12 Alberi

12.1 Rappresentazione

12.1.1 Array

Per rappresentare un albero *binario* di profondità d possiamo utilizzare un **array** di dimensione $2^{d+1}-1$. Questa scelta può portare a dei vantaggi e svantaggi:

- **Vantaggi**:
  - Accesso diretto ai nodi
- **Svantaggi**:
  - L'*altezza* dell'albero deve essere nota
  - Spreco di memoria
  - *Inserimento* e *cancellazione* sono operazioni complicate

Per questi motivi gli array si usano raramente per la rappresentazione di alberi.

12.1.2 Liste

Il modo più usato per la rappresentazione di alberi è quello delle **liste**, codificandoli come segue:

```
1  struct node {
2    int data;
3    struct node *left;
4    struct node *right;
5  }
```

Listing 21: Alberi con liste

Questa scelta ci porta a vantaggi e svantaggi:

- **Vantaggi**:
  - L'*altezza* dell'albero non deve essere nota
  - Nessuno spreco di memoria
  - *Inserimento* e *cancellazione* sono operazioni facili
- **Svantaggi**:
  - Mancanza di accesso diretto ai nodi
  - Memoria aggiuntiva per memorizzare figlio destro e sinistro

12.2 Visitare

Possiamo effettuare l'operazione di **visita** su un albero binario in tre modi diversi. Tutti questi algoritmi avranno **complessità** $O(n)$.

12.2.1 Anticipata

```
1  Anticipata(x):
2    if x != NULL
3      print(x.key)
4      Anticipata(x.left)
5      Anticipata(x.right)
```

Listing 22: Alberi con liste

Esempio 12.1.

12.2.2 Posticipata

```
1  Posticipata(x):
2    if x != NULL
3      Posticipata(x.left)
4      Posticipata(x.right)
5      print(x.key)
```

Listing 23: Alberi con liste

Esempio 12.2.

12.2.3 Simmetrica

```
1  Simmetrica(x):
2    if x != NULL
3      Simmetrica(x.left)
4      print(x.key)
5      Simmetrica(x.right)
```

Listing 24: Alberi con liste

Esempio 12.3.

12.3 Albero binario di ricerca

Un caso particolare di albero binario è un albero binario di ricerca. Questo rispecchia le seguenti proprietà, dato un nodo x, applicate ricorsivamente:

- $x.left.key \leq x.key$, ovvero tutti i nodi alla sinistra sono i minori di x
- $x.right.key \geq x.key$, ovvero tutti i nodi alla destra sono i maggiori di x

Note 12.3.1. *Per effettuare la stampa **ordinata** degli elementi dobbiamo utilizzare la visita **simmetrica**.*

12.3.1 Ricerca

Algoritmo ricorsivo per la ricerca di un elemento:

```
1  RicercaABR_R(x, k)
2    if x == NULL OR k == x.key
3      return x
4    if k < x.key
5      return RicercaABR_R(x.left, k)
6    else return RicercaABR_R(x.right, k)
```

Listing 25: Ricerca ricorsiva

Algoritmi per la ricerca del valore minimo e massimo di un albero:

```
1  RicercaMIN_I(x)
2    while x.left != NULL
3      x = x.left
4    return x
```

Listing 26: Ricerca del minimo

```
1  RicercaMAX_I(x)
2    while x.right != NULL
3      x = x.right
4    return x
```

Listing 27: Ricerca del massimo