Algorithmes et structures de données avancées : TD 4(sur machine)

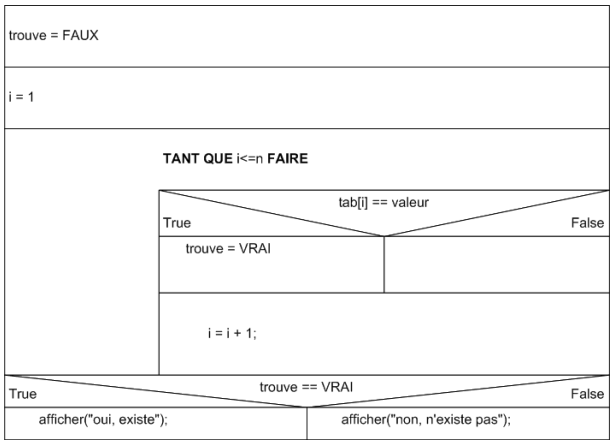
Sensibilisation à la complexité, Paradoxe Monthy Hall

Exercice 4.1 *Recherche linéaire dans un tableau à une dimension*

Vous disposez d'un tableau numériques de `n` entrées stocké dans la variable `tab`.
Dans cet exercice, nous travaillerons dans le fichier

`Z:\ APP\www\asda2008\td4\lineaire.php`

1. Affecter chaque entrée de ce tableau numériques avec des valeurs entières aléatoires entre 1 et 1000000.
2. Ecrire le programme pour le structogramme d'un algorithme suivant qui décide s'il y a une valeur stocké dans la variable `valeur` dans le tableau :



3. Pour chronométrer, vous pouvez utiliser la fonction suivante :

```
function temps()
{
  list($usec, $sec) = explode(" ", microtime());
  return ((float)$usec + (float)$sec);
}
```

Pour calculer le temps d'exécution de l'algorithme, vous devez appeler la fonction `debut = temps();` avant la recherche linéaire, et appeler la fonction `fin = temps();` après la recherche linéaire.

Pour afficher le temps, il vous suffit d'afficher d'afficher la différence :

```
echo "Temps : ".($fin - $debut)." sec.<br />";
```

4. Chronométrer le temps d'exécution pour plusieurs valeurs de `n`, par exemple `n=1000`, `n=10000`, `n=100000`, `n=1000000`, `n=2000000`, `n=4000000`, et tracer (sur papier) le graphe (abscisse la longueur d'entrée n, ordonné le temps d'exécution).

Exercice 4.2 *Recherche dichotomique dans un tableau à une dimension*

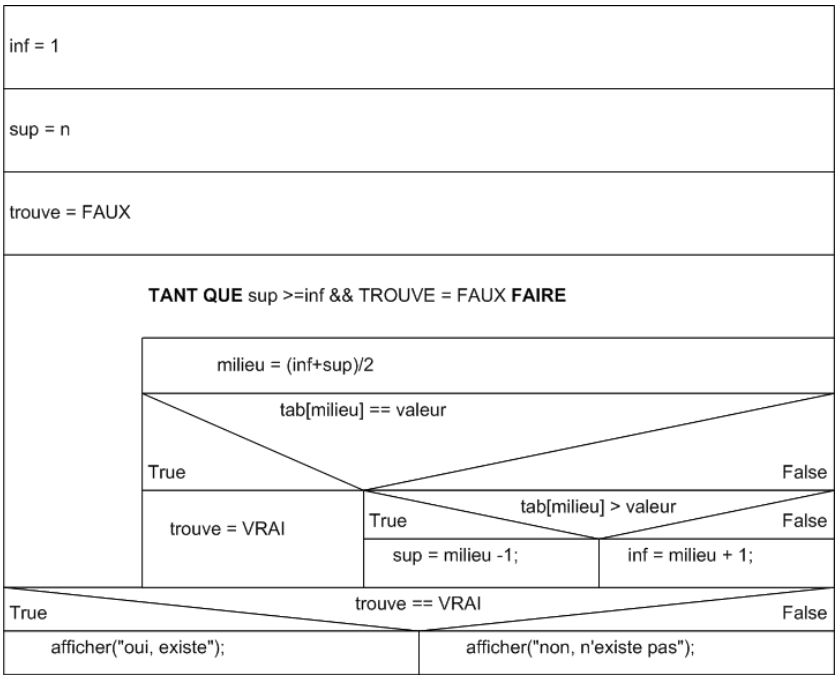
Dans cet exercice, nous travaillerons dans le fichier

`Z:\ APP\www\asda2008\td4\dichotomique.php`

1. Maintenant trier votre tableau. Pour cela, appeler la fonction suivante :

```
sort($tab);
```

2. Chronométrer le temps d'exécution du tri pour plusieurs valeurs de `n`, par exemple `n=1000`, `n=10000`, `n=100000`, `n=1000000`, `n=2000000`, `n=4000000`, et tracer (sur papier) le graphe (abscisse la longueur d'entrée n, ordonné le temps d'exécution).
3. Ecrire le programme pour le structogramme pour un algorithme qui décide s'il y a une valeur stocké dans la variable `valeur` dans ce tableau trié, et qui affiche la réponse.



4. Chronométrer le temps d'exécution pour plusieurs valeurs de `n`, par exemple `n=1000`, `n=10000`, `n=100000`, `n=1000000`, `n=2000000`, `n=4000000`, et tracer (sur papier) le graphe (abscisse la longueur d'entrée n, ordonné le temps d'exécution).

Exercice 4.3 *Divertissement*

Le problème de Monty Hall est un casse-tête probabiliste librement inspiré du jeu télévisé américain Let's Make a Deal [1]. Il est simple dans son énoncé mais non intuitif dans sa résolution et c'est pourquoi on parle parfois à son sujet de paradoxe de Monty Hall. Le nom de ce problème mathématique vient du nom de celui qui a presenté ce jeu aux États-Unis pendant treize ans, l'animateur d'origine canadienne Monty Hall.

Le jeu oppose un présentateur à un candidat (le joueur). Ce joueur est placé devant trois portes fermées, les portes numéros 1, 2, et 3. Derrière l'une d'elles se trouve une voiture (ou tout autre prix magnifique) et derrière chacune des deux autres se trouve un phantôme (ou tout autre prix sans importance). Il doit tout d'abord désigner une porte. Puis le présentateur ouvre une porte qui n'est ni celle choisie par le candidat, ni celle cachant la voiture (le présentateur sait quelle est la bonne porte dès le début). Le candidat a alors le droit ou bien d'ouvrir la porte qu'il a choisie initialement, ou bien d'ouvrir la troisième porte.

Dans un premier temps, nous allons simuler le calcul pour que le joueur garde son choix initial.

Dans cet exercice, nous travaillerons dans le fichier

`Z:\ APP\www\asda2008\td4\garder.php`

1. Ecrire un algorithme qui affecte les variables `solution` et `premierChoix` avec chacune une variable entière aléatoire compris entre 1 et 3.
2. Tester si les deux variables `solution` et `premierChoix` sont égales.
3. Encapsuler votre algorithme dans une boucle avec `n` itérations, et compter le nombre de fois que le joueur gagne avec ce premier choix. Calculer la probabilité avec la méthode de Monte Carlo (c-à-d. diviser le nombre de fois que le joueur gagne par le nombre d'itérations `n`.

Maintenant, nous allons simuler le calcul pour que le joueur change son choix initial (donc en tenant compte de l'information supplémentaire du présentateur).

Dans cet exercice, nous travaillerons dans le fichier

`Z:\ APP\www\asda2008\td4\changer.php`

4. Ecrire un algorithme qui affecte les variables `solution` et `premierChoix` avec chacune une variable entière aléatoire compris entre 1 et 3.
5. Afficher le contenu de la variable `premierChoix` et à `solution` l'écran.
6. Simuler que le présentateur (représenté par l'ordinateur) ouvre une deuxième porte **qui n'est pas celle du choix du joueur, ni celle où se trouve le prix**. Stocker le numéro de la porte dans la variable `porteOuverte` et afficher la porte sélectionné.
7. Maintenant, déterminer le choix du joueur, **qui est différent de son premier choix, et différent de la porte (vide) séléctionné par le présentateur**, et stocker le dans la variable `deuxiemeChoix`, et afficher la porte sélectionné.
8. Tester si les deux variables `deuxiemeChoix` et `solution` sont égales.

9. Encapsuler votre algorithme dans une boucle avec `n` itérations, et compter le nombre de fois que le joueur gagne avec son premier choix. Calculer la probabilité avec la méthode de Monte Carlo.

Votre affichage devrait ressembler au suivant :

```
Jeu no. 1
--------------

Solution : 1
Premier choix du joueur : 1

Porte ouverte par le présentateur : 2
Deuxieme choix du joueur: 3

Perdu


Jeu no. 2
--------------

Solution : 3
Premier choix du joueur : 1

Porte ouverte par le présentateur : 2
Deuxieme choix du joueur: 3

Gagné!!
.....
```

Comparaison

10. Comparer les probabilités de la 1ère et 2ème stratégie.