Algorithmique et Programmation
Niveau 2

Frédéric Goualard

Laboratoire des Sciences du Numérique de Nantes, UMR CNRS 6004
Office #112-11

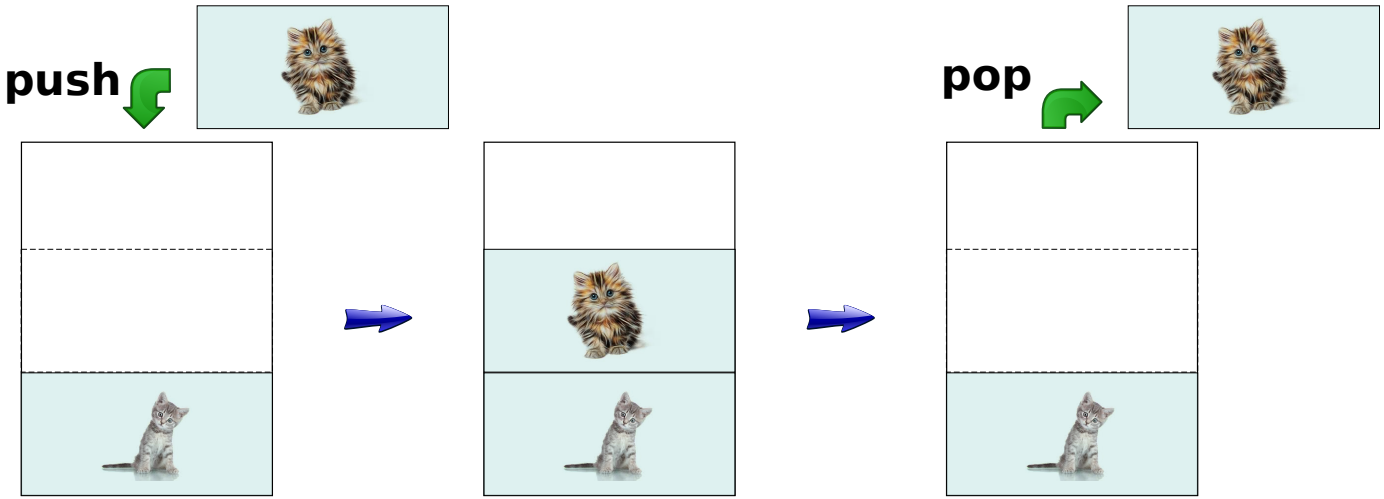
Les types abstraits et leurs implémentations

Types abstraits

- Liste chaînée, tableau, chaîne de caractères : types concrets
- Type abstrait : on connaît l'interface d'utilisation, pas l'implémentation
- Implémentation possible d'un type abstrait en terme de plusieurs types concrets *sans changer l'interface*

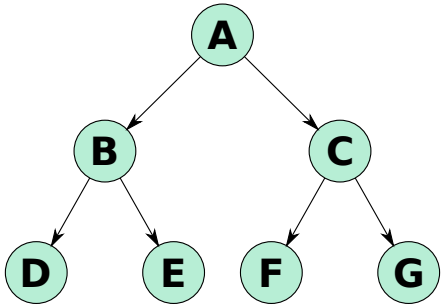
Les piles

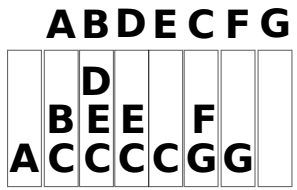
- *Last-in-First-out* (LIFO)
- Interface :
  - `push(pile,cat)`
  - `cat = pop(pile)`
  - `isempty(pile)`
  - `[cat = top(pile)]`



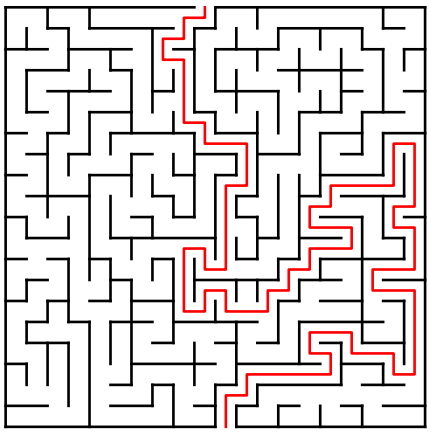
Utilisation des piles

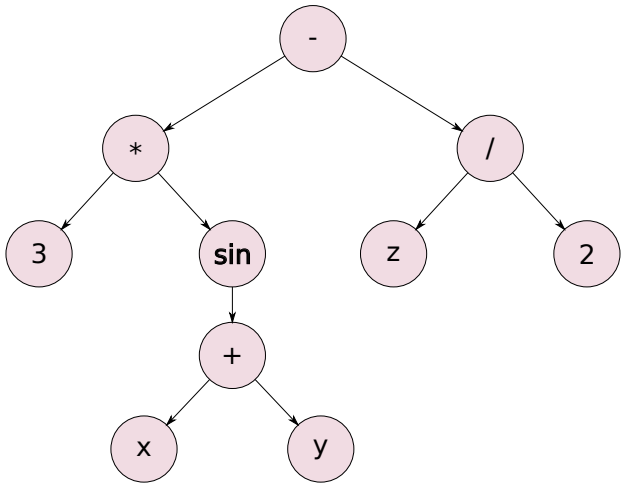
- Algorithmes avec *backtracking* (*branch and bound*)
- Recherche en profondeur d'abord

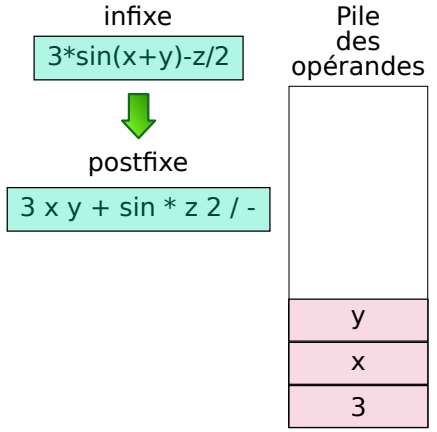




- Évaluation d'expressions arithmétiques
- ...







Implémentation possible des piles

```
#include <iostream>
#include <exception>
#include <string>
using namespace std;

using cat_t = string;
cat_t lolcat {"lolcat"};

struct catstack_t {
  cat_t *stack;
  size_t availtop; // 1ere place libre
  size_t maxsize;
};

catstack_t create_catstack(size_t maxsize)
{
  catstack_t cs;
  cs.stack = new cat_t[maxsize];
  cs.maxsize = maxsize;
  cs.availtop = 0;
  return cs;
}

void delete_catstack(catstack_t& pile)
{
  delete[] pile.stack;
}

bool isempty(const catstack_t& pile)
{
  return pile.availtop == 0;
}
```

```
bool push(catstack_t& pile, cat_t c)
{
  if (pile.availtop < pile.maxsize) {
        pile.stack[pile.availtop++] = c;
        return true;
  } else {
        return false;
  }
}

cat_t pop(catstack_t& pile)
{
  if (isempty(pile)) {
        throw logic_error("pop sur pile vide!");
        return lolcat;
  } else {
        cat_t c = pile.stack[--pile.availtop];
        return c;
  }
}

int main(void)
{
  catstack_t stack_of_cats { create_catstack(100) };
  push(stack_of_cats,"max");
  push(stack_of_cats,"tigger");
  push(stack_of_cats,"misty");
  push(stack_of_cats,"oscar");

  while (!isempty(stack_of_cats)) {
        cat_t c = pop(stack_of_cats);
        cout << c << "\n";
  }
  delete_catstack(stack_of_cats);
}
```

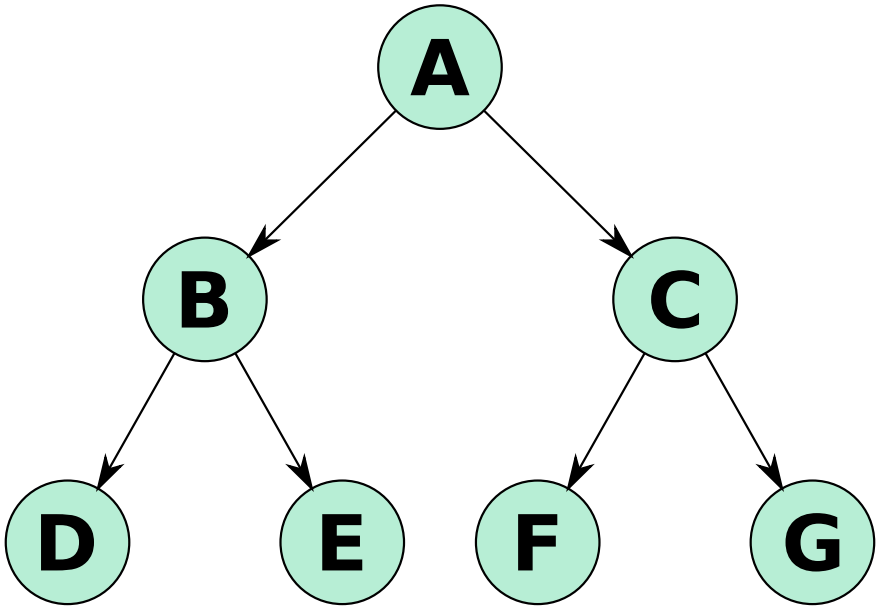
Disponible dans la STL : « stack »

Les files (ou queues)

- First-In-First-Out (FIFO)
- Interface :
  - `put(queue,courrier)`
  - `courrier=get(queue)`
  - `isempty(queue)`

Utilisation des files

- Recherche en *largeur d'abord* dans des arbres



A	
CB	A
EDC	B
GFED	C
GFE	D
GF	E
G	F
	G

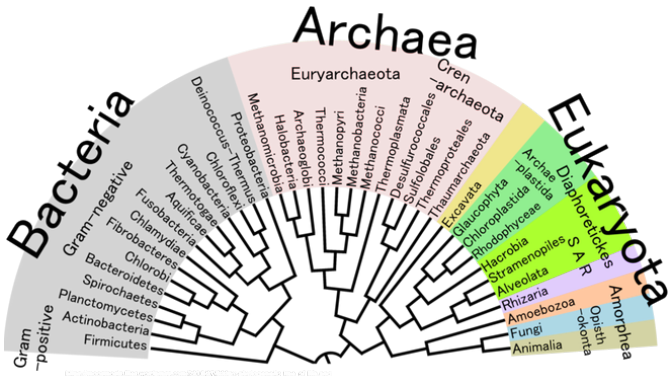
- Gestion équitable de requêtes
- Simulation d'évènements physiques en préservant la temporalité

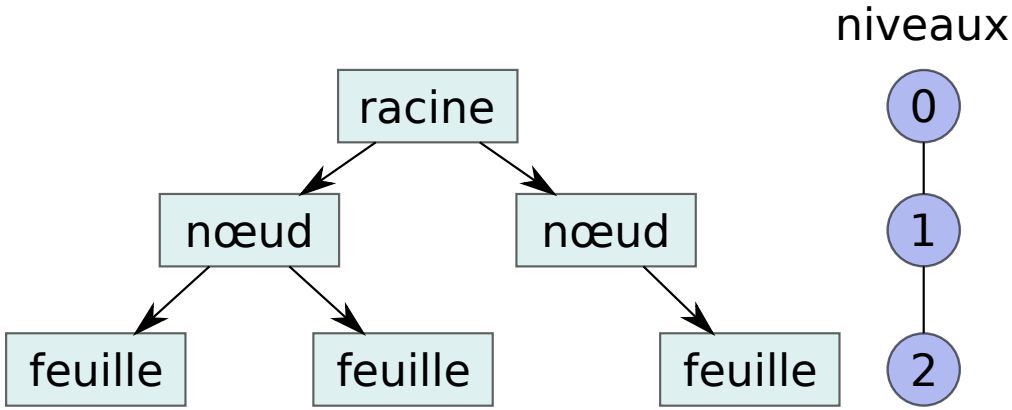
Implémentation possible des files

```
#include <iostream>
#include <exception>
#include <string>
using namespace std;

using courrier_t = string;
courrier_t courriervide  { "vide" };

struct queue_t {
  courrier_t *queue;
  size_t tail; // 1ere place libre
  size_t maxsize;
};

queue_t create_queue(size_t maxsize)
{
  queue_t cq;
  cq.queue = new courrier_t[maxsize];
  cq.maxsize = maxsize;
  cq.tail = 0;
  return cq;
}

void delete_queue(queue_t& file)
{
  delete[] file.queue;
}

bool isempty(const queue_t& file)
{
  return file.tail == 0;
}

bool put(queue_t& file, const courrier_t& c)
{
  if (file.tail < file.maxsize) {
        file.queue[file.tail++] = c;
        return true;
  } else {
        return false;
  }
}

courrier_t get(queue_t& file)
{
  if (isempty(file)) {
        throw logic_error("get sur file vide!");
        return courriervide;
  } else {
        courrier_t c = file.queue[0];
        for (size_t i = 1; i < file.tail; ++i) {
              file.queue[i-1] = file.queue[i];
        }
        --file.tail;
        return c;
  }
}

int main(void)
{
  queue_t queue_of_courriers { create_queue(100) };
  put(queue_of_courriers,"Prunelle");
  put(queue_of_courriers,"De Mesmaeker");
  put(queue_of_courriers,"Longtarin");
  put(queue_of_courriers,"Fantasio");

  while (!isempty(queue_of_courriers)) {
        courrier_t c = get(queue_of_courriers);
        cout << c << "\n";
  }
  delete_queue(queue_of_courriers);
}
```

Les arbres

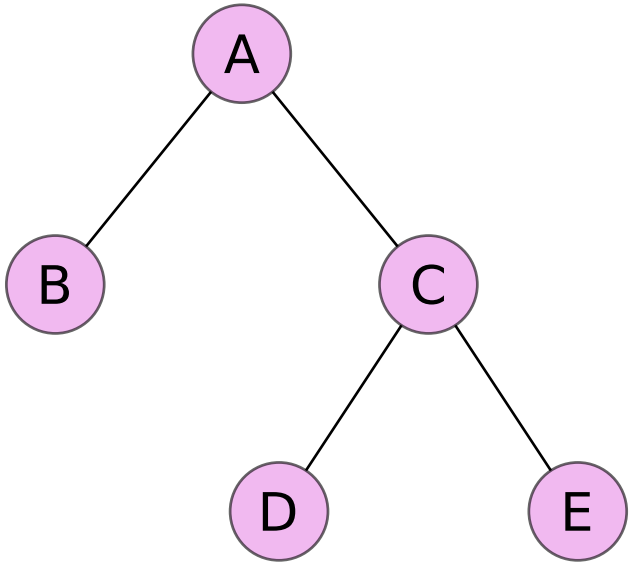
- Représentation de données hiérarchisées





- Arbres binaires, *n*-aires
- Définition récursive (« *un arbre est vide ou composé d'un nœud racine et d'une liste de sous-arbres fils* ») ⇒ *algorithmes de traitement récursifs*

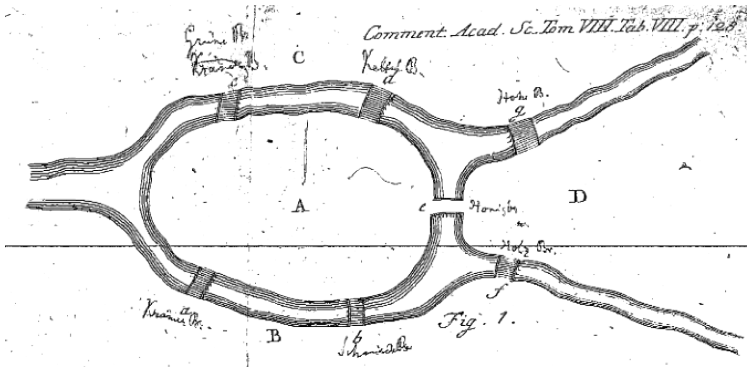
Exploration des arbres

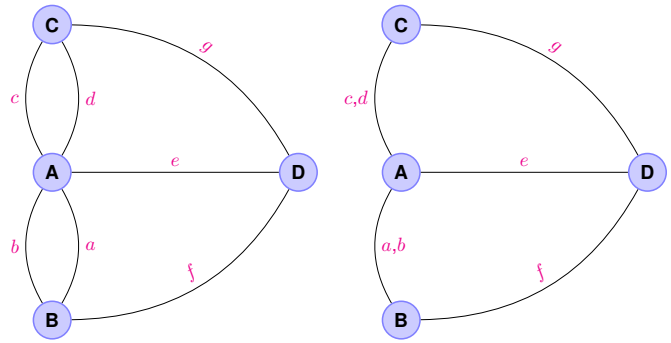


- Parcours préfixe : **A B C D E**
- Parcours infixe : **B A D C E**
- Parcours postfixe : **B D E C A**

Les graphes

1736 : les sept ponts de Königsberg — Euler





« Peut-on trouver un chemin traversant chaque pont une seule fois ? »

- Graphe (V, E) :
  - V : ensemble de nœuds (**A**,**B**,**C**,**D**)
  - E : ensemble d'arcs entre les nœuds de V (a, b, c, d, e, f, g)
- Graphes *complets*, *planaires*, *orientés*, *non orientés*
- Nombreuses applications (TSP, routage circuits électroniques, reconstruction d'ADN, ...)
- Recherche d'un chemin eulérien : problème dans **P** ($\mathcal{O}(|V| + |E|)$)

Application des graphes à la biologie

Shortest Superstring Problem (d'après http://www.bioalgorithms.info)

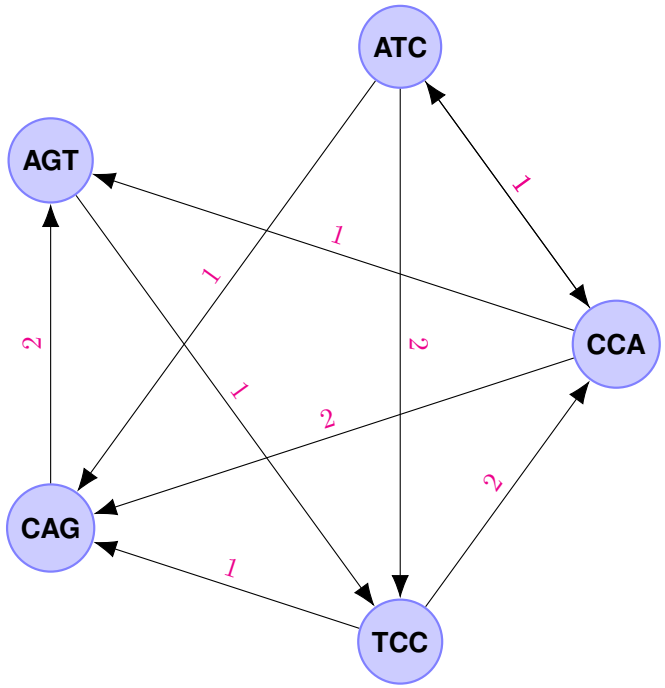
- $S = \{\mathbf{ATC}, \mathbf{AGT}, \mathbf{CAG}, \mathbf{TCC}, \mathbf{CCA}\}$
- Plus petite chaîne contenant tous les éléments de S ?
- $\texttt{overlap}(\mathrm{ATC}, \mathrm{TCC}) = 2$

  ATC
  TCC

Application des graphes à la biologie

Shortest Superstring Problem (d'après http://www.bioalgorithms.info**)**

- $S = \{\mathbf{ATC}, \mathbf{AGT}, \mathbf{CAG}, \mathbf{TCC}, \mathbf{CCA}\}$
- Plus petite chaîne contenant tous les éléments de S ?
- $\texttt{overlap}(\mathrm{ATC}, \mathrm{TCC}) = 2$
  ATC
    TCC

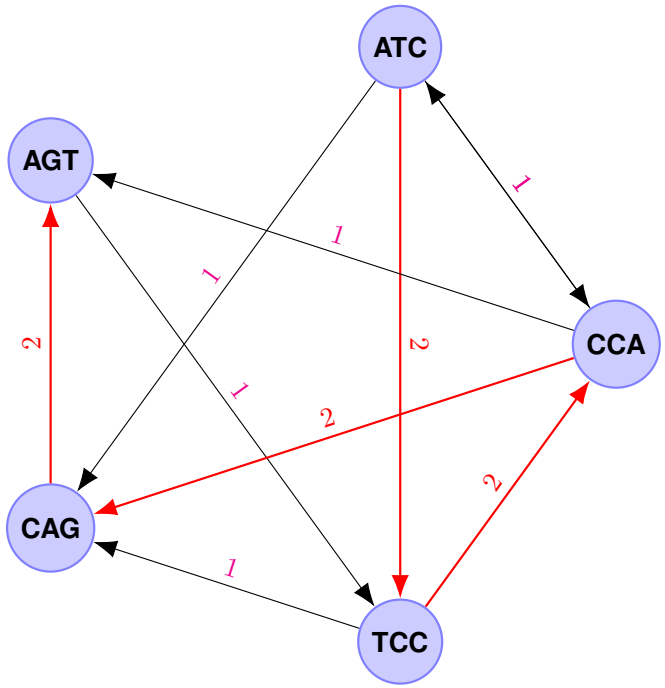


Application des graphes à la biologie

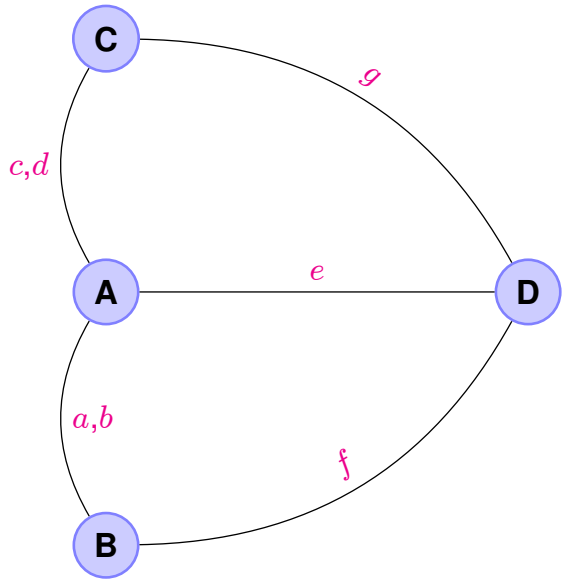
Shortest Superstring Problem (d'après http://www.bioalgorithms.info**)**

- $S = \{\mathbf{ATC}, \mathbf{AGT}, \mathbf{CAG}, \mathbf{TCC}, \mathbf{CCA}\}$
- Plus petite chaîne contenant tous les éléments de S ?
- $\texttt{overlap}(\text{ATC}, \text{TCC}) = 2$
  ATC
  TCC

- *Chemin Hamiltonien de valeur max.*
  ATC
  TCC
  CCA
  CAG
  AGT
  Solution : ATCCAGT
- Problème NP-dur



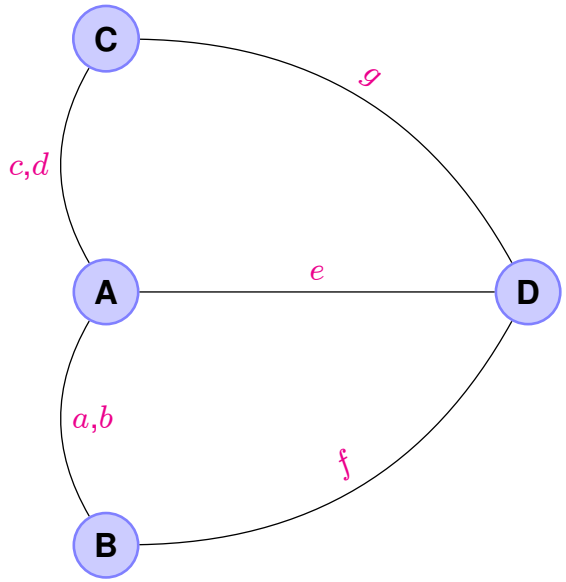
Implémentation possible des graphes



	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

Matrice d'adjacence

Implémentation possible des graphes

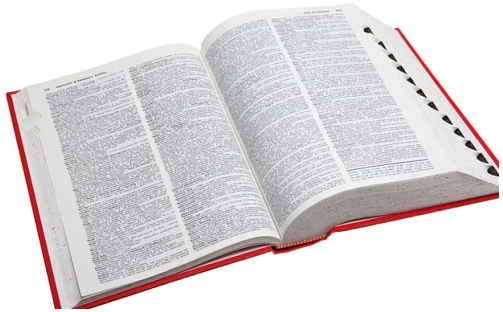


	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

Matrice d'adjacence

- Représentation par un tableau à 2 dimensions :
  - Coûteux si la matrice est symétrique
  - Très coûteux si le graphe est très peu dense
- Solution ? Utilisation de listes chaînées

Les tables de hachage (exemples)



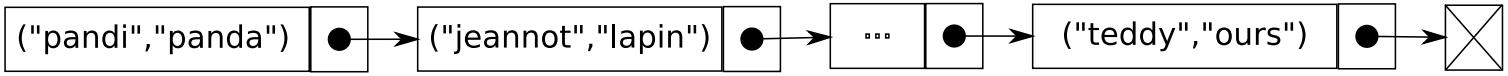
Utilisation :

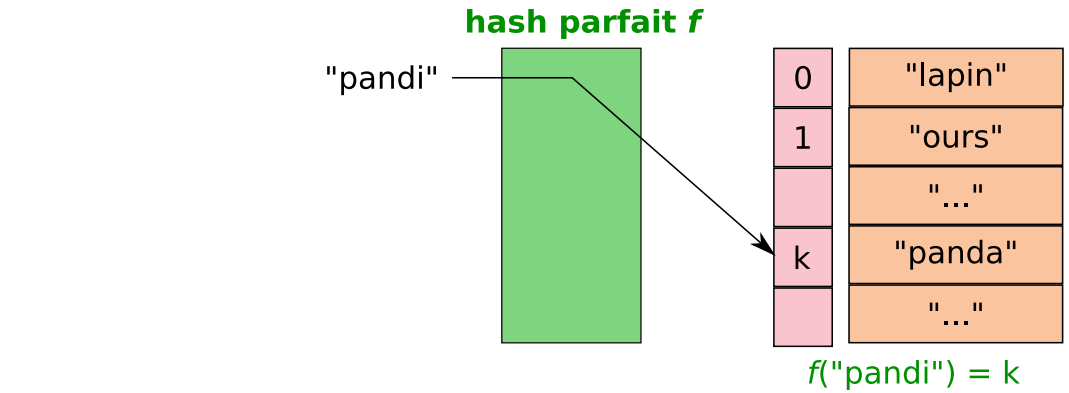
- Tableaux associatifs
- Ensembles
- ...

Les tables de hachage

zoo["pandi"] = "panda"
zoo["jeannot"] = "lapin"
...
zoo["teddy"] = "ours"

zoo["jeannot"] ?





Fin du cours