DUT MMI – IUT de Marne-la-Vallée
21/01/2019
M2202 - Algorithmique

Cours 1
Récursivité

UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

iut Marne-la-Vallée

Philippe Gambette

Organisation pratique

- **Contact**

- Courriel : philippe.gambette@u-pem.fr
  (M2202 doit apparaître dans le sujet du courriel).
- Avant ou après le cours.
- Possibilité de poser des questions, de demander des exercices supplémentaires d'entrainement.
- Interventions en TD & TP de Samy Dindane

- **Notes et devoirs**

Chez soi :

- exercices sur elearning (même espace que M1202) pour vous inciter à vous entraîner
- note de remplissage de cours à trous et/ou note globale de TP à la fin du semestre, prenant en compte votre avancée à chaque séance.

Pendant les cours :

- Devoir final le jeudi 6 juin.

Sources

- Cours de Jean-François Berdjugin à l'IUT de Grenoble
http://berdjugin.com/enseignements/inf/inf220/

- Cours de Xavier Heurtebise à l'IUT de Provence
http://x.heurtebise.free.fr

- *Le livre de Java premier langage*, d'A. Tasso

- http://xkcd.com, http://xkcd.free.fr

Programme des cours du semestre

- Récursivité
- Tris
- Notions de complexité
- Listes
- Arbres
- Programmation objet
- Langage PHP

Plan du cours 1 – Récursivité

Vu en cours :

- Introduction à la récursivité

Compléments pas vus en cours :

- Traces d'exécution de fonctions récursives

Plan du cours 1 – Récursivité

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives

Exercice du devoir sur le dessin d'une cible

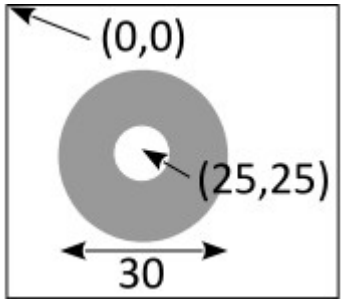
Le devoir du premier semestre proposait un exercice qui aurait pu se faire de manière “récursive” : le dessin de la cible.

On dispose d’un algorithme en Javascript **dessineDisque**(x,y,d,couleur) qui dessine un disque dans un bloc div d'une page web. Le centre de ce disque se situe aux coordonnées (x,y) (le point (0,0) se trouve en haut à gauche du bloc div), son diamètre fait d pixels et son code couleur en CSS correspond à la chaîne de caractères couleur (la bordure est de même couleur que le fond du disque).

Question 1. Écrivez deux instructions en Javascript qui dessinent un disque gris de centre (25,25) et de 30 pixels de diamètre, puis un disque blanc de centre (25,25) et de 10 pixels de diamètre.

```
dessineDisque(25,25,30,"grey");
dessineDisque(25,25,30,"rgb(255,255,255)");
```

chaîne de caractères contenant le code CSS de la couleur voulue



Exercice du devoir sur le dessin d'une cible

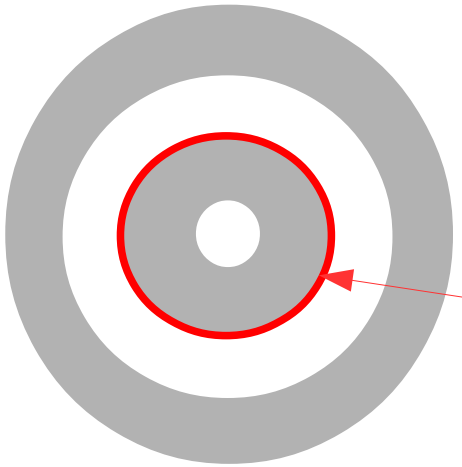
Question 2. Écrivez en Javascript un algorithme dessineCible qui prend en entrée quatre entiers x, y, d, et n, ainsi qu'une chaîne de caractères couleur codant une couleur en CSS, et qui dessine une cible de diamètre d. Cette cible est constituée de 2n disques dont la couleur alterne entre le blanc (couleur du plus petit disque) et la couleur codée par la variable d'entrée couleur. Le centre de la cible se situe aux coordonnées (x,y). Les diamètres des disques devront être calculés de telle sorte que la différence de diamètres de deux disques consécutifs soit égale au double du diamètre du plus petit disque de la cible (comme dans le dessin ci-dessus : différence de 20 pixels, soit le double du diamètre du disque blanc).

Première stratégie :

une boucle pour répéter n fois :
- dessin d'un disque gris,
- dessin d'un disque blanc un peu plus petit, à l'intérieur du gris

Exercice du devoir sur le dessin d'une cible

Question 2. Écrivez en Javascript un algorithme dessineCible qui prend en entrée quatre entiers x, y, d, et n, ainsi qu'une chaîne de caractères couleur codant une couleur en CSS, et qui dessine une cible de diamètre d. Cette cible est constituée de 2n disques dont la couleur alterne entre le blanc (couleur du plus petit disque) et la couleur codée par la variable d'entrée couleur. Le centre de la cible se situe aux coordonnées (x,y). Les diamètres des disques devront être calculés de telle sorte que la différence de diamètres de deux disques consécutifs soit égale au double du diamètre du plus petit disque de la cible (comme dans le dessin ci-dessus : différence de 20 pixels, soit le double du diamètre du disque blanc).

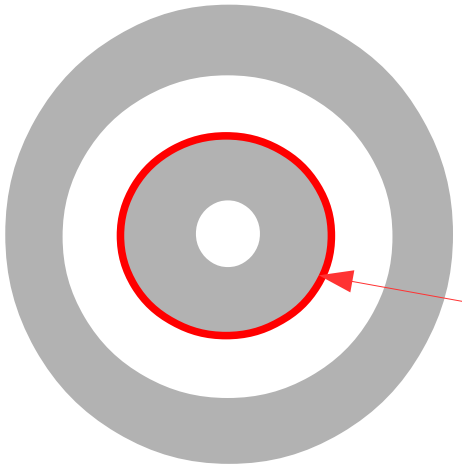


Deuxième stratégie, sans boucle :

- dessin d'un disque gris,
- dessin d'un disque blanc un peu plus petit, à l'intérieur du gris
- dessin d'une **cible contenant 2(n-1) disques**, à l'intérieur du disque blanc

Exercice du devoir sur le dessin d'une cible

Question 2. Écrivez en Javascript un algorithme dessineCible qui prend en entrée quatre entiers x, y, d, et n, ainsi qu'une chaîne de caractères couleur codant une couleur en CSS, et qui dessine une cible de diamètre d. Cette cible est constituée de 2n disques dont la couleur alterne entre le blanc (couleur du plus petit disque) et la couleur codée par la variable d'entrée couleur. Le centre de la cible se situe aux coordonnées (x,y). Les diamètres des disques devront être calculés de telle sorte que la différence de diamètres de deux disques consécutifs soit égale au double du diamètre du plus petit disque de la cible (comme dans le dessin ci-dessus : différence de 20 pixels, soit le double du diamètre du disque blanc).



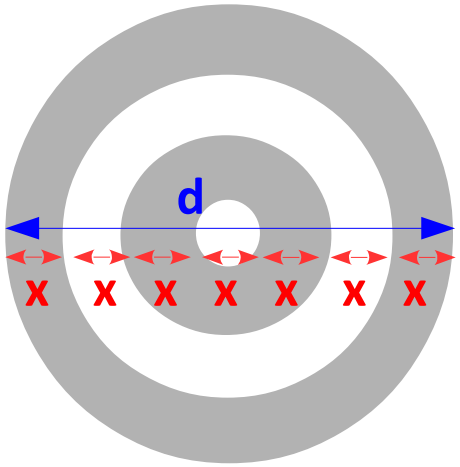
Deuxième stratégie, sans boucle :

- dessin d'un disque gris,
- dessin d'un disque blanc un peu plus petit, à l'intérieur du gris
- dessin d'une **cible faite de 2(n-1) disques**, à l'intérieur du disque blanc

} **seulement si n>0**

Exercice du devoir sur le dessin d'une cible

Comment connaître le diamètre des prochains disques à dessiner, si la cible a un diamètre **d** ?



→ La différence de diamètre entre deux disques successifs est 2**x**

→ Chaque paire de disques (un blanc, un gris) augmente le diamètre de 4**x**

→ La cible la plus petite (deux disques) a un diamètre de 4**x** – **x** **(on enlève x car le plus petit disque blanc a un diamètre de x seulement)**

→ on en déduit que **d** = 4**x***n – **x**

→ donc **d** = (4*n-1)***x**

→ donc en divisant par (4*n-1) : **x** = **d**/(4*n-1)

Exercice du devoir sur le dessin d'une cible

Première stratégie :

```
function dessineCible(x, y, d, n, couleur){
    var numDisques = 0;
    // On répète n fois :
    while(numDisques<n){
        // Dessin du disque gris :
        dessineDisque(x, y, d, couleur);
        d = d-d/(4*n-1);
        // Dessin du disque blanc :
        dessineDisque(x, y, d, "white");
        d = d-d/(4*n-1);
    }
}
```

Exercice du devoir sur le dessin d'une cible

Deuxième stratégie sans boucle :

```
function dessineCible(x, y, d, n, couleur){
    // On ne dessine des choses que si n>0
    if(n>0){
        // Dessin du disque gris :
        dessineDisque(x, y, d, couleur);
        d = d-d/(4*n-1);
        // Dessin du disque blanc :
        dessineDisque(x, y, d, "white");
        d = d-d/(4*n-1);
        // Dessin du reste de la cible,
        // c'est-à-dire une cible un peu plus petite :
        dessineCible(x, y, d, n-1, couleur);
    }
}
```

L'algorithme se répète “tout seul” jusqu'à ce que n arrive à la valeur 0...
C'est la magie de la **récursivité** !

Récursivité

« *La fonction qui s'appelle elle-même* »

2 façons de le voir :

- Une mise en abyme
- Une baguette magique algorithmique

Récursivité

- Une mise en abyme

La “minute xkcd” - Jeu de rôle sur table

http://xkcd.com/244
http://xkcd.free.fr?id=244

Récursivité

- Une mise en abyme

Photo Ethan Clements
http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html

http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme

Récursivité

- Une mise en abyme

Photo Ethan Clements
http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html

http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme

Comment dessiner ces images ?

Récursivité

- Une mise en abyme

Photo Ethan Clements
http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html

http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme

Comment dessiner ces images ?

L'image contient une plus petite version d'elle-même.

Récursivité

- Une mise en abyme

Photo Ethan Clements
http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html

http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme

Comment dessiner ces images ?

Pour dessiner la grande image, j'utilise le dessin de l'image en plus petit.

Récursivité

- Une mise en abyme

Qu’est-ce qu’une poupée russe ?

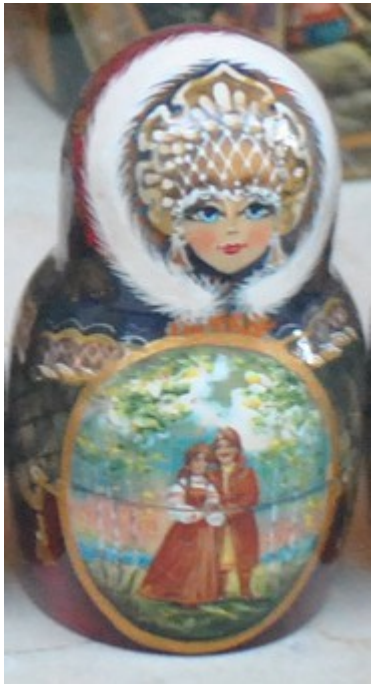


Photo Malix L.
http://www.flickr.com/photos/_malix_/6970987365/

Récursivité

- Une mise en abyme

Qu'est-ce qu'une poupée russe ?

Photo Malix L.
http://www.flickr.com/photos/_malix_/6970987365/

Une **poupée russe** est une poupée qui contient :
- soit rien
- soit une autre **poupée russe** plus petite

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple du dessin d'une cible :

Initialisation

Hérédité :

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

factorielle(1) = 1

factorielle(2) = 1x2 = 2

factorielle(3) = 1x2x3 = 6

factorielle(4) = 1x2x3x4 = 24

...

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

Initialisation :

factorielle(1) = 1

factorielle(2) = 1x2 = 2

Formule d'hérédité :

factorielle(3) = 1x2x3 = 6

factorielle(4) = 1x2x3x4 = 24

...

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

factorielle(1) = 1

factorielle(2) = 1x2 = 2

factorielle(3) = 1x2x3 = 6

factorielle(4) = 1x2x3x4 = 24

...

Initialisation :
factorielle(1) = 1

Formule d'hérédité :
factorielle(*n*) = ... factorielle(*n*-1) ...

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

factorielle(1) = 1

factorielle(2) = 1x2 = 2

factorielle(3) = 1x2x3 = 6

factorielle(4) = 1x2x3x4 = 24

...

Initialisation :
factorielle(1) = 1

Formule d'hérédité :
factorielle(*n*) = factorielle(*n*-1) x *n*

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

factorielle(1) = 1 —x2→

factorielle(2) = 1x2 = 2

factorielle(3) = 1x2x3 = 6

factorielle(4) = 1x2x3x4 = 24

...

Initialisation :
factorielle(1) = 1

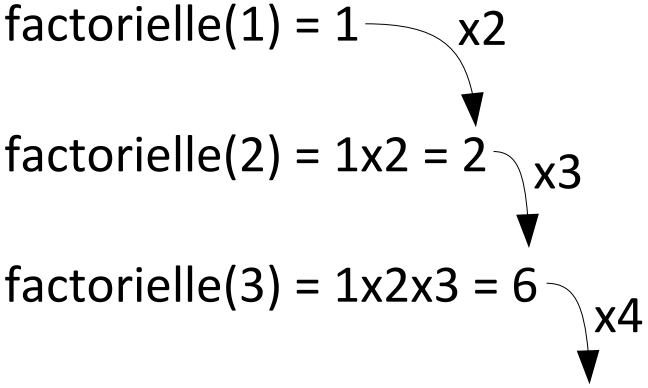
Formule d'hérédité :
factorielle(*n*) = factorielle(*n*-1) x *n*

Récursivité

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :



...

Initialisation :
factorielle(1) = 1

Formule d'hérédité :
factorielle(*n*) = factorielle(*n*-1) x *n*

Récursivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées :
Type de sortie :
Variable :
Début




Fin
```

Récursivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

Récursivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

La fonction factorielle
s'appelle elle-même !

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

entier

entier

Récursivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

version non récursive :

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        resultat ← 1
        Pour i de 2 à n faire :
                resultat ← i x resultat
        FinPour
        renvoyer resultat
Fin
```

Plan du cours 1 – Récursivité et tris

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

Récursivité

- Une baguette magique algorithmique

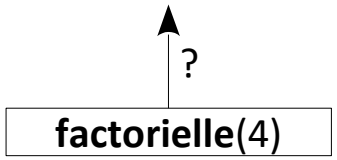
Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

Trace de **factorielle**(4) :



Récursivité

- Une baguette magique algorithmique

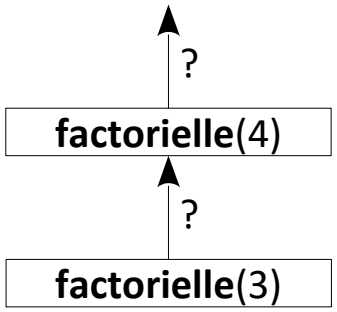
Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

Trace de **factorielle**(4) :



Récursivité

- Une baguette magique algorithmique

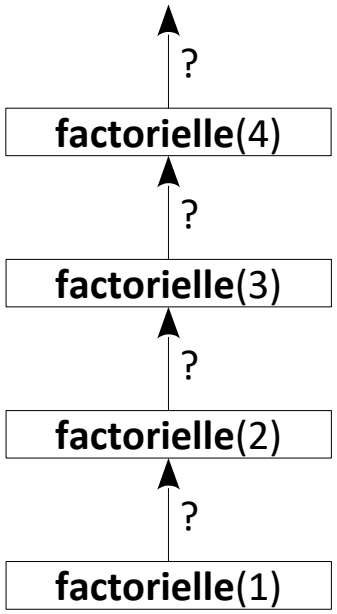
Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

Trace de **factorielle**(4) :



Récursivité

- Une baguette magique algorithmique

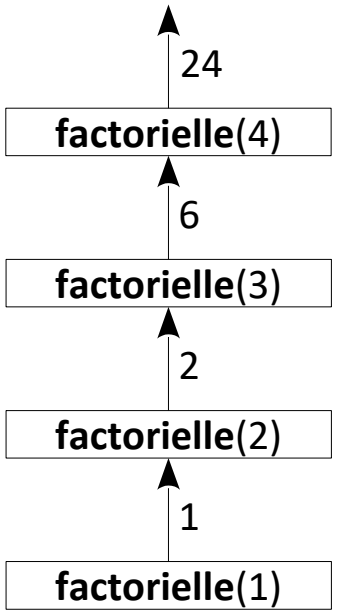
Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

```
Algorithme factorielle
Entrées : entier n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=1 alors :
                resultat ← 1
        sinon :
                resultat ← n x factorielle(n-1)
        Fin si
        renvoyer resultat
Fin
```

Trace de **factorielle**(4) :



Récursivité

Même concept que la **preuve par récurrence**

La “minute mathématique”

Théorème : Je sais monter une échelle

Récursivité

Même concept que la **preuve par récurrence**

La “minute mathématique”

Théorème : Je sais monter une échelle

Démonstration :

Initialisation : je sais monter depuis le sol jusqu'au premier barreau.

Hérédité : si je sais monter jusqu'au (n-1)-ième barreau, je saurai monter jusqu'au n-ième barreau.

Un autre algorithme récursif

La **multiplication**

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer $a \times n$:

Initialisation :

Formule d'hérédité :

Exemple :

$a \times 0 = 0$?

$a \times 1 = a$?

$a \times 2 = 2a$?

$a \times 3 = 3a$

Un autre algorithme récursif

La **multiplication**

Si je sais construire le n-ième objet à partir du (n-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer $a \times n$:

Initialisation :

$a \times 0 =$

Formule d'hérédité :

$a \times n = \ldots a \times (n\text{-}1) \ldots$

Exemple :

$a \times 0 = 0$?

$a \times 1 = a$?

$a \times 2 = 2a$?

$a \times 3 = 3a$

Un autre algorithme récursif

La **multiplication**

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer *a* x *n* :

Initialisation :

a x 0 =

Formule d'hérédité :

a x *n* = … *a* x (*n*-1) …

Exemple :

3 x 0 = 0 +3
3 x 1 = 3 +3
3 x 2 = 6 +3
3 x 3 = 9
3 x 4 = 12

Un autre algorithme récursif

La **multiplication**

Si je sais construire le n-ième objet à partir du (n-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer $a \times n$:

Initialisation :

$a \times 0 = 0$

Formule d'hérédité :

$a \times n = a \times (n\text{-}1) + a$

Un autre algorithme récursif

La **multiplication**

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer $a \times n$:

Initialisation :

$a \times 0 = 0$

Formule d'hérédité :

$a \times n = a \times (n\text{-}1) + a$

Algorithme **produit**
Entrées :
Type de sortie :
Variable :
Début

Fin

Un autre algorithme récursif

La **multiplication**

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer *a* x *n* :

Initialisation :

a x 0 = 0

Formule d'hérédité :

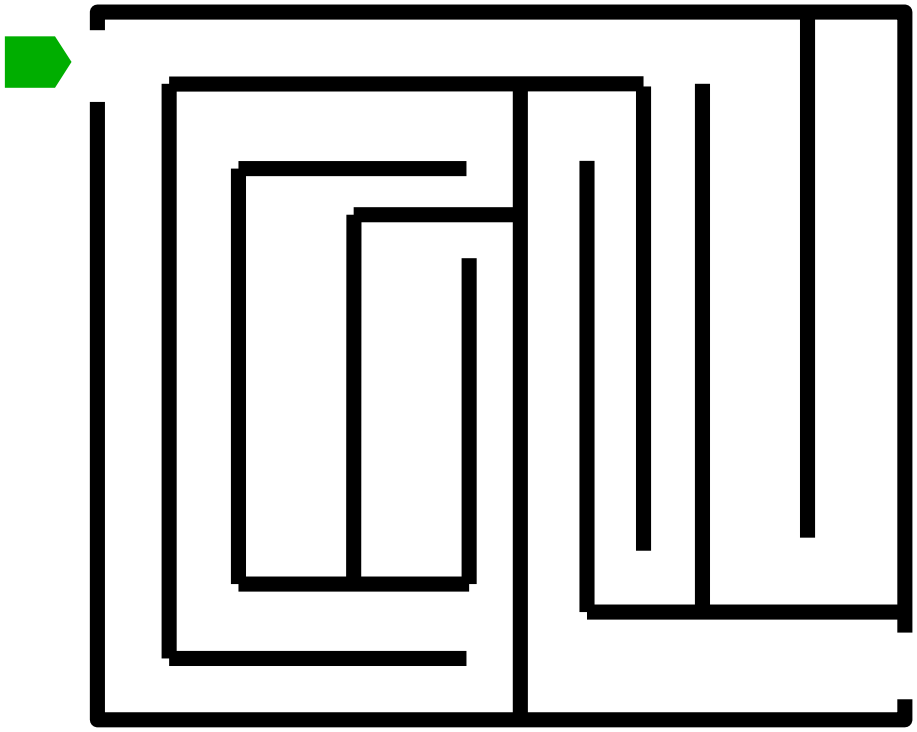
a x *n* = *a* x (*n*-1) + *a*

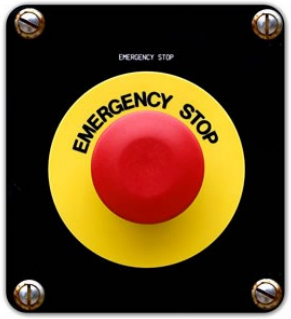
```
Algorithme produit
Entrées : entiers a et n
Type de sortie : entier
Variable : entier resultat
Début
        Si n=0 alors :
                resultat ← 0
        sinon :
                resultat ← produit(a,n-1) + a
        Fin si
        renvoyer resultat
Fin
```

Récursivité

Le labyrinthe et les robots tueurs :

10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.

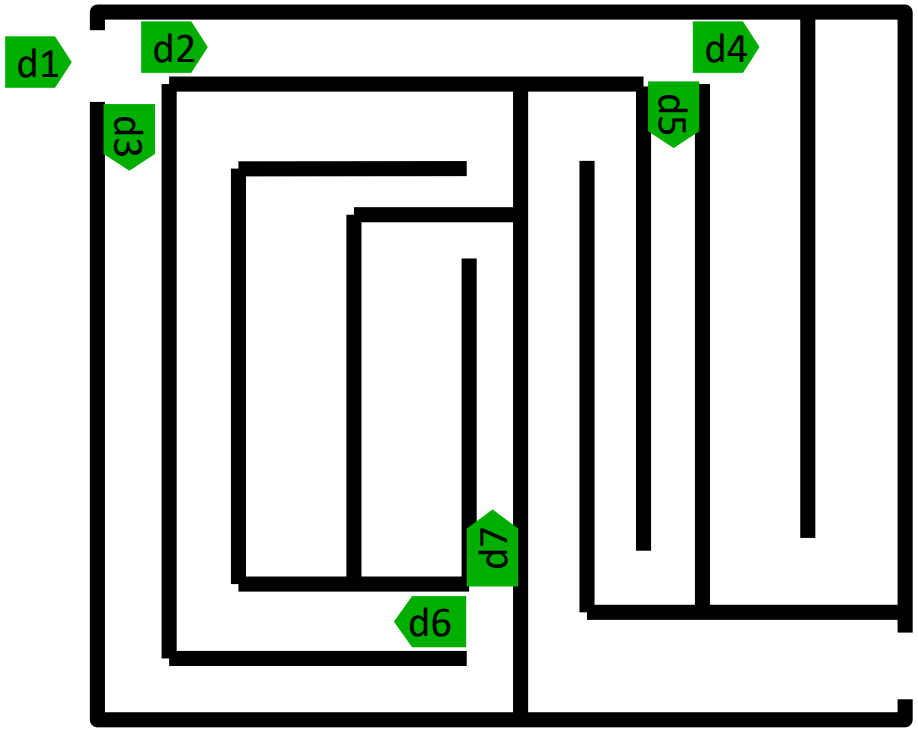




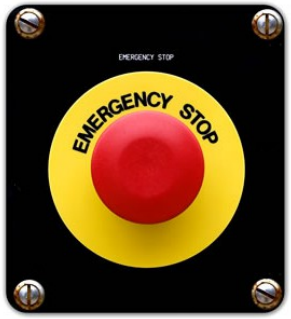
Récursivité

Le labyrinthe et les robots tueurs :

10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.



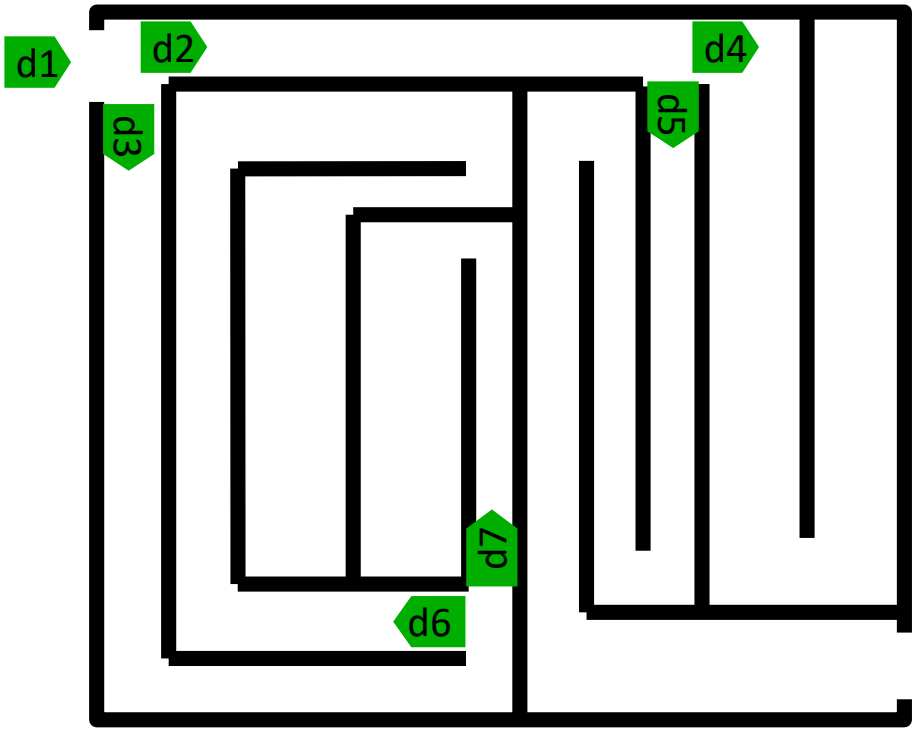
Algorithme **TrouveBouton**
Entrées : entier *nombrePersonnes*,
flottant *tempsRestant*,
chaîne de caractères *direction*
Type de sortie : booléen



Récursivité

Le labyrinthe et les robots tueurs :

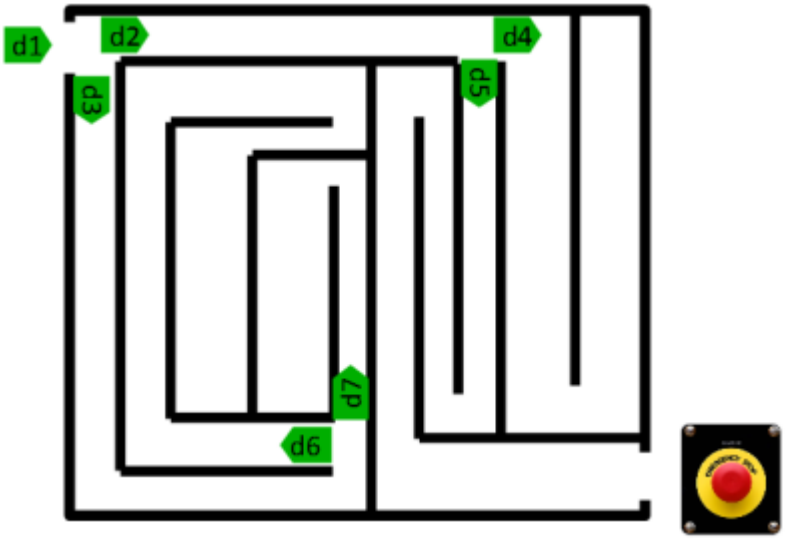
10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.



Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Récursivité



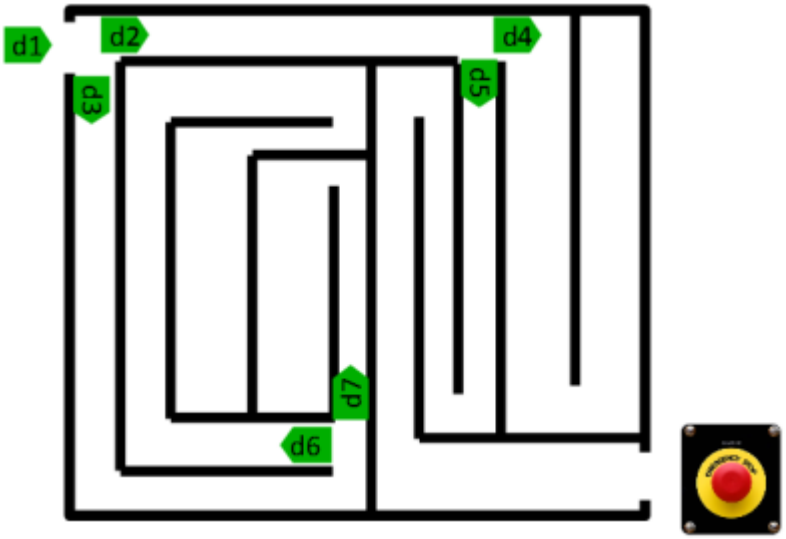
Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,”d1”) ?

Il faut faire la trace de **trouveBouton**(10,5,”d1”).

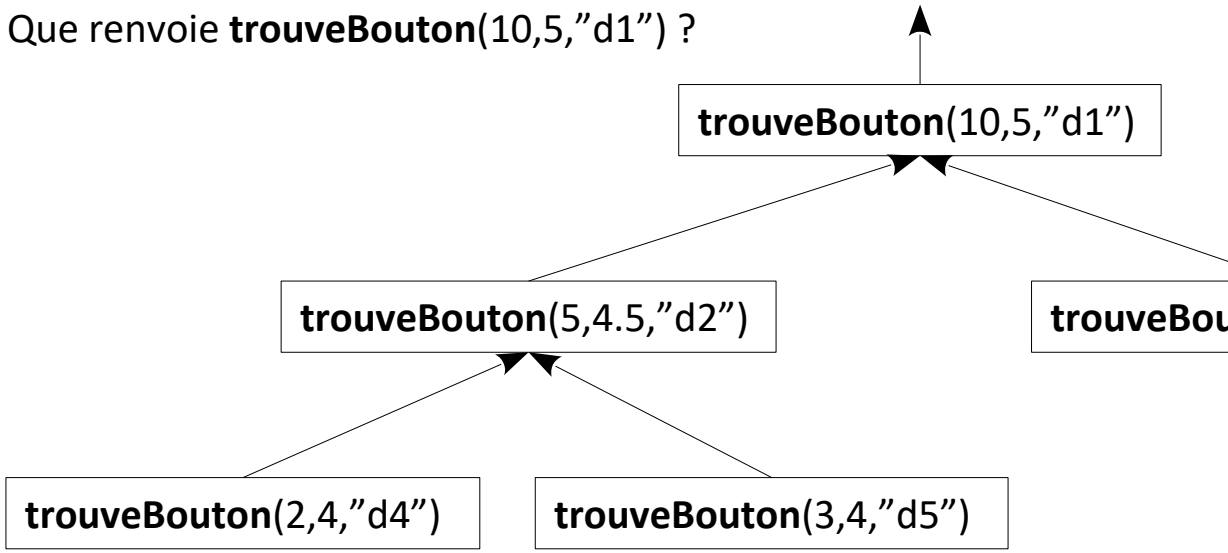
Récursivité



Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

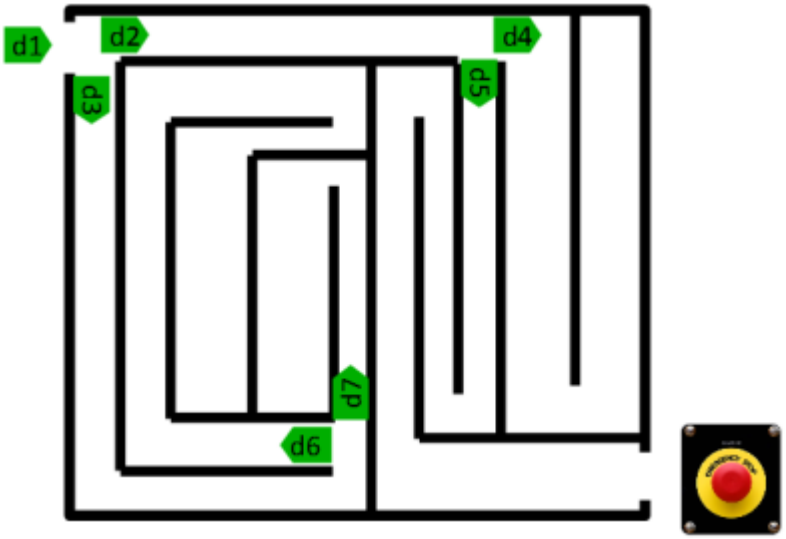
Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



Arbre d'exécution de **trouveBouton**(10,5,"d1").

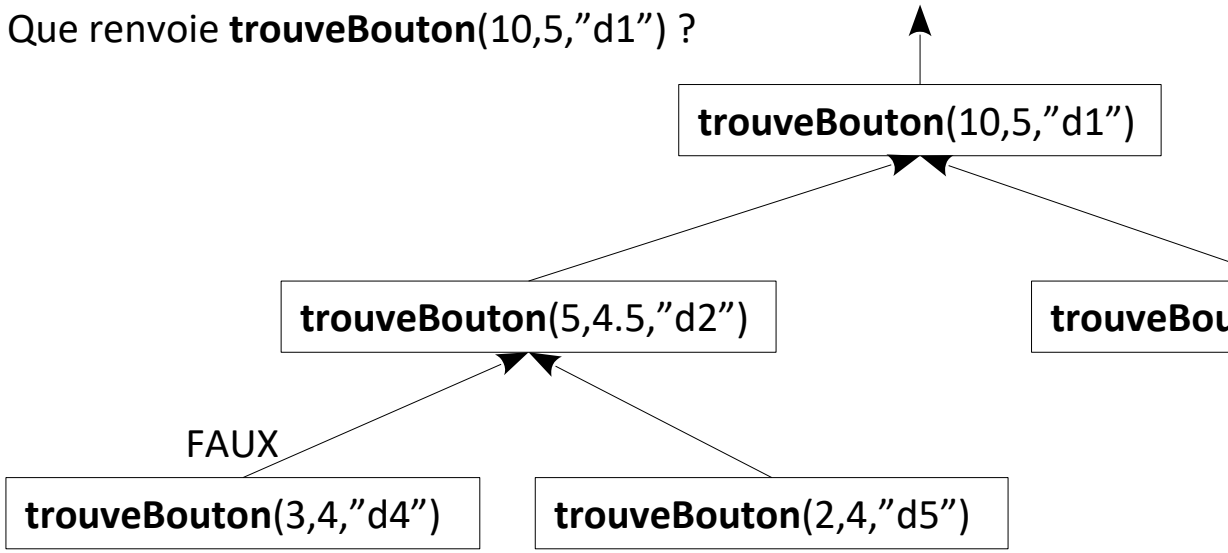
Récursivité



Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

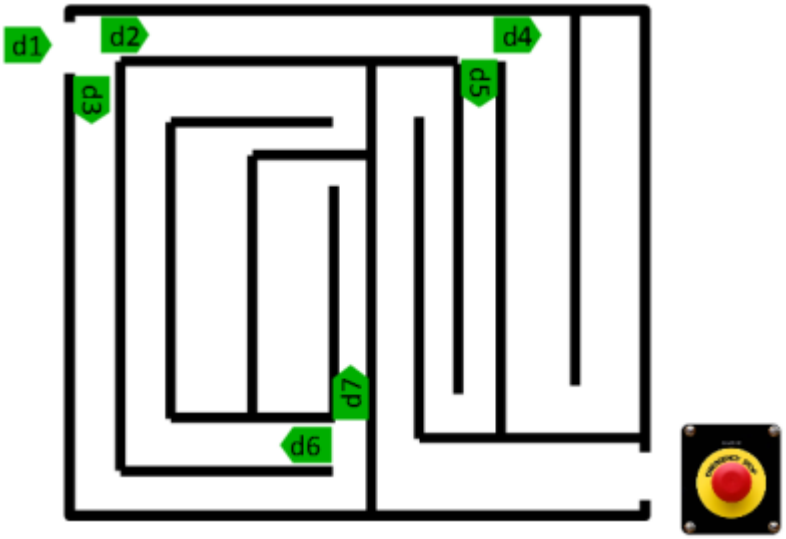
Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



Arbre d'exécution de **trouveBouton**(10,5,"d1").

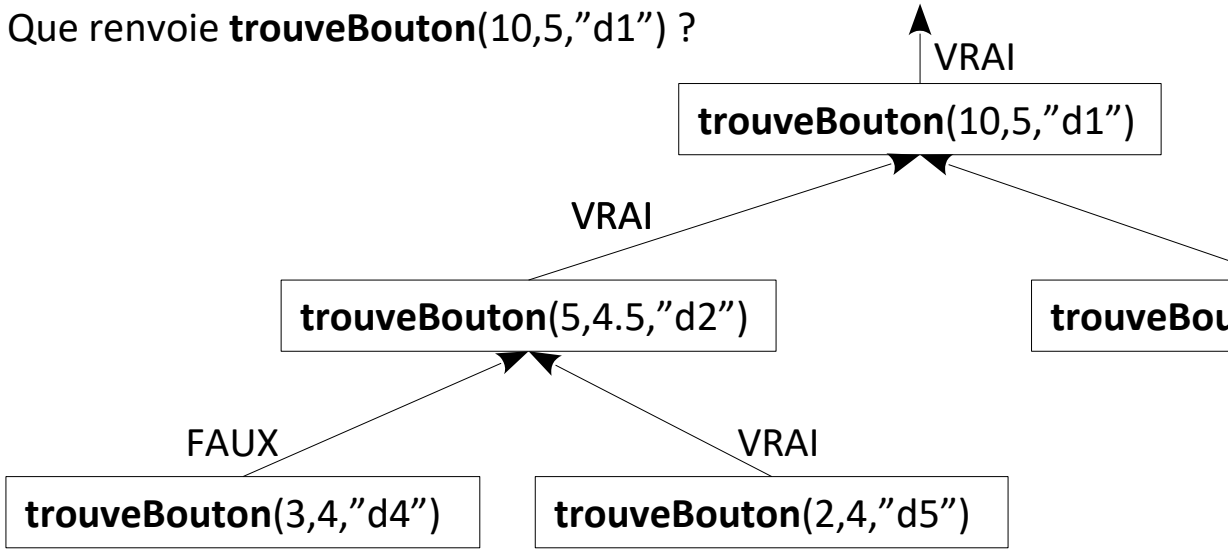
Récursivité



Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

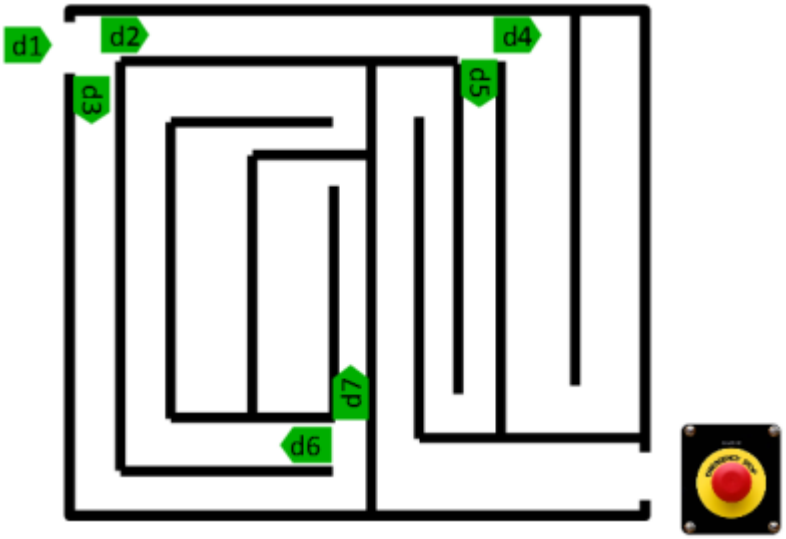
Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



Arbre d'exécution de **trouveBouton**(10,5,"d1").

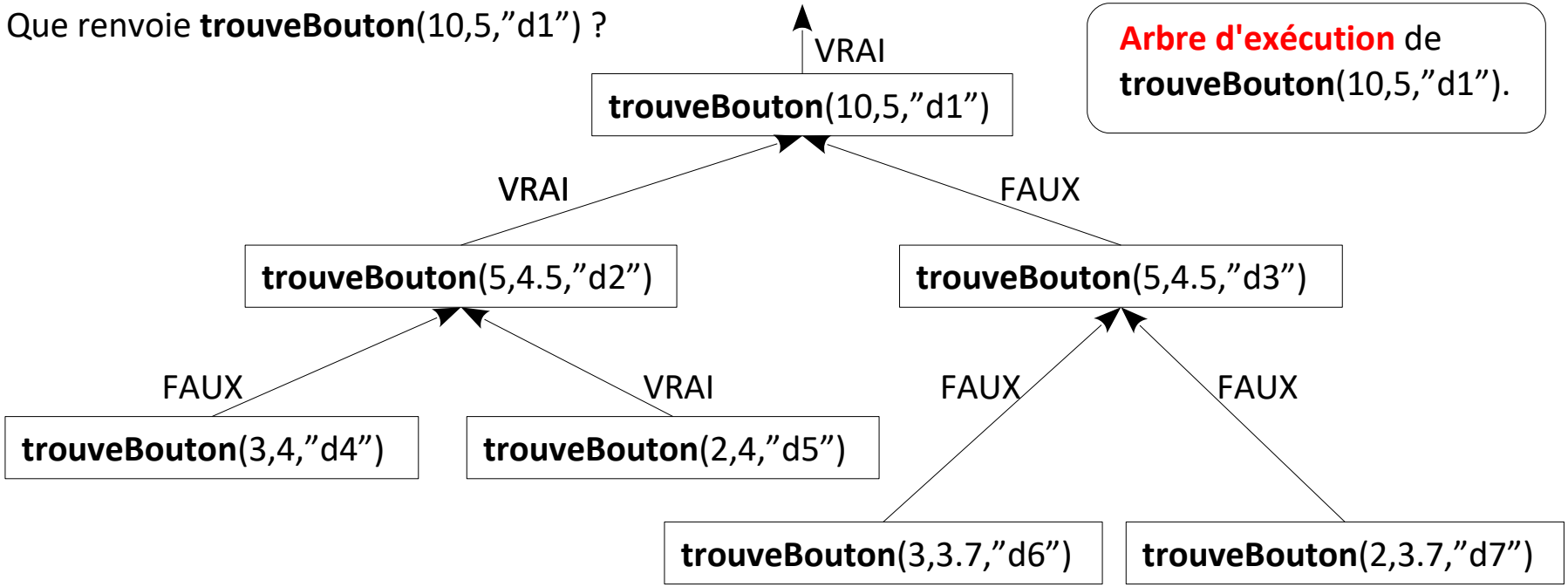
Récursivité



Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



Arbre d'exécution de **trouveBouton**(10,5,"d1").