Alberi

Le liste erano casi particolare di alberi.

Tipi di liste :

- Lista normale
- Lista doppiamente collegata

Con un minimo sforzo posso passare a strutture dati differenti come la lista doppiamente collegata in modo circolare.

Lista doppiamente collegata -> n nodi, 2n collegamenti

Albero Binario -> n nodi, n-1 collegamenti

> n nodi, log2(n) livelli, n/2 foglie

Proprietà interessanti di un albero binario:

- Ogni nodo punta al massimo a 2 figli
- Non punta mai al padre (non vado mai indietro, sono nodi nuovi)
- Ogni nodo ha al massimo 1 genitore
- Ogni nodo è aciclico
- Rispetto alla profondità dell'alboro vengono allocati un numero esponenziale di nodi
- Se ho a disposizione n nodi, allora ho a disposizione log2(n) livelli -> algoritmo di ricerca O(log n)

Un albero è comlpeto se l'ultimo livello del nostro albero è pieno oppure l'ultimo livello è parzialmente riempito (da Sx a Dx) se la struttura non è piena.

Se inseriamo solo a Sx o a Dx l'albero collassa in una lista.

Costo visita in un albero binario:

- Esplorare tutti (n) i nodi O(n)

Visita iterativa: inserisco in una struttura d'appoggio i figli

Visite degli alberi

Tipi di visita:

- Pre ordine
- in ordine
- Post ordine

Varia la posizone della stampa prima delle chiamate ricorsive

Preordine

```
rec(n){

    print(n)

    rec(n->L)
    rec(n->R)
}
```

In ordine

```
rec(n){

    rec(n->L)

    print(n)

    rec(n->R)
}
```

Si stampa il nodo più prodonfo che trovo scendendo a Sx, appena ho una deviazione a Dx (o non ci sono figli di Sx) stampo

C'è una certa ambiguità perchè si possono avere le medesime stampe con alberi differenti. Serve introdurre le *parentesi* per esempio quando si parla di espressioni aritmetiche per creare stampe uniche.

Post ordine

```
rec(n){

    rec(n->L)
    rec(n->R)

    print(n)
}
```

Nota bene: In questo caso ho preferito mettere L prima di R, ma non c'è nulla che mi vieti di fare il contrario

Utilizzo delle *parentesi* con una visita In Ordine

```
rec(n){

    print ('(')

    rec(n->L)
    print(n)
```

```
    rec(n->R)

    print (')')
}
```

Visita di Eulero

```
rec(n){

    print (n)

    rec(n->L)
    print(n)
    rec(n->R)

    print (n)
}
```