Cours 1 : Méthodes d'analyse des algorithmes - Illustration sur les algorithmes de tri

Jean-Stéphane Varré

Université Lille 1

jean-stephane.varre@univ-lille1.fr

Au menu

- exemples introductifs
- notions de pire des cas et meilleur des cas
- les tris bulle, sélection, insertion
- comportement asymptotique
- l'insertion récursive et dichotomique
- résolution des équations de partition
- les tris fusion, rapide

? Question

Comment évaluer à l'avance le comportement d'un algorithme ?

- combien de temps va-t-il mettre à s'exécuter ?
- combien d'espace mémoire va-t-il utiliser en plus des données initiales ?
- a-t-il toujours le même comportement ou bien existe-t-il des données plus ou moins favorables ?

Préliminaires

- un algorithme est une suite finie d'instructions qui résoud un problème
- un algorithme doit fonctionner sur tous les exemplaires du problème qu'il résoud
- la taille de l'exemplaire est formellement le nombre de bits mais on pourra réduire cette définition au nombre de composantes (cases d'un tableau)
- l'ensemble des exemplaires est le domaine de définition

Question

un algorithme A qui nécessite $10^{-4} \times 2^n$ instructions est-il moins performant qu'un algorithme B qui requiert $10^{-2} \times n^5$ instructions pour traiter une donnée de taille n ?

De combien de manières différentes peut-on rechercher la présence d'un élément dans un tableau ?

```
"""
  True if value v exists in t
"""
def contains (t,v)
```


Présence d'un élément dans un tableau (v1)

```
def contains(t,v):
  found = False
  for i in range (0,len(t)):
    if t[i] == v:
      found = True
  return found
```

- temps d'exécution ?
- espace mémoire occupé ?

Présence d'un élément dans un tableau (v2)

```
def contains(t,v):
  found = False
  i = 0
  while i < len(t) and not found:
    if t[i] == v:
      found = True
    i = i + 1
  return found
```

- le nombre d'opérations depend-il
  - de la taille de l'exemplaire ?
  - de la « forme » de l'exemplaire ?

Présence d'un élément dans un tableau (v3)

```
1  # version recursive
2  def contains (t,v):
3    n = len(t)
4    if n == 0:
5      return False
6    else:
7      tt = t[1:n]
8      return t[0] == v or contains(tt,v)
```

- comment évaluer les algorithmes récursifs ?
- calcul du nombre de comparaisons
- discussion de l'espace supplémentaire utilisé

Présence d'un élément dans un tableau (v4)

dans le cas où `t[i:j]` implique une recopie, on peut faire mieux en espace que la v3, on utilise une sous-fonction qui va gérer les bornes du tableau

```
1  # version recursive avec espace memoire constant
2  # p = position dans le tableau
3  def aux (t,p,v) =
4    if p == len(t) then
5      return False
6    else
7      return t[p] == v or aux(t,p+1,v)
8
9  def contains t v =
10   return aux(t,0,v)
```


Résumé (1/3)

pour évaluer la complexité des algorithmes :

- on comptera les affectations de valeurs du même type que l'exemplaire
- les comparaisons de valeurs du même type que l'exemplaire

on fait correspondre ce nombre au temps mis par l'algorithme pour s'exécuter : si on associe un temps unitaire à l'opération comptée, on obtient une évaluation du temps d'exécution

- les espaces mémoires utilisés en plus de ceux de l'exemplaire et du même type que l'exemplaire

l'opération mesurée ou l'unité de mémoire utilisée est celle qui est la plus importante vis-à-vis de nos données

Résumé (2/3)

on a mesuré pour :

- les boucles `pour` : il suffit de compter l'intervalle sur lequel elle est réalisée multiplié par le nombre d'opérations réalisées dans la boucle, cela va dépendre de la taille de la donnée
- les boucles `tant que` : il faut savoir compter le nombre exact de fois où la boucle est réalisée, cela va dépendre de la taille de la donnée et de la donnée elle-même
- les récursions : il faut établir une équation qui va sommer le nombre d'opérations réalisées lors d'un appel et le nombre de fois où la procédure est appelée récursivement, il faut aussi distinguer le cas de base

Résumé (3/3)

on a trouvé que :

- tous les algorithmes ne font pas le même nombre d'opérations (v2 et v3 réalisent le moins de comparaisons)
- tous les algorithmes ne consomment pas la même quantité de mémoire (v3 consomme plus de mémoire que v1 et v2)
- ces comptages dépendent de la taille de l'exemplaire (ici la longueur du tableau), et parfois de la nature de l'exemplaire (ici le contenu du tableau)

Différents cas

- pire des cas : correspond à un exemplaire ou un ensemble d'exemplaires pour lesquels l'algorithme s'exécute en temps maximal (ou avec un espace maximal)
  dans l'exemple, quand v n'est pas présent
- meilleur des cas : correspond à un exemplaire ou un ensemble d'exemplaires pour lesquels l'algorithme s'exécute en temps minimal (ou avec un espace minimal)
  dans l'exemple, quand v est en première position
- moyenne : correspond à la complexité moyenne sur tous les exemplaires possibles
  dans l'exemple . . . il faudrait savoir dénombrer tous les cas et réaliser la moyenne !

Il se peut que le meilleur des cas et le pire des cas soient confondus.