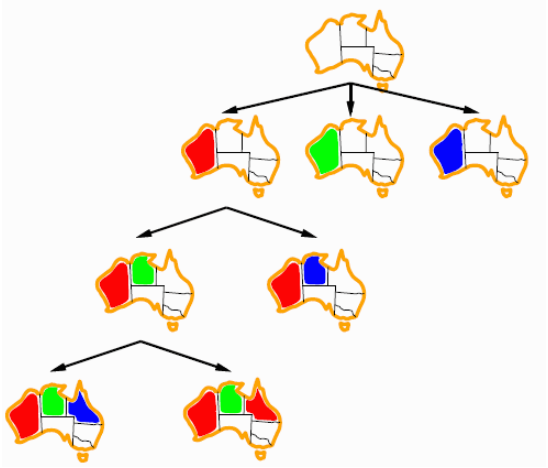
UNIVERSITÀ
DI PARMA

csp & backtracking

informatica e laboratorio di programmazione

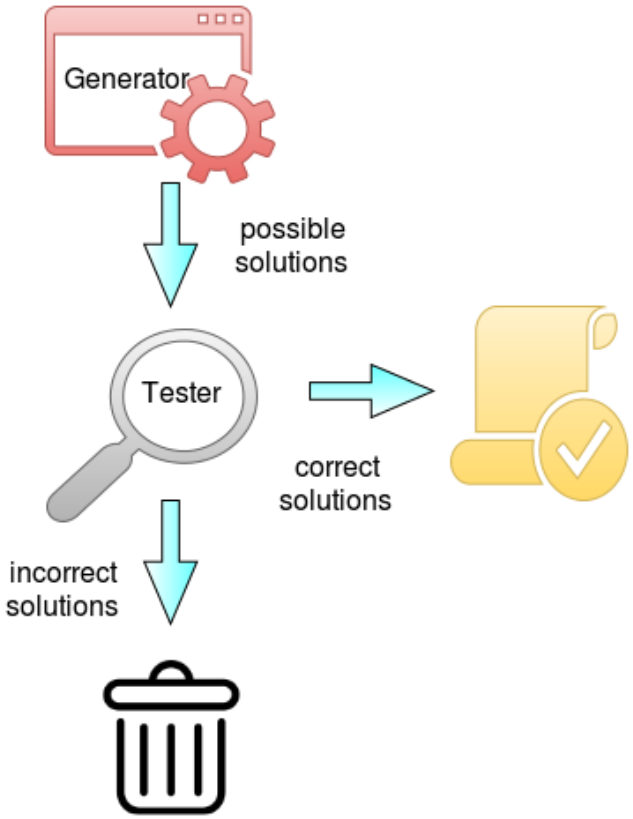


CSP (Constraint Satisfaction Problem)

- ***CSP*** = problemi di soddisfacimento di vincoli
- caratteristiche
  - i problemi di soddisfacimento di vincoli sono caratterizzati da:
    - un insieme di ***variabili*** che possono assumere ***valori*** in un certo ***dominio***
    - un insieme di ***vincoli*** che devono essere rispettati dai ***valori*** delle variabili

generate & test

- ***G&T*** è una semplice tecnica per risolvere problemi CSP
  - si assegna un ***valore*** ad ***ogni variabile***
  - si verifica se tutti i ***vincoli*** sono soddisfatti
    - se i vincoli sono ***soddisfatti*** -> è stata trovata una ***soluzione***
    - ***altrimenti*** si ***prova*** con valori ***diversi***
  - il procedimento ***continua***
    - finché ***non ci sono più*** assegnamenti nuovi da testare
  - ***tutte*** le soluzioni sono testate



criptoaritmetica – soluzione banale

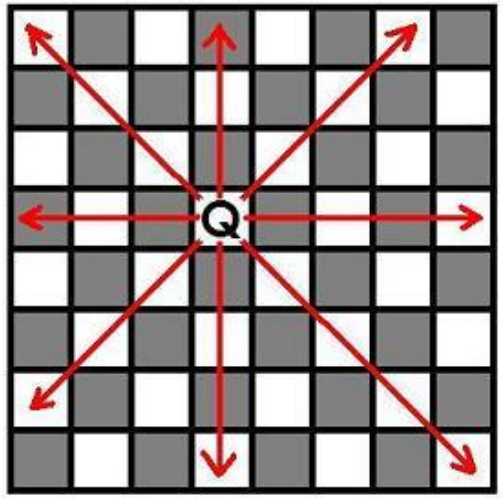
```
def controlla(t,w,o,f,u,r):
    # tutti diversi --- (vincolo 1)
    if t==w or t==o or t==f or t==u or t==r:
        return False
    if w==o or w==f or w==u or w==r:
        return False
    if o==f or o==u or o==r:
        return False
    if f==u or f==r:
        return False
    if u==r:
        return False
    # ------------------
    # verifica somma --- (vincolo 2)
    two = t*100 + w*10 + o
    four = f*1000 + o*100 + u*10 + r
    if two + two != four:
        return False
    return True
```

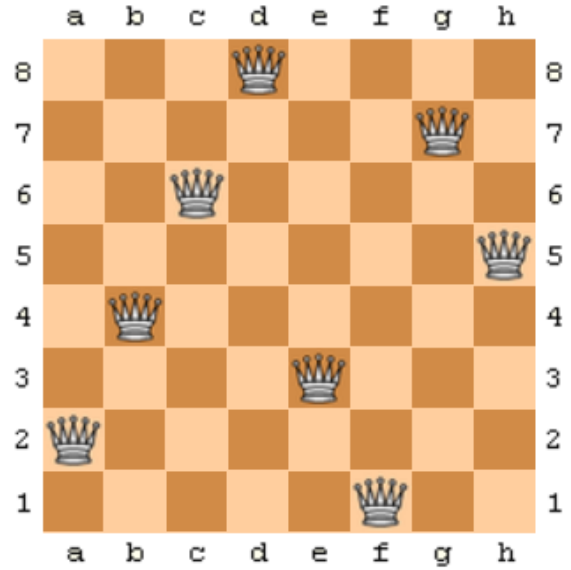
```
def main():
    # generazione valori
    for t in range(1,10):
        for w in range(10):
            for o in range(10):
                for f in range(1,10):
                    for u in range(10):
                        for r in range(10):
                            # verifica vincoli
                            if controlla(t,w,o,f,u,r):
                                print()
                                print('  ',t,w,o,'+')
                                print('  ',t,w,o,'=')
                                print('------------------')
                                print(f,o,u,r)
```

```
    T  W  O
+   T  W  O
-----------
F  O  U  R
```

problema delle 8 regine

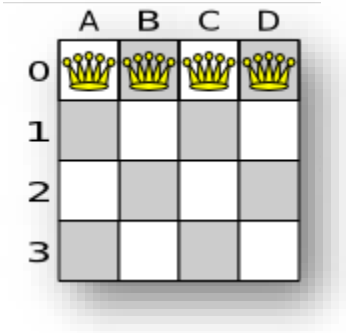
- posizionare ***8 regine*** su una scacchiera ***8x8*** in modo che nessuna di esse possa catturarne un'altra
  - nessuna regina deve avere una colonna, riga o diagonale in comune con un'altra regina
- il problema è un esempio del più generale problema delle ***n regine*** su una scacchiera ***n × n***





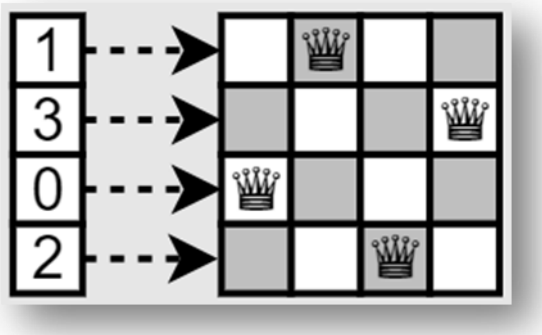
8 regine G&T

- generate
  - inserire 8 regine in tutte le possibili combinazioni in una scacchiera 8x8
- test
  - per ogni combinazione verificare se nessuna regina può catturarne un'altra



8 regine G&T semplificato

- dai vincoli si evince che ogni ***riga*** può contenere al massimo ***una regina***
  - deve contenere ***esattamente*** una regina
- è possibile quindi rappresentare una ***lista*** di 8 elementi con indici 0..7 *(n elementi con indice 0..n-1)* che contengono valori compresi fra 0 e 7 *(0 e n-1)* che rappresentano la colonna in cui è posizionata la regina sulla riga
- ***generate***
  - tutte le possibili combinazioni di 8 valori in posizioni
    - 8^8 **= 16777216 ≅ 16 milioni**
- ***test***
  - per ogni combinazione verificare se nessuna regina può catturarne un'altra

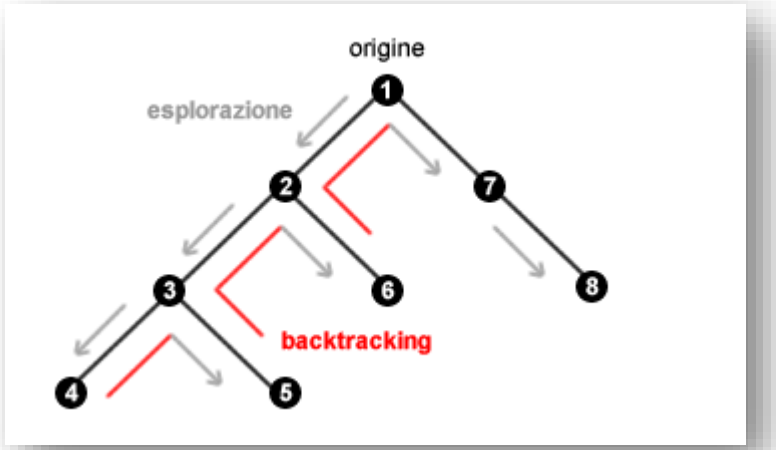


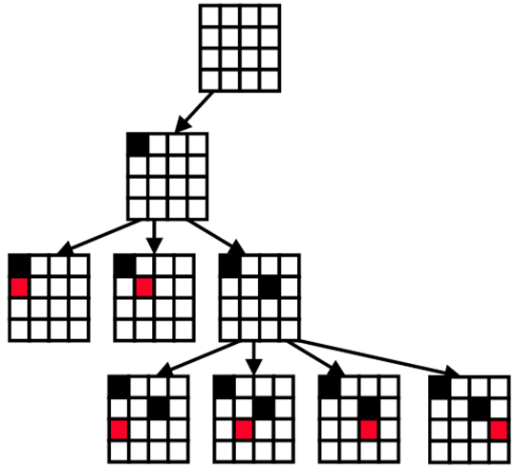
Standard Backtracking

- ***Standard Backtracking***: a seguito di ***ogni assegnamento*** si ***verifica*** se tutti i vincoli sono soddisfatti
  - se sono ***soddisfatti*** si ***continua*** verso la soluzione
  - ***altrimenti*** si verifica se la variabile appena assegnata ha ancora valori da provare
    - se sì si ***prova*** con un ***nuovo valore***
    - se no si ***torna indietro*** e si sceglie una nuova variabile
- il procedimento ***continua*** finché non ci sono più assegnamenti nuovi da provare (*o si è trovata una soluzione*)
- ***tutte*** le soluzioni sono testate

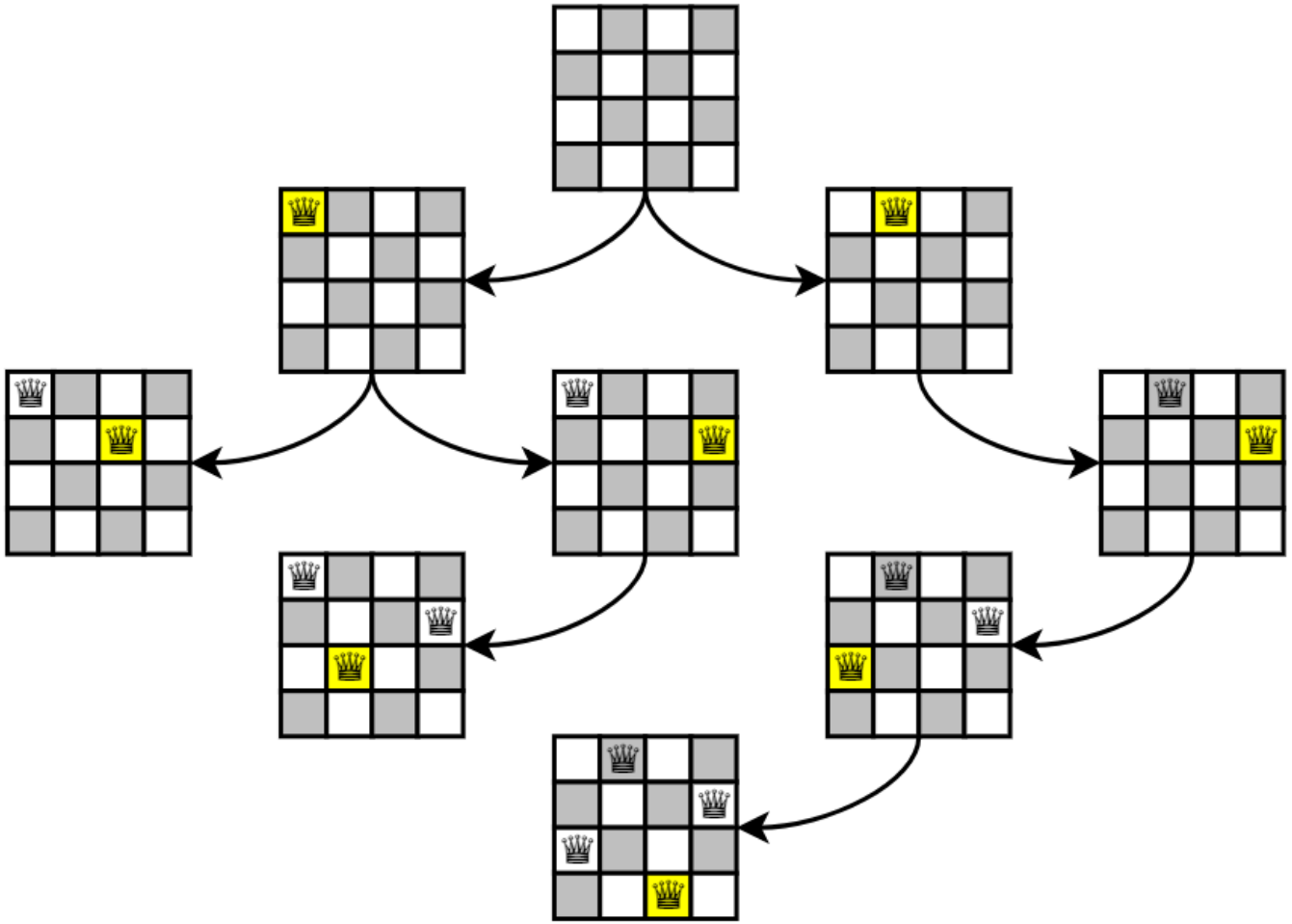
backtracking

- ***backtracking*** (monitoraggio a ritroso) tecnica di ricerca soluzioni per problemi in cui devono essere soddisfatti ***vincoli***
- ***enumera*** tutte le ***possibili soluzioni*** e ***scarta*** quelle che non soddisfano i vincoli

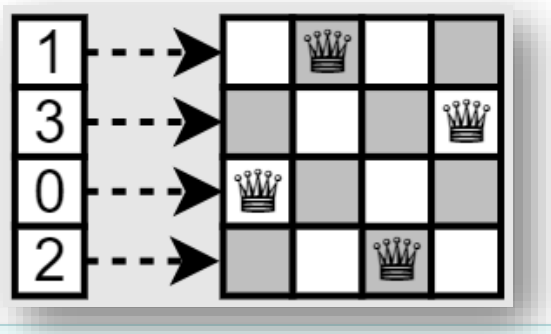




n regine - backtracking



```
def stampa_scacchiera(scacchiera: list):
    ''' visualizza la scacchiera '''
    for r in range(len(scacchiera)):
        for c in range(len(scacchiera)):
            if c == scacchiera[r]:
                print('|♛', end='')      #presente regina
            else:
                print('| ', end='')      #casella vuota
        print('|')                       #fine riga
```



*scacchiera è una lista di **int**: per ogni riga della scacchiera, memorizza la posizione della regina sulla colonna*

```
def sotto_attacco(scacchiera: list, x: int, y: int) -> bool:
    ''' true se la regina in colonna x e riga y
    è sotto attacco da altre regine '''
    # controllo le righe precedenti
    for r in range(1, y + 1):  # fino alla riga attuale
        # direzioni di controllo: ↖ ↑ ↗
        if scacchiera[y - r] in (x - r, x, x + r):
            return True
    return False
```

n regine - ricorsione

```
def posiziona_regine(scacchiera: list, r=0) -> bool:
    '''
    cerca di posizionare una regina nella scacchiera
    in riga r (le precedenti righe contengono regine
    '''
    if r == len(scacchiera):
        return True  # successo! tutte le righe contengono regine
    # prova inserimento regina in tutte le colonne
    for c in range(len(scacchiera)):
        if not sotto_attacco(scacchiera, c, r):
            scacchiera[r] = c  # possibile inserire regina in colonna c
            # passaggio alla riga successiva
            if posiziona_regine(scacchiera, r + 1):
                return True

            scacchiera[r] = None  # non è possibile inserire, backtrack
    return False
```

sudoku (数独)

- il sudoku (数独) è un gioco di logica nel quale al giocatore viene proposta una griglia di 9×9 celle, ciascuna delle quali può contenere un numero da 1 a 9, oppure essere vuota
- la griglia è suddivisa in 9 righe orizzontali, nove colonne verticali e in 9 "sottogriglie", chiamate regioni, di 3×3 celle contigue
- le griglie proposte al giocatore hanno da 20 a 35 celle contenenti un numero
- scopo del gioco è quello di riempire le caselle bianche con numeri da 1 a 9, in modo tale che in ogni riga, colonna e regione siano presenti tutte le cifre da 1 a 9 senza ripetizioni

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

soluzione ingenua (G&T) Generate and Test

- si inserisce un numero in ogni cella vuota
- si verifica che le regole del gioco siano rispettate
- se qualche regola non è rispettata si prova con altri numeri
- complessità
  - il numero totale di celle è 81
  - supponendo riempite 31 celle ne rimangono 50
  - il numero di combinazioni possibili da provare è 9^{50} (circa $5*10^{47}$)
- supponendo 1 nanosecondo per formulare una combinazione il tempo previsto è circa . . .

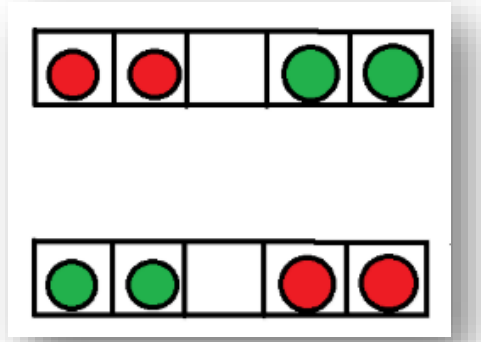
calcoliamo ...

- 1 anno = 365 giorni
- 1 anno = 8760 ore
- 1 anno = 525600 minuti
- 1 anno = 3.154×10^7 secondi
- 1 anno = 3.154×10^{16} nanosecondi
- $5 \times 10^{47} / 3.154 \times 10^{16} = 10^{31}$ anni
- età dell'universo ... 10^{10} anni :(

backtracking

esercizi

- ***11.1 puzzle di Cindy***
  - piano di gioco: ***2n+1 celle allineate***
  - si parte con ***n pedine rosse tutte a sinistra***, ***n pedine verdi tutte a destra***, ed una cella libera in mezzo
  - le pedine ***rosse*** si possono spostare solo a ***destra***, quelle ***verdi*** solo a ***sinistra*** (senza poter tornare indietro)
  - ad ogni mossa, una qualsiasi pedina può:
    - ***avanzare*** di una posizione, se davanti ha una cella libera
    - oppure ***scavalcare*** esattamente una pedina dell'altro colore, se c'è una cella libera subito dopo
  - l'applicazione deve trovare ***automaticamente*** le mosse per ***invertire*** la posizione di tutte le pedine



esercizi (2)

- ***11.2 8 regine***
  - modificare il codice proposto che trova tutte le soluzioni del problema delle 8 regine visualizzando il numero di tentativi effettuati e il numero di soluzioni trovate

- ***11.3 8 regine generate & test***
  - realizzare un'applicazione che risolve il problema delle 8 regine secondo la metodologia generate & test (versione semplificata – una sola regina per riga)
  - visualizzare il numero di tentativi effettuati e il numero di soluzioni trovate

- ***11.4 sudoku (verifica)***
  - verificare se una soluzione parziale di un sudoku è ammissibile (soddisfa i vincoli)

- ***11.5 sudoku (applicazione)***
  - realizzare un'applicazione che legge da file la situazione iniziale di un sodoku
  - permette ripetutamente all'utente
    - di inserire un valore specificando la cella
      - accetta il valore solo se rispetta i vincoli
    - di 'cancellare' un valore da una cella
  - verifica se il sudoku è completo

- ***11.4 criptoartimetica***
  - assegnare alle lettere **`U,N,O,D,E,T,R`** valori tutti differenti nell'intervallo [0,9]
  - in modo che sia verificata l'operazione:

```
UNO +
DUE =
-----
TRE
```