

# Problème générique de Dijkstra [tr06] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  UNIVERSITÉ HAUTE-ALSACE Version 21 mai 2018

## Table des matières

<b>1</b>	<b>Problème de Dijkstra : Cas <math>k = 3</math></b>	<b>2</b>
<b>2</b>	<b>Téléchargements – Utilitaires</b>	<b>3</b>
<b>3</b>	<b>Problème générique de Dijkstra / pgdijkstra</b>	<b>5</b>
3.1	Représentation des données . . . . .	5
3.2	Cas $k=2$ . . . . .	5
3.3	Cas $k=3$ : Problème de Dijkstra . . . . .	6
3.4	Cas $k=4$ et Cas générique . . . . .	7

## Java - Problème générique de Dijkstra (TP)



**Mots-Clés** Algorithmes de tris et rangs ■

**Requis** Algorithmes de tris et rangs, Complexité des algorithmes ■

**Difficulté** ● ● ○ (2 h) ■



### Objectif

Cet exercice décrit et analyse le problème générique du drapeau hollandais de E. DIJKSTRA.

# 1 Problème de Dijkstra : Cas $k = 3$

## Le problème de Dijkstra

Vous disposez de cailloux rouges, blancs, bleus alignés dans un ordre quelconque. Par **échanges successifs** de cailloux (et non par constitution de paquets séparés en un autre endroit) et en ne testant qu'**une fois** la couleur de chaque caillou, vous devez mener à une situation finale où tous les cailloux bleus se trouvent avant tous les blancs, eux-mêmes avant tous les rouges. Le nombre des cailloux de chaque couleur est quelconque.

## Objectif du problème

Le but à atteindre est la répartition des cailloux en trois ( $k = 3$ ) classes, les seules opérations autorisées étant :

- Faire des échanges successifs.
- Tester la couleur une seule fois par caillou.
- Utiliser l'emplacement initial.

## Première formalisation du problème

Elle consiste à considérer l'espace occupé par les cailloux comme une suite  $T$  de cases numérotées de 1 à  $n$ , une case contenant un seul caillou et étant identifiée par  $T[j]$  avec  $1 \leq j \leq n$ .

## Problème générique de Dijkstra

Soit  $T$  un tableau contenant des cailloux colorés, la couleur étant identifiée à un entier de  $[1..k]$  ( $k$  couleurs). C'est toujours possible : il suffit d'une fonction de couleur  $c(v)$  qui associe à chaque élément de valeur  $v$  de  $T$ , un entier de  $[1..k]$ .

## Objectif du problème générique

Le but à atteindre est la répartition des cailloux en  $k$  classes, les deux seules opérations autorisées étant :

- La permutation de deux éléments du tableau.
- Le test de la couleur d'un élément.

...(suite page suivante)...

## 2 Téléchargements – Utilitaires

Cet exercice utilise les opérations suivantes, toutes définies dans un bon nombre d'exercices de cet espace thématique :

- Fonction `saisirNombreElements`
- Procédure `afficherTri`
- Procédure `aleatoireTri`
- Procédure `permuterTab`

Elles ont été regroupées dans une bibliothèque.



### Définitions Java

```
final int TMAX = ...;
```



Fixez la constante `TMAX=50` (nombre maximum d'éléments).



**Téléchargez** le fichier suivant et mettez-le dans votre dossier.

**Java** @[UtilsTR.java]



**Copiez/collez** ensuite les lignes suivantes : Les opérations seront accessibles en notation usuelle.



**Soit** la fonction `saisirNombreElements(nmax)` qui renvoie le nombre d'éléments, saisi par l'utilisateur, entier compris dans `[1..nmax]`. Elle affiche l'invite :

```
Nombre d'éléments dans [1..[nmax]]?
```

**Java** @[saisirNombreElements] (dans UtilsTR.java)



**Soit** la procédure `afficherTri(t,n,g,h)` qui affiche, à la queue-leu-leu séparés par un espace le tout entre crochet, les `n` premières valeurs d'un `ITableau t`, les indices `g` et `h` indiquant le sous-intervalle du tri et représentés par une barre verticale. La barre de gauche est avant `g` et celle de droite est après `h`. Exemple :

```
afficherTri(t,10,4,9) ==> [1 2 3 |4 5 6 7 8 9 |10]
```

**Java** @[afficherTri] (dans UtilsTR.java)



**Soit** la procédure `aleatoireTri(t,n,vmax)H` qui initialise les `n` premiers éléments d'un `ITableau t` en utilisant `vmax` comme valeur maximale pour la fonction de génération d'un entier pseudo-aléatoire.

**Java** @[aleatoireTri] (dans UtilsTR.java)



**Soit** la procédure `permuterTab(t,j,k)` qui permute les éléments d'indice `j` et `k` d'un `ITableau t`. Les indices sont supposés valides.

**Java** `@[permuterTab]` (dans `UtilsTR.java`)



### Remarque

Si la fonction et les procédures n'ont pas été réalisées, il vous est conseillé de la(les) rédiger dans l'exercice `@[Utilitaires Tris et Rangs]`.

...(suite page suivante)...

### 3 Problème générique de Dijkstra / pgdijkstra

Nous allons étudier les cas particuliers ( $k = 2$ ,  $k = 3$  et  $k = 4$ ) puis le cas générique.

#### 3.1 Représentation des données



Définissez les couleurs `CBLEU=1`, `CBLANC=2`, `CROUGE=3` et `CVERT=4`.



Définissez une fonction `couleur(t,k)` qui, pour un indice  $k$  d'un Tableau  $t$ , renvoie la couleur

- `CBLEU` si  $t[k]=0$
- `CBLANC` si  $t[k]=1$
- `CROUGE` si  $t[k]=2$
- `CVERT` sinon (cas  $t[k]>2$ )

#### 3.2 Cas $k=2$

Il y a des cailloux de couleur `CBLEU` et `CBLANC` alignés dans un ordre quelconque. Le but est de placer les premiers à gauche et les seconds à droite.

La bonne méthode est d'élaborer un invariant qui va être maintenu tout au long de l'exécution. Plaçons-nous à une étape intermédiaire où la situation est la suivante :

1	$g$	$j$	$h$	$n$
...CBLEU ...	inconnu		...CBLANC ...	

Au départ :  $j = 1$  et  $h = n$ . Lorsque  $h < j$ , la situation désirée est atteinte. Comment progresser vers ce but ?

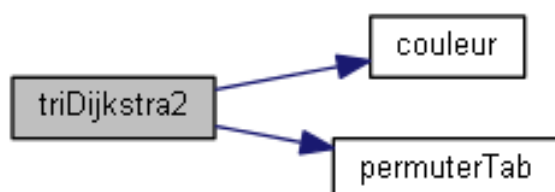
En regardant  $T[j]$  :

- S'il est de couleur `CBLEU` : incrémentez  $j$ . La quantité  $(h - j)$  diminue.
- Sinon (il est de couleur `CBLANC`) : permuter les éléments  $T[j]$  et  $T[h]$  puis décrémente  $h$ . La quantité  $(h - j)$  diminue.

La quantité  $(h - j)$  est le *variant* de la boucle. Au départ elle vaut  $n$  et à chaque itération elle décroît. Au bout d'au plus  $n$  itérations elle sera négative ou nulle.



Écrivez une procédure `triDijkstra2(t,n)` qui effectue le réarrangement décrit ci-dessus pour  $n$  cailloux d'un Tableau  $t$ .





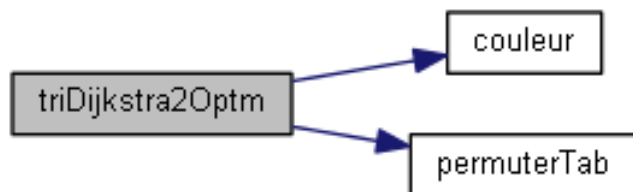
Calculez la complexité au pire de votre solution.



Pouvez-vous l'améliorer ?



Écrivez une procédure `triDijkstra2Optm(t,n)` qui effectue le réarrangement en tenant compte de la symétrie pour  $n$  cailloux d'un Tableau  $t$ .



Écrivez un programme qui saisit le nombre de cailloux, déclare un Tableau et l'initialise de façon aléatoire en prenant 2 pour valeur maximale puis en effectue la partition en deux couleurs et l'affiche.



Testez.

### 3.3 Cas $k=3$ : Problème de Dijkstra

Il y a des cailloux de couleur **CBLEU**, **CBLANC** et **CROUGE** alignés dans un ordre quelconque. Le but est de placer les premiers à gauche, les seconds au milieu et les troisièmes à droite. Reprenons la démarche par invariant et variant vue pour  $k = 2$ . La situation générique est donc la suivante :

1	$g$	$j$	$h$	$n$
...BLEU ...	...BLANC ...	inconnu	...ROUGE ...	

Au départ :  $g = 0$ ,  $j = 1$  et  $h = n + 1$ .

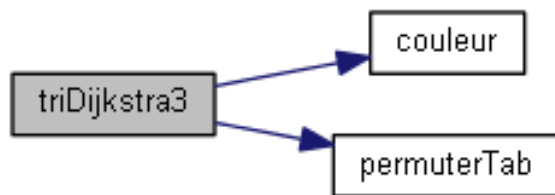
On cherche à réduire la zone inconnue de taille  $h - j$ .

Regardons  $T[j]$  :

- S'il est de couleur **CBLANC** : incrémentez  $j$  et le variant  $(h - j)$  diminue.
- S'il est de couleur **CBLEU** : incrémentez  $g$ , permuter les éléments en  $j$  et  $g$  et incrémentez  $j$ . Le variant décroît aussi.
- Sinon (il est de couleur **CROUGE**) : décrémente  $h$  puis permuter les éléments en  $j$  et  $h$ . Le variant décroît aussi.



Écrivez une procédure `triDijkstra3(t,n)` qui effectue le réarrangement décrit ci-dessus pour  $n$  cailloux d'un Tableau `t`.



Modifiez votre programme pour qu'il saisisse en plus le nombre de couleurs dans `ncouleurs` (entier), initialise aléatoirement les cailloux d'un Tableau dans  $[0..ncouleurs]$  puis effectue la partition en deux ou trois couleurs selon la valeur de `ncouleurs` et l'affiche.



Testez.



Quelle est la complexité de votre solution ?



Qu'en pensez-vous ?  
Pouvez-vous l'améliorer ?



Il est possible de trouver une solution indirecte.  
Expliquez comment.



Quel est le coût de votre solution indirecte ?

### 3.4 Cas $k=4$ et Cas générique

Il y a des boules de couleur `CBLEU`, `CBLANC`, `CROUGE` et `CVERT`. Le but est de placer les cailloux de couleur `CBLEU` à gauche, `CBLANC` puis `CROUGE` et `CVERT` à droite. La situation générique est la suivante :

1	$g$	$j$	$h$	$v$	$n$
...BLEU ...	...BLANC ...	inconnu	...ROUGE ...	...VERT ...	

Au départ :  $g = 0$ ,  $j = 1$  et  $v = h = n + 1$ . Nous cherchons à réduire la zone inconnue de taille  $h - j$ .

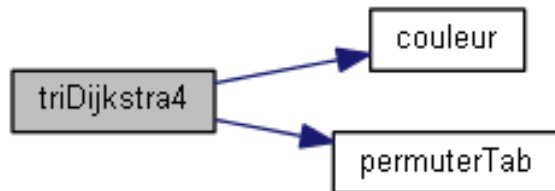
Regardons  $T[j]$  :

- S'il est de couleur `CBLANC` : incrémentez  $j$ .
- S'il est de couleur `CBLEU` : incrémentez  $g$ , permutez les éléments en  $j$  et  $g$  et incrémentez  $j$  (comme précédemment).
- S'il est de couleur `CROUGE` : décrémente  $h$  puis permutez les éléments en  $j$  et  $h$ .

- Sinon (il est de couleur **CVERT**) : décrémentez  $v$  puis permutez les éléments en  $j$  et  $v$  ; puis vérifiez si  $h \leq v$  auquel cas il faut également décrémenter  $h$  et permuter les éléments.



Écrivez une procédure **triDijkstra4(t,n)** qui effectue le réarrangement décrit ci-dessus pour  $n$  cailloux d'un **Tableau t**.



Complétez votre programme afin d'effectuer la partition en quatre couleurs si **ncouleurs** vaut 4.



Testez.



Quelle est la complexité de votre solution ?



Ici aussi, il est possible de trouver une solution indirecte.  
Expliquez comment.



Connaissant les solutions pour  $k = 1, \dots, m - 1$ ,  
Comment construire une solution pour  $k = m$  ?