Strategie Risolutive e Programmazione Dinamica

Giorgio Audrito

Volterra, 16 novembre 2022

Un problema di riferimento: *Ladro*

Situazione

- Un ladro deve rapinare un castello di $m \times n$ stanze.

Un problema di riferimento: *Ladro*

Situazione

- Un ladro deve rapinare un castello di $m \times n$ stanze.
- L'ingresso è nella stanza $(1; 1)$ e l'uscita è nella stanza $(m; n)$.

Un problema di riferimento: *Ladro*

Situazione

- Un ladro deve rapinare un castello di $m \times n$ stanze.
- L'ingresso è nella stanza $(1; 1)$ e l'uscita è nella stanza $(m; n)$.
- In ogni stanza ci sono V_{ij} euro derubabili.

Un problema di riferimento: *Ladro*

Situazione

- Un ladro deve rapinare un castello di $m \times n$ stanze.
- L'ingresso è nella stanza $(1; 1)$ e l'uscita è nella stanza $(m; n)$.
- In ogni stanza ci sono V_{ij} euro derubabili.
- In ogni stanza ci si può muovere a est o a sud.

Un problema di riferimento: *Ladro*

Situazione

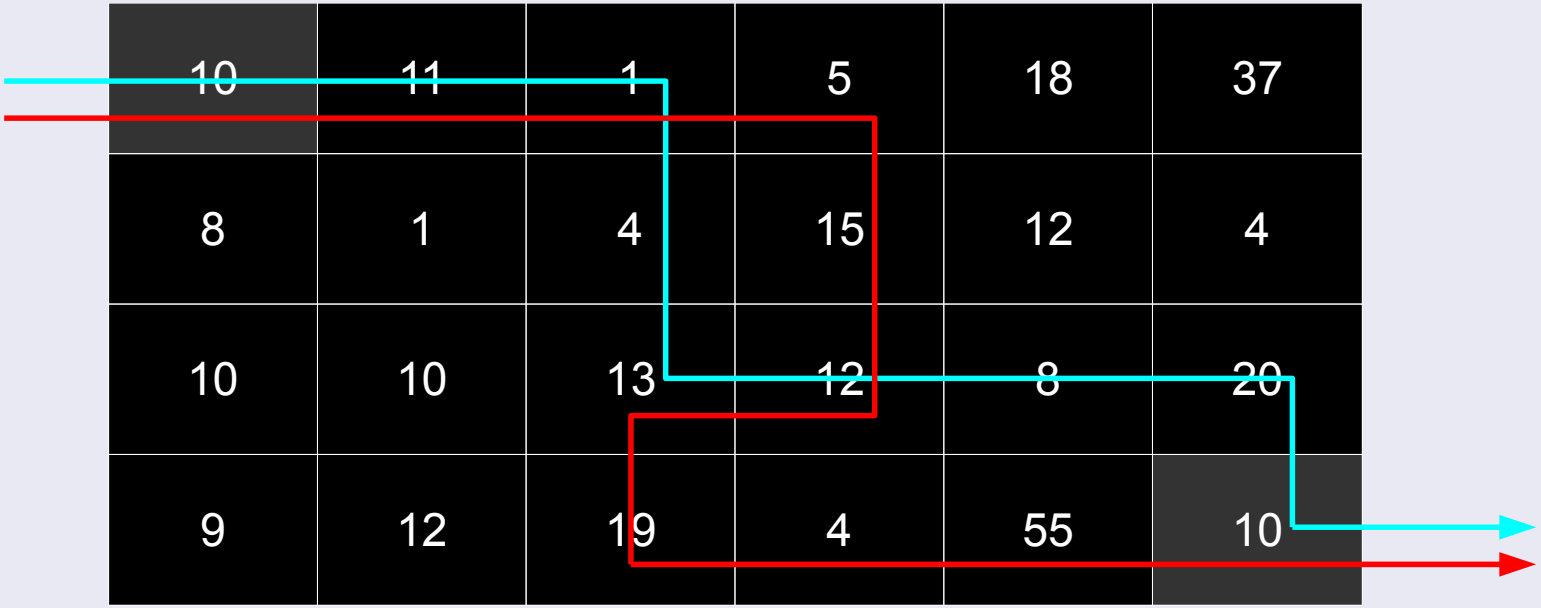
- Un ladro deve rapinare un castello di $m \times n$ stanze.
- L'ingresso è nella stanza $(1; 1)$ e l'uscita è nella stanza $(m; n)$.
- In ogni stanza ci sono V_{ij} euro derubabili.
- In ogni stanza ci si può muovere a est o a sud.

Obbiettivo

Massimizzare il valore rubato.

Un problema di riferimento: *Ladro*

Esempio



Nota

Il percorso blu è valido, quello rosso invece no.

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Caratteristiche

- Spesso è semplice da intuire.

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Caratteristiche

- Spesso è semplice da intuire.
- Quasi sempre è molto veloce in esecuzione.

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Caratteristiche

- Spesso è semplice da intuire.
- Quasi sempre è molto veloce in esecuzione.
- Purtroppo, di solito è sbagliata.

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Caratteristiche

- Spesso è semplice da intuire.
- Quasi sempre è molto veloce in esecuzione.
- Purtroppo, di solito è sbagliata.

Esempi in cui funziona

- Selezione delle attività

Tecnica Greedy

Cos'è?

Ogni volta che bisogna prendere una scelta, si sceglie la cosa al momento più promettente (tecnica *golosa*)

Caratteristiche

- Spesso è semplice da intuire.
- Quasi sempre è molto veloce in esecuzione.
- Purtroppo, di solito è sbagliata.

Esempi in cui funziona

- Selezione delle attività
- Scheduling di lavori...

Tecnica Greedy

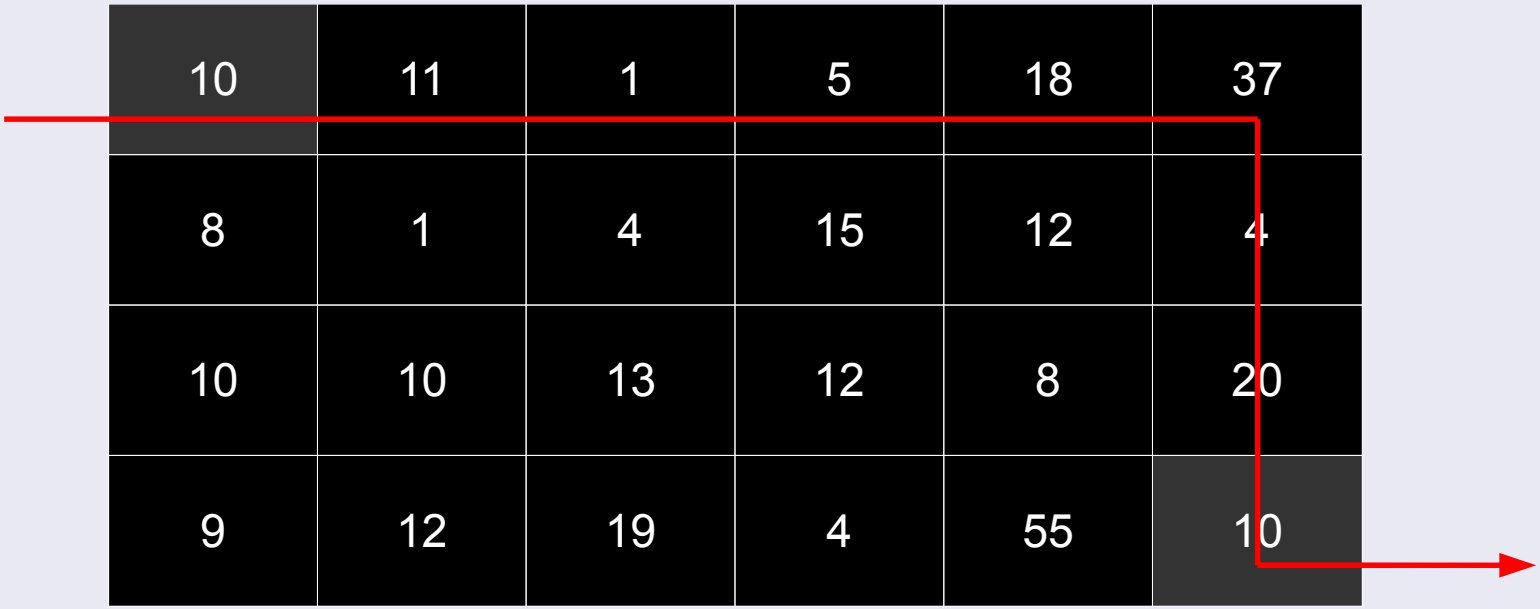
E per il problema *Ladro*?

10	11	1	5	18	37
8	1	4	15	12	4
10	10	13	12	8	20
9	12	19	4	55	10

Che soluzione troverà
l'algoritmo greedy?

Tecnica Greedy

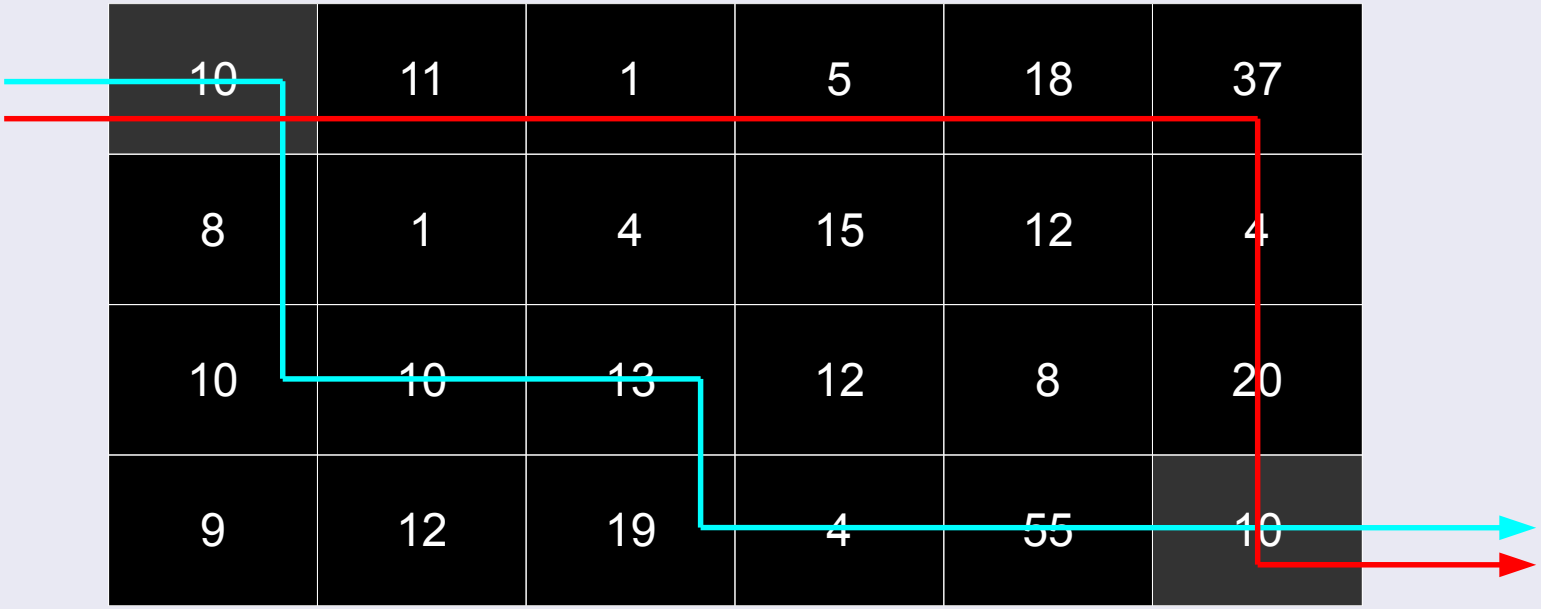
E per il problema *Ladro*?



Che soluzione troverà
l'algoritmo greedy?

Tecnica Greedy

E per il problema *Ladro*?



La soluzione trovata con la tecnica greedy (rossa) non è ottimale: si ha una somma di 116 anzichè 139 del percorso ottimo (blu).

Tecnica Greedy

E poteva andare anche peggio!!

1	1	1000	1000	1000	1000
2	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Tecnica Greedy

E poteva andare anche peggio!!

1	1	1000	1000	1000	1000
2	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

In questo caso con l'algoritmo greedy trova una somma di 10 anzichè 4005 del percorso ottimo.

Tecnica Greedy

E poteva andare anche peggio!!

1	1	1000	1000	1000	1000
2	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

In questo caso con l'algoritmo greedy trova una somma di 10 anzichè 4005 del percorso ottimo.

Proviamo a cambiare strategia.

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Caratteristiche

- Spesso è semplice da intuire.

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Caratteristiche

- Spesso è semplice da intuire.
- È sempre corretto, salvo errori di implementazione.

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Caratteristiche

- Spesso è semplice da intuire.
- È sempre corretto, salvo errori di implementazione.
- Purtroppo, di solito è estremamente lento in esecuzione, fino a diventare praticamente intrattabile.

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Caratteristiche

- Spesso è semplice da intuire.
- È sempre corretto, salvo errori di implementazione.
- Purtroppo, di solito è estremamente lento in esecuzione, fino a diventare praticamente intrattabile.

Esempi in cui funziona

- Il problema delle otto regine

Tecnica Esaustiva (*Backtracking*)

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Caratteristiche

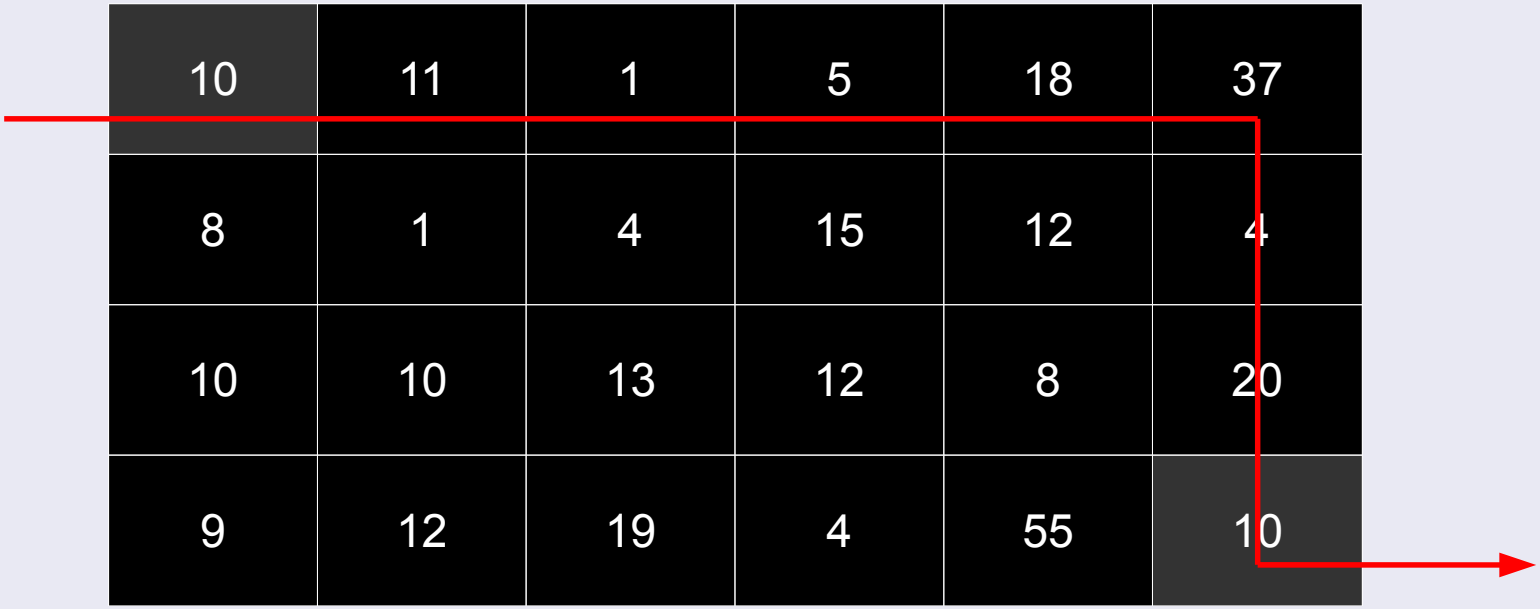
- Spesso è semplice da intuire.
- È sempre corretto, salvo errori di implementazione.
- Purtroppo, di solito è estremamente lento in esecuzione, fino a diventare praticamente intrattabile.

Esempi in cui funziona

- Il problema delle otto regine
- . . . qualunque problema con istanze abbastanza piccole è un valido esempio

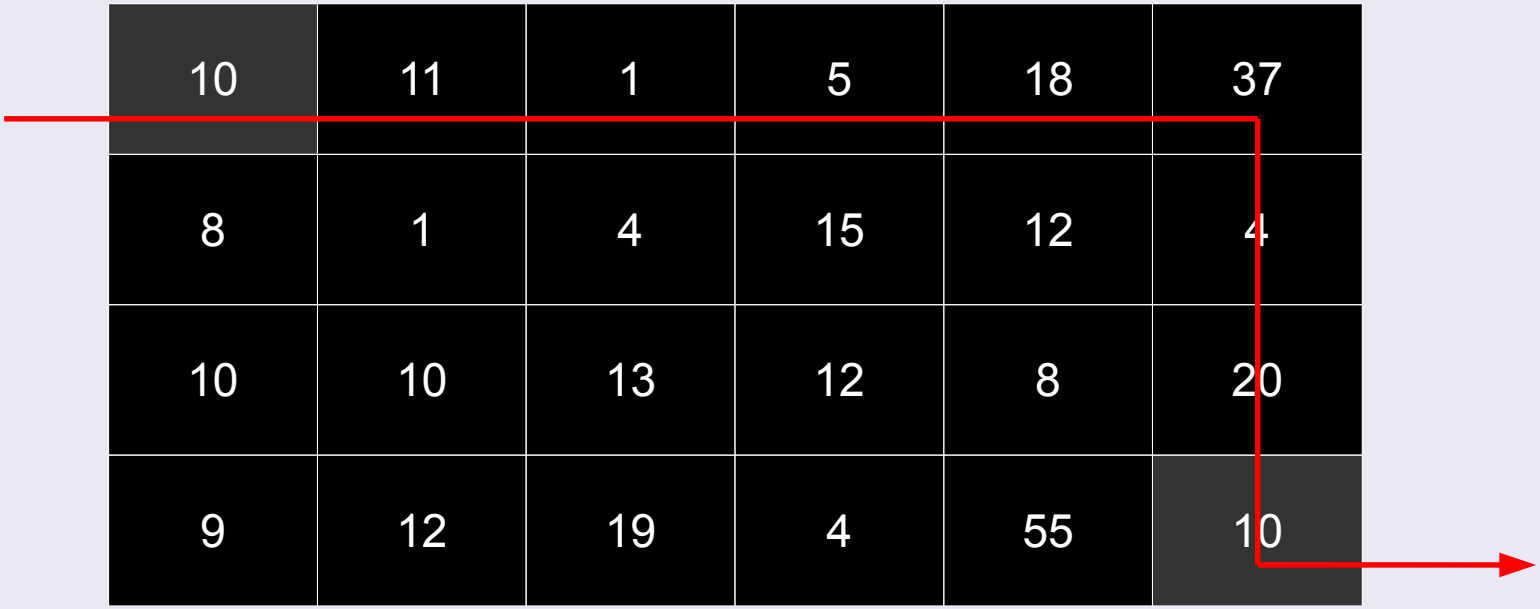
Tecnica Esaustiva (*Backtracking*)

E per il problema *Ladro*?



Tecnica Esaustiva (*Backtracking*)

E per il problema *Ladro*?



Dato che ogni percorso è formato da 6 passi a est e 4 passi a sud, l'algoritmo esaustivo impiegherà $\frac{10!}{6!4!} = 210$ passi (pari al numero di percorsi).

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)
- castello $30 \times 30 \longrightarrow$ circa 10^{17} passi

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)
- castello $30 \times 30 \longrightarrow$ circa 10^{17} passi
- castello $50 \times 50 \longrightarrow$ circa 10^{29} passi

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)
- castello $30 \times 30 \longrightarrow$ circa 10^{17} passi
- castello $50 \times 50 \longrightarrow$ circa 10^{29} passi
  circa 50 anni di tutta la potenza di calcolo globale

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)
- castello $30 \times 30 \longrightarrow$ circa 10^{17} passi
- castello $50 \times 50 \longrightarrow$ circa 10^{29} passi
  circa 50 anni di tutta la potenza di calcolo globale
- castello $70 \times 70 \longrightarrow$ circa 10^{41} passi
  $10\,000\times$ età dell'universo di tutta la potenza di calcolo globale

Tecnica Esaustiva (*Backtracking*)

In questa specifica istanza del problema l'algoritmo esaustivo è in grado di trovare la soluzione esatta in tempi ragionevoli.

Cosa succede se si ingrandisce l'input?

- castello $10 \times 10 \longrightarrow$ circa $200\,000$ passi
- castello $20 \times 20 \longrightarrow$ circa 10^{11} passi (cento miliardi)
- castello $30 \times 30 \longrightarrow$ circa 10^{17} passi
- castello $50 \times 50 \longrightarrow$ circa 10^{29} passi
  circa 50 anni di tutta la potenza di calcolo globale
- castello $70 \times 70 \longrightarrow$ circa 10^{41} passi
  $10\,000\times$ età dell'universo di tutta la potenza di calcolo globale

Proviamo di nuovo a cambiare strategia.

Divide et Impera

Cos'è?

Questa volta non procediamo per scelte successive.

Divide et Impera

Cos'è?

Questa volta non procediamo per scelte successive.
Invece, a ogni passo suddividiamo il problema in due problemi più piccoli, risolviamo separatamente i due problemi, e poi ricombiniamo le soluzioni.

Divide et Impera

Cos'è?

Questa volta non procediamo per scelte successive.
Invece, a ogni passo suddividiamo il problema in due problemi più piccoli, risolviamo separatamente i due problemi, e poi ricombiniamo le soluzioni.

Caratteristiche

- Quasi sempre è veloce in esecuzione.

Divide et Impera

Cos'è?

Questa volta non procediamo per scelte successive.
Invece, a ogni passo suddividiamo il problema in due problemi più piccoli, risolviamo separatamente i due problemi, e poi ricombiniamo le soluzioni.

Caratteristiche

- Quasi sempre è veloce in esecuzione.
- Purtroppo, di solito è difficile da intuire.

Divide et Impera

Cos'è?

Questa volta non procediamo per scelte successive.
Invece, a ogni passo suddividiamo il problema in due problemi più piccoli, risolviamo separatamente i due problemi, e poi ricombiniamo le soluzioni.

Caratteristiche

- Quasi sempre è veloce in esecuzione.
- Purtroppo, di solito è difficile da intuire.
- Purtroppo, di solito non è nemmeno applicabile.

Divide et Impera

Esempi in cui funziona

- Ricerca binaria

Divide et Impera

Esempi in cui funziona

- Ricerca binaria
- Ordinamento (algoritmo *Merge Sort*)

Divide et Impera

Esempi in cui funziona

- Ricerca binaria
- Ordinamento (algoritmo *Merge Sort*)
- In generale, in tutti i problemi che si possono suddividere in sotto-problemi indipendenti

Confronto Backtracking/Divide et Impera

Somiglianze

- Da un unico problema ci si riconduce a un certo numero di problemi più piccoli.

Confronto Backtracking/Divide et Impera

Somiglianze

- Da un unico problema ci si riconduce a un certo numero di problemi più piccoli.
- Si risolvono i problemi più piccoli con chiamate *ricorsive* dell'algoritmo.

Confronto Backtracking/Divide et Impera

Somiglianze

- Da un unico problema ci si riconduce a un certo numero di problemi più piccoli.
- Si risolvono i problemi più piccoli con chiamate *ricorsive* dell'algoritmo.
- Entrambi gli algoritmi effettuano tutte le sequenze di scomposizioni possibili.

Confronto Backtracking/Divide et Impera

Somiglianze

- Da un unico problema ci si riconduce a un certo numero di problemi più piccoli.
- Si risolvono i problemi più piccoli con chiamate *ricorsive* dell'algoritmo.
- Entrambi gli algoritmi effettuano tutte le sequenze di scomposizioni possibili.

Differenze

- I problemi a cui ci si riconduce sono separati e indipendenti nel Divide et Impera, mentre sono interconnessi e dipendenti nel Backtracking.

Confronto Backtracking/Divide et Impera

Somiglianze

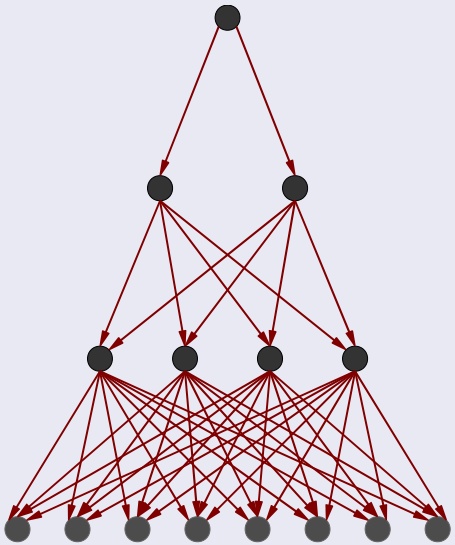
- Da un unico problema ci si riconduce a un certo numero di problemi più piccoli.
- Si risolvono i problemi più piccoli con chiamate *ricorsive* dell'algoritmo.
- Entrambi gli algoritmi effettuano tutte le sequenze di scomposizioni possibili.

Differenze

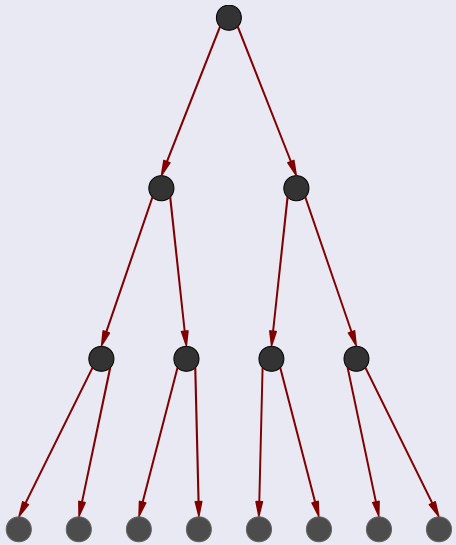
- I problemi a cui ci si riconduce sono separati e indipendenti nel Divide et Impera, mentre sono interconnessi e dipendenti nel Backtracking.
- Per questo motivo, nel Backtracking le possibili sequenze di scomposizioni sono moltissime mentre nel Divide et Impera sono poche, da cui l'enorme differenza nei tempi di esecuzione.

Confronto Backtracking/Divide et Impera

Cerchiamo di visualizzare la differenza con un grafico (*grafico delle chiamate ricorsive*).



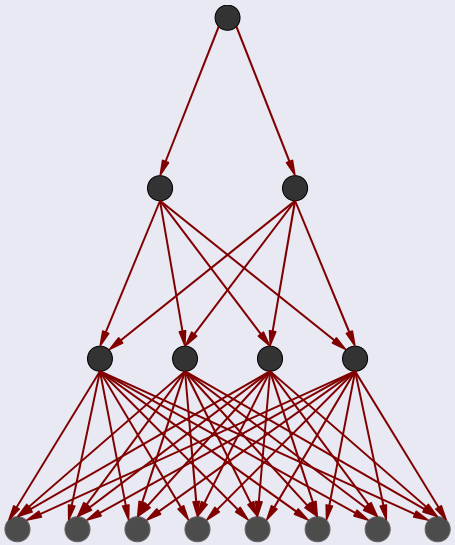
backtracking



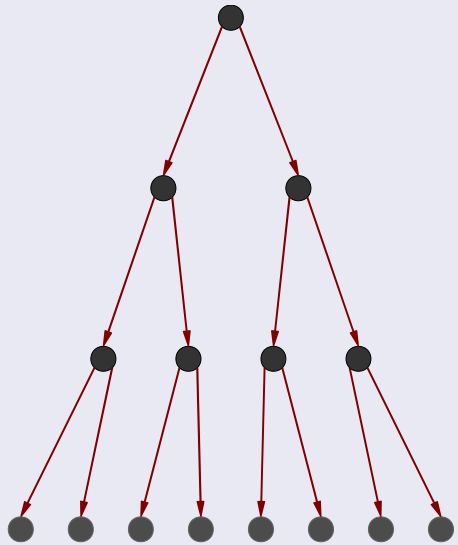
divide et impera

Confronto Backtracking/Divide et Impera

Cerchiamo di visualizzare la differenza con un grafico (*grafico delle chiamate ricorsive*).



backtracking



divide et impera

Ogni punto rappresenta un diverso sottoproblema, mentre le linee indicano a quali sottoproblemi più piccoli si riconduce un sottoproblema più grande.

Confronto Backtracking/Divide et Impera

C'è un modo per mettere insieme i vantaggi del Divide et Impera con la generalità del Backtracking?

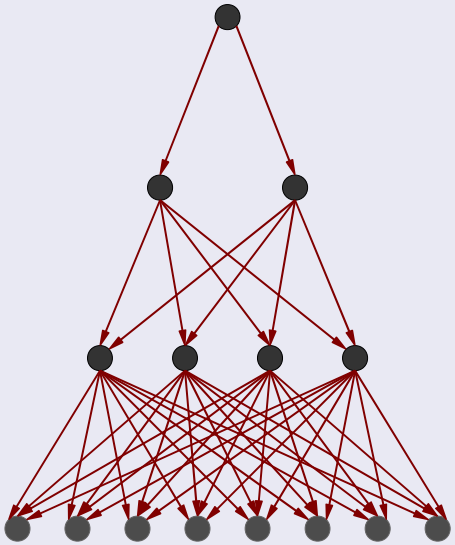
Confronto Backtracking/Divide et Impera

C'è un modo per mettere insieme i vantaggi del Divide et Impera con la generalità del Backtracking?

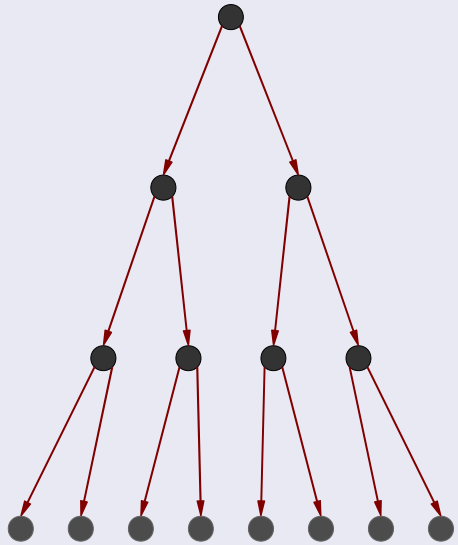
Fortunatamente, sì.

Confronto Backtracking/Divide et Impera

Osserviamo meglio il grafico delle chiamate ricorsive.



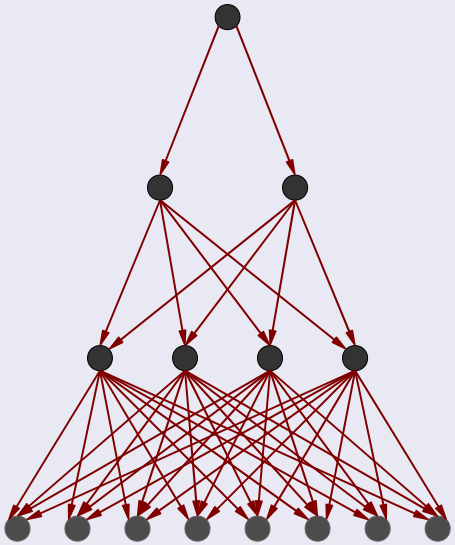
backtracking



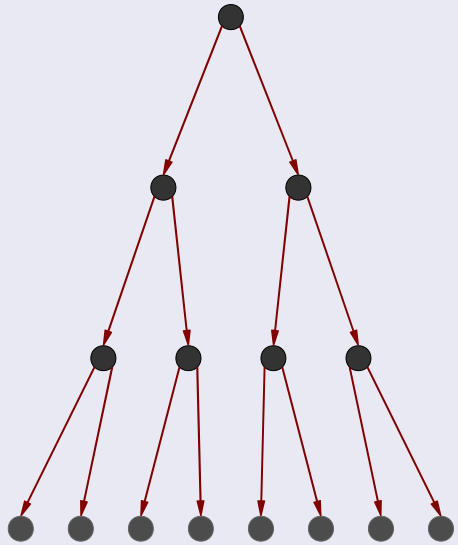
divide et impera

Confronto Backtracking/Divide et Impera

Osserviamo meglio il grafico delle chiamate ricorsive.



backtracking



divide et impera

Il Backtracking perde molto tempo ricalcolando le soluzioni agli stessi problemi. Cerchiamo di evitare di farlo così da ottenere una velocità simile a quella del Divide et Impera.

Ricorsione con Memorizzazione

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.

Ricorsione con Memorizzazione

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.
Nel frattempo però si memorizzano i risultati dei sottoproblemi in una tabella, utilizzandoli poi per evitare di ricalcolare gli stessi sottoproblemi più volte.

Ricorsione con Memorizzazione

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.
Nel frattempo però si memorizzano i risultati dei sottoproblemi in una tabella, utilizzandoli poi per evitare di ricalcolare gli stessi sottoproblemi più volte.

Caratteristiche

- È spesso applicabile.

Ricorsione con Memorizzazione

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.
Nel frattempo però si memorizzano i risultati dei sottoproblemi in una tabella, utilizzandoli poi per evitare di ricalcolare gli stessi sottoproblemi più volte.

Caratteristiche

- È spesso applicabile.
- Spesso è veloce in esecuzione.

Ricorsione con Memorizzazione

Cos'è?

Ogni volta che bisogna prendere una scelta, si provano tutte le possibilità una alla volta, finché non si trova una soluzione.
Nel frattempo però si memorizzano i risultati dei sottoproblemi in una tabella, utilizzandoli poi per evitare di ricalcolare gli stessi sottoproblemi più volte.

Caratteristiche

- È spesso applicabile.
- Spesso è veloce in esecuzione.
- È un po' più difficile da intuire.

Ricorsione con Memorizzazione

Individuamo una scaletta per cercare di ottenere un algoritmo ricorsivo con memorizzazione.

Ricorsione con Memorizzazione

Individuamo una scaletta per cercare di ottenere un algoritmo ricorsivo con memorizzazione.

Primo punto della Programmazione Dinamica

1. Identificare i sottoproblemi che è necessario considerare.

Ricorsione con Memorizzazione

Individuamo una scaletta per cercare di ottenere un algoritmo ricorsivo con memorizzazione.

Primo punto della Programmazione Dinamica

1. Identificare i sottoproblemi che è necessario considerare.

Nel problema *Ladro*

Trovare i percorsi di massimo guadagno da una qualunque casella alla fine.

Ricorsione con Memorizzazione

Individuamo una scaletta per cercare di ottenere un algoritmo ricorsivo con memorizzazione.

Secondo punto della Programmazione Dinamica

2. Trovare dei parametri che siano in grado di identificare univocamente i sottoproblemi.

Ricorsione con Memorizzazione

Individuamo una scaletta per cercare di ottenere un algoritmo ricorsivo con memorizzazione.

Secondo punto della Programmazione Dinamica

2. Trovare dei parametri che siano in grado di identificare univocamente i sottoproblemi.

Nel problema *Ladro*

Le due coordinate x, y della casella di partenza.

Ricorsione con Memorizzazione

Implementazione

- Allocare una tabella $M[..]$ indicizzata con tutti i parametri necessari (nel nostro caso x e y).

Ricorsione con Memorizzazione

Implementazione

- Allocare una tabella $M[..]$ indicizzata con tutti i parametri necessari (nel nostro caso x e y).
- Inizializzarla con un valore "impossibile" che consenta di identificare quali caselle non sono ancora state calcolate (solitamente -1).

Ricorsione con Memorizzazione

Implementazione

- Allocare una tabella $M[..]$ indicizzata con tutti i parametri necessari (nel nostro caso x e y).
- Inizializzarla con un valore "impossibile" che consenta di identificare quali caselle non sono ancora state calcolate (solitamente -1).
- Scrivere una funzione ricorsiva analogamente al Backtracking (nel nostro caso `migliorPercorso(x,y)`).

Ricorsione con Memorizzazione

Implementazione

- Allocare una tabella $M[..]$ indicizzata con tutti i parametri necessari (nel nostro caso x e y).
- Inizializzarla con un valore "impossibile" che consenta di identificare quali caselle non sono ancora state calcolate (solitamente -1).
- Scrivere una funzione ricorsiva analogamente al Backtracking (nel nostro caso `migliorPercorso(x,y)`).
- Inserire all'inizio della funzione un test che verifichi se il risultato è già stato calcolato:
  `if (M[x][y] != -1) return M[x][y];`

Ricorsione con Memorizzazione

Implementazione

- Allocare una tabella $M[..]$ indicizzata con tutti i parametri necessari (nel nostro caso x e y).
- Inizializzarla con un valore "impossibile" che consenta di identificare quali caselle non sono ancora state calcolate (solitamente -1).
- Scrivere una funzione ricorsiva analogamente al Backtracking (nel nostro caso `migliorPercorso(x,y)`).
- Inserire all'inizio della funzione un test che verifichi se il risultato è già stato calcolato:
  `if (M[x][y] != -1) return M[x][y];`
- Sostituire la riga di ritorno `return r` aggiungendoci la memorizzazione `return (M[x][y] = r)`.

Dinamica

La strategia appena vista viene chiamata *Ricorsione con Memorizzazione*.
La vera e propria Programmazione Dinamica richiede ancora un terzo punto:

Dinamica

La strategia appena vista viene chiamata *Ricorsione con Memorizzazione*.
La vera e propria Programmazione Dinamica richiede ancora un terzo punto:

Terzo punto della Programmazione Dinamica

3. Trovare un ordine in cui riempire la tabella $M[..]$, di modo che per il calcolo di un qualsiasi elemento siano sufficenti le caselle della tabella riempite prima di esso.

Dinamica

La strategia appena vista viene chiamata *Ricorsione con Memorizzazione*.
La vera e propria Programmazione Dinamica richiede ancora un terzo punto:

Terzo punto della Programmazione Dinamica

3. Trovare un ordine in cui riempire la tabella $M[..]$, di modo che per il calcolo di un qualsiasi elemento siano sufficenti le caselle della tabella riempite prima di esso.

Alcune caselle però dovranno essere calcolate per prime, senza usare altre caselle (casi base, come nell'induzione).

Dinamica

La strategia appena vista viene chiamata *Ricorsione con Memorizzazione*.
La vera e propria Programmazione Dinamica richiede ancora un terzo punto:

Terzo punto della Programmazione Dinamica

3. Trovare un ordine in cui riempire la tabella $M[..]$, di modo che per il calcolo di un qualsiasi elemento siano sufficenti le caselle della tabella riempite prima di esso.

Alcune caselle però dovranno essere calcolate per prime, senza usare altre caselle (casi base, come nell'induzione).

Nel problema *Ladro*

Il calcolo di $M[x][y]$ si riconduce a $M[x+1, y]$ e $M[x, y+1]$. L'ordine cercato è quindi $x = X_{max}..0$, $y = Y_{max}..0$.

Dinamica

Una volta risolto questo terzo punto, si può evitare totalmente la ricorsione riempiendo la tabella nell'ordine trovato con dei cicli `for`.

Dinamica

Una volta risolto questo terzo punto, si può evitare totalmente la ricorsione riempiendo la tabella nell'ordine trovato con dei cicli `for`.

Vantaggi

- Evitando la ricorsione, si risparmia un piccolo ammontare di tempo di esecuzione (meno chiamate a funzione).

Dinamica

Una volta risolto questo terzo punto, si può evitare totalmente la ricorsione riempiendo la tabella nell'ordine trovato con dei cicli `for`.

Vantaggi

- Evitando la ricorsione, si risparmia un piccolo ammontare di tempo di esecuzione (meno chiamate a funzione).
- Evitando la ricorsione, l'algoritmo risulta più compatto e più veloce da scrivere.

Dinamica

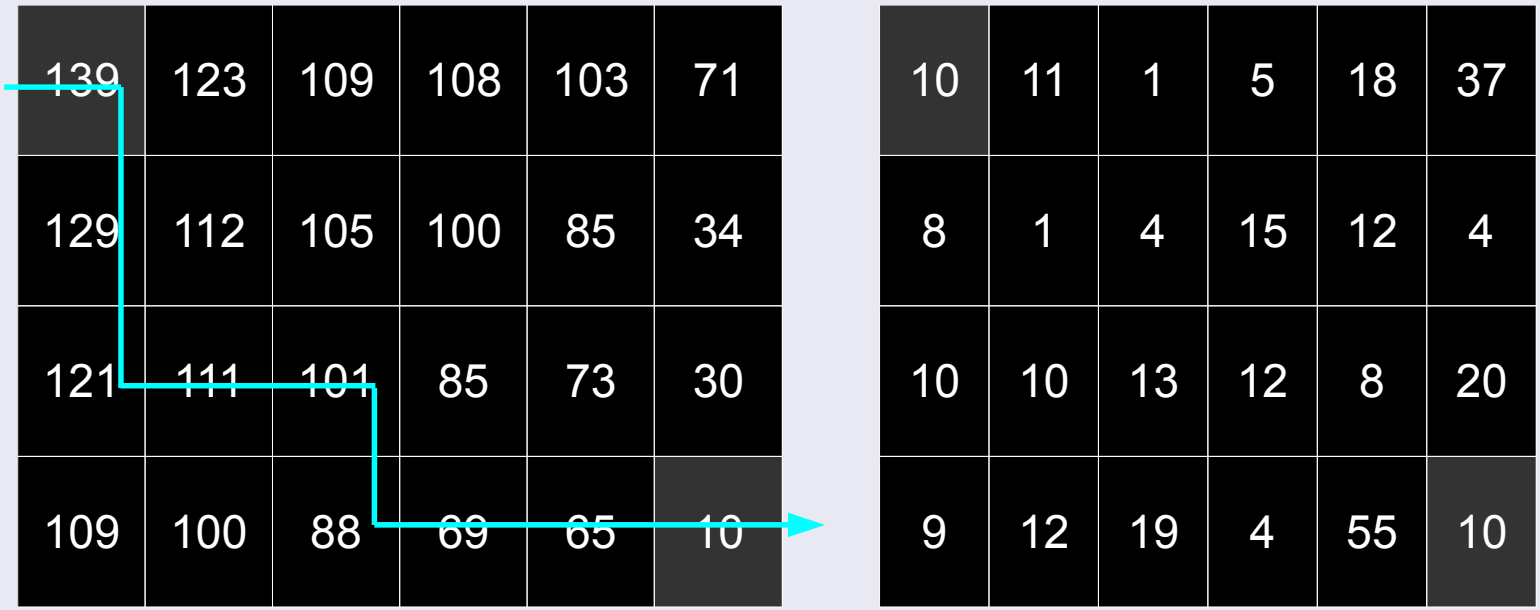
Una volta risolto questo terzo punto, si può evitare totalmente la ricorsione riempiendo la tabella nell'ordine trovato con dei cicli `for`.

Vantaggi

- Evitando la ricorsione, si risparmia un piccolo ammontare di tempo di esecuzione (meno chiamate a funzione).
- Evitando la ricorsione, l'algoritmo risulta più compatto e più veloce da scrivere.
- Riempiendo la tabella in modo sequenziale si possono sfruttare delle tecniche (che vedremo più avanti) per ridurre notevolmente l'utilizzo della memoria.

Dinamica

Esempio



La tabella che l'algoritmo di programmazione dinamica riempie per il problema *Ladro*