Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Devoir de programmation 2 (15%) CSI2110/CSI2510

Politique de retard: ***1min-24hres de retard -30% pénalité; devoirs non acceptés après 24hres.***

Le problème du traversier

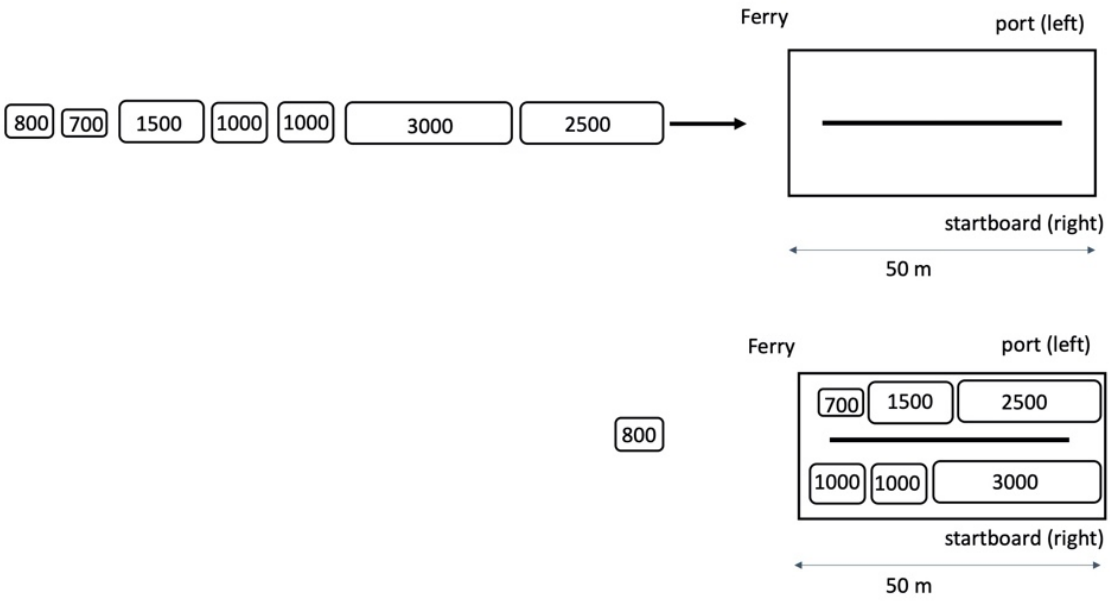
Introduction

Dans ce devoir, vous aurez à résoudre le problème du traversier, un problème proposé pour la préparation des compétitions de programmation de l'ACM. Nous vous demanderons toutefois de résoudre ce problème d'une façon spécifique et non d'utiliser les solutions disponibles communément proposées.

Vous aurez aussi à soumettre votre code en ligne afin de vérifier, de façon automatisée, la qualité de votre solution. Pour être acceptée, votre solution devra passer une série de tests et ce à l'intérieur d'un temps de calcul prédéterminé.

Description du problème

A la page suivante, vous trouverez un diagramme qui servira à illustrer le problème du traversier décrit dans ce document. Les petits rectangles représentent des véhicules à être embaquées dans un traversier (le nombre indique leur longueur en cm). Le grand rectangle représente le traversier avant et après l'embarquement.



Référence: Ferry Loading problem PC/UVa IDs: 111106/10261 https://onlinejudge.org/

Avant que les ponts soient omniprésents, les traversiers étaient communément utilisés afin de transporter des véhicules d'une rive à l'autre d'un cours d'eau. Ces traversiers avaient, le plus souvent, deux voies pour entasser les véhicules, l'une côté bâbord (port), c'est-à-dire à gauche en regardant vers l'avant et l'autre à tribord (starboard).

Les véhicules en attente de monter à bord forme une file et un opérateur les dirige vers le traversier en indiquant si ils devraient se diriger à bâbord ou à tribord de facon à faire embarquer un maximum de véhicules. Bien entendu, chaque véhicule a une longueur différente et il revient donc à l'opérateur d'inspecter la file afin de bien diriger les véhicules pour assurer un embarquement optimal considérant la longueur totale du traversier.

On vous demande donc d'écrire un programme qui pourra assister l'opérateur dans ses décisions.

Données d'entrée

Les données du problème seront fournies selon le format suivant. Un premier nombre indique le nombre de cas à tester (un cas étant ici une file de véhicule à embarquer). Ce nombre est suivi d'une ligne blanche et chacun des cas sera aussi terminé par une ligne blanche.

La première ligne de chaque cas à traiter est un entier entre 1 et 100 ; c'est la longueur du traversier en mètres. Chaque véhicule est ensuite identifié par un nombre représentant sa longueur en cm (un entier entre 100 et 3000). Cette séquence se termine par un 0. Les véhicules doivent être embarqués dans

l'ordre, de chaque côté du traversier sans dépasser la longueur totale du traversier. L'objectif est d'embarquer un maximum de véhicules.

Données de sortie

Pour chaque cas de test, la sortie produite doit suivre le format donné ci-bas. Les différentes sorties associées à chaque cas doivent être séparées par une ligne blanche.

La première ligne doit donner le nombre total de véhicules embarqués. Puis pour chacun des véhicules donnés en entrée, il faut indiquer, dans l'ordre d'arrivée, si celui-ci doit se diriger à bâbord ou à tribord. Bien entendu, il peut y avoir plus d'une solution équivalente pour chacun des cas.

Exemple

Entrée :

```
1

50
2500
3000
1000
1000
1500
700
800
0
```

Sortie :

```
6
port
starboard
starboard
starboard
port
port
```

Algorithme

Vous devez résoudre ce problème en utilisant le retour-arrière (*backtracking*) et la mémorisation tel que décrit dans l'article de Bento.[1] La mémorisation utilisera une table normale ou une table de hachage;

[1] L.M.S. Bento, V.G. Pereira de Sa, J.L. Szwarcfiter, Some illustrative examples on the use of Hash tables, Pesquisa Operacional (2015) vol35, 423-437 (Section 3.4). doi: 10.1590/0101-7438.2015.035.02.0423

vous devrez tester et comparer ces deux approches. Nous cherchons la solution optimale permettant d'embarquer un maximum de véhicules. Dans l'exemple précédent, cette solution peut s'exprimer ainsi: M=6 et x^*=(1,0,0,0,1,1) avec x^*_i= 1 représentant babord et x^*_i= 0 représentant tribord.

La méthode du retour-arrière est une stratégie d'essais et erreurs qui au lieu de tester toutes les solutions possible (il y en a $\sum_{k=0}^{n} 2^k = 2^{n+1}$), explore plutôt l'espace des solutions faisables se diriger vers des solutions équivalentes à des solutions déjà explorées.

A chaque étape, une solution partielle est considérée où un certain nombre k de véhicules est embarqué. Par exemple pour k=3, on pourrait considérer la solution partielle x=(0,1,0). L'information essentielle dans ce cas est la longueur cumulative des véhicules embarqués (ici les premiers k=3) ainsi que l'espace libre côté bâbord désigné ici par s; toute l'information additionnelle peut être déduite de ces deux nombres. Dans notre solution partielle, on pourrait avoir s=2000cm. Pour l'exemple décrit ci-haut, nous avons une longueur totale pour les 3 premiers véhicules de 2500+3000+1000=6500cm. Puisque le traversier a une longueur L=50m, l'espace occupé du côté babord est (L - s) =50m-2000cm=3000cm. Nous avons donc 6500-3000=3500cm occupé à tribord et donc 50m-3500cm=1500cm disponible de ce côté. Par conséquent, la solution partielle x=(0,1,0) se représente complètement avec l'état (k=3, s=2000), toute solution se représentant avec ce même état est complétement équivalente puisqu'elle représente un arrangement différent des même k véhicules occupant la même longueur de chaque côté du traversier.

Le retour-arrière explorera donc toutes les solutions récursivement sans jamais visiter deux fois des solutions équivalentes à celles déjà explorées. A partir d'une solution partielle comprenant k véhicules, il est possible de générer au plus deux solutions incluant k+1 véhicules; l'une dans laquelle le véhicule suivant est placé à bâbord et l'autre ou le véhicule est placé à tribord. Dans notre exemple, à partir de la solution x=(0,1,0), on peut construire la solution x_babord = (0,1,0,1) et x_tribord=(0,1,0,0) avec les états (k=4, s=1000) et (k=4, s=2000), respectivement. Dans certain cas, l'une des deux solutions peut être invalide lorsque la longueur totale est excédée. La descente dans l'arbre des solutions possible se termine lors qu'il n'y a plus de solutions valides ou lorsque le nombre total de véhicules n est atteint. On arrête aussi la descente si la solution à explorer est équivalente à une solution déjà explorée (un état déjà visité) ; pour ce faire il faut mémoriser les états déjà visités et c'est là un des éléments importants de cet algorithme. La Figure 1 donne le pseudo-code l'algorithme proposé.

La mémorisation des états visités (voir les énoncés identifies par //**// dans le pseudo-code de la Figure 1) se fera donc ici de deux facons:

1. **Big Table:** pour un problème comprenant n véhicules et un traversier de longueur L (en cm), un tableau de booléen appelé `visited` et de dimension (n+1)x(L+1) est utilisé avec `visited[k,s]` vrai si l'état `(k,s)` a déjà été visité.
2. **Hash Table:** en utilisant une table de hachage dont la dimension est beaucoup plus petite que celle de la **Big table** décrite précédemment afin de storer les états en utilisant `(k,s)` comme clés pour la fonction de hachage et ainsi rendre plus efficace la vérification des états visités.

Evidemment, la solution utilisant la **Big Table** sera la plus rapide. L'usage de la table de hachage permet ici de réduire la quantité de mémoire requise (bien moins que (n+1)x(L+1)).

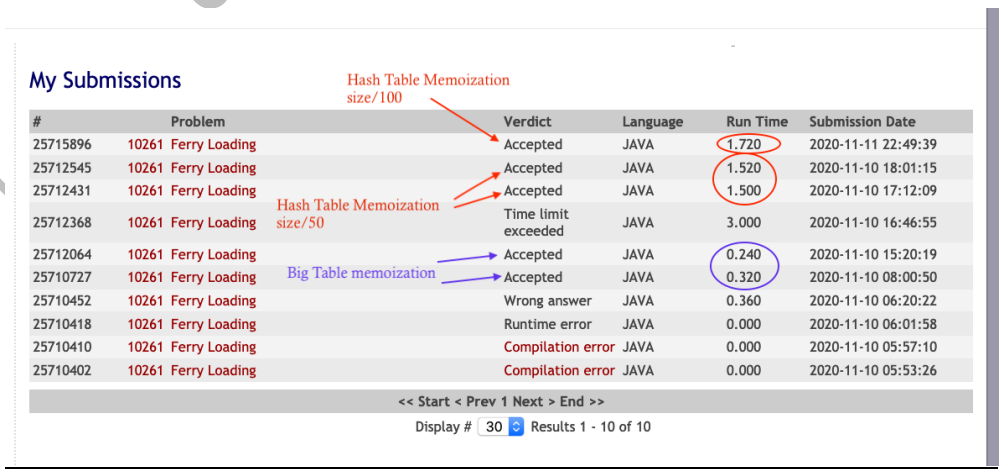
Choisir la bonne table de hachage et la bonne fonction de hachage sera donc une décision de design importante dans votre solution. Vous devrez donc identifier une solution qui sera un bon compromis entre rapidité d'exécution et utilisation de la mémoire.

Pour fin de comparaison, la Figure 2 montre notre solution courante qui permet une économie de mémoire par un facteur de 50 à 100 au prix d'un temps d'exécution plus lent qu'en utilisant la Big Table.

Figure 1: Pseudocode de la recherche de solution par retour-arrière

```
Global variables to the procedure Bakctrack:
integers bestK; arrays currX[0..n], bestX[0..n].
Main Method:
   Read data and initialize variables: n, length[0..n-1], L; bestK=-1
   BacktrackSolve(0,L);
   Write solution stored in bestX[0..bestK-1]
Recursive Method:
BacktrackSolve(int currK, currS)
  // currK cars have been added; currS space remains at the left side
    if currK>bestK then update bestK,bestX with currK, currX
    if currK < n then // there are cars left to consider
       if "possible to add next car to left" and "(currK+1,currS-length[currK]) was
           not visited" //**// then
                  currX[currk]=1; newS=currS-length[currK];
                  BacktrackSolve(currK+1,newS)
                  Mark (currK+1,newS) as visited. //**//
       if "possible to add next car to right" and "(currK+1,currS) was
           not visited" //**// then
                  currX[currk]=0;
                  BacktrackSolve(currK+1,currS)
                  Mark state(currK+1,currS)as visited //**//
```

Figure 2: Quelques solutions utilisant la BigTable et différentes tables de hachage



#	Problem	Verdict	Language	Run Time	Submission Date
25715896	10261 Ferry Loading	Accepted	JAVA	1.720	2020-11-11 22:49:39
25712545	10261 Ferry Loading	Accepted	JAVA	1.520	2020-11-10 18:01:15
25712431	10261 Ferry Loading	Accepted	JAVA	1.500	2020-11-10 17:12:09
25712368	10261 Ferry Loading	Time limit exceeded	JAVA	3.000	2020-11-10 16:46:55
25712064	10261 Ferry Loading	Accepted	JAVA	0.240	2020-11-10 15:20:19
25710727	10261 Ferry Loading	Accepted	JAVA	0.320	2020-11-10 08:00:50
25710452	10261 Ferry Loading	Wrong answer	JAVA	0.360	2020-11-10 06:20:22
25710418	10261 Ferry Loading	Runtime error	JAVA	0.000	2020-11-10 06:01:58
25710410	10261 Ferry Loading	Compilation error	JAVA	0.000	2020-11-10 05:57:10
25710402	10261 Ferry Loading	Compilation error	JAVA	0.000	2020-11-10 05:53:26

<< Start < Prev 1 Next > End >>

Display # 30 Results 1 - 10 of 10

Vos tâches:

Partie 1: Mémorisation des états en utilisant la Big Table

- Réaliser votre solution en utilisant la solution avec Big Table.
- Tester votre solution avec les fichiers donnés avec votre devoir.
- Tester votre solution en utilisant le site en ligne spécifié à la fin de ce document. Faire une capture d'écran afin de démontrer vos résultats.

Partie 2: Mémorisation des états en utilisant une table de hachage

- Concevoir la solution en utilisant une table de hachage.
- Soyez aussi optimal que possible dans le design de votre solution, vous avez une limite de 3 secondes pour trouver une solution. Vous pouvez utiliser votre propre design pour la table de hachage, une implémentation extraite des notes de cours et de labo ou utiliser les HashMap de la librairie de Java https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

Partie 3: Rapport

Écrire un court rapport comprenant:

A. Partie 1: confirmation si la solution produite a passé tous les tests avec capture d'écran pour le prouver; donner aussi les temps d'exécution.
B. Partie 2: confirmation si la solution produite a passé tous les tests avec capture d'écran pour le prouver; donner aussi les temps d'exécution et le facteur de réduction en mémoire.
C. Design de la partie 2: décrire votre table de hachage, sa dimension, la fonction de hachage et décrire votre implémentation. Décrire aussi les différents tentatives et comment vous êtes arrives à la solution finale.

Requirements

- Les specifications sur l'évaluation en ligne sont données ici:
  - https://onlinejudge.org/index.php?option=com_content&task=view&id=14&Itemid=29
  - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1202
- L'entrée sera donnée au flux standard (System.in). La solution trouvée devra être donnée au flux de sortie standard (System.out). Vous pouvez utiliser le flux d'erreurs (System.err) pour l'examen du déroulement du programme, mais attention ces affichages vont ralentir votre programme.
- Votre projet doit être soumis dans un fichier zip appelé p2<numéro d'étudiant >. Dans ce fichier compressé vous devez avoir:

- Un sous-répertoitre appelé BigTable contenant Main.java pour votre solution avec Big Table et les résultats de vos tests avec les fichiers fournis, soient output1.txt, output2.txt, etc correspondant à input1.txt, input2.txt, etc.
- Un sous-répertoire HashTable contenant Main.java pour votre solution avec table de hachage et les résultats de vos tests avec les fichiers fournis, soient output1.txt, output2.txt, etc correspondant à input1.txt, input2.txt, etc.
- Un fichier appelé report.pdf contenant votre rapport.

Barême de correction (60 points, 100%)

- Partie 1: 20%
- Partie 2: 20%
- Partie 3 A et B: 5%
- Partie 3 C: 15%