Appunti Algoritmi e Strutture Dati

Luca Seggiani

9 Aprile 2024

1 Alberi binari di ricerca, implementazione

Vediamo l'implementazione effettiva di un albero binario di ricerca, effettuata in C++ attraverso il meccanismo delle classi:

```
struct Node {
  int value;
  Node* left;
  Node* right;

  Node(int val):
    value(val), left(NULL), right(NULL) {}
};

class BinTree{
  Node root_;

public:
  BinTree() { root_ = NULL; }
  Node* getRoot() { return root_; }
};
```

La insert:

```
void insert(int val) {
  Node* node = new Node(val);
  Node* pre = NULL;
  Node* post = root_;

  while(post != NULL) {
    pre = post;
    if(val <= post->value)
      post = post->left;
    else
      post = post->right;
  }
```

```
13
14    if(pre == nullptr) {
15      root_ = node;
16      return;
17    }
18
19    if(val <= pre->val)
20      pre->right = val;
21    else
22      pre->left = val;
23
24  }
```

Funzioni per la ricerca di minimi e massimi:

```
1  Node* min() {
2    Node* temp = root_;
3    while(temp->left != NULL)
4      temp = temp->left;
5    return temp;
6  }
7
8  Node* max() {
9    Node* temp = root_;
10   while(temp->right != NULL)
11     temp = temp->right;
12   return temp;
13 }
```

Funzioni per la ricerca di la profondità di minimi e massimi:

```
1  int height(Node* tree) {
2    int hLeft;
3    int hRight;
4    if(tree == NULL)
5      return 0;
6    hLeft = height(tree->left);
7    hRight = height(tree->right);
8
9    return(1 + max(hLeft, hRight));
10 }
```

Funzioni di ricerca di nodi:

```
1  Node* search(int val) {
2    Node* temp = root_;
3    while(temp != nullptr) {
4      if(val == temp->value)
5        return temp;
6      if(val <= temp->value)
7        search(temp->left);
8      else
```

```
        search(temp->right);
    }
    return nullptr;
}
```

2 Stringhe STL

La libreria STL definisce nell'header <string> il tipo di dato stringa. Il tipo di dato stringa estende in qualche modo la stringa del comune C, ovvero un'array di tipi carattere terminata da un carattere specifico. Attraverso l'incapsulamento di questo tipo di dati, si possono avere diverse comodità, come ad esempio il conto della lunghezza della lunghezza della stringa senza necessità di scorrimenti, o funzioni membro come:

```
string stringa = "Il contrario di comunismo";
string stringa_2 = " e' egoismo";
string stringa_completa = stringa + stringa_2;
  //stringa_completa = "Il contrario di comunismo e' egoismo"
stringa.find("comunismo");
  //puntatore alla prima occorrenza di "comunismo", senno' :
    npos
stringa_2.compare("egoismo");
...
```