Alberi

DEF

Sono grafi diretti, in cui i dati sono contenuti nei dati e, le relazioni gerarchiche sono definite dagli archi.

Siano u e v nodi:

- Grado di un nodo: numero di figli.
- u antenato di v se u è raggiungibile da v risalendo di padre in padre.
- v discendente di u se u è antenato di v.

Come rappresentare gli Alberi

1. Vettore dei padri

Per un albero con n nodi uso un array con almeno n elementi. Una generica cella i contiene una coppia (info, parent) dove:

- info: contenuto informativo del nodo i
- parent: indice dell'nodo padre di i

2. Vettore posizionale

E' usato per alberi d-ari (quasi) completi. I nodi sono arrangiati nell'array per livelli.

Se gli indici partono da 0, allora: - j-esimo figlio di i è in posizione $(d \cdot i) + j$. - il padre di i è in posizione $\frac{i-1}{d}$.

Se gli indici partono da 1, allora: - j-esimo figlio di i è in posizione $(d \cdot (i-1)) + j + 1$. - il padre di i è in posizione $\frac{i-2}{d} + 1$.

In questo modo, dato un nodo i, in tempo $O(1)$ è possibile trovare tutti i suoi figli e, raggiungere il padre.

3. Rappresentazione con puntatori ai figli (nodi con numero limitato di figli).
4. Rappresentazione con liste di puntatori ai figli (nodi con numero arbitrario di figli).
5. Rappresentazione di tipo primo figlio-fratello successivo (nodi con numero arbitrario di figli).

Visite di alberi

Sono algoritmi che consentono l'accesso sistematico ai nodi e agli archi di un albero. Gli algoritmi di visita si distinguono in base al particolare ordine di accesso ai nodi.

1. Visita in profondità (DFS)

L'algoritmo di visita in profondità, parte dalla radice r e procede visitando nodi di figlio in figlio fino a raggiungere una foglia. Retrocede poi al antenato che ha ancora figli non visitati (se esiste) e ripete il procedimento a partire da uno di quei figli.

```
1   DFS(nodo r)
2       Pila St
3       St.push(r)
4       while (not St.isEmpty()) do
5           u = S.pop()
6           if(u != NULL) then
7                visita il nodo u
8                S.push(figlio_des(u))
9               S.push(figlio_sin(u))
```

Alternativa ricorsiva

```
1   DFS_recursive(nodo r)
2       if(r != NULL) then
3           visita il nodo r
4           DFS_recursive(figlio_sin(r))
5           DFS_recursive(figlio_des(r))
```

Complessità Temporale: $T(n) = O(n)$

I base al ordine in cui mettiamo le righe 3, 4, 5, ottendendo: - Visita in preordine: 3 - 4 - 5 (radice, sinistra, destra)
- Visita simmetrica: 4 - 3 - 5 (sinistra, radice, destra)
- Visita in postordine: 4 - 5 - 3 (sinistra, destra, radice)

2. Visita in ampiezza (BFS)

L'algoritmo di visita in ampiezza parte dalla radice r e procede visitando nodi per livelli successivi. Un nodo sul livello i può essere visitato solo se tutti i nodi sullivello $i-1$ sono stati visitati.

```
BFS(nodo r)
    Queue Q
    Q.enqueue(r)
    while(not Q.isEmpty()) do
        u = Q.dequeue()
        if(u != NULL) then
            visita il nodo u
            Q.enqueue(figlio_sin(u))
            Q.enqueue(figlio_des(u))
```

Complessità Temporale: $T(n) = O(n)$.

Alcuni utilizzi degli algoritmi di visita

1. Calcolo dell'altezza

```
CalcolaAltezza(nodo r)
    if (r == NULL) then return -1
    sin = CalcolaAltezza(figlio_sin(r))
    des = CalcolaAltezza(figlio_des(r))

    return 1 + max{sin, des}
```

Complessità Temporale $= T(n) = O(n)$.

2. Calcolo del numero di foglie

```
CalcoloFoglie(nodo r)
    if (r == NULL) then return 0
    if (r è foglia) return 1
    return CalcoloFoglie(figlio_sin(sin)) + CalcoloFoglie(figlio_des(r))
```

3. Calcolo del grado medio dei nodi

```
CalcolaMedia(nodo r)
    n_foglie = CalcoloFoglie(r)
    n_padri = n_nodi - n_foglie
    tot_gradi = CalcolaGradi(r)
    media = tot_gradi / (n_padri)
```

```
CalcolaGradi(nodo r)
    if (r == NULL) then return 0
    if (r è foglia) then return 0
    sin = CalcolaGradi(figlio_sin(r))
    des = CalcolaGradi(figlio_des(r))
    return sin + des + figli_r
```

4. Verificare se esiste un nodo che abbia un dato informativo

```
Search(nodo r, val)
    if(r == NULL) then return NULL
    if(chiave(r) == val) then return r
    sin = Search(figlio_sin(r), val)
    if(sin != NULL) then return sin
    return Search(figlio_des(r), val)
```