nouveau carrelage (PC)
- peinture des murs non carrelés (P)
- mise aux normes électriques (E)
- pose des sanitaires (PS)
- pose du mobilier (PM) et de l'éclairage (PE)

On va établir un tableau de préséance:

Tâches	**Durée**	**Préséance**
DS	2	-
DC	4	DS
PC	4	DC, E
P	5	PC
E	3	DC, DS
PS	3	DS, PC
PE	1	P
PM	2	P, PS

À partir duquel on construit un graphe de préséances $G = (S, A)$ tel que:

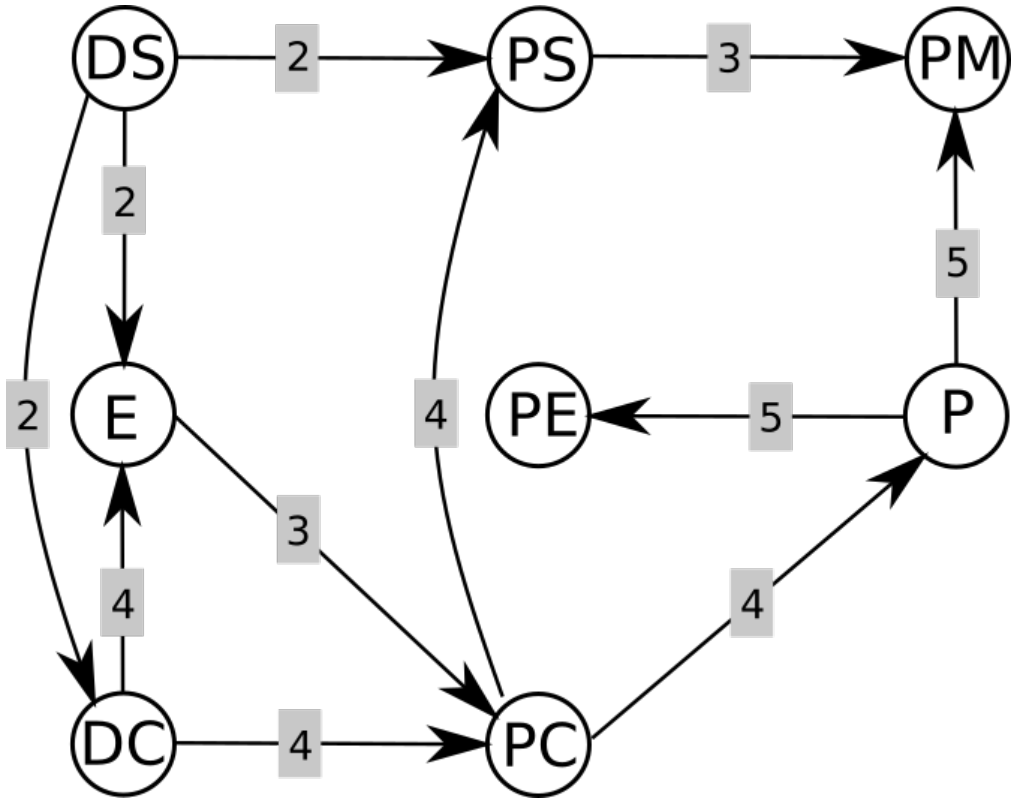
- Les nœuds $s \in S$ sont les tâches
- Les arcs $a \in A$ les préséances où l'orientation énonce la préséance:

  la source précède chronologiquement le puits

Les arcs sont pondérés par la durée de la tâche source:

Soit $a = (x \rightsquigarrow y) \in A$ *avec* $x, y \in S$ *de durée resp.* d_x *et* d_y

Alors la pondération de a vaut $w_a = d_x$


DS
PS
PM
E
PE
P
DC
PC
2
2
3
2
5
4
5
3
4
4
4

Il s'agit d'un graphe orienté acyclique au sein duquel il faut élaguer les chemins impraticables et mettre en avant ceux obligatoires.

On observe facilement qu'une tâche ne peut être effectuée tant qu'il reste ne serait-ce qu'une des tâches qui la précède en cours de traitement ou devant être traitée.

Ainsi le début possible d'une tâche est défini par le chemin le plus long puisque les arcs sont pondérés par les durées de leurs sources.

Ceci nous amène à la définition de rang d'un sommet.

Dans un graphe orienté acyclique $G = (S, A)$, le rang d'un sommet s est la longueur du chemin le plus long menant à s.

Définition (rang)

Soient le sommet $p \in S$ *et* $\Gamma^{-1}(p) = \{s \in S : (s \rightsquigarrow p) \in A\}$

Le rang est défini par:

$$\forall p \in S : \Gamma^{-1}(p) = \emptyset \implies \text{rang}(p) = 0$$

$$\forall p \in S : \Gamma^{-1}(p) \neq \emptyset \implies \text{rang}(p) = 1 + \max_{s \in \Gamma^{-1}(p)} \text{rang}(s)$$

Déroulons le calcul du rang sur notre exemple:

Tâches	**Durée**	**Préséance**	**Rang**
DS	2	-	0
DC	4	DS	U
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	U
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	U
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	U
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	U
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	U
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	2
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	U
P	5	PC	U
E	3	DC, DS	2
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	3
P	5	PC	U
E	3	DC, DS	2
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	3
P	5	PC	U
E	3	DC, DS	2
PS	3	DS, PC	U
PE	1	P	U
PM	2	P, PS	U

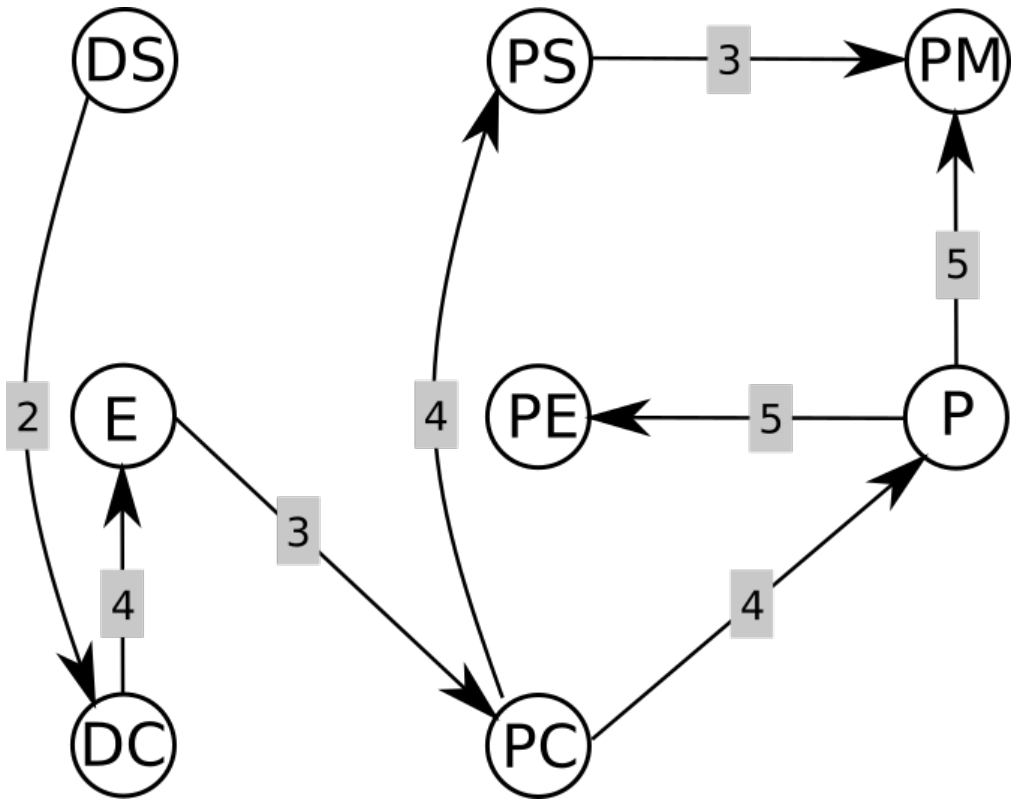
Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	3
P	5	PC	4
E	3	DC, DS	2
PS	3	DS, PC	4
PE	1	P	U
PM	2	P, PS	U

Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	3
P	5	PC	4
E	3	DC, DS	2
PS	3	DS, PC	4
PE	1	P	U
PM	2	P, PS	U

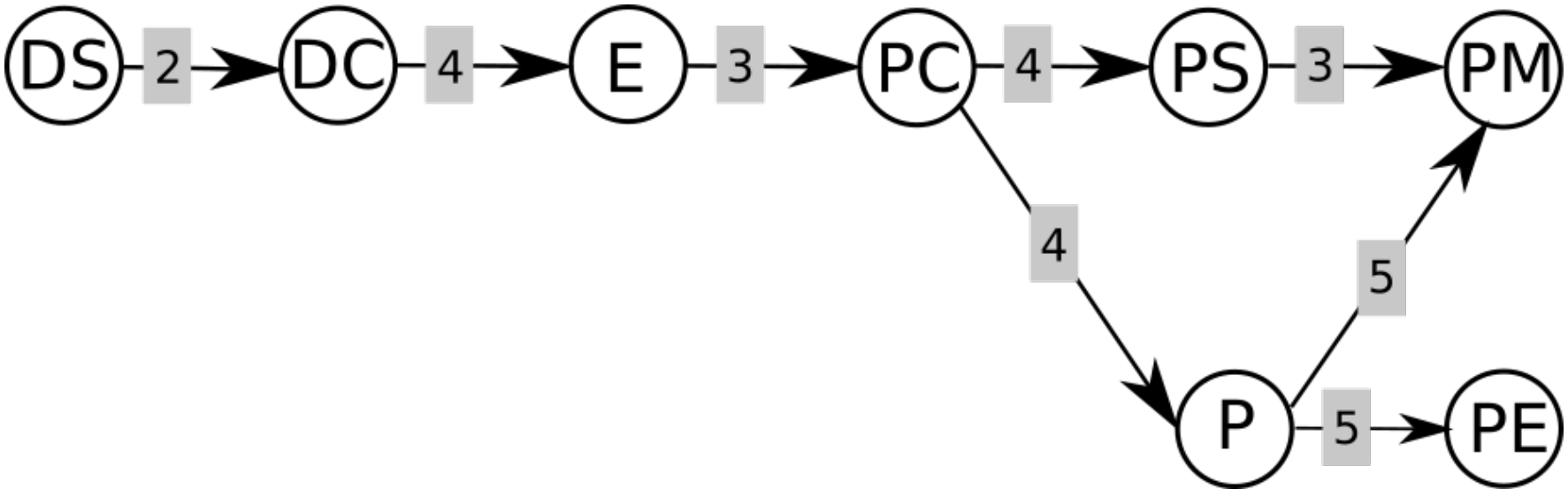
Tâches	Durée	Préséance	Rang
DS	2	-	0
DC	4	DS	1
PC	4	DC, E	3
P	5	PC	4
E	3	DC, DS	2
PS	3	DS, PC	4
PE	1	P	5
PM	2	P, PS	5

Tâches	**Durée**	**Préséance**	**Rang**
DS	2	-	0
DC	4	DS	1
E	3	DC, DS	2
PC	4	DC, E	3
P	5	PC	4
PS	3	DS, PC	4
PE	1	P	5
PM	2	P, PS	5

Muni des rangs des nœuds du graphe, il est apparaît aisément que certains arcs sont inutiles: $(DS \rightsquigarrow E)$ $(DS \rightsquigarrow PS)$ $(DC \rightsquigarrow PC)$



En réorganisant la présentation du graphe:



Remarque

Il y manque les durées des tâches finales PE et PM.

À partir du graphe obtenu après calcul des rangs et élagage, on peut déterminer un agenda des travaux avec des dates limites de début et de fin pour chaque tâche.

C'est le rôle jouer par les diagrammes de PERT (en anglais : *program evaluation and review technique*) une méthode de gestion de projet, d'ordonnancement et de planification développée aux États-Unis par la Navy dans les années 1950.

Il existe deux grandes familles de diagramme de PERT:

- La méthode PERT *potentiel-étapes* la plus ancienne, ne nous concerne pas pour ce projet;
- La méthode PERT *potentiel-tâches* (PERT-PT) créée en 1958 par M.B. Roy, sous l'acronyme MPM (Méthode des Potentiels Metra),

Nous allons reprendre le graphe précédent $G = (S, A)$ et le compléter pour qu'il devienne un diagramme de PERT potentiel-tâches.

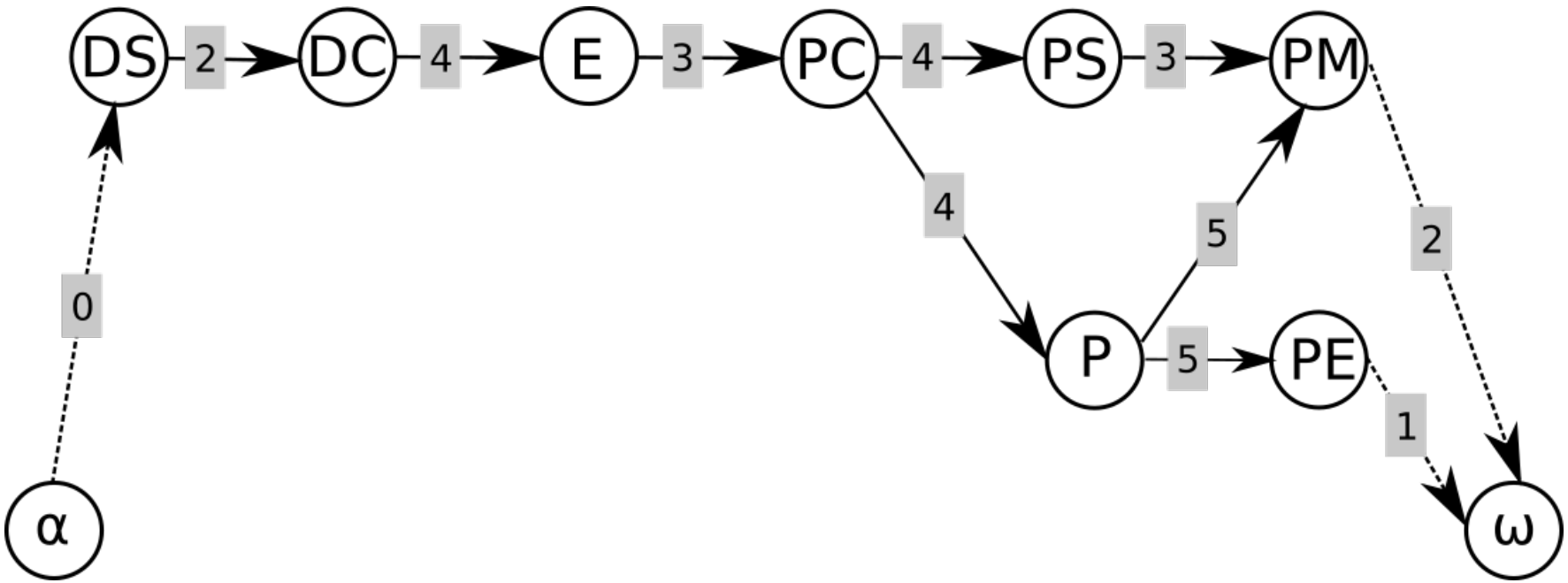
Remarque

Il est à noter que dans les diagrammes de PERT, les sommets sont les états décrivant l'avancement des travaux et les arcs sont pondérés par les durées des tâches en cours d'exécution qu'ils représentent.

L'altération que nous apportons n'a pas de conséquences majeures sur l'interprétation finale à donner au diagramme PERT.

On complète le graphe en y adjoignant deux sommets fictifs:

- α pour marquer le début du projet de rang -1, qui n'a donc pas de tâches antécédentes et qui a pour tâches successeuses celles de rang 0;
- ω pour marquer la fin du projet de rang $1 + \max_{s \in S} \mathit{rang}(s)$, qui a pour tâches antécédentes les tâches de rang maximal et qui n'a pas de tâches successeuses.


DS
DC
E
PC
PS
PM
P
PE
α
ω
2
4
3
4
3
0
4
5
5
2
1

Cette méthode permet de déterminer les dates et les marges d'un projet:

- Calculer les dates au plus tôt, au plus tard,
- Calculer les marges libres, les marges totales,
- Déterminer le chemin critique

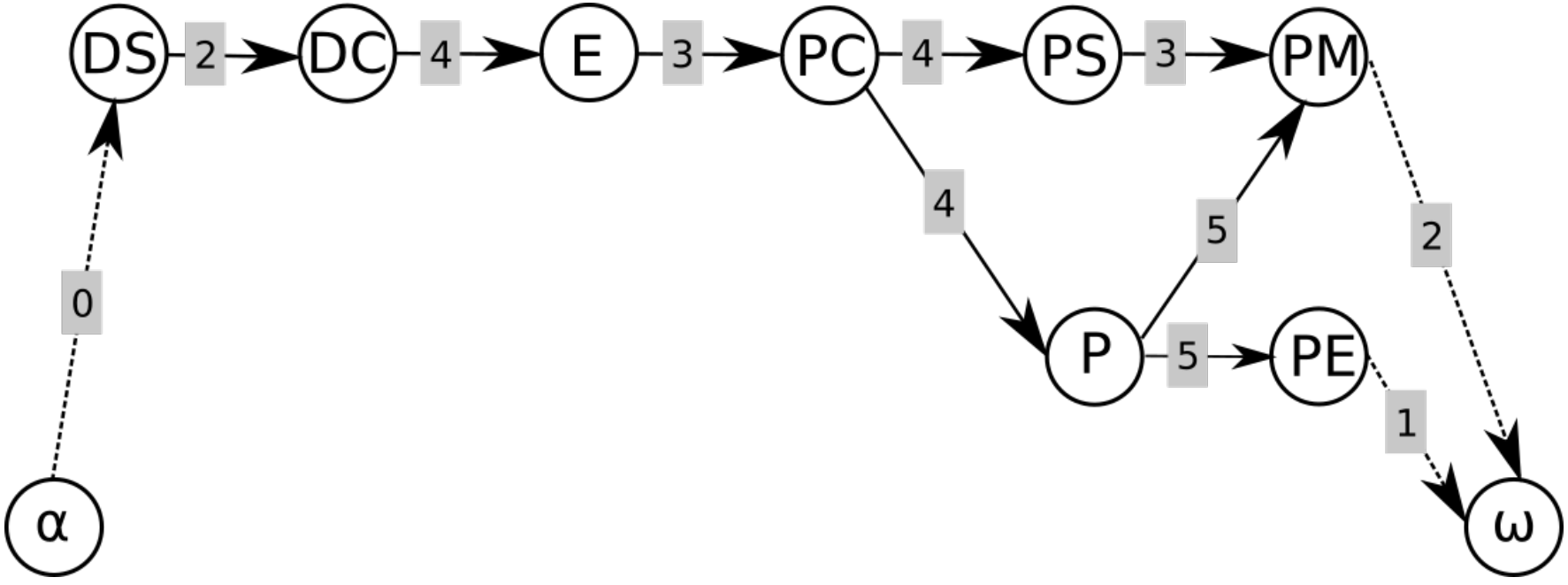
Puisqu'une tâche ne peut commencer avant que toutes celles qui la précèdent aient terminé, sa *date au plus tôt* correspond au chemin le plus long menant à cette tâche depuis le sommet α.

Définition (Date au plus tôt)

Soient le graphe $G = (S, A)$, la tâche $p \in S$ et $\Gamma^{-1}(p)$ l'ensemble des antécédentes de p.

La date au plus tôt de p est:

$$\begin{cases} d_{tôt}(p) = 0 & si\ p = \alpha \\ d_{tôt}(p) = max_{s \in \Gamma^{-1}(p)} \left(d_{tôt}(s) + durée(s)\right) & si\ p \neq \alpha \end{cases}$$



tâches	DS	DC	E	PC	P	PS	PE	PM	ω
$d_{\text{tôt}}$	0	2	6	9	13	13	18	18	20

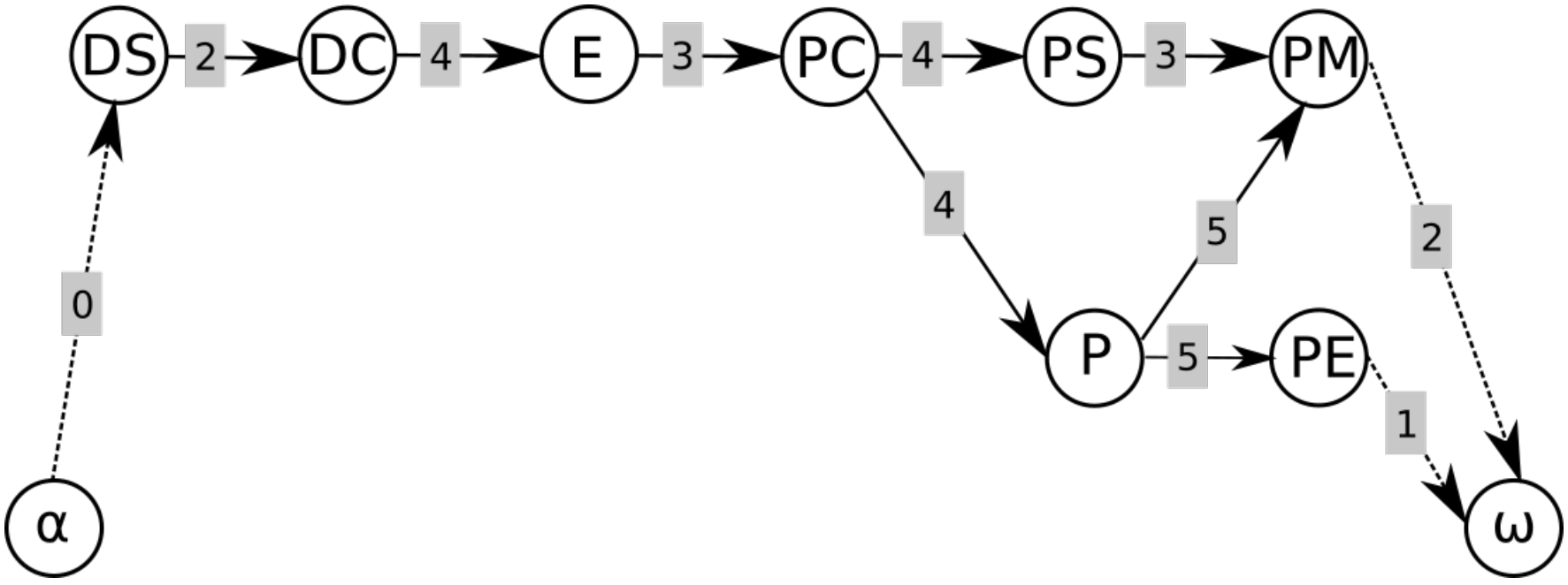
Si l'on sait quand *au plus tôt* démarrer chaque tâche, on peut également souhaiter connaître quand *au plus tard* l'on peut démarrer chacune d'elles.

Le principe du calcul des *dates au plus tard* est la réciproque de celui des dates au plus tôt; on commence par la fin.

Définition (Date au plus tard)

Soient le graphe $G = (S, A)$, *la tâche* $s \in S$ *et* $\Gamma(s)$ *l'ensemble des successeuses de* s.

$$\begin{cases} d_{tard}(s) = d_{tôt}(s) & si\ s = \omega \\ d_{tard}(s) = \min\limits_{p \in \Gamma(s)} (d_{tard}(p)) - durée(s) & si\ s \neq \omega \end{cases}$$

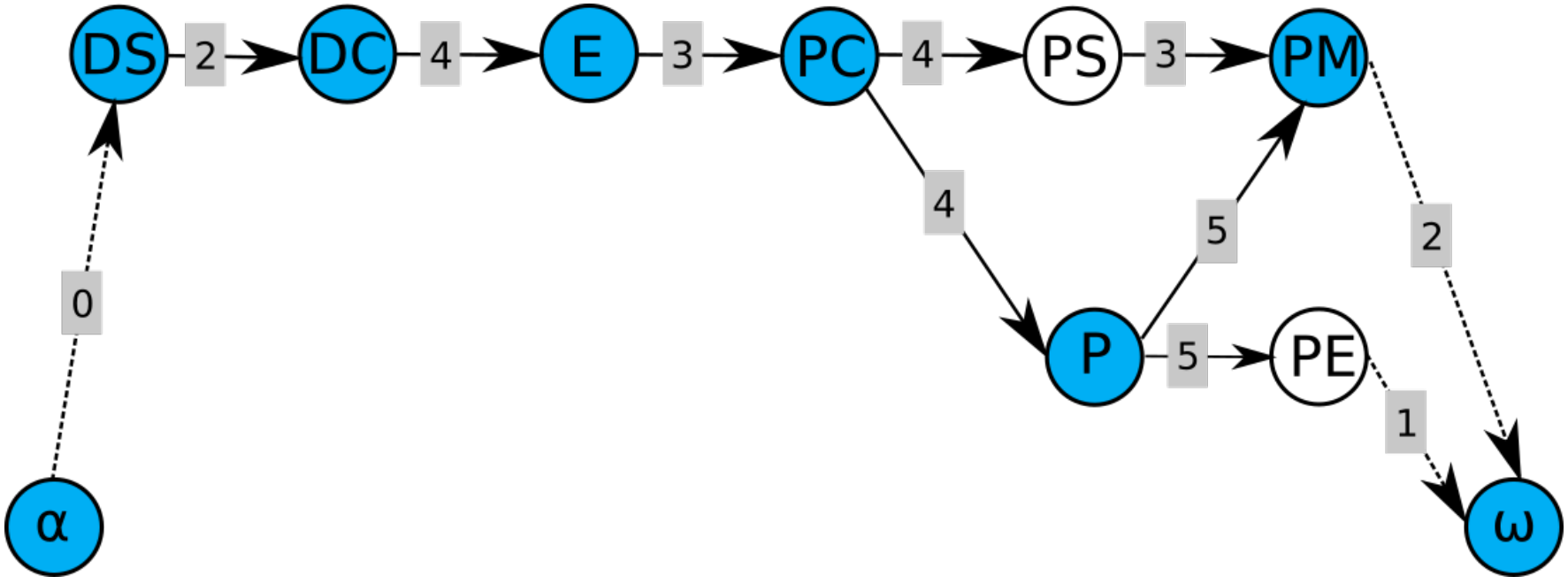


tâches	DS	DC	E	PC	P	PS	PE	PM	ω
$d_{\text{tôt}}$	0	2	6	9	13	13	18	18	20
d_{tard}	0	2	6	9	13	15	19	18	20

Le calcul des différences entre les dates au plus tôt et les dates au plus tard sont appelées les **marges totales**.

Les tâches ayant des marges totales nulles forment le **chemin critique**.

Le non respect des dates des tâches du chemin critique entraîne des retards.



tâches	DS	DC	E	PC	P	PS	PE	PM	ω
$d_{\text{tôt}}$	0	2	6	9	13	13	18	18	20
d_{tard}	0	2	6	9	13	15	19	18	20
MT	0	0	0	0	0	2	1	0	0

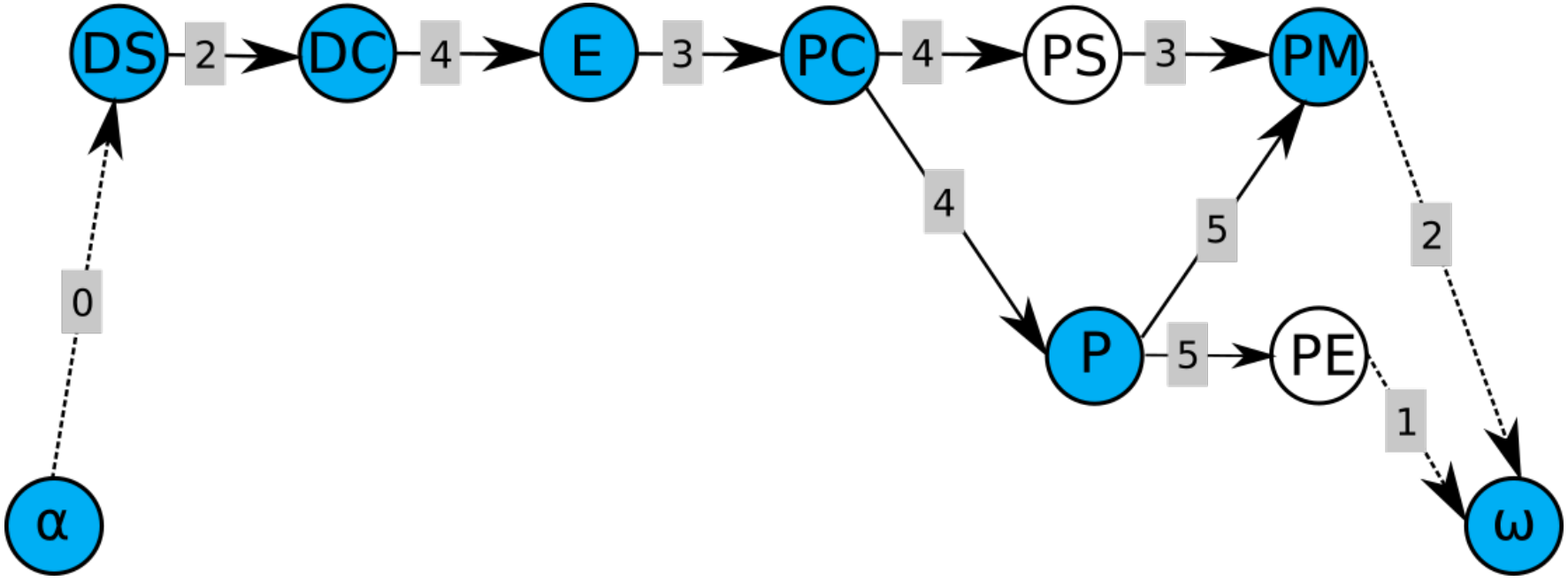
Enfin on voudra souvent connaître les latitudes dont on dispose lors des travaux.

Les *marges libres* répondent à ce souhait.

Définition (Marge libre)

Soient le graphe $G = (S, A)$, *la tâche* $s \in S$ *et* $\Gamma(s)$ *l'ensemble des successeuses de* s.

$$\begin{cases} ML(s) = 0 & si\ s = \omega \\ ML(s) = \min\limits_{p \in \Gamma(s)} (d_{tôt}(p)) - d_{tôt}(s) - durée(s) & si\ s \neq \omega \end{cases}$$



tâches	DS	DC	E	PC	P	PS	PE	PM	ω
$d_{\text{tôt}}$	0	2	6	9	13	13	18	18	20
d_{tard}	0	2	6	9	13	15	19	18	20
MT	0	0	0	0	0	2	1	0	0
ML	0	0	0	0	0	2	1	0	0

Sur le chemin critique les marges totale et libre sont identiques et nulles. Il n'y a pas de latitude possible pour les tâches de ce chemin.

Pour les autres tâches qui ont une marge libre non nulle, on peut utiliser ces marges pour retarder ces tâches sans pour autant retarder l'ensemble des travaux.

Vous allez écrire en **C** une application qui va:

- Construire le graphe d'un projet
  - Lire un fichier contenant les tâches d'un projet, leurs durées et leurs préséances,
  - Ranger ces données dans un graphe,
  - Calculer les degrés d'entrée et de sortie de chacune des tâches,
- Visualiser les tâches,
- Calculer les rangs de chacune des tâches,
- Visualiser les tâches suivant cet ordre,
- Supprimer les arcs inutiles et les afficher,
- Calculer les dates au plus tôt, les dates au plus tard, les marges totales et les marges libres.
- Afficher ces valeurs ainsi que le chemin critique.

Modélisation d'une tâche

```
1  #define UNDEF -2
2  /** Des redondances possibles avec d'autres TAs ! */
3  struct job_t {
4    char * title;                    // Nom de la tâche
5    double life;                     // Durée de la tâche
6    int input_degree;                // Son degré de dépendance
7    int output_degree;               // Les tâches qui en dépendent
8    int rank;                        // Rang de la tâche
9    int dyn_input_degree;            // Facilité de prog
10   struct list_t * precedence;      // Les tâches précédentes
11   struct list_t * posteriority;    // Les tâches ultérieures
12   double au_plus_tot;              // Date au plus tôt
13   double au_plus_tard;             // Date au plus tard
14   double marge_totale;             // Marge totale
15   double marge_libre;              // Marge libre
16   bool critique;                   // Une tâche critique ?
17 };
```

*Les fonctions **publiques** de* `struct job_t`

```
1  struct job_t * new_empty_job();
2  struct job_t * new_job(char * title);
3  void free_job(struct job_t ** ptrJ);
4
5  void view_job(struct job_t * J);
6
7  char * get_job_tilte(struct job_t * J);
8  void set_job_title(struct job_t * J, char * title);
9
10 double get_job_life(struct job_t * J);
11 void set_job_life(struct job_t * J, double life);
12
13 int get_job_iDegree(struct job_t * J);
14 void set_job_iDegree(struct job_t * J, int iDegree);
15 void incr_job_iDegree(struct job_t * J);
16 void decr_job_iDegree(struct job_t * J);
```

*Les fonctions **publiques** (Suite et fin)*

```
1  int get_job_oDegree(struct job_t * J);
2  void set_job_oDegree(struct job_t * J, int oDegree);
3  void incr_job_oDegree(struct job_t * J);
4  void decr_job_oDegree(struct job_t * J);
5
6  int get_job_rank(struct job_t * J);
7  void set_rank(struct job_t * J, int rank);
8
9  int titleJobCmp(struct job_t * J1, struct job_t * J2);
10 int iDegreeJobCmp(struct job_t * J1, struct job_t * J2);
11 int oDegreeJobCmp(struct job_t * J1, struct job_t * J2);
12 int rangJobCmp(struct job_t *J1, struct job_t * J2);
```

*Les fonction **publiques** d'élagage et de calcul des rangs et des marges*

```
1 void ranking(struct list_t * G);
2 void prune(struct list_t * G);
3
4 void marges(struct list_t * G);
```

Listes doublement chaînées génériques

```
1  struct list_t {
2    struct elmlist_t * head;
3    struct elmlist_t * tail;
4    int numelm;
5  };
6  // Create an empty list
7  struct list_t * new_list();
8  // Delete list, its elements and possibly the data
9  void del_list(struct list_t ** ptrL, void (*ptrf) ());
10 // Clean list; delete its elments but keep data and the list
11 void clean(struct list_t * L);
12 // Is list L empty ?
13 bool is_empty(struct list_t * L);
14 // Gimme the head of L
15 struct elmlist_t * get_head(struct list_t * L);
16 // Gimme the tail of L
17 struct elmlist_t * get_tail(struct list_t * L);
```

Les fonction ***publiques*** *de* `struct list_t`

```
1  // Gimme the number of elements of L
2  int get_numelm(struct list_t * L);
3  // Take out the data D of the list L if it is present
4  void take_out(struct list_t * L, void * D);
5  // Add a element holding data to the head of L
6  void cons(struct list_t * L, void * data);
7  // Add  a element holding data to the tail of L
8  void queue(struct list_t * L, void * data);
9  // Insert in L a element holding data wrt order given by
       cmp_ptrf
10 void ordered_insert(struct list_t * L, void * data, int (*
       cmp_ptrf)());
11 // Do the quick sort
12 void quick_sort(struct list_t * L, int (*cmpFct)());
13 // View list
14 void view_list(struct list_t * L, void (*ptrf)());
15 // Return a ptr to element which data is key else NULL
16 void find(struct list_t * L, void ** ptrKey, int (*cmpFct)(),
       void (*delFct)());
```

La fonction `main` ne peut être changée.

Au code source régulièrement déposé sur le serveur GitLab de l'université, vous adjoindrez un README au format **Markdown** contenant *algorithmes* du tri rapide et du calcul du rang et expliquant les modifications[1] que vous avez apportées au canevas fourni.

Have fun

[1] S'il y en a.