Lezione 31

Tecnica Backtracking

Il Backtracking è un modo sistematico di esplorare **tutte** le possibili configurazioni (soluzioni) di un dato problema algoritmico.

Il Backtracking è un modo sistematico di esplorare **tutte** le possibili configurazioni (soluzioni) di un dato problema algoritmico.

Queste configurazioni ad esempio possono essere tutti i possibili ordinamenti di un insieme di oggetti (permutazioni) o tutte le possibili sottocollezioni di questi oggetti (sottoinsiemi).

Il Backtracking è un modo sistematico di esplorare **tutte** le possibili configurazioni (soluzioni) di un dato problema algoritmico.

Queste configurazioni ad esempio possono essere tutti i possibili ordinamenti di un insieme di oggetti (permutazioni) o tutte le possibili sottocollezioni di questi oggetti (sottoinsiemi).

Altre applicazioni possono richiedere di enumerare tutti i cammini tra due nodi di un grafo o tutti gli alberi di copertura di un grafo etc. etc.

Il Backtracking è un modo sistematico di esplorare **tutte** le possibili configurazioni (soluzioni) di un dato problema algoritmico.

Queste configurazioni ad esempio possono essere tutti i possibili ordinamenti di un insieme di oggetti (permutazioni) o tutte le possibili sottocollezioni di questi oggetti (sottoinsiemi).

Altre applicazioni possono richiedere di enumerare tutti i cammini tra due nodi di un grafo o tutti gli alberi di copertura di un grafo etc. etc.

Ciò che questi problemi hanno in comune è che bisogna generare ciascuna delle possibili configurazioni esattamente una volta.

Il Backtracking è un modo sistematico di esplorare **tutte** le possibili configurazioni (soluzioni) di un dato problema algoritmico.

Queste configurazioni ad esempio possono essere tutti i possibili ordinamenti di un insieme di oggetti (permutazioni) o tutte le possibili sottocollezioni di questi oggetti (sottoinsiemi).

Altre applicazioni possono richiedere di enumerare tutti i cammini tra due nodi di un grafo o tutti gli alberi di copertura di un grafo etc. etc.

Ciò che questi problemi hanno in comune è che bisogna generare ciascuna delle possibili configurazioni esattamente una volta.

Per evitare perdite o replicazioni di configurazioni bisogna definire un ordine sistematico di generazione tra le possibili configurazioni.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$,

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e cerchiamo di estendere la soluzione parziale con un elemento $a_{k+1} \in S_{k+1}$.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e cerchiamo di estendere la soluzione parziale con un elemento $a_{k+1} \in S_{k+1}$.

Fin quando l'estensione genera una soluzione parziale continuiamo ad estenderla.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e cerchiamo di estendere la soluzione parziale con un elemento $a_{k+1} \in S_{k+1}$.

Fin quando l'estensione genera una soluzione parziale continuiamo ad estenderla.

S_{k+1} potrebbe anche essere vuoto,

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e cerchiamo di estendere la soluzione parziale con un elemento $a_{k+1} \in S_{k+1}$.

Fin quando l'estensione genera una soluzione parziale continuiamo ad estenderla.

S_{k+1} potrebbe anche essere vuoto, ciò significa che non c'è modo di estendere la corrente soluzione parziale.

Rappresentiamo le configurazioni (soluzioni) con un vettore $SOL = (a_1, a_2, ..., a_n)$, dove ciascun elemento a_i va selezionato da un insieme ordinato S_i di possibili candidati per la posizione i.

La procedura di ricerca e generazione delle configurazioni procede facendo crescere le soluzioni di un elemento alla volta.

Ad ogni passo della ricerca avremo costruito una soluzione parziale con elementi fissati per i primi k elementi del vettore, dove $k \leq n$. Da questa soluzione parziale $(a_1, a_2, ..., a_k)$ costruiamo l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e cerchiamo di estendere la soluzione parziale con un elemento $a_{k+1} \in S_{k+1}$.

Fin quando l'estensione genera una soluzione parziale continuiamo ad estenderla.

S_{k+1} potrebbe anche essere vuoto, ciò significa che non c'è modo di estendere la corrente soluzione parziale. Quando ciò accade bisogna fare un passo indietro (vale a dire backtrack) e rimpiazzare l'ultimo elemento a_k nella soluzione con un altro candidato in $a'_k \neq a_k \in S_k$.

Il Backtracking costruisce un albero

Il Backtracking costruisce un albero (albero delle soluzioni)

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni.

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
     calcola S_{k+1}
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
      calcola S_{k+1}
      WHILE S_{k+1} ≠ ∅ {
        a_{k+1} ← un elemento di S_{k+1},
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
      calcola S_{k+1}
      WHILE S_{k+1} ≠ ∅ {
        a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
      calcola S_{k+1}
      WHILE S_{k+1} ≠ ∅ {
        a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
        S_{k+1} ← S_{k+1} − {a_{k+1}}
```

Il Backtracking costruisce un albero (albero delle soluzioni) dove ciascun vertice interno x è una soluzione parziale e c'è un arco che va dal vertice x al vertice y se il vertice y è stato creato estendendo la soluzione parziale del vertice x.

Le foglie dell'albero sono le soluzioni. L' albero delle soluzioni offre un modo alternativo di pensare al backtracking in quanto il processo di costruzione delle soluzioni corrisponde esattamente a quello di effettuare una visita in profondità dell'albero.
Questo modo di vedere il Backtracking come visita in profondità di un albero ci conduce alla seguente procedura ricorsiva generale:

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
       a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
       S_{k+1} ← S_{k+1} − {a_{k+1}}
       BACKTRACKING(SOL, k + 1)
     } }
```

Per enumerare tutte le soluzioni (che stanno sulle foglie)

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali,

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali, tuttavia la visita in profondità è di gran lunga da preferirsi per quanto riguarda il risparmio della risorsa spazio.

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali, tuttavia la visita in profondità è di gran lunga da preferirsi per quanto riguarda il risparmio della risorsa spazio.

Nella visita in profondità lo stato corrente della ricerca è rappresentato completamente dal cammino che va dalla radice allo stato corrente,

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali, tuttavia la visita in profondità è di gran lunga da preferirsi per quanto riguarda il risparmio della risorsa spazio.

Nella visita in profondità lo stato corrente della ricerca è rappresentato completamente dal cammino che va dalla radice allo stato corrente, il che richiede uno spazio proporzionale alla **profondità** dell'albero.

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali, tuttavia la visita in profondità è di gran lunga da preferirsi per quanto riguarda il risparmio della risorsa spazio.

Nella visita in profondità lo stato corrente della ricerca è rappresentato completamente dal cammino che va dalla radice allo stato corrente, il che richiede uno spazio proporzionale alla **profondità** dell'albero.

Nella ricerca in ampiezza la coda memorizza tutti i nodi al livello corrente, il che richiede uno spazio proporzionale all'**ampiezza** dell'albero.

Per enumerare tutte le soluzioni (che stanno sulle foglie) si sarebbe potuta utilizzare anche una visita in ampiezza dell'albero delle soluzioni parziali, tuttavia la visita in profondità è di gran lunga da preferirsi per quanto riguarda il risparmio della risorsa spazio.

Nella visita in profondità lo stato corrente della ricerca è rappresentato completamente dal cammino che va dalla radice allo stato corrente, il che richiede uno spazio proporzionale alla **profondità** dell'albero.

Nella ricerca in ampiezza la coda memorizza tutti i nodi al livello corrente, il che richiede uno spazio proporzionale all'**ampiezza** dell'albero.

Per la maggior parte dei problemi di interesse l'ampiezza dell'albero delle soluzioni parziali cresce esponenzialmente rispetto all' altezza.

Primo esempio: Generare tutti i sottoinsiemi

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$\emptyset$ $\{s_1\}$ $\{s_2\}$ $\{s_3\}$

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$$\begin{array}{cccc} \emptyset & \{s_1\} & \{s_2\} & \{s_3\} \\ \{s_1, s_2\} & \{s_1, s_3\} & \{s_2, s_3\} & \{s_1, s_2, s_3\} \end{array}$$

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$$\begin{array}{cccc} \emptyset & \{s_1\} & \{s_2\} & \{s_3\} \\ \{s_1, s_2\} & \{s_1, s_3\} & \{s_2, s_3\} & \{s_1, s_2, s_3\} \end{array}$$

Per rappresentare un sottoinsieme degli n oggetti possiamo utilizzare il vettore caratteristico di n elementi dove nell'entrata i-esima troviamo il valore 0 se l'oggetto i-esimo non appartiene al sottoinsieme, il valore 1 altrimenti.

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$\emptyset$ $\{s_1\}$ $\{s_2\}$ $\{s_3\}$
$\{s_1, s_2\}$ $\{s_1, s_3\}$ $\{s_2, s_3\}$ $\{s_1, s_2, s_3\}$

Per rappresentare un sottoinsieme degli n oggetti possiamo utilizzare il vettore caratteristico di n elementi dove nell'entrata i-esima troviamo il valore 0 se l'oggetto i-esimo non appartiene al sottoinsieme, il valore 1 altrimenti.

Esempio:

$(0, 0, 0)$ $(1, 0, 0)$ $(0, 1, 0)$ $(0, 0, 1)$

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$$\begin{array}{cccc} \emptyset & \{s_1\} & \{s_2\} & \{s_3\} \\ \{s_1, s_2\} & \{s_1, s_3\} & \{s_2, s_3\} & \{s_1, s_2, s_3\} \end{array}$$

Per rappresentare un sottoinsieme degli n oggetti possiamo utilizzare il vettore caratteristico di n elementi dove nell'entrata i-esima troviamo il valore 0 se l'oggetto i-esimo non appartiene al sottoinsieme, il valore 1 altrimenti.

Esempio:

$$\begin{array}{cccc} (0,0,0) & (1,0,0) & (0,1,0) & (0,0,1) \\ (1,1,0) & (1,0,1) & (0,1,1) & (1,1,1) \end{array}$$

Primo esempio: Generare tutti i sottoinsiemi

Dato un insieme $S = \{s_1, s_2, \ldots, s_n\}$ vogliamo generare tutti i 2^n sottoinsiemi di S.

Esempio: $S = \{s_1, s_2, s_3\}$, vogliamo generare

$$\begin{array}{cccc} \emptyset & \{s_1\} & \{s_2\} & \{s_3\} \\ \{s_1, s_2\} & \{s_1, s_3\} & \{s_2, s_3\} & \{s_1, s_2, s_3\} \end{array}$$

Per rappresentare un sottoinsieme degli n oggetti possiamo utilizzare il vettore caratteristico di n elementi dove nell'entrata i-esima troviamo il valore 0 se l'oggetto i-esimo non appartiene al sottoinsieme, il valore 1 altrimenti.

Esempio:

$$\begin{array}{cccc} (0,0,0) & (1,0,0) & (0,1,0) & (0,0,1) \\ (1,1,0) & (1,0,1) & (0,1,1) & (1,1,1) \end{array}$$

Quindi generare tutti i sottoinsiemi di $S = \{s_1, s_2, \ldots, s_n\}$ è perfettamente equivalente a generare tutti i 2^n vettori binari di lunghezza n.

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$.

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
```

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
```

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
```

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
```

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
```

Usando il il vettore caratteristico, l'insieme S_k dei candidati per la k-ma posizione è dato dai due elementi $\{0, 1\}$. Lo pseudocodice della procedura è il seguente

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
     }
    }
```

Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```

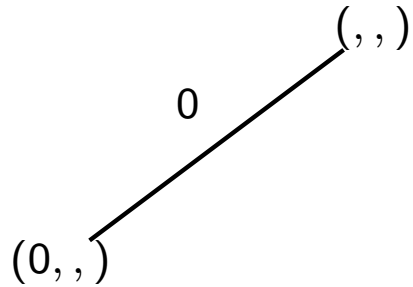
Un esempio

$(, ,)$

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```

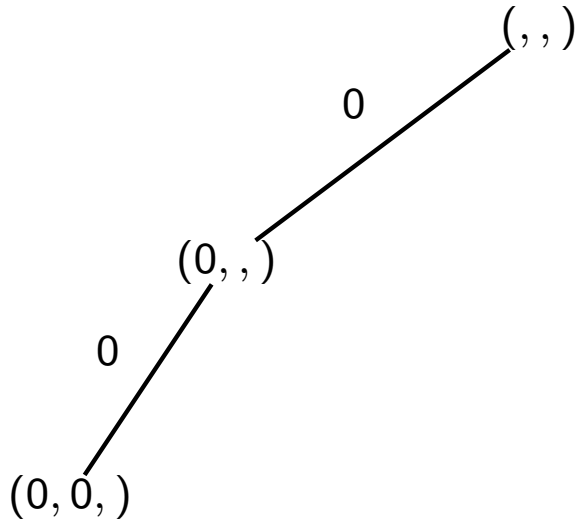
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



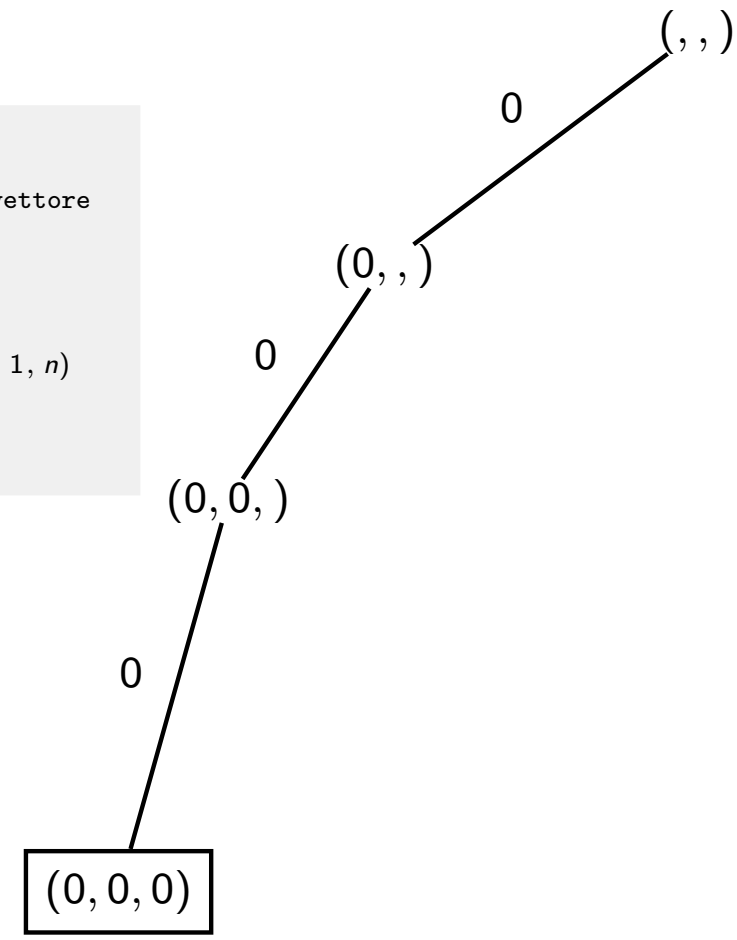
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



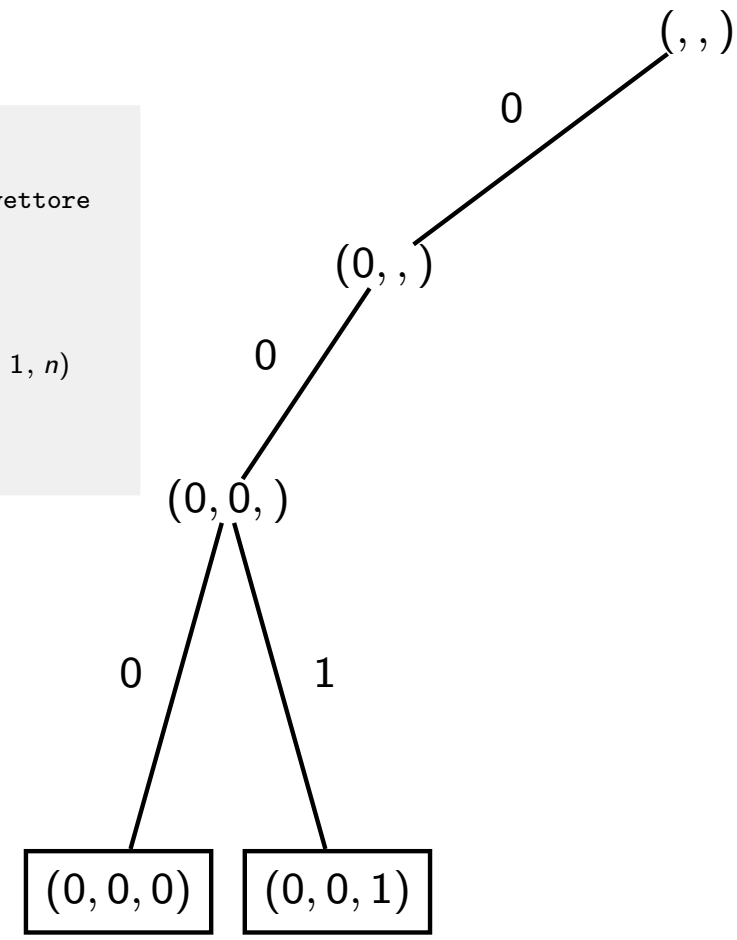
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



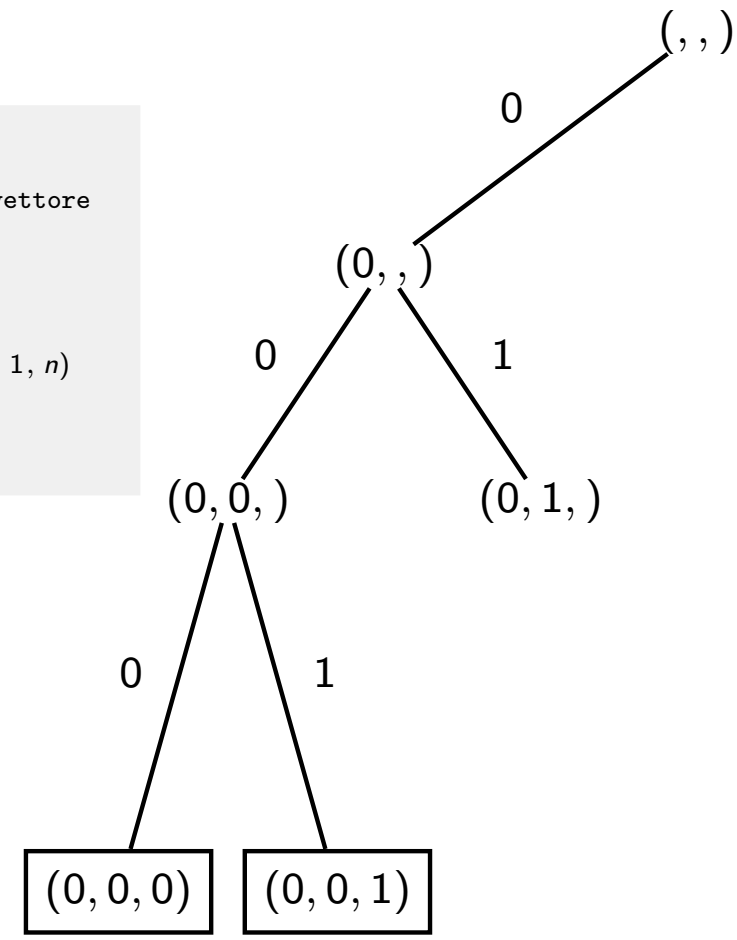
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



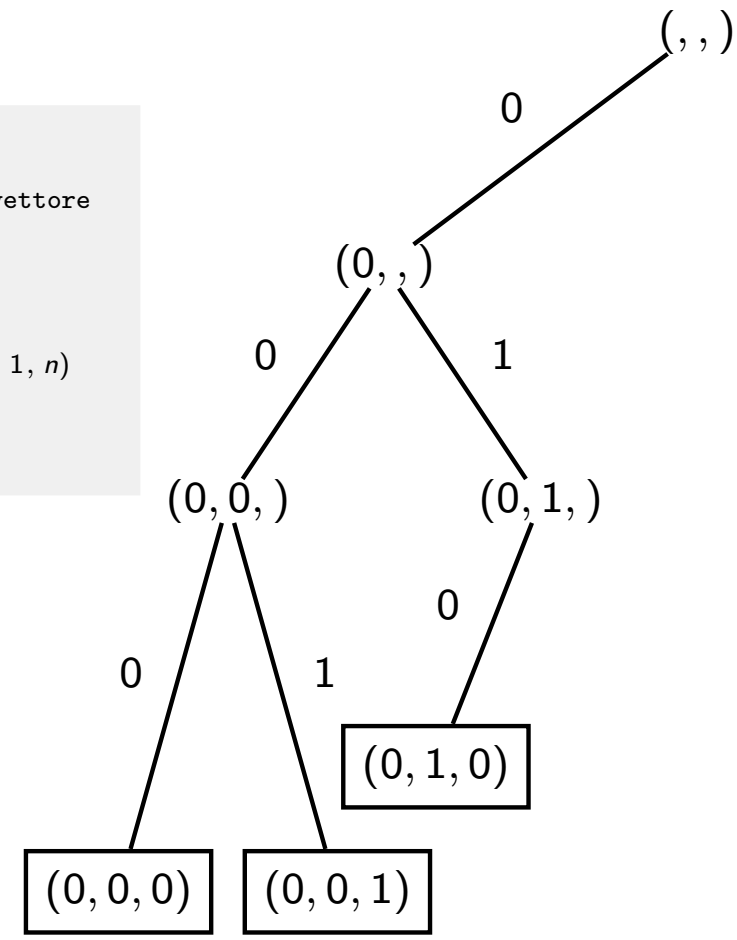
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



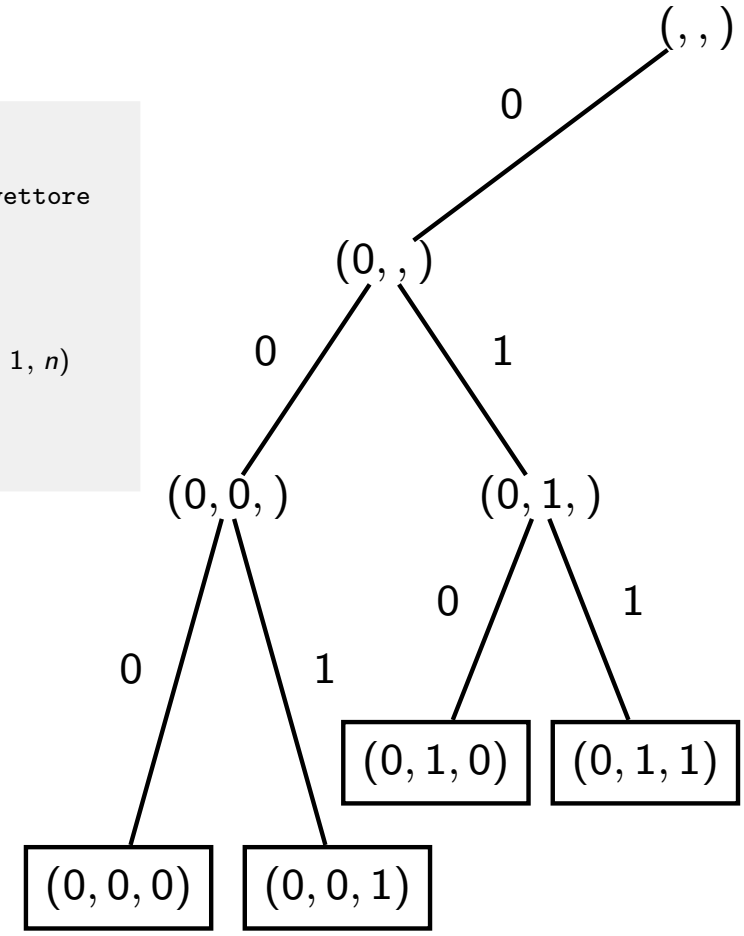
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



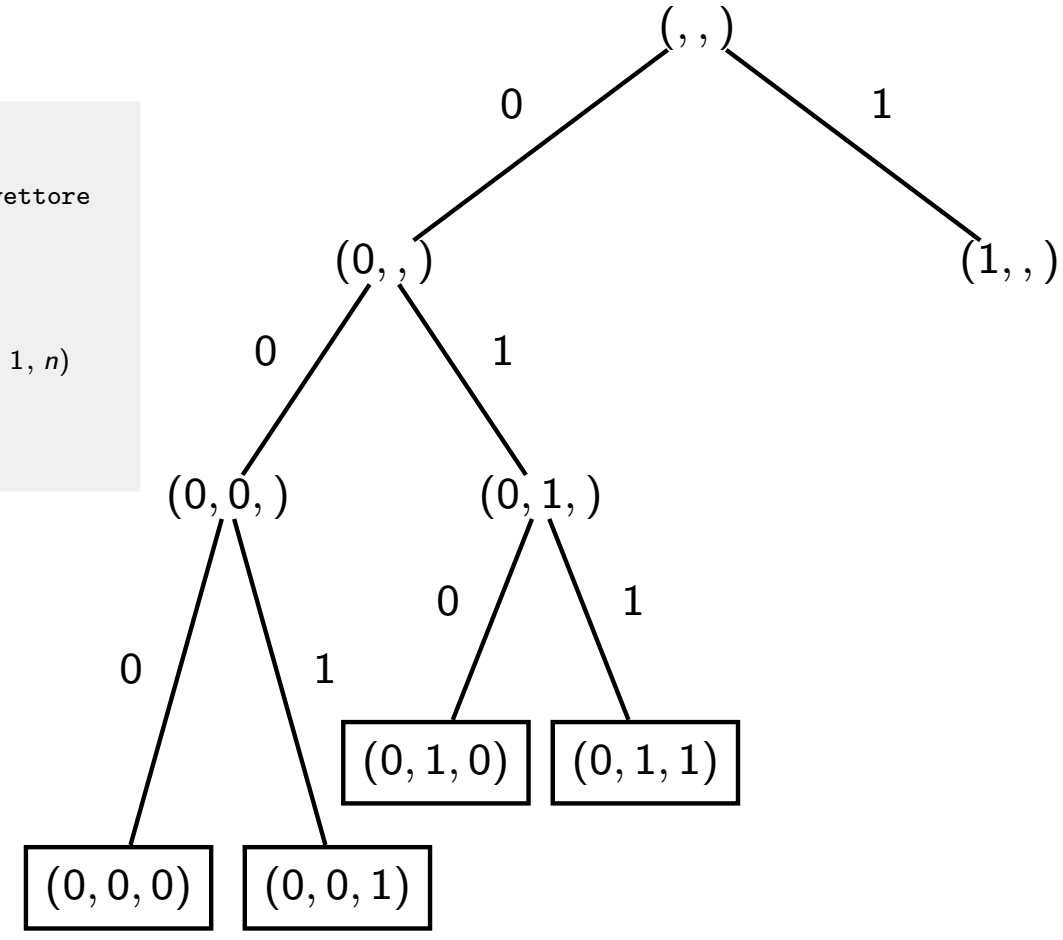
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



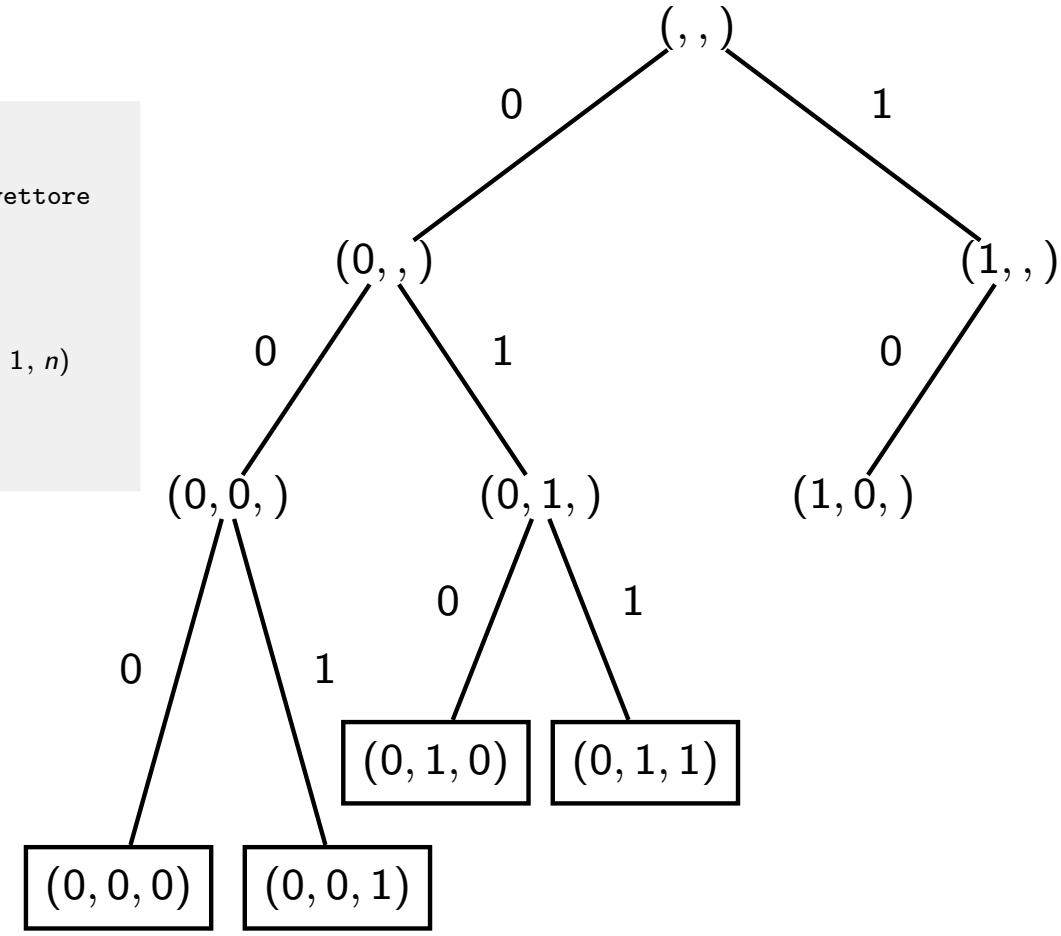
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



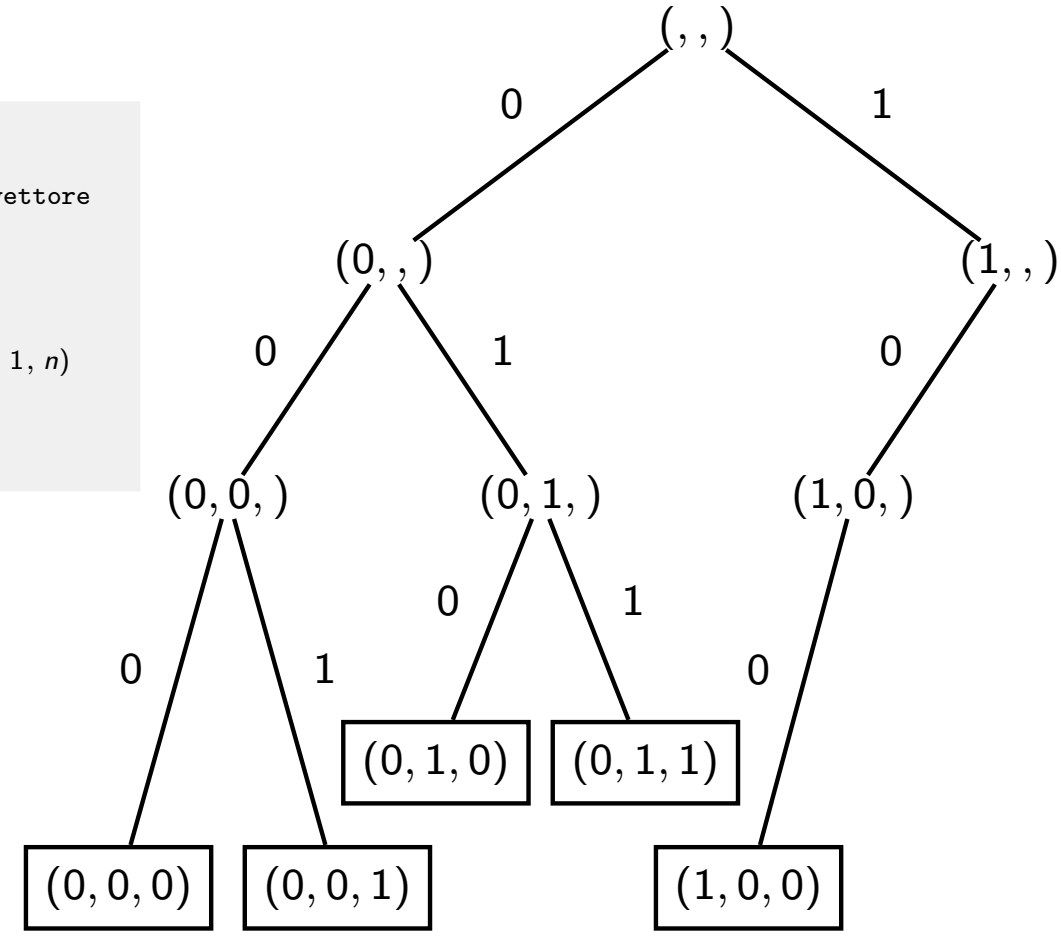
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



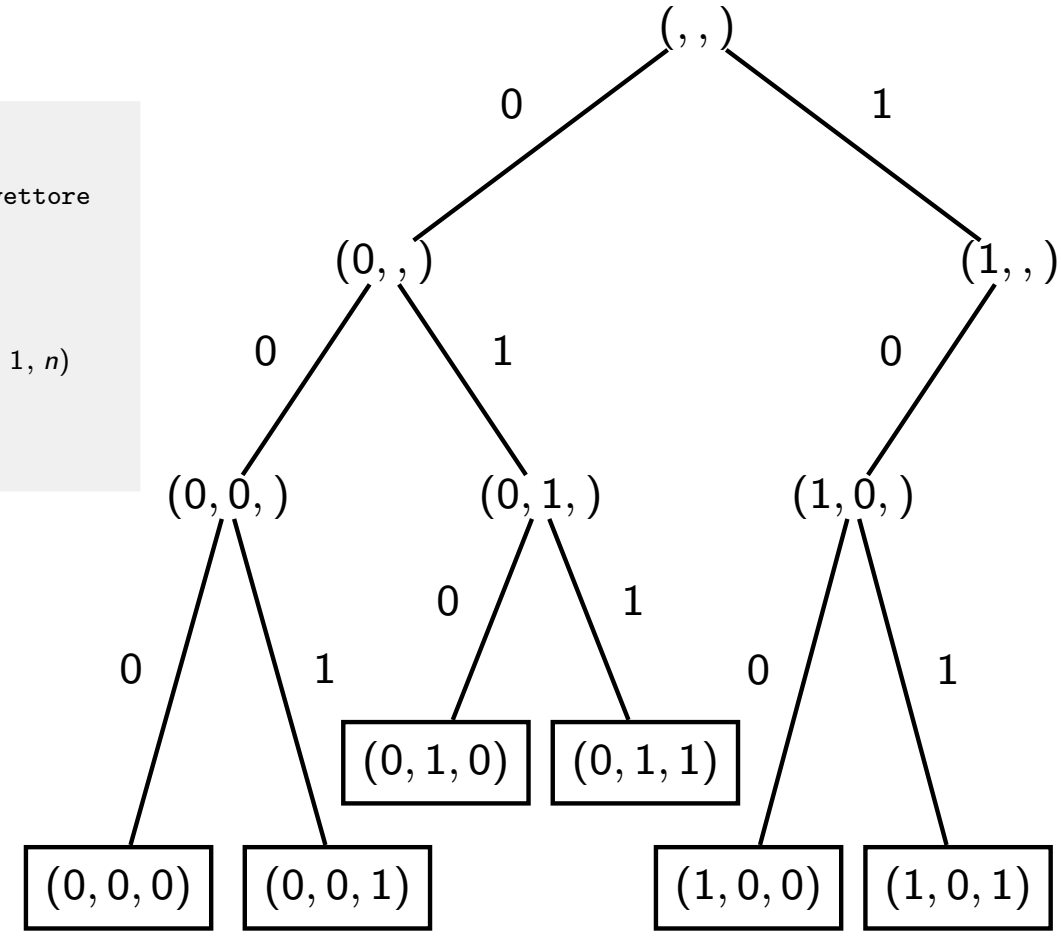
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



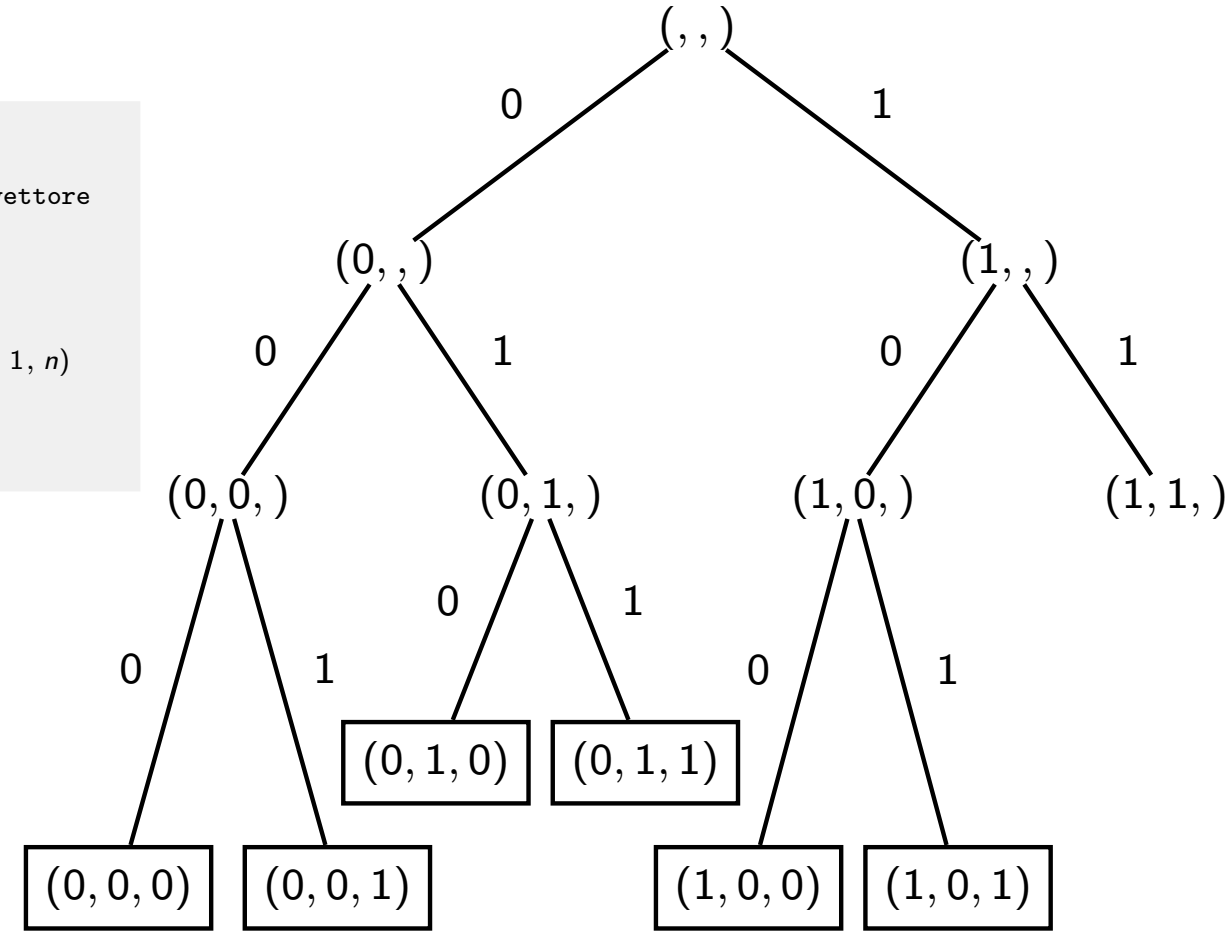
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



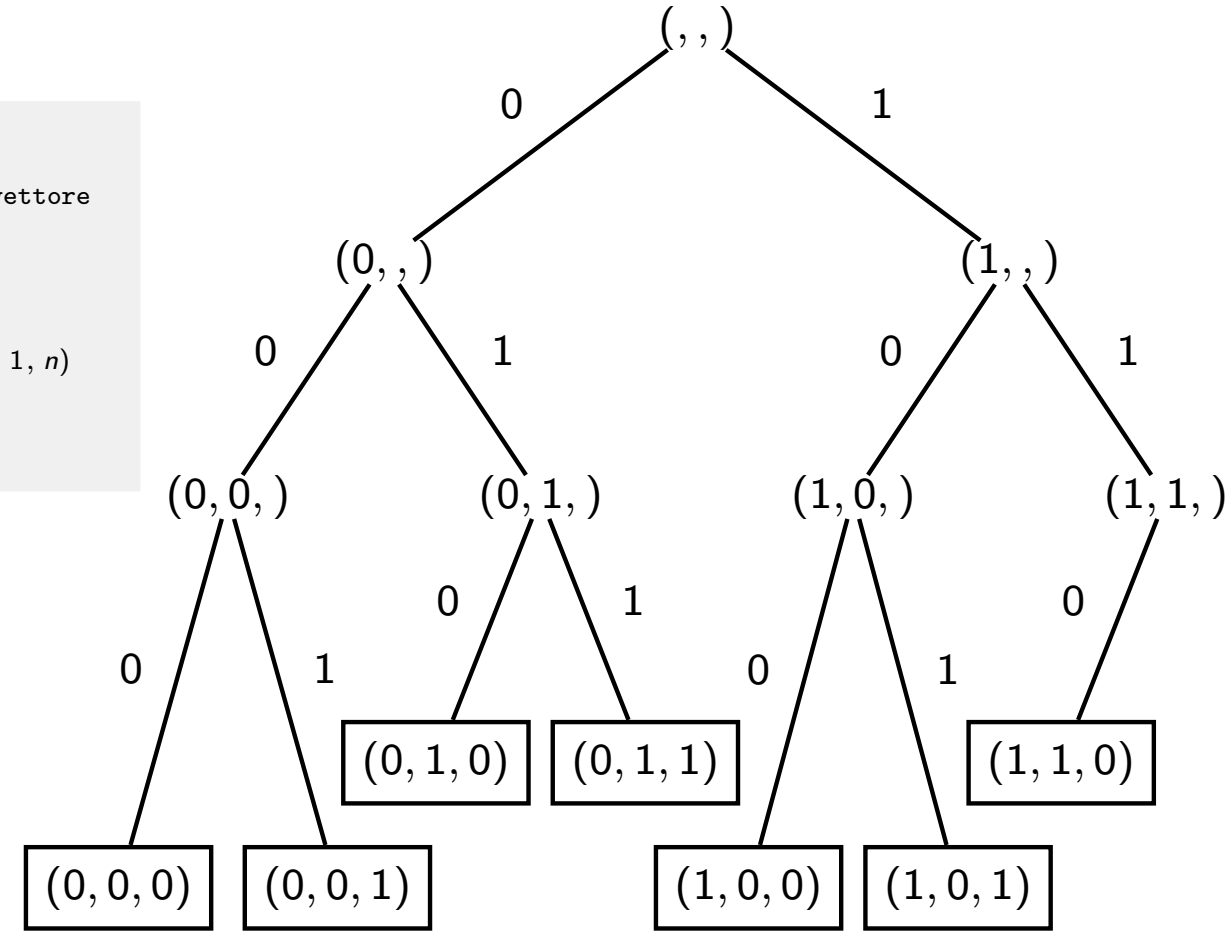
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



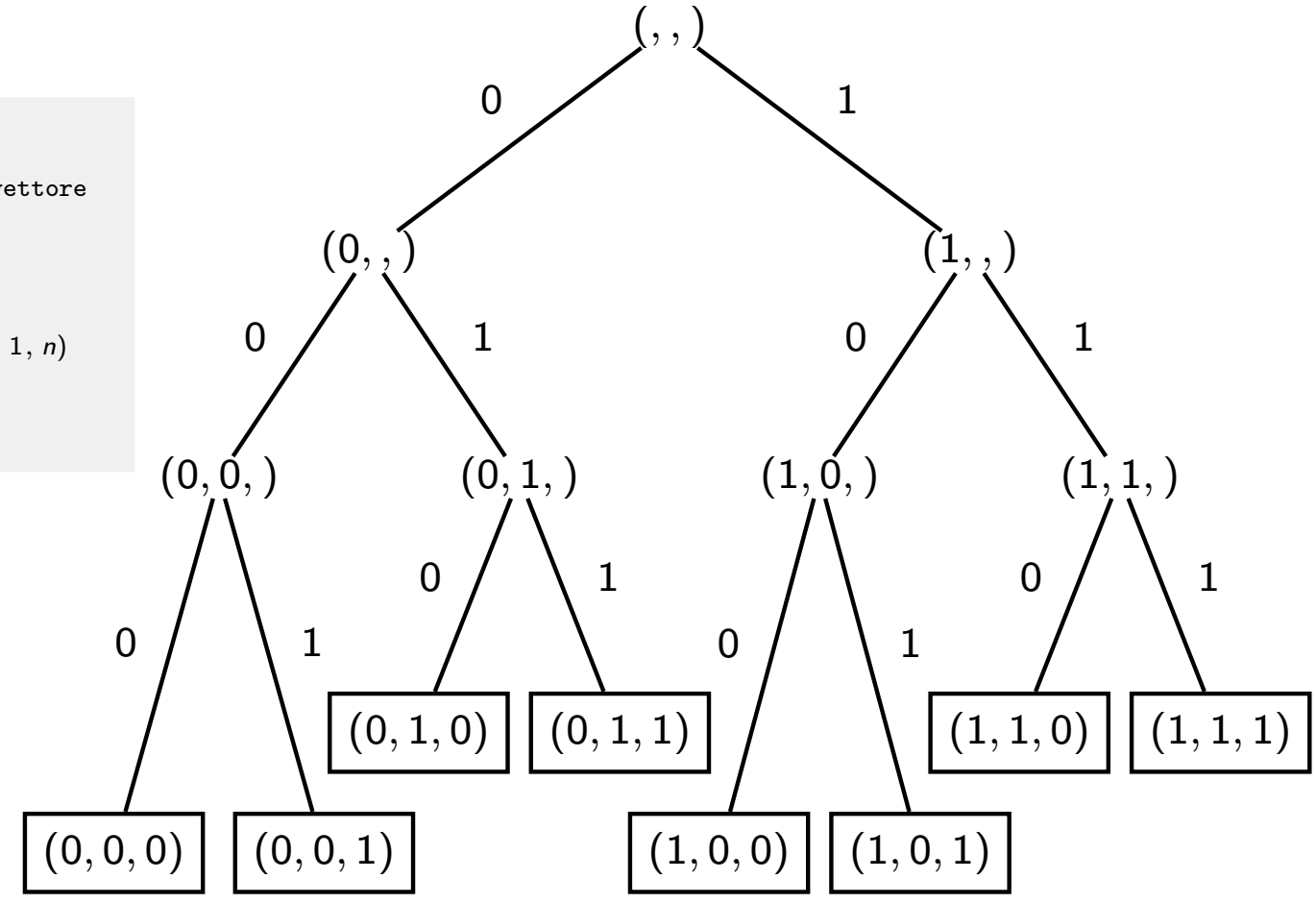
Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



Un esempio

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore
SOL
     } ELSE {
FOR(i=0, i<3, i=i+1){
       SOL[k + 1]=i
       SOTTOINSIEMI(SOL, k + 1, n)
      }
     }
```



Analisi

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
     }
    }
```

La procedura viene invocata da $SOTTOINSIEMI(SOL, 0, n)$.

Analisi

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
     }
    }
```

La procedura viene invocata da $SOTTOINSIEMI(SOL, 0, n)$. L'albero delle soluzioni che ne deriva è un albero binario completo di altezza n in quanto ogni nodo a livello $k < n$ ha esattamente due figli.

Analisi

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
     }
    }
```

La procedura viene invocata da $SOTTOINSIEMI(SOL, 0, n)$. L'albero delle soluzioni che ne deriva è un albero binario completo di altezza n in quanto ogni nodo a livello $k < n$ ha esattamente due figli. Il costo della visita di un nodo interno richiede $O(1)$ mentre quello di una foglia richiede $O(n)$ (per la stampa).

Analisi

```
SOTTOINSIEMI(SOL, k, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
     FOR(i=0, i<2, i=i+1){
        SOL[k + 1]=i
        SOTTOINSIEMI(SOL, k + 1, n)
     }
    }
```

La procedura viene invocata da $SOTTOINSIEMI(SOL, 0, n)$. L'albero delle soluzioni che ne deriva è un albero binario completo di altezza n in quanto ogni nodo a livello $k < n$ ha esattamente due figli. Il costo della visita di un nodo interno richiede $O(1)$ mentre quello di una foglia richiede $O(n)$ (per la stampa). Il costo della visita sarà dato dalla somma dei costi dei $2^n - 1$ nodi interni e delle 2^n foglie ed è quindi $O(n2^n)$.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare
(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare

(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare

(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Ci sono n diverse scelte per il valore del primo elemento di una permutazione di $\{1, 2, \ldots, n\}$.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare

(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Ci sono n diverse scelte per il valore del primo elemento di una permutazione di $\{1, 2, \ldots, n\}$. Fissato il primo elemento a_1 della permutazione, ci sono $n - 1$ candidati per la seconda posizione in quanto possiamo avere un qualunque valore eccetto a_1.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare

(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Ci sono n diverse scelte per il valore del primo elemento di una permutazione di $\{1, 2, \ldots, n\}$. Fissato il primo elemento a_1 della permutazione, ci sono $n - 1$ candidati per la seconda posizione in quanto possiamo avere un qualunque valore eccetto a_1. Ripetendo questo argomento si ottiene un totale di $n!$ distinte permutazioni.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare

(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Ci sono n diverse scelte per il valore del primo elemento di una permutazione di $\{1, 2, \ldots, n\}$. Fissato il primo elemento a_1 della permutazione, ci sono $n - 1$ candidati per la seconda posizione in quanto possiamo avere un qualunque valore eccetto a_1. Ripetendo questo argomento si ottiene un totale di $n!$ distinte permutazioni.

Per rappresentare una permutazione possiamo utilizzare un vettore di n elementi dove nella cella i-esima troviamo l'i-esimo elemento della permutazione.

Secondo esempio: Generazioni di Permutazioni

Dati n elementi $s_1, s_2, \ldots, s_n$, vogliamo generare tutte le possibili permutazioni degli elementi sottoinsiemi di S.

Esempio: Dati s_1, s_2, s_3, vogliamo generare
(s_1, s_2, s_3) (s_1, s_3, s_2) (s_2, s_1, s_3)
(s_2, s_3, s_1) (s_3, s_1, s_2) (s_3, s_2, s_1)

Ci sono n diverse scelte per il valore del primo elemento di una permutazione di $\{1, 2, \ldots, n\}$. Fissato il primo elemento a_1 della permutazione, ci sono $n - 1$ candidati per la seconda posizione in quanto possiamo avere un qualunque valore eccetto a_1. Ripetendo questo argomento si ottiene un totale di $n!$ distinte permutazioni.

Per rappresentare una permutazione possiamo utilizzare un vettore di n elementi dove nella cella i-esima troviamo l'i-esimo elemento della permutazione.

L'insieme S_k dei candidati per la k-esima posizione è dato dall'insieme di elementi che non compaiono tra i $k - 1$ elementi della soluzione parziale.

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$.

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
     IF(ELEM[i] = 0) THEN {
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
     IF(ELEM[i] = 0) THEN {
       SOL[k + 1] = i
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
     IF(ELEM[i] = 0) THEN {
       SOL[k + 1] = i
       ELEM[i] = 1
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
     IF(ELEM[i] = 0) THEN {
       SOL[k + 1] = i
       ELEM[i] = 1
       PERMUTAZIONI(SOL, k + 1, ELEM, n)
```

Per verificare in modo efficiente se un dato elemento è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$,dove

$$ELEM(i) = 1 \text{ se e solo se } s_i \text{ è già nella soluzione parziale}$$

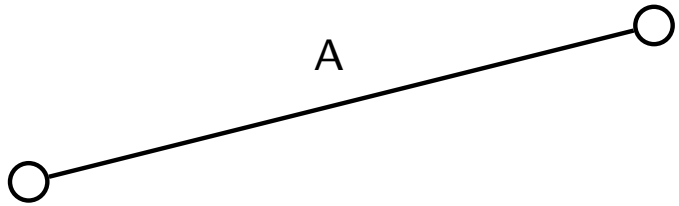
All'inizio, $ELEM(i) = 0, \forall i$. Lo pseudocodice della procedura è:

```
PERMUTAZIONI(SOL, k, ELEM, n)
IF (k = n) THEN {stampa il vettore SOL
    } ELSE {
    FOR(i=1, i<n+1, i=i+1) {
     IF(ELEM[i] = 0) THEN {
       SOL[k + 1] = i
       ELEM[i] = 1
       PERMUTAZIONI(SOL, k + 1, ELEM, n)
       ELEM[i] = 1
     }
} }
```

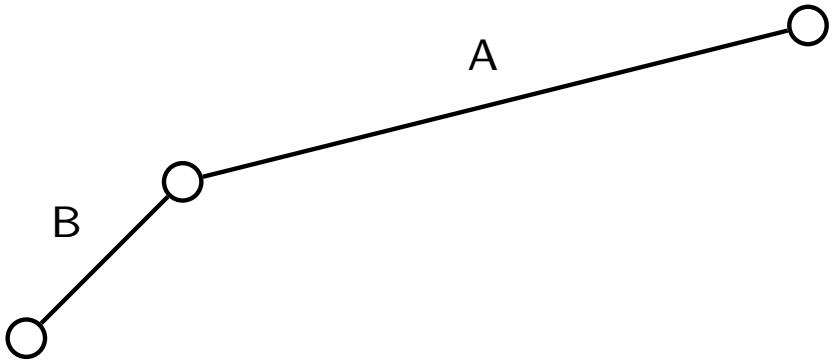
La procedura viene invocata da PERMUTAZIONI$(SOL, 0, ELEM, n)$.

Un esempio: le permutazioni di A, B, C

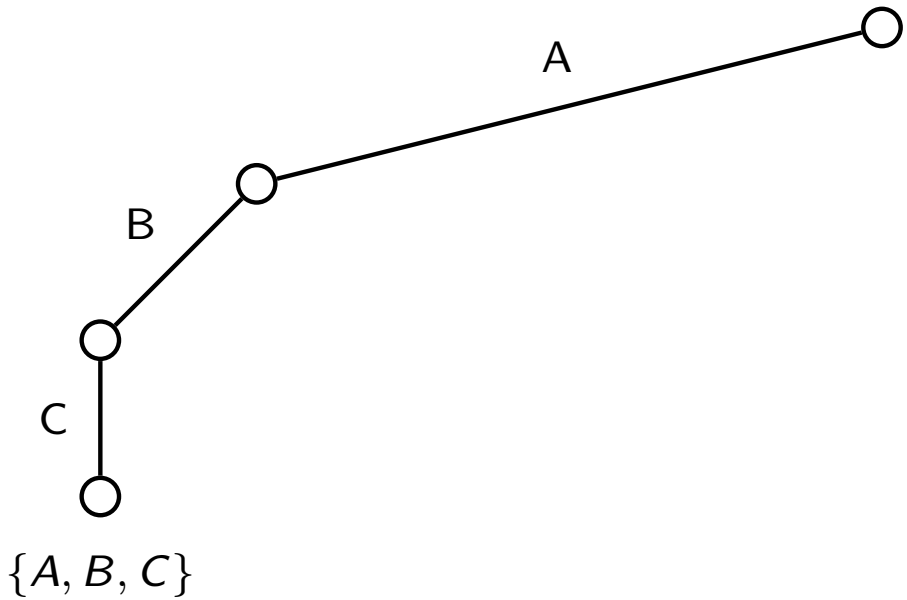
Un esempio: le permutazioni di A, B, C



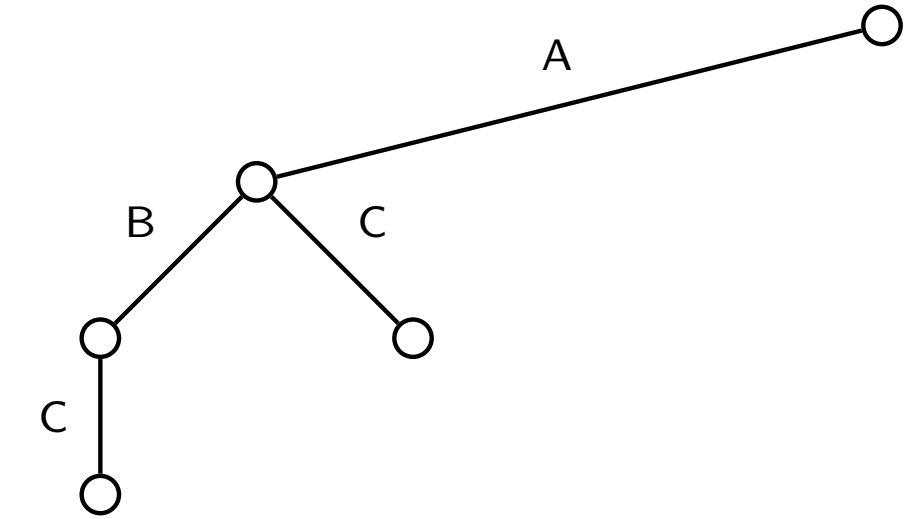
Un esempio: le permutazioni di A, B, C



Un esempio: le permutazioni di A, B, C

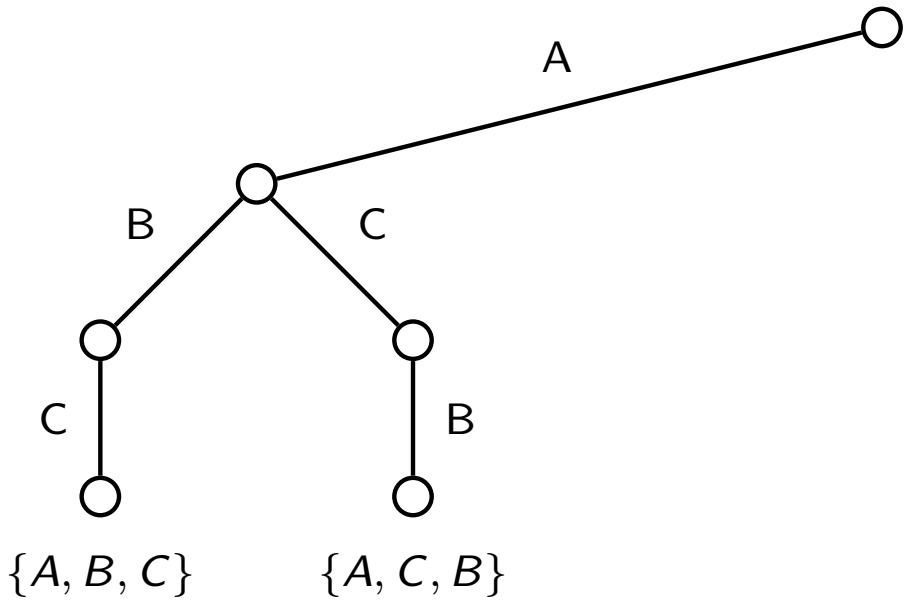


Un esempio: le permutazioni di A, B, C

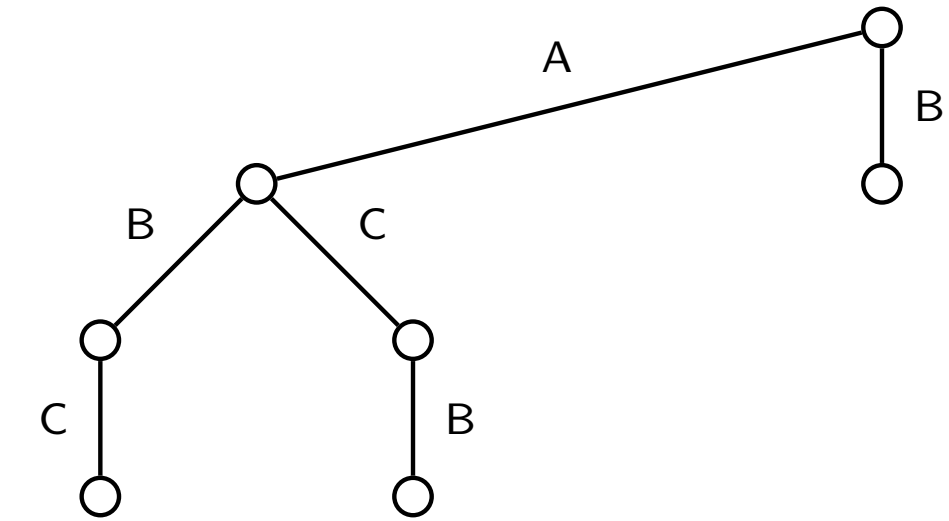


$\{A, B, C\}$

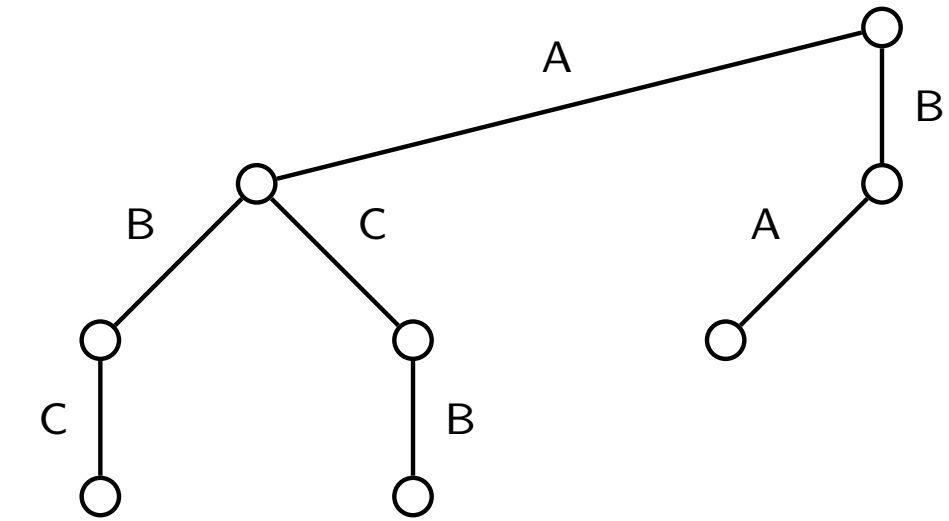
Un esempio: le permutazioni di A, B, C



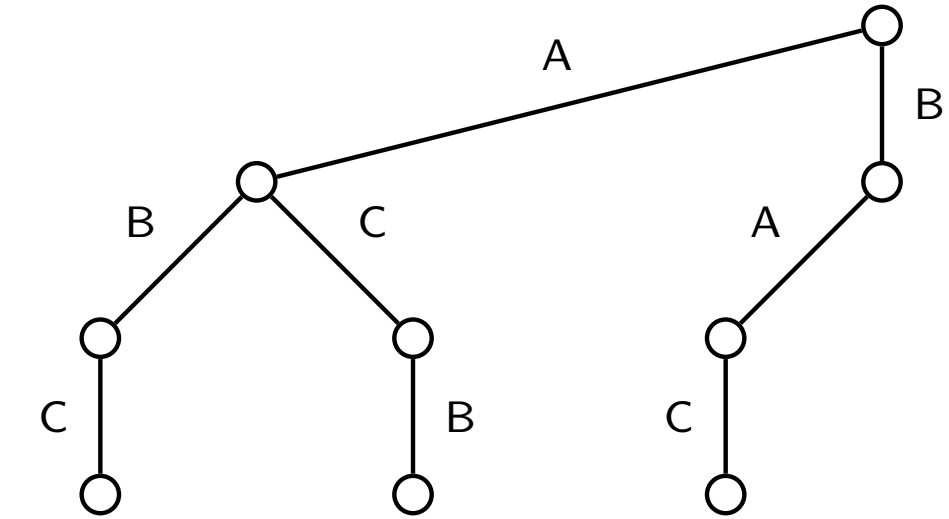
Un esempio: le permutazioni di A, B, C



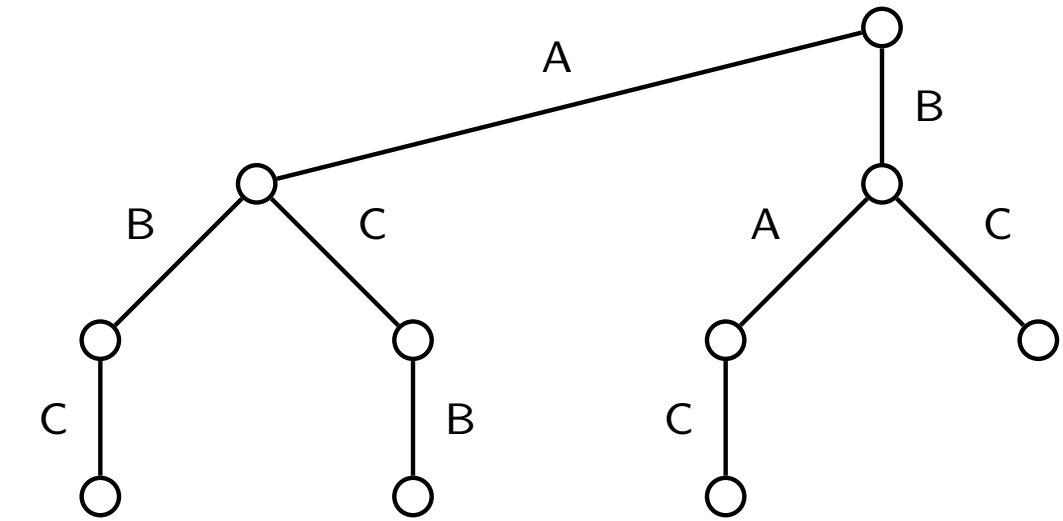
Un esempio: le permutazioni di A, B, C



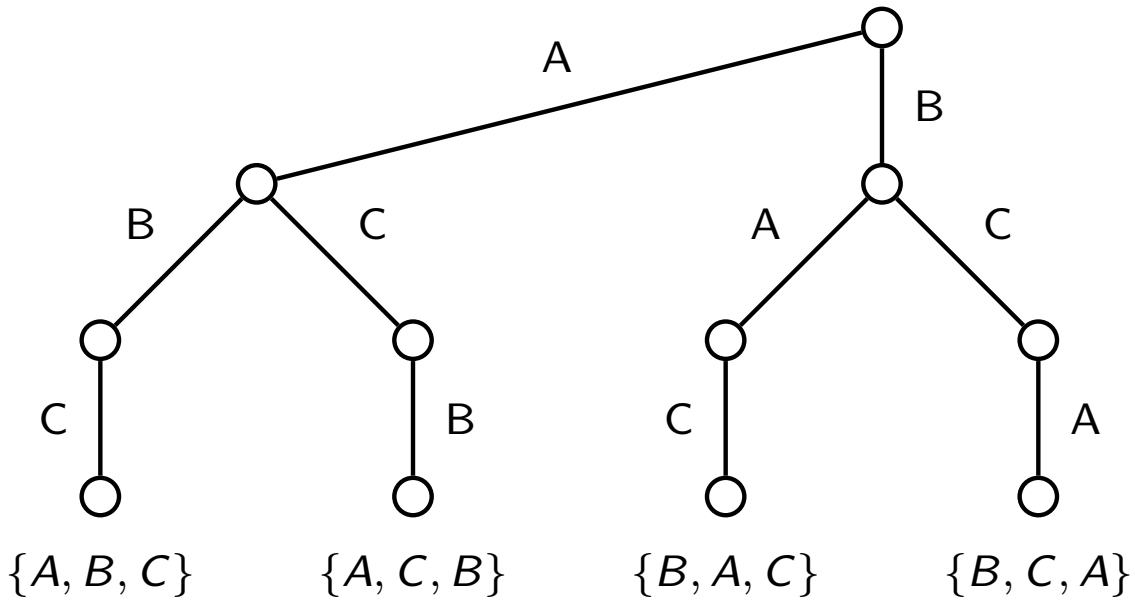
Un esempio: le permutazioni di A, B, C



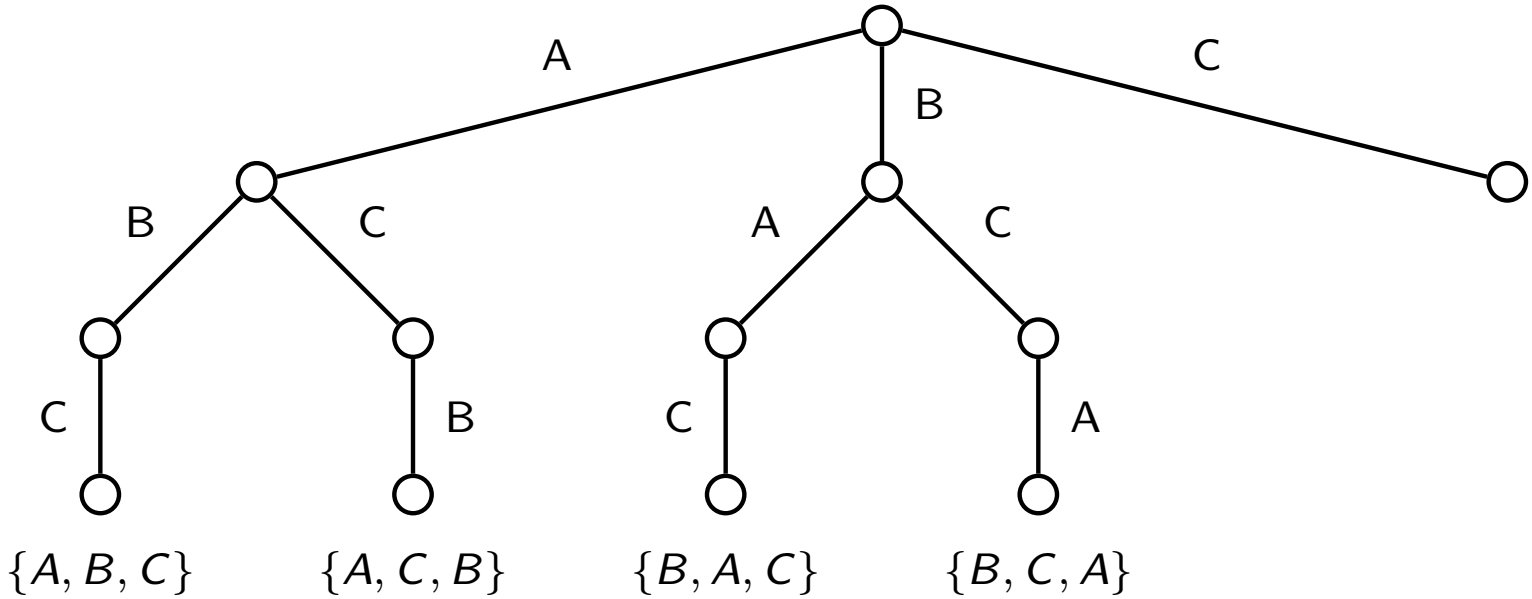
Un esempio: le permutazioni di A, B, C



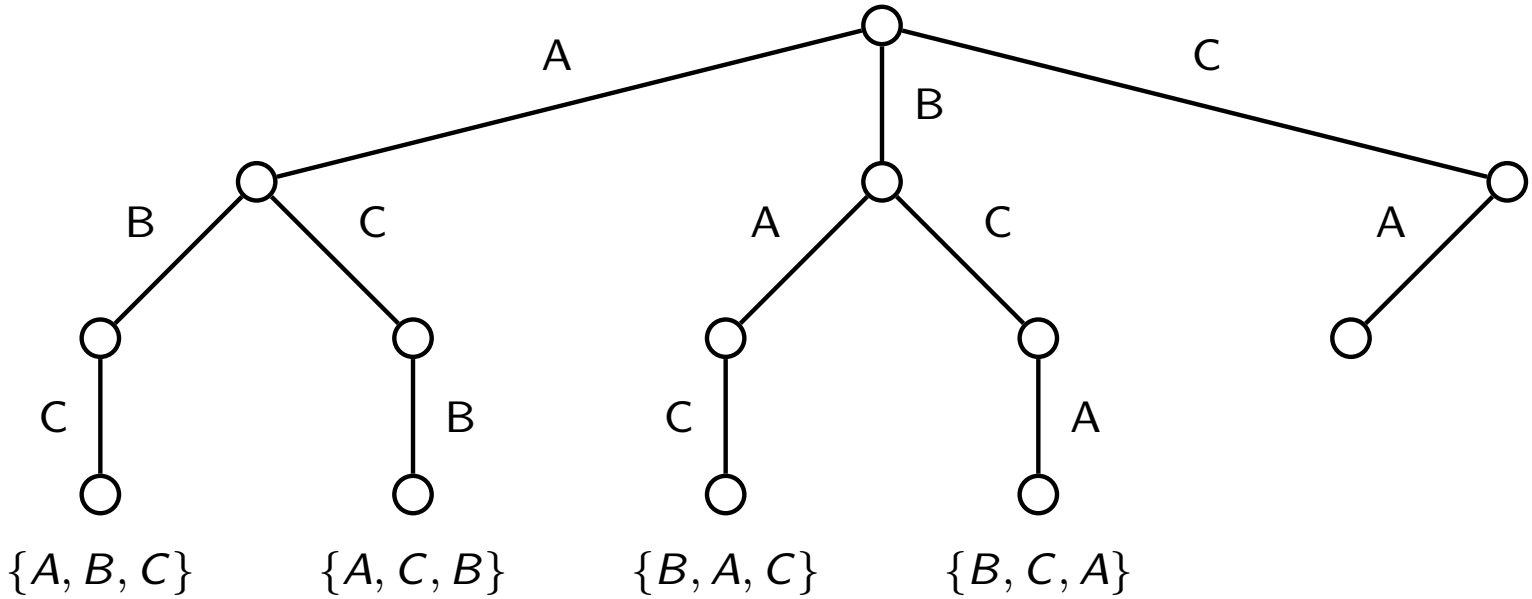
Un esempio: le permutazioni di A, B, C



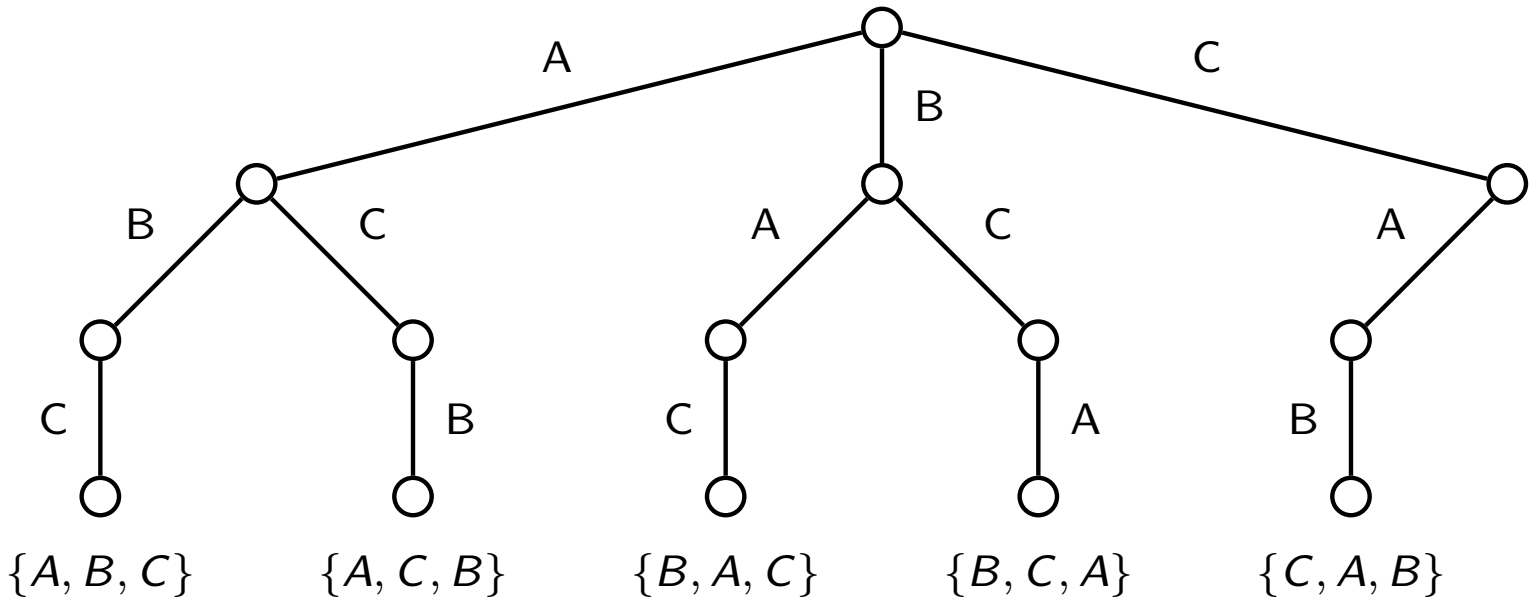
Un esempio: le permutazioni di A, B, C



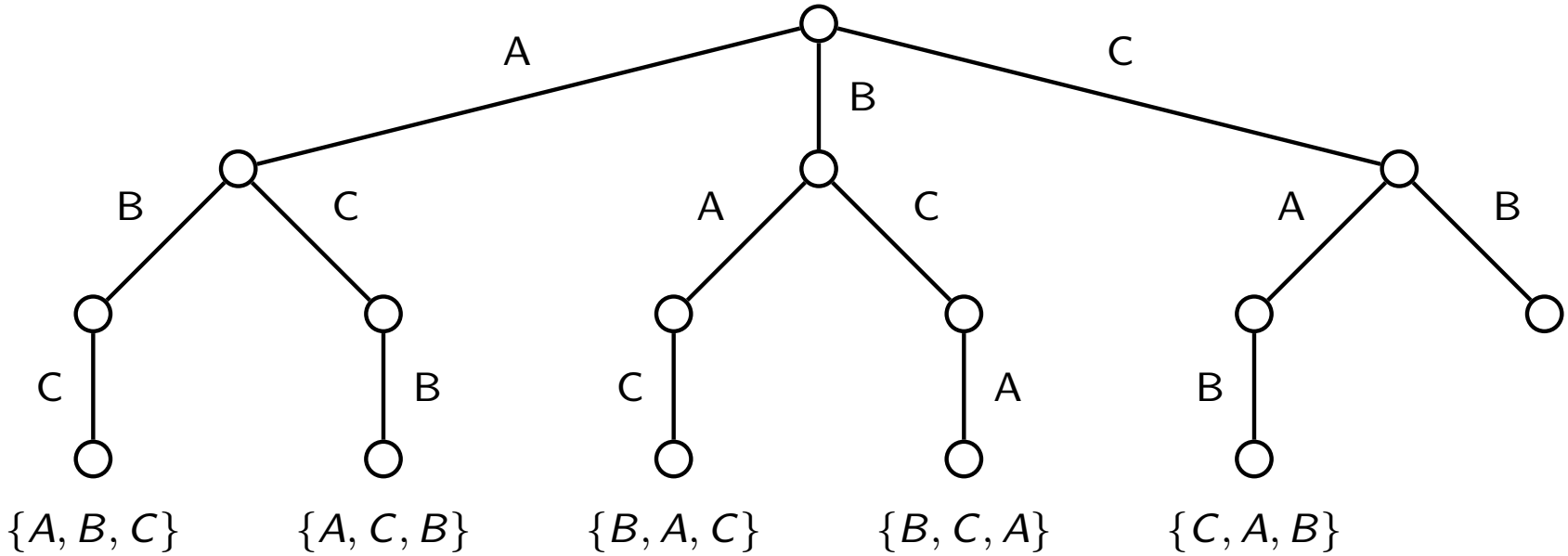
Un esempio: le permutazioni di A, B, C



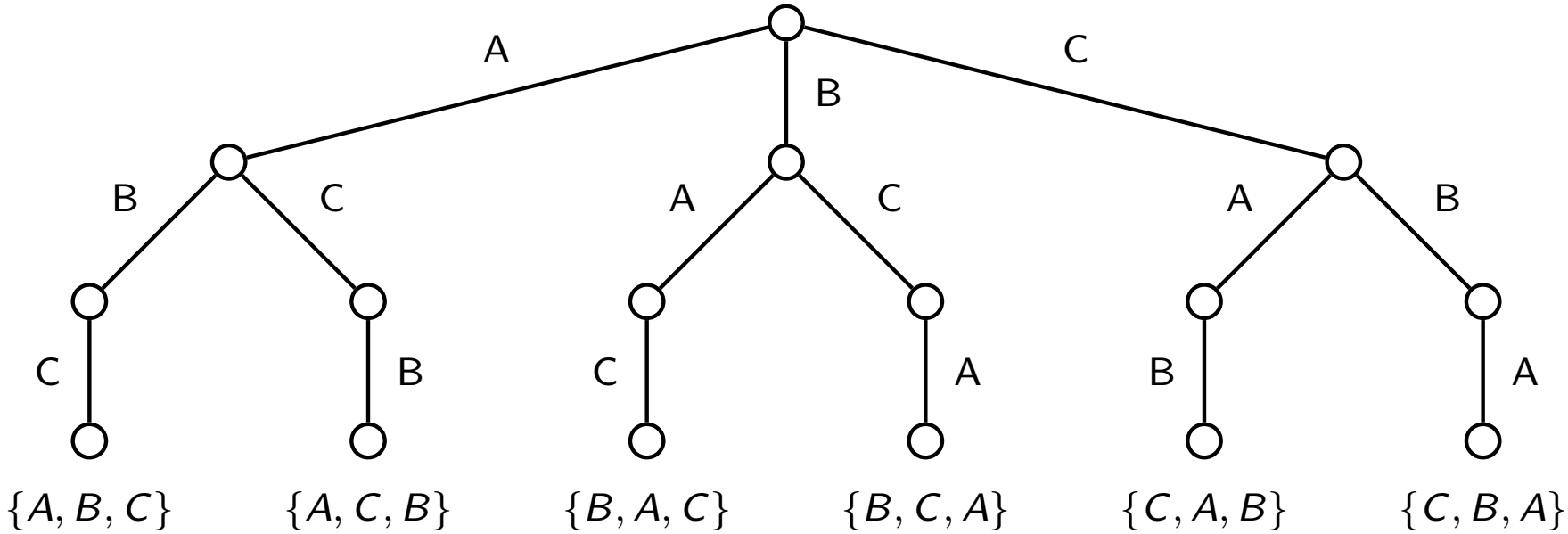
Un esempio: le permutazioni di A, B, C



Un esempio: le permutazioni di A, B, C



Un esempio: le permutazioni di A, B, C



L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$)

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$).

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!}$$

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!}$$

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!} = n! \sum_{j=1}^{n} \frac{1}{j!}$$

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!} = n! \sum_{j=1}^{n} \frac{1}{j!} < n! \sum_{j=1}^{n} \frac{1}{2^{j-1}}$$

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!} = n! \sum_{j=1}^{n} \frac{1}{j!} < n! \sum_{j=1}^{n} \frac{1}{2^{j-1}} = O(n!)$$

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!} = n! \sum_{j=1}^{n} \frac{1}{j!} < n! \sum_{j=1}^{n} \frac{1}{2^{j-1}} = O(n!)$$

Il costo della visita di un nodo interno o di una foglia richiede richiede $O(n)$.

L'albero delle soluzioni che ne deriva è un albero di altezza n (un livello per ciascuno degli elementi $s_i \in \{s_1, \ldots, s_n\}$) ed $n!$ foglie (una per ogni permutazione).

Un generico nodo interno al livello $k < n$ corrisponde all'aver scelto k elementi $s_{i_1}, \ldots, s_{i_k}$ nelle prime posizioni della permutazione.

Quindi, ogni nodo al livello k ha $n - k$ figli (i restanti elementi da scegliere per formare l'intera permutazione $s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots s_{i_n}$). (quindi a livello k l'albero ha esattamente $\frac{n!}{(n-k)!}$ nodi).

Il numero dei nodi dell'albero è quindi

$$\sum_{k=0}^{n-1} \frac{n!}{(n-k)!} = n! \sum_{k=0}^{n-1} \frac{1}{(n-k)!} = n! \sum_{j=1}^{n} \frac{1}{j!} < n! \sum_{j=1}^{n} \frac{1}{2^{j-1}} = O(n!)$$

Il costo della visita di un nodo interno o di una foglia richiede richiede $O(n)$. Il costo della visita sarà dato dalla somma dei costi dei nodi dell'albero ed è quindi $O(n \cdot n!)$.

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero,

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà (**soluzioni ammissibili**).

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà (**soluzioni ammissibili**).

Un possibile approccio è quello di generare tutte le soluzioni e di stampare poi solo quelle che soddisfano le proprietà (quelle ammissibili).

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà (**soluzioni ammissibili**).

Un possibile approccio è quello di generare tutte le soluzioni e di stampare poi solo quelle che soddisfano le proprietà (quelle ammissibili). Un tale approccio tende ad essere piuttosto costoso.

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà (**soluzioni ammissibili**).

Un possibile approccio è quello di generare tutte le soluzioni e di stampare poi solo quelle che soddisfano le proprietà (quelle ammissibili). Un tale approccio tende ad essere piuttosto costoso. Durante la visita dell'albero delle soluzioni, se su di un nodo interno si ha abbastanza informazione per concludere che il suo sottoalbero **non** contiene soluzioni ammissibili, allora è del tutto inutile esplorare il sottoalbero.

Fin'ora abbiamo visto come per problemi di enumerazione conviene strutturare l'insieme delle soluzioni come le foglie di un albero, albero che verrà poi visitato con una visita in profondità.

Supponiamo ora di essere interessati a soluzioni con particolari proprietà (**soluzioni ammissibili**).

Un possibile approccio è quello di generare tutte le soluzioni e di stampare poi solo quelle che soddisfano le proprietà (quelle ammissibili). Un tale approccio tende ad essere piuttosto costoso. Durante la visita dell'albero delle soluzioni, se su di un nodo interno si ha abbastanza informazione per concludere che il suo sottoalbero **non** contiene soluzioni ammissibili, allora è del tutto inutile esplorare il sottoalbero.

Una tecnica generale per cercare di ridurre i costi consiste nel definire, quando possibile, delle funzioni di taglio in grado di potare l'albero su cui si effettua la visita.

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti

Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
       stampa il vettore SOL
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
       stampa il vettore SOL
    } ELSE {
      FOR (i=0, i<2,i=i+1) {
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
       stampa il vettore SOL
    } ELSE {
      FOR (i=0, i<2,i=i+1) {
          SOL[k + 1] =i
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
       stampa il vettore SOL
    } ELSE {
      FOR (i=0, i<2,i=i+1) {
          SOL[k + 1] =i
          SOTTOINSIEMI-VINCOLATI(SOL, k + 1, c, n)
  }   }
```

Enumerazione di sottoinsiemi di cardinalità limitata.

Dato un insieme di n oggetti ed un intero c, con $1 \leq c \leq n$, vogliamo enumerare tutti i sottoinsiemi con esattamente c oggetti
Potremmo riutilizzare la procedura che genera tutti i sottoinsiemi modificata in modo che prima di stampare una soluzione verifica se l'insieme generato ha esattamente c elementi.

```
SOTTOINSIEMI-VINCOLATI(SOL, k, c, n)
IF(k = n) THEN {
    tot = 0
    FOR(i=1, i<n+1, i=i+1) {
       IF (SOL[i] = 1) {
          tot = tot + 1
       }   }
    IF (tot = c) {
       stampa il vettore SOL
    } ELSE {
      FOR (i=0, i<2,i=i+1) {
          SOL[k + 1] =i
          SOTTOINSIEMI-VINCOLATI(SOL, k + 1, c, n)
   }   }
```

La complessità dell'algoritmo è $O(n2^n)$ in quanto l'albero delle soluzioni viene visitato comunque per intero indipendentemente dal numero di soluzioni ammissibili.

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!}$$

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot\ldots\cdot(n-c)}{c!}$$

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot\ldots\cdot(n-c)}{c!} = O(n^c)$$

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot\ldots\cdot(n-c)}{c!} = O(n^c)$$

quanto più è piccolo c rispetto ad n, tanto più l'algoritmo di complessità $O(n2^n)$ risulta inefficiente.

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot \ldots \cdot (n-c)}{c!} = O(n^c)$$

quanto più è piccolo c rispetto ad n, tanto più l'algoritmo di complessità $O(n2^n)$ risulta inefficiente.
Possiamo potare l'albero delle soluzioni in quanto:

1. Un nodo corrispondente ad una soluzione parziale in cui sono gia stati inseriti più di c elementi ha come foglie del suo sottoalbero solo soluzioni con più di c elementi

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot\ldots\cdot(n-c)}{c!} = O(n^c)$$

quanto più è piccolo c rispetto ad n, tanto più l'algoritmo di complessità $O(n2^n)$ risulta inefficiente.
Possiamo potare l'albero delle soluzioni in quanto:

1. Un nodo corrispondente ad una soluzione parziale in cui sono gia stati inseriti più di c elementi ha come foglie del suo sottoalbero solo soluzioni con più di c elementi e quindi nel sottoalbero di questo nodo non sono presenti soluzioni ammissibili.

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot \ldots \cdot (n-c)}{c!} = O(n^c)$$

quanto più è piccolo c rispetto ad n, tanto più l'algoritmo di complessità $O(n2^n)$ risulta inefficiente.
Possiamo potare l'albero delle soluzioni in quanto:

1. Un nodo corrispondente ad una soluzione parziale in cui sono gia stati inseriti più di c elementi ha come foglie del suo sottoalbero solo soluzioni con più di c elementi e quindi nel sottoalbero di questo nodo non sono presenti soluzioni ammissibili.
2. un nodo corrispondente ad una soluzione parziale la somma dei cui elementi più tutti gli elementi non ancora considerati è inferiore a c ha come foglie del suo sottoalbero solo soluzioni con meno di c elementi

Però gli elementi da enumerare sono

$$\binom{n}{c} = \frac{n!}{c!(n-c)!} = \frac{n(n-1)\cdot\ldots\cdot(n-c)}{c!} = O(n^c)$$

quanto più è piccolo c rispetto ad n, tanto più l'algoritmo di complessità $O(n2^n)$ risulta inefficiente.
Possiamo potare l'albero delle soluzioni in quanto:

1. Un nodo corrispondente ad una soluzione parziale in cui sono gia stati inseriti più di c elementi ha come foglie del suo sottoalbero solo soluzioni con più di c elementi e quindi nel sottoalbero di questo nodo non sono presenti soluzioni ammissibili.
2. un nodo corrispondente ad una soluzione parziale la somma dei cui elementi più tutti gli elementi non ancora considerati è inferiore a c ha come foglie del suo sottoalbero solo soluzioni con meno di c elementi e quindi nel sottoalbero di questo nodo non sono presenti soluzioni ammissibili.

- Nella visita dell'albero possiamo quindi utilizzare due funzioni di taglio, una che interviene per la visita del sottoalbero sinistro del nodo correntemente in esame e l'altra che interviene per la visita del sottoalbero destro del nodo.

- Nella visita dell'albero possiamo quindi utilizzare due funzioni di taglio, una che interviene per la visita del sottoalbero sinistro del nodo correntemente in esame e l'altra che interviene per la visita del sottoalbero destro del nodo.
- La prima fa sì che il figlio sinistro venga visitato solo se escludere l'elemento $k+1$-esimo dalla soluzione che si sta costruendo non preclude la possibilità di poter ottenere una soluzione con almeno c elementi.

- Nella visita dell'albero possiamo quindi utilizzare due funzioni di taglio, una che interviene per la visita del sottoalbero sinistro del nodo correntemente in esame e l'altra che interviene per la visita del sottoalbero destro del nodo.
- La prima fa sì che il figlio sinistro venga visitato solo se escludere l'elemento $k+1$-esimo dalla soluzione che si sta costruendo non preclude la possibilità di poter ottenere una soluzione con almeno c elementi.
- La seconda fa sì che il nodo destro venga visitato solo se aggiungere l'elemento $k+1$-esimo alla soluzione parziale non genera una nuova soluzione parziale con più di c elementi.

$n = 4, c = 2$

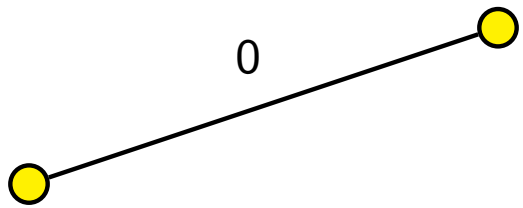
Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c “1”.

$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c “1”.

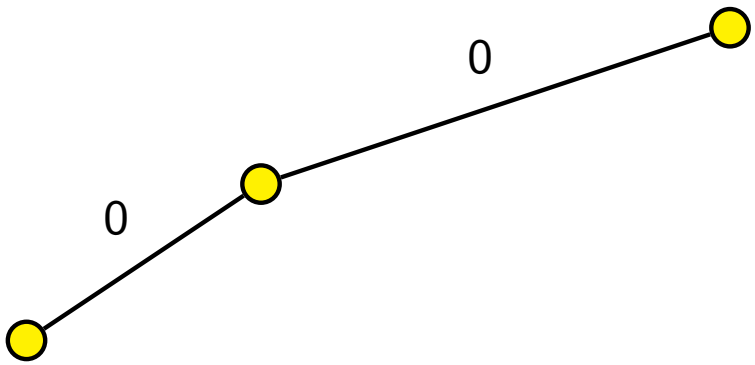
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c “1”.



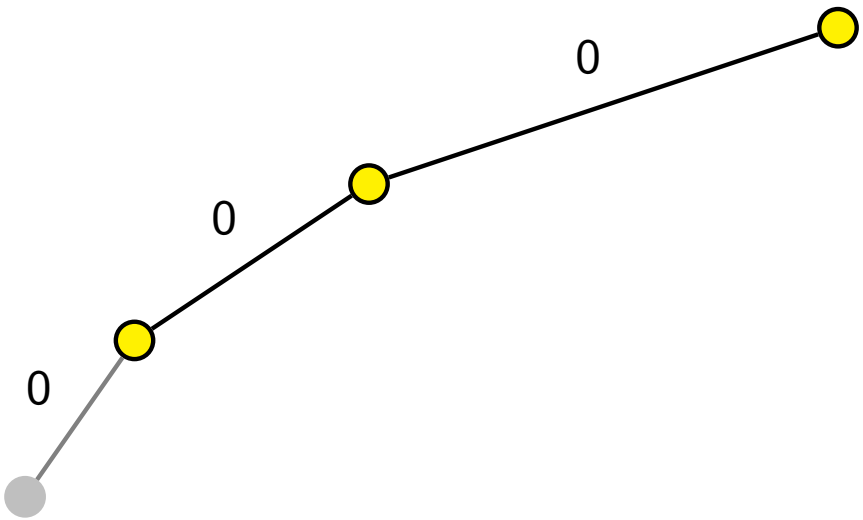
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



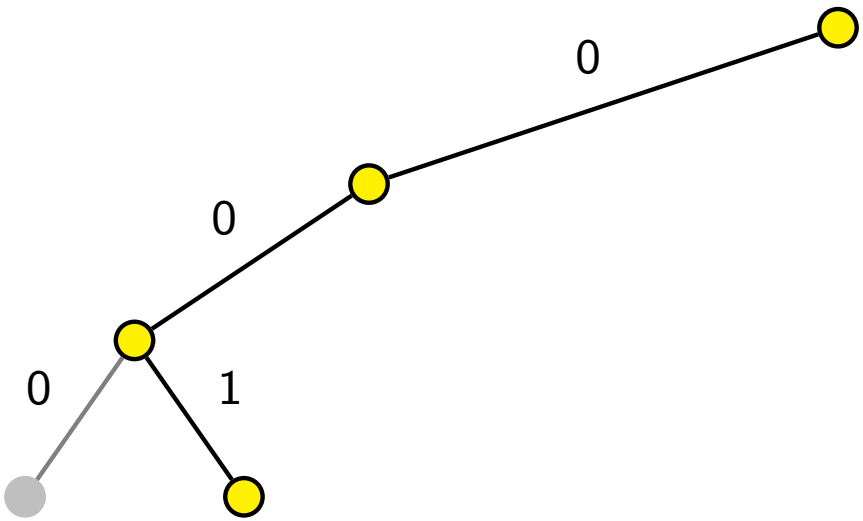
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



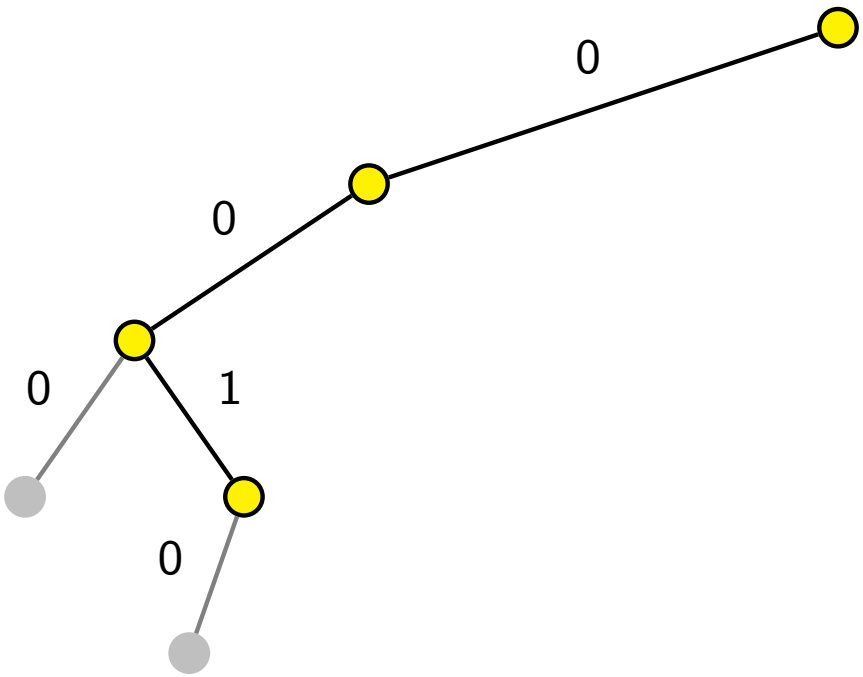
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



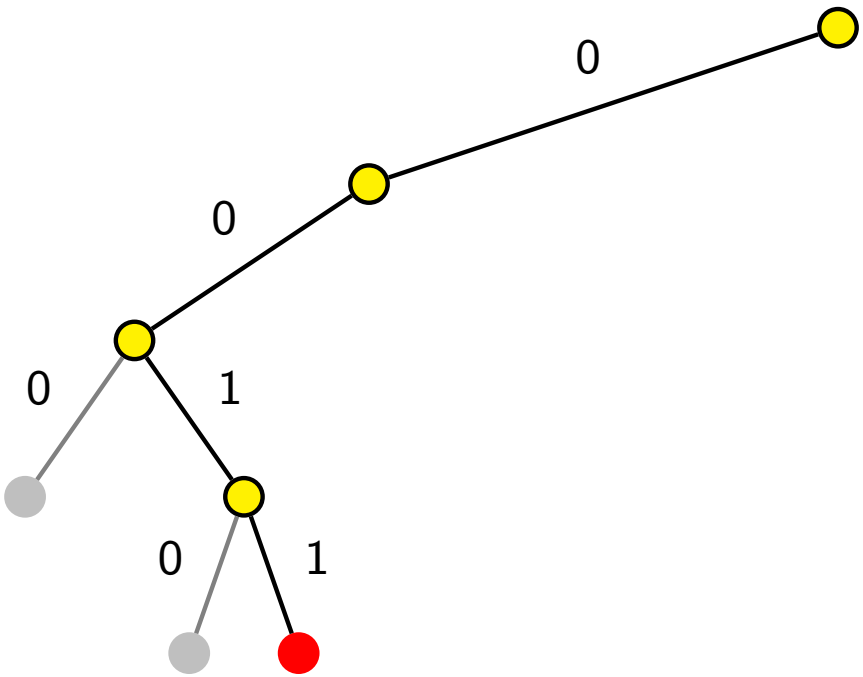
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



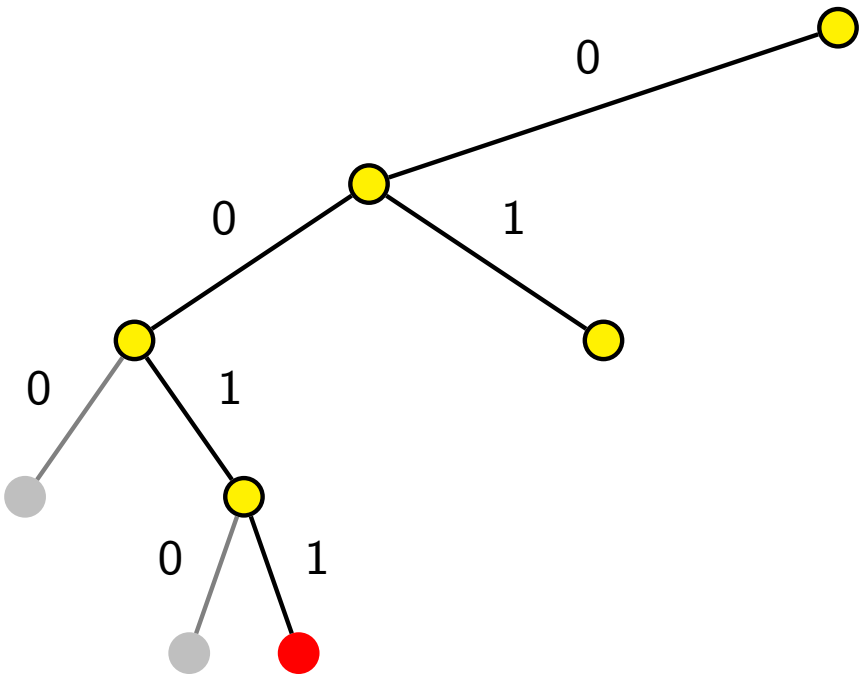
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



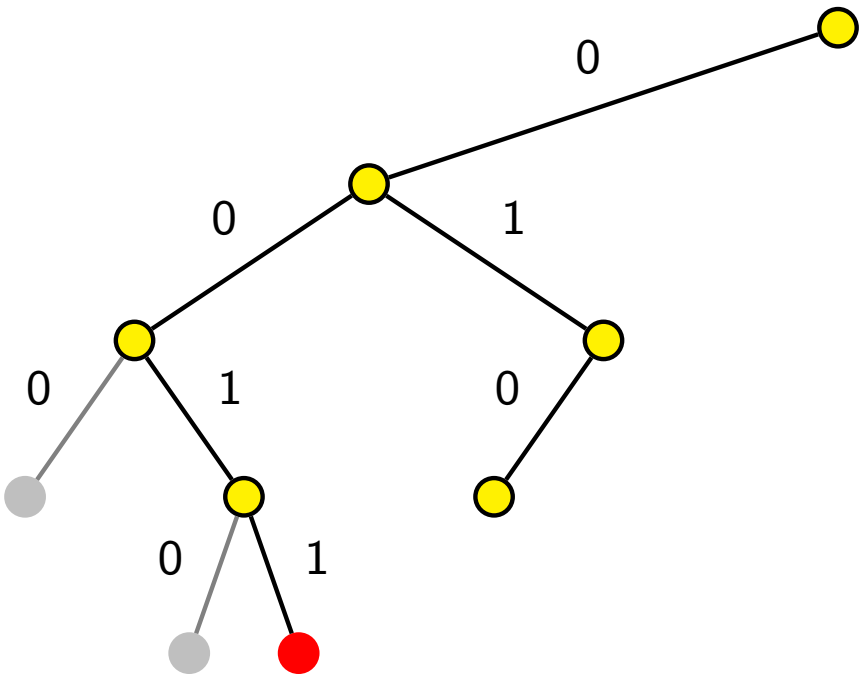
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



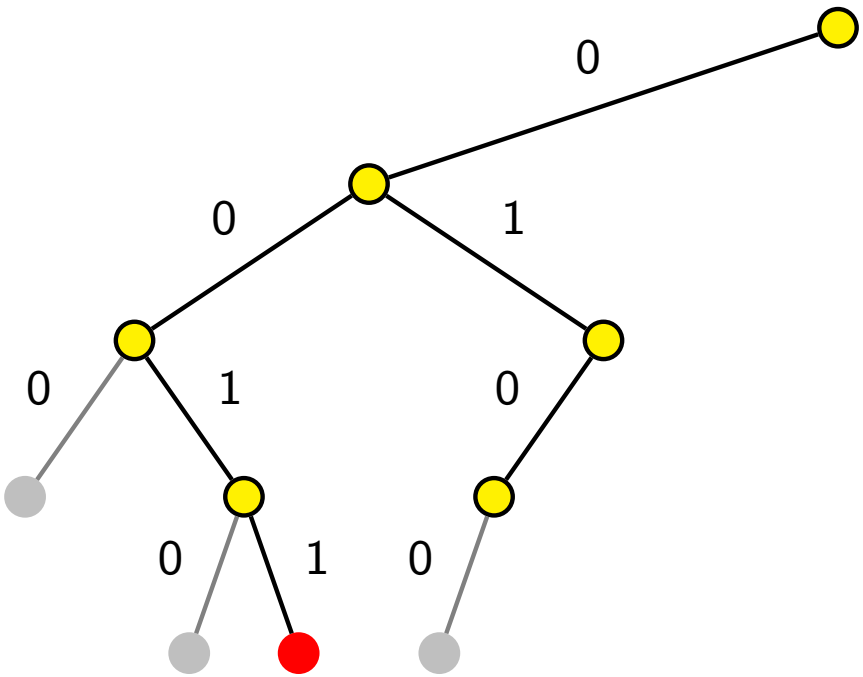
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



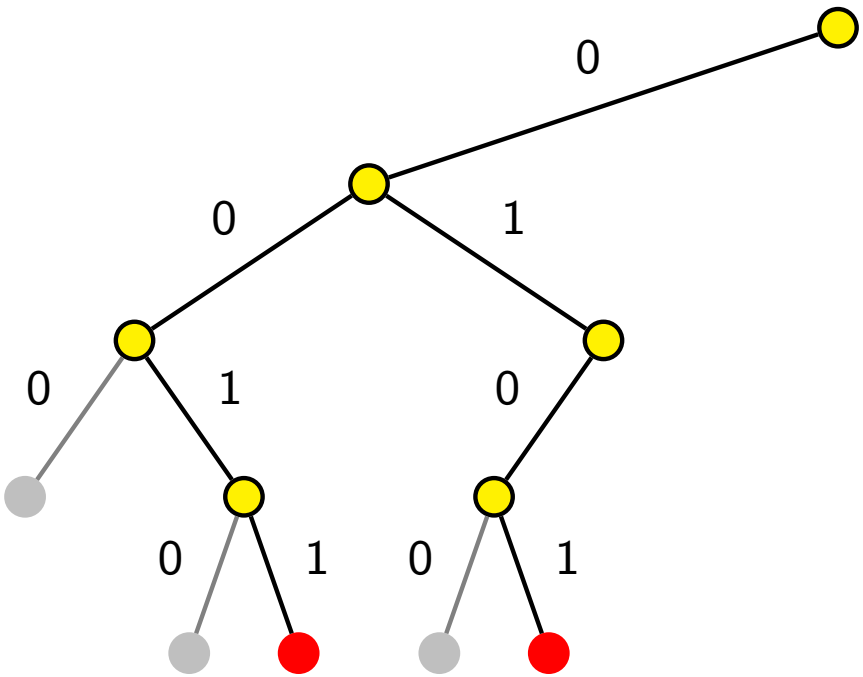
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



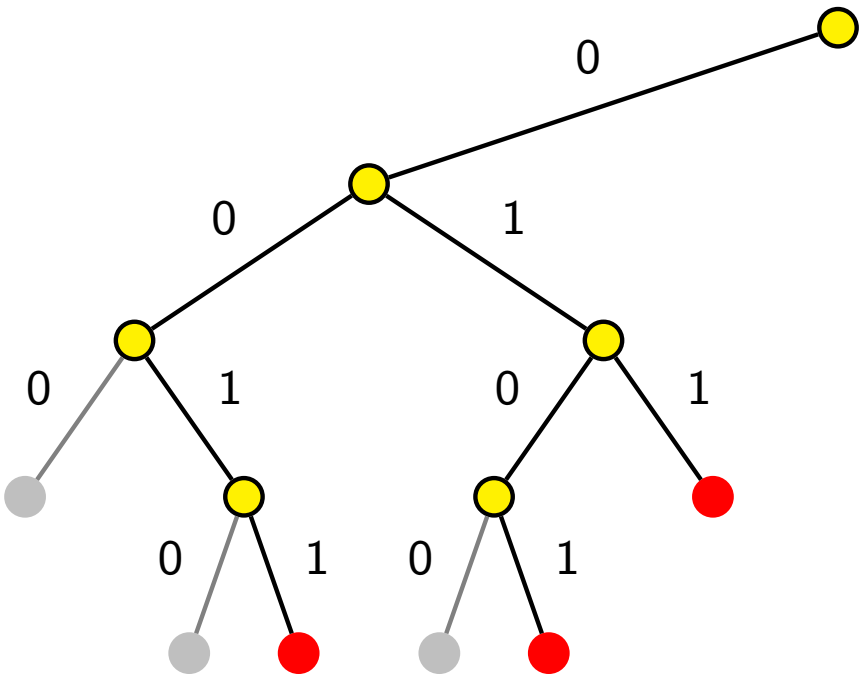
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



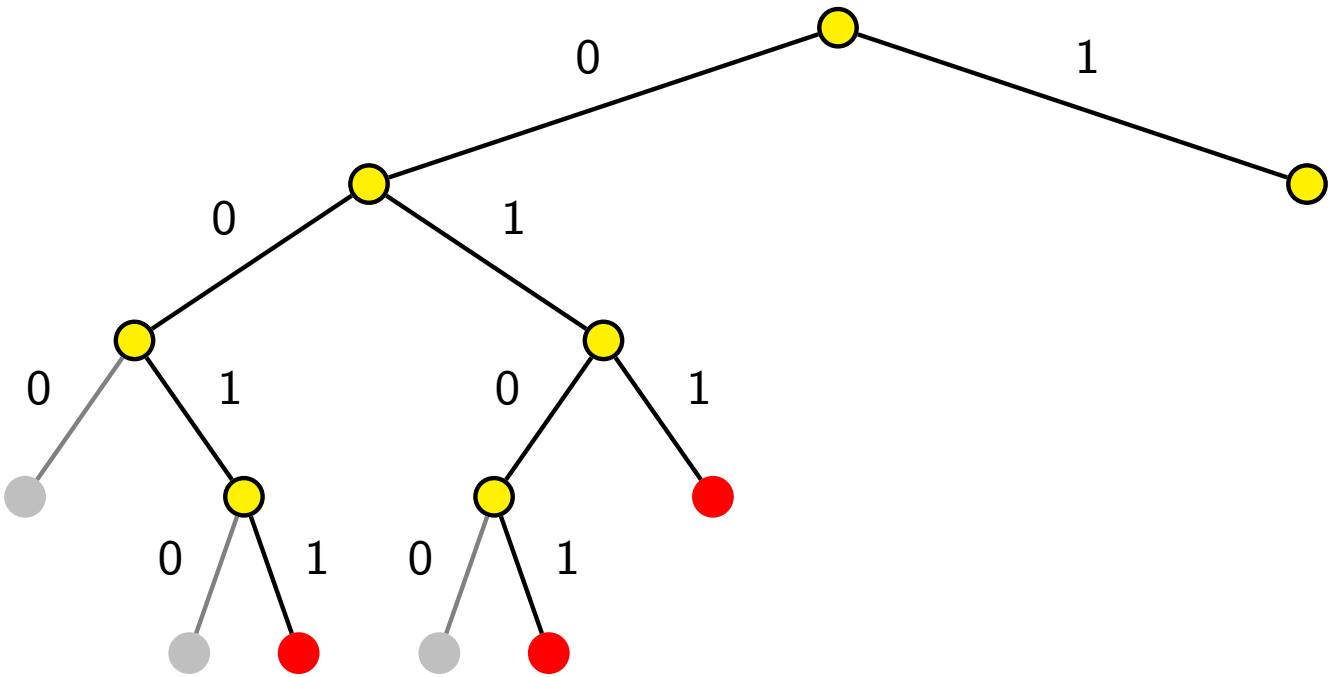
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



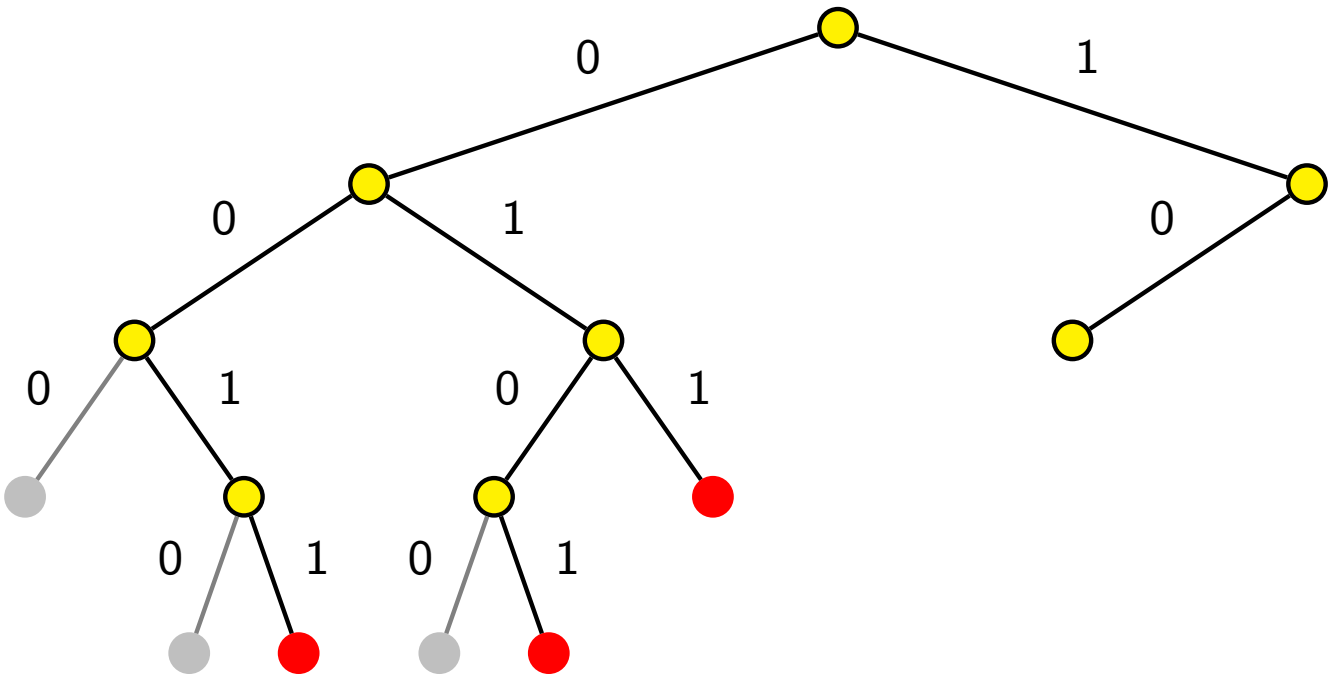
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



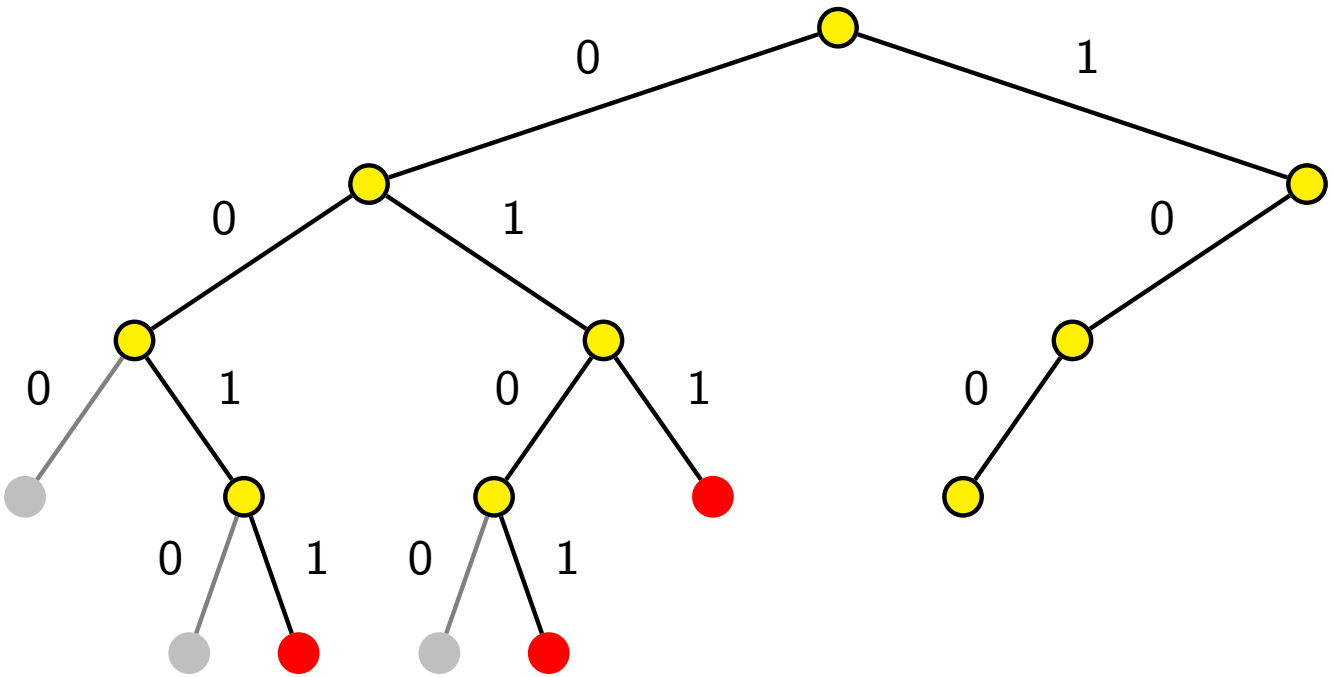
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



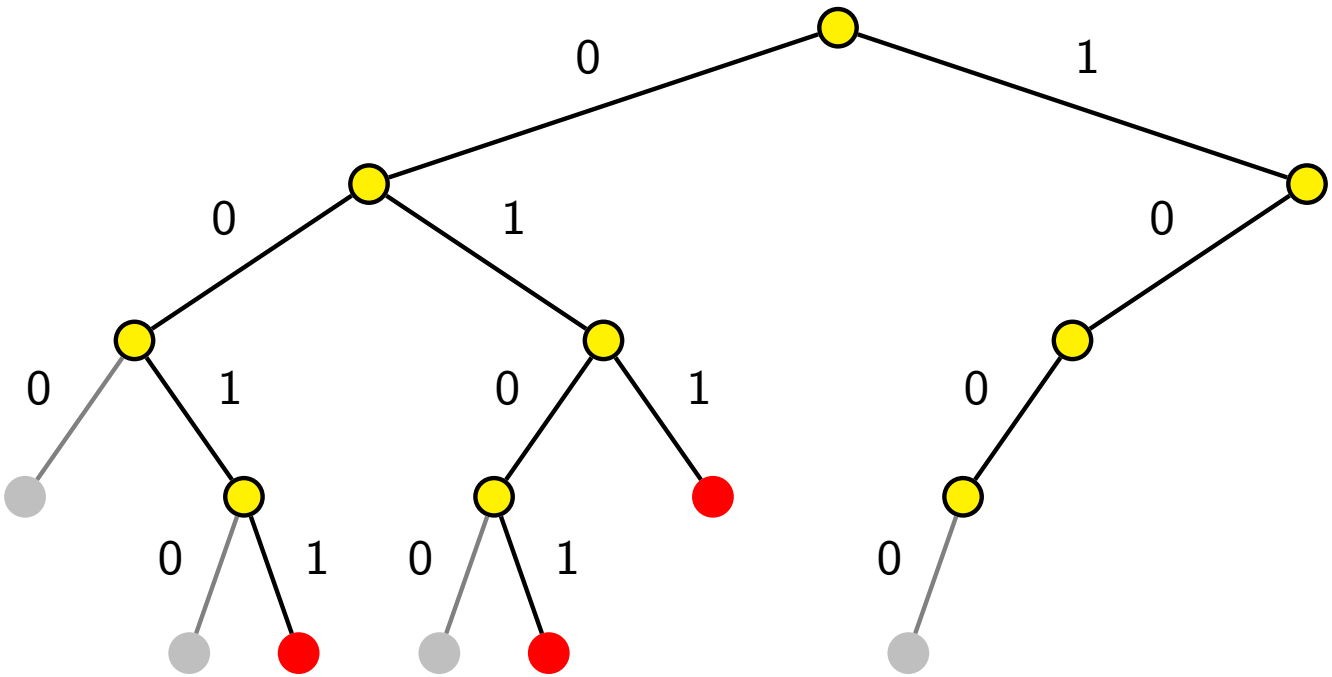
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



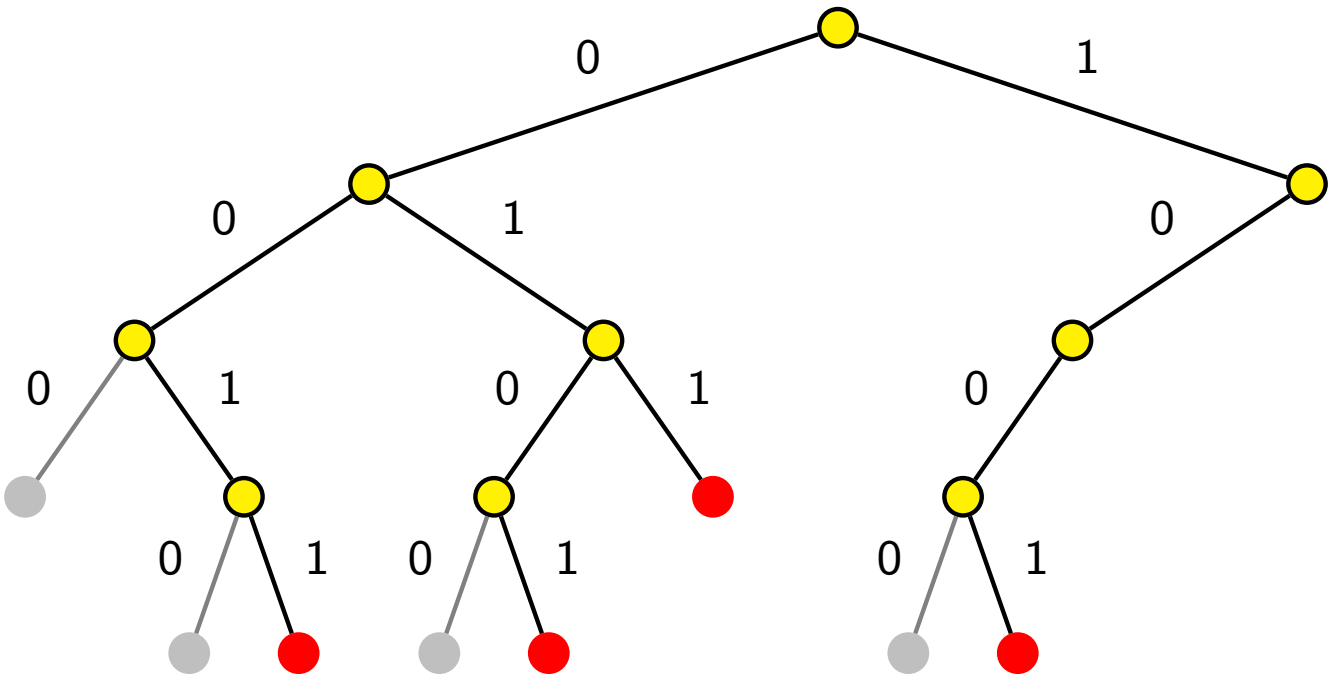
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



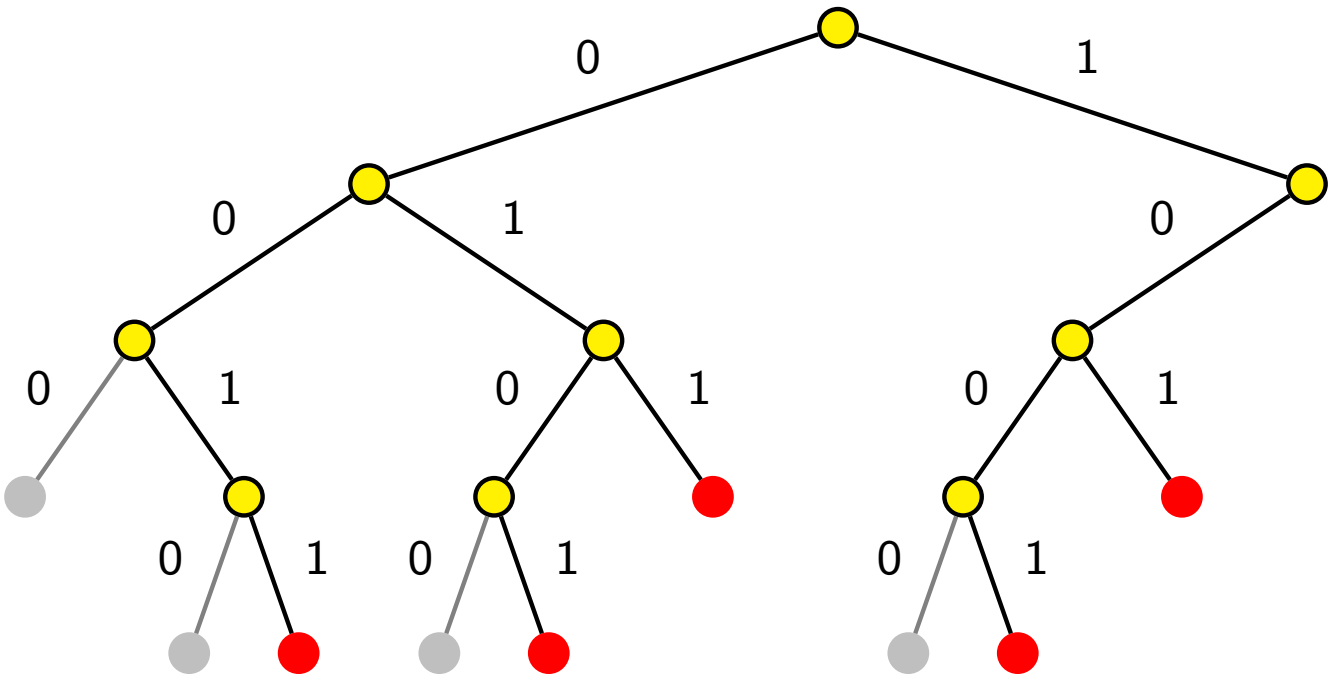
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



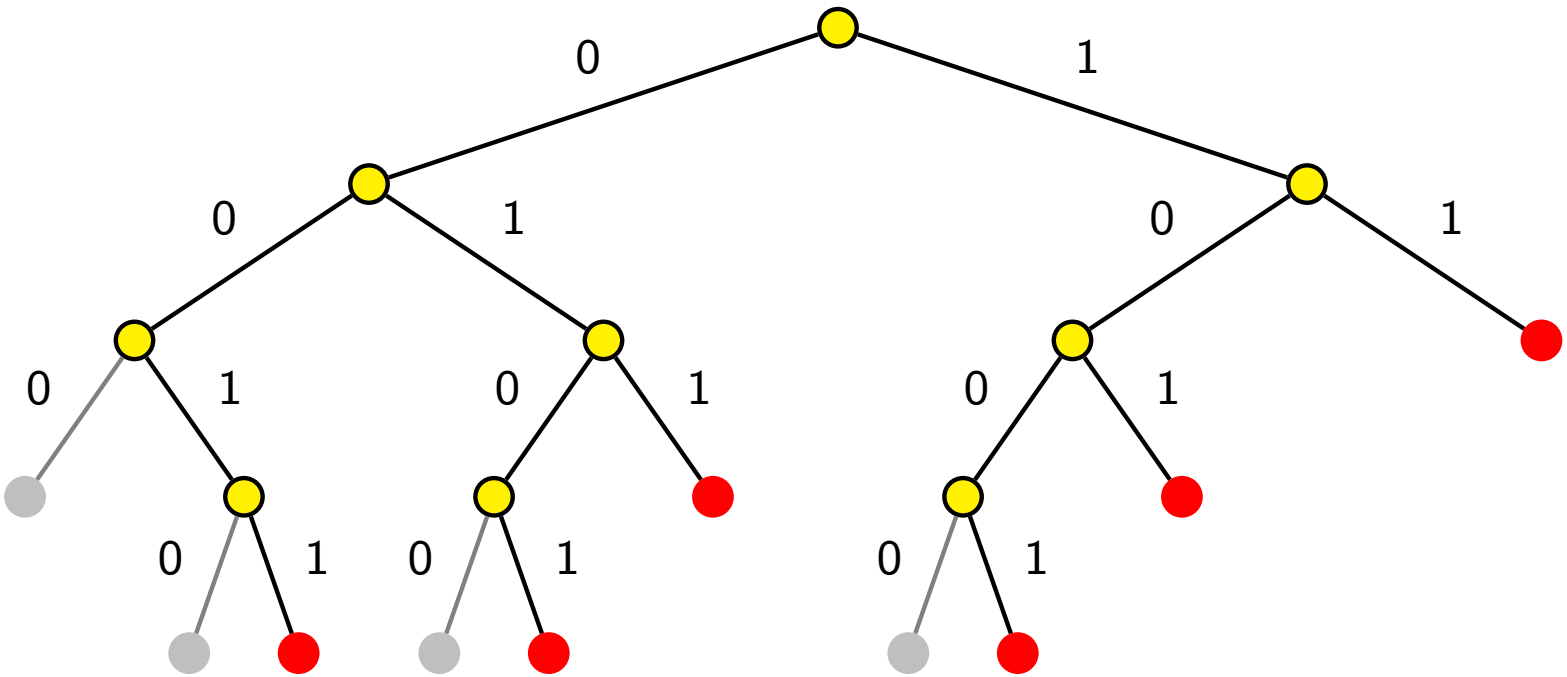
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



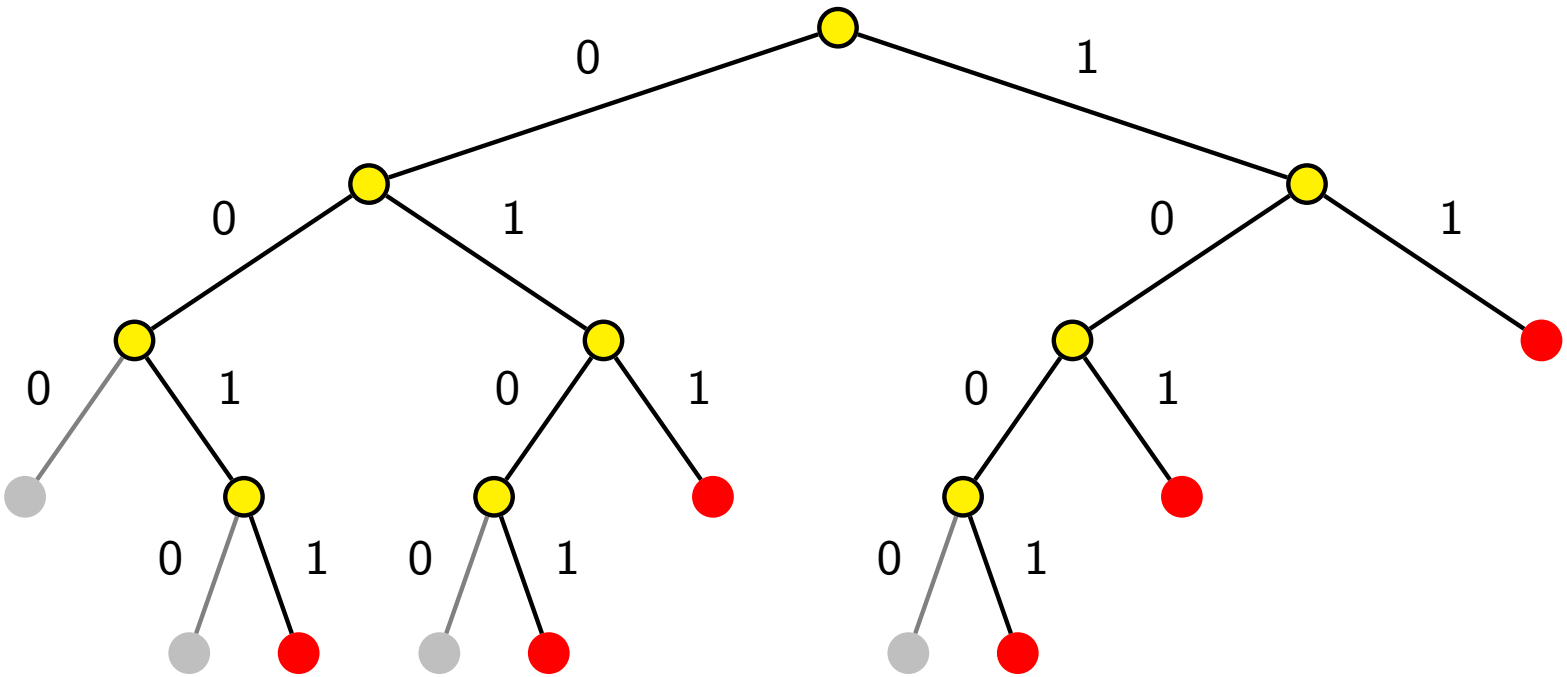
$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



$n = 4, c = 2$

Ricordiamo che generare tutti i sottoinsiemi di cardinalità c di un insieme con n elementi è perfettamente equivalente a generare tutti i vettori binari di lunghezza n che contengono esattamente c "1".



I nodi gialli corrispondono a soluzioni parziali che possono ancora portare a soluzioni complete al problema, i nodi grigi a soluzioni parziali che (se li avessimo generate) non possono portare a soluzioni complete al problema, i nodi rossi corrispondono a soluzioni complete al problema.

Usiamo la variabile *tot* =numero di elementi presenti nella soluzione parziale.

Usiamo la variabile *tot* =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n - (k + 1) + tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n - (k + 1) + tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot + 1 < c$.

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n - (k + 1) + tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot + 1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n - (k + 1) + tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot + 1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

`SOTTOINSIEMI-VINCOLATI1`(SOL, k, c, n, tot)

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n - (k + 1) + tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot + 1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
         SOL[k + 1] = 0
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
         SOL[k + 1] = 0
         SOTTOINSIEMI(SOL, k + 1, c, n, tot)
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
         SOL[k + 1] = 0
         SOTTOINSIEMI(SOL, k + 1, c, n, tot)
         }
         IF(tot < c) {
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
         SOL[k + 1] = 0
         SOTTOINSIEMI(SOL, k + 1, c, n, tot)
         }
         IF(tot < c) {
         SOL[k + 1] = 1
```

Usiamo la variabile tot =numero di elementi presenti nella soluzione parziale.

La funzione di taglio per il sottoalbero sinistro è la verifica che $n-(k+1)+tot \geq c$ (ciò assicura che restano ancora da considerare un numero di oggetti sufficienti a completare la soluzione parziale in modo da ottenere un sottoinsieme di almeno c oggetti).

La funzione di taglio per il sottoalbero destro è la verifica che $tot+1 < c$.

In questo modo siamo sicuri che durante la visita dell'albero delle soluzioni un nodo dell'albero viene visitato se e solo se nel suo sottoalbero è presente almeno una soluzione ammissibile

```
SOTTOINSIEMI-VINCOLATI1(SOL, k, c, n, tot)
IF(k = n){stampa il vettore SOL
    }ELSE{
      IF(n − (k + 1) + tot ≥ c){
          SOL[k + 1] = 0
          SOTTOINSIEMI(SOL, k + 1, c, n, tot)
          }
          IF(tot < c) {
          SOL[k + 1] = 1
          SOTTOINSIEMI-VINCOLATI1(SOL, k + 1, c, n, tot + 1)
    }     }
```

Teorema

Sia h l'altezza dell'albero delle soluzioni,

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia.

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Prova. Poichè un nodo dell'albero viene visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, il sottoalbero effettivamente visitato avrà come foglie le $D(n)$ soluzioni ammissibili.

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Prova. Poichè un nodo dell'albero viene visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, il sottoalbero effettivamente visitato avrà come foglie le $D(n)$ soluzioni ammissibili.

Ogni nodo interno visitato appartiene ad almeno uno dei cammini radice-foglia ciascuno dei quali ha lunghezza al più h.

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Prova. Poichè un nodo dell'albero viene visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, il sottoalbero effettivamente visitato avrà come foglie le $D(n)$ soluzioni ammissibili.

Ogni nodo interno visitato appartiene ad almeno uno dei cammini radice-foglia ciascuno dei quali ha lunghezza al più h. Quindi i nodi interni visitati sono al più $hD(n)$.

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Prova. Poichè un nodo dell'albero viene visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, il sottoalbero effettivamente visitato avrà come foglie le $D(n)$ soluzioni ammissibili.

Ogni nodo interno visitato appartiene ad almeno uno dei cammini radice-foglia ciascuno dei quali ha lunghezza al più h. Quindi i nodi interni visitati sono al più $hD(n)$.

Il costo della visita è dato dalla somma dei costi di visita dei nodi interni e delle foglie effettivamente visitate.

Teorema

Sia h l'altezza dell'albero delle soluzioni, $O(f(n))$ il tempo richiesto dalla visita di un suo nodo interno ed $O(g(n))$ il tempo richiesto dalla visita di una sua foglia. Se nel corso della visita vengono visitati **esclusivamente** *nodi nei cui sottoalberi sono presenti soluzioni ammissibili, il tempo richiesto dalla visita è $O(hf(n)D(n) + g(n)D(n))$ dove $D(n)$ è il numero di soluzioni ammissibili.*

Prova. Poichè un nodo dell'albero viene visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, il sottoalbero effettivamente visitato avrà come foglie le $D(n)$ soluzioni ammissibili.

Ogni nodo interno visitato appartiene ad almeno uno dei cammini radice-foglia ciascuno dei quali ha lunghezza al più h. Quindi i nodi interni visitati sono al più $hD(n)$.

Il costo della visita è dato dalla somma dei costi di visita dei nodi interni e delle foglie effettivamente visitate.

Da quanto detto questo costo è $O(hf(n)D(n) + g(n)D(n))$.

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$
- il costo per la visita di una foglia è $O(n)$

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$
- il costo per la visita di una foglia è $O(n)$
- il numero $D(n)$ di soluzioni ammissibili (vale a dire sottoinsiemi con esattamente c degli n oggetti) è $\binom{n}{c} = O(n^c)$

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$
- il costo per la visita di una foglia è $O(n)$
- il numero $D(n)$ di soluzioni ammissibili (vale a dire sottoinsiemi con esattamente c degli n oggetti) è $\binom{n}{c} = O(n^c)$
- Dal Teorema precedente quindi che la complessità di tempo della procedura è $O(n^{c+1})$

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$
- il costo per la visita di una foglia è $O(n)$
- il numero $D(n)$ di soluzioni ammissibili (vale a dire sottoinsiemi con esattamente c degli n oggetti) è $\binom{n}{c} = O(n^c)$
- Dal Teorema precedente quindi che la complessità di tempo della procedura è $O(n^{c+1})$
- che è, in generale, moloto inferiore a $O(n2^n)$ del primo algoritmo che generava prima tutti i n sottoinsiemi e stampava solo quelli che rispettavano i vincoli sulla cardinalità

Ritornando al problema del calcolo della complessità della procedura `SOTTOINSIEMI-VINCOLATI1` notiamo che:

- l'albero delle soluzioni ha altezza n,
- il costo per la visita di un nodo interno è $O(1)$
- il costo per la visita di una foglia è $O(n)$
- il numero $D(n)$ di soluzioni ammissibili (vale a dire sottoinsiemi con esattamente c degli n oggetti) è $\binom{n}{c} = O(n^c)$
- Dal Teorema precedente quindi che la complessità di tempo della procedura è $O(n^{c+1})$
- che è, in generale, moloto inferiore a $O(n2^n)$ del primo algoritmo che generava prima tutti i n sottoinsiemi e stampava solo quelli che rispettavano i vincoli sulla cardinalità (ricordiamo che le soluzioni ammissibili sono $\Theta(n^c)$).

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Il problema è di enumerare tutti i cicli hamiltoniani di un dato grafo $G = (V, E)$.

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Il problema è di enumerare tutti i cicli hamiltoniani di un dato grafo $G = (V, E)$.

Possiamo rappresentare i cicli hamiltoniani come permutazioni degli n nodi del grafo G.

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Il problema è di enumerare tutti i cicli hamiltoniani di un dato grafo $G = (V, E)$.

Possiamo rappresentare i cicli hamiltoniani come permutazioni degli n nodi del grafo G. Ovviamente **non** tutte le $n!$ permutazioni degli n nodi sono necessariamente un ciclo.

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Il problema è di enumerare tutti i cicli hamiltoniani di un dato grafo $G = (V, E)$.

Possiamo rappresentare i cicli hamiltoniani come permutazioni degli n nodi del grafo G. Ovviamente **non** tutte le $n!$ permutazioni degli n nodi sono necessariamente un ciclo. Un grafo completo ha $(n-1)!$ cicli hamiltoniani

Quarto esempio: Cicli hamiltoniani

Come altro esempio di utilizzo di funzioni di taglio nell'enumerazione di soluzioni ammissibili consideriamo ora una problema su grafi.

Sia G in grafo di n nodi. Un ciclo hamiltoniano per G è un ciclo nel grafo che attraversa esattamente n archi e tocca tutti i nodi del grafo.

Il problema è di enumerare tutti i cicli hamiltoniani di un dato grafo $G = (V, E)$.

Possiamo rappresentare i cicli hamiltoniani come permutazioni degli n nodi del grafo G. Ovviamente **non** tutte le $n!$ permutazioni degli n nodi sono necessariamente un ciclo. Un grafo completo ha $(n-1)!$ cicli hamiltoniani mentre un grafo sparso potrebbe non averne neanche uno (si pensi ad esempio ad un albero).

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Una funzione di taglio è quella che evita la generazione di permutazioni parziali in cui fra un nodo e il successivo non sia presente un arco in E

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Una funzione di taglio è quella che evita la generazione di permutazioni parziali in cui fra un nodo e il successivo non sia presente un arco in E (permutazioni di questo genere non potranno completarsi in soluzioni ammissibili).

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Una funzione di taglio è quella che evita la generazione di permutazioni parziali in cui fra un nodo e il successivo non sia presente un arco in E (permutazioni di questo genere non potranno completarsi in soluzioni ammissibili).

Inoltre, lo stesso ciclo hamiltoniano può essere rappresentato da n diverse permutazioni in funzione del nodo da cui si parte fa partire il ciclo.

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Una funzione di taglio è quella che evita la generazione di permutazioni parziali in cui fra un nodo e il successivo non sia presente un arco in E (permutazioni di questo genere non potranno completarsi in soluzioni ammissibili).

Inoltre, lo stesso ciclo hamiltoniano può essere rappresentato da n diverse permutazioni in funzione del nodo da cui si parte fa partire il ciclo. Per evitare di stampare più volte uno stesso ciclo assumiamo quindi che il primo elemento della permutazione sia sempre il vertice 1 del grafo (vale a dire $SOL[1] = 1$).

Affinchè una permutazione sia un ciclo hamiltoniano per il grafo G, tra un nodo e quello che segue nella permutazione deve esserci un arco in E e lo stesso deve valere anche per l'ultimo nodo ed il primo nodo della permutazione.

Una funzione di taglio è quella che evita la generazione di permutazioni parziali in cui fra un nodo e il successivo non sia presente un arco in E (permutazioni di questo genere non potranno completarsi in soluzioni ammissibili).

Inoltre, lo stesso ciclo hamiltoniano può essere rappresentato da n diverse permutazioni in funzione del nodo da cui si parte fa partire il ciclo. Per evitare di stampare più volte uno stesso ciclo assumiamo quindi che il primo elemento della permutazione sia sempre il vertice 1 del grafo (vale a dire $SOL[1] = 1$).

Per verificare in modo efficiente se un dato vertice è presente o meno nella soluzione parziale utilizziamo un vettore $ELEM$ i cui valori sono inizializzati a zero e porremo $ELEM[i] = 1$ quando il vertice i è posto nella soluzione parziale corrente.

Il vettore $SOL[1] \ldots S[n]$ conterrà i vertici della soluzione.

`CICLI-HAMILTONIANI`$(SOL, k, ELEM, n, G)$

Il vettore $SOL[1] \ldots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
```

Il vettore $SOL[1] \ldots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
        SOL[k + 1] = i
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
        SOL[k + 1] = i
        ELEM[i] = 1
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
       SOL[k + 1] = i
       ELEM[i] = 1
       CICLI-HAMILTONIANI(SOL, k + 1, ELEM, n, G)
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
        SOL[k + 1] = i
        ELEM[i] = 1
        CICLI-HAMILTONIANI(SOL, k + 1, ELEM, n, G)
        ELEM[i] = 0
```

Il vettore $SOL[1] \dots S[n]$ conterrà i vertici della soluzione.

```
CICLI-HAMILTONIANI(SOL, k, ELEM, n, G)
IF((k = n)&&(SOL[n], SOL[1]) ∈ E) {stampa il vettore SOL
   }ELSE{
    FOR(i=2,i<n+1,i=i+1){
    IF((ELEM[i] = 0)&&(SOL[k], i) ∈ E)
       SOL[k + 1] = i
       ELEM[i] = 1
       CICLI-HAMILTONIANI(SOL, k + 1, ELEM, n, G)
       ELEM[i] = 0
       }
    }
```

La procedura viene invocata da
`CICLI-HAMILTONIANI`$(SOL, 1, ELEM, n, G)$.

La procedura viene invocata da
`CICLI-HAMILTONIANI`$(SOL, 1, ELEM, n, G)$.

Per quanto visto sull'enumerazione di permutazioni la visita totale dell'albero richiede $O(n!)$, tuttavia grazie alla funzione di taglio l'albero viene parzialmente potato.

La procedura viene invocata da
`CICLI-HAMILTONIANI`$(SOL, 1, ELEM, n, G)$.

Per quanto visto sull'enumerazione di permutazioni la visita totale dell'albero richiede $O(n!)$, tuttavia grazie alla funzione di taglio l'albero viene parzialmente potato.

É ovvio che la funzione di taglio proposta non assicura che un nodo venga visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, quindi il Teorema non può essere utilizzato e la complessità è ancora $O(n!)$.

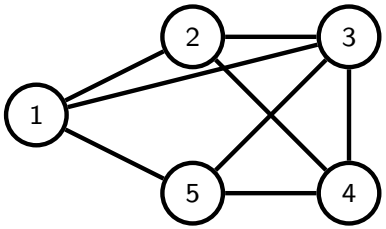
La procedura viene invocata da
`CICLI-HAMILTONIANI`$(SOL, 1, ELEM, n, G)$.

Per quanto visto sull'enumerazione di permutazioni la visita totale dell'albero richiede $O(n!)$, tuttavia grazie alla funzione di taglio l'albero viene parzialmente potato.

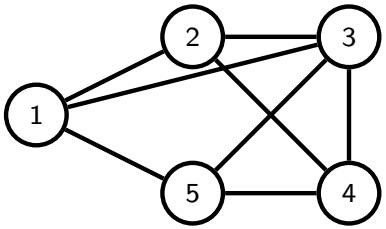
É ovvio che la funzione di taglio proposta non assicura che un nodo venga visitato solo se nel suo sottoalbero sono presenti soluzioni ammissibili, quindi il Teorema non può essere utilizzato e la complessità è ancora $O(n!)$.

Quel che si può dire è che, quanto più il grafo è sparso tanto più l'albero delle soluzioni verrà potato e i tempi richiesti della visita ridotti.

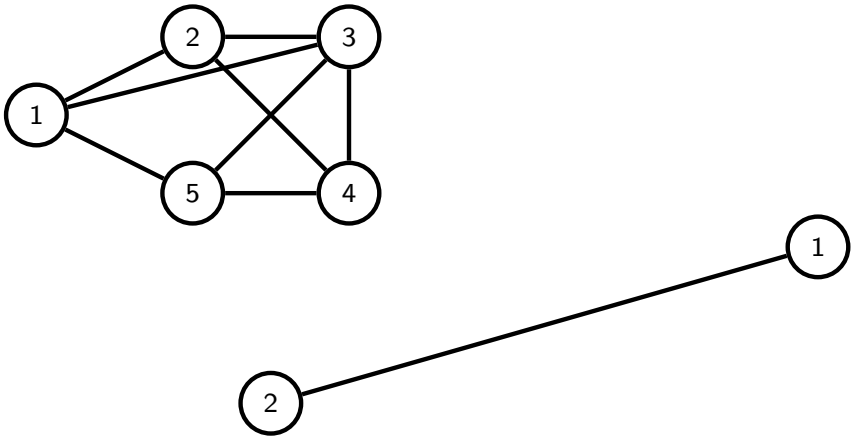
Esempio



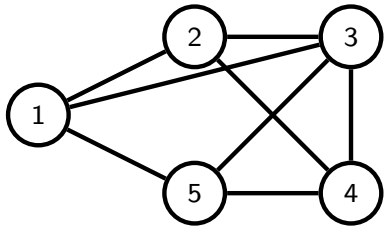
Esempio

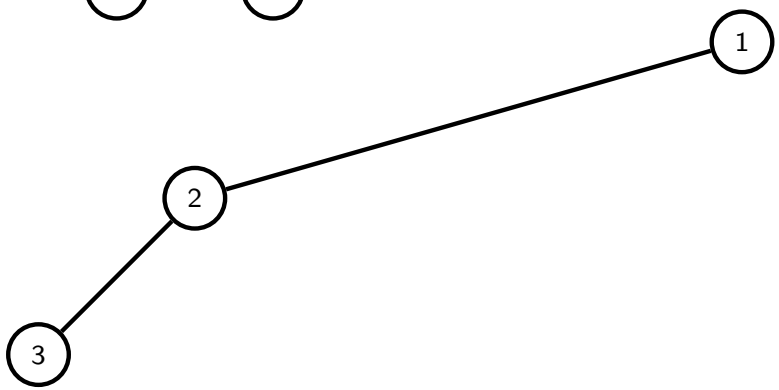


Esempio

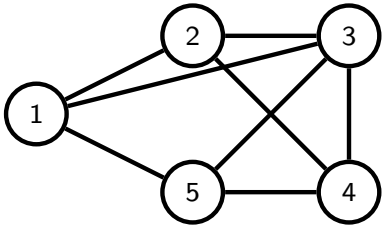


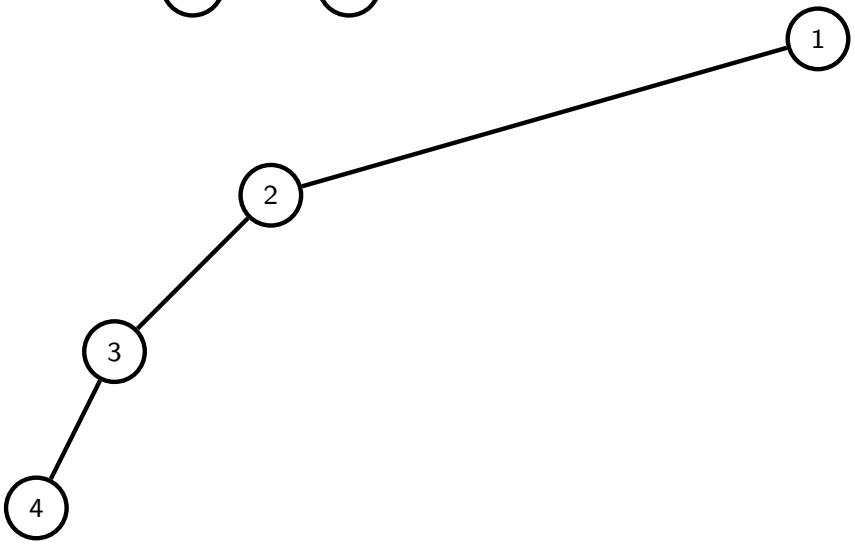
Esempio



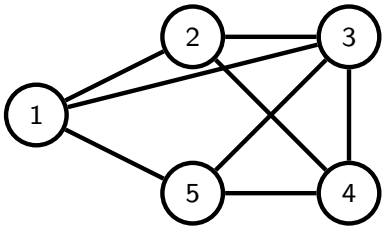


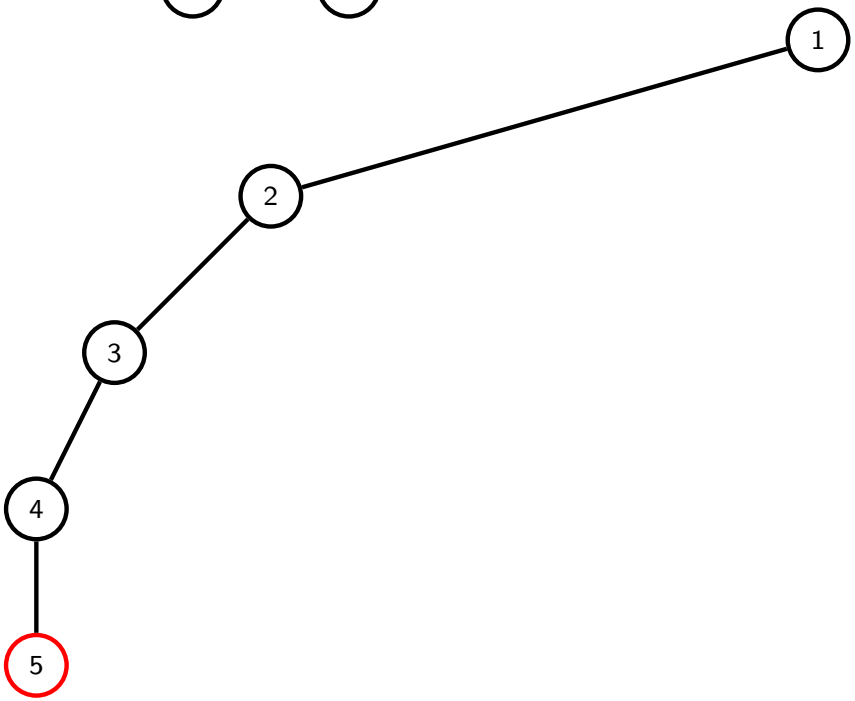
Esempio



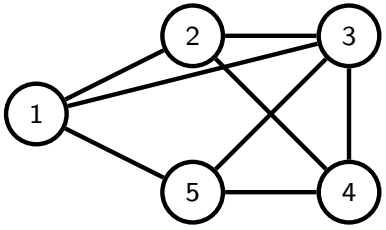


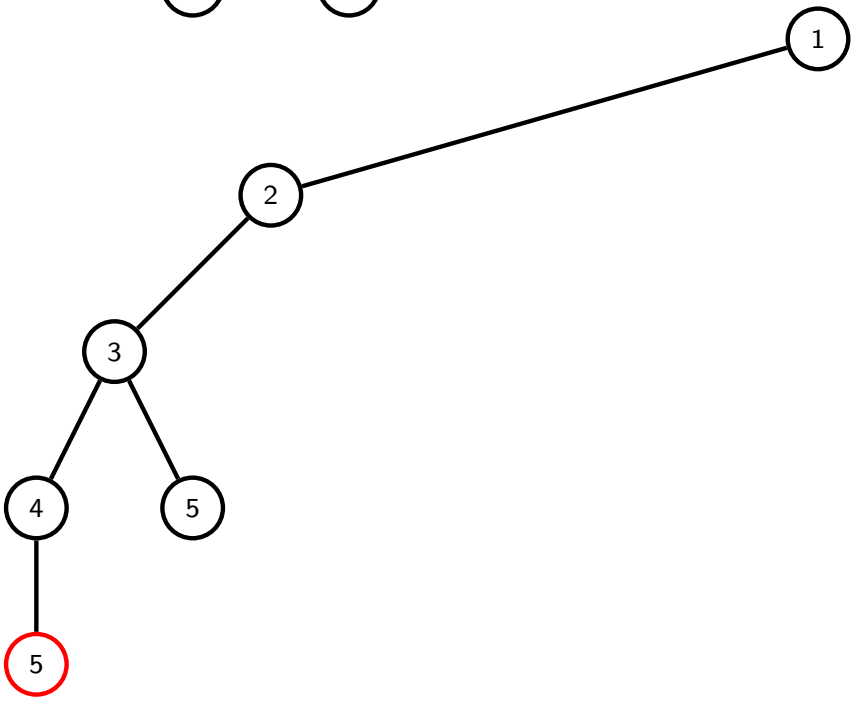
Esempio



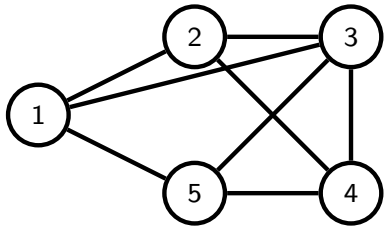


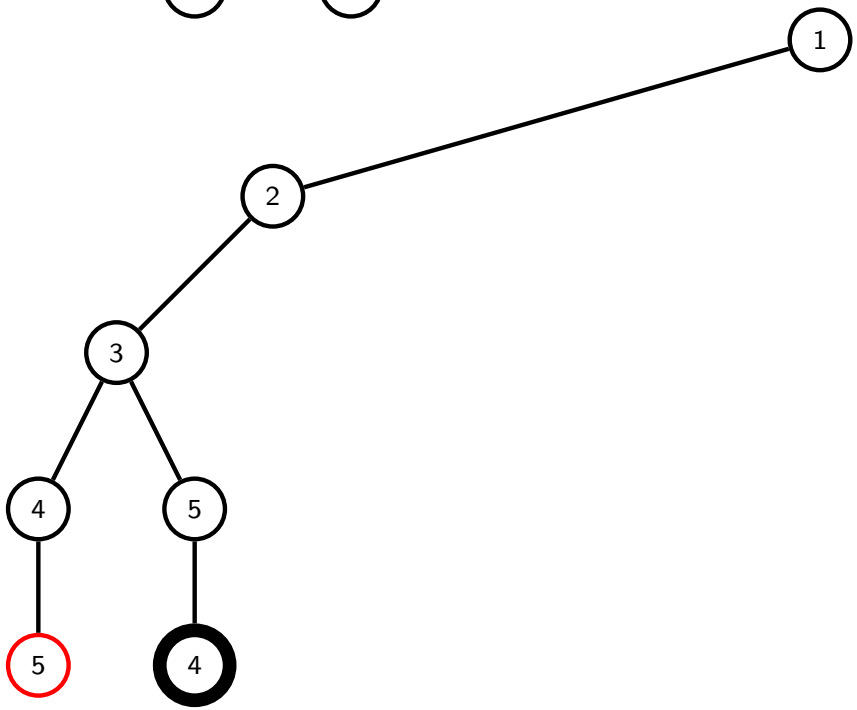
Esempio



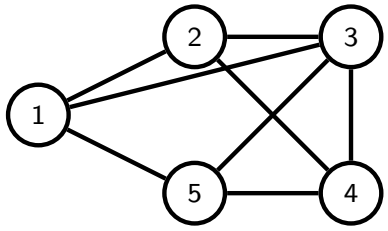


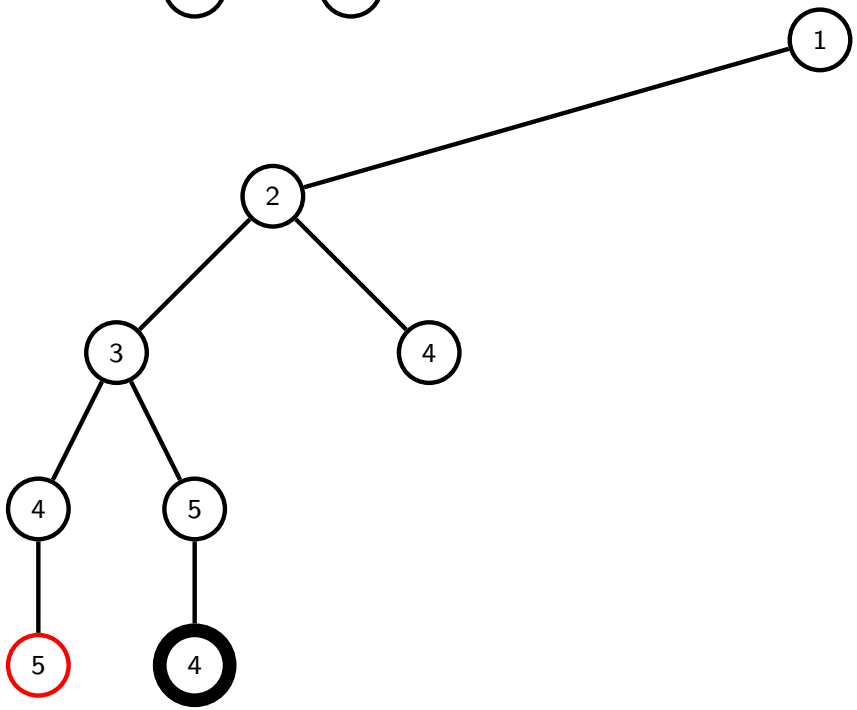
Esempio



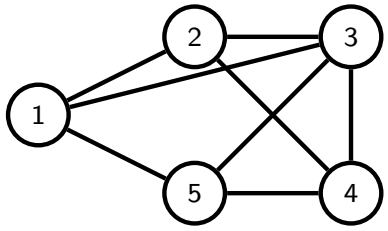


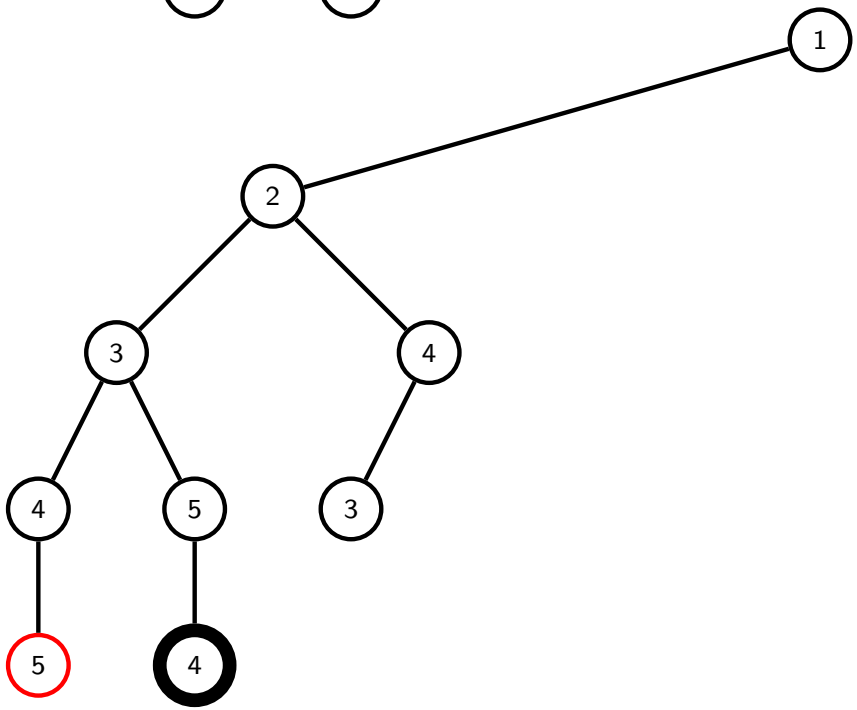
Esempio



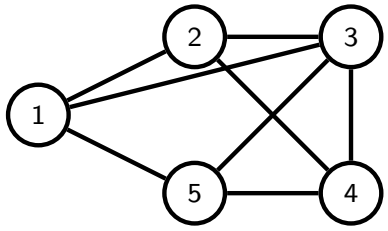


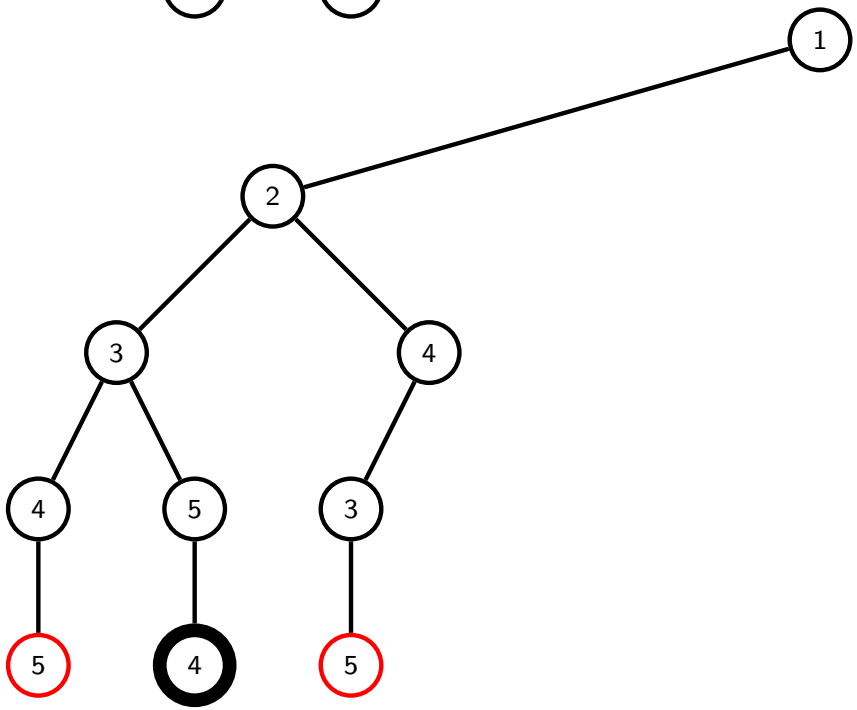
Esempio



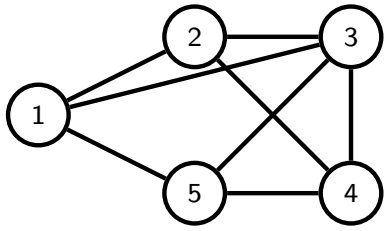


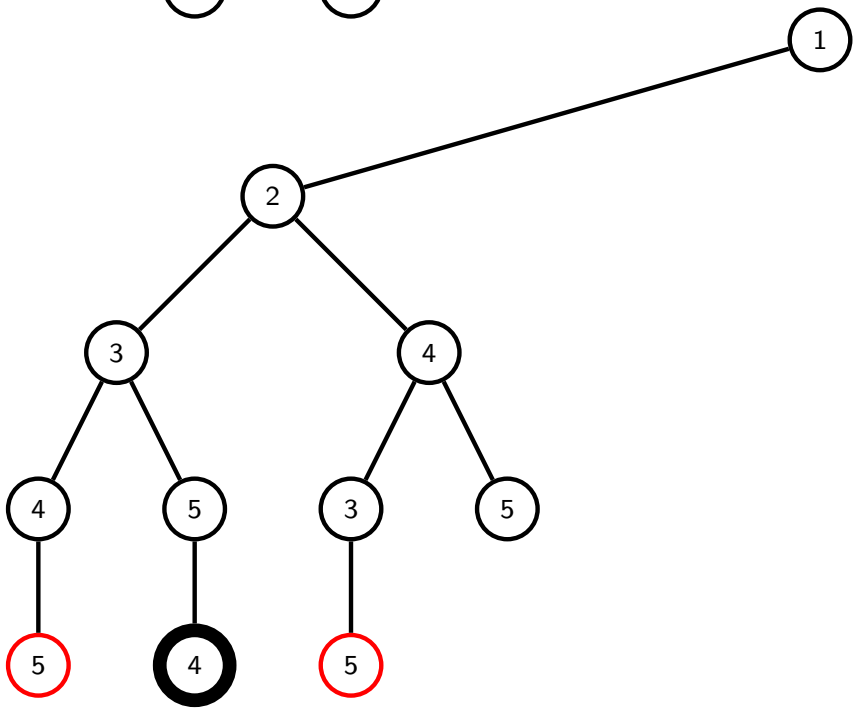
Esempio



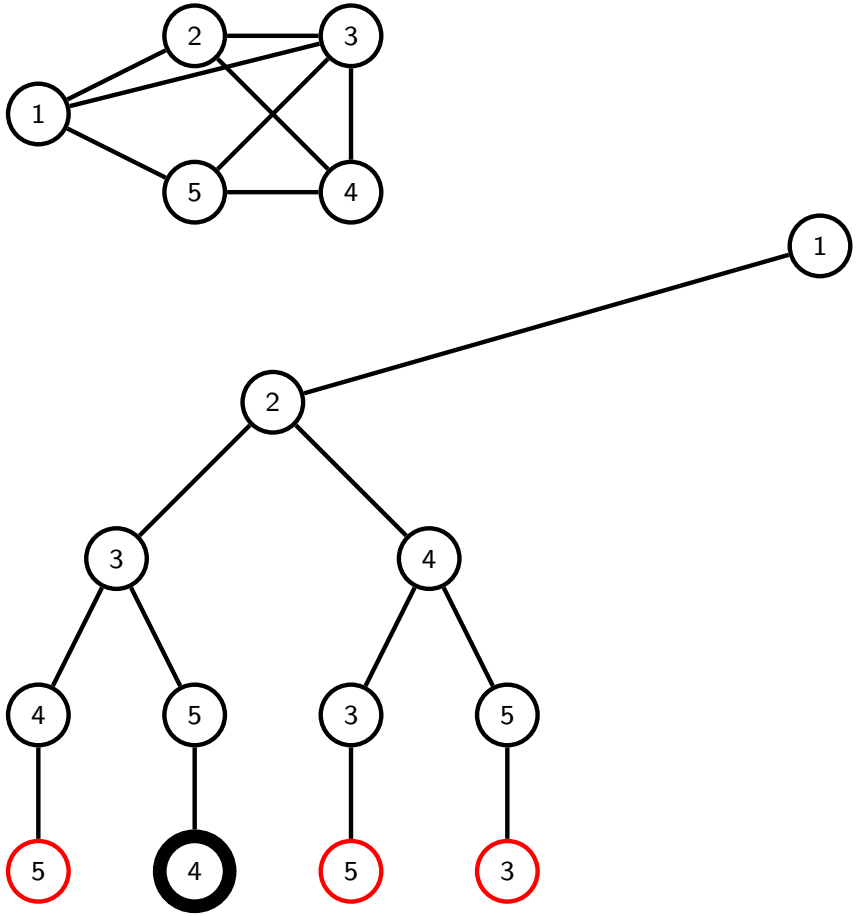


Esempio

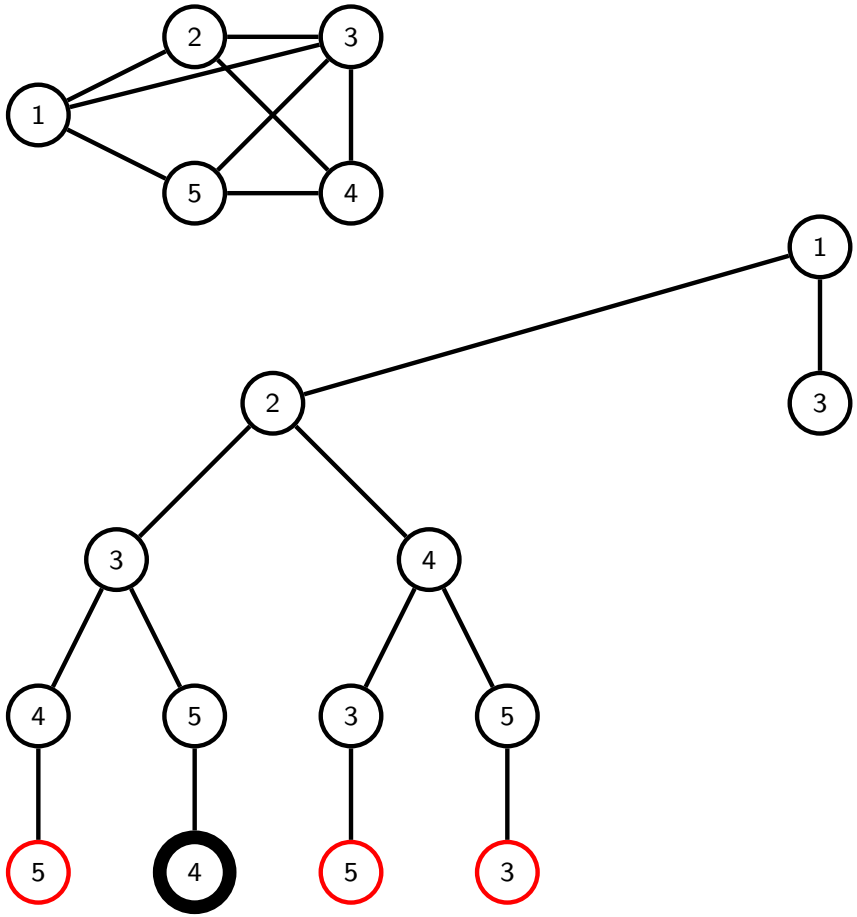




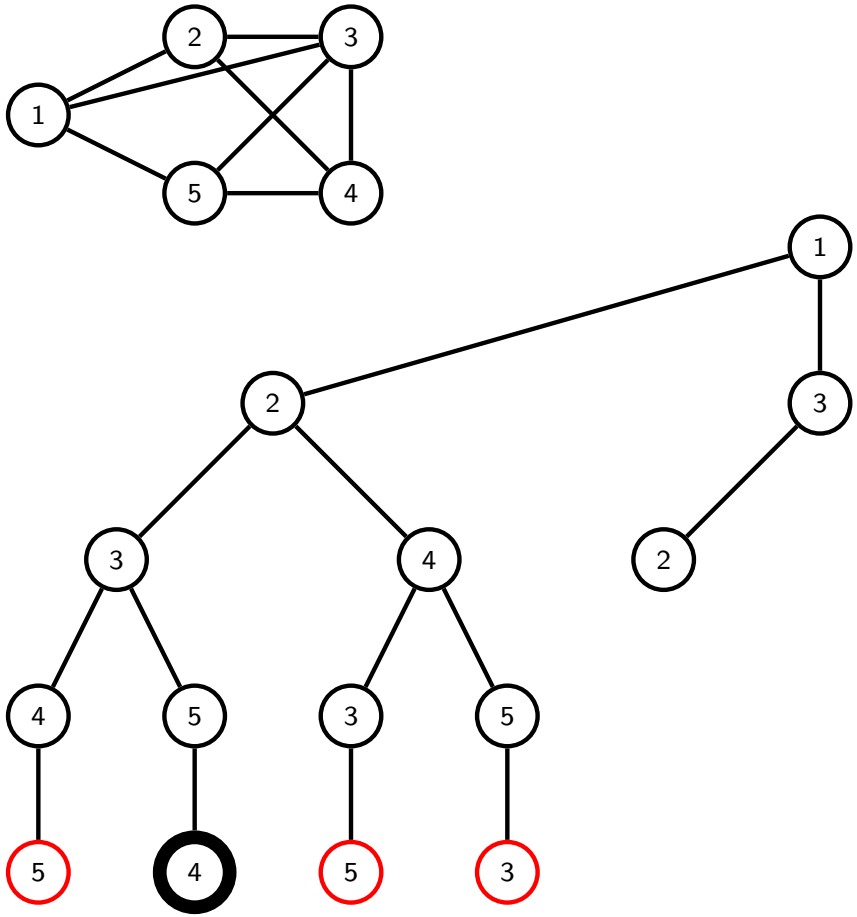
Esempio



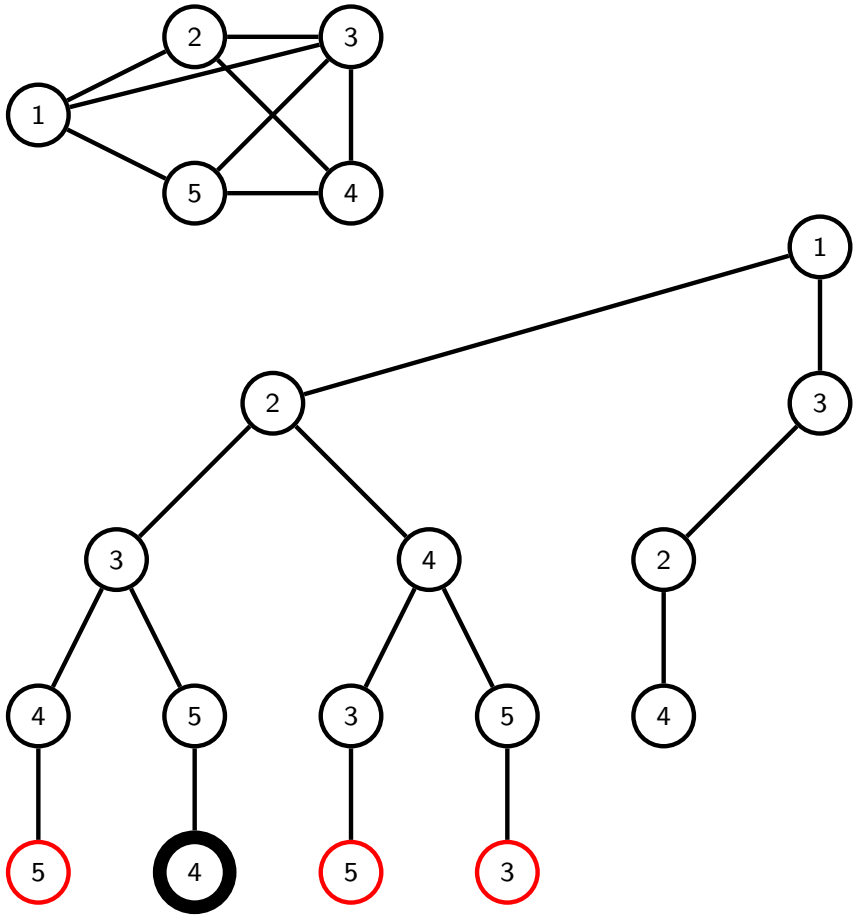
Esempio



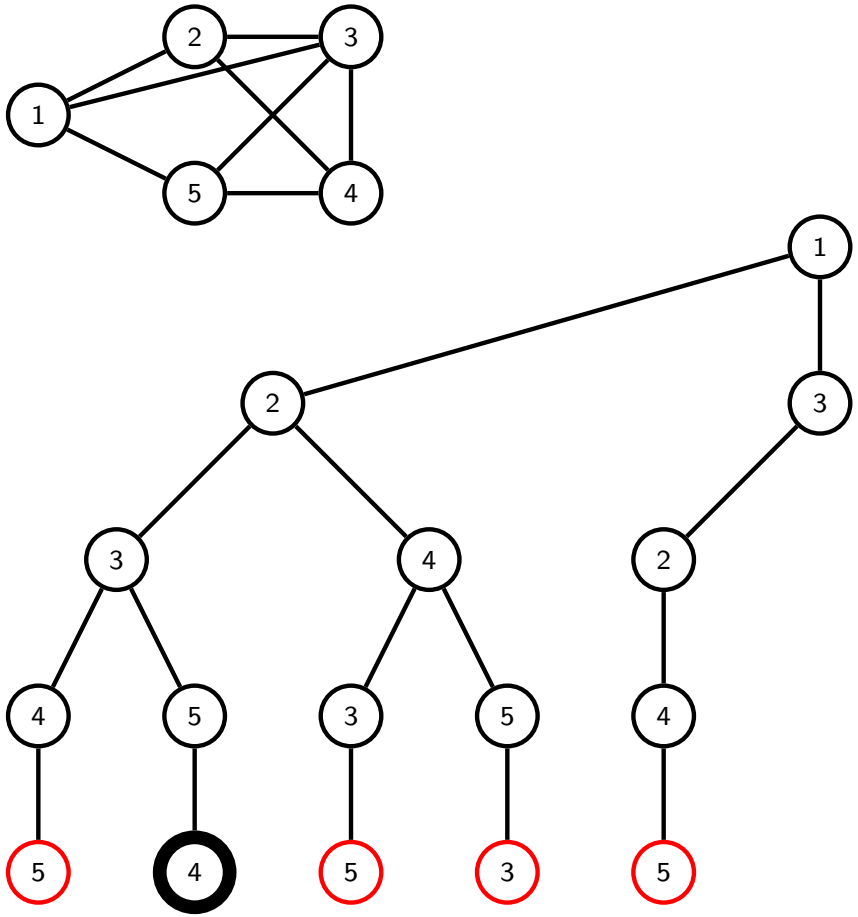
Esempio



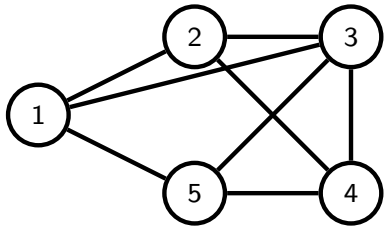
Esempio

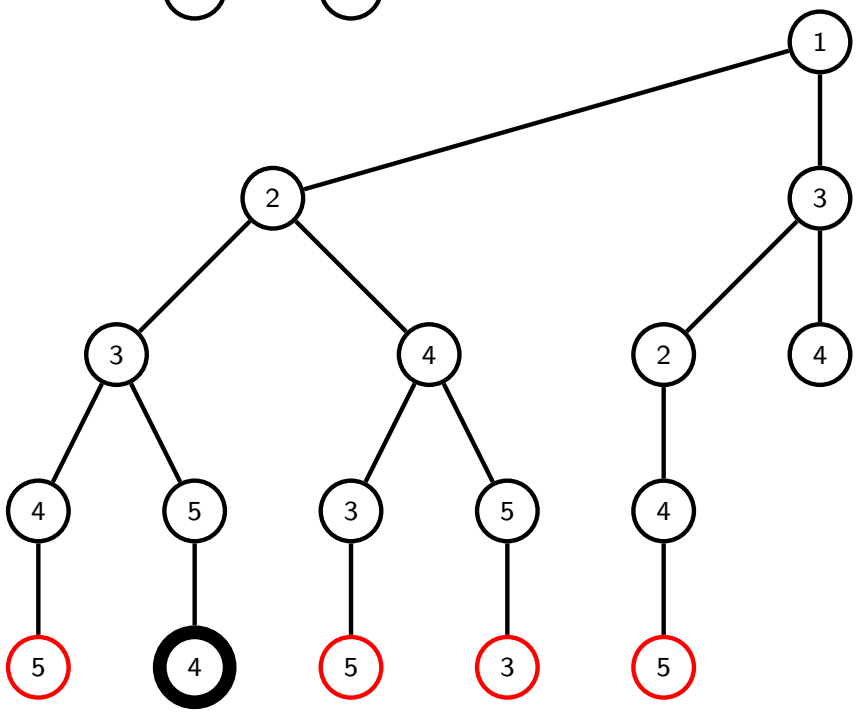


Esempio

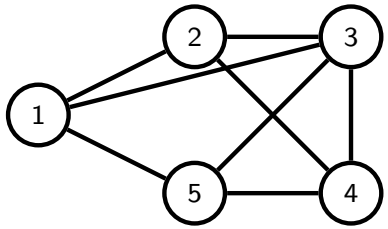


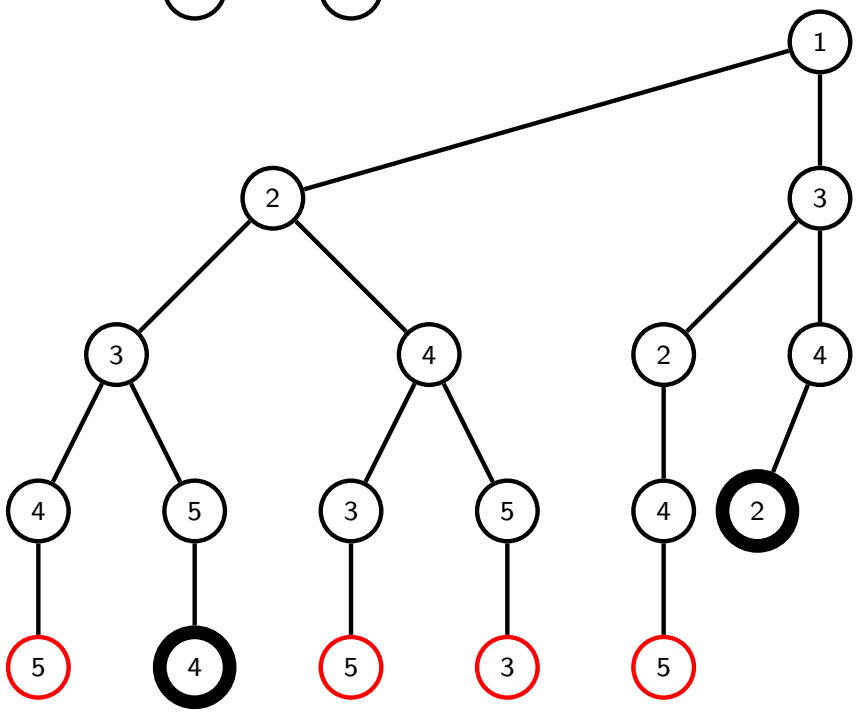
Esempio



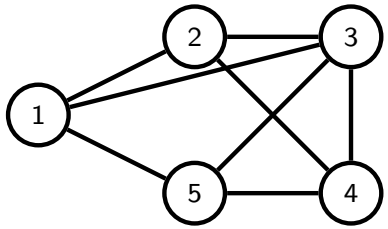


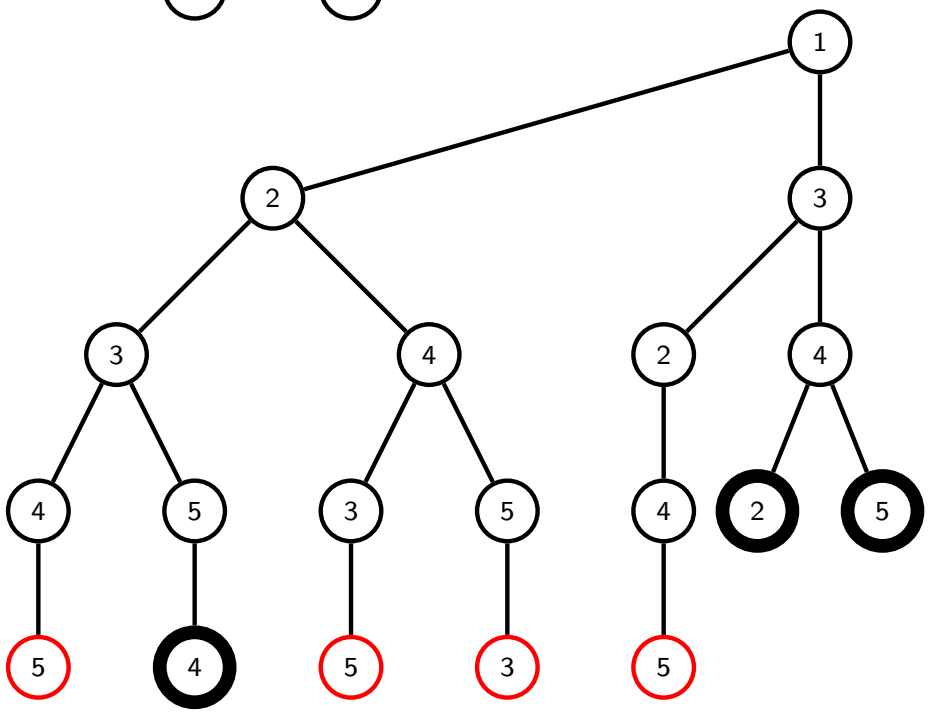
Esempio



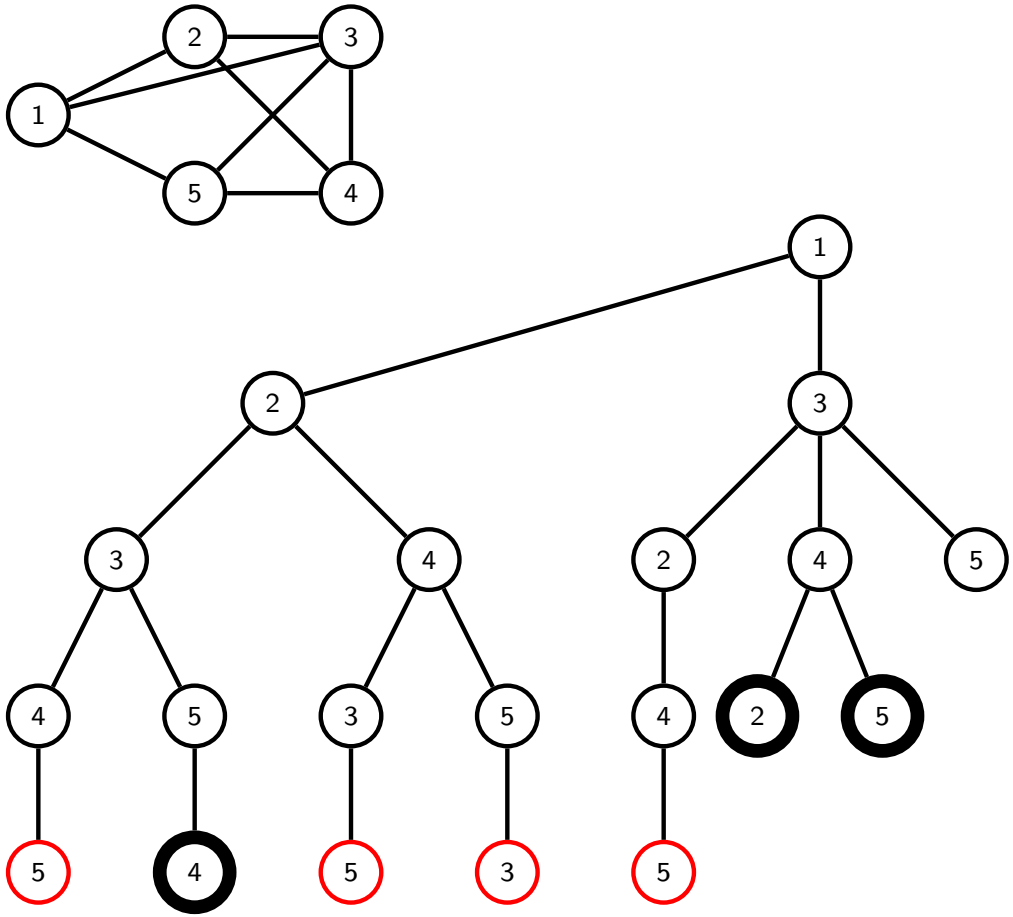


Esempio

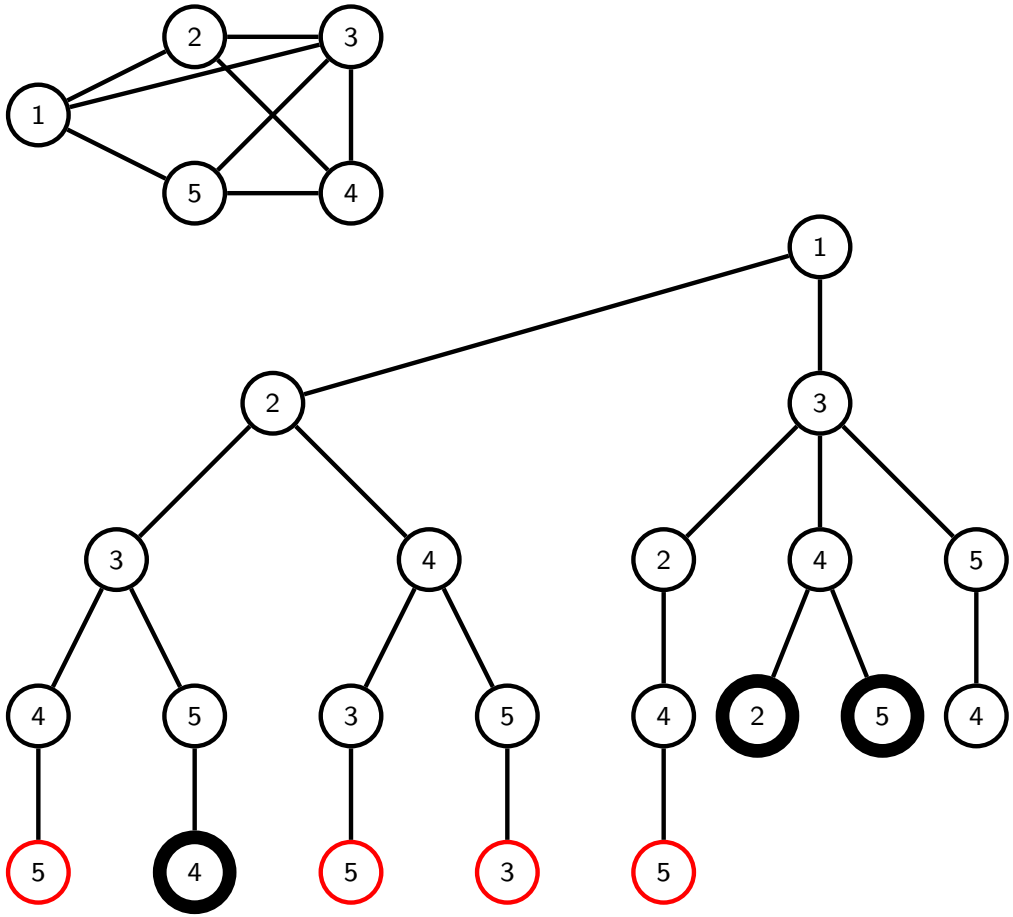




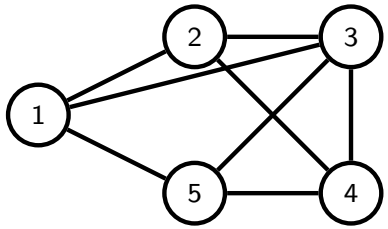
Esempio

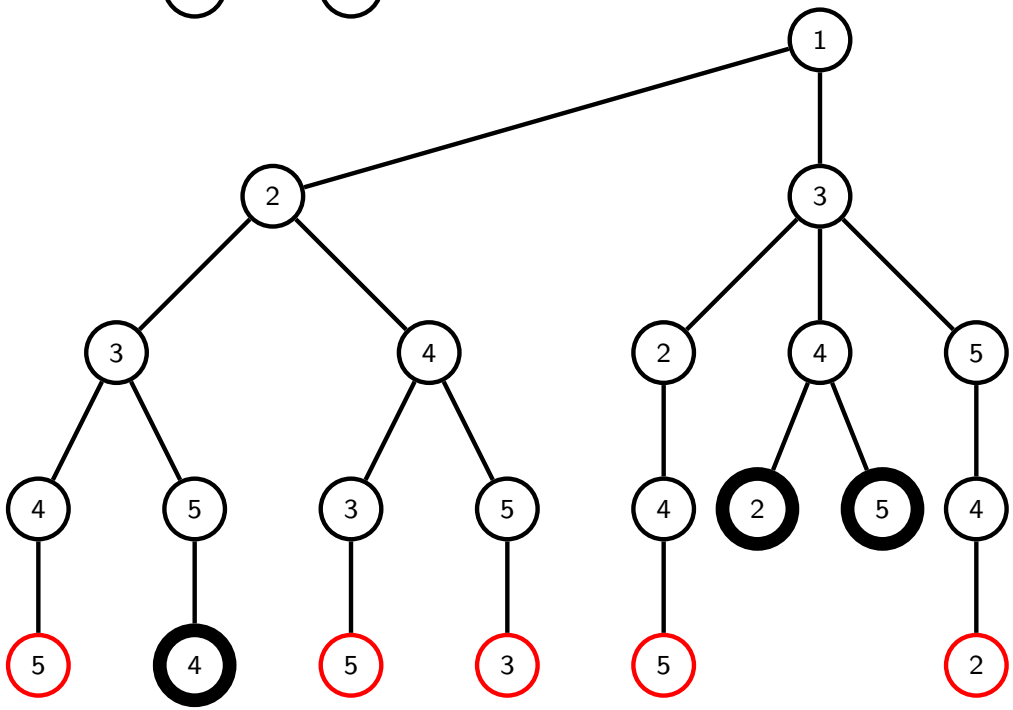


Esempio

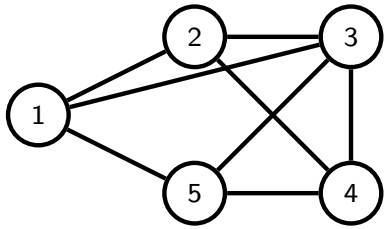


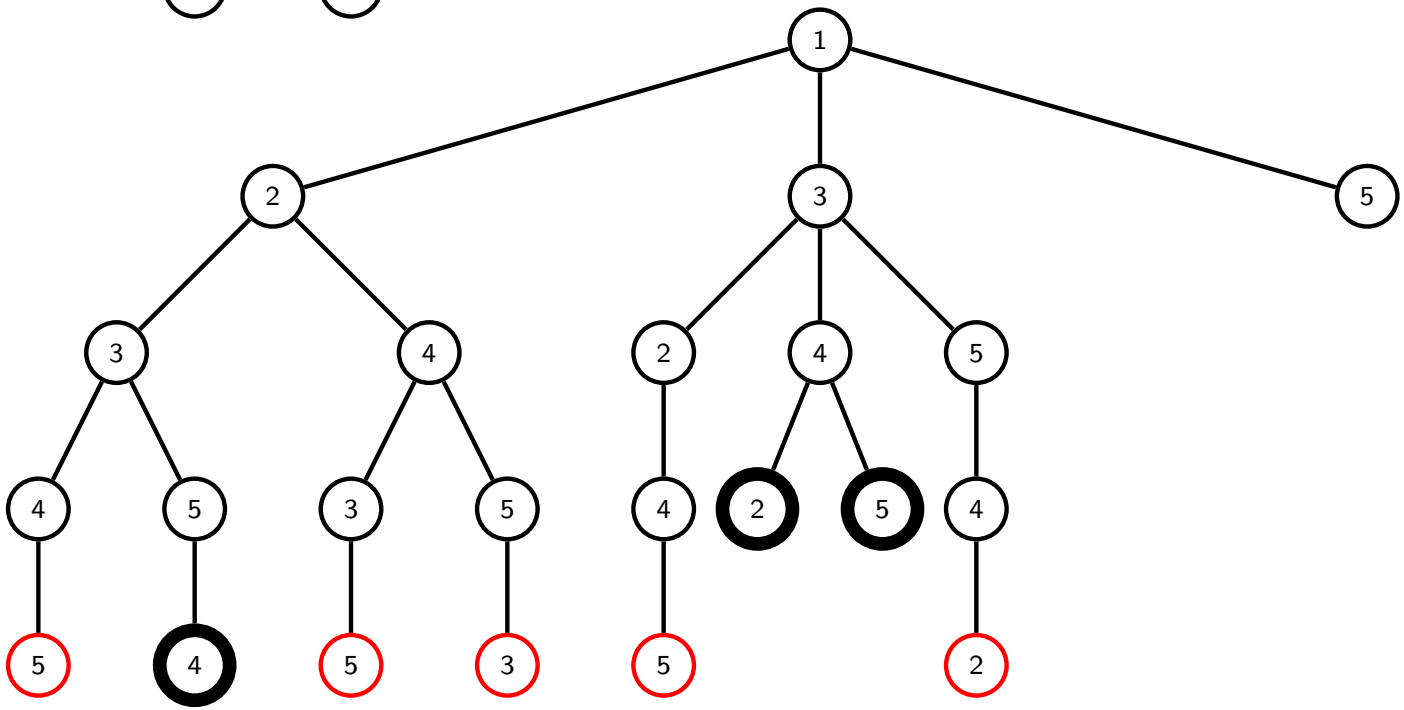
Esempio



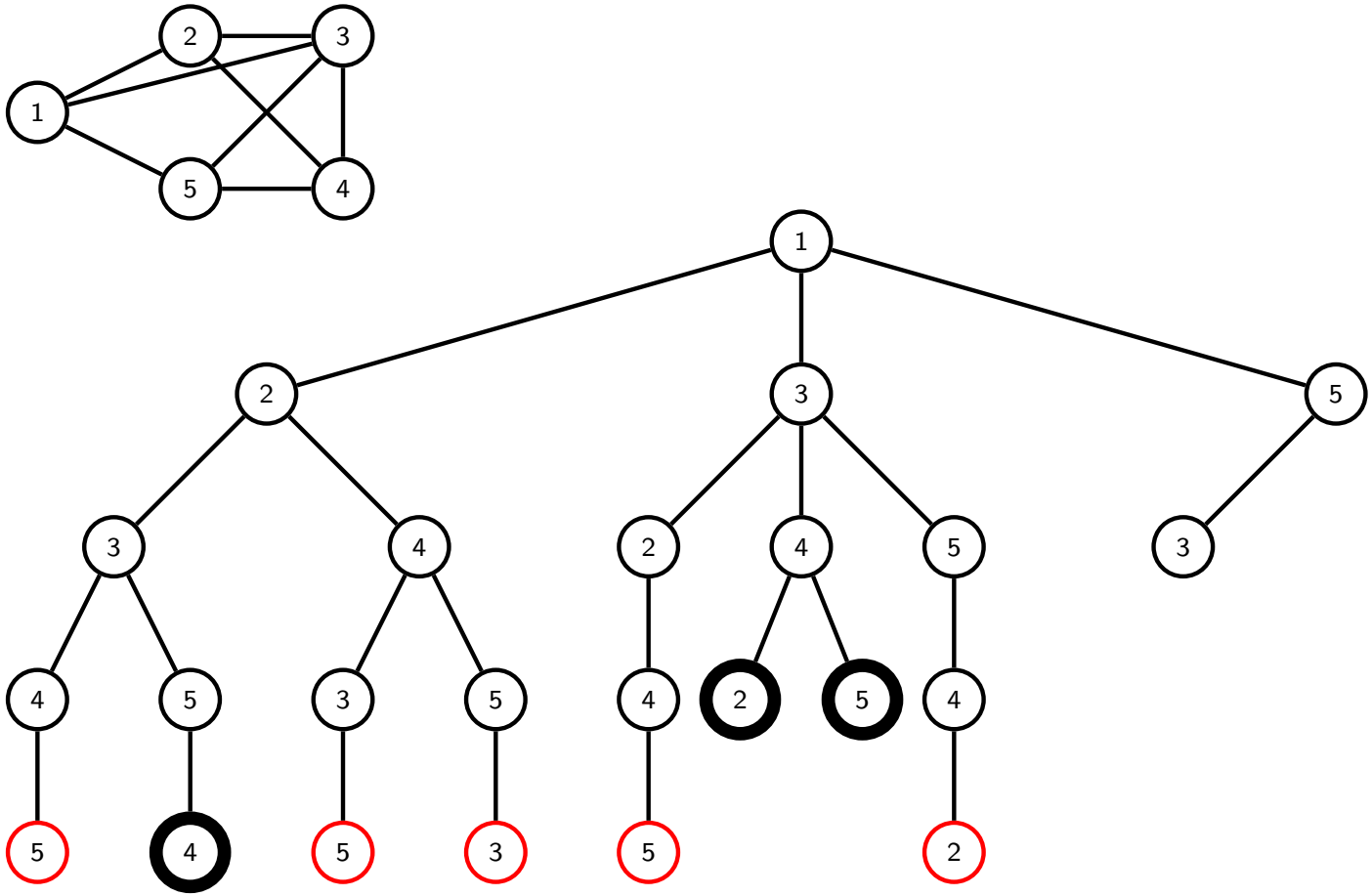


Esempio

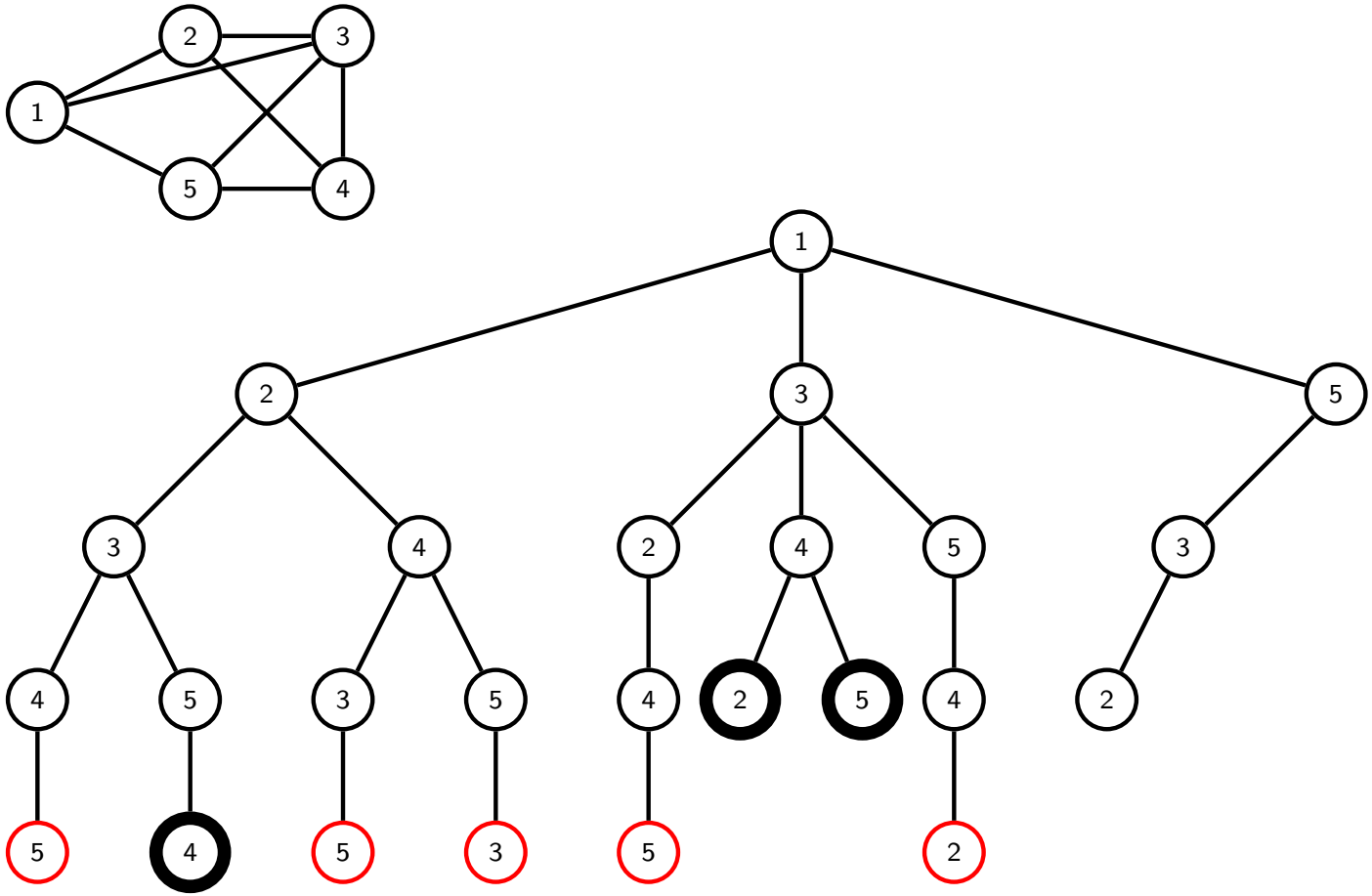




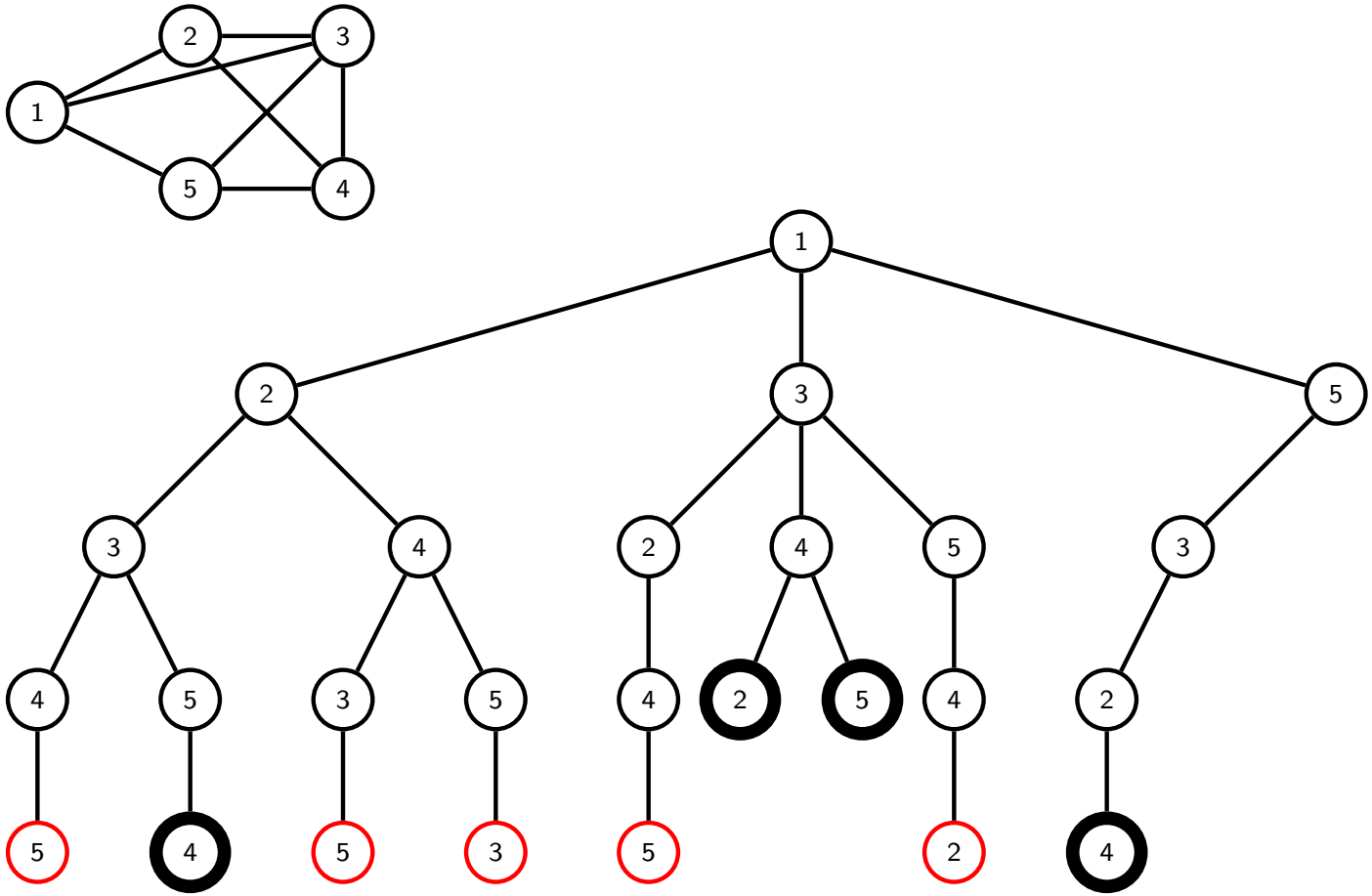
Esempio



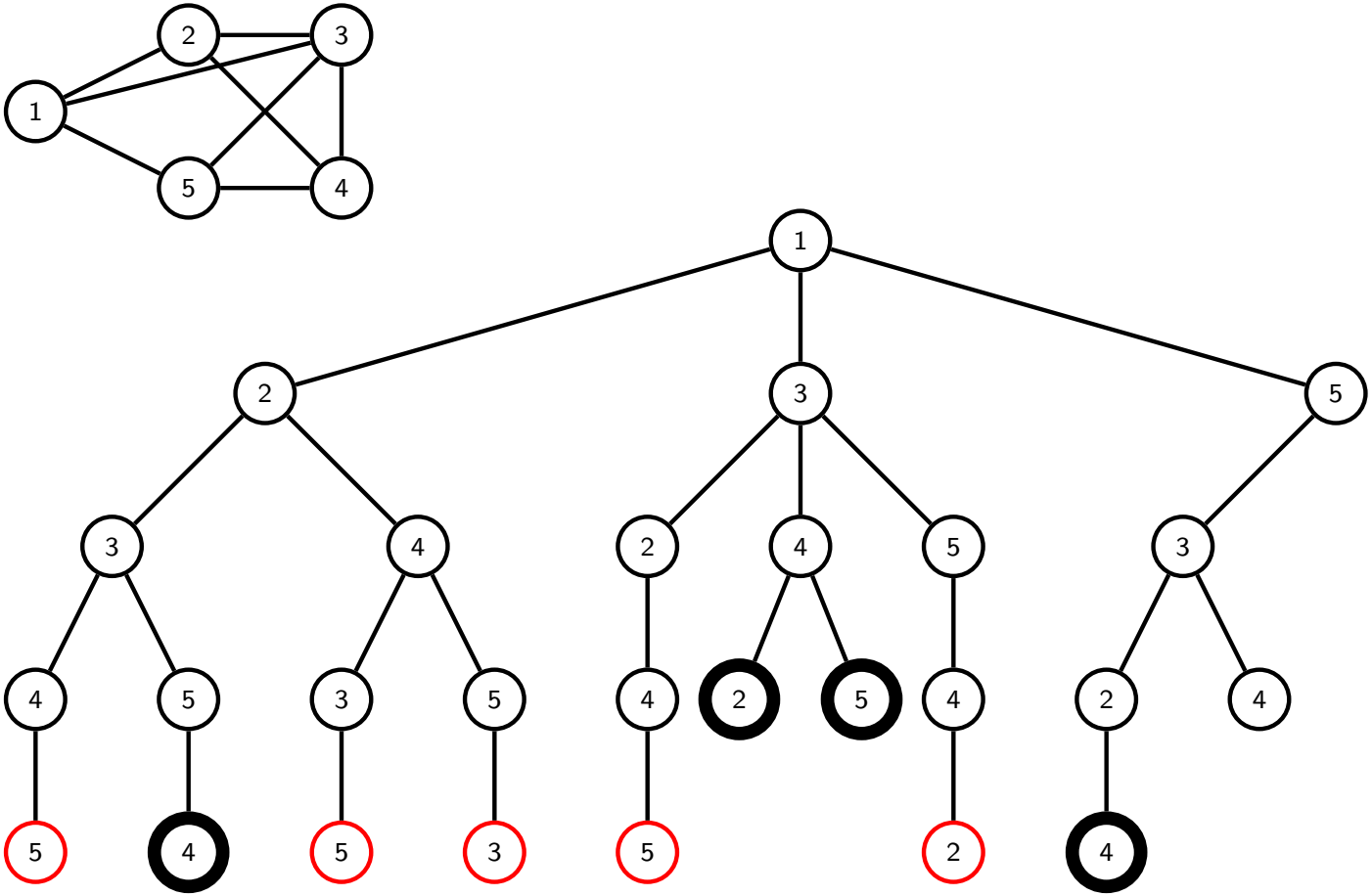
Esempio



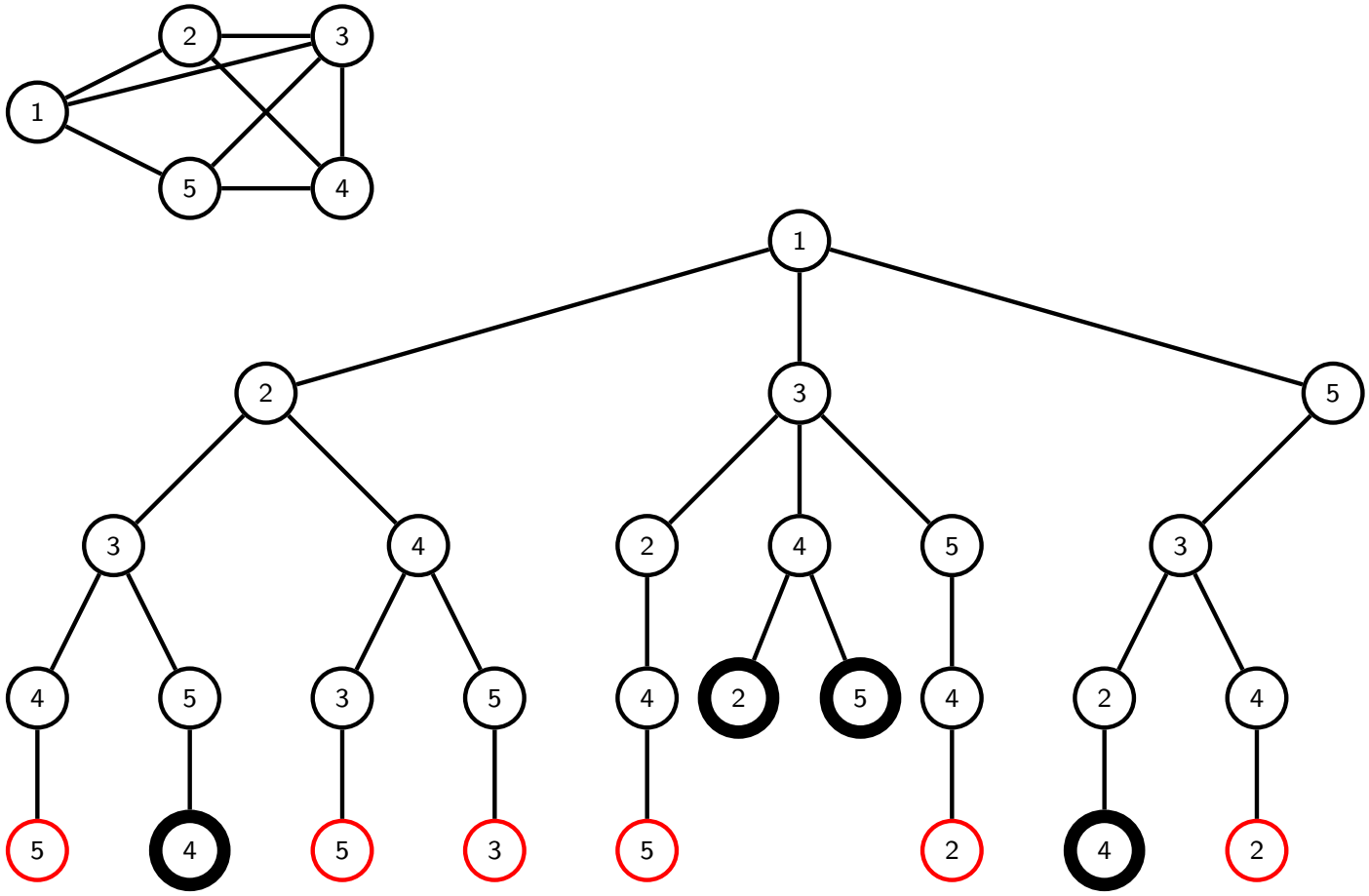
Esempio



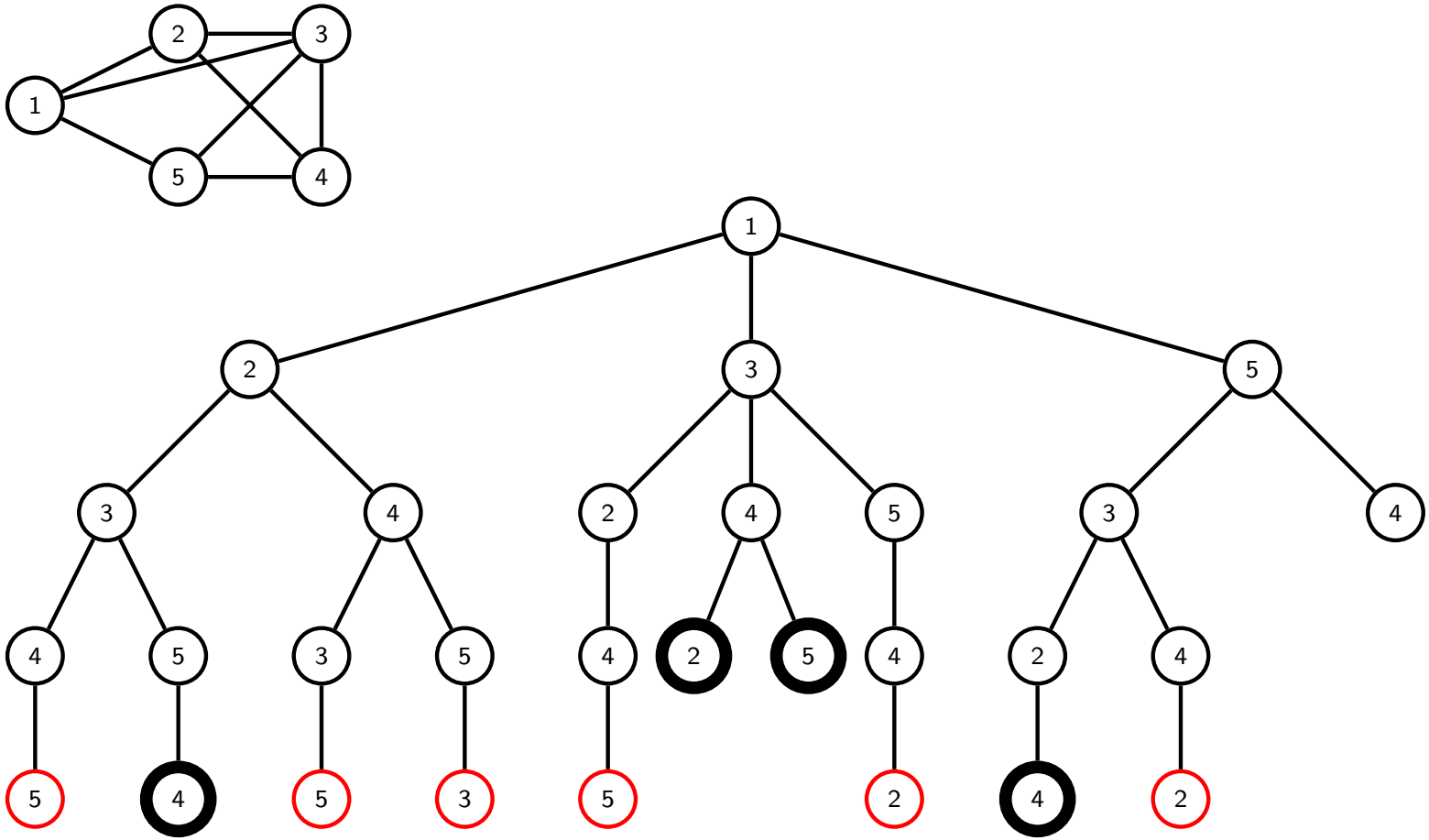
Esempio



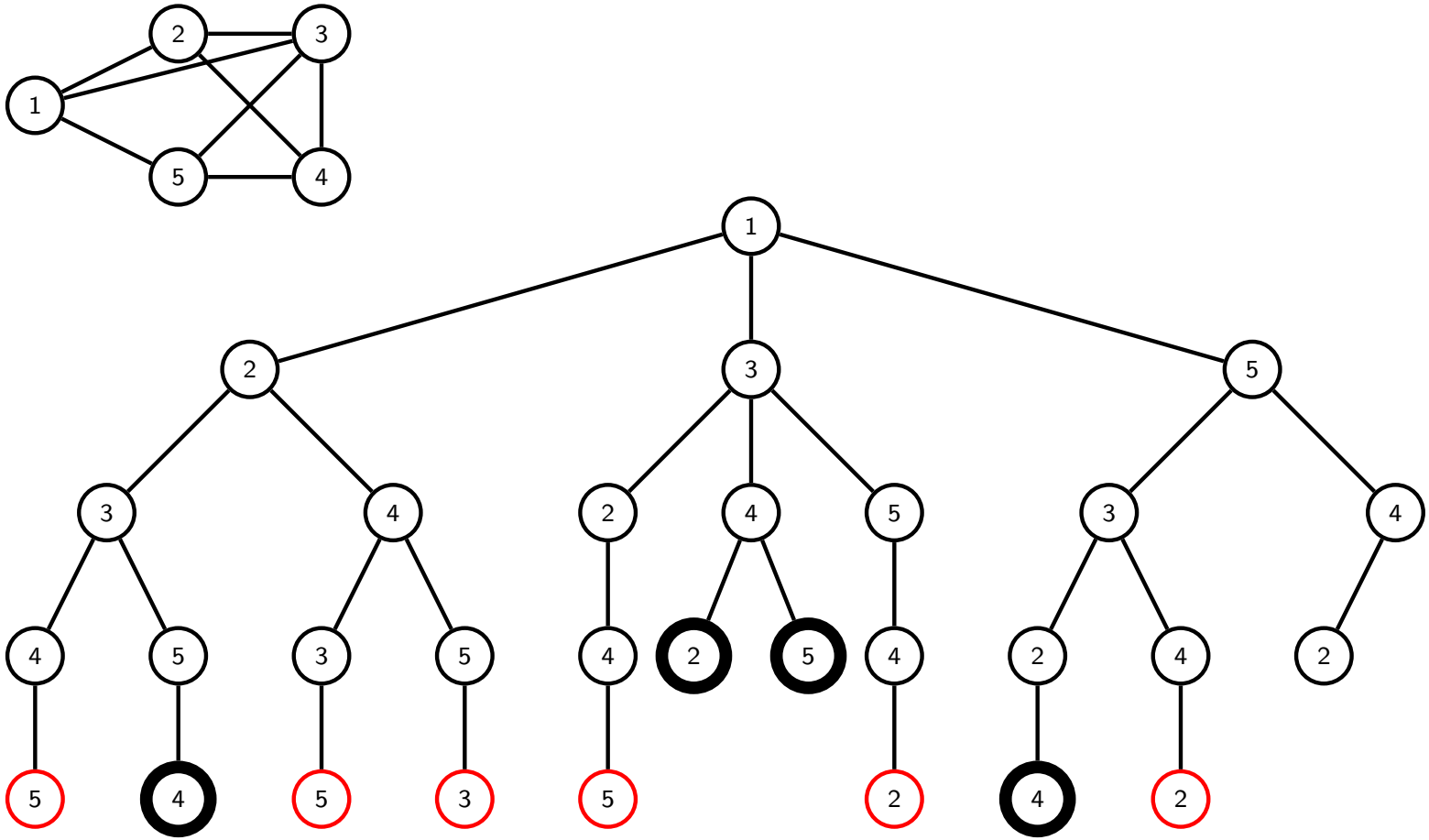
Esempio



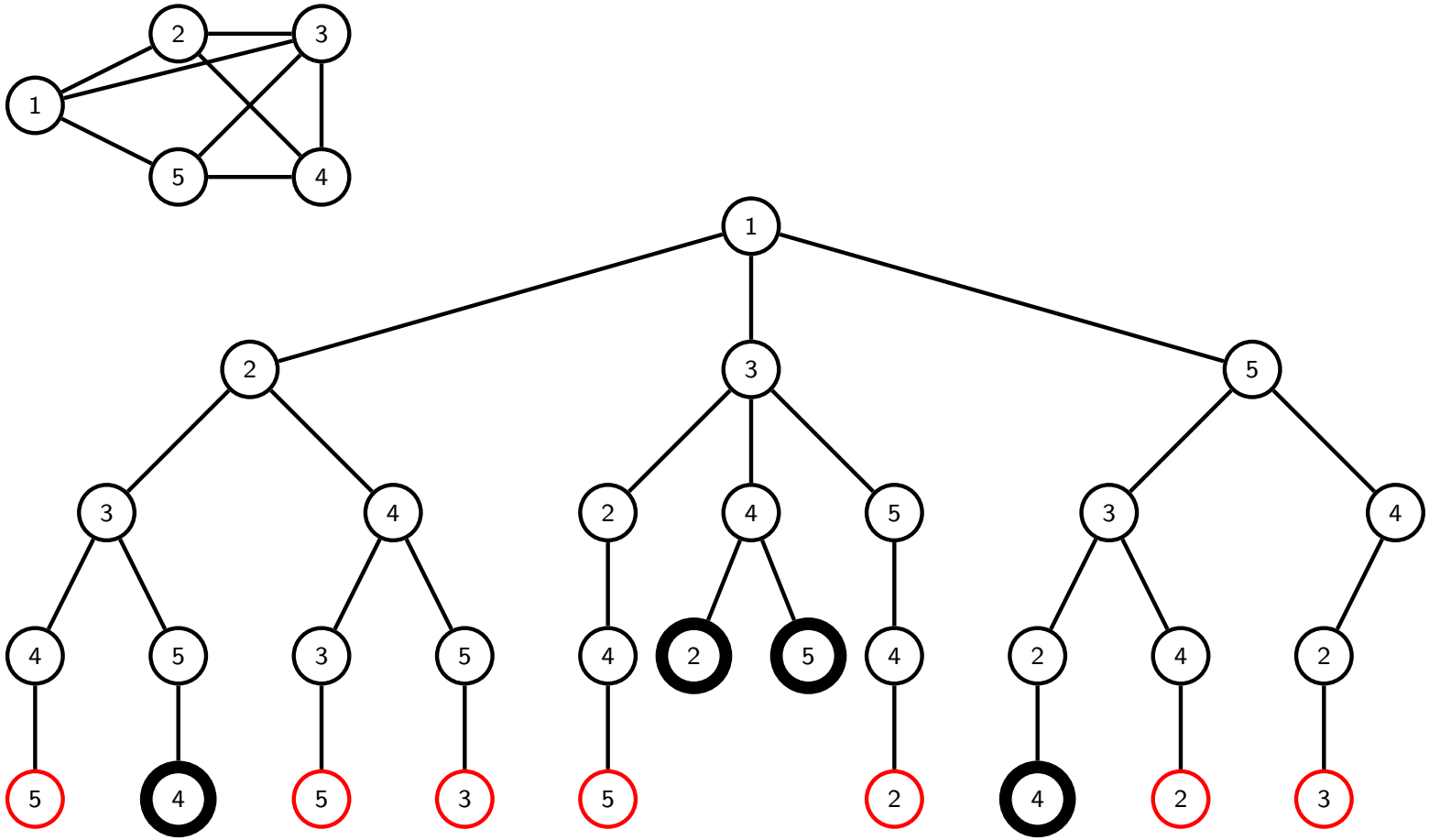
Esempio



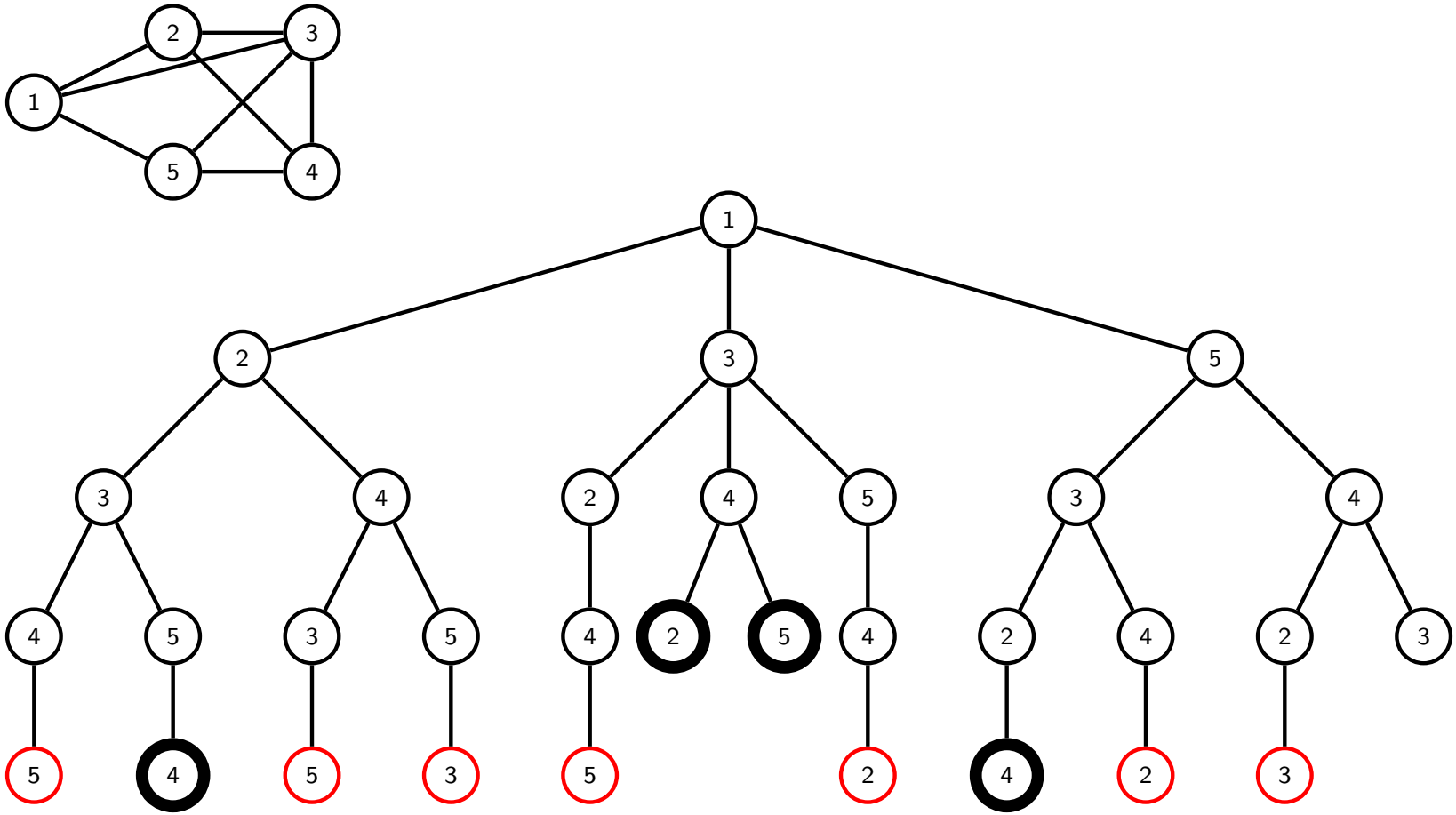
Esempio



Esempio



Esempio



Esempio