COMP2521 23T3

Motivation

11163

Insertion

Searc

Deterior

variant

Applications

Appendix

COMP2521 23T3 Tries

Kevin Luxa

cs2521@cse.unsw.edu.au

Many applications require searching through a set of strings

Examples:

Predictive text Autocomplete Approximate string matching Spell checking

Predictive text



For example, pressing "4663" can be interpreted as the word good, home, hood or hoof

Tries

Insertior

. .

_ ----

. . . .

Applications

.

Appendix

Autocomplete



Motivation

How can we implement a set of strings using data structures covered so far?

AVL tree

Performance: $O(\log n)$ worst case

Hash table

Performance: O(1) average case, O(n) worst case

Motivation

AVL trees and hash tables are efficient, but...

...searching requires user to provide the full string...

...which is not always possible in the above applications (or would be inefficient)

Possible solution: tries

Motivatior

Tries

Insertio

Searc

Detetic

Analys

Application

Appendix

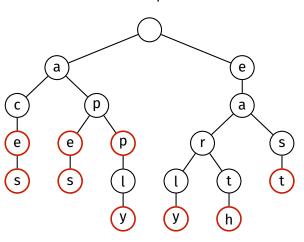
A trie...

- is a tree data structure
- used to represent a set of strings
 - e.g., all the distinct words in a document, a dictionary, etc.
 - we will call these strings keys or words
- supports string matching queries in O(m) time
 - where m is the length of the string being searched for

Note: the word trie comes from retrieval, but pronounced as "try" not "tree"

Tries

Example:



Keys in the trie: ace aces ape apes app apply early earth east

Tries

Insertio

_

Deletio

Analysi

variants

Application

Appendix

Important features of tries:

- Each link represents an individual character
- A key is represented by a path in the trie
- Each node can be tagged as a "finishing" node
 - A "finishing" node marks the end of a key
- Each node may contain data associated with key
- Unlike a search tree, the nodes in a trie do not store their associated key
 - Instead, keys are implicitly defined by their position in the trie

```
Representation
```

```
COMP2521
23T3
```

violivatio

Representation

...,......

Coarch

Jearch

. .

Allatysis

variants

Application

Appendix

```
Assuming alphabetic strings:

#define ALPHABET_SIZE 26

struct node {
    struct node *children[ALPHABET_SIZE];
    bool finish; // marks the end of a key
    Data data; // data associated with key
};
```

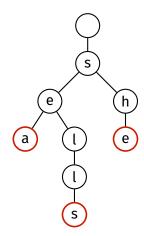
Representation

Search

. . .

Variants

Consider this trie:



Representation

Example

MOLIVALION

Representation

msertioi

Search

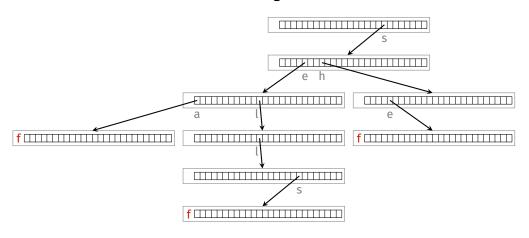
. . .

Allatysis

Variatils

Annendix

Concrete representation: (f = finishing node)



Deleti

. .

.

. . . .

Application

Appendix

Process for insertion:

- Start at the root
 - For each character c in the key (from left to right):
 - If there is no child node corresponding to c, create one
 - ullet Descend into the child node corresponding to c
- Mark the resulting node as a finishing node and insert data (if any)

P2521 3T3 Trie Insertion Example

Insert the following words into an initially empty trie:

sea shell sell shore she

reivacio

Insertion

Sear

Detetit

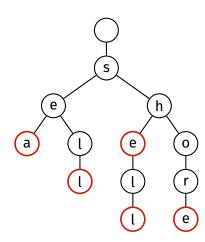
Anatysi

Amaliantian

Appendix

Insert the following words into an initially empty trie:

sea shell sell shore she



Trie Insertion

Pseudocode

MOLIVALIOI

Insertion

Searc

Deletio

Analy

Variant

Application

Annendix

```
Trie insertion can be implemented recursively.
```

```
trieInsert(t, key, data):
    Inputs: trie t
            key of length m and associated data
    Output: t with key and data inserted
    if t is empty:
        t = \text{new node}
    if m = 0:
        t->finish = true
        t->data = data
    else:
        first = kev[0]
        rest = key[1..m - 1] // i.e., slice off first character from key
        t->children[first] = trieInsert(t->children[first], rest, data)
    return t
```

EXERCISE Try writing an iterative version.

Trioc

Insertio

Search

Dete

Arraty

variants

Application

Appendix

Search is similar to insertion:

- Start at the root
- For each character c in the key (from left to right):
 - If there is no child node corresponding to c, return false
 - Descend into the child node corresponding to c
- If the resulting node is a finishing node, then return true, otherwise return false

Motivation

Incortion

msertion

Search

Deletio

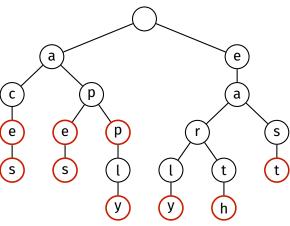
Analysi

variants

Applications

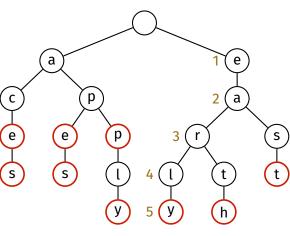
Appendix

Search for "early"



Search

Search for "early"



Found!

Motivation

Insertion

msertion

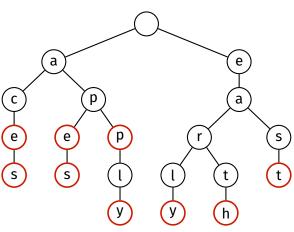
Search

Deletio

Analysi

Applications

Search for "apple"



Trie Search

Motivation

....

IIISertio

Search

Deletic

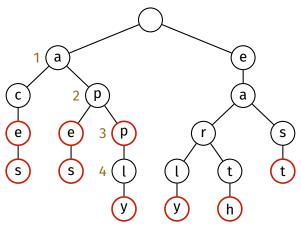
Allatysi

Variatits

Applications

Appendix

Search for "apple"



Not found - node for "appl" has no child node for 'e'

Motivation

11100

Search

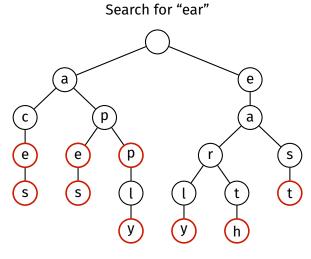
Deletio

Analysi

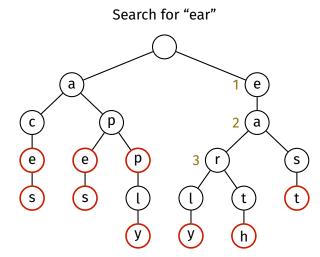
Varianto

Applications

Appendix



Search



Not found - node for "ear" is not a finishing node

Trie Search

Pseudocode

Motivatio

Insertio

Search

Deletic

Application

A man an alise

Trie search can be implemented recursively.

EXERCISE Try writing an iterative version.

Insertio

Soarch

Deletion

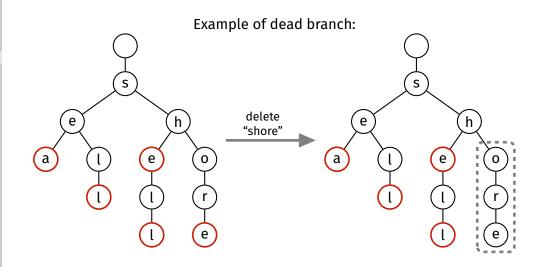
Application

Annondiv

Deletion is trickier...

- Can simply find node corresponding to given key and mark it as a non-finishing node
- ...but this can leave behind dead branches
 - i.e., branches that don't contain any finishing nodes
 - dead branches waste memory

Deletion



Motivation

......

Insertio

Cooreb

Deletion

Annlication

.

Appendix

Process for deletion:

- Find node corresponding to given key
 - If node doesn't exist, do nothing
- Mark the node as a non-finishing node
- While current node is not a finishing node and has no child nodes:
 - Delete current node and move up to parent
 - Handled recursively

Example

MOLIVALIOI

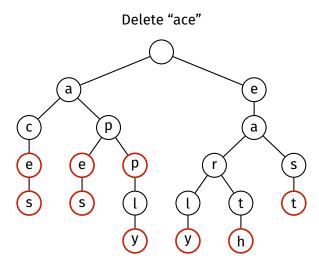
Insertio

Deletion

Allatys

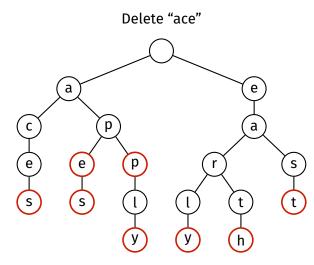
Applications

Appendix



Example

Deletion



Deleted - marked node for "ace" as a non-finishing node

Example

MOLIVALIOI

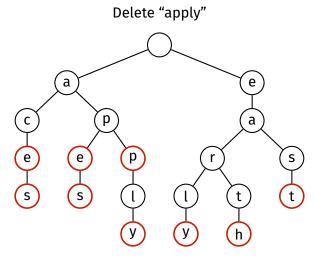
Deletion

,

* 61116116

Applications

Appendix



Example

Motivation

Tries

msertio

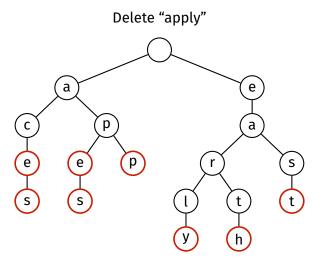
Deletion

Allatys

.

Application:

Appendix



Deleted - deleted nodes corresponding to "apply" and "appl"

Example

MOLIVALIOI

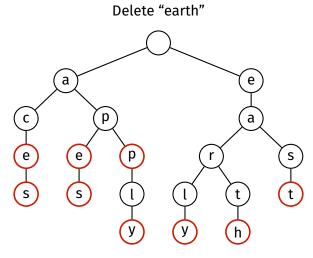
....

Deletion

.

Applications

Appendix



Example

Deletion

Delete "earth" a е е

Deleted - deleted nodes corresponding to "earth" and "eart"

Pseudocode

Motivatio

Trie deletion is implemented recursively.

```
trieDelete(t, key):
Deletion
                Inputs: trie t
                        key of length m
                Output: t with key deleted
                if t is empty:
                    return t
                else if m=0:
                    t->finish = false
                else:
                    first = key[0]
                    rest = key[1..m - 1]
                    t->children[first] = trieDelete(t->children[first], rest)
                if t->finish = false and t has no child nodes:
                    return NULL
                else:
                    return t
```

Motivation

Insertio

111001010

Deletio

Analysis

.

Annlication

Appendix

Analysis of standard trie:

- O(m) insertion, search and deletion
 - where m is the length of the given key
 - each of these needs to examine at most m nodes
- O(nR) space
 - where n is the total number of characters in all keys
 - where *R* is the size of the underlying alphabet (e.g., 26)

Analysis

Motivation

......

Insertion

111501010

Deletio

Analysis

...,

Application

Appendix

Simple trie representation consumes an enormous amount of memory

- Each node contains ALPHABET_SIZE pointers
 - If keys are alphabetic, then this is 26 pointers...
 - ...which is $8 \times 26 = 208$ bytes on an 64-bit machine!
 - If keys can contain any ASCII character, then this is 128 pointers!
- Even if trie contains many keys, most child pointers will be unused

Motivatioi

11165

Insertio

Deletic

Analys

Variants

Linked list

Binary tree Alphabet reduction

Applications

Application

Different representations exist to reduce memory usage at the cost of increased running time:

- Use a singly linked list to store child nodes
- Alphabet reduction break each character into smaller chunks, and treat these chunks as the characters

Linked list of children

tivatio

sertio

Search

Deletio

Analysi

Linked list of

children Binary tree

Alphabet reductio

Applications

Appendix

One technique to reduce memory usage:

Have each node store a linked list of its children instead of an array of ALPHABET_SIZE pointers



Linked list of children

Motivation

I no o unhi o un

Deletion

Analys

Variants

Linked list of children

Alphabet reduction

Applications

Appendi

```
Instead of:
```

```
e h ...
```

We have:



```
struct child *children;
bool finish;
Data data;
};

struct child {
   char c;
   struct node *node;
   struct child *next;
};
```

struct node {

Linked list of children

MOLIVALIOI

Insertio

Searc

Deletic

Allatysi

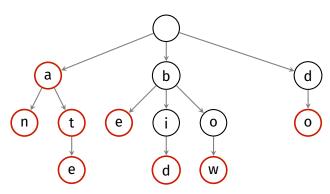
Linked list of

children

Alphabet reduction

Appendix

Consider the following trie:



.

Variants

Linked list of children

0 01 0 01 0

sertion

Search

Detetit

Analysi

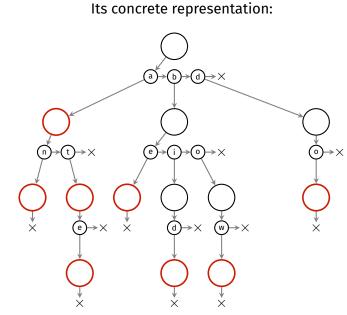
Variants
Linked list of

children

Alphabet reduction Compressed tries

Applications

Appendix



Motivatior

1110010101

Deletio

Analys

Linked lie

Binary tree

Alphabet

Compressed tries

Applications

Appendix

We can simplify this representation by merging each linked list node with its corresponding trie node

This produces the left-child right-sibling binary tree representation

```
struct node {
   char c;
   struct node *children;
   struct node *sibling;
   bool finish;
   Data data;
};
```

MOLIVALIOI

Tries

Deletic

Analysi

Variants

cnilaren

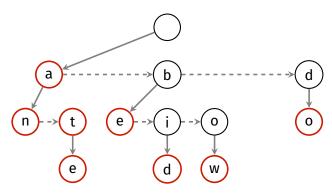
Binary tree

Alphabet reduction

Annlications

Appendix

Concrete representation of above trie:



Left-child right-sibling binary tree

Motivatio

.

....

Deletio

Allaly

children Binary tree

Alphabet reduction

Application:

Annend

Analysis:

- This representation uses much less space
 - Each node just stores one extra pointer to its sibling instead of ALPHABET_SIZE pointers
- But this is at the expense of running time
 - Need to traverse up to ALPHABET_SIZE nodes before reaching desired child

Variants Alphabet reduction

Motivation

Insertio

Analysi

Linked list of children

Alphabet reduction

Applications

Applications

Another technique to reduce memory usage: alphabet reduction

Break each 8-bit character into two 4-bit nybbles

This reduces the branching factor, i.e., the number of pointers in each node

MOLIVALIOI

_

Deletic

Analysi

Linked list

children Binary tree

Alphabet reduction

Annlications

, ippirederone

Appendix

For example, the word "sea" consists of the following bytes:

S	е	a
01110011	01100101	01100001

We break it into 4-bit nybbles like so:

s e		<u> </u>	a		
01110011		01100101		01100001	
0111	0011	0110	0101	0110	0001

Instead of storing the word "sea", we now insert the following word:

0111 0011 0110

110 0101

0110

0001

Variants Alphabet reduction

otivatior

Incartio

Deletio

Analys

Linked list o children

Alphabet reduction

.. .

Applications

Appendi

Analysis:

- This representation uses much less space
 - Much fewer pointers per node
- But this is at the expense of running time
 - Path to each key is twice as long lookups need to visit twice as many nodes



Variants Compressed tries

MOLIVALIOI

111361 (101

Deletio

Analysi

Linked list of

Binary tree

Compressed tries

Applications

Appendix

Another technique to reduce memory usage: use a compressed trie

In a compressed trie, each node contains ≥ 1 character

Obtained by merging non-branching chains of nodes Specifically, non-finishing nodes with only one child are merged with their child



Compressed tries

Variants

.....

sertio

aarch

Deletic

Analysi

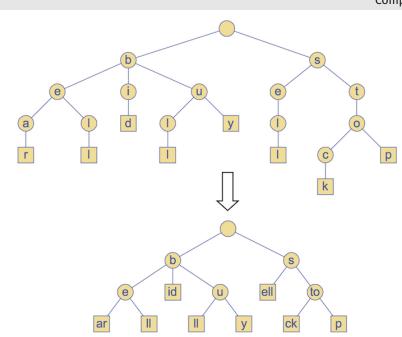
Linked list o

Binary tree

Compressed tries

Applications

Appendix





Applications Word finding

Motivation

.

1115010101

Deletio

Variants

Applications
Word finding

Predictive text

Appendix

Idea:

Given a document, preprocess it by storing all words in a trie, and for each word, store the location of all its occurrences

When user searches for a word, can query the trie instead of scanning entire document

Applications

Word finding

Motivation

Insertion

Dolotio

Analys

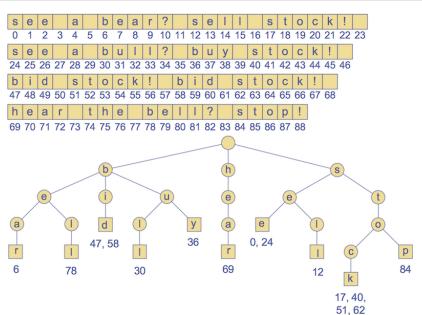
Variant

Application

Word finding

riedictive text

Appendix



Motivation

111361110

Detetic

.

A 1 A

Word finding
Predictive text

Appendix

Predictive text

Given a series of button presses (e.g., on a keypad), where each button can represent multiple letters, find all possible matching words



Motivation

Tries

Insertio

Searc

Deteti

. .

Word finding

Predictive text

Appendix

https://forms.office.com/r/aPF09YHZ3X



COMP2521 23T3

Motivatio

Tries

Insertion

Search

Detection

Analysis

Variants

Applications

Appendix

Insertion example

Appendix

Insertion example

Insert the following words into an initially empty trie:

Motivation

iries

IIISertio

00010

.. .

Applications

Annondiv

Insertion example

Insert the following words into an initially empty trie:



Motivation

Insertion

Search

Detetio

Milalysis

variants

Applications

Ammondin

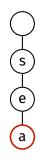
Insertion example

Insert the following words into an initially empty trie:



Insertion example

Insert the following words into an initially empty trie:



Motivation

11100

Insertioi

Jearei

. . .

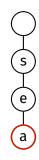
Applications

. .

Insertion example

sea she

Insert the following words into an initially empty trie:



Motivation

Insertion

Deletic

Analysi

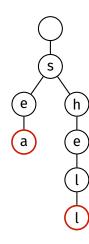
Variants

Application:

Appendi

Insertion example

Insert the following words into an initially empty trie:



Motivation

Insertior

Deletic

Analysi

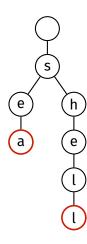
Variants

Application:

Appendi

Insertion example

Insert the following words into an initially empty trie:



Motivation

Insertior

......

Deletio

Analysis

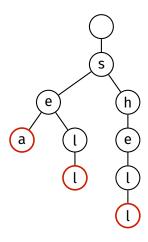
Variants

Applications

Appendix

Insertion example

Insert the following words into an initially empty trie:



Motivation

Insertior

Deletio

Analysi:

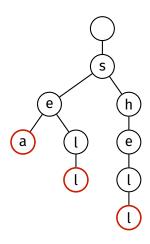
Variants

Applications

Appendix

Insertion example

Insert the following words into an initially empty trie:



Motivation

Tries

Insertior

Analyci

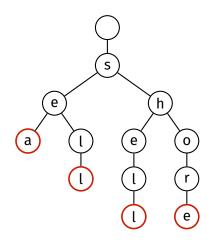
Varianto

Applications

. ...

Insertion example

Insert the following words into an initially empty trie:



Motivation

Trios

Insertior

. .

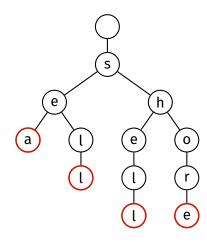
.

Annlications

..

Insertion example

Insert the following words into an initially empty trie:



Motivation

Incombine

msertion

Deletion

Analysis

Variants

Applications

Appendi

Insertion example

Insert the following words into an initially empty trie:

