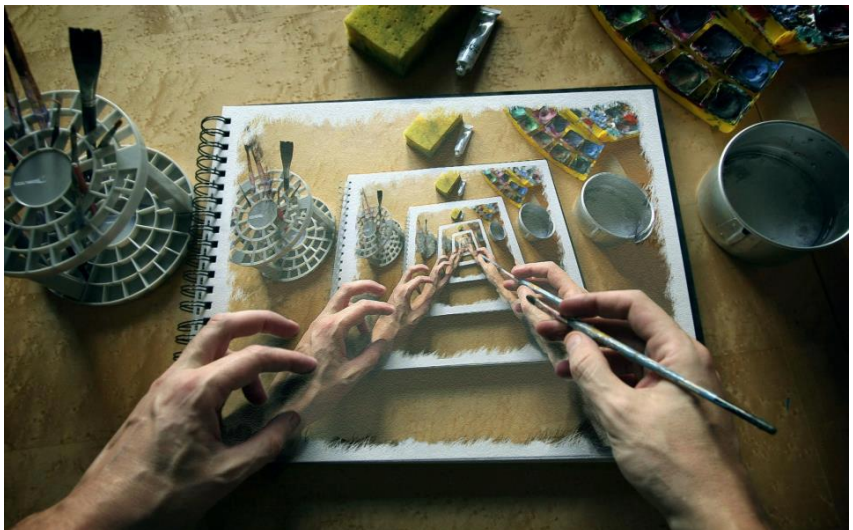
Algorithmique avancée : introduction à la récursivité (partie 1)

Marin Bougeret
IUT Montpellier

September 10, 2019



Plan général

Partie I : algorithmes récursifs

- conception d'algorithmes récursifs (I) : exemples introductifs
- conception d'algorithmes récursifs (II) : tests, idée de preuve, exemples
- conception d'algorithmes récursifs (III) : diviser pour régner

Partie II : structures récursives (listes, arbres..)

- ..

Partie III : complexité

- ..

Outline

Votre évolution (en plus joli)

Rappel

Méthode pour concevoir un algorithme récursif A

I) écrire A en supposant que A fonctionne déjà pour des entrées plus petites :

- penser à une grande entrée x
- supposer que pour tout $x' \in E$ plus petite que x, $A(x')$ donnera la bonne réponse
- en s'autorisant à faire de tel(s) appel(s) récursif(s), déduire la bonne réponse pour x

II) ajouter des cas de base:

- ajouter des cas de base pour traiter les entrées de E provoquant une erreur dans le code *I*)

$x \in E$ provoque une erreur ds *I*) si au moins une des cond. est v.

1. il y a une instruction incorrecte (division par 0, sortie tab...)
2. il y a un appel recursif $A(x')$ incorrect (x' ne vérifie pas les prérequis ($x' \notin E$), ou x' n'est pas plus petite que x)
3. tous les appels récursifs sont corrects, mais le calcul pour en déduire le résultat pour x est faux

Plan général

Après avoir suivi la (merveilleuse) méthode de conception, on obtient son algorithme.

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
    int temp1 = sautPuce(n-2);
    int temp2 = sautPuce(n-1);
    return temp1+temp2;
 }
}
```

- on sait que sautPuce(5) doit retourner 8.
- comment tester (à la main) un tel code ?

Comment tester un code récursif ?

Solution 1 (bof)

On se lance dans le calcul de f(5) .. essayons au tableau

Comment tester un code récursif ?

Solution 1 (bof)

On se lance dans le calcul de f(5) .. essayons au tableau

Moment de souffrance

Comment tester un code récursif ?

Solution 1 (bof)

On se lance dans le calcul de f(5) .. essayons au tableau

Moment de souffrance

Bilan

- au pire, on se perd complètement
- au mieux, on trouve le bon résultat, mais dérouler le calcul ne donne aucune intuition sur "pourquoi ça marche"

Comment tester un code récursif ?

Solution 2 (bien)

- on teste $f(0)$: facile!
  - $\Rightarrow$ maintenant, on sait que $f(0)$ est correct, et on ne le redéroulera plus jamais
  - $\Rightarrow$ on sait donc pour toujours que $f(0)$ est correct

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
   int temp1 = sautPuce(n-2);
   int temp2 = sautPuce(n-1);
   return temp1+temp2;
 }
}
```

Comment tester un code récursif ?

Solution 2 (bien)

- on teste $f(1)$ en supposant $f(0)$ vrai (inutile ici)
  - $\Rightarrow$ maintenant, on sait que $f(1)$ est correct, et on ne le redéroulera plus jamais
  - $\Rightarrow$ on sait donc pour toujours que $f(0)$ et $f(1)$ sont corrects

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
   int temp1 = sautPuce(n-2);
   int temp2 = sautPuce(n-1);
   return temp1+temp2;
 }
}
```

Comment tester un code récursif ?

Solution 2 (bien)

- on teste $f(2)$ en supposant $f(x)$ vrai pour $x \leq 1$
  - cette fois ci tester $f(2)$ devient facile!

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
    int temp1 = sautPuce(n-2);
    int temp2 = sautPuce(n-1);
    return temp1+temp2;
 }
}
```

Comment tester un code récursif ?

Solution 2 (bien)

- on teste $f(3)$ en supposant $f(x)$ vrai pour $x \leq 2$
  - cette fois ci tester $f(3)$ devient facile!

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
   int temp1 = sautPuce(n-2);
   int temp2 = sautPuce(n-1);
   return temp1+temp2;
 }
}
```

Comment tester un code récursif ?

Solution 2 (bien)

- jusqu'à arriver à $f(5)$

```
int sautPuce(int n){
 //n >= 0
 //calcule le nb de traj. en partant de n
 if(n==0) return 1;
 if(n==1) return 1;
 else{
   int temp1 = sautPuce(n-2);
   int temp2 = sautPuce(n-1);
   return temp1+temp2;
 }
}
```

Comment tester un code récursif ?

Bilan

- c'est plus facile ..
- et surtout, cela donne une idée de la preuve que l'algorithme est correct pour la raison suivante :

Vers une preuve par récurrence

Pour simplifier reprenons l'algorithme suivant

```
int somme(int x){
  if(x==1) // cas de base
    return 1;
  else{
    int temp = somme(x-1);// temp = 1+2+..+x-1
    return temp+x;
  }
}
```

Comment prouver que somme est correct ?

Une idée de comment prouver son algorithme

Pour prouver la correction, on considère la situation suivante :

- bob, l'adversaire, n'est pas convaincu que somme est correct
- pour le lui prouver, on va jouer au jeu suivant :

1) on dit à bob "si tu penses que somme est incorrect, donne moi une valeur x_0 pour laquelle tu penses que somme(x_0) n'est pas correct"
2) bob choisit x_0 (par exemple $x_0 = 543$)
3) on exécute comme précédemment les tests en partant de 0 jusqu'à arriver à 543

Comment argumenter pendant ces tests ?

Une idée de comment prouver son algorithme

Notre arme

Dans notre code on sait que pour tout $x \geq 1$, si somme(x-1) est correct alors somme(x) le sera aussi

- on vérifie somme(0), pas de problème, c'est correct
  - $\Rightarrow$ on se met donc d'accord avec bob que, somme(0) est correct, et qu'on ne le re-vérifiera plus
- maintenant, bob est d'accord pour supposer que somme(0) est correct

Une idée de comment prouver son algorithme

Notre arme

Dans notre code on sait que pour tout $x \geq 1$, si somme(x-1) est correct alors somme(x) le sera aussi

- on vérifie donc somme(1) ..... en supposant somme(0) correct :
  - on utilise notre arme avec $x = 1$

  $\Rightarrow$ on se met donc d'accord avec bob que, somme(1) est correct, et qu'on ne le re-vérifiera plus
- maintenant, bob est d'accord pour supposer que somme(x) est correct pour $x \leq 1$

Une idée de comment prouver son algorithme

Notre arme

Dans notre code on sait que pour tout $x \geq 1$, si somme(x-1) est correct alors somme(x) le sera aussi

- on vérifie donc somme(2) ... en supposant somme(x) correct pour $x \leq 1$:
  - on utilise notre arme avec $x = 2$

  $\Rightarrow$ on se met donc d'accord avec bob que, somme(2) est correct, et qu'on ne le re-vérifiera plus

Une idée de comment prouver son algorithme

Notre arme

Dans notre code on sait que pour tout $x \geq 1$, si somme(x-1) est correct alors somme(x) le sera aussi

- on vérifie donc somme(3) ... en supposant somme(x) correct pour $x \leq 2$:
  - on utilise notre arme avec $x = 3$

  $\Rightarrow$ on se met donc d'accord avec bob que, somme(2) est correct, et qu'on ne le re-vérifiera plus

Une idée de comment prouver son algorithme

Notre arme

Dans notre code on sait que pour tout $x \geq 1$, si somme(x-1) est correct alors somme(x) le sera aussi

- .. et ainsi de suite jusqu'à 543
- grâce à notre arme (qui marche pour tout x), ce procédé ne peut pas échouer
- on arrivera donc à convaincre bob pour toute valeur x_0 qu'il pourrait choisir
- cela "prouve" donc que somme est correct (il faudrait écrire proprement une récurrence, mais l'idée est là)

Bilan sur les tests

Conclusion

- A retenir : lorsque vous voulez tester à la main, commencez "depuis la plus petite valeur"
- (on s'intéressera peu aux preuves .. malheureusement :)

Outline

Phase 1

- JE recopie la méthode au tableau
- JE suis la méthode de conception devant vous
- VOUS me remerciez pour ce moment d'une grande pédagogie

```
boolean somme(int []t,int i){
 //prerequis 0 <= i <= t.length
 //action retourne somme des elements dans t[i
     ..(t.length-1)]


    return t[i]+somme(t,i+1);
}
```

```
boolean somme(int []t,int i){
 //prerequis 0 <= i <= t.length
 //action retourne somme des elements dans t[i
     ..(t.length-1)]
  if(i==t.length)
    return 0;
  else
    return t[i]+somme(t,i+1);
}
```

```
boolean somme(int []t,int i){
 //prerequis 0 <= i < t.length
 //action retourne somme des elements dans t[i
     ..(t.length-1)]


    return t[i]+somme(t,i+1);
}
```

```
boolean somme(int []t,int i){
 //prerequis 0 <= i < t.length
 //action retourne somme des elements dans t[i
     ..(t.length-1)]
   if(i==t.length-1)
     return t[i];
   else
     return t[i]+somme(t,i+1);
}
```

Phase 2

- JE vous donne un algo complet
- VOUS essayez de déterminer si il est correct en appliquant la méthode

sommeBiz2

```
double sommeBiz2 (int x){
//sommeBiz2(int x) = 1/2 + 1/3 + .. + 1/x si x
    >= 2, et 0 sinon
//prerequis x >= 0

 if (x == 0)
   return 0;
 int tmp = sommeBiz2 (x -1);
       // tmp = 1/2 + 1/3 + .. + 1/(x -1) si x
           >= 3, et 0 sinon
 return tmp +1/x ;
}
```

Correct ou ...


IN YOUR FACE
BOOYAH!

sommeBiz2

```
double sommeBiz2 (int x){
//sommeBiz2(int x) = 1/2 + 1/3 + .. + 1/x si x
    >= 2, et 0 sinon
//prerequis x >= 0

 if (x == 0)
   return 0;
 int tmp = sommeBiz2 (x -1);
        // tmp = 1/2 + 1/3 + .. + 1/(x -1) si x
           >= 3, et 0 sinon
 return tmp +1/x ;
}
```

$x \in E$ qui provoquent une erreur dans I):

- 0: point 2) (appel rec incorrect)
- 1: point 3) (appel rec correct mais calcul faux)

On doit avoir des cas de base pour $x = 0$ et $x = 1$

sommeBiz2

```
double sommeBiz2 (int x){
//sommeBiz2(int x) = 1/2 + 1/3 + .. + 1/x si x
    >= 2, et 0 sinon
//prerequis x >= 0

 if (x == 0)
   return 0;
 if (x == 1)
   return 0;
 int tmp = sommeBiz2 (x -1);
       // tmp = 1/2 + 1/3 + .. + 1/(x -1) si x
           >= 3, et 0 sinon
 return tmp +1/x ;
}
```

Cette version est correcte.

recherche

```
boolean recherche(int []t,int i, int x){
 //prerequis 0 <= i <= t.length
 //action determine si x dans t[i..(t.length-1)]

 if(t[i]==x)
   return true;
 else
  return recherche(t,i+1,x);
}
```

Correct ou ...

recherche

```
boolean recherche(int []t,int i, int x){
 //prerequis 0 <= i <= t.length
 //action determine si x dans t[i..(t.length-1)]

 if(t[i]==x)
   return true;
 else
  return recherche(t,i+1,x);
}
```

$x \in E$ qui provoquent une erreur dans I):
remarque : ici I) <=> tout le code

- $i = t.length$: point 1) (instruction incorrecte) et 2) (appel rec incorrect)

On doit avoir un cas de base pour $i == t.length$

recherche

```
boolean recherche(int []t,int i, int x){
 //prerequis 0 <= i <= t.length
 //action determine si x dans t[i..(t.length-1)]
 if(i==t.length) return false;
 else
   if(t[i]==x)
     return true;
   else
    return recherche(t,i+1,x);
}
```

- cette version est correcte
- c'est le premier exemple du cours où le bloc I] contient un cas de base

croissantAux

```
boolean croissantAux(int []t,int i){
 //prerequis 0 <= i <= t.length
 //action determine si t[i..(t.length-1)] est
     trie croissant
 if(i==t.length)
   return true;
 else
  return croissantAux(t,i+1) && (t[i] <= t[i+1])
}
```

Correct ou ...

croissantAux

```
boolean croissantAux(int []t,int i){
 //prerequis 0 <= i <= t.length
 //action determine si t[i..(t.length-1)] est
     trie croissant
 if(i==t.length)
   return true;
 else
  return croissantAux(t,i+1) && (t[i] <= t[i+1])
}
```

$x \in E$ qui provoquent une erreur dans I):

- $i = t.length$: point 1) (instruction incorrecte) et 2) (appel rec incorrect)
- $i = t.length - 1$: point 1) (instruction incorrecte)

On doit avoir un cas de base pour $i == t.length$ et $i == t.length - 1$

croissantAux

```
boolean croissantAux(int []t,int i){
 //prerequis 0 <= i <= t.length
 //action determine si t[i..(t.length-1)] est
     trie croissant
 if(i==t.length-1)
   return true;
 if(i==t.length)
   return true;
 else
  return croissantAux(t,i+1) && (t[i] <= t[i+1])
}
```

- cette version est correcte

recherche2

```
boolean recherche2(int []t,int i, int x){
 //prerequis 0 <= i  (on impose pas i < ..)
 //action determine si x dans t[i..(t.length-1)]

 if(i==t.length) return false;
 else
   if(t[i]==x)
     return true;
    else
     return recherche2(t,i+1,x);
}
```

Remarque

recherche2 est bien définie lorsque $i \geq t.length$: il faut répondre faux.

Correct ou ...

Faux !

recherche

```
boolean recherche2(int []t,int i, int x){
 //prerequis 0 <= i  (on impose pas i < ..)
 //action determine si x dans t[i..(t.length-1)]

 if(i==t.length) return false;
 else
   if(t[i]==x)
     return true;
    else
     return recherche2(t,i+1,x);
}
```

$x \in E$ qui provoquent une erreur dans I):

- $i = t.length$: point 1) (instruction incorrecte)
- .. et en fait tous les $i \geq t.length$ (instruction incorrecte)

On doit avoir un cas de base pour tous les $i >= t.length$

recherche2

```
boolean recherche2(int []t,int i, int x){
 //prerequis 0 <= i  (on impose pas i < ..)
 //action determine si x dans t[i..(t.length-1)]

 if(i>=t.length) return false;
 else
   if(t[i]==x)
     return true;
    else
     return recherche2(t,i+1,x);
}
```

Correct ?

Oui!