# Design and Analysis of Algorithm

## Backtrack (I)

## Backtrack Paradigm

Recursive approach is essentially travelling the whole tree defined by the recusive relation.

- The subtrees may repeat, so we need to cache intermediate results to improve efficiency. This is exactly the essence of dynamic programming.

# Backtrack Paradigm

Recursive approach is essentially travelling the whole tree defined by the recusive relation.

- The subtrees may repeat, so we need to cache intermediate results to improve efficiency. This is exactly the essence of dynamic programming.

For some problems, the subtrees will not overlap.

- In such case, there is no better algorithm other than travelling the entire tree. But, we can travel the entire tree smartly.
- This is what backtrack technique concerns: stop visiting the subtree if the solution won't appear and backtrack to the parent node
  - basic backtrack strategy: Domino property defined by problem constraint
  - advanced backtrack strategy: branch-and-bound

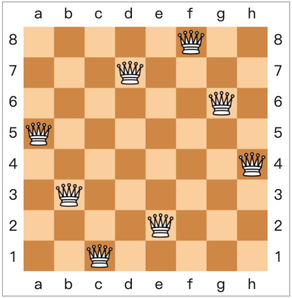
# Outline

## Example 1: Eight Queen Problems

Eight queens puzzle. Placing eight chess queens on an $8 \times 8$ chessboard so that no two queens threaten each other.

- a solution requires that no two queens share the same row, column, or diagonal.

Eight queens puzzle is a special case of the more general $n$ queens problem: placing $n$ non-attacking queens on an $n \times n$ chessboard.



## Counting Solutions

Solution is an $n$-dimension vector over $[n]$: exist for all natural numbers $n$ with the exception of $n = 2, 3$.

- Eight queens puzzle has $92$ distinct solutions, the entire solution space is $C_{64}^{8} = 4,426,165,368$.
- If solutions that differ only by the symmetry operations of rotation and reflection of the board are counted as one, the puzzle has $12$ solutions, called as fundamental solutions.

| $n$ | fundamental | all |
|---|---:|---:|
| $8$ | $12$ | $92$ |
| $9$ | $46$ | $352$ |
| $10$ | $92$ | $724$ |
| $\dots$ | $\dots$ | $\dots$ |
| $26$ | $2,789,712,466,510,289$ | $22,317,699,616,364,044$ |
| $27$ | $29,363,495,934,315,694$ | $234,907,967,154,122,528$ |

## Background of Eight Queen Puzzle

### Origin of Eight Queen Puzzle

Max Bezzel first proposed this problem in 1848, Frank Nauck gave the first solution in 1850 and extended it to $n$ queen puzzles. Many mathematicians including Carl Guass also studied this problem.

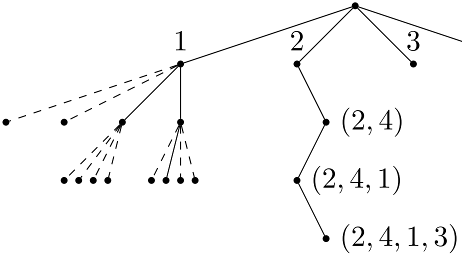
Edsger Dijkstra exemplified the power of *depth-first backtracking algorithm* via this problem.

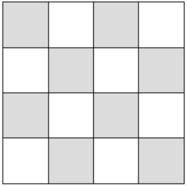
There is no known formula for the exact number of solutions, or even for its asymptotic behavior. The $27 \times 27$ board is the highest-order board that has been completely enumerated.

How to solve?

- modeling all possible solutions as leaf nodes of a tree
- traversal the solution space via travelling the tree

## Demo of Quadtree for $4$ Queens Puzzle



Travel the tree via depth-first order to find all solutions

- $i$-th level node represent $i$-th element in solution vector
- in the $i$-th level, the branching choice is less than $n - i$
- leaves correspond to solutions

## Example 2: $0$-$1$ Knapsack Problem

Problem. Given $n$ items with value $v_i$ and weight $w_i$, as well as a knapsack with weight capacity $W$. The number of each item is $1$. Find a solution that maximize the value.

Solution. $n$ dimension vector $(x_1, x_2, \dots, x_n) \in \{0,1\}^n$, $x_i = 1 \Leftrightarrow$ selecting item $i$

Nodes: $(x_1, x_2, \dots, x_k)$ corresponds to partial solution

Search space. In all level, the branching choice is always $2 \rightsquigarrow$ perfect binary tree with $2^n$ leaves

Candidate solution. Satisfy constraint $\sum_{i=1}^{n} w_i x_i \leq W$

Optimal solution. The candidate solutions that achieve maximal values.

# A Demo

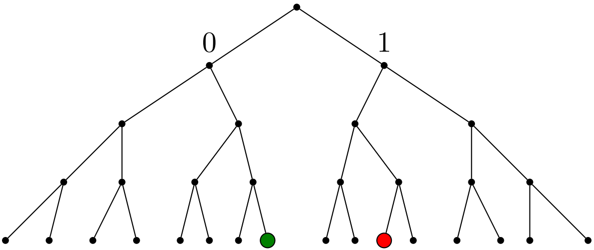
Table: $n = 4$, $W = 13$

| item | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value | 12 | 11 | 9 | 8 |
| weight | 8 | 6 | 4 | 3 |

Two candidate solutions

1. $(0, 1, 1, 1)$: $v = 28$, $w = 13$
2. $(1, 0, 1, 0)$: $v = 21$, $w = 12$

Optimal solution is $(0, 1, 1, 1)$

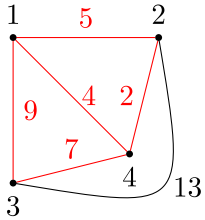


## Example 3: Traversal Salesman Problem

Problem. Given $n$ cities $C = \{c_1, c_2, \ldots, c_n\}$ and $d(c_i, c_j) \in \mathbb{Z}^+$. Find a cycle with minimal length that travels each city once.

Solution. A permutation of $(1, 2, \ldots, n)$ — $(k_1, k_2, \ldots, k_n)$ such that

$$\min \left\{ \sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1}) \right\}$$




$C = \{1, 2, 3, 4\}$

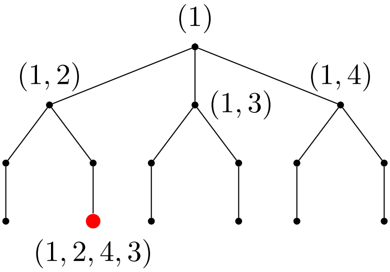
$d(1, 2) = 5$, $d(1, 3) = 9$

$d(1, 4) = 4$, $d(2, 3) = 13$

$d(2, 4) = 2$, $d(3, 4) = 7$

Solution is $(1, 2, 4, 3)$, length of cycle is $5 + 2 + 7 + 9 = 23$

## Search Space of TSP




Any node can serve as the root, cause TSP is defined over an undirected graph.

Search space. In the $i$-th level, the branching choice is always $n-i$

- obtain a tree with $(n-1)!$ leaves $\rightsquigarrow$ number of all possible permutations over $\{1,\ldots,n\}$

## Summary

Classical examples of Backtrack

- $n$ queens puzzle, $0$-$1$ knapsack, TSP

Solution: vector

Search space: tree

- nodes correspond to partial solutions, leaves correspond to candidate solutions

Search order: depth-first, breadth-first, jump-hop

# Outline

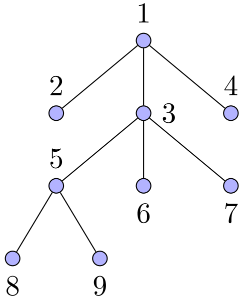
## Main Idea of Backtrack

Scope of application. Search or optimization problem

Search space. Tree

- leaves: candidate solution
- nodes: partial solution

How to search. Systematically traversal the tree: DFS, BFS, ...




DFS: $1 \to 2 \to 3 \to 5 \to 8 \to 9 \to 6 \to 7 \to 4$

BFS: $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9$

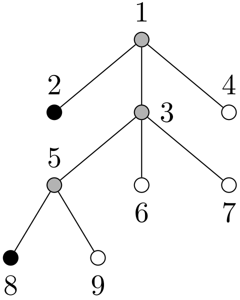
## States of Nodes

The tree is explored dynamically. Let $v$ be the candidate node (corresponding to partial solution) and $P$ be the predicate that checks if $v$ satisfies constraint.

- $P(v) = 1 \Rightarrow$ expand
- $P(v) = 0 \Rightarrow$ backtrack to parent node

States of node

- white: unexplored
- gray: visiting its subtree
- black: finishing the traversal of this subtree




DFS: $1 \to 2 \to 3 \to 5 \to 8$

- finished visiting: $2, 8$
- being visited: $1, 3, 5$
- unexplored: $9, 6, 7, 4$

## Basic Backtrack Technique: Domino Property

At node $v = (x_1, \ldots, x_k)$

$$P(x_1, \ldots, x_k) = 1 \Leftrightarrow (x_1, \ldots, x_k) \text{ meet some property}$$

Example. $n$ queens puzzle, placing $k$ queens in positions without attacking each other

Domino property $\rightsquigarrow$ admit safe backtrack

$$P(x_1, x_2, \ldots, x_{k+1}) = 1 \Rightarrow P(x_1, x_2, \ldots, x_k) = 1, 0 < k < n$$

Converse-negative proposition

$$P(x_1, x_2, \ldots, x_k) = 0 \Rightarrow P(x_1, x_2, \ldots, x_{k+1}) = 0, 0 < k < n$$

$k$-dimension vector does not satisfy constraint $\Rightarrow$ its $k+1$-dimension extension does not satisfy constraint either

- this property guarantees that backtracking will not miss any solution
- safely backtrack when $P(x_1, x_2, \ldots, x_k) = 0$

## A Counterexample

Find integer solutions for inequality

$$5x_1 + 4x_2 - x_3 \leq 10, 1 \leq x_k \leq 3, k = 1, 2, 3$$

$P(x_1, \ldots, x_k) = 1$ iff $\sum_{i=1}^{k} a_i x_i \leq 10$

Does not satisfy Domino property

$$5x_1 + 4x_2 - x_3 \leq 10 \nRightarrow 5x_1 + 4x_2 \leq 10$$

## A Counterexample

Find integer solutions for inequality

$$5x_1 + 4x_2 - x_3 \leq 10, 1 \leq x_k \leq 3, k = 1, 2, 3$$

$P(x_1, \ldots, x_k) = 1$ iff $\sum_{i=1}^{k} a_i x_i \leq 10$

Does not satisfy Domino property

$$5x_1 + 4x_2 - x_3 \leq 10 \not\Rightarrow 5x_1 + 4x_2 \leq 10$$

## A Counterexample

Find integer solutions for inequality

$$5x_1 + 4x_2 - x_3 \le 10, 1 \le x_k \le 3, k = 1, 2, 3$$

$P(x_1, \ldots, x_k) = 1$ iff $\sum_{i=1}^{k} a_i x_i \le 10$

Does not satisfy Domino property

$$5x_1 + 4x_2 - x_3 \le 10 \not\Rightarrow 5x_1 + 4x_2 \le 10$$

## A Counterexample

Find integer solutions for inequality

$$5x_1 + 4x_2 - x_3 \leq 10, 1 \leq x_k \leq 3, k = 1, 2, 3$$

$P(x_1, \ldots, x_k) = 1$ iff $\sum_{i=1}^{k} a_i x_i \leq 10$

Does not satisfy Domino property

$$5x_1 + 4x_2 - x_3 \leq 10 \not\Rightarrow 5x_1 + 4x_2 \leq 10$$


Modification to satisfy Domino property: set $x_3' = 3 - x_3$

$$5x_1 + 4x_2 + x_3' \leq 13, 1 \leq x_1, x_2 \leq 3, 0 \leq x_3' \leq 2$$

## Summary

The premise condition to use backtrack: Domino property

General steps of backtrack algorithm

- Define solution vector (include the range of every element), $(x_1, x_2, \ldots, x_n) \in X_1 \times \cdots \times X_n$
- After fixing $(x_1, x_2, \ldots, x_{k-1})$, update admissible range of $x_k$ as $A_k \subseteq X_k$ using predicate $P$
- Decide if Domino property is satisfied
- Decide the search strategy: DFS, BFS
- Decide the data structure to store the search path

## Backtrack Recursive Template

**Algorithm 1:** BackTrack($n$) //output all solutions

**for** $k = 1$ *to* $n$ **do** $A_k \leftarrow X_k$; //initialize
ReBack($1$);

**Algorithm 2:** ReBack($k$) //$k$ is the current depth of recursion

**if** $k = n$ **then return** solution $(x_1, \ldots, x_n)$;
**else**
    **while** $A_k \neq \emptyset$ **do**
        $x_k \leftarrow A_k$ //according to some order;
        $A_k \leftarrow A_k - \{x_k\}$;
        update $A_{k+1}$, ReBack($k + 1$);
    **end**

- The above is the oversimplify pseudocode.
- One must be careful when dealing with domains $A_k$ and solution vector $x$ when coding (value transfer vs. reference transfer)

## Backtrack Iterative Template

**Algorithm 3:** BackTrack($n$) //all solutions

1: **for** $k = 1$ *to* $n$ **do** $A_k \leftarrow X_k$; //initialize

2: $k \leftarrow 1$;

3: **while** $A_k \neq \emptyset$ **do**

4: $\quad x_k \leftarrow A_k$; $A_k \leftarrow A_k - \{x\}$;

5: $\quad$ **if** $k < n$ **then** $k \leftarrow k + 1$;

6: $\quad$ **else** $(x_1, x_2, \ldots, x_n)$ is solution;

7: **end**

8: **if** $k > 1$ **then** $k \leftarrow k - 1$; goto 3;

- $A_k$ is determined by $(x_1, \ldots, x_{k-1})$
- The algorithm terminates when all $A_i$ are empty. Otherwise, it will backtrack (line 8).

# Outline

## Loading Problem

Problem. Given $n$ containers with weight $w_i$, two boats with weight capacity $W_1$ and $W_2$ s.t. $w_1 + \cdots + w_n \leq W_1 + W_2$.

Goal. If there exists a scheme to load the $n$ containers on two boats. Please give a scheme if it is solvable.

## Loading Problem

Problem. Given $n$ containers with weight $w_i$, two boats with weight capacity $W_1$ and $W_2$ s.t. $w_1 + \cdots + w_n \leq W_1 + W_2$.

Goal. If there exists a scheme to load the $n$ containers on two boats. Please give a scheme if it is solvable.

Example

- $w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$, $W_1 = 152$, $W_2 = 130$

Solution: load $1, 3, 6, 7$ on boat 1 and the rest on boat 2

## Loading Problem

Problem. Given $n$ containers with weight $w_i$, two boats with weight capacity $W_1$ and $W_2$ s.t. $w_1 + \cdots + w_n \leq W_1 + W_2$.

Goal. If there exists a scheme to load the $n$ containers on two boats. Please give a scheme if it is solvable.

---

Example

- $w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$, $W_1 = 152$, $W_2 = 130$

---

Solution: load $1, 3, 6, 7$ on boat 1 and the rest on boat 2

Main idea: Let the total weights be $W$.

1. Load on boat 1 first. Using backtrack to find a solution that maximizes $W_1^*$, where $W_1^*$ is the real capacity.
2. Then check if $W - W_1^* \leq W_2$. Return "yes" if true and "no" otherwise.

## Pseudocode

**Algorithm 4:** Loading($W_1$)

1: $W_1^* \leftarrow 0$; $C \leftarrow 0$; $i \leftarrow 1$;
2: **while** $i \leq n$ **do** //line 3-4: whether to load container $i$
3: **if** $C + w_i \leq W_1$ **then** $C \leftarrow C + w_i$, $x[i] \leftarrow 1$, $i = i + 1$ ;
4: **else** $x[i] \leftarrow 0$, $i \leftarrow i + 1$;
5: **end**
6: **if** $W_1^* < C$ **then** record solution, $W_1^* \leftarrow C$;
7: **while** $i > 1$ *and* $x[i] = 0$ **do** $i - 1$; //find a backtrack node
8: **if** $i = 0$ **then return** optimal solution; //backtrack to root
9: **else** $x[i] \leftarrow 0$; $C \leftarrow C - w_i$; $i = i + 1$, goto 2 ; //$x[i] = 1$: continue to search

line 7-9: find a backtrack point

1. line 8: have travelled all the tree and back to the root
2. line 9: find a left branch, means there still exist unexplored right branch $\rightsquigarrow$ change it to right branch

## Demo

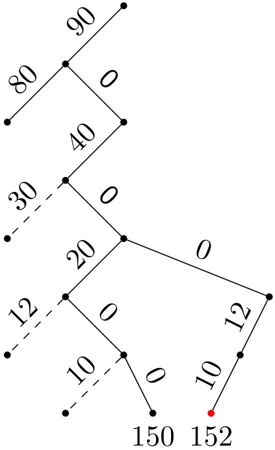
$w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$

$W_1 = 152,\ W_2 = 130$

# Demo

$$w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$$

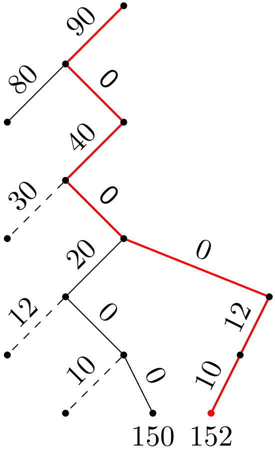
$$W_1 = 152,\ W_2 = 130$$



# Demo

$$w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$$

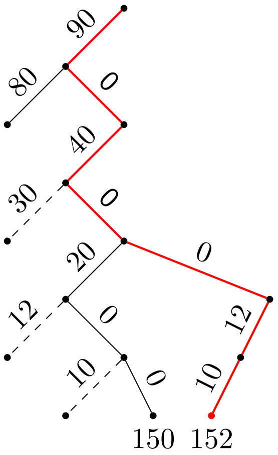
$W_1 = 152,\ W_2 = 130$



## Demo

$$w_1 = 90, w_2 = 80, w_3 = 40, w_4 = 30, w_5 = 20, w_6 = 12, w_7 = 10$$

$W_1 = 152$, $W_2 = 130$




it is loadable

- $1, 3, 6, 7$ on boat $1$
- $2, 4, 5$ on boat $2$

time complexity $W(n) = O(2^n)$

## Outline

## Graph Coloring Problem

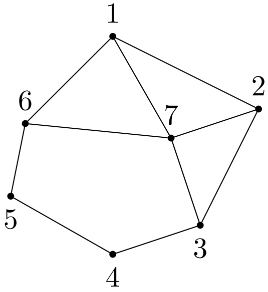
Problem. Undirected graph $G$ and $m$ colors. Coloring the vertices to ensure the connected two vertices with different color.

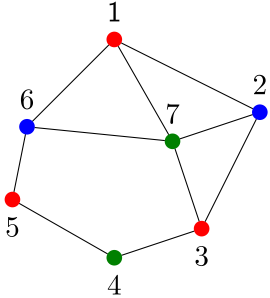
Goal. Output all possible coloring schemes. Output “no” if there is none.

# Graph Coloring Problem

Problem. Undirected graph $G$ and $m$ colors. Coloring the vertices to ensure the connected two vertices with different color.

Goal. Output all possible coloring schemes. Output "no" if there is none.




$$n = 7, m = 3$$

Input. $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, color set $\{1, 2, \ldots, m\}$

Solution vector. $(x_1, x_2, \ldots, x_n)$, $x_i \in [m]$

$(x_1, \ldots, x_k)$ gives partial solution for vertice set $\{1, 2, \ldots, k\}$

Search tree. $m$-fork tree

Constraint. At node $(x_1, \ldots, x_k)$, the set of available colors for node $k + 1$ is not empty.

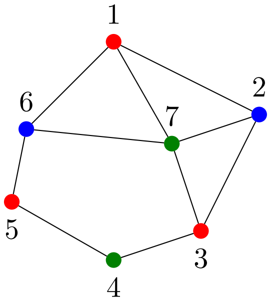
- If the nodes in adjacent list have used up $m$ colors, then node $k + 1$ is not colorable. In this case, back to parent node. (Domino property obviously holds)

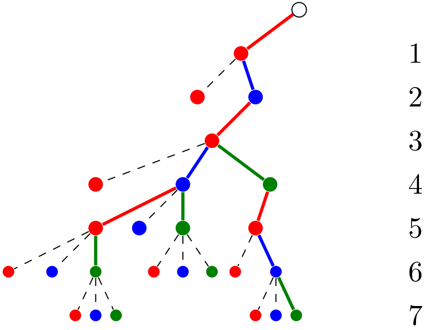
Search strategy: DFS

Time complexity: $O(nm^n)$

- the depth of tree is $n \Rightarrow$ totally at most $m^n$ nodes
- every step need to find usable colors $\Rightarrow$ require $O(n)$ cost

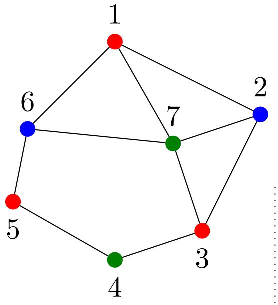
# Demo

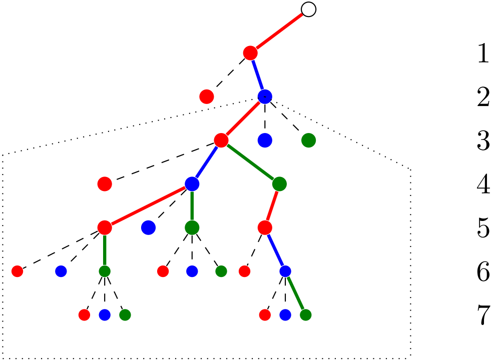



the first solution vector: $(1, 2, 1, 3, 1, 2, 3)$

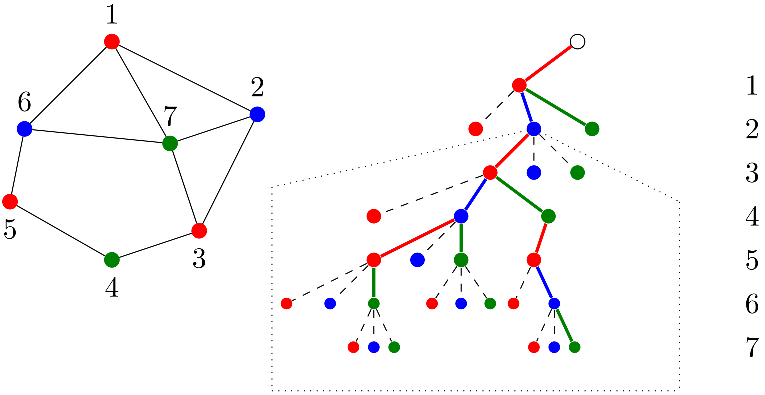
# The Structure of Search Tree



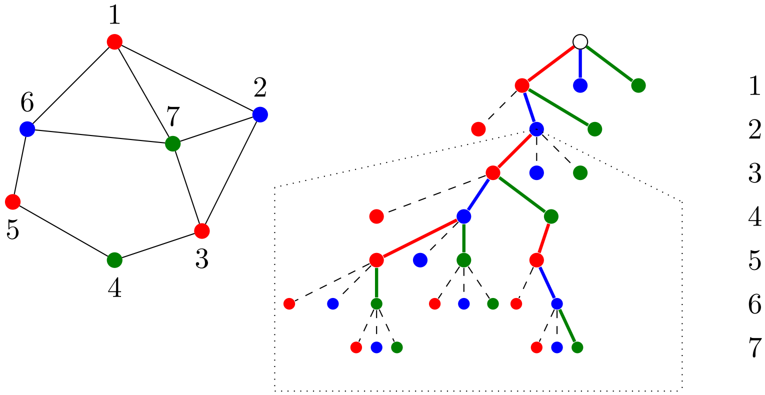

The first solution vector: $(1, 2, 1, 3, 1, 2, 3)$

# The Structure of Search Tree




The first solution vector: $(1, 2, 1, 3, 1, 2, 3)$

## The Structure of Search Tree




The first solution vector: $(1, 2, 1, 3, 1, 2, 3)$

## Complexity Analysis

Time complexity: $O(nm^n)$

Symmetry $\rightsquigarrow$ only need to search at most $1/6$ solution space

- the permutation over $(1, 2, 3)$ is $6 \rightsquigarrow$ for any specific solution, there exist $6$ homogeneous solution
- level-$2$ has $2$-fold solution (e.g. color blue and green are exchangeable), level-$1$ has $3$-fold solution (node $1$ can pick red, green or blue); the closer to the root, the more choice of replacement.

## Complexity Analysis

Time complexity: $O(nm^n)$

Symmetry $\rightsquigarrow$ only need to search at most $1/6$ solution space

- the permutation over $(1, 2, 3)$ is $6 \rightsquigarrow$ for any specific solution, there exist $6$ homogeneous solution
- level-$2$ has $2$-fold solution (e.g. color blue and green are exchangeable), level-$1$ has $3$-fold solution (node $1$ can pick red, green or blue); the closer to the root, the more choice of replacement.

Additional reasoning also helps to reduce search scope

- Example: if node $1, 2, 3$ have been colored differently, then node $7$ is definitely non-colorable because it connects with node $1, 2, 3 \rightsquigarrow$ backtrack from this node
- Need trade-off between search and decide

# Applications of Graph Coloring

## Arrangement of meeting room

There are $n$ events to be arranged, if the slots of event $i$ and event $j$ overlap, we say $i$ and $j$ are not compatible. How to arrange these events with smallest number of meeting rooms?

# Applications of Graph Coloring

## Arrangement of meeting room

There are $n$ events to be arranged, if the slots of event $i$ and event $j$ overlap, we say $i$ and $j$ are not compatible. How to arrange these events with smallest number of meeting rooms?

## Modeling

- Treat event as node, if $i, j$ are not compatible, then add an edge between $i$ and $j$.
- Treat meeting rooms as colors.

The arrangement problem is transformed to finding a coloring scheme with smallest colors.

# Outline

## Estimation of Leaves

Sometimes, we need to know the size of problems (captured by the number of nodes)

- Finding the exact number may require to travel the whole tree exhaustively, which is equivalent to solve the problem.

### Monte Carlo method

1. Choose a random path from root until there is no more branching, i.e., randomly and sequentially assign values to $x_1, x_2, \ldots,$ until the vector cannot be further expanded.
2. Assume other $|A_i| - 1$ branches has the same path as selected one, calculate the nodes of search tree
3. Repeat step 1 and 2, compute the average number of nodes.

## Estimate $n$ Queen Puzzle

**Algorithm 5:** MonteCarlo$(n, t)$

**Input:** $n = \#$ number of queens, $t = \#$ number of sampling
**Output:** $\ell$, average number of node of $t$ times sampling

```
ℓ ← 0;
for i = 1 to t do                      //sampling time is t
    m ← Estimate(n);          //number of nodes;
    ℓ ← ℓ + m;
end
ℓ ← ℓ/t;
```

## One Sampling

Parameter

- $\ell$ is the total number of nodes
- $k$ is the depth
- $r_{\text{prev}}$: # (nodes on the previous level)
- $r_{\text{current}}$: # (nodes on the current level)
- $r_{\text{current}} = r_{\text{prev}} \times \#(\text{branches})$
- $n$ is the depth of tree

Computation oder: randomly select until reaching the leaves

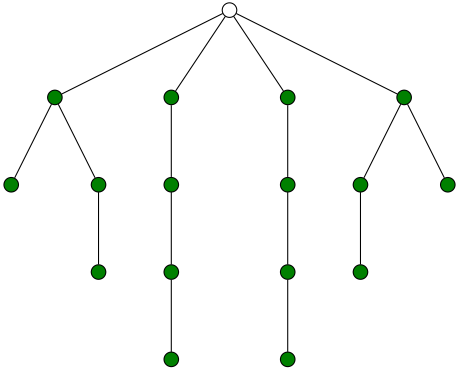
$$r_{\text{prev}} = 2,\ r_{\text{current}} = r_{\text{prev}} \cdot 3 = 6$$

# Pseudocode

**Algorithm 6:** Estimate($n$)

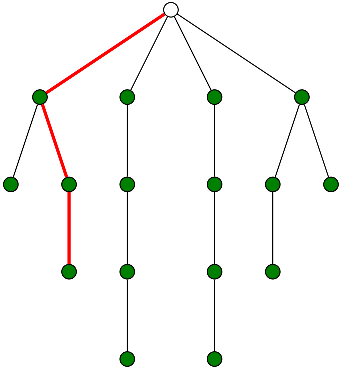
$\ell \leftarrow 1$; $r_{\text{prev}} \leftarrow 1$; $k \leftarrow 1$; //the root node;
**while** $k \leq n$ **do**
  **if** $A_k = \emptyset$ **then return** $\ell$; //no more branch
  $x_k \xleftarrow{R} A_k$ //randomly select a branch;
  $r_{\text{current}} \leftarrow r_{\text{prev}} \times |A_k|$ //number of nodes on $k$ level;
  $\ell \leftarrow \ell + r_{\text{current}}$ ;
  $r_{\text{prev}} \leftarrow r_{\text{current}}$;
  $k \leftarrow k + 1$;
**end**

# Real Case: $4$-Queens Puzzle

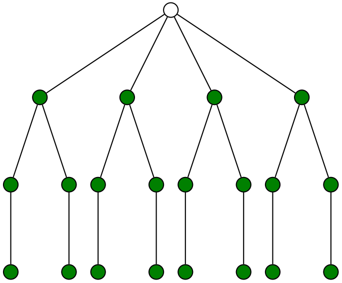


$17$ nodes

## Random Selected Path $1$

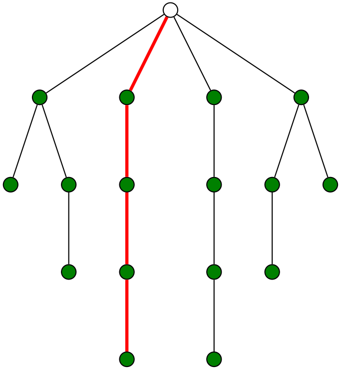


case 1: $(1, 4, 2)$

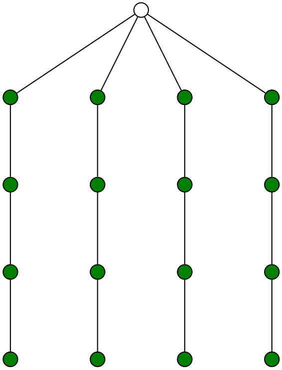


$21$ nodes

## Randomly Selected Path $2$

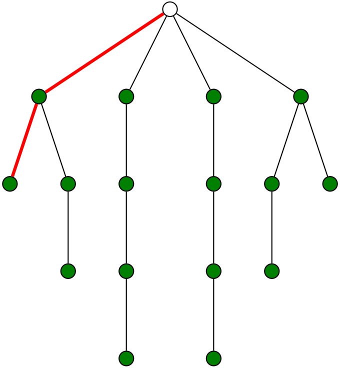


case 2: $(2, 4, 1, 3)$

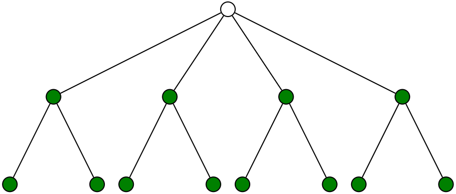


$17$ nodes

# Randomly Selected Path $3$



case 3: $(1, 3)$



$13$ nodes

## Estimation Result

Suppose sampling four times

- case 1: $1$
- case 2: $1$
- case 3: $2$

Average number of nodes: $(21 \times 1 + 17 \times 1 + 13 \times 2)/4 = 16$

The real number of nodes: $17$

- more samplings will make the estimation approaches the real number