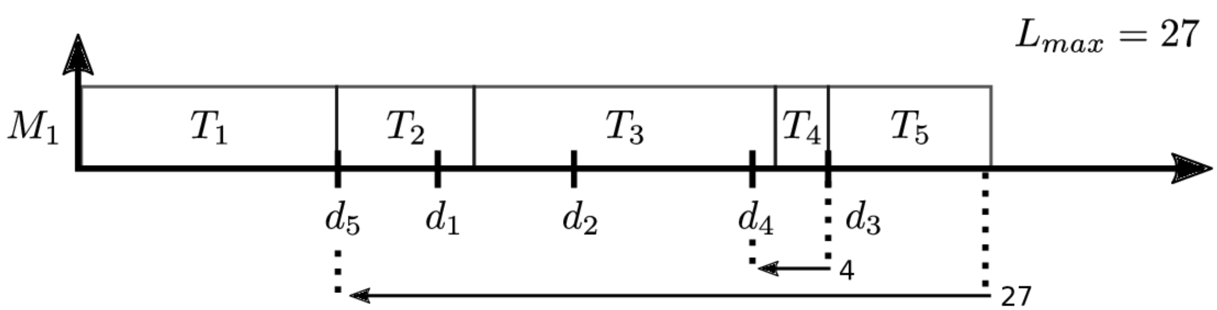
Ordonnancement sur une machine

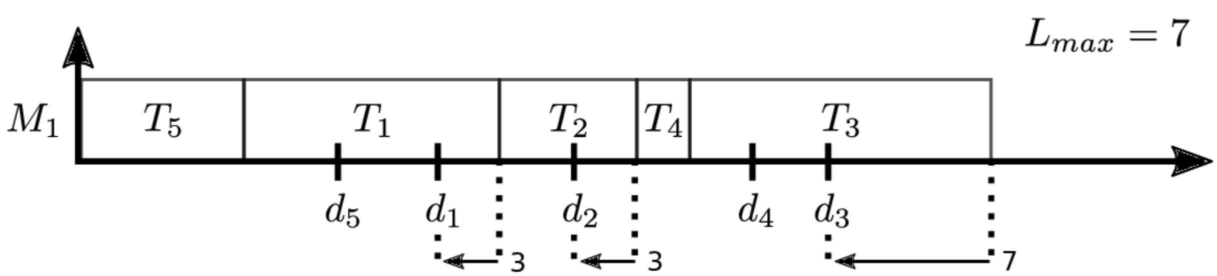
Prise en compte de due time, release time et deadline

4 Le problème $1||L_{max}$

Ce problème consiste dans la minimisation du retard maximal des taches $L_{max} = max(l_i)$ avec $l_i = c_i - s_i$. Ce problème est facile car un algorithme optimal consiste de ranger les taches par rapport à leur temps souhaitée de fin (**due time**) en ordre croissant.

$$p = \{10, 6, 12, 2, 7\}$$
$$d = \{14, 20, 30, 26, 10\}$$



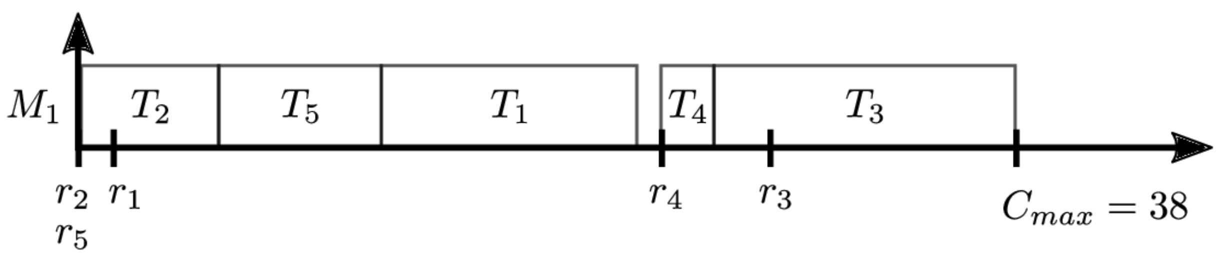


<u>**A faire:**</u>
1.1. Écrire une fonction `S__Lmax` dans le fichier `sm_functions.jl` implémentant l'algorithme pour ce problème.
1.2. Tester l'algorithme sur les données $p = \{5, 3, 2, 4, 1, 3, 6, 4\}$, $d = \{15, 14, 8, 16, 20, 10, 28, 25\}$ dans un nouveau ficher `p4.jl`. La variable pour d sera nommée `due_time`. Inclure-le également dans l'appel de la fonction `plotScheduleOneMachine` (regarder la déclaration de cette fonction dans `plot_functions.jl` pour trouver le nom de la variable).

5 Le problème $1|r_i|C_{max}$

Ce problème correspond à la minimisation du temps total du traitement des taches quand elles ne sont pas disponibles dès le début. Chaque tache est donc caractérisée par son temps de traitement (p_i) et son temps de disponibilité (r_i), ou release time. L'algorithme de résolution de manière optimale consiste dans **l'ordonnancement des taches par rapport à l'ordre croissante des temps de disponibilité**.

$$p = \{10, 6, 12, 2, 7\}$$
$$r = \{2, 0, 28, 24, 0\}$$



A faire:
2.1. Écrire une fonction `S_r_Cmax` dans le fichier `sm_functions.jl` implémentant l'algorithme de résolution de ce problème. Adapter la fonction `decode_order` afin de accepter un paramètre `release_time` avec une valeur par défaut `nothing` (l'équivalent de Null ou None dans d'autre langages de programmation).
2.2. Adapter aussi la fonction `admissible` afin de vérifier que aucune tache ne commence avant son temps de disponibilité (inclure aussi un paramètre `release_time` avec une valeur par défaut `nothing`).
2.3. Tester l'algorithme sur les données $p = \{2, 3, 1, 2, 4\}$, $r = \{8, 0, 4, 1, 7\}$ dans un nouveau ficher `p5.jl`. La variable pour r sera nommée `release_time`. Inclure-le également dans l'appel de la fonction `plotScheduleOneMachine` à travers un appel explicite de cette variable. (ex. `plotScheduleOneMachine(start_time, complete_time, r = release_time)`)

6 Le probleme $1|\tilde{d}_i|<C>$

Ce problème est similaire à $1||<C>$, sauf qu'on considère que chaque tache doit finir avant un temps limite, qu'on appelle **deadline**. Ce problème est aussi simple à résoudre, car il existe un algorithme de complexité linéaire pour trouver la solution optimale.

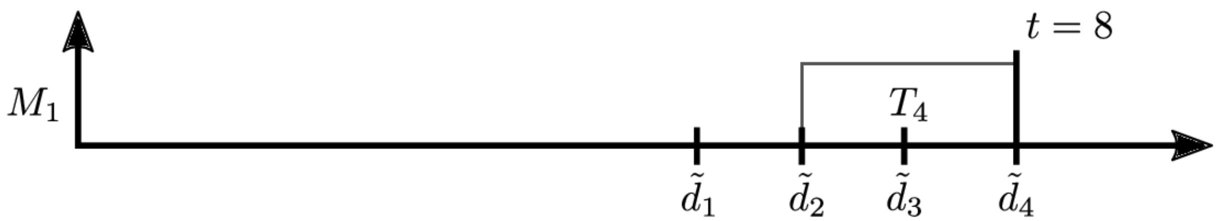
L'algorithme de Smith contient les pas suivants:

1. Initialiser $U = \{1, ..., n\}$, $t = \sum_1^n p_i$ et $O = \{\}$;
2. Calculer $V = \{i \in U \mid \tilde{d}_i \geqslant t\}$;

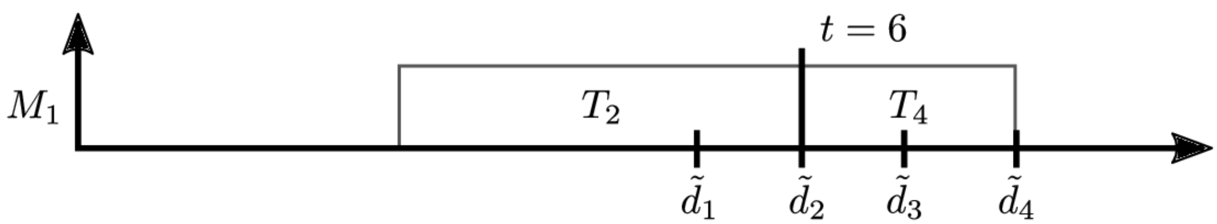
3. Si $V = \emptyset$ et $U \neq \emptyset$ alors il n'existe de solution O;
4. Si $V = U = \emptyset$ alors la solution O est optimale;
5. Trouver $j = \arg\max_{i \in V} p_i$;
6. Mettre à jour $U = U - \{j\}$, $O = \{j\} + O$ et $t = t - p_j$;
7. Revenir à 2.

$$p = \{2, 3, 1, 2\} \quad \tilde{d} = \{5, 6, 7, 8\}$$

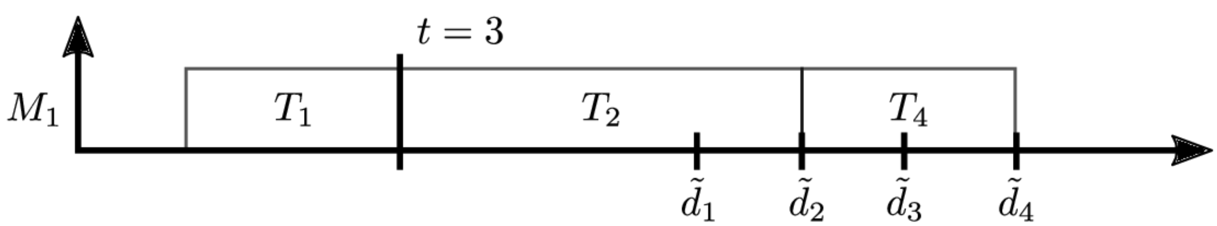
$$U = \{1, 2, 3, 4\},\ t = 8,\ V = \{4\} \quad \rightarrow \quad j = 4$$



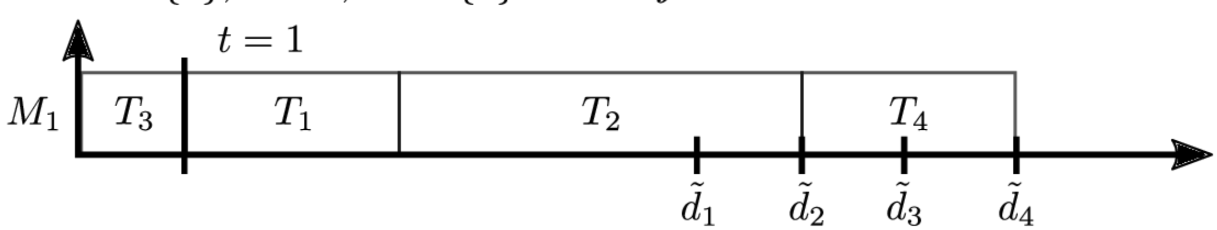
$$U = \{1, 2, 3\},\ t = 6,\ V = \{2, 3\} \quad \rightarrow \quad j = 2\ (p_2 > p_3)$$



$$U = \{1, 3\},\ t = 3,\ V = \{1, 3\} \quad \rightarrow \quad j = 1\ (p_1 > p_3)$$



$$U = \{3\},\ t = 1,\ V = \{3\} \quad \rightarrow \quad j = 3$$



A faire:
3.1. Écrire une fonction `S_dd_Cavg` dans le fichier `sm_functions.jl` implémentant l'algorithme de Smith. (voir notions utiles en bas)
3.2. Adapter la fonction `admissible` afin de vérifier que aucune tache ne se termine après sa date limite. Afin de pouvoir utiliser parfois le paramètre `release_time` ou le paramètre `deadline`, utiliser la syntaxe ci-dessous:

```
function admissible(start_time, complete_time; release_time = nothing,
↪  deadline = nothing)
```

3.3. Tester l'algorithme sur les données $p = \{2, 3, 1, 2, 4\}$, $\tilde{d} = \{5, 6, 8, 12, 10\}$ dans un nouveau ficher `p6.jl`. La variable pour $\tilde{d}$ sera nommée `deadline`. Inclure-le également dans l'appel de la fonction `plotScheduleOneMachine` (regarder la déclaration de cette fonction dans `plot_functions.jl` pour trouver le nom de la variable).

Notions utiles:
Un vecteur uni-dimensionnel peut être indexé par un vecteur d'indexes:

```
julia> y = [1,2,3,4,5]
5-element Array{Int64,1}:
1
2
3
4
5

julia> y[[2,3]]
2-element Array{Int64,1}:
2
3
```

Un vecteur uni-dimensionnel peut être indexé par un vecteur de booléens de même taille afin de récupérer que les indexes correspondant au valeurs `true`:

```
julia> y[[false, true, true, false, true]]
3-element Array{Int64,1}:
2
3
5
```

La fonction `map` peut générer ce vecteur de booléens en appliquant une fonction:

```
julia> map(x -> x % 2 == 1, y)
5-element Array{Bool,1}:
true
false
true
false
true
```

`x -> expression` est une façon d'écrire une fonction sans nom qu'on utilise que dans une expression. La fonction `map` applique cette fonction pour chaque élément du vecteur `y`.

La fonction `filter` permet également de récupérer les éléments d'un vecteur en appliquant une fonction booléenne sur chacun de ses éléments:

```
julia> filter(x -> x % 2 == 1, y)
3-element Array{Int64,1}:
1
3
5
```

La fonction `findmax` permet de trouver la valeur maximale et son index à partir d'un vecteur:

```
julia> findmax(y)
(5, 5)
```