JOBSHOP À 2 PIÈCES PAR PROG. DYNAMIQUE

Problème

On se place dans un atelier, ayant un nombre quelconque de machine mais seulement deux pièces à réaliser A et B.
Pour réaliser A on doit effectuer les tâches $(a_i)_{i\in[\![1..n]\!]}$
——— B ——— $(b_j)_{j\in[\![1..m]\!]}$

On note $(p_i)_{i\in[\![1..n]\!]}$ les durées resp. des tâches $(a_i)_{i\in[\![1..n]\!]}$
——— $(q_j)_{j\in[\![1..m]\!]}$ ——— $(b_j)_{j\in[\![1..m]\!]}$

Enfin on dira que les tâches a_i et b_j sont incompatibles si elles doivent être réalisées sur la même machine, et ne peuvent donc pas être réalisées en même temps.
Le but est de minimiser la date à laquelle les deux pièces sont terminées.

$J_m \mid n=2 \mid C_{max}$

- J_m : jobshop à un nombre quelconque de machines
- $n=2$: avec 2 tâches seulement ⚠ ce n'est pas le n de $a_1 \dots a_n$
- C_{max} : dernière date de fin.

Relation de récurrence

On note pour $i\in[\![1..n]\!]$, $j\in[\![1..m]\!]$ $t_{i,j}$ = le temps minimal pour exécuter les tâches $a_1, \dots, a_i$ dans cet ordre et les tâches $b_1, \dots, b_j$ dans cet ordre, le tout sans conflit.

Par convention $t_{0,0} = 0$

De plus $\forall i\in[\![1..n]\!]$ $t_{i,0} = \sum_{i'=1}^{i} p_{i'}$ (Il suffit de faire les tâches $a_1 \dots a_i$ toutes les unes après les autres, ça donne bien la somme des durées de ces tâches)

$\forall j\in[\![1..m]\!]$ $t_{0,j} = \sum_{j'=1}^{j} q_{j'}$ (idem)

$\forall i\in[\![1..n]\!]$, $\forall j\in[\![1..m]\!]$ → si a_i et b_j ne sont pas incompatibles

alors $t_{i,j} = \min(t_{i-1,j} + p_i ;\ t_{i,j-1} + q_j ;\ t_{i-1,j-1} + \max(p_i, q_j))$

- $t_{i-1,j} + p_i$: on avait déjà fait $a_1 — a_{i-1}$ et $b_1 — b_j$ en $t_{i-1,j}$ puis on fait a_i en p_i.
- $t_{i,j-1} + q_j$: on avait déjà fait $a_1 — a_i$ et $b_1 — b_{j-1}$ en $t_{i,j-1}$ au mieux, on fait ensuite b_j en q_j
- $t_{i-1,j-1} + \max(p_i, q_j)$: on avait déjà fait $a_1 — a_{i-1}$ et $b_1 — b_{j-1}$ en $t_{i-1,j-1}$ puis on fait a_i et b_j, qui commencent en même temps et qui sont toutes deux finies après $\max(p_i, q_j)$

→ si a_i et b_j sont incompatibles

alors $t_{i,j} = \min(t_{i-1,j} + p_i, t_{i,j-1} + q_j)$

algo

Le type de relation de récurrence, où l'on peut, à partir des résultats de plusieurs sous problèmes, résoudre le problème, laisse apparaître une structure de programmation dynamique et on a alors un algorithme tout trouvé.
J'écris ici une version non optimisée et qui ne retourne que la valeur (ici la durée) d'une solution optimale (ici un ordonnancement optimal) et non la solution elle même.
Pour améliorer ces deux points, voir la fiche sur le calcul d'un alignement entre deux mots qui est très semblable au calcul qu'on va faire ici.

Entrée P un tableau indexé par $[1..n]$ contenant les durées $(p_i)_{i\in[1..n]}$
Q ——— $[1..m]$ ——— $(q_j)_{j\in[1..m]}$
MA ——— $[1..n]$ indiquant la machine utilisée pour $(a_i)_{i\in[1..n]}$
MB ——— $[1..m]$ ——— $(b_j)_{j\in[1..m]}$

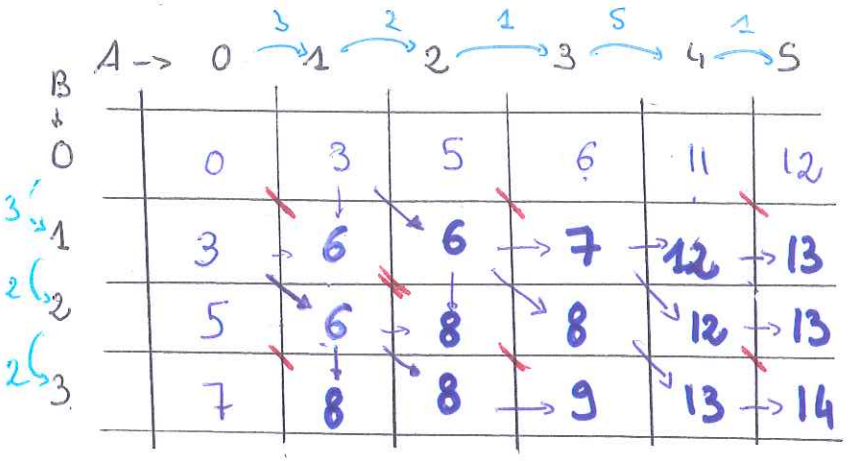
Sortie le temps minimal pour terminer les deux pièces.

```
TEMPS_MIN (P, Q, MA, MB)
    n ← taille de P
    m ← ——— Q
    T ← tableau indexé par [0..n] x [0..m]
    // initialisation
    T[0,0] ← 0
    Pour i de 1 à n
        T[i, 0] ← T[i-1, 0] + P[i]
    Pour j de 1 à m
        T[0, j] ← T[0, j-1] + Q[j]
    // réc. = remplissage du tableau
    Pour i de 1 à n
        Pour j de 1 à m
            Si MA[i] ≠ MA[j]   // on n'est pas face à un conflit
            alors T[i,j] ← min(T[i, j-1]+Q[j] ; T[i-1, j]+P[i] ; T[i-1, j-1]+max(P[i], Q[j]))
            sinon T[i, j] ← (———————————)
    // résultat
    retourner T[n, m].
```

ex

$m = 5$ $P = 3\ 2\ 1\ 5\ 1$ $MA = 1\ 2\ 1\ 3\ 1$

$m = 3$ $Q = 3\ 2\ 2$ $MB = 1\ 2$



incompatibilités
- sur la machine M_1
= ——— M_2
- initialisation
- nic

Donc ici il faut au min 14 unité de temps pour réaliser les pièces A et B.

En remontant les flèches dans le tableau, on trouve même un ordonnancemt réalisant cet optimum :

de 13 à 14 on a fait a_5 sur M_1
- 8 à 13 ——— a_4 — M_3
- 8 à 10 ——— b_3 — M_1) → pas d'incompatibilité
- 6 à 8 ——— b_2 — M_2
- 6 à 7 ——— a_3 — M_1) → ———
- 3 à 6 ——— b_1 — M_1
- 3 à 5 ——— a_2 — M_2) → ———
- 0 à 3 ——— a_1 — M_1

Ce qu'on peut aussi représenter comme suit.