
TECNICO SUPERIORE
WEB DEVELOPER
FULL STACK
#14 - Backtracking
WEB ANALYTICS
TESTING FEATURES
DEVELOPMENT
CONTENT MANAGEMENT
RESPONSIVE DESIGN
ITS DIGITAL ACADEMY
MARIO VOLPATO
niuk
innovation and knowledge

Introduzione

Nei problemi che abbiamo affrontato (e negli algoritmi usati per risolverli) il nostro obiettivo era trovare una soluzione cosiddetta "accettabile", ossia una soluzione che soddisfasse un insieme di criteri definito.

- **Cammini minimi**: una sequenza di nodi tale che il numero/peso di archi attraversati sia minimo.
- **Ordinamento topologico**: una sequenza di nodi tale che gli archi siano sempre orientati da sinistra a destra

Altri problemi (che non abbiamo visto)

- **Zaino**: sottoinsieme di oggetti di peso inferiore alla capacità
- **Sottosequenza comune**: una stringa che è sottosequenza di entrambe le stringhe date in input
- **Resto**: trovare l’insieme minore di monete per il resto, dato un insieme di tagli di monete

Esplorazione

In alcuni problemi è richiesto o necessario esplorare l'intero spazio delle soluzioni ammissibili.

Elencare tutte le soluzioni ammissibili

Esempio: Elencare tutte le permutazioni di un insieme

Soluzione: Algoritmi di enumerazione

[1,2,3] [1,3,2] [2,1,3] [2,3,1] [3,2,1] [3,1,2]

Esplorazione

In alcuni problemi è richiesto o necessario esplorare l’intero spazio delle soluzioni ammissibili.

Ricerca

Trovare una soluzione ammissibile in uno spazio delle soluzioni molto grande

Esempio: Trovare una sequenza di mosse per il gioco del 15
Soluzione: Algoritmi di enumerazione, fermandosi alla prima soluzione trovata

Esplorazione

In alcuni problemi è richiesto o necessario esplorare l'intero spazio delle soluzioni ammissibili.

Conteggio

Contare tutte le soluzioni ammissibili

Esempio: contare il numero di modi in cui è possibile esprimere un valore n come somma di k numeri primi.
Soluzione: Se non è possibile contare in modo analitico, bisogna enumerare tutte le soluzioni ammissibili e contarle.

Esplorazione

In alcuni problemi è richiesto o necessario esplorare l'intero spazio delle soluzioni ammissibili.

Ottimizzazione

Trovare una delle soluzioni ammissibili migliori (ottimizzazione) rispetto ad un certo criterio di valutazione

Esempio: trovare il cammino di peso <u>massimo</u> da *s* a *d* in un grafo pesato
Soluzione: Enumerare tutti i cammini da *s* a *d*,restituire quello di peso massimo

Ma...

Costruire tutte le soluzioni è costoso, ma purtroppo a volte è l’unica strada percorribile.

A nostro vantaggio, però, a volte lo spazio delle soluzioni non deve essere analizzato interamente.

Backtracking

"Prova a fare qualcosa; se non va bene, disfalo e prova qualcos’altro: ritenta, e sarai più fortunato"

Un metodo sistematico per esplorare uno spazio di ricerca,utilizzando la ricorsione per memorizzare le scelte fatte finora.

Una tecnica algoritmica che, come altre, deve essere personalizzata per ogni applicazione individuale.

Organizzazione generale

Una soluzione viene rappresentata come un vettore di scelte S[1. . . n]

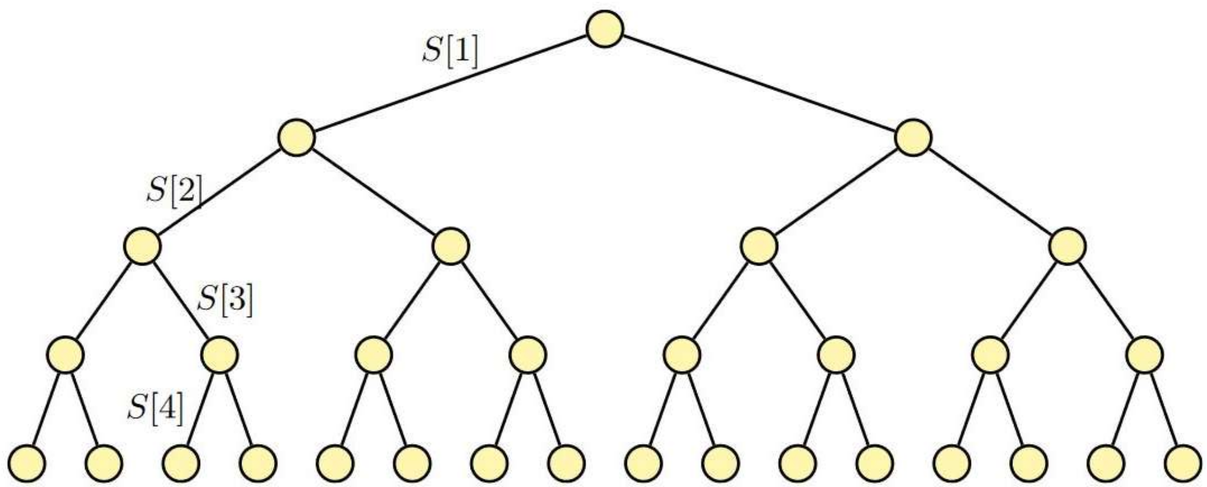
Il contenuto degli elementi S[i] è preso da un insieme di scelte C dipendente dal problema

Esempi

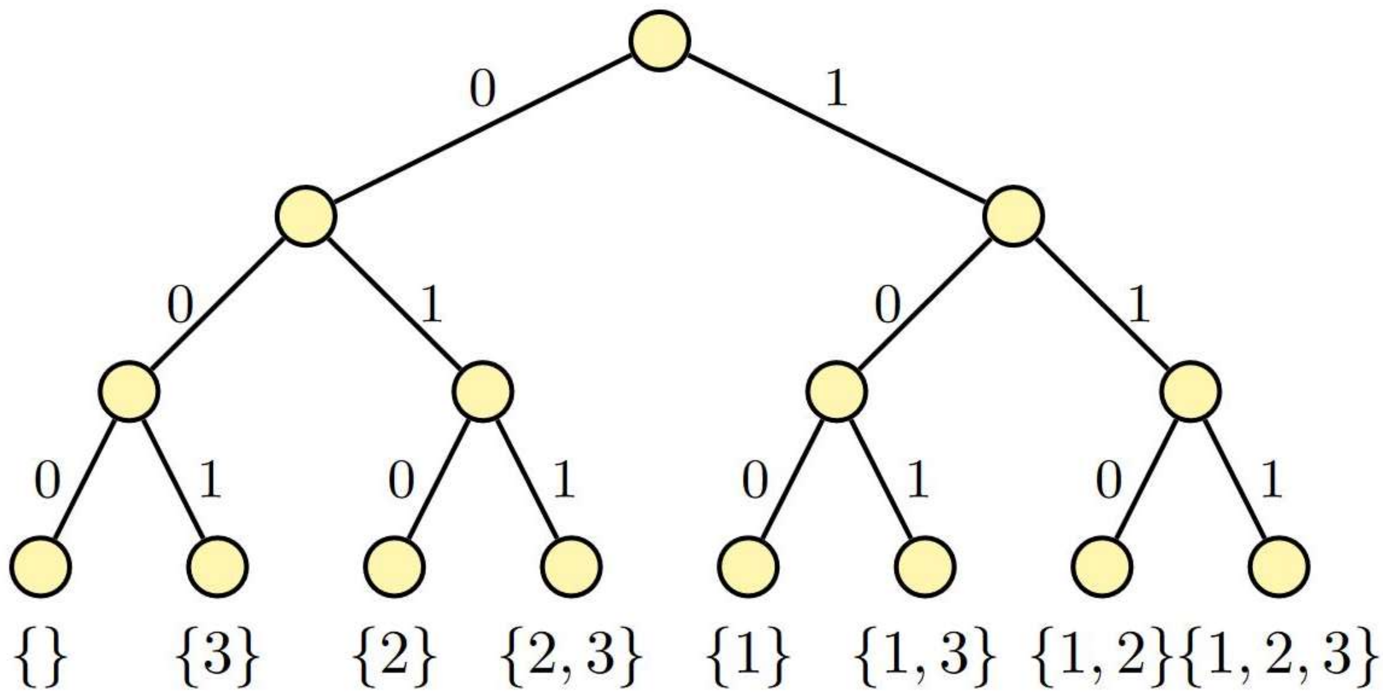
- Insieme di C elementi, possibili soluzioni *sottoinsiemi di C*
- Insieme di C elementi, possibili soluzioni *permutazioni di C*
- C mosse di gioco, possibili soluzioni *sequenze di mosse*
- C archi di un grafo, possibili soluzioni *percorsi sul grafo*

Albero delle decisioni

- Albero di decisione ≡ Spazio di ricerca
- Radice ≡ Soluzione parziale vuota
- Nodi interni dell’albero di decisione ≡ Soluzioni parziali
- Foglie in un albero di decisione ≡ Soluzioni ammissibili



Elencare i sottoinsiemi: albero delle decisioni



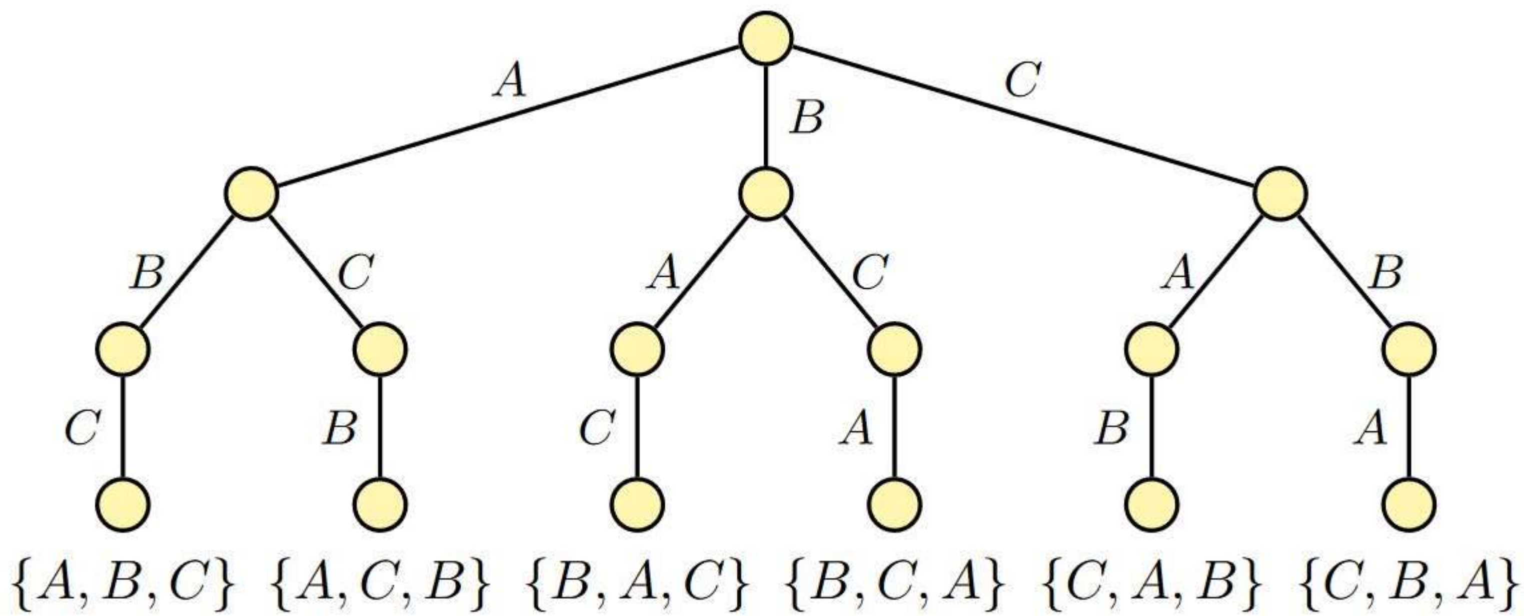
Elencare i sottoinsiemi: complessità

Come richiesto dal problema, tutto lo spazio possibile viene esplorato. La complessità è $\Theta(n \cdot 2^n)$

E’ grande? Sì.

Potevamo fare meglio? No, perché vanno generate tutte le soluzioni.

Elencare le permutazioni: albero delle decisioni



Elencare le permutazioni: complessità

Costo della stampa: $\Theta(n)$

Costo delle copie del vettore lungo un cammino: $O(n^2)$

Numero di foglie: $n!$

Complessità: $O(n^2 n!)$

Dobbiamo veramente provare tutto?

Partiamo da un esempio: vogliamo elencare tutti i sottoinsiemi *di k elementi* di un insieme S={1, ..., n}

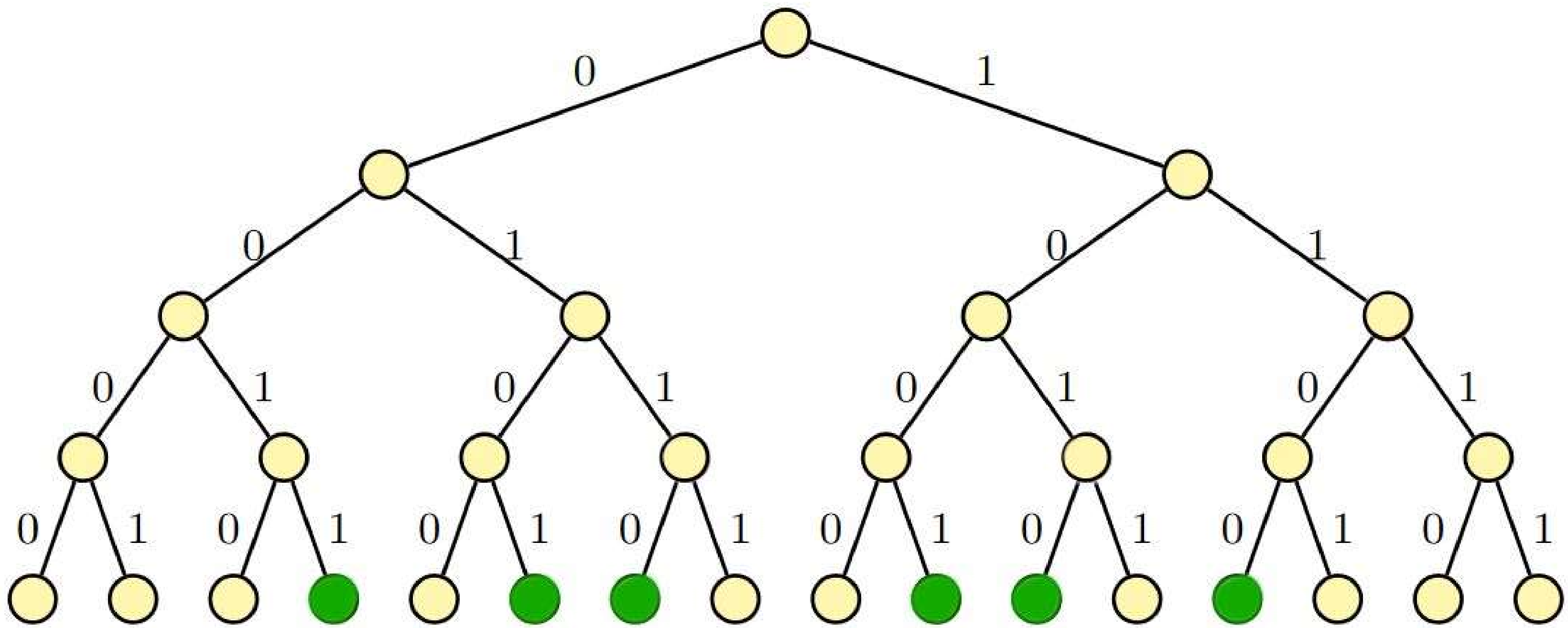
Ad esempio, i sottoinsiemi di due elementi dell'insieme S={A, B, C, D} sono {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}.

Lo spazio delle soluzioni è l'insieme di tutti i sottoinsiemi di S.

Dobbiamo quindi elaborare solo le soluzioni che hanno *k* elementi.

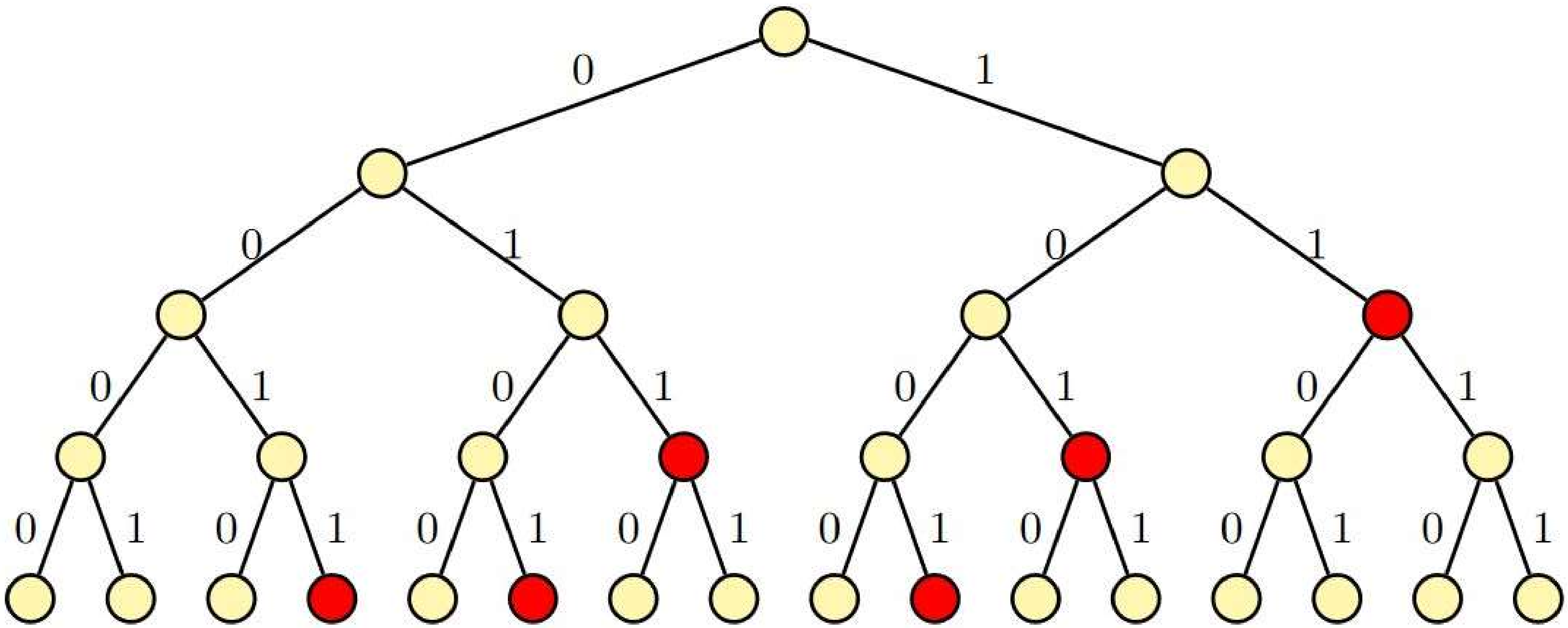
Albero delle decisioni per n=4 e k=2

I nodi verdi rappresentano soluzioni ammissibili.



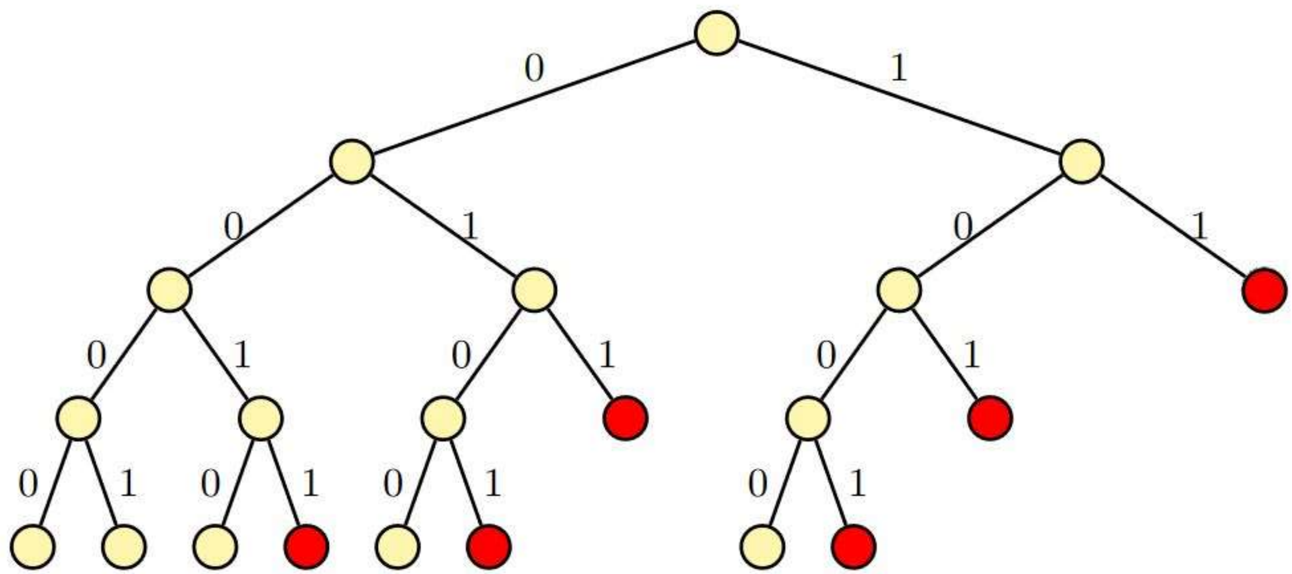
Si può fare di meglio?

I nodi rossi sono quelli in cui `missing` vale 0 per la prima volta.



Pruning, “potatura”

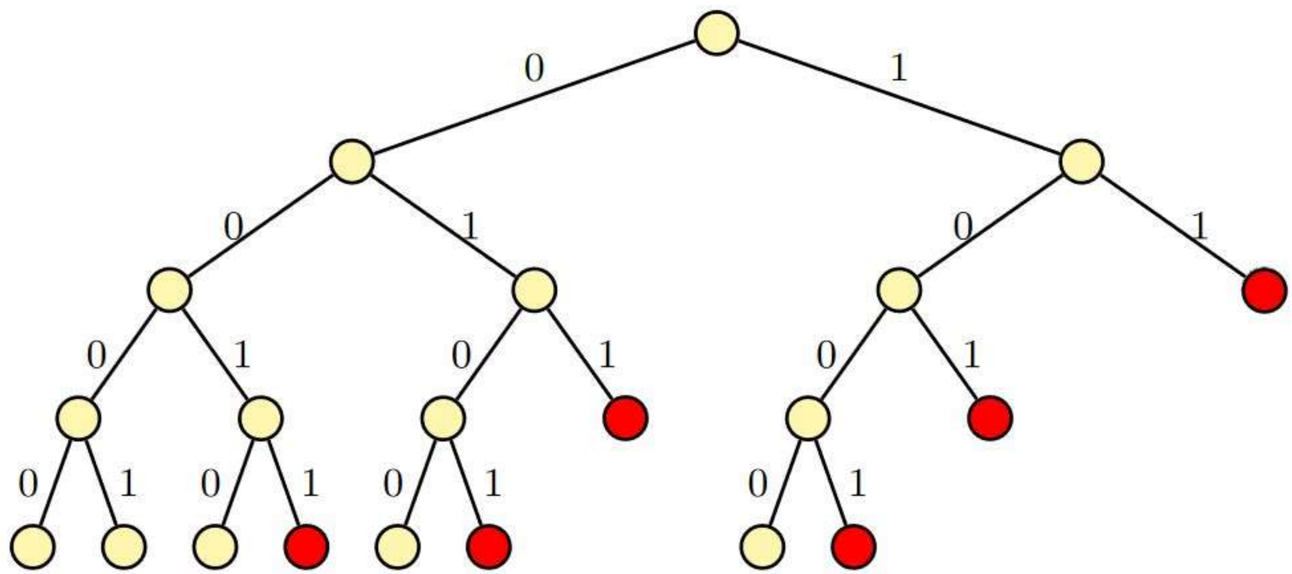
“Rami” dell’albero che sicuramente non portano a soluzioni ammissibili possono essere “potati”, e la valutazione viene fatta nelle soluzioni parziali radici del sottoalbero da potare.



Pruning, “potatura”

Abbiamo rimosso i casi in cui abbiamo già una soluzione ammissibile.

In quale altro caso sappiamo che è inutile proseguire?

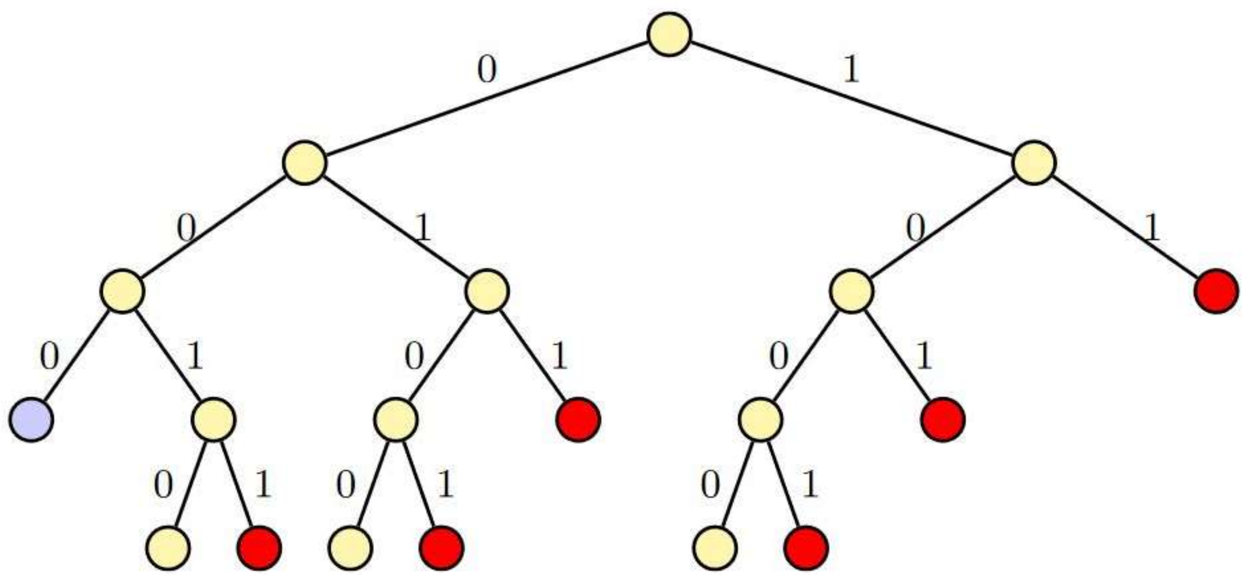


ITS DIGITAL ACADEMY MARIO VOLPATO

niuko innovation and knowledge

Pruning, “potatura”

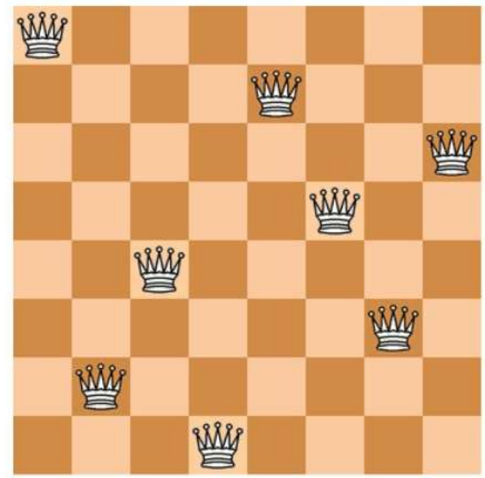
Il nodo in azzurro rappresenta un ramo che non può dare origine a soluzioni (non sono rimaste abbastanza scelte ancora da effettuare).



Un ultimo esempio: il problema delle 8 regine

Posizionare 8 regine in una scacchiera 8×8, in modo tale che nessuna regina ne “minacci” un’altra.

Partiamo da una soluzione “stupida” e proviamo a “raffinarla”.



Un ultimo esempio: il problema delle 8 regine

Come rappresentiamo la soluzione?

Matrice binaria: `S[i] = 1` se c'è una regina in posizione `i`

`S[i] = 0` altrimenti

Insieme delle scelte: `{true, false}`

Numero di soluzioni: $2^{64} \approx 1.84 \cdot 10^{19}$

Forse possiamo fare di meglio.

Un ultimo esempio: il problema delle 8 regine

Semplificare la rappresentazione.

Array di interi: `S[i] = k`, c'è una regina nella casella `k`

Insieme delle scelte: $\{1, \ldots, n^2\}$

Pruning: restituisce il sottoinsieme di mosse legali

Numero di soluzioni: $(n^2)^n = 64^8 = 2^{48} \approx 2.81 \cdot 10^{14}$

Già meglio, ma cosa cambia tra una soluzione come {1, 9, …} e una come {9, …, 1, …}?

ITS DIGITAL ACADEMY MARIO VOLPATO

niuko innovation and knowledge

Un ultimo esempio: il problema delle 8 regine

<u>Non mettere le regine in caselle precedenti.</u>

Array di interi: `S[i] = k`, c’è una regina nella casella `k`

Insieme delle scelte: $\{1, \ldots, n^2\}$

Pruning: restituisce le mosse legali e `S[i] > S[i-1]`

Numero di soluzioni: $(n^2)^n/n! = 2^{48}/40320 \approx 6.98 \cdot 10^9$

Meglio, ma una soluzione come {1, 9, …} è ancora accettabile.

Un ultimo esempio: il problema delle 8 regine

<u>Non mettere più regine sulla stessa riga o colonna.</u>

Array di interi: `S[i] = k`, c'è una regina nella colonna `k` della riga `i`

Insieme delle scelte: `{1, …, n}`

Pruning: elimina le diagonali

Numero di soluzioni: $n! = 8! = 40320 \approx 4.03 \cdot 10^4$

Soluzioni effettivamente visitate = 15720

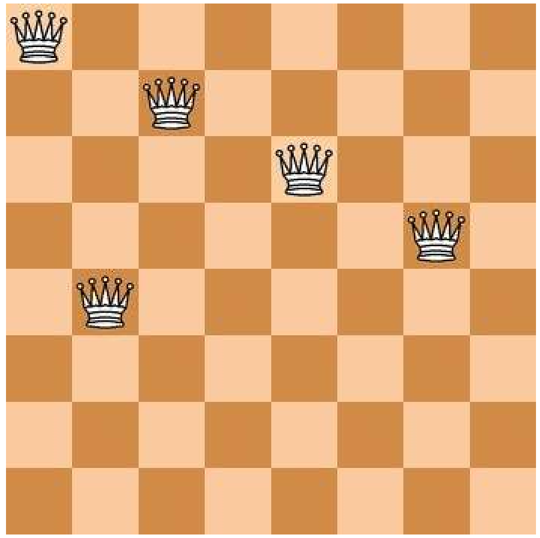
ITS DIGITAL ACADEMY MARIO VOLPATO

niuko innovation and knowledge

Un ultimo esempio: il problema delle 8 regine

```
queens(int n, int[] S, int i)
if i > n then
    print S
else
    for j = 1 to n do  % Prova a piazzare la regina nella colonna j
        boolean legal = true
        for k = 1 to i − 1 do          % Verifica le regine precedenti
            if S[k] == j or S[k] == j + i − k or S[k] == j − i + k then
                legal = false
        if legal then
            S[i] = j
            queens(n, S, i + 1)
```

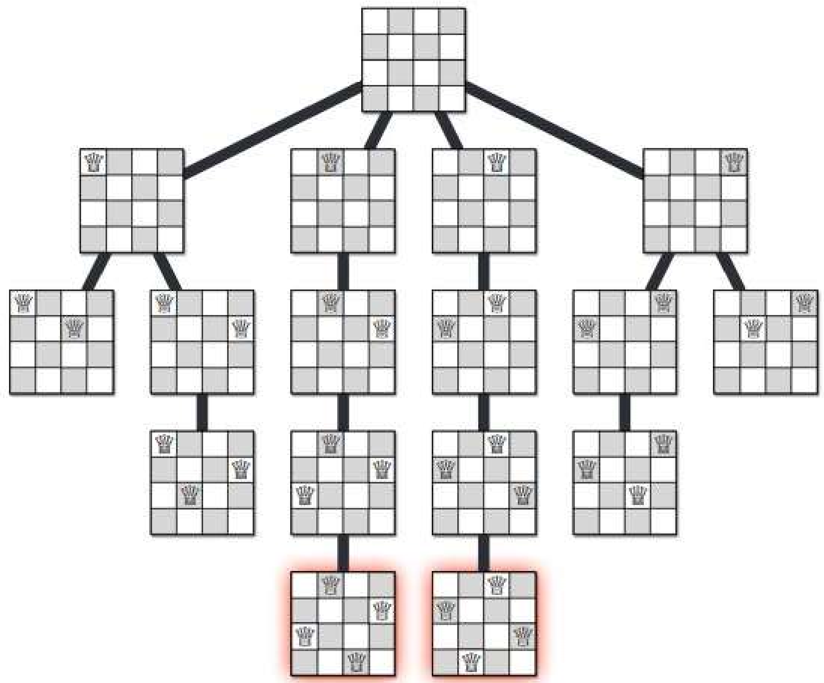
Un ultimo esempio: il problema delle 8 regine



https://upload.wikimedia.org/wikipedia/commons/1/1f/Eight-queens-animation.gif

Un ultimo esempio: il problema delle 8 regine

Proviamo ad analizzare una versione più semplice, con sole 4 regine in una scacchiera 4×4



Esercizi

- Stampare tutti i numeri binari di lunghezza N presa in input.
  - Quali sono le scelte possibili?
  - Quando ci dobbiamo fermare?
- Il mini-sudoku è una versione semplificata del noto gioco, dove i numeri vanno da 1 a 3 e la griglia è 3x3. Scrivere un algoritmo che usa backtrack per risolvere la griglia sottostante:

1		
	3	