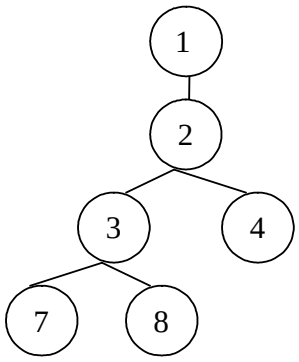
ALGORITMICA - Verifica del 17/12/2003
SOLUZIONI

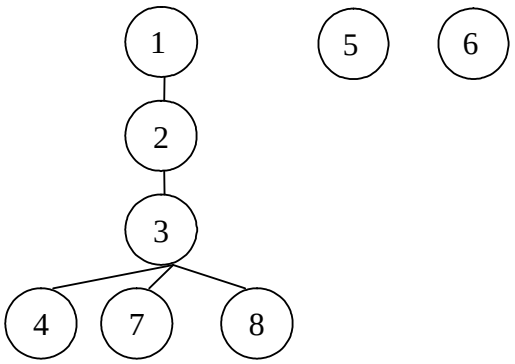
Soluzione Esercizio 1

Si ricordi che le Liste di adiacenza sono ordinate in modo crescente per destinazione.
La visita BFS a partire dal nodo 1 produce il seguente vettore:
d(1)=0, d(2)=1, d(3)=2, d(4)=2, d(5) = +infty, d(6)=+infty, d(7)=3, d(8)=3,
e l'ordine in cui i vertici sono scoperti è: **1, 2, 3, 4, 7, 8**
La visita DFS produce il seguente vettore degli intervalli (scoperta/fine visita):
I(1)=[1,12], I(2)=[2,11], I(3)=[3,10], I(4)=[4,5], I(5)=[13,14], I(6)=[15,16], I(7)=[6,7], I(8)=[8,9],
e l'ordine in cui i vertici sono scoperti è : **1, 2, 3, 4, 7, 8, 5, 6.**

Albero BFS



Foresta DFS



Soluzione Esercizio 2

Progettiamo un algoritmo che restituisce una tripla di valori **<n_nodi, n_rossi, n_neri>** tale che
n_nodi = numero di nodi dell'albero che presentano un ugual numero di discendenti rossi e neri (tra quelli fino a ora esaminati nella visita);
n_rossi = numero di nodi rossi che discendono dal nodo correntemente visitato;
n_neri = numero di nodi neri che discendono dal nodo correntemente visitato.

A questo punto progettiamo una procedura ricorsiva definita per il nodo generico **u**:

```
Conta(u)
 {
        if (u == NIL) return <0,0,0>;

        < n_nodi_left , n_rossi_left, n_neri_left > = Conta(left[u]);
        < n_nodi_right , n_rossi_right, n_neri_right > = Conta(right[u]);

        n_rossi = n_rossi_left + n_rossi_right;
        n_neri = n_neri_left + n_neri_right;
        if (colore[u] == nero) { n_neri ++;} else { n_rossi ++; }

        n_nodi = n_nodi_left + n_nodi_right;
        if (n_neri == n_rossi) n_nodi ++;

        return <n_nodi, n_rossi, n_neri>;
}
```

La complessità è *O(n)*, con *n* numero di nodi dell'albero visitato. La procedura può essere invocata sull'albero T come Conta(root[T]).

Soluzione Esercizio 3

Specializziamo la procedura Configurazioni vista in classe come segue:

```
Insieme_Indipendente(G,K,h)
{
        for i = 0 to 1
        {
            IS[h]=i;
            if (h==n)  Controlla(G,K,IS);
            else  Insieme_Indipendente(G,K,h+1);
        }
}

Controlla(G,K,IS)
{
        card = 0;
        for j = 1 to n
                card = card + IS[j];
        if (card == K)
        {       // i K vertici selezionati sono inseriti in un insieme Set:
                for each v in V
                        { if (IS[v] == 1)   Insert(Set,v); }
                // per ogni coppia u,v di vertici in Set si controlla se u e v sono adiacenti:
                for each u,v in Set
                        { if ((u,v) in E) return; }
                // se si supera il controllo precedente (quindi nessuna coppia di vertici in Set
                // è adiacente) si termina il calcolo: esiste una soluzione!
                success;
        }
}
```

Per verificare se il grafo G contiene un insieme indipendente di almeno K vertici, si effettua la chiamata iniziale Insieme_Indipendente(G,K,1).

La complessità è $O(2^n + n^k \ (n + K^2))$ se il grafo è rappresentato tramite matrice di adiacenza. Il primo termine additivo tiene conto del costo per generare tutte le configurazioni su n elementi (vertici del grafo G), il secondo termine tiene conto del costo della verifica che l'insieme Set sia indipendente. Questo costo è $O(n + K^2)$ e viene speso soltanto sui sottoinsiemi di K vertici, che sono appunto $O(n^k)$.

Il problema proposto appartiene alla classe NP. Il certificato può essere visto come un sottoinsieme dei vertici di G, passati al verificatore come vettore binario IS tale che IS[j]=1 se il vertice j appartiene al certificato. A questo punto l'algoritmo di verifica si comporta come segue:

```
Verifica_IS(G,K,IS)
{
        card = 0;
        for j = 1 to n
                card = card + IS[j];
        if (card != K) return false;
        // card = K
        for each v in V
                { if (IS[v] == 1)   Insert(Set,v); }
        for each u,v in IS_Set
                { if ((u,v) in E)  return false;}
        return true;
}
```

Il costo computazionale della verifica risulta $O(n + K^2) = O(n^2)$, quindi polinomiale nella dimensione del grafo G.