Tri par piles

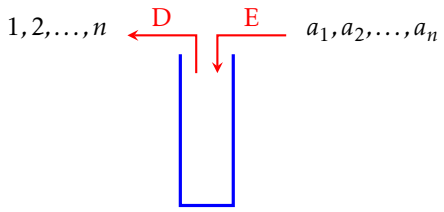
On suppose donné une classe *Pile* offrant les méthodes suivantes :

- `p.empty()` détermine si la pile `p` est vide ;
- `p.push(x)` empile `x` au sommet de la pile `p` ;
- `p.pop()` retourne et supprime le sommet de la pile `p` ;
- `p.peek()` retourne *sans le supprimer* le sommet de la pile `p`.

1. Tri avec une pile

Durant ce TD, on convient d'appeler *séquence* toute permutation des entiers d'un intervalle $[\![1,n]\!]$ avec $n \in \mathbb{N}$, et on dit qu'une séquence $a = (a_1, a_2, \ldots, a_n)$ est triable par une pile lorsqu'il est possible, à l'aide d'une pile initialement vide, d'ordonner ces éléments en utilisant les opérations suivantes :

- *empiler* l'élément suivant de la séquence d'entrée (E) ;
- *dépiler* le sommet de la pile en direction de la sortie (D).



Par exemple, la séquence $(4,1,3,2)$ peut être triée par la succession d'opérations EEDEEDDD.

Question 1.

a) parmi les séquences suivantes, lesquelles peuvent être triées par une pile ?

$$(2,4,1,3) \qquad (3,1,2,5,4) \qquad (4,5,3,7,2,1,6)$$

b) Montrer que lors d'un tri réussi, les éléments de la pile sont à tout moment rangés par ordre croissant (en partant du sommet).

c) Justifier que si une séquence est triable par une pile, il existe une unique manière de la trier, et en déduire une fonction `triPile(a)` qui prend en argument une séquence représentée par une liste a et qui retourne un booléen traduisant si la séquence est triable ou pas.

Question 2. Soient $1 \leqslant k \leqslant n$ deux entiers, a une séquence de longueur n et b une séquence de longueur k. On dit que b est un *motif* de a s'il existe $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$ tel que b et $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$ soient isomorphes pour la relation d'ordre. Par exemple, $(1,3,2,4)$ est un motif de $(3,1,\underline{2},8,\underline{5},\underline{4},7,\underline{9},6)$ car $(1,3,2,4)$ et $(2,5,4,9)$ sont isomorphes.

Montrer qu'une séquence a est triable par une pile si et seulement si $(2,3,1)$ n'est pas un motif de a.

Question 3. On note c_n le nombre de séquences triables de $[\![1,n]\!]$. À l'aide de la question précédente, prouver que

$$c_{n+1} = \sum_{k=0}^{n} c_k c_{n-k} \qquad \text{(avec la convention } c_0 = 1\text{)}.$$

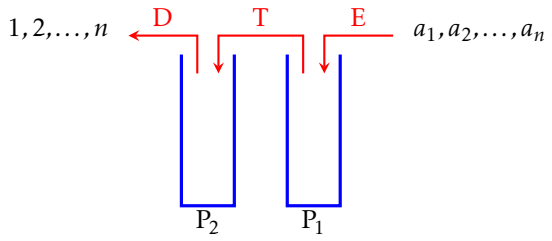
De cette relation il résulte que c_n est égal au n^{e} nombre de Catalan, dont la valeur est : $c_n = \dfrac{1}{n+1}\binom{2n}{n}$.

(Il n'est pas demandé d'établir cette égalité bien connue).

2. Tri avec deux piles en série

On dit qu'une séquence $(a_1, a_2, \ldots, a_n)$ est triable par deux piles en série lorsqu'il est possible à l'aide de deux piles d'ordonner la séquence $(a_1, a_2, \ldots, a_n)$ en utilisant les opérations suivantes :

- *empiler* dans la pile P_1 l'élément suivant de la séquence d'entrée (E) ;
- *transférer* un élément de la pile P_1 vers la pile P_2 (T) ;
- *dépiler* le sommet de la pile P_2 en direction de la sortie (D).



Question 4.

a) Montrer que la séquence $(2, 4, 3, 1)$ est triable au moins de deux manières différentes.

b) De combien de manière la séquence $(3, 2, 1)$ peut-elle être triée ?

c) Plus généralement, montrer que la séquence $(n, n-1, \ldots, 2, 1)$ est triable de 2^{n-1} façons différentes.

d) Montrer que la séquence $(2, 4, 3, 5, 7, 6, 1)$ n'est pas triable.

Un algorithme glouton

Le tri à l'aide d'une seule pile est facile à étudier car comme l'a montré la question 1.c, un seul algorithme est possible. La question précédente vient de montrer qu'avec deux piles en série le nombre de solutions peut être exponentiel : à chaque étape plusieurs choix peuvent être envisagés. On appelle algorithme *glouton* tout algorithme qui suit une heuristique permettant de faire ce choix. Cette heuristique ne garantit pas toujours de conduire à la solution souhaitée, mais doit donner des résultats acceptables dans un bon nombre de cas.

Question 5.

a) Montrer que si une séquence est triable par deux piles, le contenu de la seconde est toujours rangé par ordre croissant.

b) Montrer en donnant un contre-exemple que la première des deux piles n'est pas forcément rangée, ni par ordre croissant ni par ordre décroissant.

Fort de cette constatation nous allons nous intéresser à l'algorithme glouton qui consiste à effectuer à chaque étape le mouvement légal le plus à gauche possible, à savoir :

(i) l'opération D si le prochain élément devant sortir est au sommet de P_2 ;

(ii) dans le cas contraire, l'opération T si elle respecte la contrainte de croissance dans P_2 ;

(iii) dans le cas contraire, l'opération E s'il reste des éléments à empiler.

Cet algorithme échoue si à un moment donné du processus aucune action n'est possible.

Question 6.

a) Appliquer cet algorithme à la séquence $(2, 4, 3, 1)$.

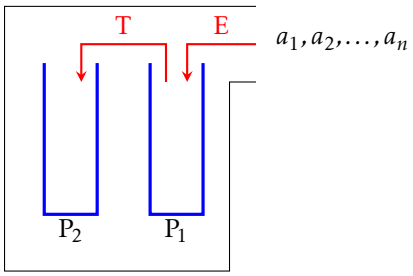
b) Montrer que toute séquence triable par une pile est aussi triable par deux piles en suivant cet algorithme.

c) Trouver une séquence de longueur 5 pour laquelle cet algorithme échoue, et vérifier qu'elle peut néanmoins être triée par deux piles.

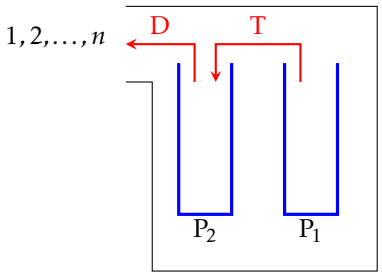
d) À l'aide de deux instances de la classe *Pile*, rédiger un algorithme `triPileLeftToRight(a)` qui prend en argument une séquence et qui retourne un booléen traduisant la réussite ou l'échec de cet algorithme pour trier `a`.

Tri par sas

Une autre approche du problème consiste à considérer l'ensemble des deux piles comme un « sas » : nul entier ne peut en sortir avant que l'ensemble des n entiers n'y soit entré.

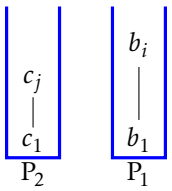


entrée des n éléments dans le sas

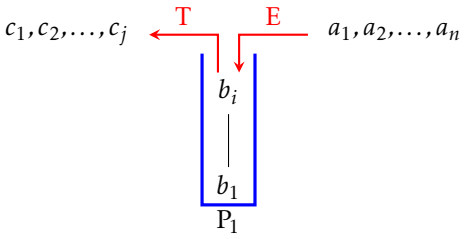


sortie des n éléments du sas

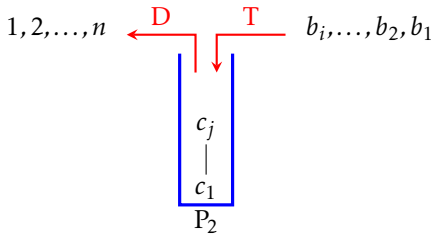
Considérons une séquence pouvant être triée par sas et la situation des deux piles à l'instant où le dernier élément vient d'être introduit dans le sas. Notons $b_1, \ldots, b_i$ les éléments présents dans p_1 et $c_1, \ldots, c_j$ ceux présents dans p_2 :



On peut observer que l'entrée dans le sas revient à trier avec la pile P_1 en considérant la pile P_2 comme la sortie, et la sortie du sas à trier avec la pile P_2 en considérant la pile P_1 comme l'entrée :



entrée dans le sas



sortie du sas

Autrement dit, la connaissance de c rend le processus déterministe.

Question 7. Rédiger une fonction `valide(a, c)` qui prend en argument les deux listes a et c et qui retourne un booléen traduisant si a peut être triée par sas en passant par la configuration intermédiaire (b, c).

Question 8. On suppose donné une fonction `combinations(l, r)` qui prend en arguments une suite finie l et un entier r et qui engendre la liste des suites extraites de l ayant r éléments [1].

a) Rédiger une fonction `triSas(a)` qui prend en argument une séquence a et qui retourne un booléen exprimant si a est triable par sas.

b) Exprimer le coût de `triSas` en fonction du nombre n d'éléments de la séquence a.

> *Dans un article récent (2013), il a été montré qu'il existe un algorithme de coût quadratique qui détermine si une séquence est effectivement triable par sas, et que le problème du tri par deux files en série pouvait se réduire (là encore en coût quadratique) au problème du tri par sas. Ces deux résultats prouvent qu'il existe un algorithme de coût polynomial pour déterminer si une séquence est triable par deux files en série. En revanche, il n'existe pas à l'heure actuelle de caractérisation simple des séquences effectivement triables (permettant par exemple de les dénombrer).*

1. Cette fonction existe dans le module `itertools`.