

Introduction

Definitions

Methods for
searching

Lists: sorted
and unsorted

Sequential search
(unsorted list)

Binary search (sorted
list)

Reminders

Self-organizing
lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

Search

Comp Sci 1575 Data Structures



Searching for items in a dictionary

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

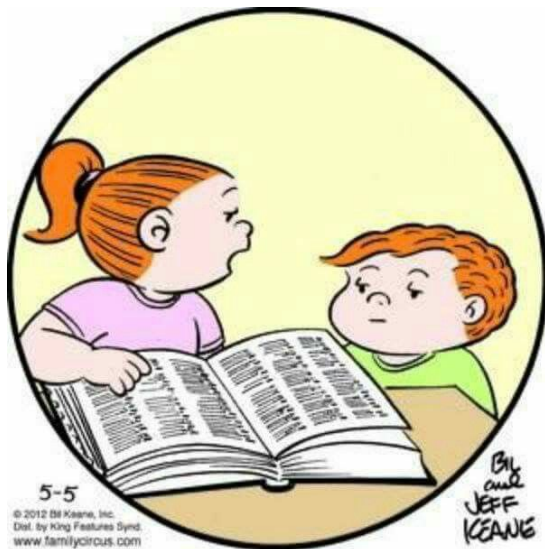
Count

Move-to-front

Transpose

Example application

Bit vectors



“It’s a dictionary. This is how people used to know how to spell.”

Overview of data structures so far

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

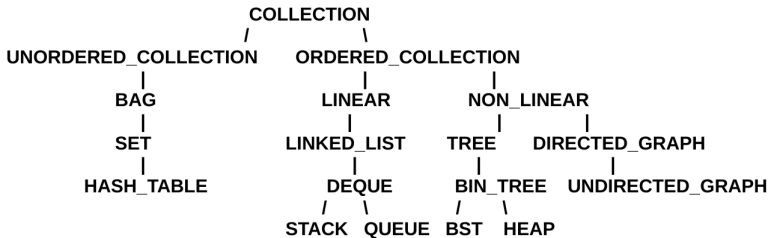
Count

Move-to-front

Transpose

Example application

Bit vectors



Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Is searching the most frequently performed task on a computer?
- Is all computation a form of search?

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Search can be viewed abstractly as a process to determine if an element with a particular value is a member of a group.
- The more common view of searching is an attempt to find the record within a collection of records that has a particular key value, or those records in a collection whose key values meet some criterion such as falling within a range of values.

Introduction

Definitions

 Methods for
 searching

 Lists: sorted
 and unsorted

 Sequential search
 (unsorted list)

 Binary search (sorted
 list)

Reminders

 Self-organizing
 lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Suppose that we have a collection L of n records of the form $(k_1, l_1), (k_2, l_2), \dots, (k_n, l_n)$ where l_j is information associated with key k_j from record j for $1 \leq j \leq n$.
- Given a particular key value K , the search problem is to locate a record (k_j, l_j) in L such that $k_j = K$, if one exists.
- **Searching** is a systematic method for locating the record, or records, with key value $k_j = K$.
- **Successful search** is one in which a record with key $k_j = K$ is found.
- **Unsuccessful search** is one in which no record with $k_j = K$ is found (and no such record exists).
- **Exact-match query** is a search for the record whose key value matches a specified key value.
- **Range query** is a search for all records whose key value falls within a specified range of key values.

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

1 Introduction

Definitions

Methods for searching

2 Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

3 Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Sequential and list methods (today)
- Direct access by key value, hashing (next class)
- Tree indexing methods (Maybe last day of class)

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for
searching

Lists: sorted
and unsorted

Sequential search
(unsorted list)

Binary search (sorted
list)

Reminders

Self-organizing
lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

How does T increase with n ?

```
// Return pos of value k in A of size n
int seqSearch(int A[], int n, int k)
{
    for(int i = 0; i < n; i++)
        if(A[i] == k)
            return i;
    return -1; // -1 signifies not found
}
```

Constant simple operations plus for() loop:

$$T(n) = cn$$

Is this always true?

What if our array is randomly sorted?

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)**
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Binary search on sorted lists

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count


Move-to-front

Transpose

Example application

Bit vectors

| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Key | 11 | 13 | 21 | 26 | 29 | 36 | 40 | 41 | 45 | 51 | 54 | 56 | 65 | 72 | 77 | 83 |



Introduction

Definitions

 Methods for
searching

 Lists: sorted
and unsorted

 Sequential search
(unsorted list)

 Binary search (sorted
list)

Reminders

 Self-organizing
lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

```

// Return pos of value k in A of size n
int binary(int A[], int n, int k)
{
    int low = 0;
    int high = n - 1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        if(k > A[mid])
            low = mid + 1;
        if(k < A[mid])
            high = mid - 1;
        else
            return mid; // found
    }
    return -1;
}

```


Binary search on sorted lists

Introduction

- Definitions
- Methods for searching

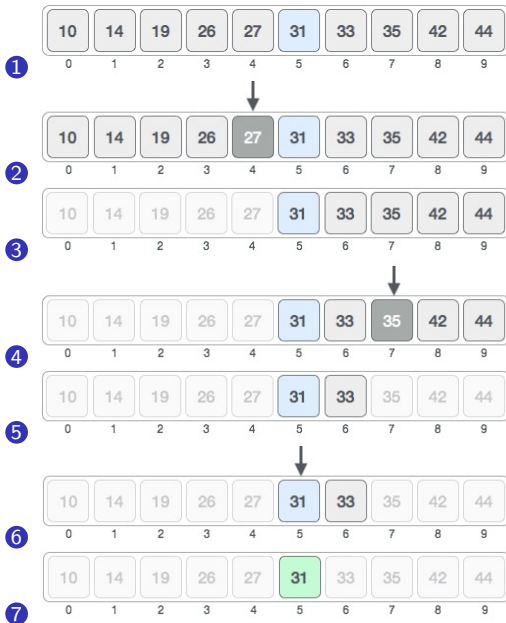
Lists: sorted and unsorted

- Sequential search (unsorted list)
- Binary search (sorted list)
- Reminders

Self-organizing lists

- Problem
- Solutions
- Count
- Move-to-front
- Transpose
- Example application

Bit vectors



Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Binary search may be $\log n$ time, but what is the cost of maintaining a sorted array?
- What algorithm would excel at this?

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Set of (*Key*, *Value*) pairs, search for unique key
- Unsorted array list based dictionary (review UALdict.* files under dictionary day)
- Sorted array list based dictionary (review SALdict.* files under dictionary day)

Check out search(v) on: <https://visualgo.net/en/bst>

- Binary search tree implements a form of binary search (review BST*. * files under BST day)

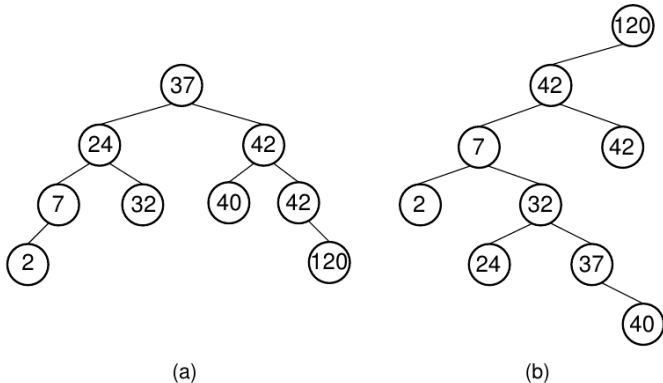


Figure 5.13 Two Binary Search Trees for a collection of values. Tree (a) results if values are inserted in the order 37, 24, 42, 7, 2, 40, 42, 32, 120. Tree (b) results if the same values are inserted in the order 120, 42, 42, 7, 2, 32, 37, 24, 40.

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors



Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- What is the cost of maintaining a sorted BST?
- Is it better than a sorted list?

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem**
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- In the real world, individual records are not often accessed with equal probability.
- So, organize a list with more frequent items earlier in the list, to speed up search times (still not usually as fast as binary search)
- However, we often don't know the frequencies ahead of time, and they may change
- Further, we would like to avoid the cost of having to repeatedly sort our lists.
- Solutions?

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions**
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions
 Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)
 Binary search (sorted list)
 Reminders

Self-organizing lists

Problem
 Solutions
 Count
 Move-to-front
 Transpose
 Example application

Bit vectors

- Modifying a list in real-time to change its order based on access frequency for each key will tend to sort the list by access frequency

- For example: {A, B, C, D, E, F, G, H} could be queried for access of the following items in the following order:

F D F G E G F A D F G E

and then modified by several rules:

- ① When a record's frequency count goes up, it moves forward in the list to become the last record with that value for its frequency count, resulting in F G D E A B C H, with a total cost of 45 comparisons (search and re-ordering)
- ② When a record is accessed, move it to the front, producing E G F D A B C H, for a total cost of 54 comparisons
- ③ When a record is accessed, swap it with the key preceding its position in the list, producing A B F D G E C H, with a total of 62 comparisons

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- Store a count of accesses to each record and always maintain records in this order.
- Whenever a record is accessed, move it toward the front of the list if its number of accesses becomes greater than a record preceding it.
- Count will store the records in the order of frequency that has actually occurred so far.
- Besides requiring space for the access counts, count does not react well to changing frequency of access over time; once a record has been accessed a large number of times under the frequency count system, it will remain near the front of the list regardless of further access history.

1) Count method

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

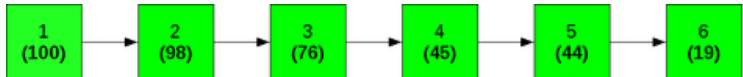
Count

Move-to-front

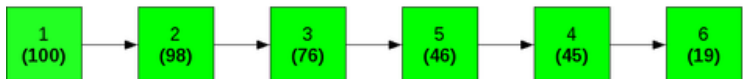
Transpose

Example application

Bit vectors



If node 5 is accessed three times its count will become 46 and it will be moved to the front



2) Move-to-front method

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

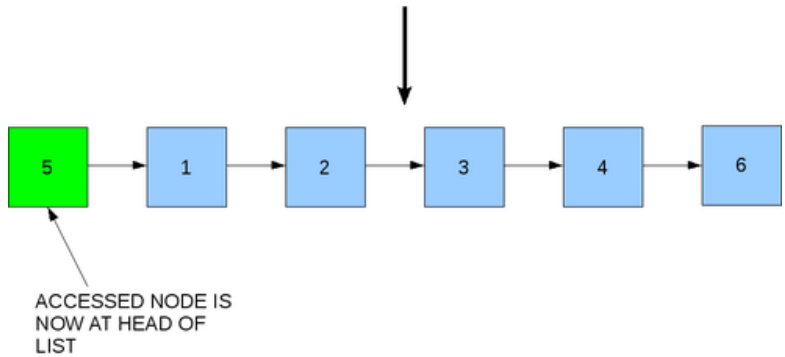
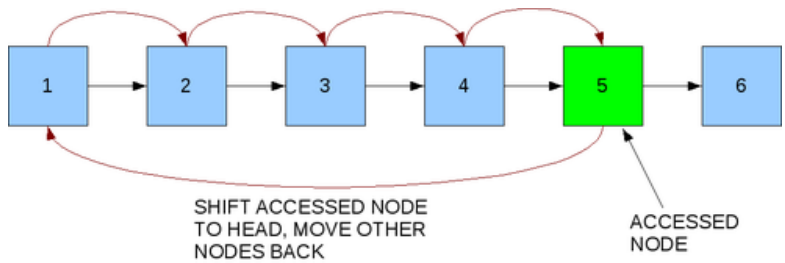
Example application

Bit vectors

- Bring a record to the front of the list when it is found, pushing all the other records back one position.
- This is analogous to the least recently used buffer (LRU) replacement strategy.
- This heuristic is easy to implement if the records are stored using a linked list, and not for an array.
- Move-to-front responds well to local changes in frequency of access, in that if a record is frequently accessed for a brief period of time it will be near the front of the list during that period of access.
- Move-to-front does poorly when the records are processed in sequential order, especially if that sequential order is then repeated multiple times.

Move-to-front method

- Introduction
- Definitions
- Methods for searching
- Lists: sorted and unsorted
- Sequential search (unsorted list)
- Binary search (sorted list)
- Reminders
- Self-organizing lists
- Problem
- Solutions
- Count
- Move-to-front**
- Transpose
- Example application
- Bit vectors



Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

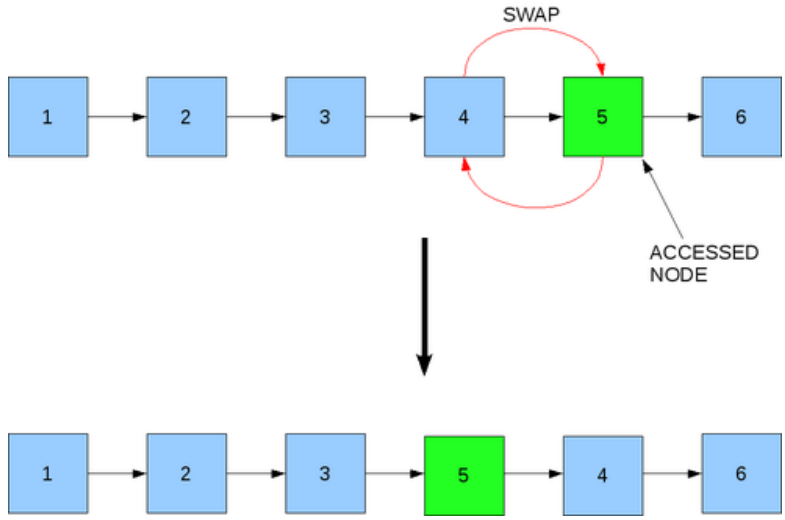
Example application

Bit vectors

- Swap any record found with the record immediately preceding it in the list.
- Transpose is good for list implementations based on either linked lists or arrays.
- Frequently used records will, over time, move to the front of the list, and records that were once frequently accessed but are no longer used will slowly drift toward the back.
- Some pathological sequences of access can make transpose perform poorly. For example if two items are accessed alternating and repeating, they would not move globally. A variation on transpose would be to move the accessed record forward in the list by some fixed number of steps, which addresses this problem.

3) Transpose method

- Introduction
- Definitions
- Methods for searching
- Lists: sorted and unsorted
- Sequential search (unsorted list)
- Binary search (sorted list)
- Reminders
- Self-organizing lists
- Problem
- Solutions
- Count
- Move-to-front
- Transpose**
- Example application
- Bit vectors



Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

Procedure:

- ① If the word has been seen before, transmit the current position of the word in the list. Move the word to the front of the list.
- ② If the word is seen for the first time, transmit the word. Place the word at the front of the list.

Both the sender and the receiver keep track of the position of words in the list in the same way (using the move-to-front rule), so they agree on the meaning of the numbers that encode repeated occurrences of words. For example:

Pre-transmit: the car on the left hit the car I left
 Compressed: the car on 3 left hit 3 5 I 5

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- 1 Introduction
 - Definitions
 - Methods for searching
- 2 Lists: sorted and unsorted
 - Sequential search (unsorted list)
 - Binary search (sorted list)
 - Reminders
- 3 Self-organizing lists
 - Problem
 - Solutions
 - Count
 - Move-to-front
 - Transpose
 - Example application
- 4 Bit vectors

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

- A bit array (also known as bit map , bit set, bit string, or bit vector) can enable forms of search
- For example, the bit array for the set of primes in the range $[0 : 15]$. The bit at position i is set to 1 if and only if i is prime:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

For example, if we had two bit arrays, one for prime numbers and one for odd numbers, we could search for the set of numbers between 0 and 15 that are both prime and odd numbers via:

0011010100010100 & 0101010101010101

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

| X | Y | X&Y | X Y | X^Y | ~(X) |
|----------|----------|----------------|------------|------------|-------------|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Introduction

Definitions

Methods for searching

Lists: sorted and unsorted

Sequential search (unsorted list)

Binary search (sorted list)

Reminders

Self-organizing lists

Problem

Solutions

Count

Move-to-front

Transpose

Example application

Bit vectors

| Operator | Symbol | Form | Operation |
|-------------|--------|--------------|--|
| left shift | << | $x \ll y$ | all bits in x shifted left y bits |
| right shift | >> | $x \gg y$ | all bits in x shifted right y bits |
| bitwise NOT | ~ | $\sim x$ | all bits in x flipped |
| bitwise AND | & | $x \& y$ | each bit in x AND each bit in y |
| bitwise OR | | $x y$ | each bit in x OR each bit in y |
| bitwise XOR | ^ | $x \wedge y$ | each bit in x XOR each bit in y |