

14 - Paradigma selettivo

Paradigma selettivo

Gli algoritmi del paradigma selettivo visitano lo spazio di ricerca con l'obiettivo di trovare un elemento ammissibile.

Ogni algoritmo fa riferimento all'interno dello spazio di ricerca, che viene esplorato con sistematicità secondo una modalità definita. Questo approccio metodico garantisce la copertura necessaria per l'individuazione della soluzione.

Appartengono a questo paradigma due tecniche principali:

- Enumerativa;
- Backtracking;

Esempio: ordinamento di un vettore

si consideri il problema dell'ordinamento di un vettore di interi V , di dimensioni n , secondo l'ordine non decrescente dei suoi elementi.

In questo contesto, lo spazio di ricerca è costituito dall'insieme delle permutazioni di V . La dimensione di tale spazio è determinata dal fattoriale della dimensione del vettore, ovvero n .

La funzione di ammissibilità verifica il rispetto del criterio di ordinamento: essa controlla che non esistano due indici h, k (compresi tra 2 ed n) tali che, se $h > k$, risulti $V(h) < V(k)$. La funzione di risposta è l'identità.

A titolo esemplificativo, dato un vettore con tre elementi $v(1), v(2), v(3)$, sono possibili 6 ordinamenti distinti all'interno dello spazio di ricerca.

Sia il vettore **$v(1), v(2), v(3)$**

Sono possibili **6** ordinamenti

$v(1) v(2) v(3)$	$v(1) v(3) v(2)$	$v(2) v(1) v(3)$
$v(2) v(3) v(1)$	$v(3) v(1) v(2)$	$v(3) v(2) v(1)$

Esiste una distinzione fondamentale tra due categorie di algoritmi:

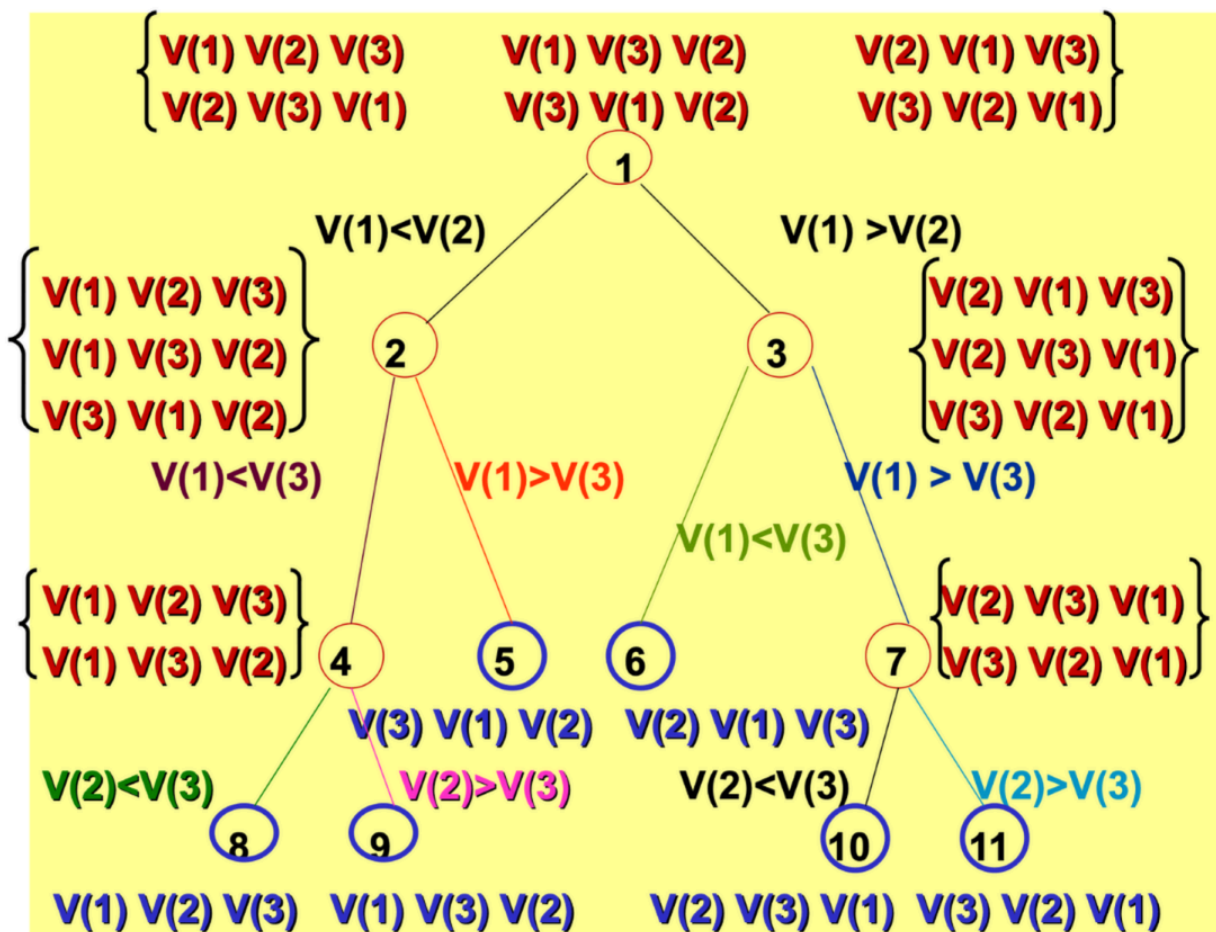
- **Algoritmo selettivo:** Effettua una visita dello spazio di ricerca esaminando le varie possibilità per individuare una permutazione che soddisfi la condizione di ammissibilità.
- **Algoritmo generativo:** Deriva la soluzione applicando un procedimento diretto sull'istanza del problema, senza necessità di visitare esplicitamente lo spazio di ricerca.

Nel contesto dell'esempio di ordinamento, la funzione di ammissibilità guida la selezione, indicando che la soluzione coincide con quella specifica permutazione che rispetta il criterio di ordinamento richiesto.

Rappresentazione dello spazio di ricerca

Il processo di esplorazione dello spazio di ricerca viene modellato mediante una struttura ad albero, la quale permette di visualizzare le scelte logiche effettuate dall'algoritmo:

- **Nodi:** contengono i confronti necessari per discriminare tra varie permutazioni e indirizzare la ricerca verso la soluzione corretta.
- **Foglie:** ogni foglia rappresenta un possibile ordinamento finale; di fatto l'albero partiziona lo spazio di ricerca in modo che ogni terminazione corrisponda a una specifica permutazione.
- **Profondità:** La profondità dell'albero è indicatore della complessità computazionale, in quanto rappresenta il numero di confronti necessari per determinare la soluzione nel caso peggiore.



Tecnica enumerativa

La tecnica enumerativa è basata sulla sistematica ispezione, elemento per elemento, dello spazio di ricerca associato ad una istanza di un problema. Questa metodologia garantisce la terminazione del processo, a condizione che lo spazio di ricerca sia finito.

Il criterio di arresto varia a seconda della natura del problema affrontato:

- **Problemi di ricerca:** Il processo termina quando si è individuato un elemento ammissibile o quando è esaurito lo spazio di ricerca.
- **Problemi di ottimizzazione:** È necessario confrontare tutti gli elementi ammissibili ai fini della selezione; di conseguenza, la terminazione è comunque data dall'esaurimento dello spazio di ricerca.

Per garantire la possibilità di una visita sistematica dello spazio di ricerca, si associa ad esso una relazione di ordinamento totale. Considerando lo spazio Z_i associato a un'istanza i , l'introduzione di tale ordinamento permette di definire tre metodi operativi essenziali:

- **Identificazione dell'inizio:** Un metodo per stabilire il primo elemento da considerare.
- **Avanzamento:** Un metodo per stabilire l'elemento successivo.
- **Controllo di completezza:** Un metodo per verificare se si sono esaminati tutti gli elementi.

Algoritmo enumerativo per problemi di ricerca

La procedura enumerativa per la risoluzione di problemi di ricerca segue uno schema iterativo volto a individuare il primo elemento che soddisfi i requisiti richiesti.

L'algoritmo opera secondo i seguenti passi:

- Si considera il primo elemento x appartenente allo spazio di ricerca.
- Si valuta la condizione di ammissibilità $a(x)$. Se $a(x)$ è vero, l'algoritmo termina immediatamente fornendo $o(x)$ come risultato.
- Se tutti gli elementi dello spazio di ricerca sono stati esaminati senza trovare un elemento ammissibile, l'algoritmo termina restituendo \emptyset per indicare l'assenza di soluzioni.
- Se l'elemento corrente non è ammissibile e lo spazio di ricerca non è ancora esaurito, si considera come nuovo x l'elemento successivo dello spazio di ricerca e si ripete la procedura a partire dalla verifica di ammissibilità.

Esempio: il gioco dell'otto

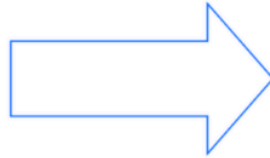
Il problema viene modellato definendo due configurazioni distinte:

- **INIZIO:** Una disposizione di partenza dei tasselli numerati sulla griglia.
- **FINALE:** la configurazione obbiettivo che si intende raggiungere.

Nello specifico, l'obiettivo è disporre i tasselli numerati in modo ordinato lungo i bordi della griglia.

LO STATO INIZIALE

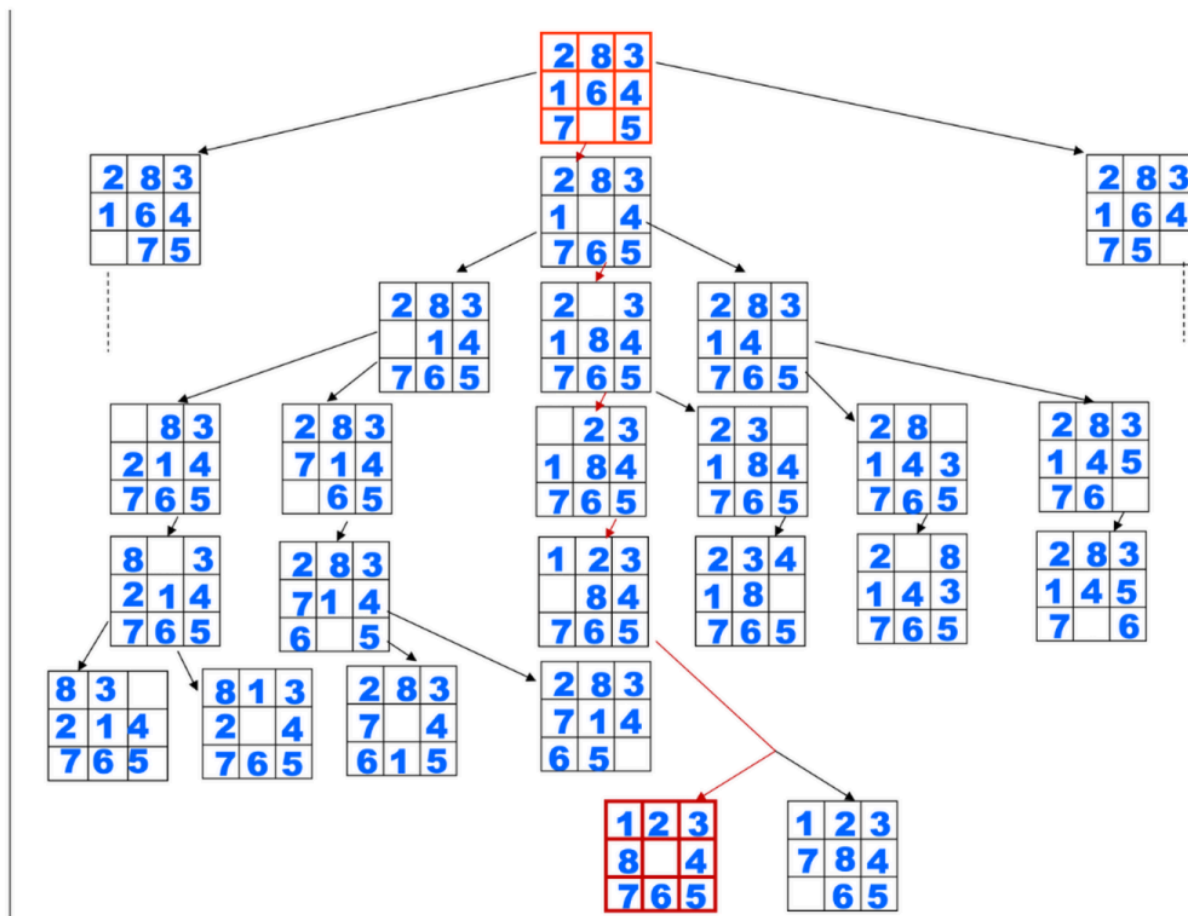
2	8	3
1	6	4
7		5

**LO STATO FINALE**

1	2	3
8		4
7	6	5

La soluzione può essere ricercata esaminando tutte le possibili disposizioni che vengono generate attraverso l'esecuzione di mosse successive.

Dal punto di vista formale, questo procedimento di ricerca è rappresentabile mediante un albero, dove i nodi corrispondono alle configurazioni possibili generate mosso dopo mosso a partire dallo stato iniziale.



Il procedimento si sviluppa attraverso una struttura ad albero:

- **Radice dell'albero:** Rappresenta la configurazione iniziale della griglia.
- **Generazione dei successori:** A partire dal nodo radice, vengono generati i nodi figli. Ogni figlio corrisponde a una nuova configurazione ottenuta effettuando una singola mossa valida (spostamento di un tassello nella casella vuota).

- **Ramificazione:** Il processo prosegue iterativamente. Da ogni nuova configurazione generata, si esplorano le ulteriori mosse possibili, espandendo l'albero in profondità e in ampiezza. Ad esempio, da una configurazione in cui la casella vuota permette due movimenti diversi, si genereranno due rami distinti nell'albero di ricerca.

Questa visualizzazione dimostra come la tecnica enumerativa esplori sistematicamente le varianti del gioco: ogni nodo visitato viene confrontato con la configurazione finale desiderata (tasselli ordinati) per verificare se rappresenta la soluzione.

L'albero mappa quindi l'intera sequenza di decisioni e stati che l'algoritmo deve attraversare per passare dallo stato iniziale a quello finale o per determinare che un percorso non porta alla soluzione.

Algoritmo Enumerativo per Problemi di Ottimizzazione

A differenza dei problemi di ricerca, dove l'arresto può avvenire alla prima occorrenza valida, nei problemi di ottimizzazione è necessario esplorare lo spazio per identificare la soluzione migliore secondo un criterio prestabilito. La procedura si articola nei seguenti passi:

1. Si considera il primo elemento x dello spazio di ricerca.
2. Se l'elemento corrente è ammissibile ($a(x) = true$), si procede all'aggiornamento della soluzione ottima corrente in due casi specifici:
 - Se x è la **prima soluzione** ammissibile individuata in assoluto.
 - Se x è migliore della soluzione ottima corrente precedentemente memorizzata. In entrambi i casi, si pone la soluzione ottima corrente uguale a x .
3. Se tutti gli elementi dello spazio di ricerca sono stati considerati, l'algoritmo termina fornendo il risultato finale:
 - Restituisce un indicatore di fallimento (ad esempio I) se non sono state trovate soluzioni ammissibili.
 - Restituisce $o(x)$, ovvero la soluzione ottima corrente, se è stata individuata.
4. Qualora lo spazio di ricerca non sia esaurito, si considera come nuovo x l'elemento successivo e si ripete il procedimento a partire dal passo di valutazione.

Esempio: Il gioco dell'8 problema di ottimizzazione

Mentre in un problema di ricerca puro l'obiettivo si limita a individuare una qualsiasi configurazione che soddisfi i requisiti finali, è possibile riformulare il gioco dell'otto come un problema di ottimizzazione.

In questo scenario l'obiettivo non è semplicemente trovare una soluzione, bensì individuare una soluzione ottima secondo un criterio specifico, definito come "il prima possibile".

Operativamente, questo approccio comporta due conseguenze fondamentali:

- **Criterio di valutazione:** si valuta la qualità della soluzione in base al numero di mosse necessarie per raggiungerla. La soluzione ottima corrisponde al percorso che richiede il

numero minimo di mosse.

- **Esplorazione esaustiva:** Per garantire l'ottimalità, non ci si può fermare alla prima soluzione trovata. È necessario esplorare l'intero albero di ricerca per confrontare i vari cammini e identificare con certezza quello più breve.

Tecnic a di Backtracking

Nella tecnica di backtracking, la modalità di visita dello spazio di ricerca differisce dall'enumerazione semplice. La generazione degli elementi avviene secondo un processo suddiviso in **stadi**.

Il funzionamento si basa su tre principi strutturali:

1. **Composizione:** Ogni elemento dello spazio di ricerca è costituito da diverse componenti; ad ogni stadio del processo viene scelta e aggiunta una singola componente.
2. **Soluzione Parziale:** Dato che gli elementi sono strutturati in componenti, dopo aver completato i stadi (con $i < n$), l'algoritmo ha costruito una **soluzione parziale**.
3. **Riconoscimento del Fallimento:** La caratteristica distintiva della tecnica è la capacità di analisi durante la costruzione. In molti problemi è possibile determinare anticipatamente che la soluzione parziale generata è fallimentare, ovvero che non potrà mai essere completata in una soluzione ammissibile.

Una volta riconosciuto che una soluzione parziale non è valida, l'algoritmo **interrompe** il processo di costruzione su quel ramo specifico (senza generare i successivi stadi inutili) e tenta altre vie alternative, effettuando appunto il "ritorno all'indietro".

Esempio: problema dello string matching

Il problema consiste nel trovare la prima occorrenza di una sequenza pattern P all'interno di un testo T .

Un approccio tenterebbe di confrontare le stringhe partendo da ogni posizione possibile. Questo ciclo continua fino al riconoscimento completo del pattern o all'esaurimento del testo. Sebbene efficace, questo metodo può risultare inefficiente perchè, ad ogni fallimento, scarta tutte le informazioni acquisite dai confronti parziali precedenti, ripartendo quasi da zero. Il meccanismo di **backtracking** si attiva quando il riconoscimento fallisce: l'algoritmo torna indietro e riprende l'analisi dalla posizione che segue l'inizio del tentativo precedente. Tuttavia, analizzando i confronti già effettuati, è possibile ottimizzare il processo (algoritmo di Knuth-Morris-Pratt).

Le due stringhe sono formate da caratteri dello stesso alfabeto e vale il vincolo dimensionale $m \leq n$.

Consideriamo le seguenti sequenze:

- $T = 110111011001$

- $P = 110110$

L'algoritmo inizia confrontando i caratteri uno ad uno partendo dall'inizio.

I primi 5 caratteri di P coincidono con i primi 5 caratteri di T .

Il confronto fallisce al sesto carattere.

In questo momento gli indici sono:

- $i = 6$ (indice sul testo).
- $j = 6$ (indice sul pattern)

In un algoritmo banale, al fallimento si effettuerebbe un backtrack completo: si riporterebbe l'indice del testo indietro a $i = 2$ e quello del pattern a $j = 1$. Questo equivarrebbe a traslare il pattern di una sola posizione a destra.

L'approccio ottimizzato sfrutta la sottosequenza di P che è stata già riconosciuta prima dell'errore, ovvero: 11011.

Ora cerchiamo il punto di incastro più lungo possibile per fare un salto in avanti sicuro.

1. Proviamo a salvare 4 caratteri:

- La coda di ciò che ho letto è 1011.
- La testa del mio pattern è 1101.
- Sono uguali? NO.
- Se sposto il pattern in avanti di poco, i pezzi non combaciano. Non posso fermarmi qui.

2. Proviamo a salvare 3 caratteri:

- La coda è 011
- La testa è 110.
- Sono uguali? NO.
- Nemmeno qui c'è incastro. Devo scorrere ancora.

3. Proviamo a salvare 2 caratteri:

- La coda è 11.
- La testa è 11.
- Sono uguali? SÌ!
- Ho trovato un aggancio! I due 11 che ho appena letto nel testo possono fungere da "inizio" per il mio pattern.
Poiché i primi due caratteri del pattern (11) sono uguali agli ultimi due caratteri letti correttamente nel testo, non è necessario rileggere quei caratteri.

È conveniente ripartire mantenendo l'indice del testo fermo a $i = 6$ (valore al momento del fallimento) e spostando l'indice del pattern a $j = 3$. In questo modo:

1. Non si effettua backtrack su i (si evita di rileggere il testo).
2. Si effettua backtrack solo su j , allineando il prefisso noto con il suffisso appena letto.

La situazione riprende con il seguente allineamento:

$T = 1101 \mathbf{110110} 01$

$P = \mathbf{110110}$

Con $i = 6$ e $j = 3$, il confronto prosegue.

Se anche questo allineamento fallisce, si ripete la logica.

Il tentativo successivo, con i parametri aggiornati (in particolare con un ulteriore shift che porta a $j = 2$ se il precedente fallisce), avrà successo.

Tecnica di backtracking: albero di ricerca

Sia P un problema e sia dato un metodo per associare ad ogni istanza i di P uno spazio di ricerca Z_i .

Affinchè il backtracking sia applicabile, deve essere definito un modo per strutturare ogni elemento di Z_i in un numero finito di componenti.

L'albero di ricerca associato all'istanza i è un albero costruito secondo le seguenti regole:

- **Radice (livello 0):** Rappresenta una soluzione parziale fittizia (solitamente vuota).
- **Nodi interni (livello $j > 0$):** Ogni nodo situato a un livello j rappresenta una **soluzione parziale** S in cui sono state scelte le prime j componenti. Tale nodo possiede tanti figli quanti sono i modi possibili di aggiungere alla soluzione parziale S la $(j + 1)$ esima componente.
- **Foglie:** Ogni foglia dell'albero rappresenta un elemento completo dello spazio di ricerca Z_i .

Per operare sull'albero di ricerca associato a un'istanza i tramite lo spazio Z , è fondamentale definire tre strumenti metodologici che guidano l'algoritmo:

- Un metodo per rappresentare formalmente ogni soluzione parziale corrispondente a un nodo dell'albero.
- Un metodo per stabilire se una soluzione parziale viola i vincoli imposti dal problema.
- Una funzione, detta **funzione di ammissibilità** c , che determina se proseguire l'esplorazione di un ramo o interromperlo.

La funzione di ammissibilità è definita come valori booleani:

$$c : \text{nodi} \rightarrow \{true, false\}$$

Il comportamento della funzione

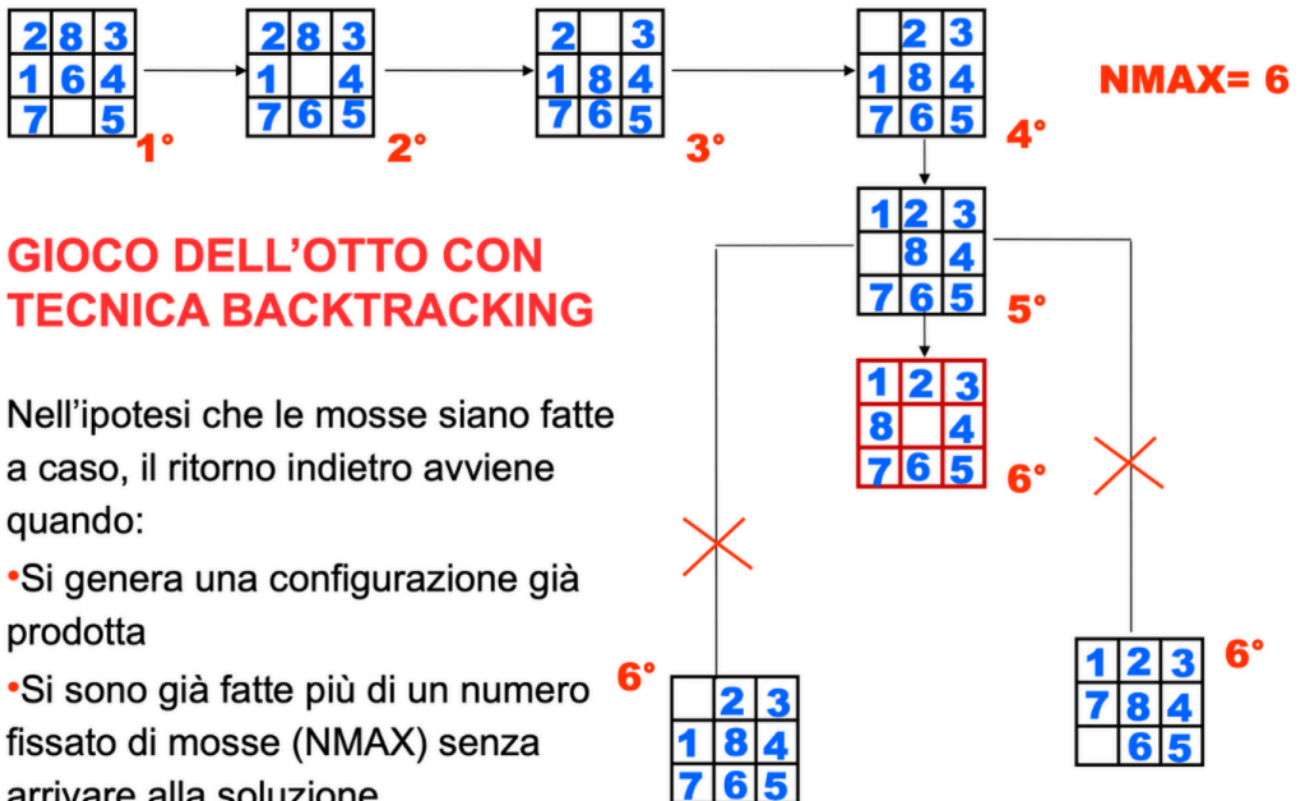
- **Per i nodi interni:** $c(n) = true$ se e solo se la soluzione parziale **viola i vincoli**. In questo caso, l'algoritmo effettua il backtracking (taglia il ramo) poiché è inutile proseguire la costruzione.
- **Per le foglie:** Se il nodo n è una foglia, $c(n) = true$ indica che l'elemento completo dello spazio di ricerca corrispondente è **non ammissibile**.

GIOCO DELL'OTTO CON TECNICA BACKTRACKING

L'applicazione della tecnica di backtracking al problema del "gioco dell'otto" permette di ottimizzare la ricerca introducendo criteri di arresto per i percorsi non promettenti.

In questo scenario, si ipotizza che la generazione delle mosse avvenga in modo casuale. Per gestire l'esplorazione dell'albero di ricerca, vengono definiti vincoli precisi che determinano quando l'algoritmo deve interrompere un ramo e tornare indietro.

Viene fissato un parametro di profondità massima, indicato come $NMAX$ (nell'esempio, $NMAX = 6$).



Il meccanismo di backtracking si attiva al verificarsi di due condizioni specifiche:

1. **Ridondanza (Cicli):** Si genera una configurazione della griglia che è già stata prodotta precedentemente nello stesso percorso. Questo controllo evita che l'algoritmo entri in cicli infiniti ritornando su stati già visitati.
2. **Superamento del Limite:** Sono state effettuate più di un numero fissato di mosse ($NMAX$) senza giungere alla configurazione finale (soluzione). Questo vincolo impedisce l'esplorazione di percorsi eccessivamente lunghi e inefficienti.

Nell'albero rappresentato, i nodi che raggiungono il livello 6 (o che violano la condizione di unicità) vengono tagliati, costringendo l'algoritmo a risalire verso i nodi padre per tentare alternative diverse.

Esempio: problema del partizionamento di un insieme

Il comportamento dell'algoritmo di backtracking per il problema del partizionamento viene descritto formalmente mediante la rappresentazione ad albero dello spazio di ricerca.

Dato un insieme $Y = \{y_1, y_2, \dots, y_n\}$ di interi positivi la cui somma totale è $2M$, il problema consiste nel verificare l'esistenza di un sottoinsieme di Y la cui somma degli elementi sia esattamente pari a M .

Consideriamo l'istanza specifica $Y = \{8, 5, 1, 4\}$. L'obiettivo è cercare un sottoinsieme la cui somma sia 9.

La soluzione viene espressa mediante un **vettore binario** $[x_1, \dots, x_4]$.

I valori del vettore rispettano la seguente logica:

- $x_i \in \{0, 1\}$
- $x_i = 1$ se e solo se l'elemento y_i appartiene al sottoinsieme scelto ($y_i \in X$).

L'algoritmo di backtracking invece di generare tutte le possibili quadruple simultaneamente, costruisce il vettore soluzione partendo dal vettore vuoto e aggiungendo una componente alla volta.

La struttura dell'albero di ricerca riflette questo processo:

- **Radice:** Rappresenta il vettore vuoto (nessuna scelta effettuata).
- **Livelli successivi:** A ogni livello si determina il valore di una variabile. Al livello 1 si diramano i nodi corrispondenti a $x_1 = 1$ (inclusione del primo elemento) e $x_1 = 0$ (esclusione).
- **Foglie:** Solo nelle foglie si ottiene la configurazione completa del vettore. Tra tutte le foglie generate, solo alcune rappresentano soluzioni ammissibili (somma uguale a M).

L'esplorazione dell'albero avviene tramite una **visita in pre-ordine** (depth-first).

ESEMPIO:

Analizzando l'albero per l'istanza $Y = \{8, 5, 1, 4\}$: Consideriamo il nodo in cui sono stati selezionati i primi due elementi, ovvero $x_1 = 1$ (scelgo 8) e $x_2 = 1$ (scelgo 5).

La somma parziale risulta:

$$8 + 5 = 13$$

Poiché 13 è già superiore al target richiesto (9), è certo che proseguendo su questo cammino non si troverà alcuna soluzione ammissibile. L'algoritmo, quindi, interrompe la discesa su questo ramo e torna indietro, evitando di esplorare le configurazioni successive inutili.

Albero di ricerca per il problema del partizionamento di $Y=\{8,5,1,4\}$. Sui nodi sono le configurazioni di X generate

