ASD 2 - Labo 5

Mini-projet

Correcteur Orthographique

Entrées / Sorties

- **Les données d'entrée**
  - Un dictionnaire anglais (fourni)
  - Le fichier textuel à corriger (certains sont fournis)
- **En sortie**
  - Un fichier mentionnant tous les mots mal orthographiés et les suggestions trouvées à l'aide du dictionnaire
  - Le temps de chargement du dictionnaire en mémoire
  - Le temps d'exécution nécessaire pour corriger un texte

Marche à suivre - Dictionnaire

- **Lire, nettoyer et stocker le fichier `dictionary.txt` dans la structure de données**

- **Lire mot par mot le fichier texte en entrée**
  - Convertir en minuscule toutes les lettres de chaque mot
  - Garder uniquement les caractères a-z et les apostrophes en milieu de mot

Marche à suivre - Correction

- **Vérifier l'existence de chaque mot dans le dictionnaire**
  - Présent → rien à faire
  - Absent → générer toutes les variantes possibles du mot en se basant sur quatre hypothèses
  - Vérifier l'existence de chaque variante dans le dictionnaire
    - Si la variante existe dans le dictionnaire → elle sera proposée comme correction possible
- **Afficher les mots mal orthographiés avec les suggestions dans un fichier, gardez l'ordre des mots du fichier initial**

Variantes orthographiques

1. L'utilisateur a tapé une lettre supplémentaire
   acqueux → aqueux (il y a un c en trop)
2. L'utilisateur a oublié de taper une lettre
   aqeux → aqueux (il manque la lettre u)
3. L'utilisateur a mal tapé une lettre
   awueux → aqueux (il y a un w à la place du q)
4. L'utilisateur a échangé 2 lettres (**consécutives**)
   auqeux → aqueux (u et q intervertis)

Format fichier de sortie

- Pour chaque texte pour lequel vous vérifierez l'orthographe, vous générerez un fichier texte comportant
  - les mots mal orthographiés (**préfixés d'une étoile**)
  - suivi immédiatement des propositions de correction vérifiées (**préfixées du numéro de l'hypothèse**)

```
*lates
1:ates
1:lats
1:late
2:alates
2:elates
2:plates
…
3:bates
3:cates
3:dates
…
4:altes
```

Application: correction + suggestions orthographiques

dictionary.txt

```
afterward
among
analog
apologize
behavior
catalog
center
color
color's
colors
defense
favor
favorite
flavor
gray
judgment
labeled
labeling
Labor
learned
…
```

+

input_sh.txt

```
Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever.  You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.net


Title: The Adventures of Sherlock Holmes

Author: Arthur Conan Doyle

Posting Date: April 18, 2011 [EBook #1661]
First Posted: November 29, 2002

Language: English

…
```

=

output.txt

```
*gutenberg's
2:guttenberg's
2:guttenberg's
*boscombe
*trepoff
*eglow
1:glow
2:reglow
3:aglow
3:eglon
*eglonitz
*egria
1:eria
2:egeria
3:agria
3:egrid
*kramm
1:kram
1:kram
3:dramm
3:krama
3:krams
…
```

Stockage du dictionnaire

- **2 versions demandées**

  - Avec un container STL de votre choix
    A justifier

  - Avec un *Ternary Search Trie*
    Classe à mettre en œuvre par vous même

Ternary Search Tries?

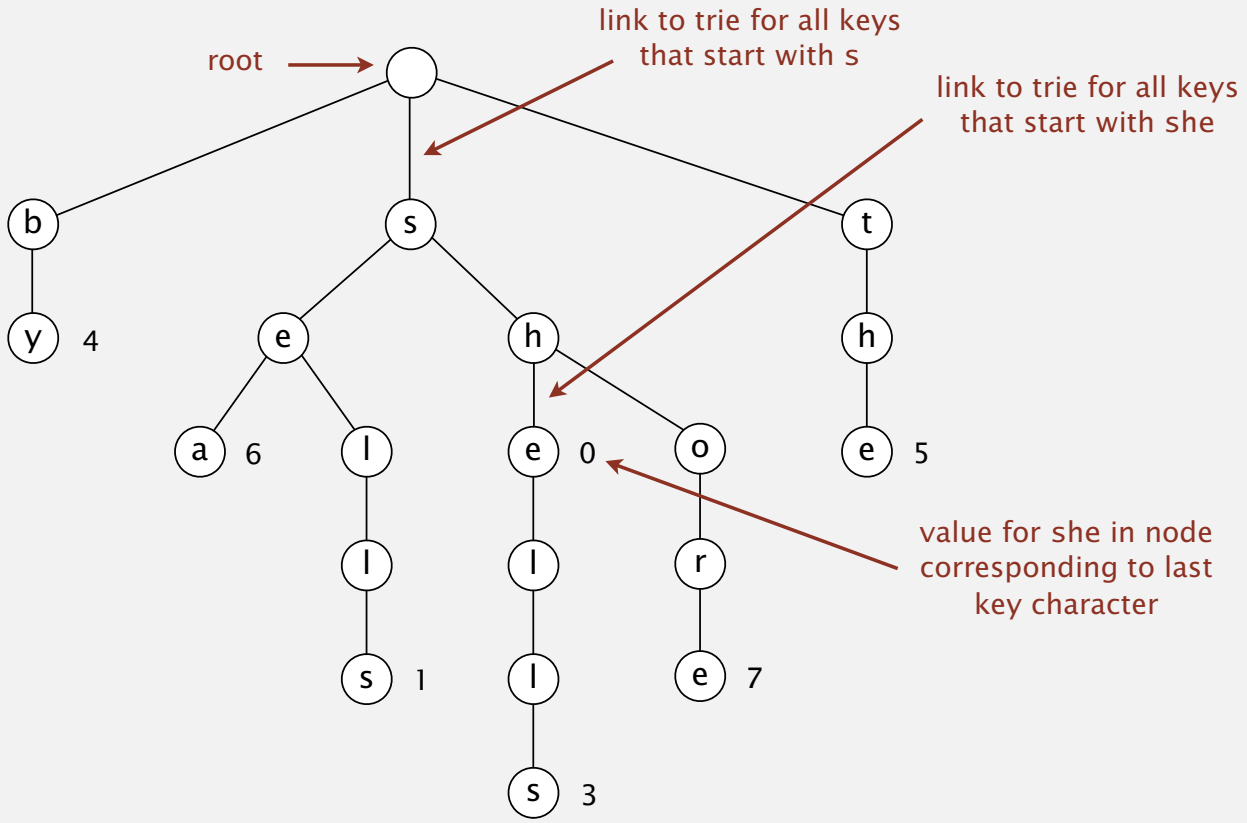
- **Structure de données apparentée aux arbres de recherche**
- **Brève introduction ici, et diverses ressources externes proposées**
- **Classe *TernarySearchTrie* à mettre en œuvre et utiliser**

Tries

Tries. [from retrieval, but pronounced "try"]

- Store characters in nodes (not keys).
- Each node has R children, one for each possible character.
- For now, we do not draw null links.

for now we do not draw null links

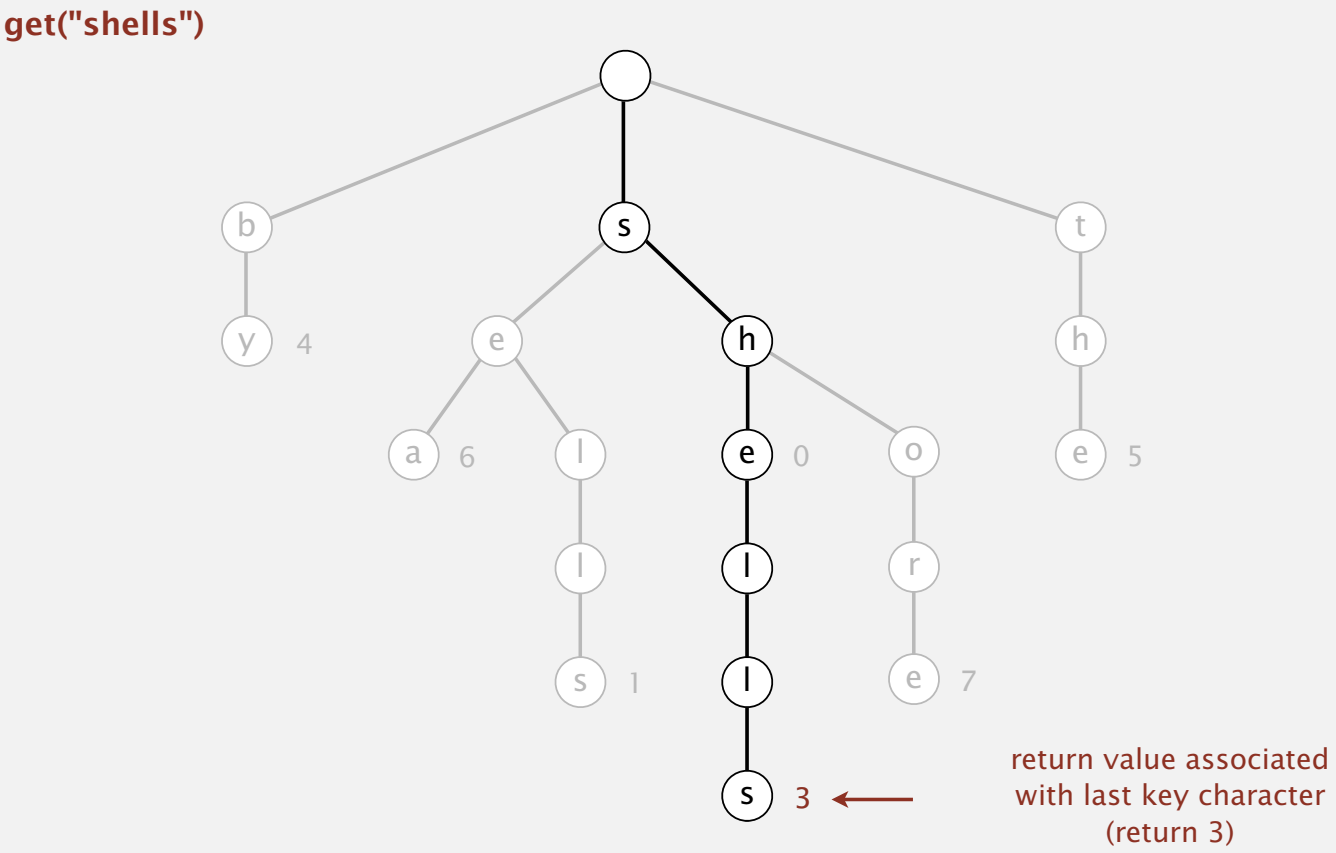


key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5

Search in a trie

Follow links corresponding to each character in the key.

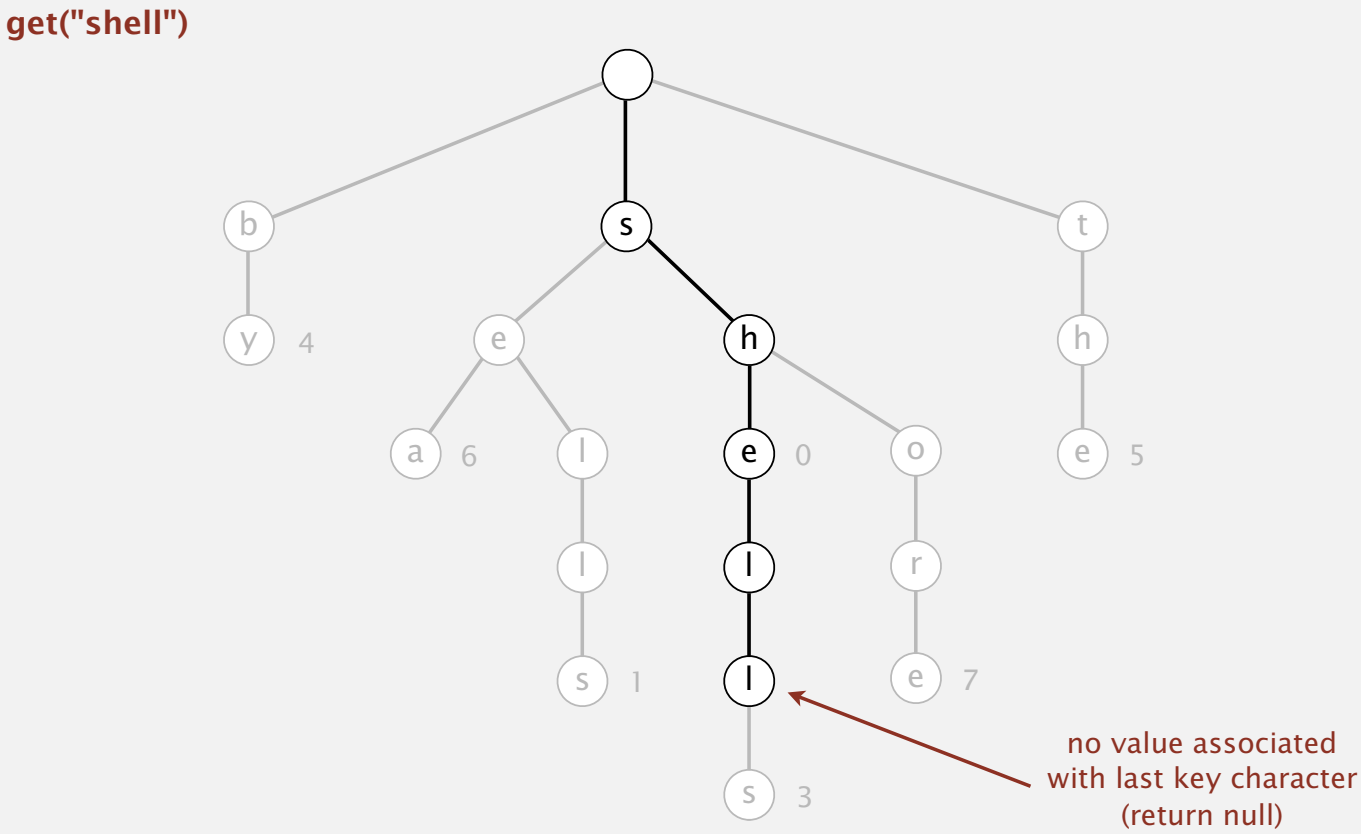
- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

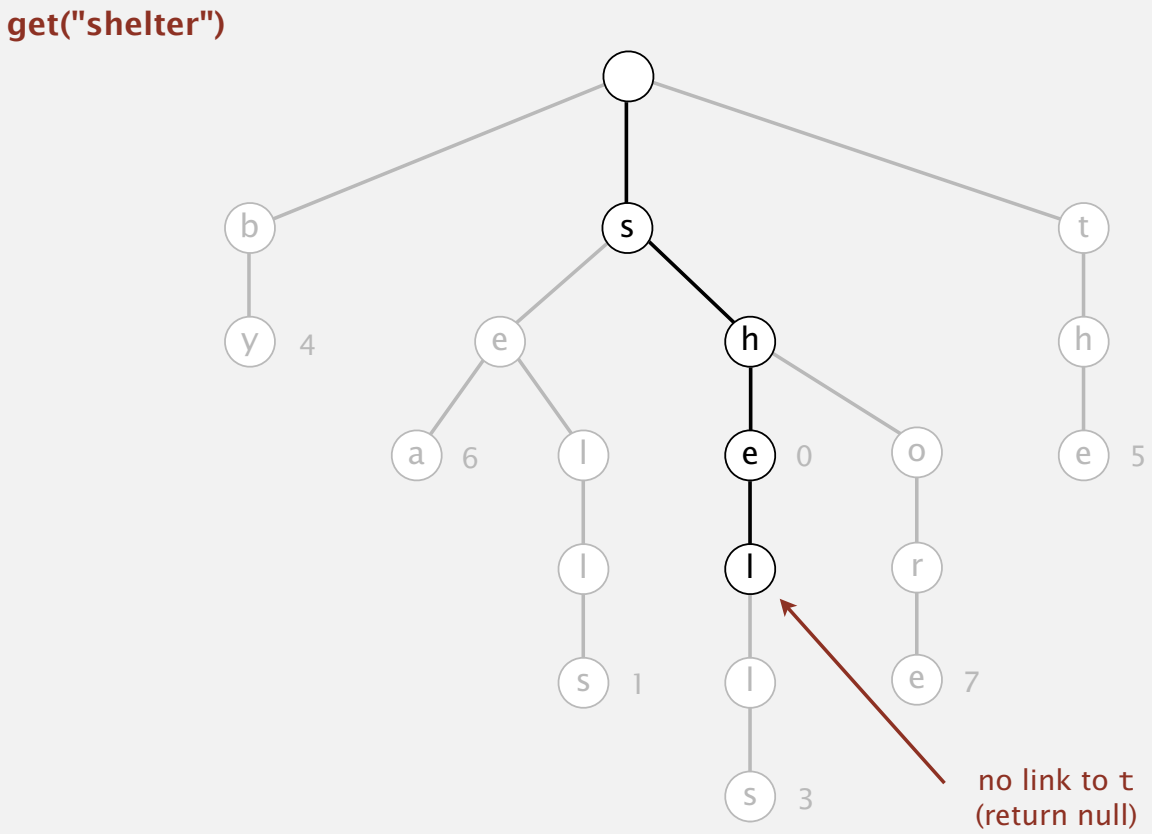
- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Limite des tries

- On doit réserver des tableaux de taille 26 ou même 256 pour chaque nœud

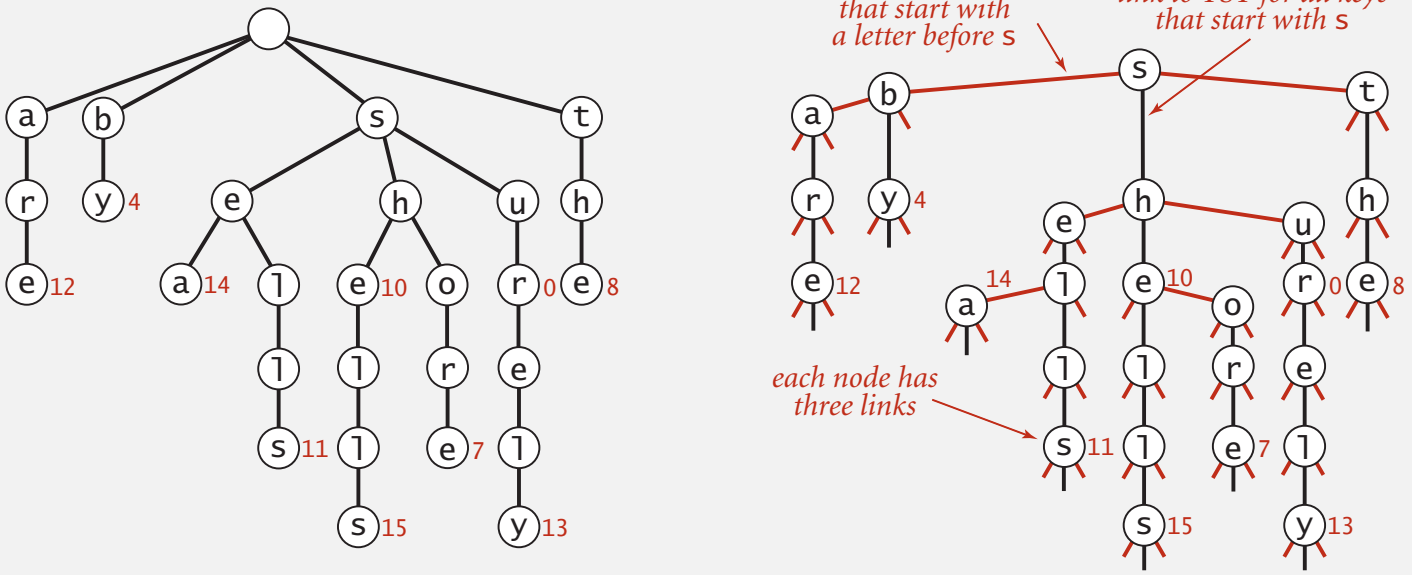
- Dans un tries avec beaucoup de mots, les nœuds éloignés du sommet seront très peu dense. Nous trouverons donc principalement des tableaux composés de près de 256 `nullptr`

  → Très gourmand en mémoire

- Nous allons donc voir dans ce laboratoire une structure à mi chemin entre les arbres de recherche et les tries → Ternary Search Tries

Ternary search tries

- Store characters and values in nodes (not keys).
- Each node has 3 children: smaller (left), equal (middle), larger (right).



TST representation of a trie

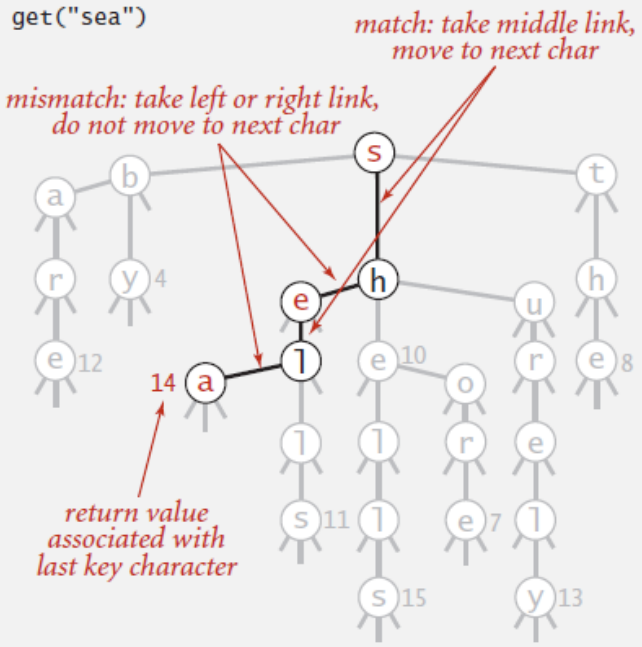
Search in a TST

Follow links corresponding to each character in the key.

- If less, take left link; if greater, take right link.
- If equal, take the middle link and move to the next key character.

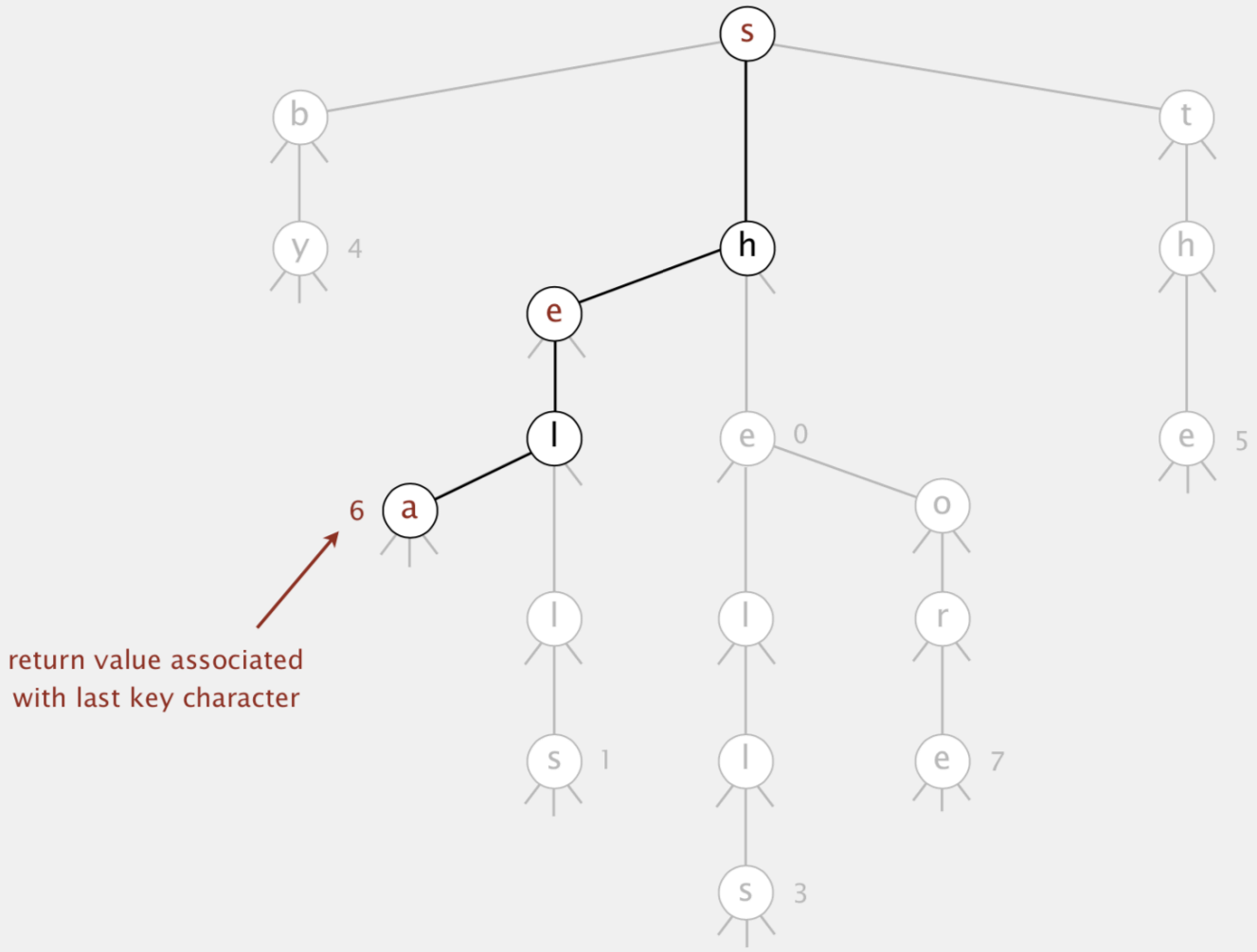
Search hit. Node where search ends has a non-null value.

Search miss. Reach a null link or node where search ends has null value.



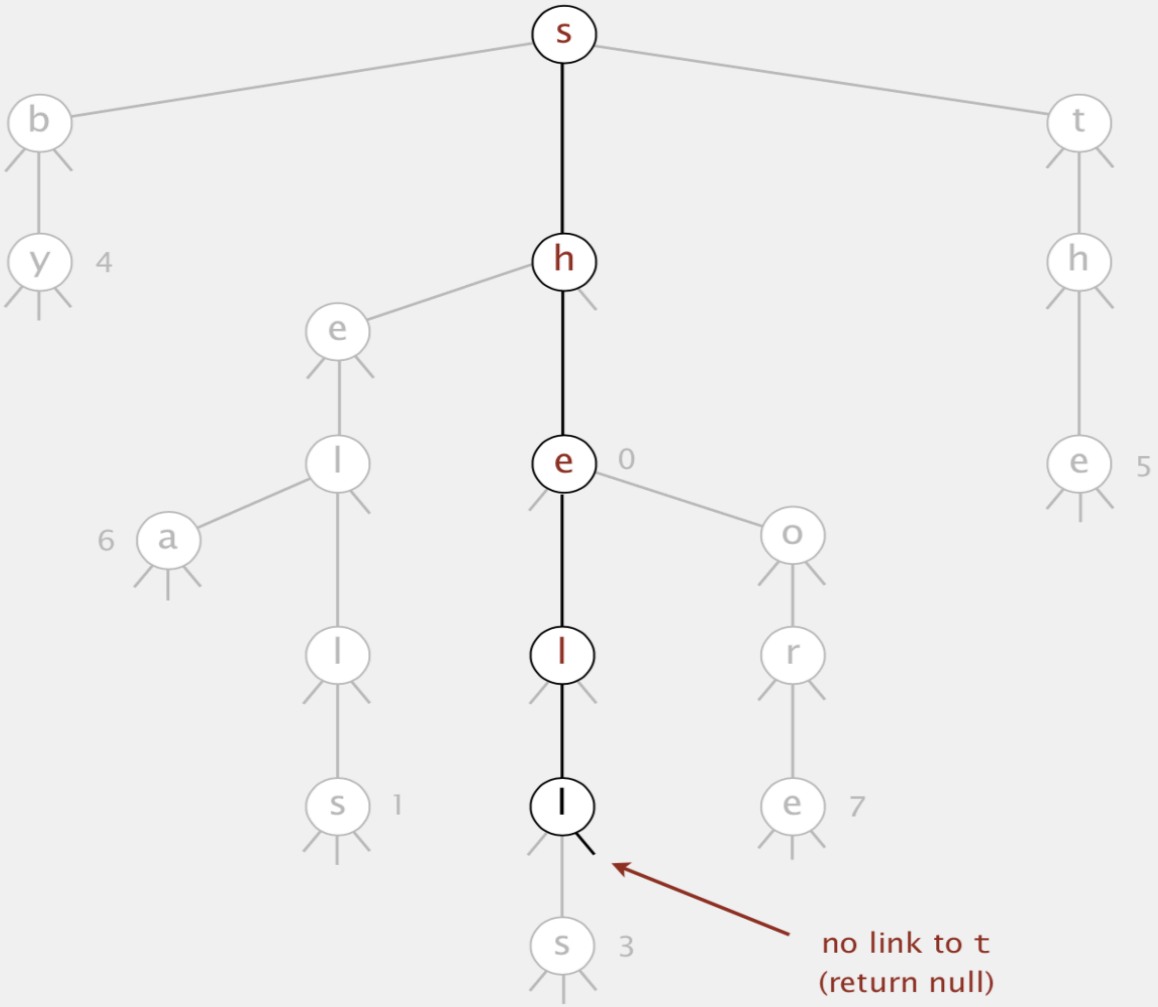
Search hit in a TST

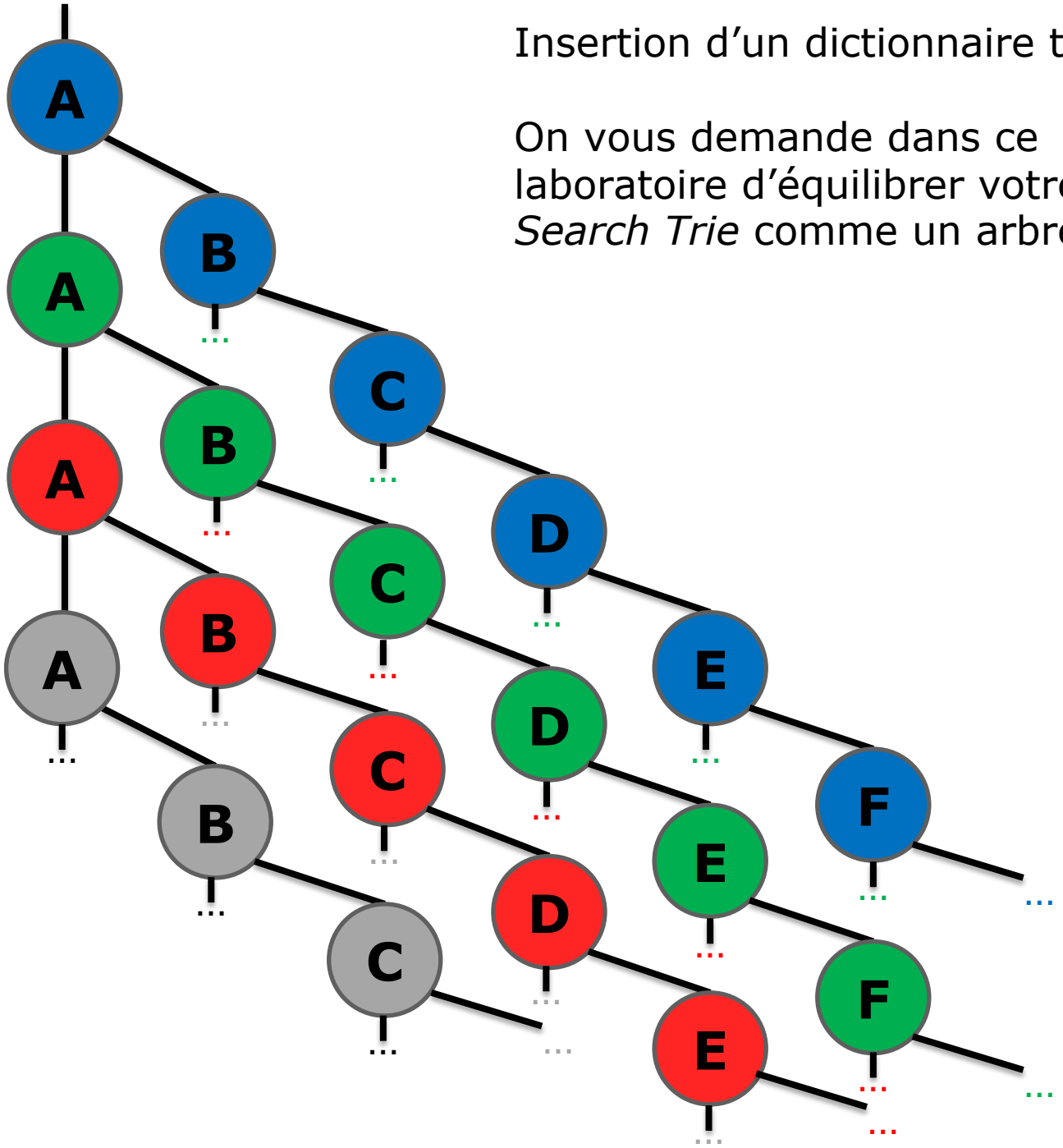
get("sea")



Search miss in a TST

get("shelter")



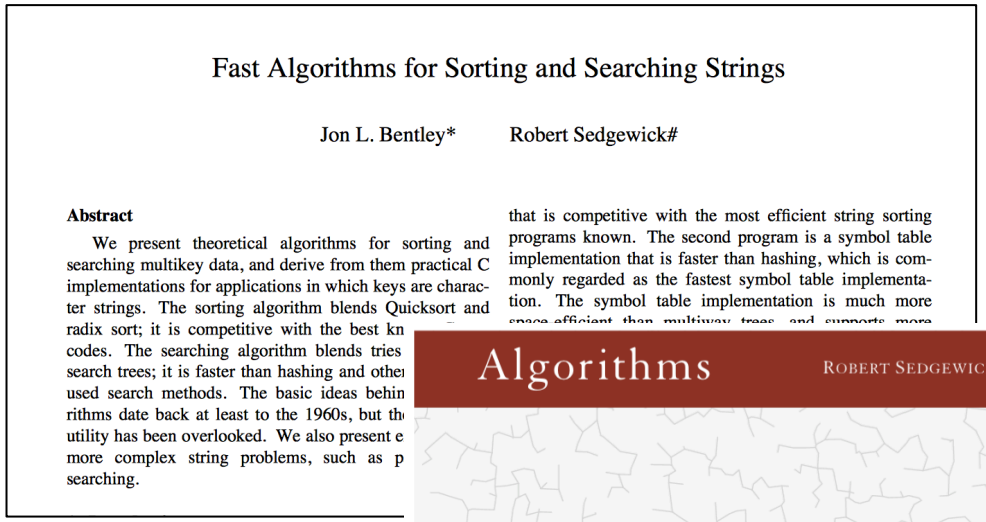


Insertion d'un dictionnaire trié…

On vous demande dans ce laboratoire d'équilibrer votre *Ternary Search Trie* comme un arbre *AVL*

Ressources

BentleyAndSedgewick – Fast Algorithms for Sorting and Searching Strings.pdf



Fast Algorithms for Sorting and Searching Strings

Jon L. Bentley* Robert Sedgewick#

Abstract

We present theoretical algorithms for sorting and searching multikey data, and derive from them practical C implementations for applications in which keys are character strings. The sorting algorithm blends Quicksort and radix sort; it is competitive with the best known codes. The searching algorithm blends tries and search trees; it is faster than hashing and other used search methods. The basic ideas behind rithms date back at least to the 1960s, but the utility has been overlooked. We also present e more complex string problems, such as p searching.

that is competitive with the most efficient string sorting programs known. The second program is a symbol table implementation that is faster than hashing, which is commonly regarded as the fastest symbol table implementation. The symbol table implementation is much more

Chapitre 5.2

52Tries.pdf

Ressources

AVLTree.h

```
//
//  AVLTree.h
//  SearchTrees
//
//  Created by Olivier Cuisenaire on 25.11.14.
//  Copyright (c) 2014 Olivier Cuisenaire. All rights reserved.
//

#ifndef SearchTrees_AVLTree_h
#define SearchTrees_AVLTree_h

#include <algorithm>

template < typename KeyType, typename ValueType >
class AVLTree {
private:

    //
    // Noeux de l'arbre. contient une cle, une valeur, et les liens vers les
    // sous-arbres droit et gauche.
    //
    struct Node {
    public:
        KeyType key;
        ValueType value;
        Node* right; // sous arbre avec des cles plus grandes
        Node* left;  // sous arbre avec des cles plus petites
        int nodeSize;
        int nodeHeight;
        Node(KeyType key, ValueType value) : key(key),value(value), right(nullptr), left(null
    };

    //
```

BinarySearchTree.h

```
//
//  BinarySearchTree.h
//  SearchTrees
//
//  Created by Olivier Cuisenaire on 24.11.14.
//  Copyright (c) 2014 Olivier Cuisenaire. All rights reserv
//

#ifndef SearchTrees_BinarySearchTree_h
#define SearchTrees_BinarySearchTree_h

#include <algorithm>

template < typename KeyType, typename ValueType >
class BinarySearchTree {
private:

    //
    // Noeud de l'arbre. contient une cle, une valeur, et le
    // sous-arbres droit et gauche.
    //
    struct Node {
    public:
        KeyType key;
        ValueType value;
        Node* right; // sous arbre avec des cles plus grande
        Node* left;  // sous arbre avec des cles plus petites
        int nodeSize;
        Node(KeyType key, ValueType value) : key(key),value(value), right(nullptr), left(nullptr), nodeSize(1) { }
    };

    //
    // Racine de l'arbre.
```