Progetto Corda

Alberto Ferrari

Alberto Ferrari - uniPr

Array (visita e ricerca)

Array

- Struttura *statica omogenea*
  - non in tutti i linguaggi ... array dinamici
- *Accesso diretto* a ogni elemento attraverso l'indice (complessità dell'accesso *O(1)*)

Algoritmo di visita

- Percorrere *una e una sola volta* tutti gli elementi
- Es. stampa array

visita

C

```c
void visita_array(int a[], int n)
{
  int i;
  for (i = 0; i < n; i++)
      elabora(a[i]);
}
```

“se elabora ha complessità x passi la complessità dell’algoritmo risulta = 1+n(1+x+1)+1 = (x+2)n+2 = O(n) ”

Ricerca

- Stabilire se un valore è *presente* all'interno dell'array restituendo l'indice dell'elemento o -1 se non presente
- **Ricerca sequenziale** (o *ricerca lineare*): algoritmo per trovare un elemento in un insieme non ordinato
  - Si effettua la scansione dell'array sequenzialmente
- **Ricerca binaria** (o *ricerca dicotomica* o *riceca binaria*): algoritmo per trovare un elemento in un insieme ordinato
  - Si inizia la ricerca dall'elemento centrale, si confronta questo elemento con quello cercato:
    - se corrisponde, la ricerca termina indicando che l'elemento è stato trovato
    - se è superiore, la ricerca viene ripetuta sugli elementi precedenti
    - se invece è inferiore, la ricerca viene ripetuta sugli elementi successivi

Es1 - Algoritmo di ricerca lineare

- Scrivere l'algoritmo di *ricerca lineare* di un elemento in un array *non ordinato*

```c
int ricerca_lineare(int a[], int n, int valore)
```

- Calcolare la complessità computazionale
  - nel caso ottimo, pessimo e medio
- Definire la classe di complessità asintotica nel caso medio

Ricerca lineare

C

```c
int linearSearch(int array[], int size, int val)
{
  for (int i=0;i<size;++i)
      if (array[i]==val)
          return i;
  return -1;
}
```

- Caso peggiore O(n)
- Caso ottimo O(1)
- Caso medio O(n/2)

Es2 - Algoritmo di ricerca binaria

- Scrivere l'algoritmo di *ricerca binaria* di un elemento in un array *ordinato*

```c
int ricerca_binaria(int a[], int n, int valore)
```

- Calcolare la complessità computazionale
  - nel caso ottimo, pessimo e medio
- Definire la classe di complessità asintotica nel caso medio

Ricerca binaria

C

```c
int binarySearch(int array[], int size, int val)
{
    int first, last, medium;
    first = 0;
    last = size - 1;
    while(first <= last) {
        medium = (first + last) / 2;
        if(array[medium] == val)
            return medium; // value found
        if(array[medium] < val)
            first = medium + 1;
        else
            last = medium - 1;
    }
    return -1; // not found
 }
```

“Caso peggiore O(log2n) - Caso ottimo O(1) - Caso medio O(log2n) ”

Complessità computazionale e tempo di esecuzione

- Fare una *misurazione empirica* della complessità temporale dei due algoritmi realizzando un programma che genera in modo casuale un array ordinato poi testa le due funzioni e calcola i tempi di risposta nel caso ottimo medio e pessimo
- Utilizzare un numero elevato di elementi per l'array

Esempio - calcolo tempo C

C

```c
#include <stdio.h>
#include <time.h>
#include <sys/timeb.h>
int main(void) {
    struct timeb start,end;
    ftime(&start);
    // --- your code here ---
    ftime(&end);
    int sec,mil;
    sec = end.time - start.time;
    mil = end.millitm - start.millitm;
    if (mil < 0) {
      mil += 1000;
      sec--;
    }
    printf("elapsed time %ds %dms\n",sec,mil);
    return 0;
}
```

Esempio - numero casuale in C

C

```c
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
int main(void) {
    int n;
    srand(time(NULL));
    n = rand() % 100 + 1;
    printf("%d \n",n);
  return EXIT_SUCCESS;
}
```

Alberto Ferrari
Universita' degli Studi di Parma