

Chapter 7 Searching

This chapter introduces the problem of **searching a list to find a particular entry**.

Our discussion centers on two well-known algorithms: **sequential search** and **binary search**.

We shall develop several sophisticated mathematical tools, used both to demonstrate the correctness of algorithms and to calculate how much work they must do.

These mathematical tools include **invariant assertions**, **comparison trees**, and the **big-O notations**. Finally, we shall obtain **lower bounds** showing conditions under which any searching algorithm must do at least as much work as binary search.

Contents Points

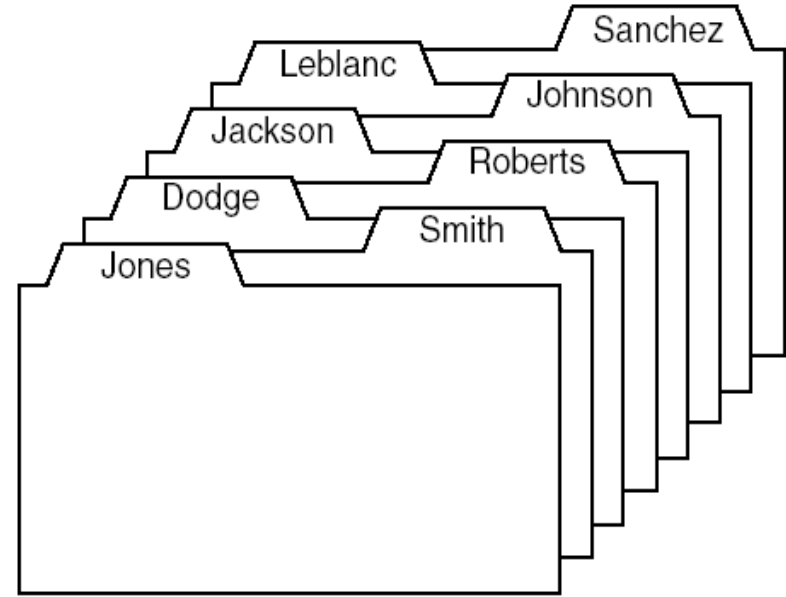
- ❑ Introduction, notation
- ❑ Sequential Search
- ❑ Binary Search
- ❑ Comparison Trees
- ❑ Lower Bounds

Introduction and Notation

□ Records and Keys

We are given a *list* of *records* (记录表), where each record is associated with one piece of information, which we shall call a *key*. (关键字)

We are given one key, called the *target* (目标), and are asked to *search* the list to find the record(s) (if any) whose key is the same as the target.



对应每一个目标关键字，可能有0个或多个对应记录

Introduction

Average Searching Length:
how many comparisons have
been made in the average case

□ Algorithm Analysis

We often ask how many times one key is compared with another during a search. This gives us a good measure of the total amount of work that the algorithm will do.

(以关键字比较次数作为算法效率的量度) (ASL)

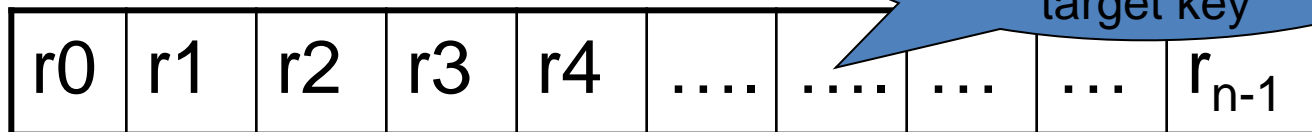
□ Internal Searching and external searching

Internal searching means that all the records are kept in high-speed memory. In *external searching*, most of the records are kept in disk files. We study only internal searching. (内部查找、外部查找)

□ contiguous lists only

只讨论顺序存储结构

Find a record which
key is same as the
target key



Records and Keys in C++

The records (from a **class** Record) that are stored in the list being searched (generally called the list) must conform to the following minimal standards **（最低要求）**

□ Every Record is associated to a key (of a type or **class** called Key).

（每一条记录对应着一个关键字）

□ Key objects can be compared with the standard operators

== , != , < , > , <= , >= . （关键字可以相互比较）

□ There is a conversion operation to turn any Record into its associated Key.

（将记录转换为对应关键字的操作）

□ Record objects can be compared to each other or to a Key by first converting the Record objects to their associated keys.

（记录之间可以相互比较，也可以与关键字进行比较）

Records and Keys in C++

□ Examples of the conversion operation:

- A method of the class Record:
operator Key() const,

转换函数，特殊类型的成员函数，定义了一个由用户定义的转换，把一个类对象转换成其他的类型

- A constructor for the class Key, with declaration
Key(const Record &);

- If the classes Key and Record are identical, no conversion needs to be defined, since any Record is automatically a Key.

□ We do not assume that a Record has a Key object as a data member, although often it does. We merely assume that the compiler can turn a Record into its corresponding Key.

Parameters for Search Functions

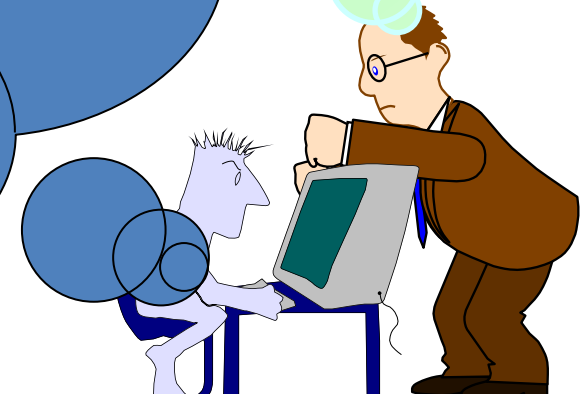
(查找)

Where to search?

Search what?

How to return searching result?

Searching in a contiguous list
stored records;
Search a record which has the
same key as the target key;
Return the **position** if the target
key exists in the list else return
not_present



Parameters for Search Functions

(查找算法的参数)

- Each searching function has two input parameters: (入口参数)
 - First is the **list** to be searched;
 - Second is the **target** key for which we are searching.
- Each searching function will also have an output parameter and a returned value (出口参数和函数返回值)
 - The returned value has type **Error_code** and indicates whether or not the search is successful in finding an entry with the target key.

Parameters for Search Functions

(查找算法的参数)

- If the search is successful, then the returned value is success, and the output parameter called position will locate the target within the list. (查找成功时, 出口参数指向目标关键字所在记录的位置)
- If the search is unsuccessful, then the value **not_present** is returned, and the output parameter may have an undefined value or a value that will differ from one method to another.

```
Error_code sequential_search(const  
List<Record> &the_list,  
const Key &target, int &position)
```

Key and record Definition in C++

方案1:选择固有的类型作为记录和关键字的类型

□ To select existing types as records and keys, a client could use type definition statements such as

```
typedef int Key;
```

```
typedef Key Record;
```

The simplest case



Key Definition in C++

方案2: 设计自己的**Key**类和**record**类

A client can design new classes that display appropriate behavior based on the following skeletons

```
// Definition of a Key class:
```

```
class Key{
```

```
public:
```

```
    // Add any constructors and methods for key data.构造函数及方法
```

```
private:
```

```
    // Add declaration of key data members here. key类的数据成员声明
```

```
};
```

```
// Declare overloaded comparison operators for keys:
```

```
//声明key类的操作符重载函数
```

```
bool operator == (const Key &x, const Key &y);
```

```
bool operator > (const Key &x, const Key &y);
```

```
bool operator < (const Key &x, const Key &y);
```

```
bool operator >= (const Key &x, const Key &y);
```

```
bool operator <= (const Key &x, const Key &y);
```

```
bool operator != (const Key &x, const Key &y);
```



Record Definition in C++

设计自己的Record类

```
// Definition of a Record class:  
class Record{  
public:  
    operator Key( ); //implicit conversion from Record to Key .  
    // Add any constructors and methods for Record objects.  
private:  
    // Add data components.  
};
```

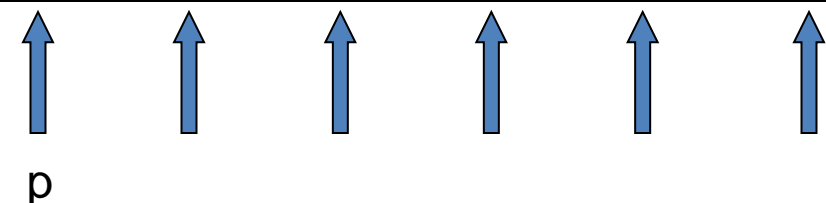
Sequential Search and Analysis

□ Algorithm

1、Begin at one end of the list and scan down it until the desired key is found or the other end is reached. (从表的一端开始依次扫描查找表直至找到或到达表的另一个端点)

2、find the position : ? ?
of the target : 11 8
In the_list

position	0	1	2	3	4	5	6	7	8	9
Record (key)	5	3	7	1	9	11	15	19	11	17



p

Sequential Search

```
Error_code sequential_search(const List<Record> &the_list,  
                             const Key &target, int &position)  
{  
  
}
```

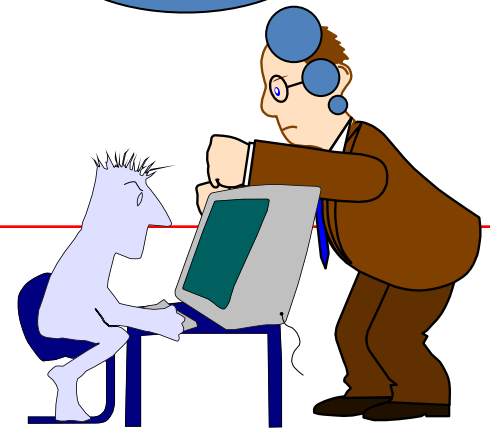
Sequential Search

常量输入参数

输出参数

```
Error_code sequential_search(const List<Record> &the_list,  
                             const Key &target, int &position)  
{  
    int s = the_list.size( );  
    for (position = 0; position < s; position++ ) {  
        Record data;  
        the_list.retrieve(position, data);  
        if (data == target) return success;  
    }  
    return not_present;  
}
```

Does the algorithm depend on the list's implementation?



Analysis

To analyze the behavior of an algorithm that makes comparisons of keys, we shall use the count of these key comparisons as our measure of the work done.

□ The number of comparisons of keys done in sequential search of a list of length n is (without sentinel)

- Unsuccessful search: n comparisons
- Successful search, best case: 1 comparison
- Successful search, worst case: n comparisons
- Successful search, average case: $(n + 1) / 2$ comparisons.

$$ASL = \sum_{i=0}^{n-1} P_i C_i$$



Average Search
Length

P_i : Frequency of the searching of the no. i key .

C_i : How many comparisons been made when finding the no. i key.

In general , $P_i = 1/n$

$$ASL = (1+2+3+....+n-1+n)/n = (n+1)/2$$

Sequential Search with sentinel

Outline of the algorithm

- ❑ insert an extra item at the end with key target
- ❑ scan the list from the beginning
- ❑ delete the last item we have inserted
- ❑ return the searching result according to different cases

Example:

find the position : ? ?
of the target : 11 8

position	0	1	2	3	4	5	6	7	8	9	10
Record (key)	1	3	5	7	9	11	13	15	17	19	<u>81</u>

Example:

find the position : ? ?
of the target : 11 8

position	0	1	2	3	4	5	6	7	8	9	10
Record (key)	1	3	5	7	9	11	13	15	17	19	

Diagram illustrating a search process on a sorted array. The array contains keys [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]. Two blue arrows labeled 'p' point to the first and sixth elements (keys 1 and 11), representing the current search range.

position	0	1	2	3	4	5	6	7	8	9	10
Record (key)	1	3	5	7	9	11	13	15	17	19	

Diagram illustrating a search process on a sorted array. The array contains keys [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]. Two blue arrows labeled 'p' point to the first and tenth elements (keys 1 and an empty slot), representing the current search range.

```
Error_code sequential_search(List<Record> &the_list,  
const Record &target, int &position)
```

/ **Post:** If an entry in the_list has key equal to target, then return success and the output parameter position locates such an entry within the list. Otherwise return not_present and position becomes invalid. */*

```
{  
    Record data;  
    int s = the_list.size( );  
    the_list.insert(s, target);//插入target对应记录? ?  
    for (position = 0; ; position++) {  
        the_list.retrieve(position, data);  
        if (data == target) break;//顺序查找target记录  
    }  
    the_list.remove(s, data); //删除刚插入的target记录  
    if (position < s) return success;  
    return not_present;  
}
```