Intermediate Programming

— Search algorithm—

Waseda Univ.

Today's Topics

- Table (database)
- Search algorithms
  - Sequential search
  - Binary search

Table (database)

Table field

ID	Name	Address	Phone
0	Waseda Taro	Shinjuku	012-345-6666
1	Okuma Hana	Ikebukuro	012-345-7777
⋮	⋮	⋮	⋮
99	Riko Jiro	Shibuya	012-345-9999

row

- Table is a set of data elements within a database. It consists of fields (columns), and rows.
- For example, in the above table, each row contains a personal data and fields denotes a particular type of data such as ID, name, etc.
- Table has a specified number of fields, but can have any number of rows.
- Operations in the table are insertion, revision, delete, search, sorting, etc.

Search algorithm

Table

field

row

ID	Name	Address	Phone
0	Waseda Taro	Shinjuku	012-345-6666
1	Okuma Hana	Ikebukuro	012-345-7777
⋮	⋮	⋮	⋮
99	Riko Jiro	Shibuya	012-345-9999

- Search is a method for finding a particular row in a table.
- The search algorithm finds a row containing particular data of a field. Such data is given by a user.

  Ex. Find a row that contains "Waseda Taro" in the "Name" field.

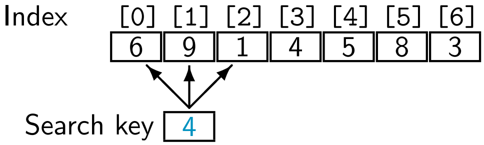
  The field in which we search particular data is named key. The data given by a user is called the search key.

Search algorithm

- Typical search algorithms:
  - Sequential search (linear search)
  - Sentinel search
  - Binary search
  - Hashing
  - etc.

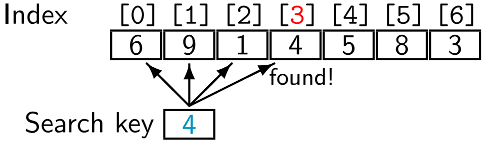
Sequential search algorithm

- Sequential search is the simplest search algorithm that checks each data in the table until the desired data is found or the key is exhausted.
- For the simplicity, let the table be an array of `double` type. The field contains only one data of `double` type.
- When a user give a value (search key), this algorithm find the index of the target that is equal to the search key.
    - If such target doesn't exists, this algorithm returns "not found".



Sequential search algorithm

- This algorithm checks each element until the element is equal to the search key. If the desired element is found, it returns the index of the element.



- For N elements, the best case cost is only one comparison (the search key is equal to the first element) but the worst case cost and the expected cost of sequential search are both $\mathcal{O}(N)$.

Sequential search algorithm

- Let's implement the following program:

```
for (i = 0; i < N; i++) {
  if (Data[i] == key) {
    break;     /* Search key is found. */
  }
}
if (i < N) {  /* Search key is found until the looping. */
  printf("found. index: %d\n", i);
} else {      /* Key is exhausted. */
  printf("not found.\n");
}
```

- This program decides whether the looping breaks or not by the "`if`" stament with the same condition as that of the "`for`" stament (`i<N`).

Exercise

Exercise: seqsearch.c

Let an array of `double` type (Data) be given by 5, 3, 1, 4, 2.
Write a program that find the index of the target equal to the search key by sequential search algorithm.

- Output of this program is as follows:

```
key? 4
5, 3, 1, 4, 2,
Data[0]=5 is not equal to 4.
Data[1]=3 is not equal to 4.
Data[2]=1 is not equal to 4.
Data[3]=4 is equal to 4!
found. index: 3
```

Answer

```c
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);
  // Start sequential search
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

Answer

```c
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);

  // Start sequential search
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

Answer

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);
  // Start sequential search
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

Answer

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);

  // Start sequential search
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

Binary search algorithm

- Binary search algorithm finds the position of the target within a sorted array. This algorithm begins by comparing the search key to the middle element of the array. If the search key is less than the middle element, then this algorithm continues on the lower half of the array. This process eliminates half of elements.
- For an array with N elements, almost $\log_2 N$ times comparison are necessary ($\mathcal{O}(\log N)$). Only 20 times comparison are needed for a milion elements.

Algorithm

For sorted N elements: $x_0, x_1, ..., x_{N-1}$ $(x_0 \leq x_1 \leq ... \leq x_{N-1})$, we find the index i that satisfies $x_i = x^*$, where x^* is a given search key.

Step 1. Let $l = 0$ and $r = N-1$.

Step 2. If $l < r$ doesn't hold, go to Step 4.

Step 3. Let $m = (l + r)/2$.

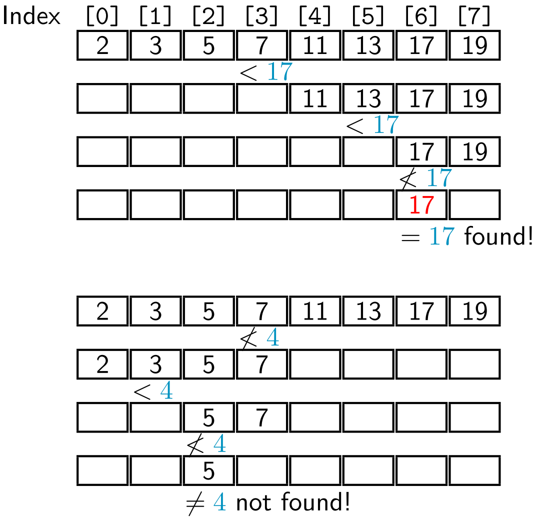
- If $x_m < x^*$, $l \leftarrow m + 1$.
- Otherwise, $r \leftarrow m$.

Go to Step 2.

Step 4. If $x_l = x^*$, return l.
Otherwise, the algorithm doesn't find the search key in the array.

Binary search algorithm

- Example for $N = 8$.
- Compare whether the middle element is less than the search key. If yes, continues on the upper half of the array. Otherwise, continues on the lower half of the array.
- When only one element remains, compare whether that is equal to the search key. If yes, return l as the index of the target. Otherwise, return "not found".



Summary

- Table (database)
- Search algorithms
  - Sequential search
  - Binary search