Appunti Algoritmi e Strutture Dati

Luca Seggiani

12 Marzo 2024

1 Insertion sort

L'insertion sort utilizza il metodo più intuitivo per ordinare un vettore: si prende ad ogni iterazione il primo elemento, si confronta con ogni suo precedente finchè non si trova la sua posizione corretta, e poi si spostano tutti i precedenti necessari di una posizione in avanti per lasciargli posso. Notiamo che nell'implementazione in pseudocodice il while usato per spostare gli elementi ha la conseguenza di lasciare il primo elemento con un valore duplicato: usiamo allora la variabile ausiliaria current per rimettere il valore giusto al suo posto.

```
void sort(int* vett, int dim) {
  int current = 0;
  int p = 0;
  for(int i = 1; i < dim; i++) {
    current = vett[i];
    p = i - 1;

    while(p >= 0 && vett[p] > current) {
      vett[p + 1] = vett[p];
      p--;
    }
    vett[p + 1] = current;
  }
}
```

Possiamo fare l'analisi della complessità e trovare che, come per tutti gli algoritmi di ordinamento in-place, la complessità massima è di $O(n^2)$. Il worst case sarà chiaramente quello di un vettore ordinato al contrario, cioe in ordine decrescente.

2 Debugging

Il processo di debugging consiste nel rimuovere eventuali errori dal codice scritto. Si possono distinguere le seguenti categorie di debugging:

- Visuale: osservando lo stato del programma in un qualsiasi momento (ad esempio scrivendo sul buffer di uscita le informazioni che ci interessano);
- Debugger: utilizzando appositi software chiamati debugger (GDB, DDD) che forniscono alcuni strumenti utili al debugging quali i breakpoint, l'analisi della memoria in tempo reale, ecc...;
- Compilatore: utilizzando i messaggi di errore forniti dallo stesso compilatore;
- Analisi della memoria: utilizzando software come Valgrind per gestire correttamente la memoria ed individuare leak e segmentazioni.

3 Programmi in memoria dinamica

Utilizziamo la programmazione in memoria dinamica nel caso in cui le dimensioni d'istanza del problema che ci interessa non siano note a tempo di compilazione. Diciamo ad esempio di voler immagazinare una serie di n numeri. Potremmo voler usare una lista, ma a quel punto dovremmo accettare di dover richiedere tutte le nostre letture sulla struttura dati in tempo n. Utilizzando invece un vettore dinamico, avremmo una complessità di $O(1)$ per ogni accesso. Tutte queste strutture dati possono essere trovate nell'implementazione della libreria standard, ovvero la:

Standard Template Library (STL)
La libreria STL definisce una serie di container, ovvero strutture dati predefinite che possono venire utilizzate nel nostro codice. Vediamo per esempio un vettore definito attraverso la STL:

```
1 vector<int> stlArray
```

qui vector indica il tipo di container, lal voce <int> definisce il tipo di dati immagazinati nel container, e stlArray il nome del vettore. Per aggiungere un elemento al vettore, potremo ad'esempio;

```
1 stlArray.push_back(val);
```

chiamando la funzione membro push_back sulla stlArray (altro non è che un'istanza di classe) con argomento val. Per leggere un valore dal vettore potremmo usare la ridefinizione dell'operatore []:

```
stlArray[index];
```

potremo ottenere iteratori sulla testa e sulla coda del vettore con:

```
stlArray.begin();
stlArray.end();
```

ed ottenerne la dimensione con:

```
stlArray.size();
```

Riguardo agli algoritmi di ordinamento visti finora, la libreria STL fornisce una funzione:

```
sort(stlArray.begin(), stlArray.end());
```

dove i due argomenti sono iteratori generici tra i quali verrà effettuato l'ordinamento. L'algoritmo dell'ordinamento è implementation-dependant, cioò dipende dall'implementazione usata, ma assicura sempre una complessità di $O(n \log n)$.

Ulteriori informazioni su qualsiasi componente della STL si può trovare su siti come:
`https://cplusplus.com/reference`