

Algorithmique et structures de données

Cours INF TC1

Romain Vuillemot
`romain.vuillemot@ec-lyon.fr`

Département Math-Info
École Centrale de Lyon

2021-2022

Sommaire

Arbres

- Définition
- Structure de données
- Parcours d'arbres

Stratégies de programmation

- Types de problèmes
- Diviser pour régner
- Programmation dynamique
- Algorithmes gloutons

Séances de cours de INF TC1

- ▶ Cours 1: introduction aux algorithmiques et structures de données
Hash maps, Piles, Files, Listes de priorité, Listes chaînées
- ▶ Cours 2: stratégies de programmation
Arbres, Parcours Profondeur/Largeur, Diviser pour Régner, Programmation Dynamique, Glouton
- ▶ Cours 3: structure de données avancées
Graphes, Arbres couvrant, Parcours de Graphe, Recherche de Chemin
- ▶ Cours 4: algorithmes avancés
Algorithmes de Recherche, Tri, Heuristiques, Automates

Arbres

Sommaire

Arbres

- Définition
- Structure de données
- Parcours d'arbres

Stratégies de programmation

- Types de problèmes
- Diviser pour régner
- Programmation dynamique
- Algorithmes gloutons

Définition

Un **arbre** est un ensemble de sommets et arêtes (graphe) non-orienté, connexe et acyclique.

Exemples (en informatique..) ?

Définition

Un **arbre** est un ensemble de sommets et arêtes (graphe) non-orienté, connexe et acyclique.

Exemples (en informatique..) ?

Exemples. Représentation d'arbre généalogique (hiérarchie), réseau de diffusion d'information, système de fichiers, ..

Arbres

Définition

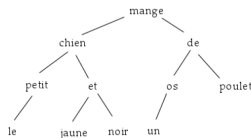
Un **arbre** est un ensemble de sommets et arêtes (graphe) non-orienté, connexe et acyclique.

Exemples (en informatique..) ?

Exemples. Représentation d'arbre généalogique (hiérarchie), réseau de diffusion d'information, système de fichiers, ..

- ▶ **Arbres binaires de recherche, arbres d'appels récurifs, ..**
- ▶ Arbres : n-aire, binaire , ABOH (ordonné horizontalement), AVL (Adelson-Velskii and Landis, self-balancing trees), ...
- ▶ B-arbres, Forêts, Treillis, Balanced tree

Représentation des arbres :



Arbres

Définition

Un **arbre** est un ensemble de sommets et arêtes (graphe) non-orienté, connexe et acyclique.

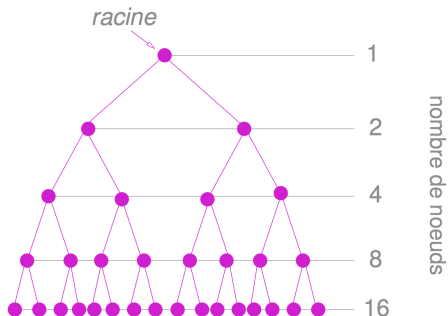
Quelques définitions :

- ▶ La racine est le noeud de niveau 0
- ▶ Les enfants sont les noeuds avec un parent
- ▶ Les feuilles sont les noeuds sans enfant
- ▶ la hauteur est la profondeur maximale des noeuds d'un arbre
- ▶ le degré d'un noeud est le nombre de ses descendants (enfants)
- ▶ le degré d'un arbre est le plus grand des degrés de ses noeuds

Arbres

Définition

Exemple (Arbres binaires de recherche) : structure de donnée qui permet de représenter des valeurs ordonnées et muni d'opérations d'insertion, de suppression, et de recherche.



Question :

Quel est la hauteur h d'un tel arbre ?

Arbres

Définition

Question :

Quel est la hauteur h d'un tel arbre ?

$$n = 2^{(h+1)} - 1$$

$$n + 1 = 2^{(h+1)}$$

$$\log(n + 1) = \log(2^{(h+1)})$$

$$\log(n + 1) = (h + 1)\log(2)$$

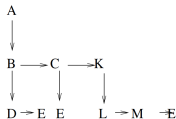
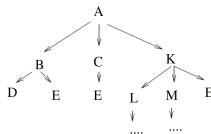
$$\log(n + 1)/\log(2) = h + 1$$

d'où $h = \log(n + 1)/\log(2) - 1$ h est au moins égal à $\log(n)$

Arbres

Structure de données

Listes d'adjacence (dict)



Abstraction For^et

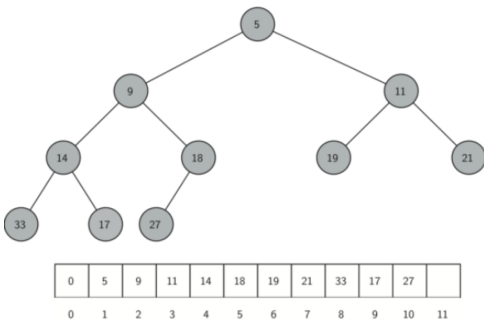
A	B	C	K
B	D	E	
C	E		
D			
E			
K	L	M	E
L		
M		
....			

Représentation par Listes

```
tree = {  
    'A' : ['B', 'C', 'K'],  
    'B' : ['D', 'E'],  
    'C' : ['E'],  
    'K' : ['L', 'M', 'E']  
}
```

Arbres Structure de données

Tableau (list)



Si arbre binaire complet/équilibré : si l'indice du nœud est égal à i , alors la place désignant son fils gauche est à $2i$ et la place désignant son fils droit est à $2i + 1$.

Arbres

Structure de données

Ensembles (set)

```
tree = { "a" : set(["b", "c"]),  
         "b" : set(["a", "d"]),  
         "c" : set(["a", "d"]),  
         "d" : set(["e"]),  
         "e" : set(["a"])  
       }
```

Les set permettent de stocker une liste non-ordonnée de sommets uniques. Cette liste correspond aux branches de l'arbre.

Arbres

Structure de données

Liste d'adjance avec valeurs ((dict))

```
tree = { '1': [ { '2': 0 }, { '3': 0 } ],  
        '2': [ { '4': 0 }, { '5': 0 } ],  
        '3': [ { '6': 0 }, { '7': 0 } ],  
        '4': [ { '8': 0 }, { '9': 0 } ],  
        '5': [ { '10': 0 } ]  
    }
```

Chaque noeud de l'arbre est un dictionnaire dont la clé est l'identifiant du noeud et la valeur est une liste. Cette liste contient un dictionnaire qui associe une valeur à chaque noeud.

Tuples

```
T = (13,
     (4,
      (1, None, None, None),
      (2, None, None, None),
      (3, None, None, None),
      (4, None, None, None)),
     (8,
      (5, None, None, None),
      (6, None, None, None),
      (7, None, None, None),
      (8, None, None, None)),
     (9,
      (9, None, None, None),
      (10, None, None, None),
      (11, None, None, None),
      (12, None, None, None)),
     (12,
      (13, None, None, None),
      (14, None, None, None),
      (15, None, None, None),
      (16, None, None, None)))
```

Un exemple d'arbre quaternaire où chaque noeud a 4 enfants au plus.
L'indentation dans ce cas est purement informative et l'arbre pourrait être stocké sur une seule ligne.

Objets (POO)

```
class Tree:
    def __init__(self):
        self.left = None
        self.right = None
        self.data = None

root = Tree()
root.data = "root"
root.left = Tree()
root.left.data = "left"
root.right = Tree()
root.right.data = "right"
```

Chaque noeud de l'arbre est une instance de la classe Tree qui est reliée à une autre (comme une liste chaînée). Le noeud possède des propriétés stockées dans un objet data.

Oobjets récurifs + liste

```
class Tree(object):  
    def __init__(self, name='root', children=None):  
        self.name = name  
        self.children = []  
        if children is not None:  
            for child in children:  
                self.add_child(child)  
    def __repr__(self):  
        return self.name  
    def add_child(self, node):  
        assert isinstance(node, Tree)  
        self.children.append(node)  
  
t = Tree('*', [Tree('1'),  
                Tree('2'),  
                Tree('+', [Tree('3'),  
                           Tree('4')])])])])
```

Intialisation en utilisant le constructeur de la classe Tree.

Module

Dot file

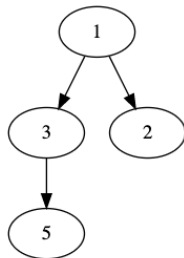
```
# Create Digraph object
dot = Digraph()

# Add nodes
dot.node('1')
dot.node('3')
dot.node('2')
dot.node('5')

# Add edges
dot.edges(['12', '13', '35'])

# Visualize the graph
dot
```

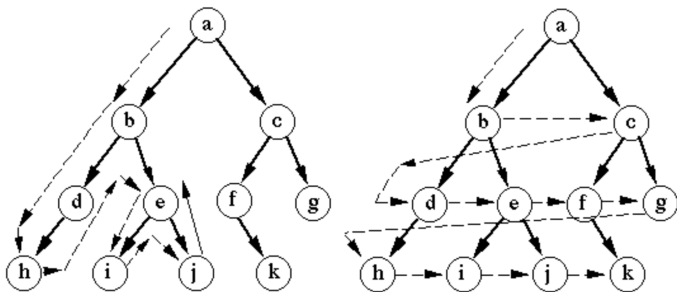
GraphViz



Arbres

Parcours d'arbres

Deux stratégies de parcours : **profondeur** et en **largeur**



Pseudo-code du parcours en **profondeur** :

1. Mettre le nœud source dans la **pile**.
2. Retirer le nœud du début de la pile pour le traiter.
3. Mettre tous les voisins non explorés dans la pile (au début).
4. Si la pile n'est pas vide reprendre à l'étape 2.

Arbres

Parcours d'arbres

Code **parcours en profondeur** (Deep First Traversal – DFT)

```
def dfs(graph, start):  
    stack = [start]  
    while stack:  
        vertex = stack.pop()  
        print(vertex) # traitement  
        stack.extend(graph[vertex])  
  
graph = {'A': set(['B', 'C']),  
        'B': set(['D', 'E', 'F']),  
        'C': set([]),  
        'D': set([]),  
        'E': set([]),  
        'F': set([])  
        }  
  
dfs(graph, 'A') # A B D F E C
```

Pour le parcours en **profondeur**, il existe différents types de traitements : *pré-ordre* (prefix), *mi-ordre* (infix) et *post-ordre* (postfix).

Soit :

- ▶ R = Racine
- ▶ D = sous-arbre Droit
- ▶ G = sous-arbre Gauche.

Il y a trois types (principaux) de parcours : observer la place de R

- ▶ Préfixé : R G D : pré-ordre
- ▶ Infixé : G R D : mi-ordre
- ▶ Postfixé : G D R : post-ordre

Arbres

Parcours d'arbres

Pour le parcours en **profondeur**, il existe différents types de traitements : *pré-ordre*, *post-ordre* et *mi-ordre*.

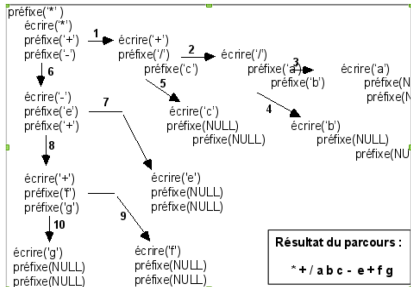
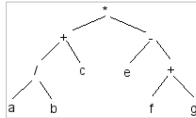
```
def Prefixe(R):
    if not vide (R):
        traiter(R); # R
        Prefixe(gauche (R)); # G
        Prefixe(droit(R)) ; # D

def Infixe(R):
    if not vide (R):
        Infixe(gauche (R)) ; # G
        traiter(R); # R
        Infixe(droit(R)); # D

def Postfixe(R):
    if not vide (R):
        Postfixe(gauche (R)); # G
        Postfixe(droit(R)); # D
        traiter(R); # R
```


Arbres

Exemple : Trace du parcours préfixé (pré-ordre) de l'arbre d'une expressions :

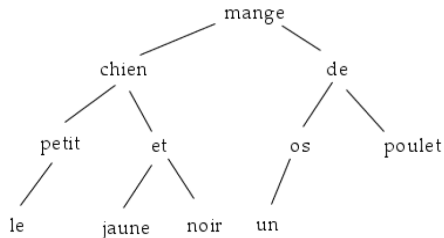


Arbres

Parcours d'arbres

Question :

Quel type de parcours pour retrouver les informations dans l'ordre dans l'arbre binaire suivant ?

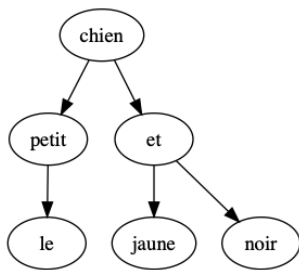


Arbres

Parcours d'arbres

Solution. Structure de donnée d'arbre en Objet ou en Dictionnaire et parcours en profondeur *infixe* (G R D).

<https://gitlab.ec-lyon.fr/rvuillem/inf-tc1/-/blob/master/TD01/code/arbre-parcours-profondeur.py>



```
from graphviz import Digraph

name = 'arbre-viz-profondeur'

g = Digraph('G', filename = name + '.gv',
            format='png') # pdf par default

g.edge('chien', 'petit')
g.edge('chien', 'et')
g.edge('petit', 'le')
g.edge('et', 'jaune')
g.edge('et', 'noir')

# génère et affiche le graphe
g.view()
```

Pseudo-code du parcours en **largeur** :

1. Mettre le nœud source dans la **file**.
2. Retirer le nœud du début de la file pour le traiter.
3. Mettre tous les voisins non explorés dans la file (à la fin).
4. Si la file n'est pas vide reprendre à l'étape 2.

Arbres

Parcours d'arbres

Code **parcours en largeur** (Breath First Traversal – BFS)

```
tree = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

def bfs(graph, node):
    queue = []
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            queue.append(neighbour)

bfs(tree, 'A')
```

Arbres

Parcours d'arbres

Code **parcours en largeur** avec mémorisation

```
import collections
class graph:
    def __init__(self, gdict=None):
        if gdict is None:
            gdict = {}
        self.gdict = gdict

def bfs(graph, startnode):
    seen, queue = set([startnode]), collections.deque([startnode])
    while queue:
        vertex = queue.popleft()
        marked(vertex)
        for node in graph[vertex]:
            if node not in seen:
                seen.add(node)
                queue.append(node)

def marked(n):
    print(n)

tree = { "a" : set(["b", "c"]),
         "b" : set(["e", "d"]),
         "c" : set(["f"]),
         "f" : set(["i"]),
         "e" : set(["g", "h"]),
         "d" : set(), "i" : set(), "g" : set(), "h" : set()
       }

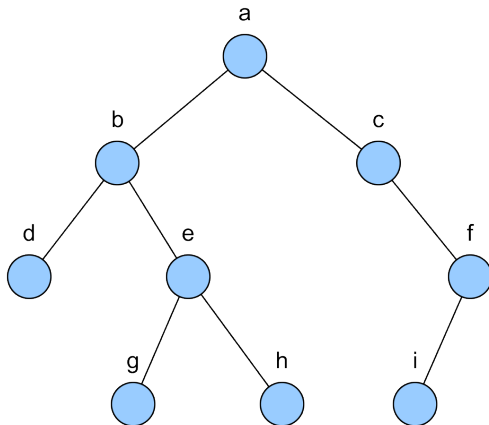
bfs(tree, "a")
```

Question :

Quel est le rendu du parcours en largeur sur cet arbre ?

Arbres

Parcours d'arbres



Arbres

Parcours d'arbres

Exemple. Qu'affiche le code ci-dessous ?

```
G = {
    'A' : ['B', 'C', 'F'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : ['G'],
    'E' : ['F'],
    'F' : ['G', 'H'],
    'G' : ['H'],
    'H' : []
}

v = set()
```

```
def traitement(v, G, n):
    if n not in v:
        v.add(n)
        print(n)
        for m in G[n]:
            traitement(v, G, m)

if __name__ == '__main__':
    traitement(v, G, 'A')

    if len(v) > 0:
        print("Que se passe-t-il ?")
```


Arbres

Parcours d'arbres

Exemple. Qu'affiche le code ci-dessous ?

```
G = {
    'A' : ['B', 'C', 'F'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : ['G'],
    'E' : ['F'],
    'F' : ['G', 'H'],
    'G' : ['H'],
    'H' : []
}

v = set()
```

```
def traitement(v, G, n):
    if n not in v:
        v.add(n)
        print(n)
        for m in G[n]:
            traitement(v, G, m)

if __name__ == '__main__':
    traitement(v, G, 'A')

    if len(v) > 0:
        print("Que se passe-t-il ?")
```

A B D G H E F C et Que se passe-t-il ?

Sommaire

Arbres

- Définition
- Structure de données
- Parcours d'arbres

Stratégies de programmation

- Types de problèmes
- Diviser pour régner
- Programmation dynamique
- Algorithmes gloutons

Stratégies de programmation

Un **problème** est une question ou un (ensemble) de questions auxquels un algorithme doit répondre.

Il en existe plusieurs classes:

- ▶ Problème de **décision** où l'on répond par un résultat binaire (au lieu de produire un résultat transformé ou nouveau).
- ▶ Problème de **recherche** qui consiste à renvoyer une information stockée dans une structure de données ou calculée dans un espace de recherche.
- ▶ Problème de **dénombrement** afin d'identifier toutes les solutions possibles.

Il n'existe pas qu'une seule méthode de résolution pour ces problèmes. Les problèmes peuvent évidemment être combinés entre eux, on parlera alors de problèmes **hybrides**.

Une **stratégie de programmation** est un ensemble d'actions aillant pour but de résoudre un problème précis de manière *précise*.

- ▶ Exemples de stratégies : **diviser pour régner, programmation dynamique, algorithme glouton**, ..
- ▶ Peuvent être regroupées en stratégies *top-down* vs *bottom-up*
- ▶ Certains algorithmes nécessitent des phases de pré/post-traitement afin de permettre de mettre en œuvre une stratégie
- ▶ Peuvent être combinées (stratégies *hybrides*)

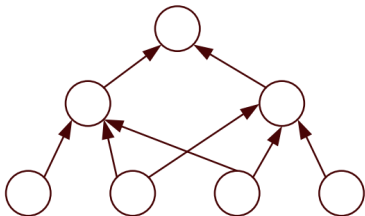
Il existe d'autres stratégies : backtracking, algorithmes approximatifs, recherche combinatoire, etc.

Stratégies de programmation

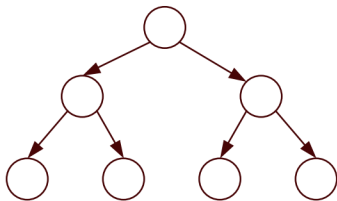
Types de problèmes

Remarque : approches top-down vs bottom-up

Bottom-Up



Top-Down



Notes :

1. Il est possible de passer de l'un à l'autre.
2. Les structures d'arbres sont très fréquentes pour stocker les résultats et évaluer la solution finale.

Diviser pour régner

Stratégies de programmation

Diviser pour régner

La stratégie **diviser pour régner** consiste à :

1. *Diviser* le problème en sous-problèmes de même type.
2. *Régner* afin de résoudre les problèmes de manière récursive.
3. *Combiner* les réponses de manière appropriée.

Stratégies de programmation

Diviser pour régner

La stratégie **diviser pour régner** consiste à :

1. *Diviser* le problème en sous-problèmes de même type.
2. *Régner* afin de résoudre les problèmes de manière récursive.
3. *Combiner* les réponses de manière appropriée.

Il s'agit d'une approche récursive comme vue précédemment !

Stratégies de programmation

Diviser pour régner

La stratégie **diviser pour régner** consiste à :

1. *Diviser* le problème en sous-problèmes de même type.
2. *Régner* afin de résoudre les problèmes de manière récursive.
3. *Combiner* les réponses de manière appropriée.

Il s'agit d'une approche récursive comme vue précédemment !

Des exemples :

- ▶ **Recherche dichotomique**
- ▶ **Tri fusion**, tri rapide
- ▶ Transformation de Fourier Rapide (Cooley–Tukey)
- ▶ Multiplications rapide (Karatsuba)
- ▶ ...

Stratégies de programmation

Diviser pour régner

Exemple (Recherche dichotomique) : à partir d'une liste *triée* comparer un élément à chercher avec la valeur du milieu du tableau, et en fonction du résultat répéter sur la sous-partie gauche ou droite du tableau.

Stratégies de programmation

Diviser pour régner

Exemple (Recherche dichotomique) : à partir d'une liste *triée* comparer un élément à chercher avec la valeur du milieu du tableau, et en fonction du résultat répéter sur la sous-partie gauche ou droite du tableau.

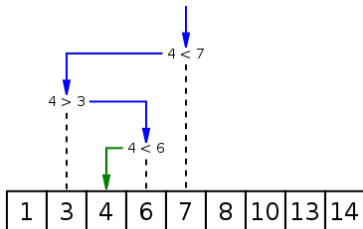
Exemple : [1, 2, 4, 6, 7, 8, 10, 13, 14] chercher 4

Stratégies de programmation

Diviser pour régner

Exemple (Recherche dichotomique) : à partir d'une liste *triée* comparer un élément à chercher avec la valeur du milieu du tableau, et en fonction du résultat répéter sur la sous-partie gauche ou droite du tableau.

Exemple : [1, 2, 4, 6, 7, 8, 10, 13, 14] chercher 4

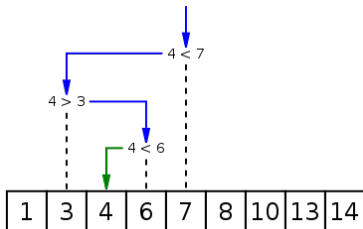


Stratégies de programmation

Diviser pour régner

Exemple (Recherche dichotomique) : à partir d'une liste *triée* comparer un élément à chercher avec la valeur du milieu du tableau, et en fonction du résultat répéter sur la sous-partie gauche ou droite du tableau.

Exemple : [1, 2, 4, 6, 7, 8, 10, 13, 14] chercher 4



Complexité $O(\log N)$ (mais suppose la liste déjà triée !)

Stratégies de programmation

Diviser pour régner

Code. Recherche dichotomique (version itérative)

```
def recherche_dichotomique(element, liste_triee):  
    a = 0  
    b = len(liste_triee)-1  
    m = (a+b)//2  
    while a < b :  
        if liste_triee[m] == element :  
            return m  
        elif liste_triee[m] > element :  
            b = m-1  
        else :  
            a = m+1  
    m = (a+b)//2  
    return a
```

Code. Recherche dichotomique (version récursive)

```
def dichotomique_rec(element, liste_triee, a=0, b=-1):  
    if a == b :  
        return a  
    if b == -1 :  
        b = len(liste_triee)-1  
    m = (a+b)//2  
    if liste_triee[m] == element :  
        return m  
    elif liste_triee[m] > element :  
        return dichotomique_rec(element, liste_triee, a, m-1)  
    else :  
        return dichotomique_rec(element, liste_triee, m+1, b)  
  
n = 100  
print(dichotomique_rec(n/2, range(n)))
```


Stratégies de programmation

Diviser pour régner

Exemple (Tri fusion) : diviser une liste en sous-listes, chacune contenant 1 élément (cas d'arrêt); fusionner les sous-listes en conservant l'ordre.

Stratégies de programmation

Diviser pour régner

Exemple (Tri fusion) : diviser une liste en sous-listes, chacune contenant 1 élément (cas d'arrêt); fusionner les sous-listes en conservant l'ordre.

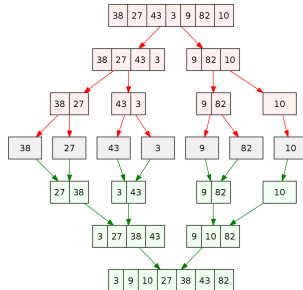
Exemple : [38, 27, 43, 3, 9, 82, 10]

Stratégies de programmation

Diviser pour régner

Exemple (Tri fusion) : diviser une liste en sous-listes, chacune contenant 1 élément (cas d'arrêt); fusionner les sous-listes en conservant l'ordre.

Exemple : [38, 27, 43, 3, 9, 82, 10]



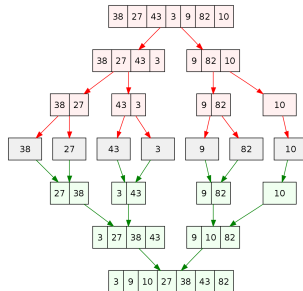
Complexité ?

Stratégies de programmation

Diviser pour régner

Exemple (Tri fusion) : diviser une liste en sous-listes, chacune contenant 1 élément (cas d'arrêt); fusionner les sous-listes en conservant l'ordre.

Exemple : [38, 27, 43, 3, 9, 82, 10]



Complexité ?
 $O(n \log(n))$

Stratégies de programmation

Diviser pour régner

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves

        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half

    i = j = k = 0

    # Copy data to temp arrays L[] and R[]
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            arr[k] = L[i]
            i+=1
        else:
            arr[k] = R[j]
            j+=1
        k+=1

    # Checking if any element was left
    while i < len(L):
        arr[k] = L[i]
        i+=1
        k+=1

    while j < len(R):
        arr[k] = R[j]
        j+=1
        k+=1
```

Bilan/limites

- ▶ Similaire à la récursion (appels, ..)
- ▶ Peut être implémenté de manière non-réursive avec piles, files, ..
- ▶ On peut être amené à évaluer le même problème (si il y a dépendance entre eux), dans ce cas utiliser la programmation dynamique !

Programmation dynamique

Stratégies de programmation

Programmation dynamique

La **programmation dynamique** consiste à décomposer un problème en sous-problèmes, *résoudre* les sous-problèmes; et les *combiner* pour obtenir la solution au problème initial. Les étapes sont les suivantes :

1. Caractériser la structure d'une solution optimale.
2. Définir récursivement la valeur d'une solution optimale.
3. Re-construire la solution optimale à partir des calculs.

Stratégies de programmation

Programmation dynamique

Remarques :

- ▶ S'applique aux problèmes qui ont une sous-structure optimale.
- ▶ Également aux problèmes dont les solutions sont souvent *liées entre elles* (ce qui différencie de diviser pour régner).
- ▶ Approche de *memoization* qui consiste à stocker une solution intermédiaire (par exemple dans une table)

Des exemples :

- ▶ **Suite de Fibonacci**
- ▶ **Rod cutting**
- ▶ **Recherche du plus court chemin**
- ▶ Alignement de séquences, recherche de plus longue sous-séquence
- ▶ ...

Beaucoup d'exemples existent !¹

¹<https://medium.com/@codingfreak/top-50-dynamic-programming-practice-problems-4208fed71aa3>

Stratégies de programmation

Programmation dynamique

Exemple (Suite de Fibonacci): calculer le n -ème nombre de la suite de Fibonacci, laquelle est déterminée de la façon suivante :

$$fib(n) = fib(n - 1) + fib(n - 2), n \in N$$

1, 1, 2, 3, 5, 8, 13, 21, .. pour définir le n -ème nombre ($n = 9$)

Stratégies de programmation

Programmation dynamique

Exemple (Suite de Fibonacci): calculer le n -ème nombre de la suite de Fibonacci, laquelle est déterminée de la façon suivante :

$$fib(n) = fib(n - 1) + fib(n - 2), n \in \mathbb{N}$$

1, 1, 2, 3, 5, 8, 13, 21, .. pour définir le n -ème nombre ($n = 9$)

Approche récursive (naïve)?

Exemple (Suite de Fibonacci): calculer le n -ème nombre de la suite de Fibonacci, laquelle est déterminée de la façon suivante :

$$fib(n) = fib(n - 1) + fib(n - 2), n \in N$$

1, 1, 2, 3, 5, 8, 13, 21, .. pour définir le n -ème nombre ($n = 9$)

Approche récursive (naïve)?

```
def fib(n):  
    if n < 2:  
        return n  
    else:  
        return fib(n - 1) + fib(n - 2)  
  
print(list(map(fib, range(8))))  
> [0, 1, 1, 2, 3, 5, 8, 13]
```

Exemple (Suite de Fibonacci): calculer le n -ème nombre de la suite de Fibonacci, laquelle est déterminée de la façon suivante :

$$fib(n) = fib(n - 1) + fib(n - 2), n \in N$$

1, 1, 2, 3, 5, 8, 13, 21, .. pour définir le n -ème nombre ($n = 9$)

Approche récursive (naïve)?

```
def fib(n):  
    if n < 2:  
        return n  
    else:  
        return fib(n - 1) + fib(n - 2)  
  
print(list(map(fib, range(8))))  
> [0, 1, 1, 2, 3, 5, 8, 13]
```

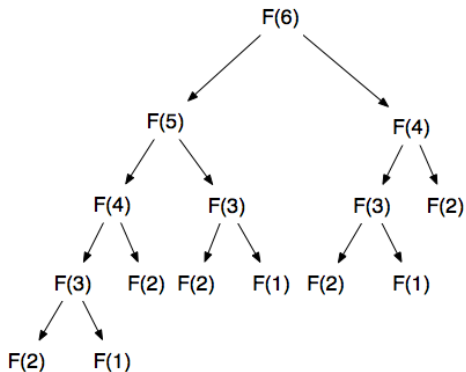
Question :

Quelle est l'arbre d'appels récursifs ?

Stratégies de programmation

Programmation dynamique

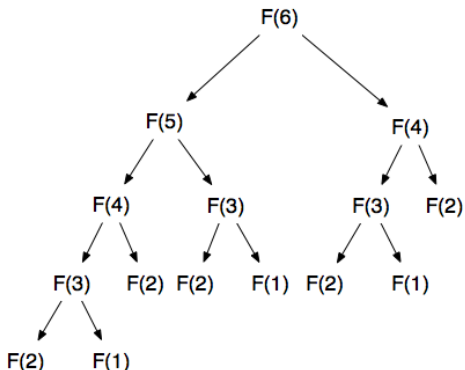
Arbre d'appels récursif (pour $n = 6$) :



Stratégies de programmation

Programmation dynamique

Arbre d'appels récursif (pour $n = 6$) :



Besoin de stocker (mémoiser) les calculs déjà effectués et ré-utilisés !

Stratégies de programmation

Programmation dynamique

Utilisation d'une **table de hachage** pour mémoriser :

```
def fib(n, lookup):  
  
    # Cas d'arrêt  
    if n == 0 or n == 1 :  
        lookup[n] = n  
  
    # On calcule la valeur si pas déjà calculée  
    if lookup[n] is None:  
        lookup[n] = fib(n-1 , lookup)  + fib(n-2 , lookup)  
  
    # On renvoie la n-eme valeur  
    return lookup[n]  
  
def main():  
    n = 6  
    max = 100  
    # Initialise la table de cache  
    lookup = [None]*(max)  
    print("Le nombre de fibonacci est ", fib(n, lookup))  
    # Le nombre de fibonacci est 8  
  
if __name__=="__main__":  
    main()
```


Stratégies de programmation Programmation dynamique

Exemple (Rod cutting) : un bâton de longueur n et une table de prix de découpe, quel est la découpe la plus lucrative ?

longueur (i)		1	2	3	4	5	6	7	8

prix (pi)		1	5	8	9	10	17	17	20

Stratégies de programmation Programmation dynamique

Exemple (Rod cutting) : un bâton de longueur n et une table de prix de découpe, quel est la découpe la plus lucrative ?

longueur (i)		1	2	3	4	5	6	7	8

prix (p_i)		1	5	8	9	10	17	17	20



(a)



(b)



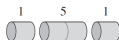
(c)



(d)



(e)



(f)



(g)



(h)

Stratégies de programmation Programmation dynamique

Exemple (Rod cutting) : un bâton de longueur n et une table de prix de découpe, quel est la découpe la plus lucrative ?

longueur (i)		1	2	3	4	5	6	7	8

prix (p_i)		1	5	8	9	10	17	17	20



(a)



(b)



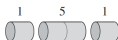
(c)



(d)



(e)



(f)



(g)



(h)

La stratégie optimale est (c) découper en 2 pièces de longueur 2 qui ont une valeur totale $5 + 5 = 10$.

Stratégies de programmation Programmation dynamique

Exemple (Rod cutting) : un bâton de longueur n et une table de prix de découpe, quel est la découpe la plus lucrative ?

longueur (i)		1	2	3	4	5	6	7	8

prix (p_i)		1	5	8	9	10	17	17	20



(a)



(b)



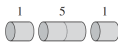
(c)



(d)



(e)



(f)



(g)



(h)

La stratégie optimale est (c) découper en 2 pièces de longueur 2 qui ont une valeur totale $5 + 5 = 10$.

Cas général : $V_n = \max_{1 \leq i \leq n} (p_i + V_{n-i})$

Stratégies de programmation

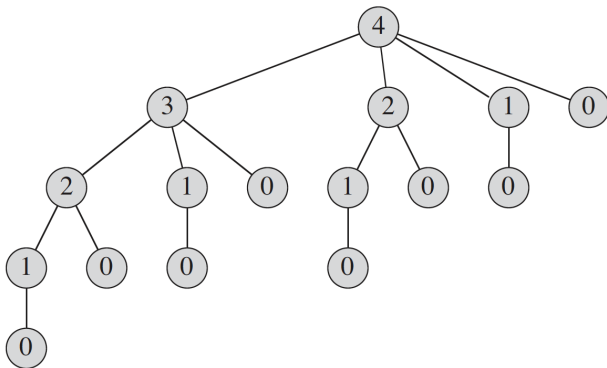
Programmation dynamique

Tableau de calcul.

Stratégies de programmation

Programmation dynamique

Arbre d'appels récursifs pour $n = 4$ (montre les chemins possibles depuis la racine).



Stratégies de programmation

Programmation dynamique

```
INT_MIN = 0

def cutRod(price, n):

    # Initialisation tables de cache
    val = [0 for x in range(n+1)]
    val[0] = 0

    for i in range(1, n+1):
        max_val = INT_MIN
        for j in range(i):
            max_val = max(max_val, price[j] + val[i-j-1])
        val[i] = max_val

    return val[n]

arr = [1, 5, 8, 9, 10, 17, 17, 20]
size = len(arr)
print("Valeur maximum de découpe " + str(cutRod(arr, size)))
```

Bilan/Limites

- ▶ Il faut étudier chaque problème au cas par cas.
- ▶ Le stockage d'un nombre important de résultats partiels, ce qui demande un usage important de la mémoire.
- ▶ Adapté à certains problèmes seulement (min, max, comptage du nombre de solutions)

Algorithmes gloutons

La stratégie d'un **algorithme glouton** consiste à faire des choix optimaux *locaux* avec l'intention d'atteindre un optimal *global*.

Des exemples

- ▶ Recherche de plus court chemin (Dijkstra, A*, ..)
- ▶ Arbres couvrant minimaux (Kruskal, Prim), **coloriage de graphe**, voyageur de commerce, etc.. (en général les problèmes liés aux graphes)
- ▶ **Codage de Huffman**
- ▶ KNN (K Nearest Neighbours) résolution du voyageur de commerce
- ▶ Sac à dos, Rendu de monnaie, ..
- ▶ ...

Stratégies de programmation

Algorithmes gloutons

Exemple (Sac à dos) consiste à remplir un sac à dos, ne pouvant supporter plus d'une certaine masse, avec tout ou partie d'un ensemble donné d'objets ayant chacun une masse et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser la masse maximum.

Exemple (Sac à dos) solution gloutonne

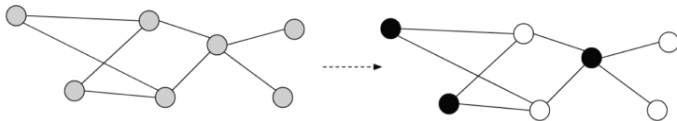
```
def sacDos(W, wt, val, n):  
  
    if n == 0 or W == 0 : # Conditions d'arrêt  
        return 0  
  
    if (wt[n-1] > W): # on ne prend pas les els. avec poids trop fort  
        return sacDos(W, wt, val, n-1)  
    else:  
        return max(val[n-1] + sacDos(W-wt[n-1], wt, val, n-1),  
                    sacDos(W, wt, val, n-1))  
  
val = [50,100,150,200]  
wt = [8,16,32,40]  
W = 64  
n = len(val)  
print(sacDos(W, wt, val, n))
```

Stratégies de programmation

Algorithmes gloutons

Exemple (Coloriage de graphe) consiste à assigner une couleur à chaque nœud différente de son voisin

Graphe 2-colorable

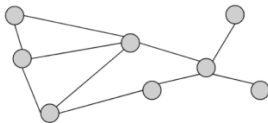
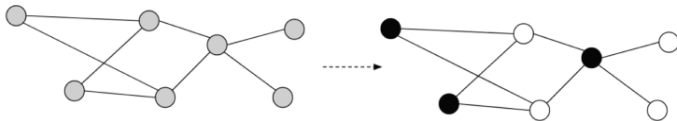


Stratégies de programmation

Algorithmes gloutons

Exemple (Coloriage de graphe) consiste à assigner une couleur à chaque nœud différente de son voisin

Graphe 2-colorable

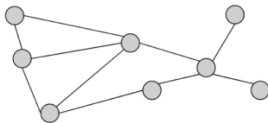
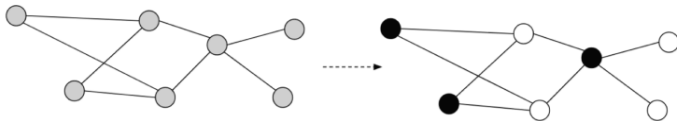


Stratégies de programmation

Algorithmes gloutons

Exemple (Coloriage de graphe) consiste à assigner une couleur à chaque nœud différente de son voisin

Graphe 2-colorable



Problème NP-Complet pour plus de 3 couleurs..

Exemple (Coloriage de graphe)

```
Entrée :  
liste ordonnée V des n sommets d'un graphe G  
liste ordonnée C de couleurs  
Pour i variant de 1 à n  
  v = V[i]  
  couleur = la première couleur de C non utilisée par les voisins de v  
  colorier(v, couleur)  
Fin pour
```


Exemple (Codage de Huffman) : création d'un code préfixe permettant de compression (sans perte) de l'information. Dans un premier temps construire l'arbre de fréquence des caractères, et ensuite parcourir l'arbre pour obtenir le code (0 pour les branches de gauche, 1 pour celles de droite).

Exemple (Codage de Huffman) : création d'un code préfixe permettant de compression (sans perte) de l'information. Dans un premier temps construire l'arbre de fréquence des caractères, et ensuite parcourir l'arbre pour obtenir le code (0 pour les branches de gauche, 1 pour celles de droite).

Étapes à suivre :

1. Créer une file avec les fréquences des lettres
2. Prendre les deux caractères les moins fréquents de la liste
3. Créer un nœud dans un arbre qui contient la somme des fréquences, et les deux caractères comme enfants. Ajouter ce nœud dans la file et supprimer les enfants.
4. Répéter 2 et 3 jusqu'à plus d'éléments, le dernier élément étant la racine de l'arbre.

Exemple (Codage de Huffman) : création d'un code préfixe permettant de compression (sans perte) de l'information. Dans un premier temps construire l'arbre de fréquence des caractères, et ensuite parcourir l'arbre pour obtenir le code (0 pour les branches de gauche, 1 pour celles de droite).

Étapes à suivre :

1. Créer une file avec les fréquences des lettres
2. Prendre les deux caractères les moins fréquents de la liste
3. Créer un nœud dans un arbre qui contient la somme des fréquences, et les deux caractères comme enfants. Ajouter ce nœud dans la file et supprimer les enfants.
4. Répéter 2 et 3 jusqu'à plus d'éléments, le dernier élément étant la racine de l'arbre.

Exemple : AAAA BB CCC (inclure les espaces !)

Exemple (Codage de Huffman) : création d'un code préfixe permettant de compression (sans perte) de l'information. Dans un premier temps construire l'arbre de fréquence des caractères, et ensuite parcourir l'arbre pour obtenir le code (0 pour les branches de gauche, 1 pour celles de droite).

Étapes à suivre :

1. Créer une file avec les fréquences des lettres
2. Prendre les deux caractères les moins fréquents de la liste
3. Créer un nœud dans un arbre qui contient la somme des fréquences, et les deux caractères comme enfants. Ajouter ce nœud dans la file et supprimer les enfants.
4. Répéter 2 et 3 jusqu'à plus d'éléments, le dernier élément étant la racine de l'arbre.

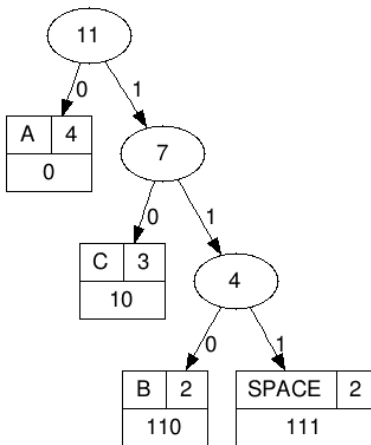
Exemple : AAAA BB CCC (inclure les espaces !)

Fréquences A (4) C (3) B (2) _ (2)

Stratégies de programmation
Algorithmes gloutons

Exemple (Codage de Huffman)

Exemple (Codage de Huffman)



Bilan/Limites

- ▶ Souvent faciles à comprendre et à implémenter
- ▶ Résultat progressif
- ▶ Peut parfois conduire à la pire solution (voir Voyageur de commerce) car ne considère pas le problème dans sa globalité
- ▶ Mêmes limites que les approches récursives (si implémenté en récursif)