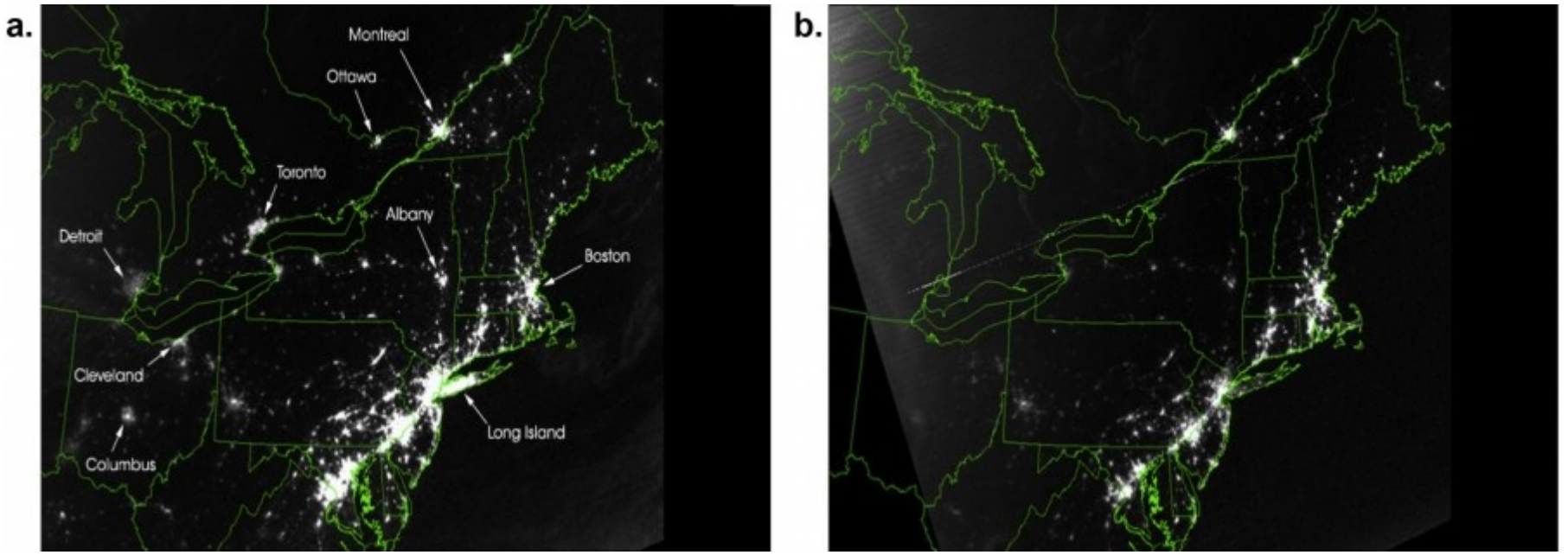
Université de Paris

Algorithmie Avancée

Mise en Contexte / Mise en Oeuvre

Année 2020-2021 par Prof. Nicolas Loménie
Sur la base du cours de Prof. Etienne Birmelé (2016-2020)

Vulnérabilité et Inter-connectivité

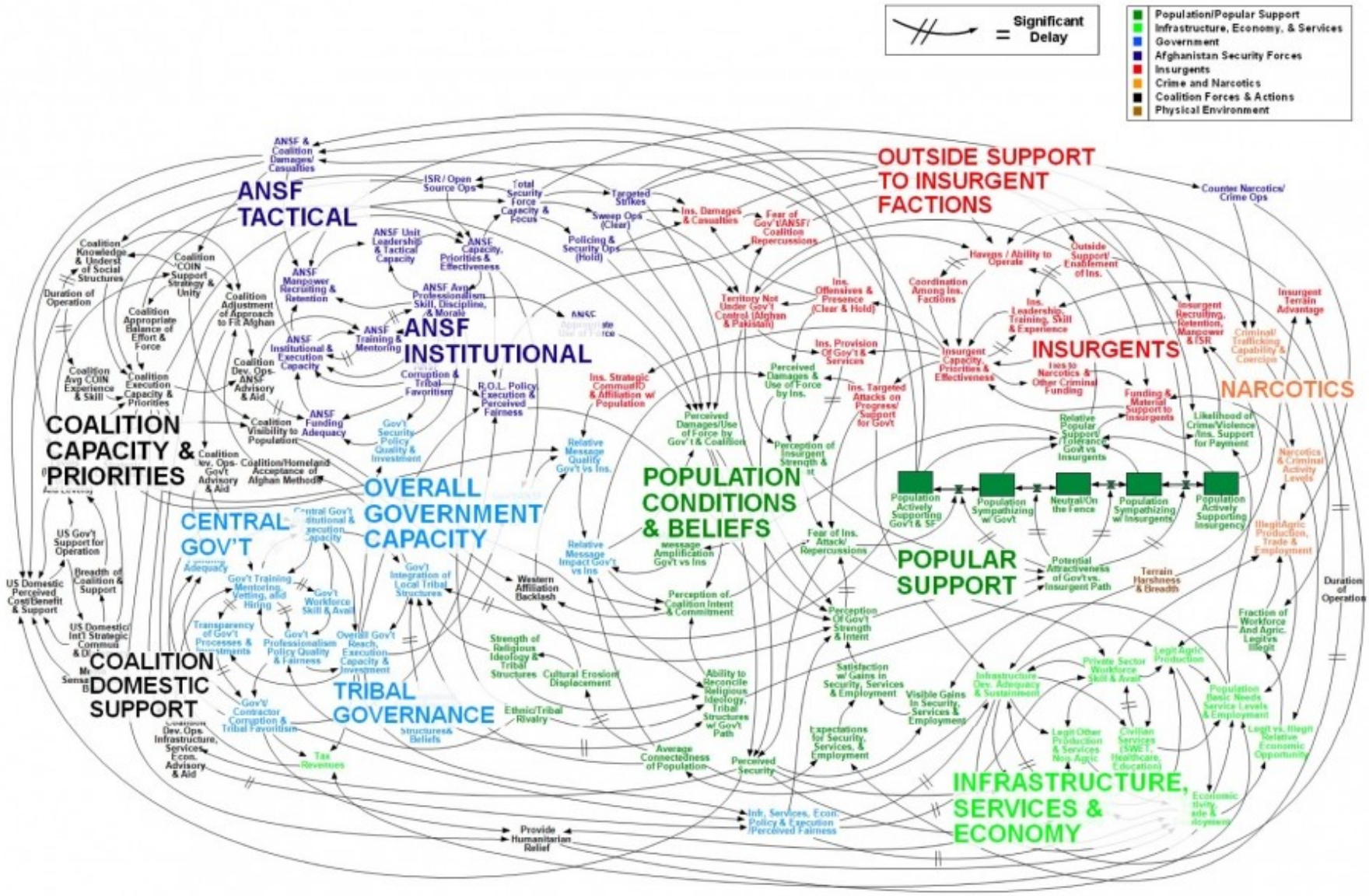


2003 North American Blackout
Satellite image on Northeast United States on August 13th, 2003,at 9:29pm (EDT),
a. 20 hours before the 2003 blackout.
b. The same as above, but 5 hours after the blackout.

https://www.quantmetry.com/blog/tout-est-graphe-detection-de-communautes-theorie-et-retour-dexperience/

http://networksciencebook.com/chapter/1#networks

Vulnérabilité et Inter-connectivité



Networks at the Heart of Complex Systems

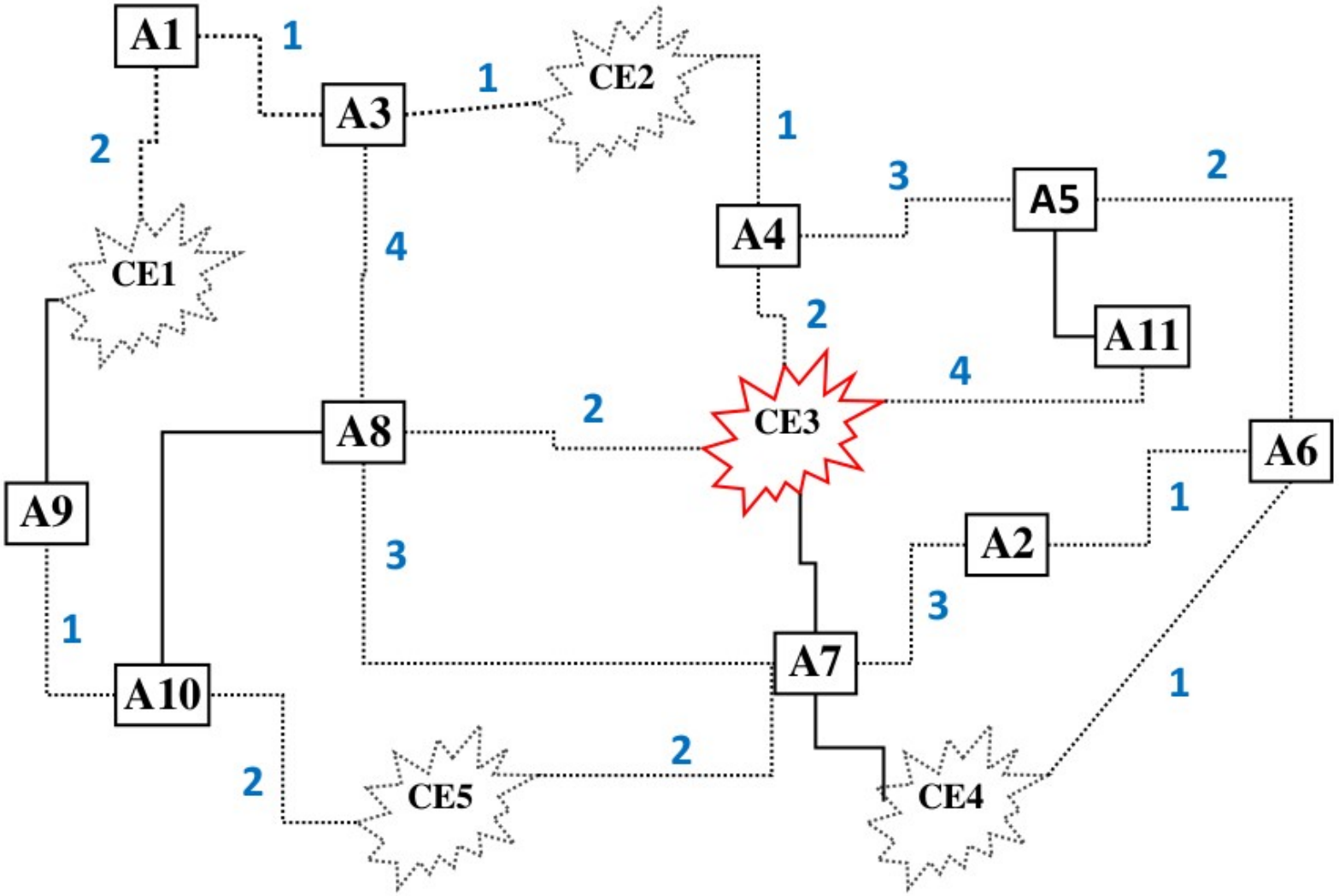
"I think the next century will be the century of complexity."

Stephen Hawking

Situation Analysis Afghanistan 2010

http://mybusinessagility.com/vuca/

Vulnérabilité et Inter-connectivité



Centrale Électrique, Cyclone, Agglomération, 1 seule équipe de techniciens pour réparer
Application des ACPM / MST

Rappel cours introduction

Algorithmie : résolution de problèmes via structures de données

Pas de livre magique mais des stratégies universelles :

- **la méthode incrémentale** : résoudre un problème P(n) à partir d'une solution de P(n-1) ; une méthode par récurrence, qui peut donner lieu aussi bien à des programmes itératifs qu'à des programmes récursifs ; dans le cas d'un problème d'optimisation, la méthode gloutonne consiste à construire une solution de P(n) en prolongeant une solution de P(n-1) par un choix localement optimal ;
- **la méthode << diviser pour régner >>** : méthode descendante par décomposition en sous-problèmes en transformant un problème P(n) en deux problèmes P(n/2) **(un O(n log n) apparaît souvent alors en terme de complexité voir master theorem Landau https://fr.wikipedia.org/wiki/Master_theorem);**
- **la programmation dynamique** : méthode ascendante, utilisable pour des problèmes d'optimisation, qui consiste à construire la solution d'un problème à partir des solutions de tous les sous-problèmes ; elle s'applique quand toute sous-solution d'une solution optimale est optimale pour le sous-problème correspondant.

Mais le coeur du problème c'est la **Structure de Données (Data Structure)** : la modélisation du problème (monoïde, graphe, matroïde en math / pile, file, tas, table, arbre en info → math discrète)

Peut-on tout résoudre par un algorithme ? P=NP ? En un temps raisonnable ? Le fameux problème du Voyageur de Commerce

https://www.youtube.com/watch?v=5NJCEDusvB4

Programmation Dynamique

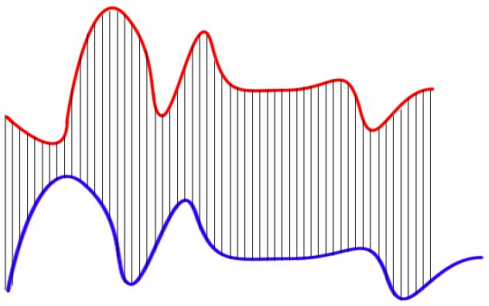
Dynamic programming matrix:

i		j →	0	1 T	2 G	3 C	4 T	5 C	6 G	7 T	8 = N A
			0	−6	−12	−18	−24	−30	−36	−42	−48
1	T		−6	5	−1	−7	−13	−19	−25	−31	−37
2	T		−12	−1	3	−3	−2	−8	−14	−20	−26
3	C		−18	−7	−3	8	2	3	−3	−9	−15
4	A		−24	−13	−9	2	6	0	1	−5	−4
5	T		−30	−19	−15	−4	7	4	−2	6	0
M = 6	A		−36	−25	−21	−10	1	5	2	0	11

(sequence **x**) ↓

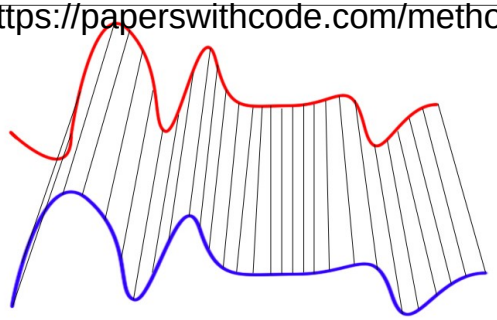
Optimum alignment scores 11:

T	**–**	**–**	**T**	**C**	**A**	**T**	**A**
T	**G**	**C**	**T**	**C**	**G**	**T**	**A**
+5	−6	−6	+5	+5	−2	+5	+5

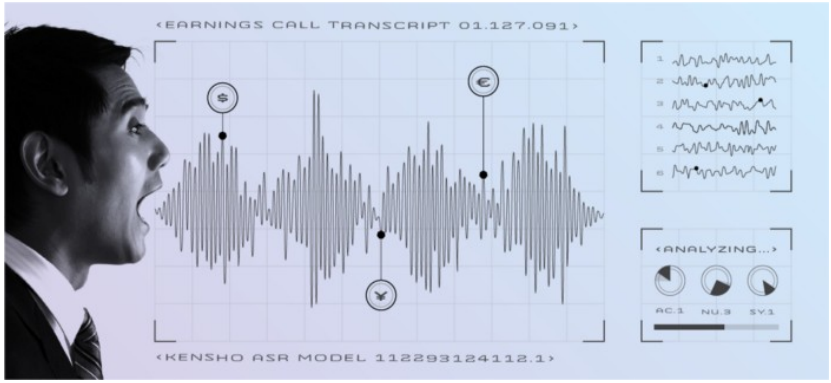


Euclidean Matching

https://paperswithcode.com/method/dtw



Dynamic Time Warping Matching

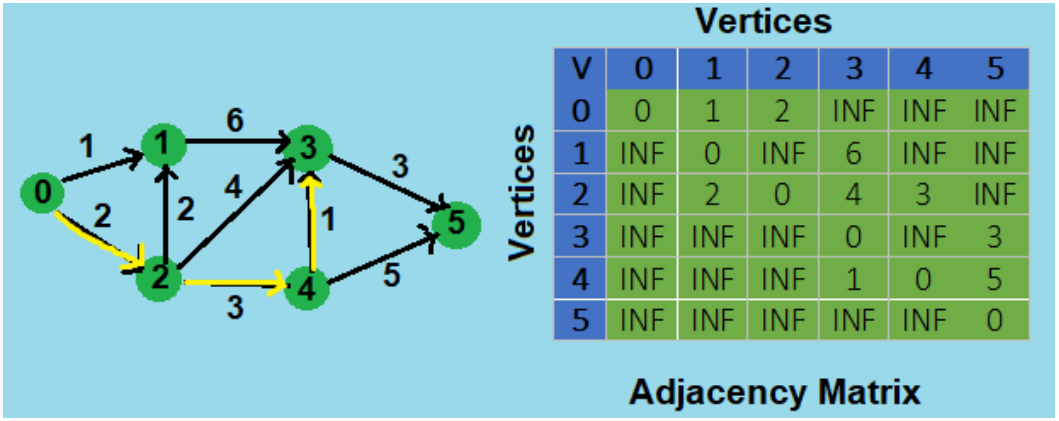


https://blog.kensho.com/speech-recognition-for-finance-86983b2b97bd

https://avikdas.com/2019/04/15/a-graphical-introduction-to-dynamic-programming.html

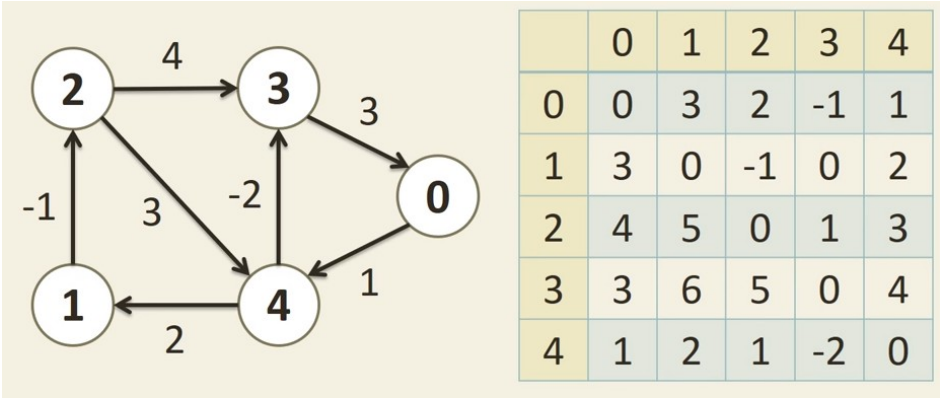
https://www.nature.com/articles/nbt0704-909

Programmation Dynamique



V	0	1	2	3	4	5
0	0	1	2	INF	INF	INF
1	INF	0	INF	6	INF	INF
2	INF	2	0	4	3	INF
3	INF	INF	INF	0	INF	3
4	INF	INF	INF	1	0	5
5	INF	INF	INF	INF	INF	0

la programmation dynamique : méthode ascendante, utilisable pour des problèmes d'optimisation, qui consiste à construire la solution d'un problème à partir des solutions de tous les sous-problèmes ; elle s'applique quand toute sous-solution d'une solution optimale est optimale pour le sous-problème correspondant.



	0	1	2	3	4
0	0	3	2	-1	1
1	3	0	-1	0	2
2	4	5	0	1	3
3	3	6	5	0	4
4	1	2	1	-2	0

https://www.algotree.org/algorithms/floyd-warshall/ All shortest paths algo
http://www14.in.tum.de/lehre/2018WS/ada/index.html.en
https://iq.opengenus.org/shortest-path-with-k-edges/
https://www.youtube.com/watch?v=DYqbevmCrZM
https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm

Théorie des Graphes 6

- AlgoAvanceeParE_Birmele.pdf

  Support de cours de Prof. Etienne Birmelé

  Planche 105 à 120 (Fin Arbre Couvrant – Début Cycle Couvrant)

https://brilliant.org/wiki/floyd-warshall-algorithm

Mise En Oeuvre

Liste, Pile, File, AVL

Concaténation de listes : efficace en O(1) vs. fusion de tableaux

Pile système pour appel de procédures :
- Pile de récursion
- Pile d'évaluation d'expression arithmétique

File système pour gérer les processus en attente :
- avec priorité ; File de priorité pour système de réservation, gestion de pistes d'aéroport…

AVL voir cours précédent

Nécessité de modèle pour être réutilisé en fonction des applications :
Types abstraits, classes d'interface etc. notamment en POO

https://www.enseignement.polytechnique.fr/profs/informatique/Jean-Eric.Pin/PDF/Amphi9.pdf
http://emmanuel.adam.free.fr/site/spip.php?article130
http://emmanuel.adam.free.fr/site/spip.php?article128

Mise En Oeuvre

Classes existantes en orienté objet : C++, Java ?
Par exemple, la ***Class ArrayList***, avec un **itérateur** qui correspond à l'indice entier du tableau.

Sinon il faut définir une classe qui implémente l'interface ***Iterable*** avec des méthodes ***add(), head(),remove(), isEmpty(),contains(), size()*** *et* possède une **Interface** de type ***Iterator*** c'est-à-dire des méthodes ***next()***, ***hasNext()***

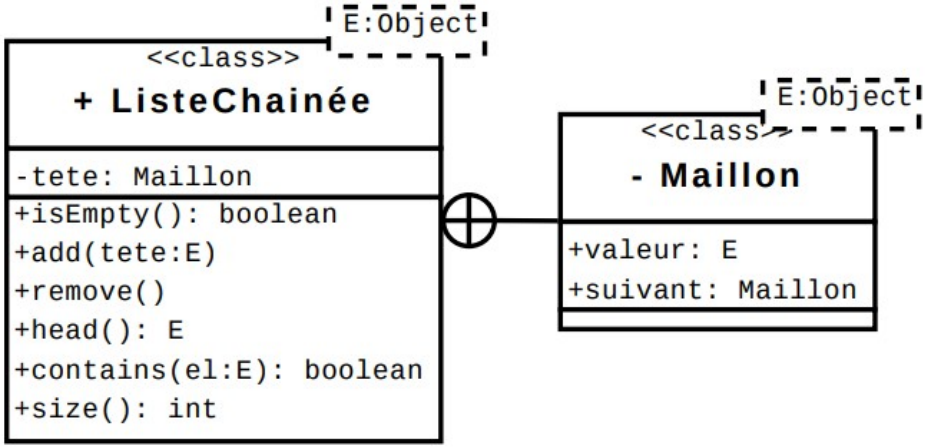
```java
import java.lang.Iterable;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * Début de la classe qui permet la représentation
 * de listes sous forme de listes chaînées.
 * L'interface "Iterable" spécifie que la liste dispose
 * d'une méthode iterator().
 */
public class ListeChaînée<E> implements Iterable<E> {

  (...)

  // Methode de la classe "ListeChaînée" permettant de
  // récupérer un itérateur.
  public Iterator<E> iterator(){
        return new ListeChaînéeIterator(this);
  }

  /**
   * Classe d'itérateurs dédiée au type de liste
   * "ListeChaînée".
   */
  private class ListeChaînéeIterator implements Iterator<E>{
```




```java
public E next() throws NoSuchElementException {
  // S'il n'y a pas d'élément suivant, l'utilisateur
  // n'a pas le droit d'utiliser la méthode next().
  // On lève donc une exception.
  if(! this.hasNext()){
    throw new NoSuchElementException();
  } else {
```

Heureusement, https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html depuis 1.2
https://riptutorial.com/fr/java/example/14754/linkedlist-en-tant-que-file-fifo : La classe java.util.LinkedList , lors de l'implémentation de java.util.List est une implémentation polyvalente de l'interface java.util.Queue fonctionnant également selon le principe FIFO (First In, First Out) .

Mise En Oeuvre

Classes existantes en orienté objet : C++, Java ?

Versions SDK < Java 1.2 :
Array, **Vector** (implémente **List** maintenant), **Stack**, **Hashtable** (implémente **Map** maintenant), **Properties** et **BitSet** avec l'interface **Enumeration**.

Puis **API Collections** : socle riche d' implémentations d'objets de type collection enrichies au fur et à mesure des versions du SDK . Deux grandes familles à mettre en rapport avec les structures en Python :

java.util.Collection : pour gérer un groupe d'objets
java.util.Map : pour gérer des éléments de type paires de clé/valeur

Maintenant, pour faciliter le parcours des **Collections** et leur tri : interfaces **Iterator**, **ListIterator**, **Comparable**, **Comparator** (depuis version 2 du JDK)

Puis gestion des accès concurrents pour :

- **List**
- **Set**
- **Map**
- **Queue**

https://www.jmdoudoux.fr/java/dej/chap-collections.htm

java.util

Class LinkedList<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.AbstractSequentialList<E>
        java.util.LinkedList<E>

Type Parameters:

E - the type of elements held in this collection

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>

```
public class LinkedList<E>
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

Mise En Oeuvre

Classes existantes en orienté objet : C++, Java ?

Versions SDK > Java 5 :

	Utilisation générale	Utilisation spécifique	Gestion des accès concurrents
List	ArrayList LinkedList	CopyOnWriteArrayList	Vector Stack CopyOnWriteArrayList
Set	HashSet TreeSet LinkedHashSet	CopyOnWriteArraySet EnumSet	CopyOnWriteArraySet ConcurrentSkipListSet
Map	HashMap TreeMap LinkedHashMap	WeakHashMap IdentityHashMap EnumMap	Hashtable ConcurrentHashMap ConcurrentSkipListMap
Queue	LinkedList ArrayDeque PriorityQueue		ConcurrentLinkedQueue LinkedBlockingQueue ArrayBlockingQueue PriorityBlockingQueue DelayQueue SynchronousQueue LinkedBlockingDeque

org.apache.commons.collections

Class BinaryHeap

```
java.lang.Object
  |
  +--java.util.AbstractCollection
        |
        +--org.apache.commons.collections.BinaryHeap
```

All Implemented Interfaces:

Buffer, java.util.Collection, PriorityQueue

https://commons.apache.org/proper/commons-collections/javadocs/api-2.1.1/org/apache/commons/collections/BinaryHeap.html

https://www.geeksforgeeks.org/min-heap-in-java/

https://www.sanfoundry.com/java-program-implement-binomial-heap/

Mise En Oeuvre

Classes existantes en orienté objet : C++, Java ?

On en est à JDK 8 (2020). Par exemple pour les tas,

Min heap:

```
PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>();
```

Max heap:

```
PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>(new Comparator<Integer>
    @Override
    public int compare(Integer o1, Integer o2) {
        return - Integer.compare(o1, o2);
    }
});
```

Ou plus élégamment :

```
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Comparator.reverseOrder());
```

org.apache.commons.collections

Interface PriorityQueue

All Known Implementing Classes:
BinaryHeap, SynchronizedPriorityQueue

A priori, à ce jour (2020), pas d'implémentation native de Tas binomiaux ou de Fibonacci pour améliorer les complexités.
A faire soi-même.
Mais ***TreeSet*** pas mal sans doute.
A tester soi-même.

java.util

Class TreeSet<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractSet<E>
      java.util.TreeSet<E>

Type Parameters:

E - the type of elements maintained by this set

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>

En C++, classes ***unordered_map, unordered_set*** et ***pair***
pour représenter les listes de sommets, sommets marqués, paires clé-valeur.

http://www.cplusplus.com/reference/queue/queue/(classe ***Queue*** pour File FIFO et classe ***Stack*** pour Pile LIFO)

http://www.cplusplus.com/reference/queue/priority_queue/