Appunti Algoritmi e Strutture Dati

Luca Seggiani

19 Marzo 2024

1 Mergesort su vettori non in-place

Implementiamo un'algoritmo di ordinamento mergesort sui vettori non in-place, ovvero che sfrutta un vettore ausiliario implementato come vettore STL (per l'ausilio della funzione push_back). Il corpo del mergesort sarà:

```
void mergeSort(int arr[], int beg, int end) {
    if(beg + 1 < end) {
        int mid = (beg + end) / 2;
        mergeSort(arr, beg, mid);
        mergeSort(arr, mid, end);
        merge(arr, beg, mid, end);
    }
}
```

dove la chiamata ricorsiva è fatta sulle due sottoarray di dimensione dimezzata dell'array di partenza. Notare che la suddivisione si ferma giunti ad array di dimensione monadica. La merge è implementata come segue:

```
void merge(int arr[], int beg, int mid, int end) {
    int iS = beg;
    int iD = mid;

    vector<int> temp;

    while(true) {
        if(arr[iS] <= arr[iD]) {
            temp.push_back(arr[iS++]);
            if(iS >= mid) {
                while(iD < end) temp.push_back(arr[iD++]);
                break;
            }
        } else {
            temp.push_back(arr[iD++]);
            if(iD >= end) {
                while(iS < mid) temp.push_back(arr[iS++]);
```

```
                    break;
                }
            }
        }

        for(int i = 0; i < temp.size(); i++) {
            arr[i + beg] = temp[i];
        }

}
```

Si inizializzano due indici, iS e iD, che partono rispettivamente dall'inizio e dal punto medio della sottoarray. Da qui in poi si scorrono entrambi gli indici, selezionando ogni volta l'elemento minore, finchè uno dei due non sfora il suo bound (il punto medio per iS, la fine della sottoarray per iD). A questo punto si effettua l'unrolling completo della parte di sottoarray rimanente attraverso l'indice rimasto libero. Infine si copiano i valori selezionati e ordinati, che sono stati scritti su un vettore dinamico, sul vettore di partenza a partire dalla posizione d'inizio della sottoarray. Una tipica esecuzione di questo algoritmo, su dimensione di istanza $n = 15$, sarà:

```
111 86  23  33  21  21  95  92  15  4 49  65  43  9 15
11  23  86  33  21  21  95  92  15  4 49  65  43  9 15
11  23  86  33  21  21  95  92  15  4 49  65  43  9 15
11  23  86  21  33  21  95  92  15  4 49  65  43  9 15
11  23  86  21  33  21  95  92  15  4 49  65  43  9 15
11  23  86  21  21  33  95  92  15  4 49  65  43  9 15
11  21  21  23  33  86  95  92  15  4 49  65  43  9 15
11  21  21  23  33  86  95  15  92  4 49  65  43  9 15
11  21  21  23  33  86  95  15  92  4 49  65  43  9 15
11  21  21  23  33  86  95  4 15  49  92  65  43  9 15
11  21  21  23  33  86  95  4 15  49  92  43  65  9 15
11  21  21  23  33  86  95  4 15  49  92  43  65  9 15
11  21  21  23  33  86  95  4 15  49  92  9 15  43  65
11  21  21  23  33  86  95  4 9 15  15  43  49  65  92
4 9 11  15  15  21  21  23  33  43  49  65  86  92  95
4 9 11  15  15  21  21  23  33  43  49  65  86  92  95
11  23  86  33  21  21  95  92  15  4 49  65  43  9 15
11  23  86  33  21  21  95  92  15  4 49  65  43  9 15
11  23  86  21  33  21  95  92  15  4 49  65  43  9 15
11  23  86  21  33  21  95  92  15  4 49  65  43  9 15
11  23  86  21  21  33  95  92  15  4 49  65  43  9 15
11  21  21  23  33  86  95  92  15  4 49  65  43  9 15
11  21  21  23  33  86  95  15  92  4 49  65  43  9 15
11  21  21  23  33  86  95  15  92  4 49  65  43  9 15
11  21  21  23  33  86  95  4 15  49  92  65  43  9 15
11  21  21  23  33  86  95  4 15  49  92  43  65  9 15
11  21  21  23  33  86  95  4 15  49  92  43  65  9 15
```

```
11  21  21  23  33  86  95  4 15  49  92  9 15  43  65
11  21  21  23  33  86  95  4 9 15  15  43  49  65  92
4 9 11  15  15  21  21  23  33  43  49  65  86  92  95
4 9 11  15  15  21  21  23  33  43  49  65  86  92  95
```

2 Ordinamenti multivalore

Supponiamo di avere vettori formati da elementi con più di un valore su cui è stabilita una relazione d'ordine, come ad esempio:

```
  struct Richiesta {
    int id_;
    int prio_;
    Richiesta(int id, int prio) : id_(id), prio_(prio) {}
  }
```

nella STL, potremo usare la funzione sort sfruttando l'argomento comparatore:

```
sort(first, last comparatore);
```

definiamo ad esempio una funzione di confronto:

```
bool confrontaRichiesta(Richiesta r_1, Richiesta r_2) {
  if (r_1.id_ < r_2.id_) retirm true;
  else if {r_1.id_ == r_2.id_} {
    if(r_1.prio_ > r_2.prio_) return true;
    else return false;
  } else return false;
}
```

potremo adesso passare come argomento la funzione confrontaRichiesta(Richiesta, Richiesta), alla funzione sort() su un qualsiasi vettore di elementi Richiesta, così che venga usata nei confronti dell'ordinamento.

3 Debugging della memoria con Valgrind

Valgrind è un applicativo divenuto praticamente standard per la programmazione in c++, che controlla la corretta gestione della memoria dinamica. Per fare il debugging di un programma attraverso Valgrind, occorrerà prima di tutto compilare il nostro eseguibile abilitando i flag di debugging:

```
g++ -g programma.cpp -o programma
```

e chiamare poi Valgrind sull'eseguibile creato:

```
valgrind programma
```