Backtracking (1)

By: Aminul Islam

Based on Chapter 5 of Foundations of Algorithms

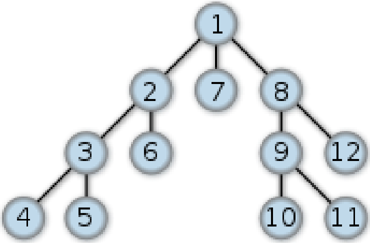
Objectives

Objectives

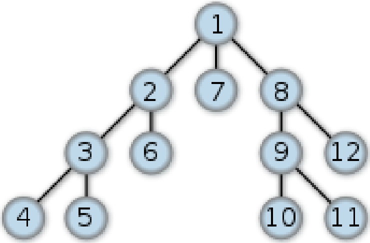
- Describe the backtrack programming technique
- Determine when the backtracking technique is an appropriate approach to solving a problem
- Define a state space tree for a given problem
- Define when a node in a state space tree for a given problem is promising/non-promising
- Create an algorithm to prune a state space tree
- Create an algorithm to apply the backtracking technique to solve a given problem

Depth-First Search (DFS)

Depth-First Search (DFS)

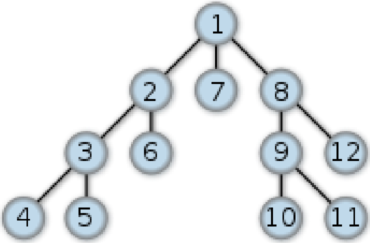


Depth-First Search (DFS)



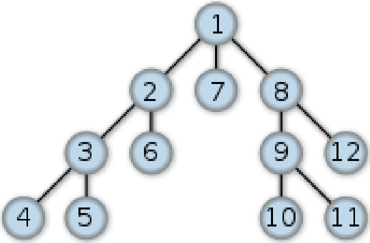
- The root is visited first

Depth-First Search (DFS)



- The root is visited first
- followed by its descendants

Depth-First Search (DFS)

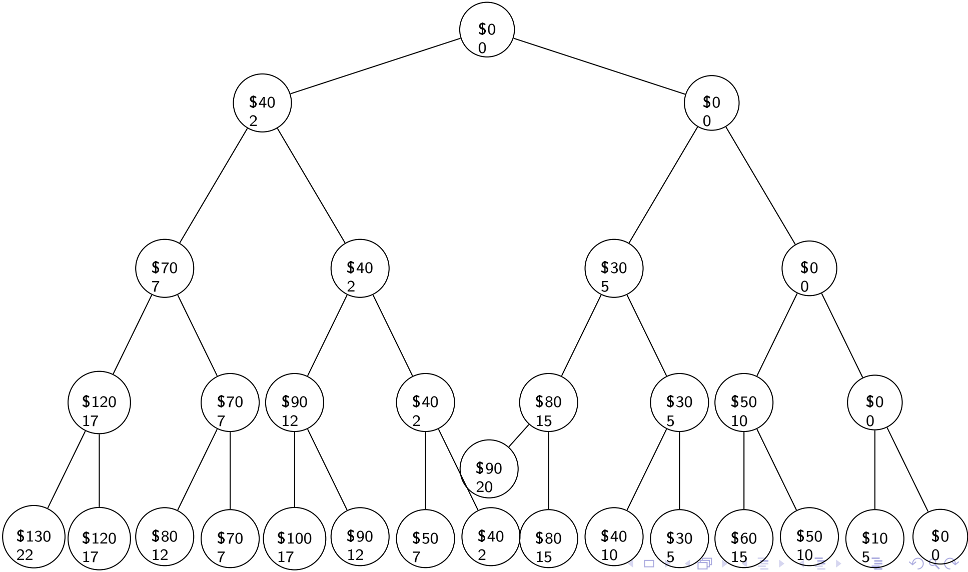


- The root is visited first
- followed by its descendants
- Assume the children of a node is visited from left to right

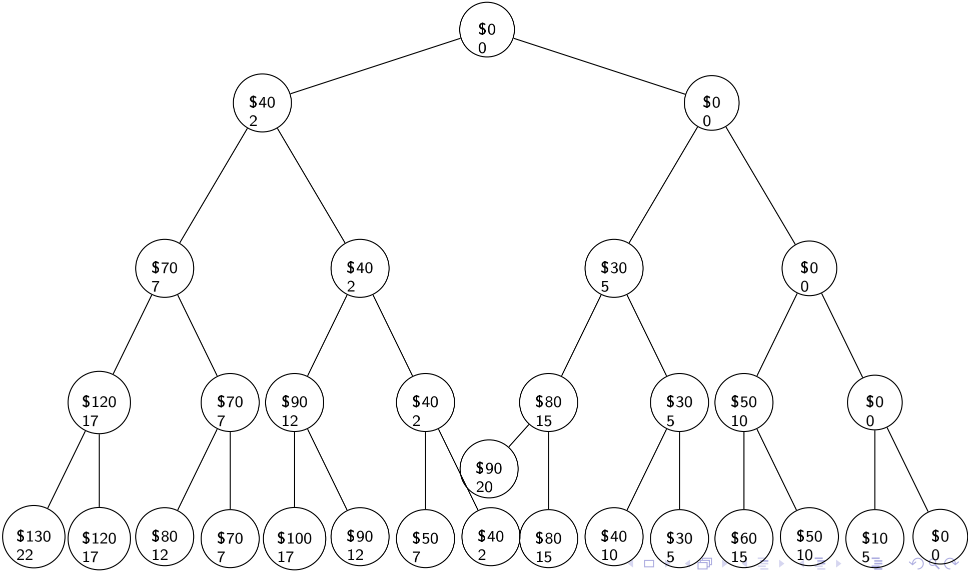
Example of **Depth-First Search (DFS)**

Example of **Depth-First Search (DFS)**

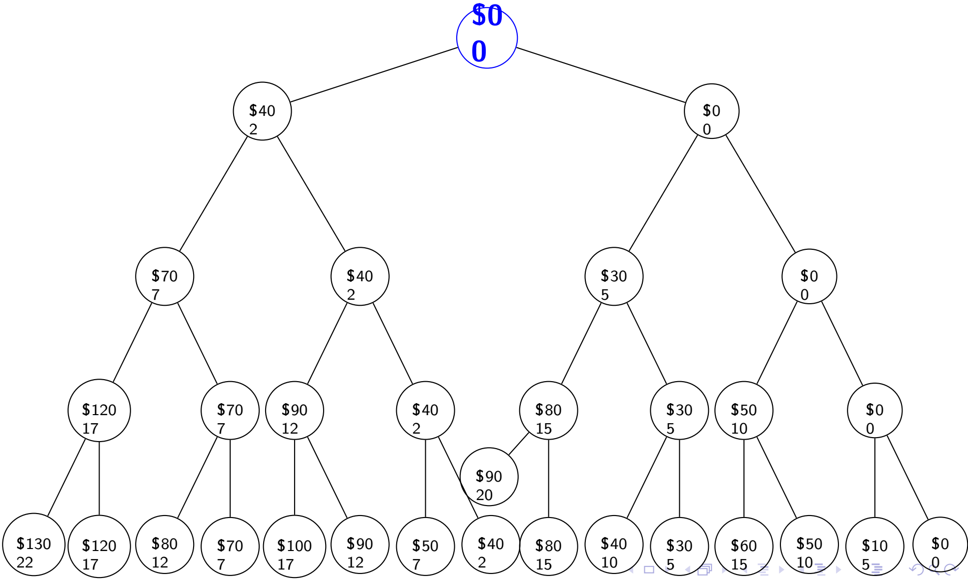
Example of **Depth-First Search (DFS)**



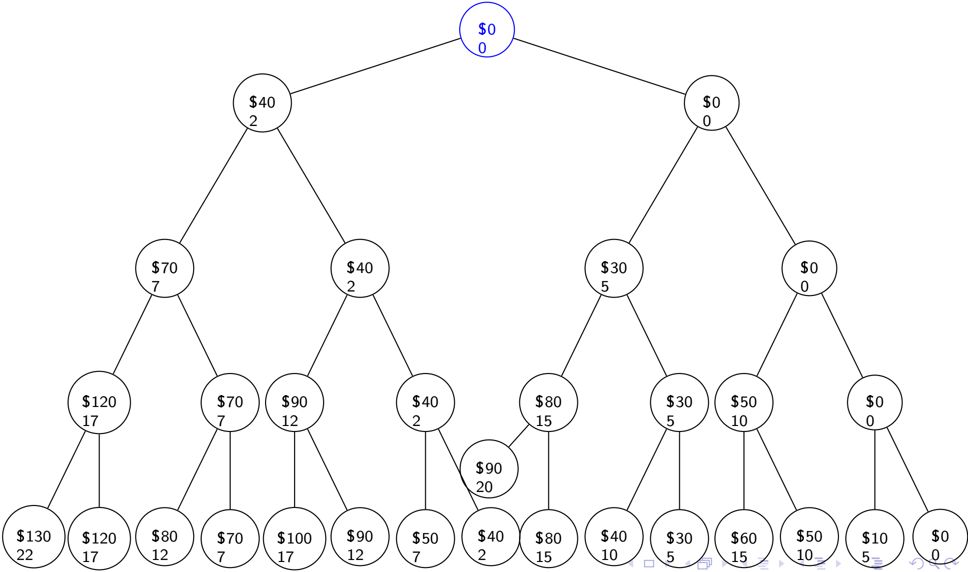
Example of **Depth-First Search (DFS)**



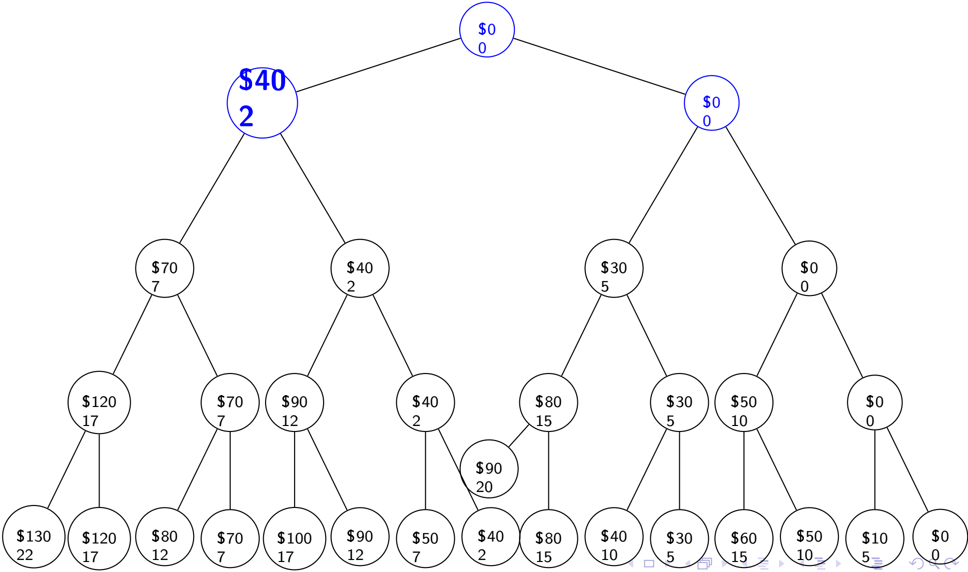
Example of **Depth-First Search (DFS)**



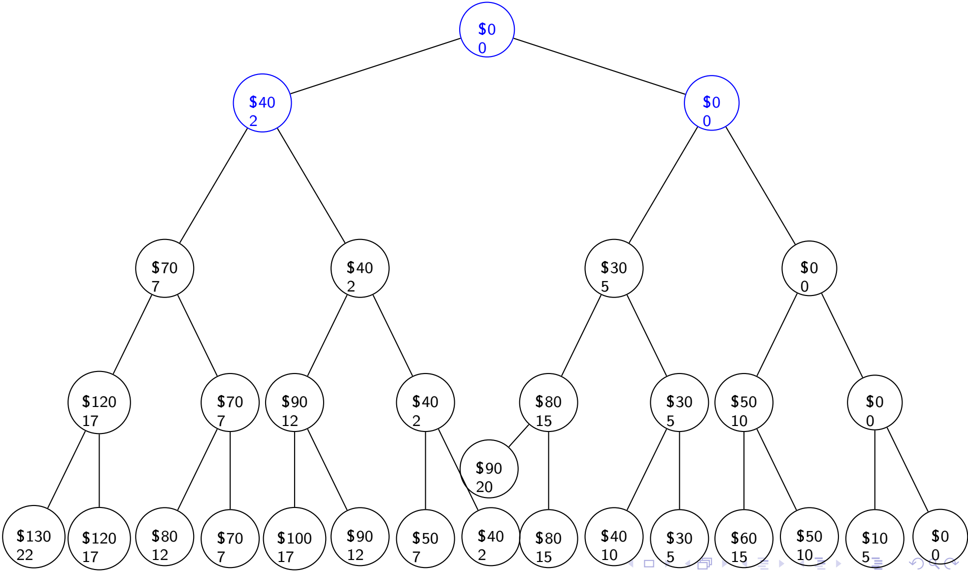
Example of **Depth-First Search (DFS)**



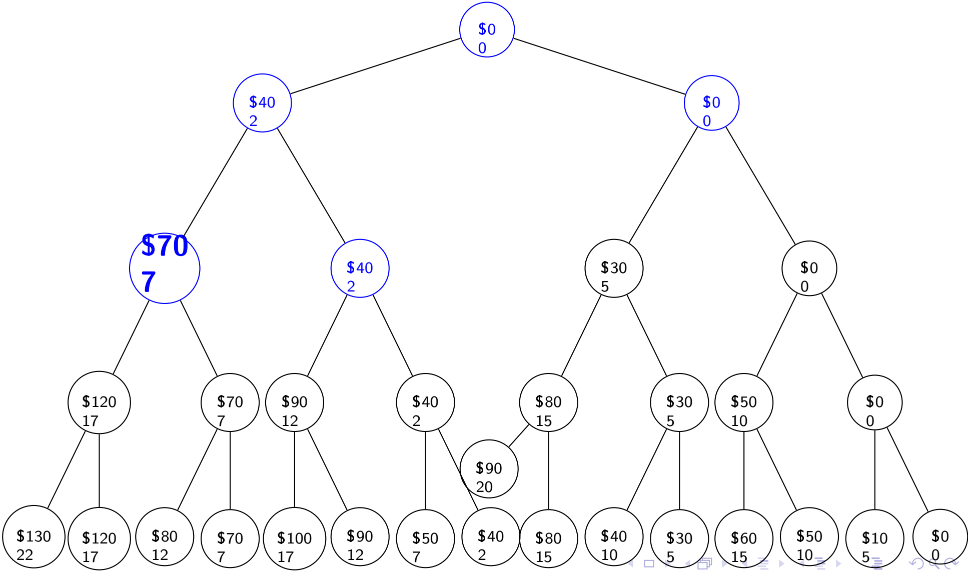
Example of **Depth-First Search (DFS)**



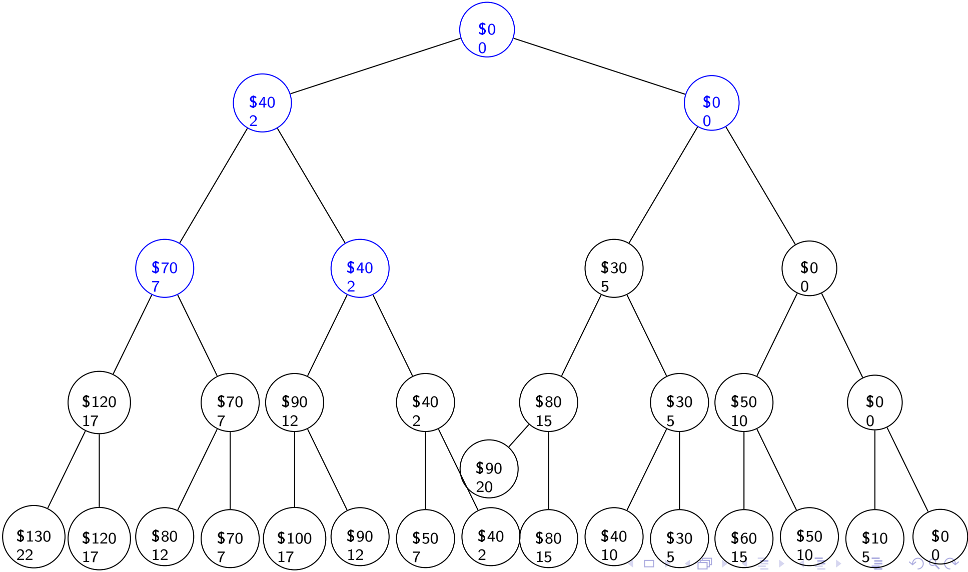
Example of **Depth-First Search (DFS)**



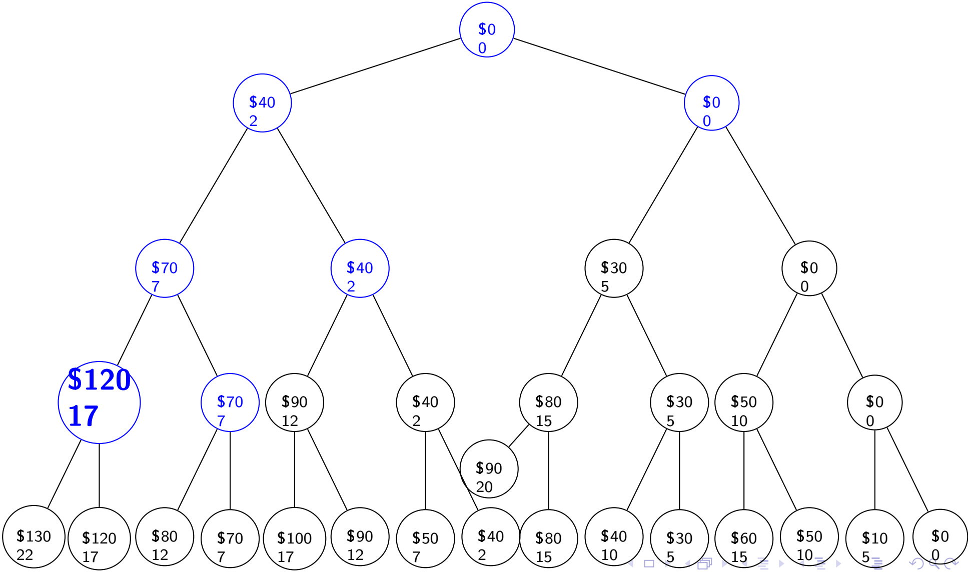
Example of **Depth-First Search (DFS)**



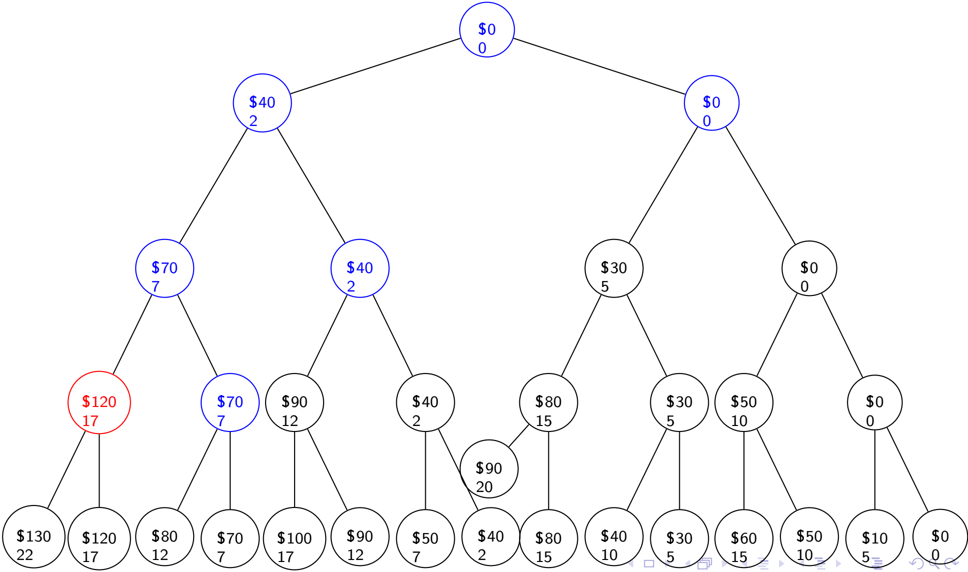
Example of **Depth-First Search (DFS)**



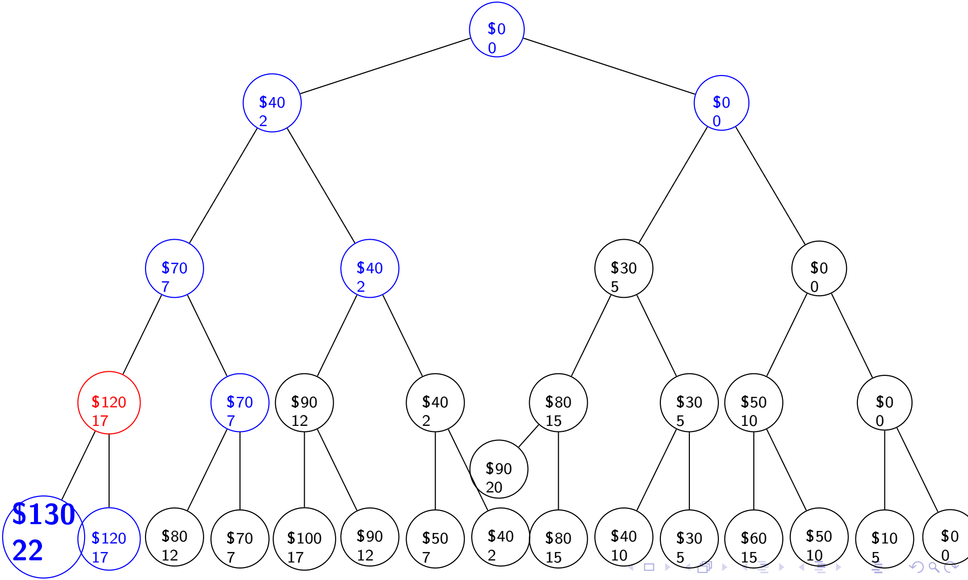
Example of **Depth-First Search (DFS)**



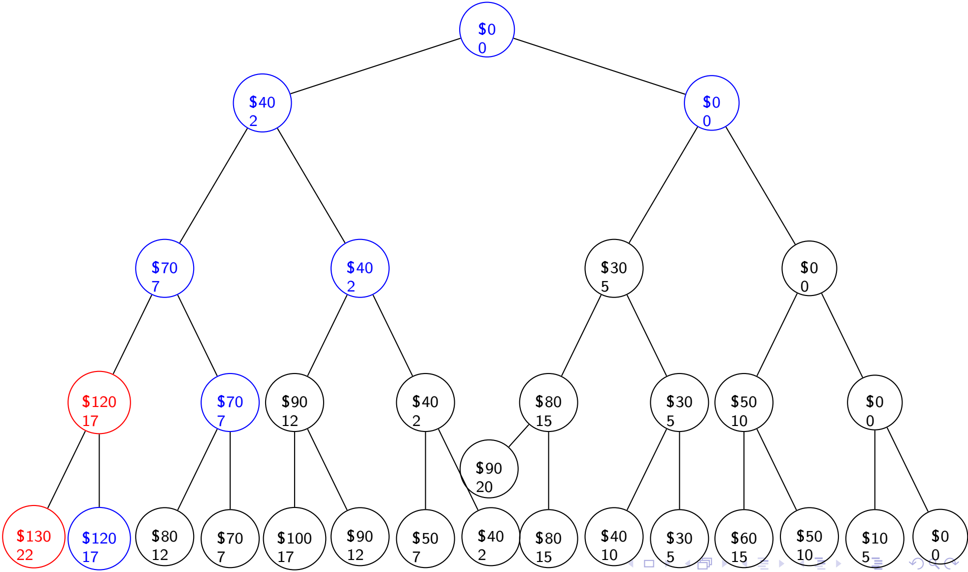
Example of **Depth-First Search (DFS)**



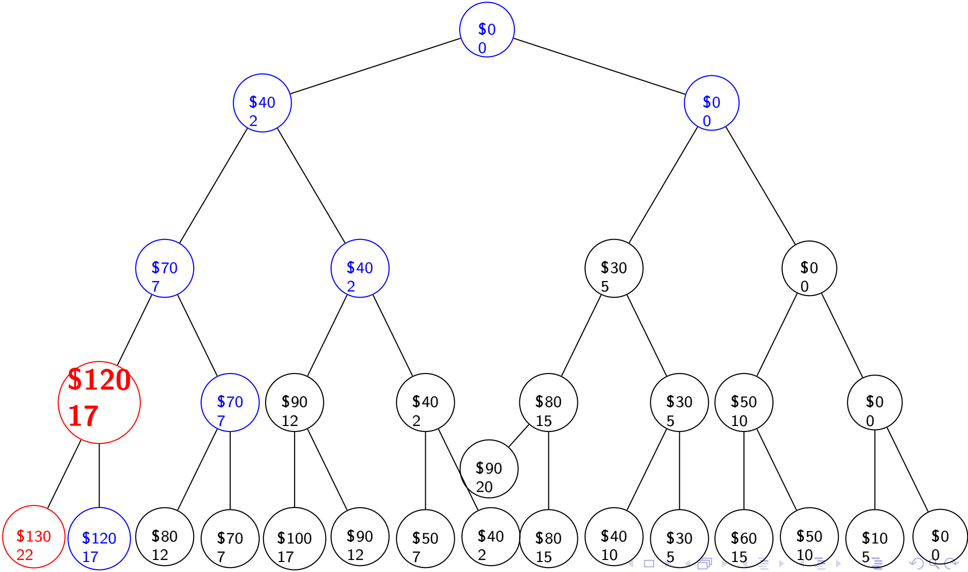
Example of **Depth-First Search (DFS)**



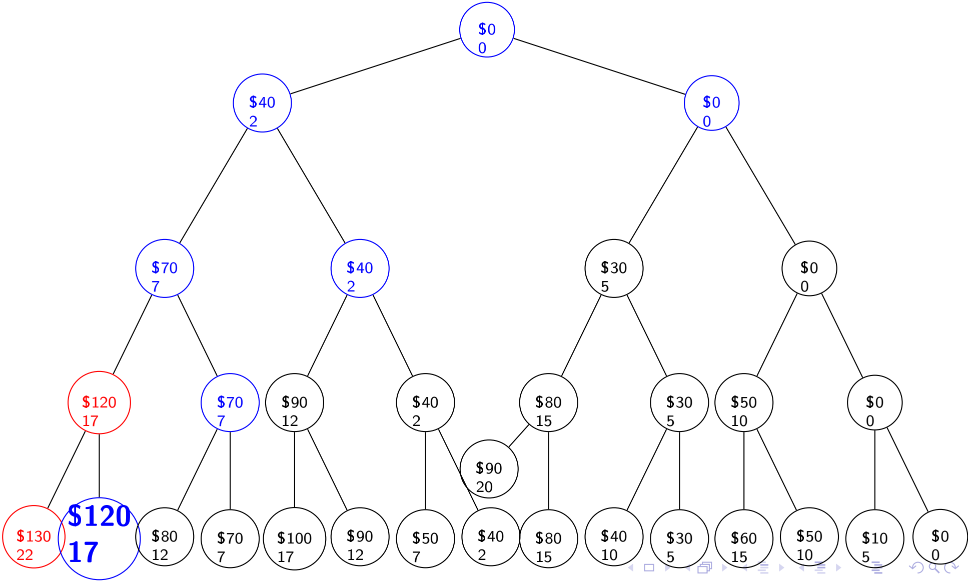
Example of **Depth-First Search (DFS)**



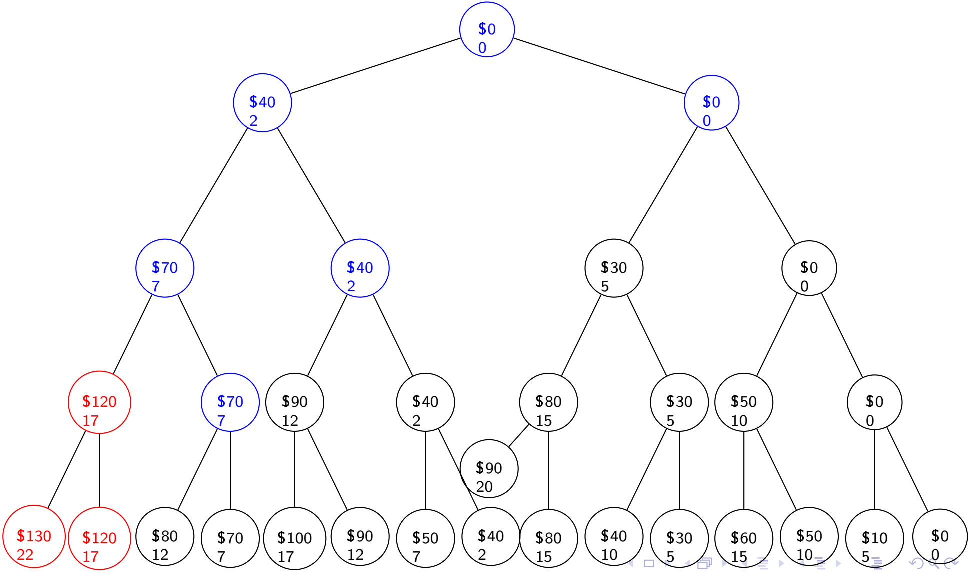
Example of **Depth-First Search (DFS)**



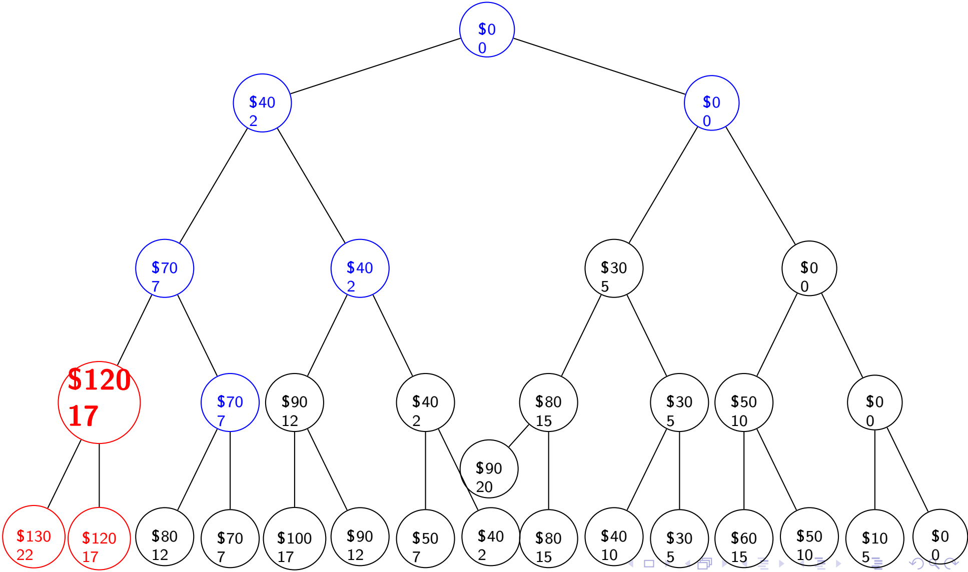
Example of **Depth-First Search (DFS)**



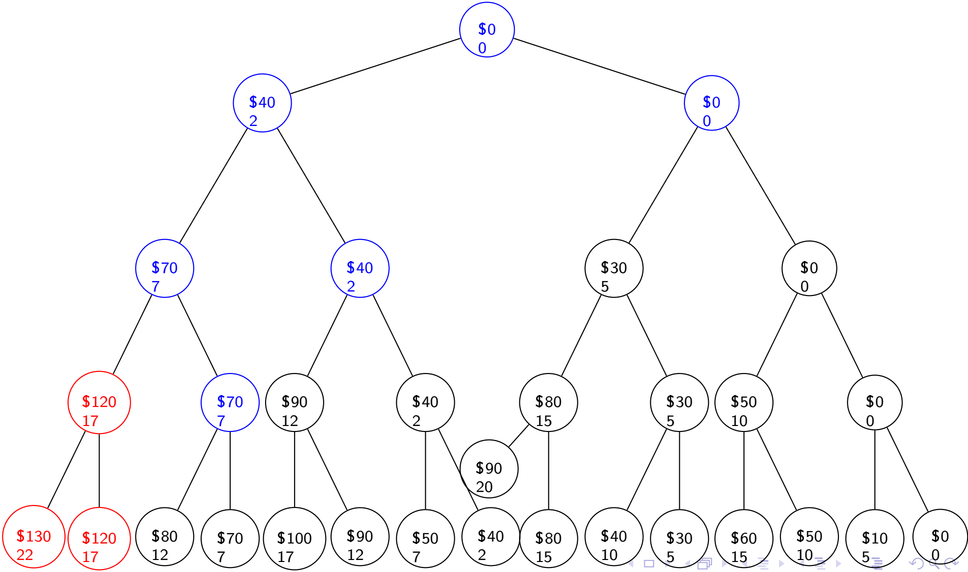
Example of **Depth-First Search (DFS)**



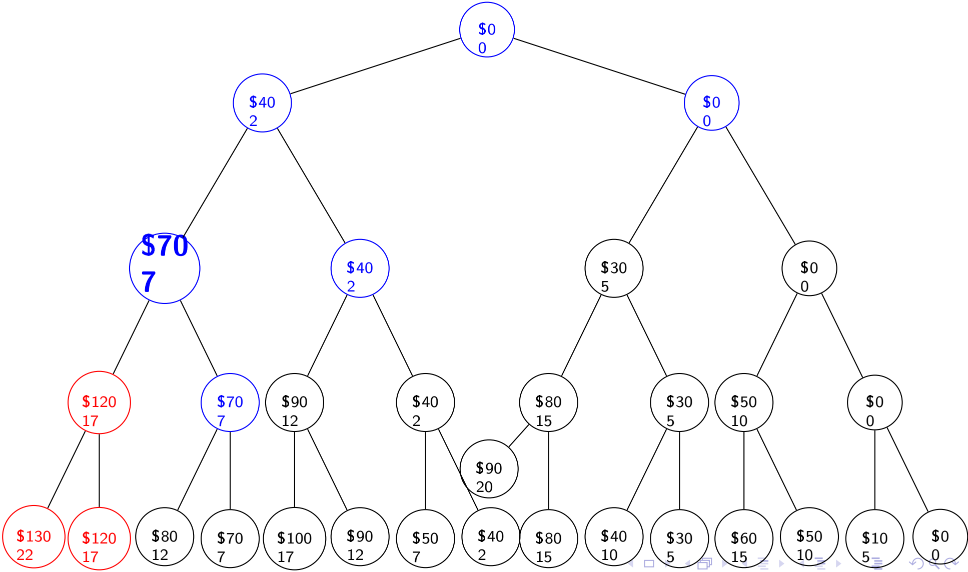
Example of **Depth-First Search (DFS)**



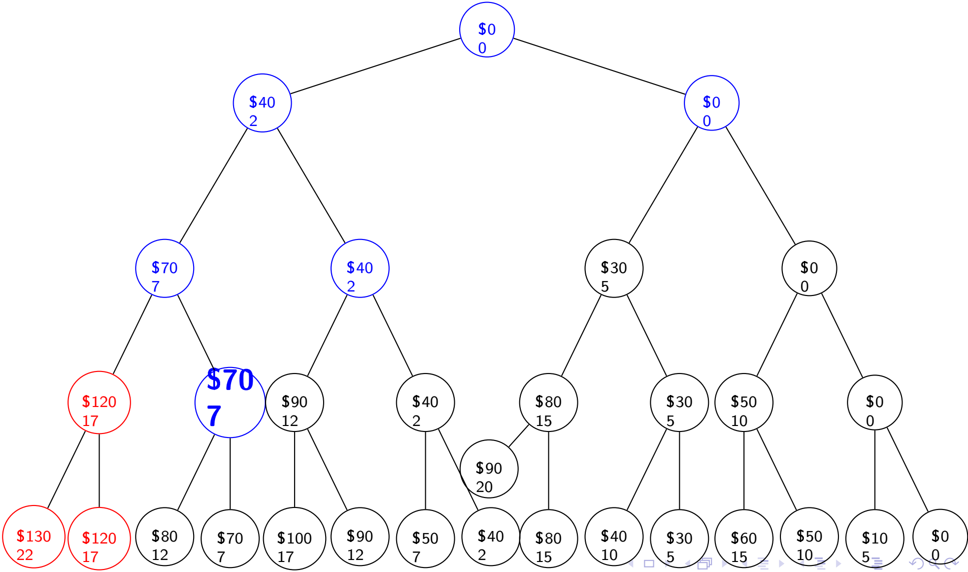
Example of **Depth-First Search (DFS)**



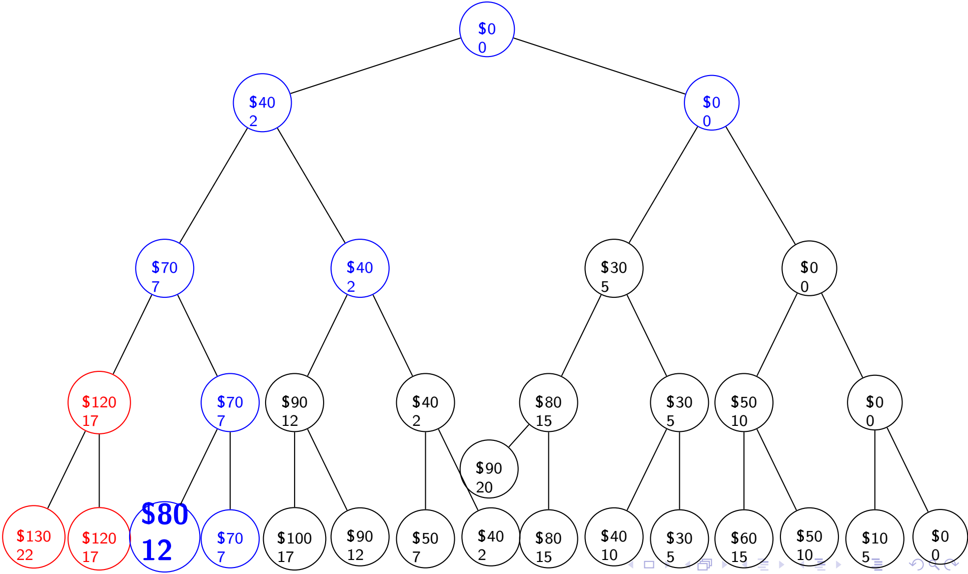
Example of **Depth-First Search (DFS)**



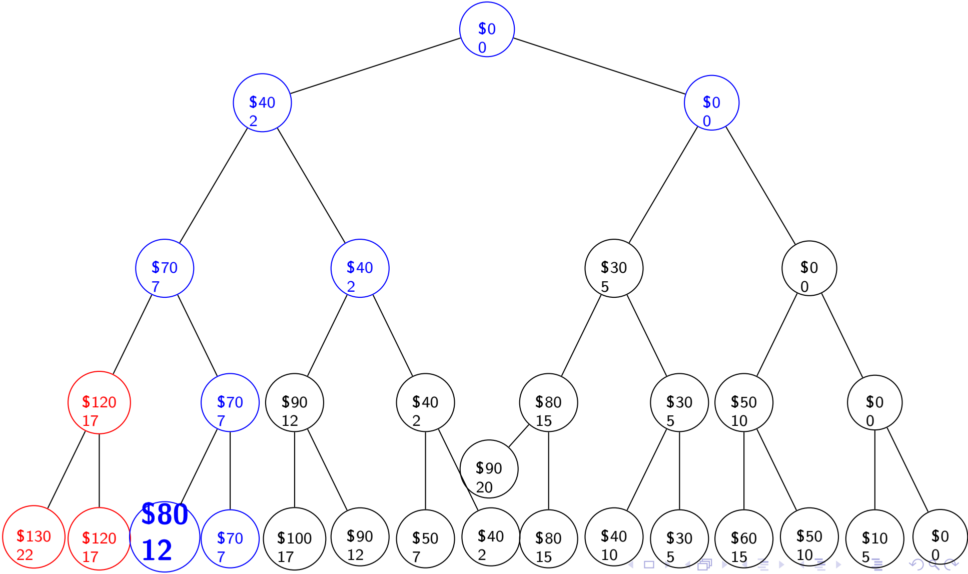
Example of **Depth-First Search (DFS)**



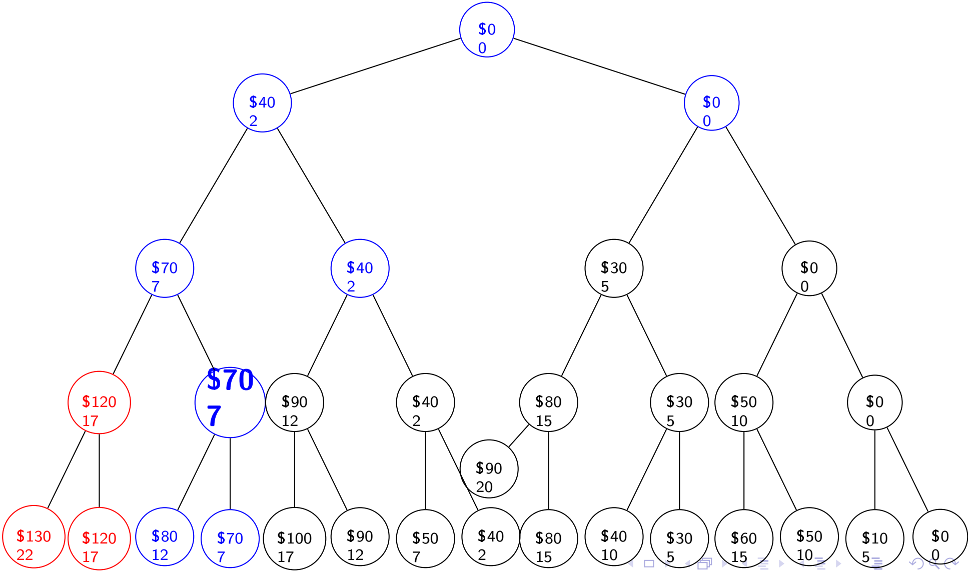
Example of **Depth-First Search (DFS)**



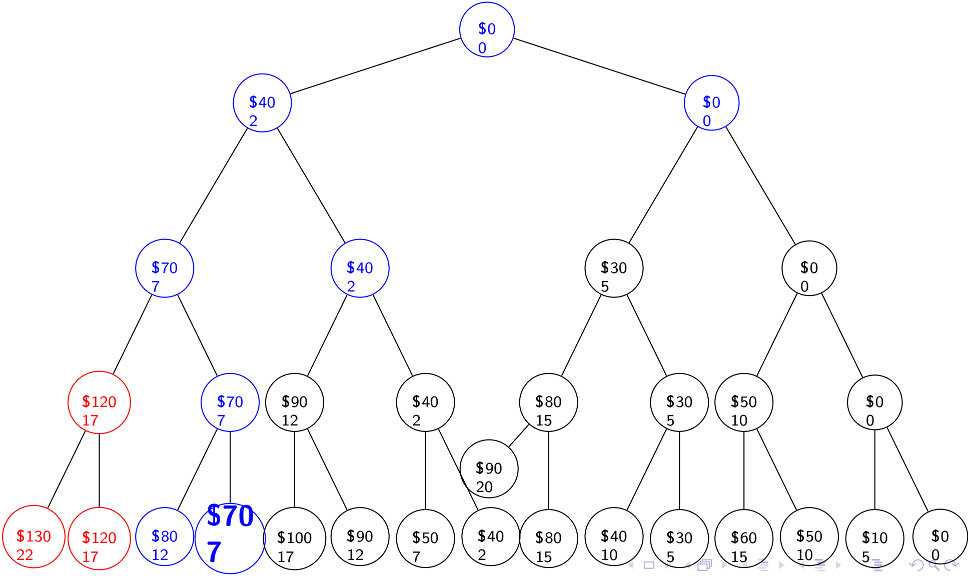
Example of **Depth-First Search (DFS)**



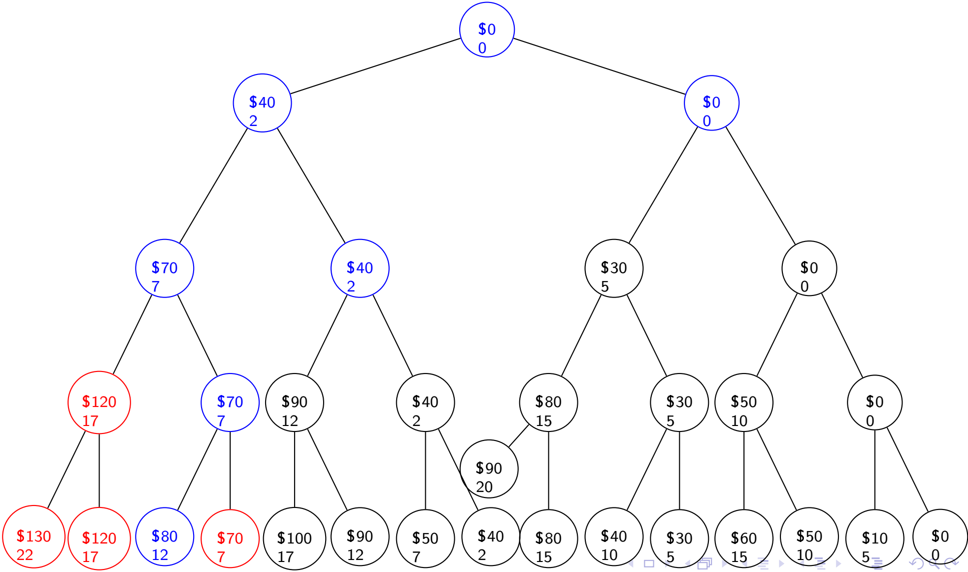
Example of **Depth-First Search (DFS)**



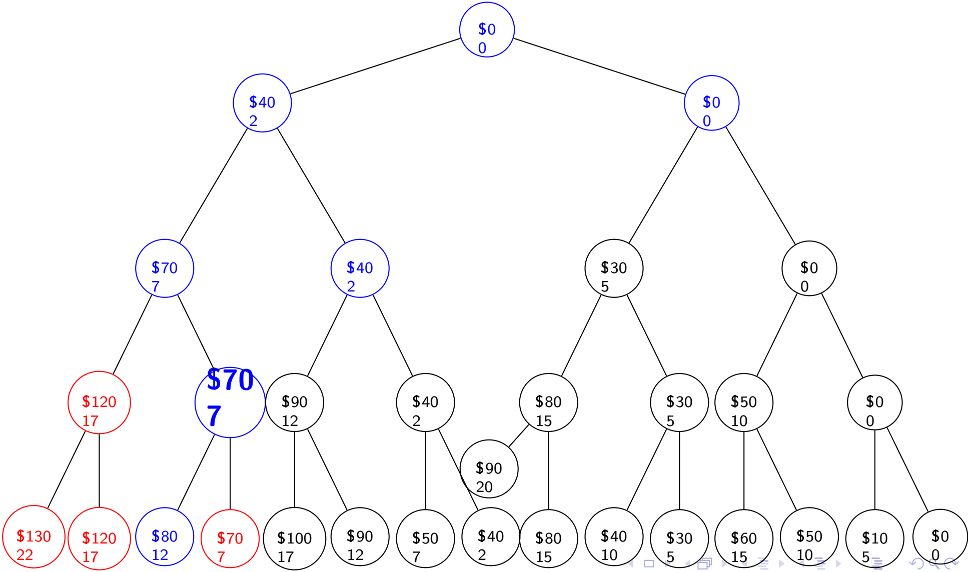
Example of **Depth-First Search (DFS)**



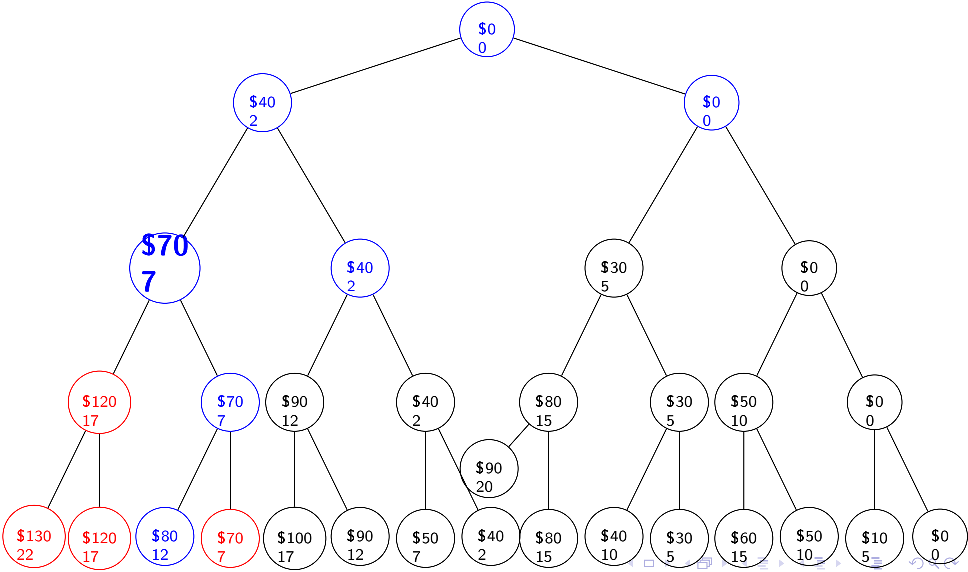
Example of **Depth-First Search (DFS)**



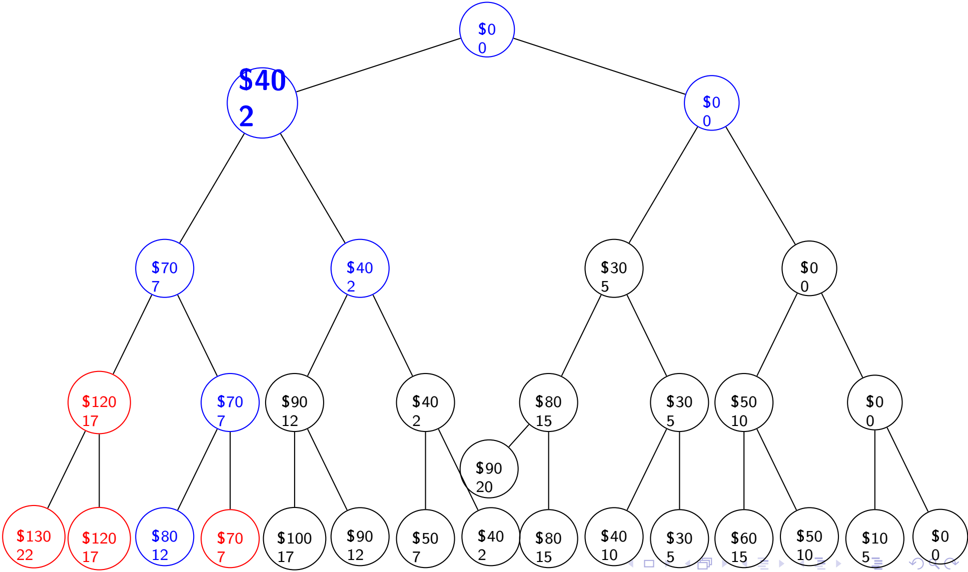
Example of **Depth-First Search (DFS)**



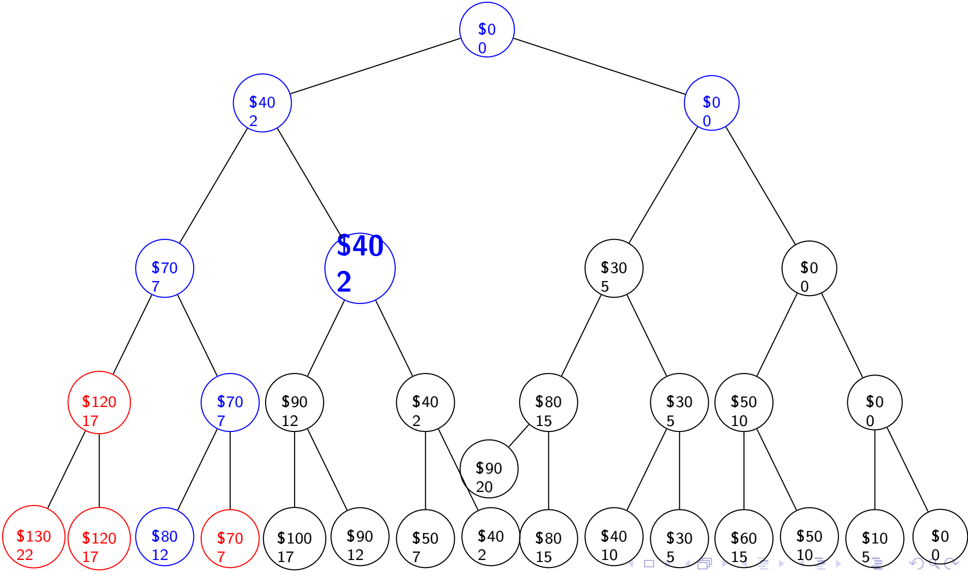
Example of **Depth-First Search (DFS)**



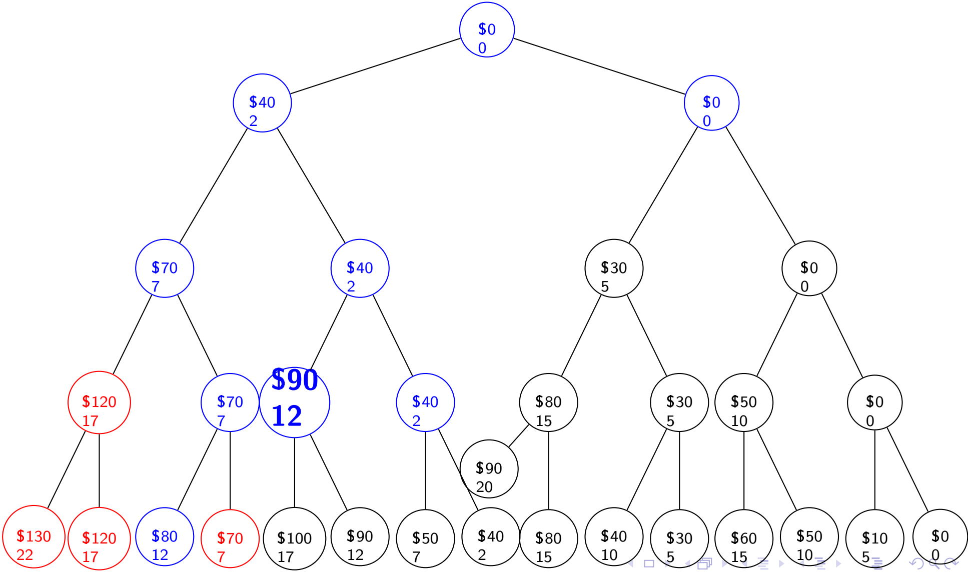
Example of **Depth-First Search (DFS)**



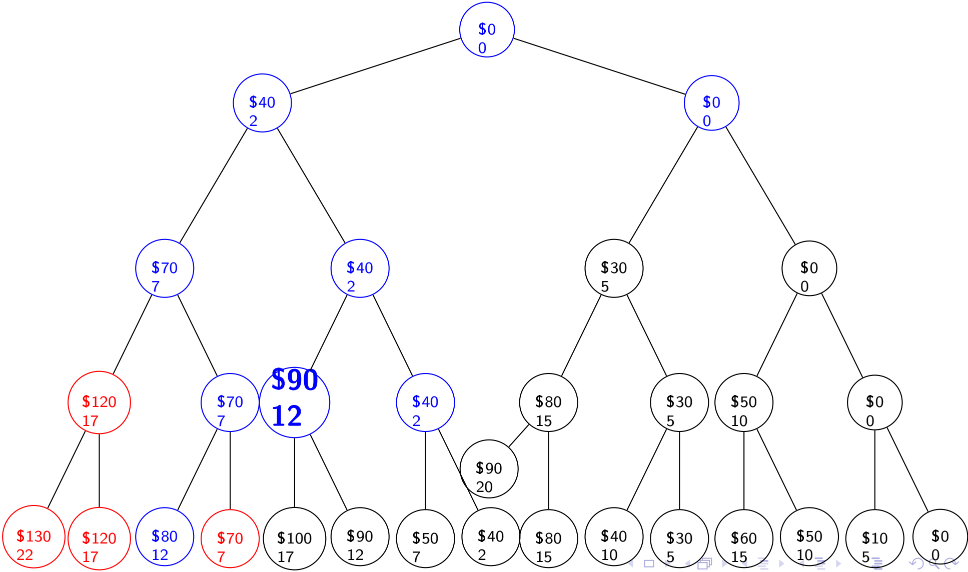
Example of **Depth-First Search (DFS)**



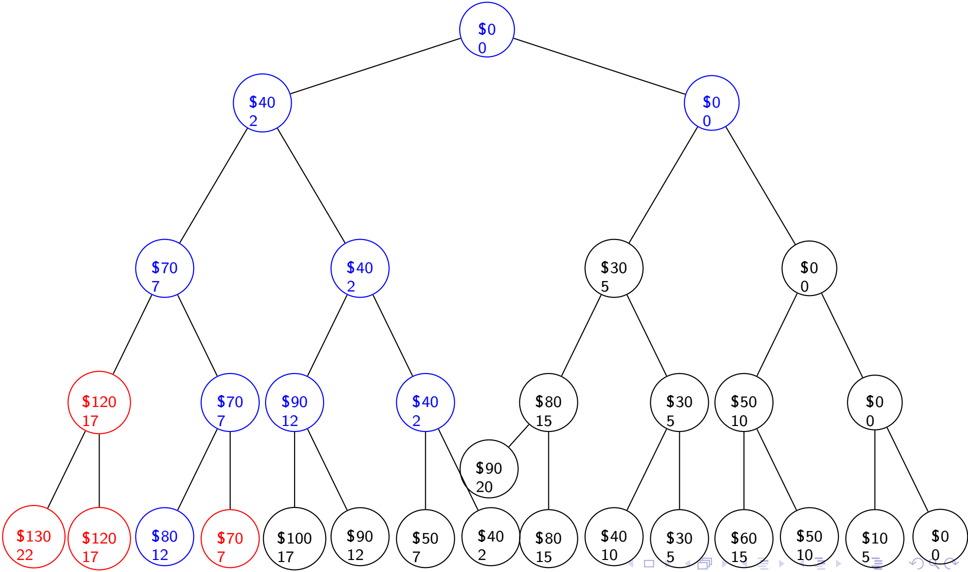
Example of **Depth-First Search (DFS)**



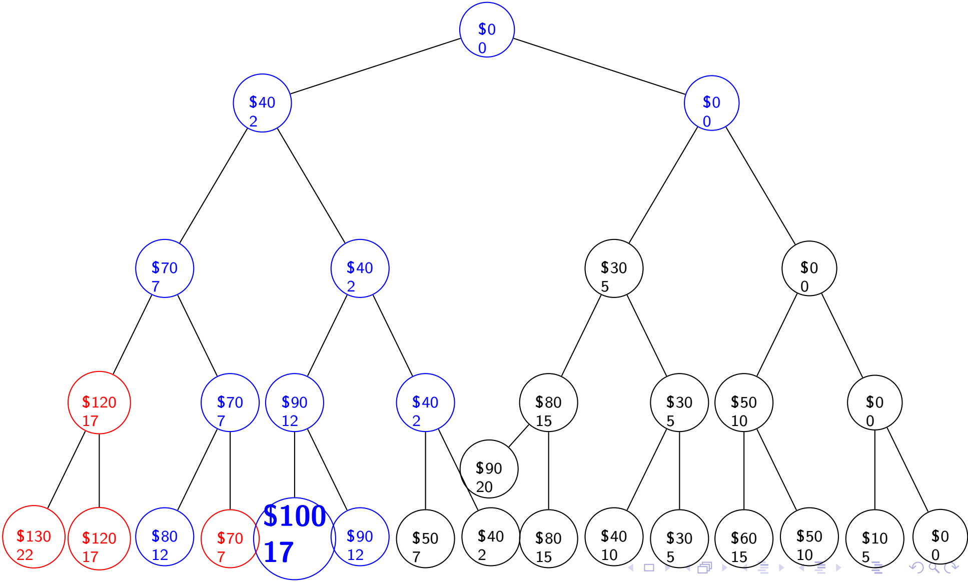
Example of **Depth-First Search (DFS)**



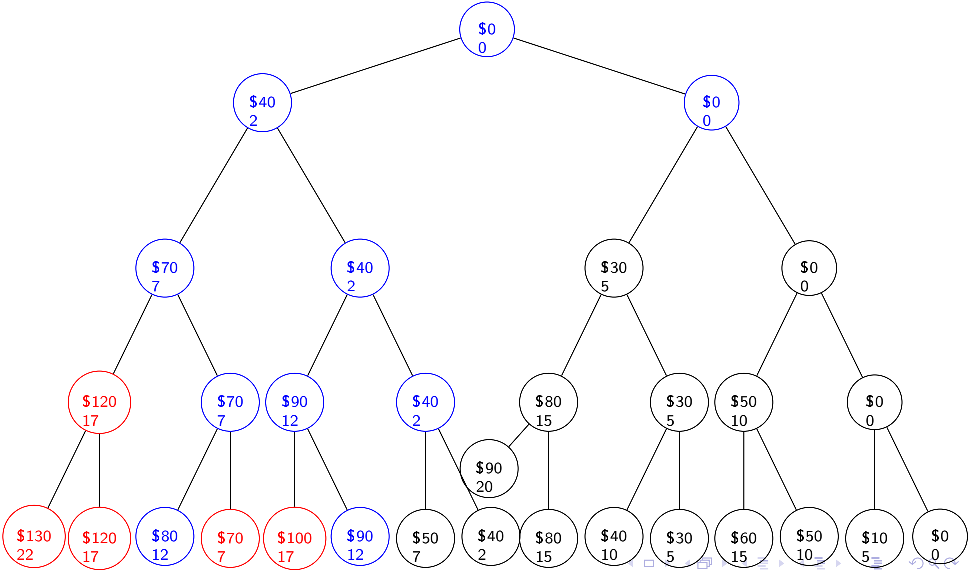
Example of **Depth-First Search (DFS)**



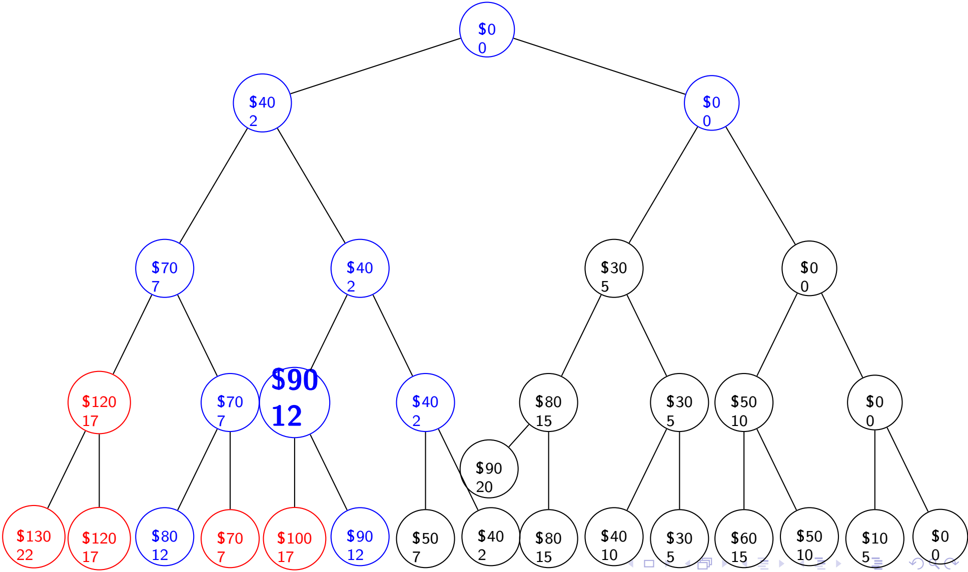
Example of **Depth-First Search (DFS)**



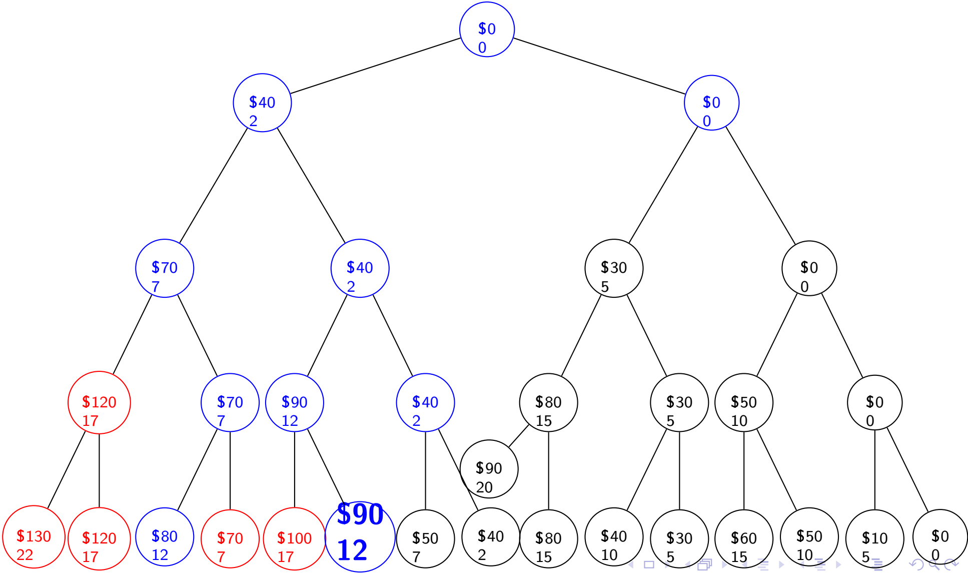
Example of **Depth-First Search (DFS)**



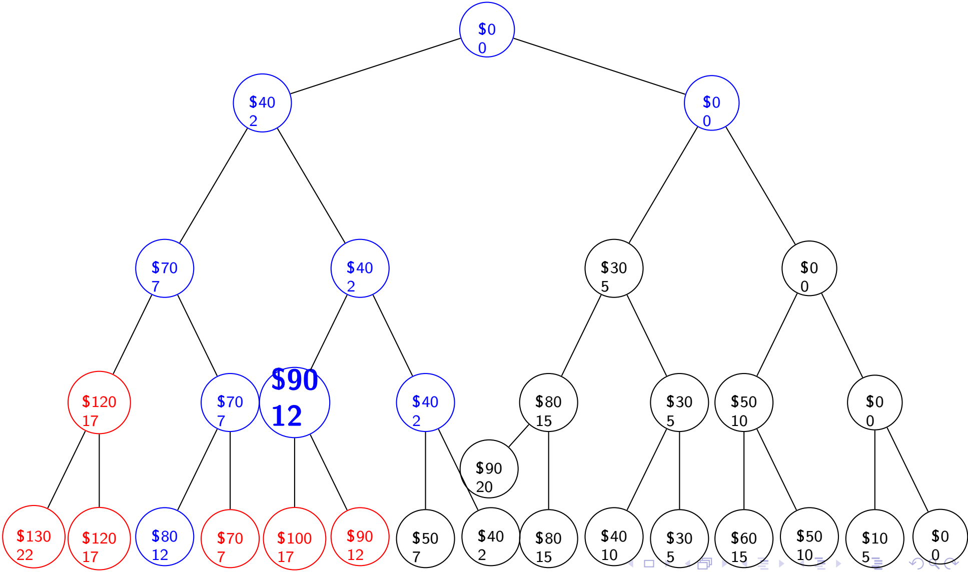
Example of **Depth-First Search (DFS)**



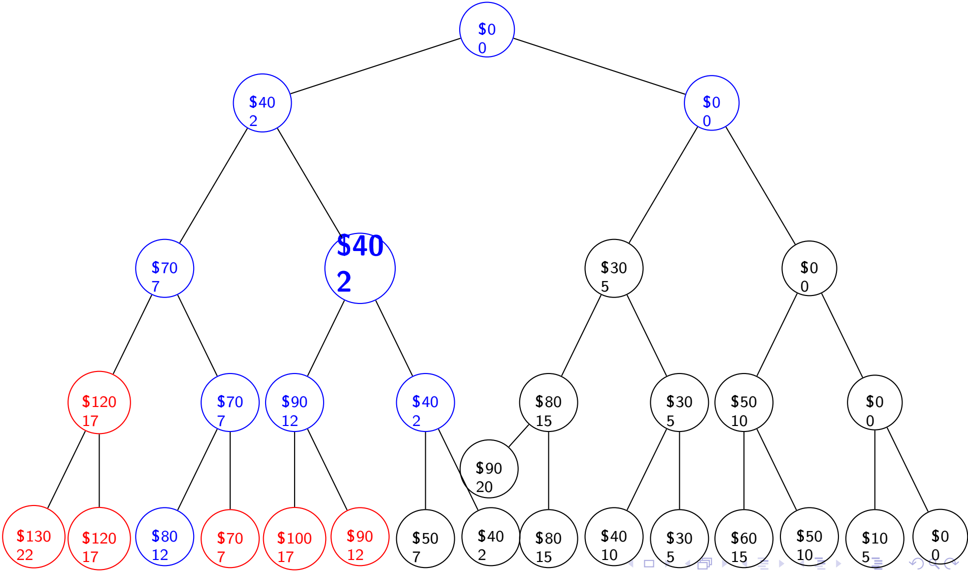
Example of **Depth-First Search (DFS)**



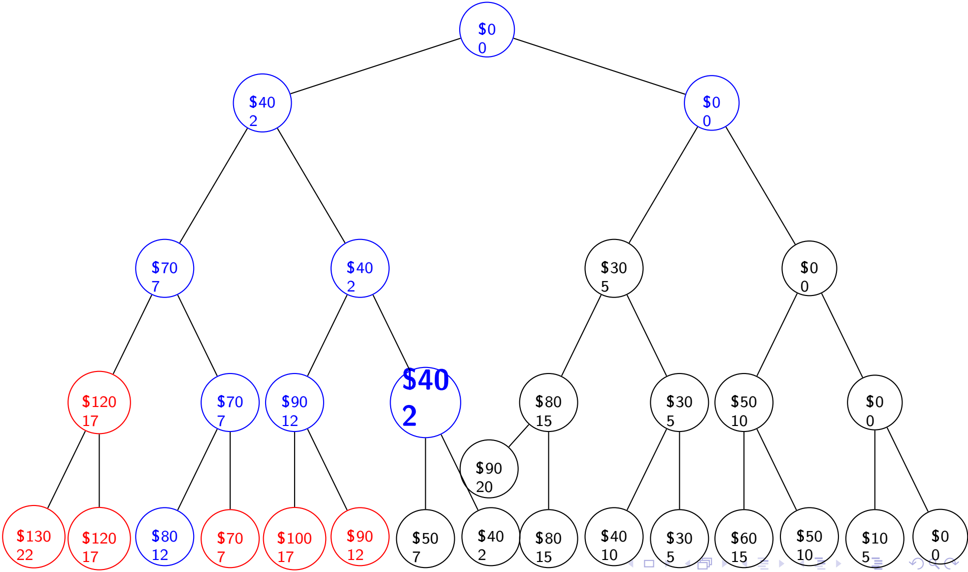
Example of **Depth-First Search (DFS)**



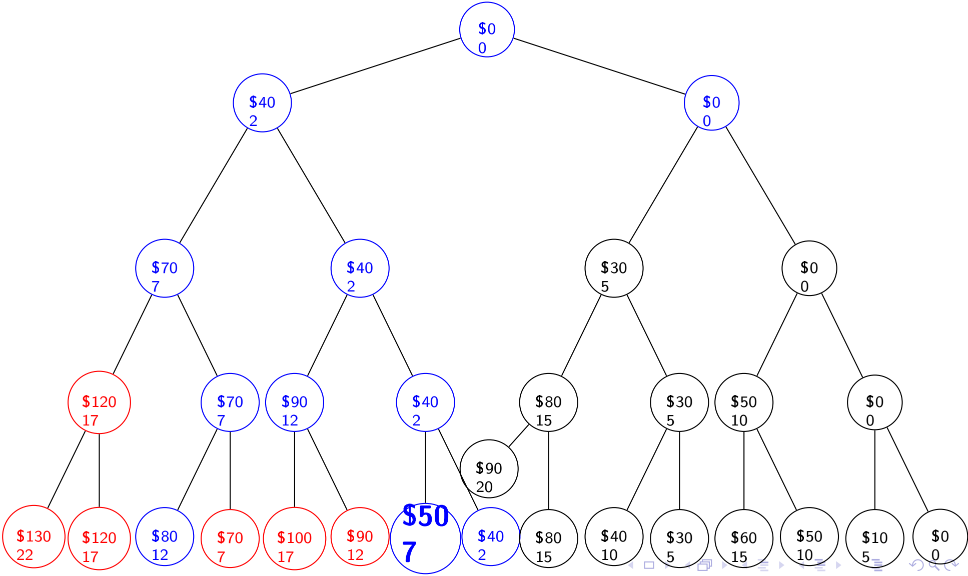
Example of **Depth-First Search (DFS)**



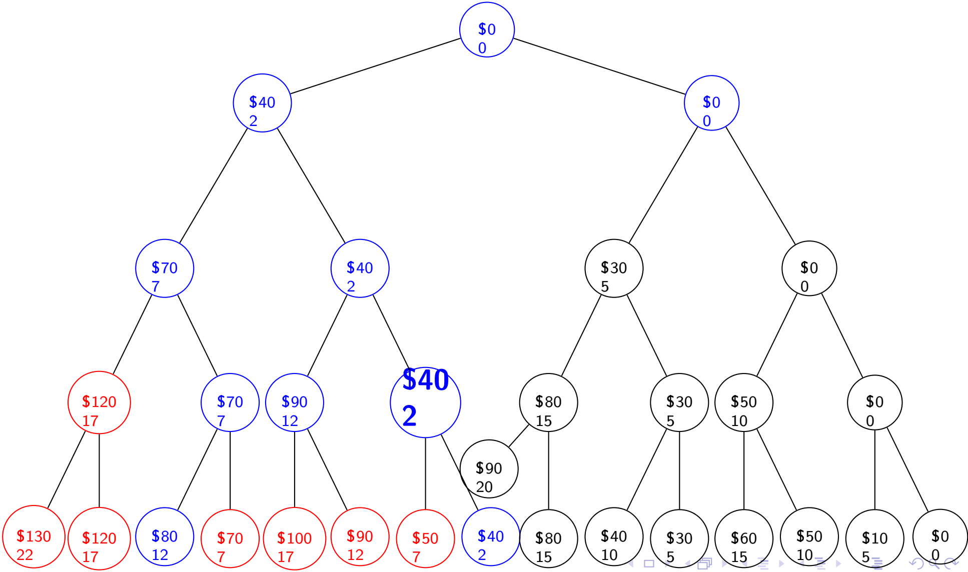
Example of **Depth-First Search (DFS)**



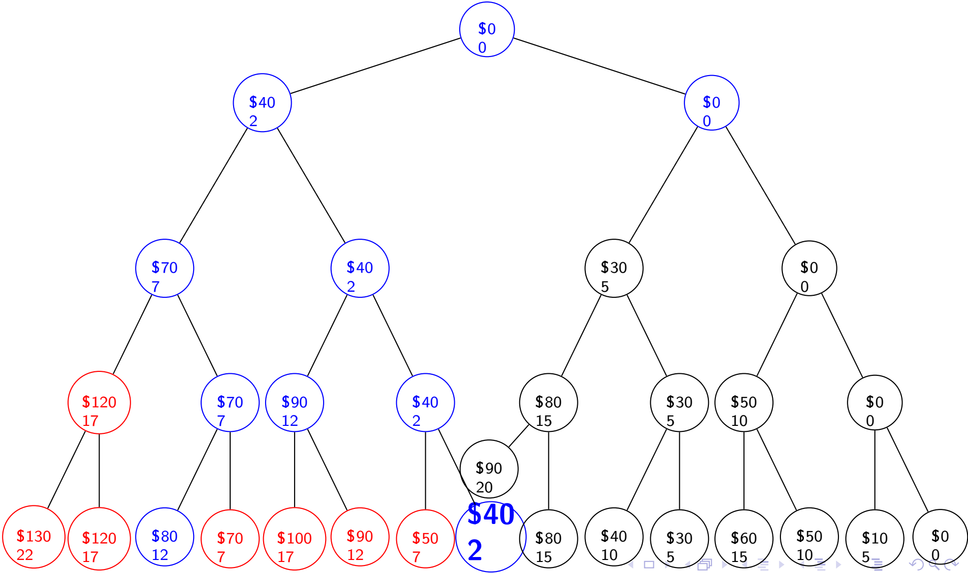
Example of **Depth-First Search (DFS)**



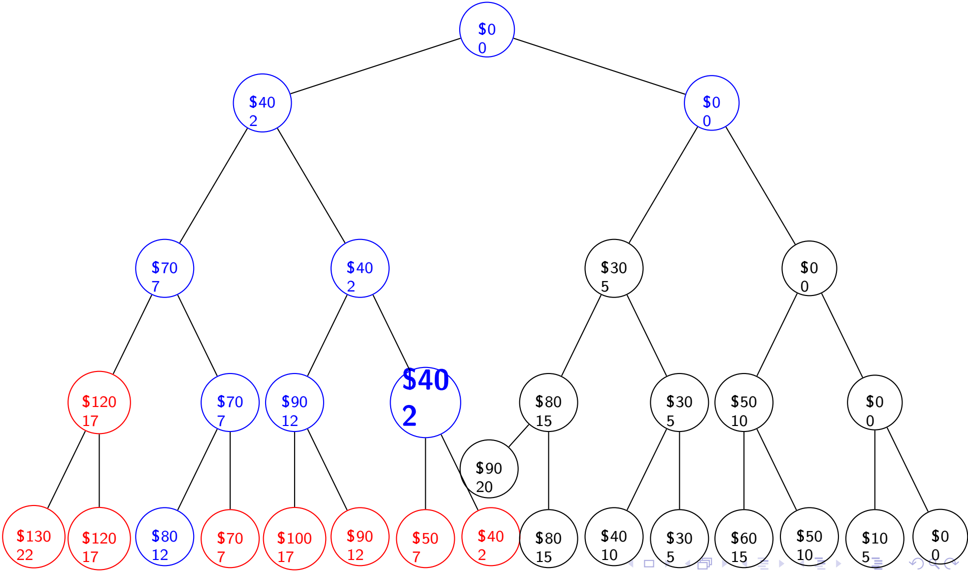
Example of **Depth-First Search (DFS)**



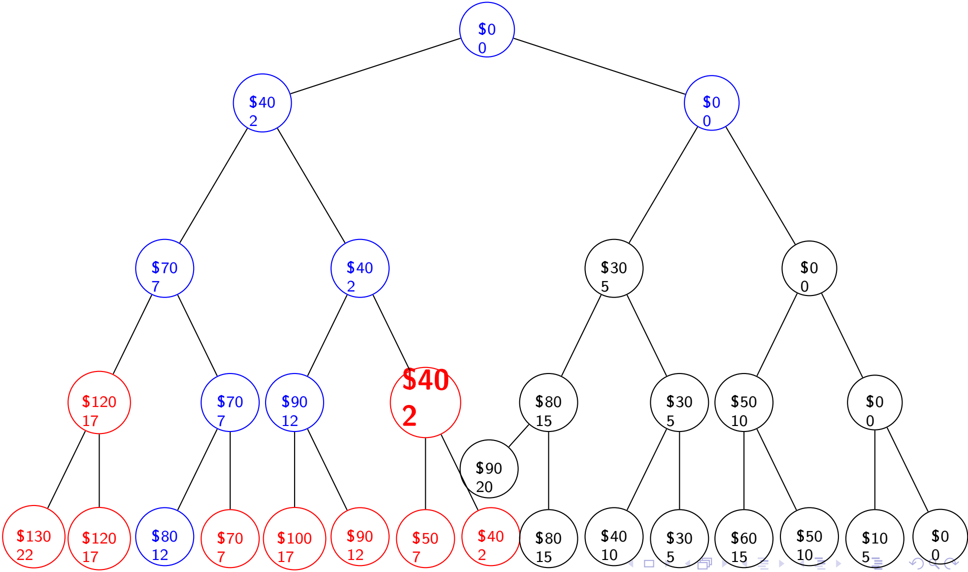
Example of **Depth-First Search (DFS)**



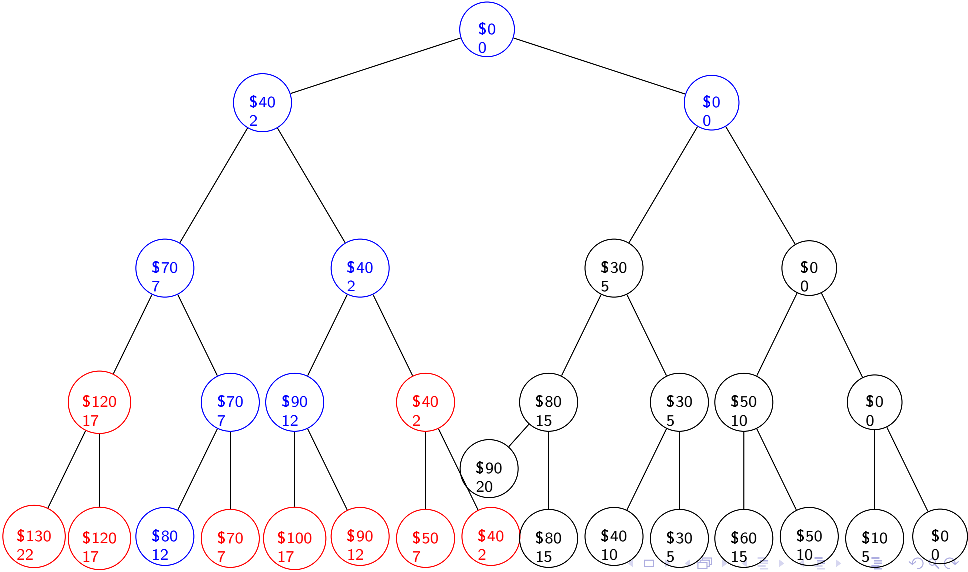
Example of **Depth-First Search (DFS)**



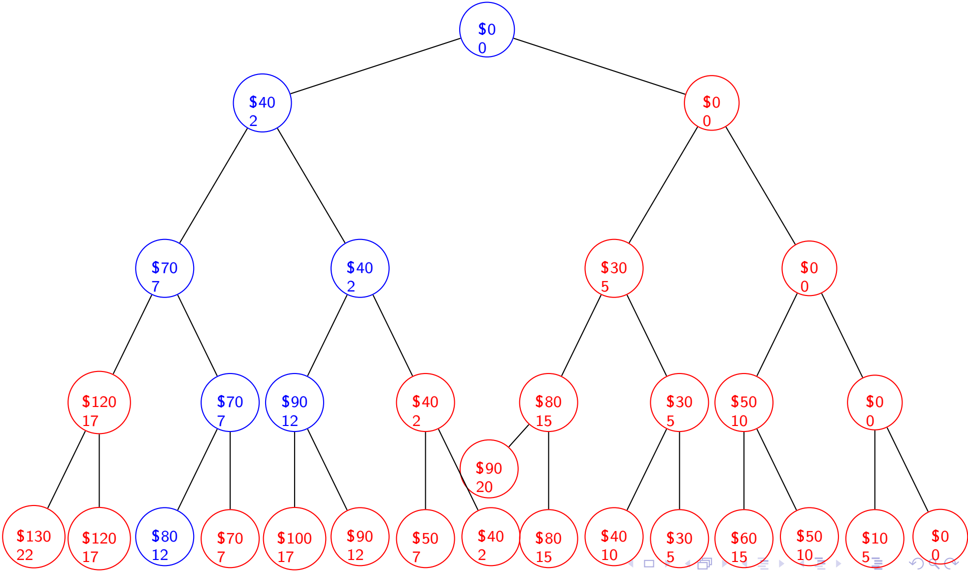
Example of **Depth-First Search (DFS)**



Example of **Depth-First Search (DFS)**



Example of **Depth-First Search (DFS)**



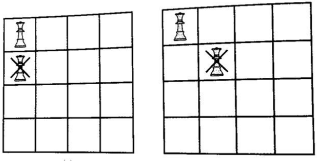
N-Queen Problem

N-Queen Problem

- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other

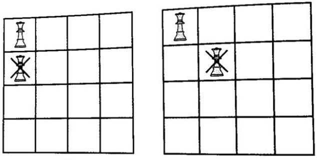
N-Queen Problem

- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other



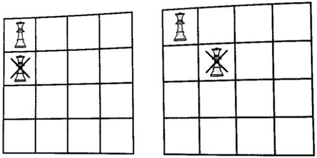
N-Queen Problem

- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal



N-Queen Problem

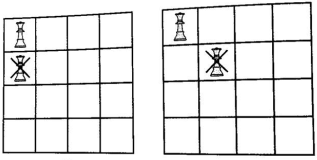
- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal
- Sequence: n positions where queens are placed



N-Queen Problem

- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal
- Sequence: n positions where queens are placed

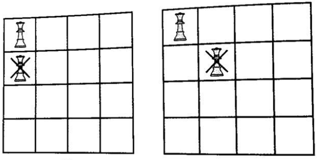
- Set: n^2 positions on the board



N-Queen Problem

- Goal: position n queens on a $n \times n$ board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal
- Sequence: n positions where queens are placed

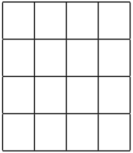
- Set: n^2 positions on the board
- Criterion: no two queens threaten each other



4-Queen Problem (DFS)

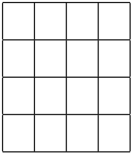
4-Queen Problem (DFS)

- Consider a 4-queen problem for simplicity



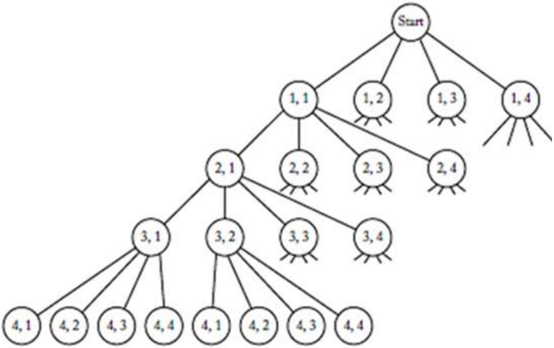
4-Queen Problem (DFS)

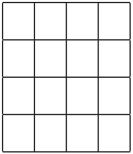
- Consider a 4-queen problem for simplicity
- Assign each queen a different row



4-Queen Problem (DFS)

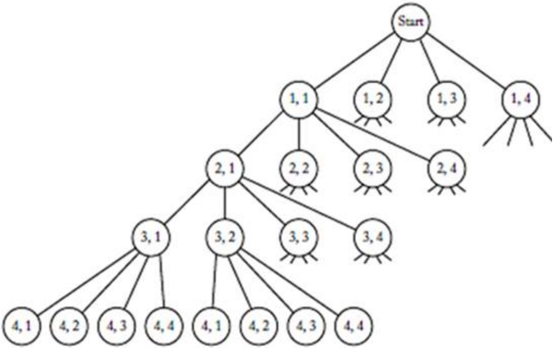
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions

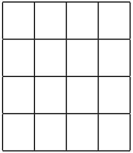




4-Queen Problem (DFS)

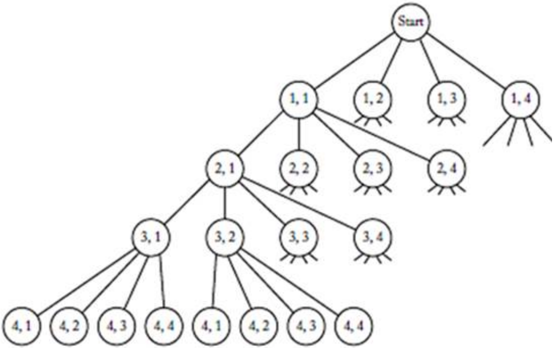
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions
- We can represent this idea in a 'State space tree'

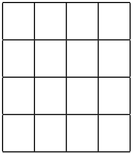




4-Queen Problem (DFS)

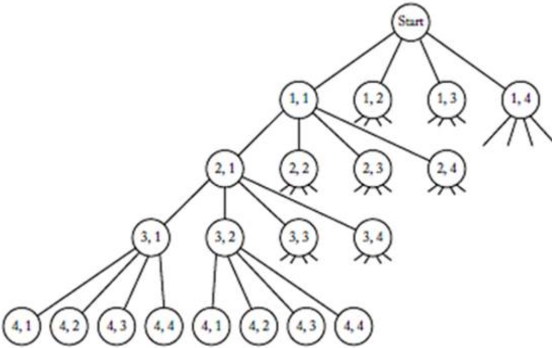
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions
- We can represent this idea in a 'State space tree'
- What would be the total number of nodes in this 'State space tree'

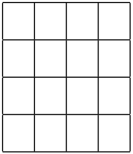




4-Queen Problem (DFS)

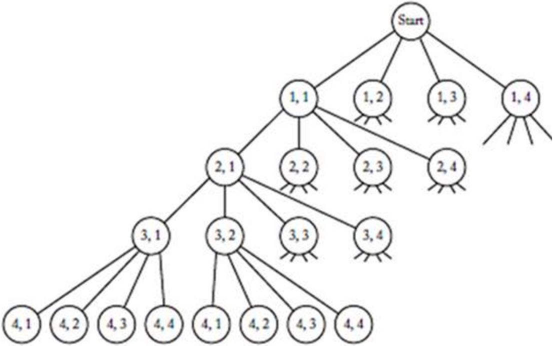
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions
- We can represent this idea in a 'State space tree'
- What would be the total number of nodes in this 'State space tree'
  $1 + 4 + 4^2 + 4^3 + 4^4$

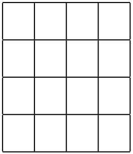




4-Queen Problem (DFS)

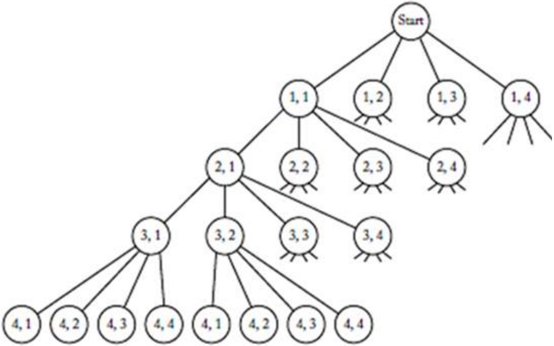
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions
- We can represent this idea in a 'State space tree'
- What would be the total number of nodes in this 'State space tree'
  $1 + 4 + 4^2 + 4^3 + 4^4$
- What would be the total number of sequence in this 'State space tree'?

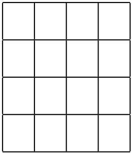




4-Queen Problem (DFS)

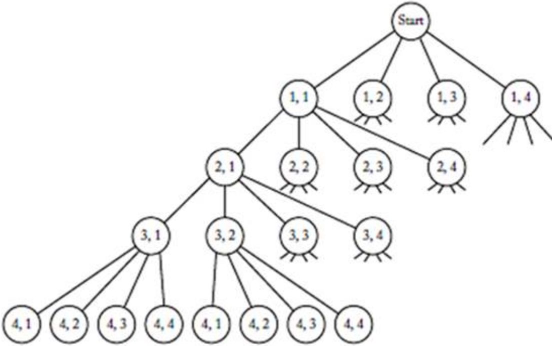
- Consider a 4-queen problem for simplicity
- Assign each queen a different row
- Check which column combinations yield solutions
- We can represent this idea in a 'State space tree'
- What would be the total number of nodes in this 'State space tree'
  $1 + 4 + 4^2 + 4^3 + 4^4$
- What would be the total number of sequence in this 'State space tree'? 4^4



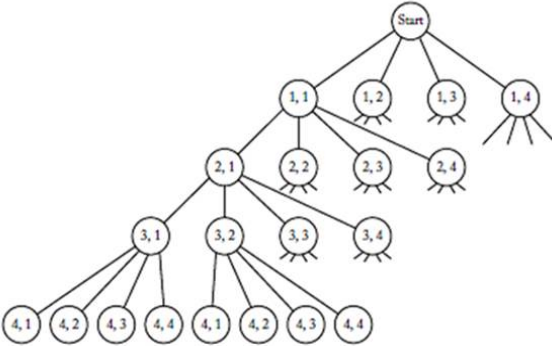


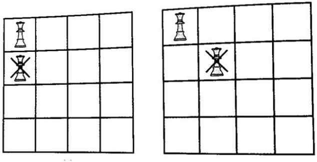
4-Queens problem (improving from DFS)

4-Queens problem (improving from DFS)

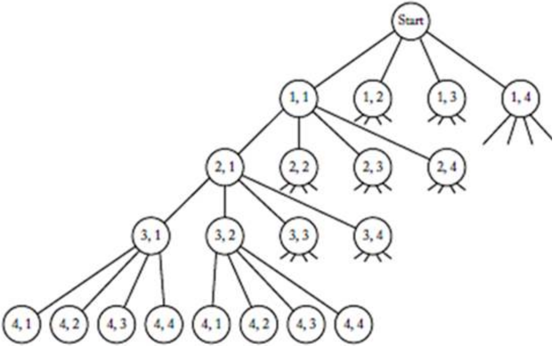


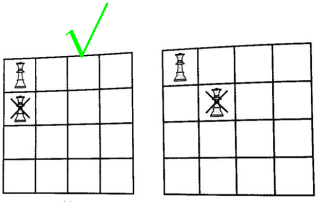
4-Queens problem (improving from DFS)



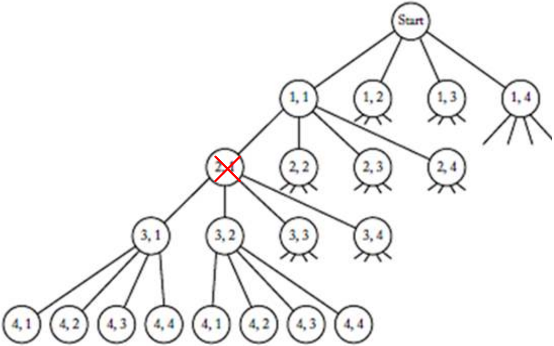


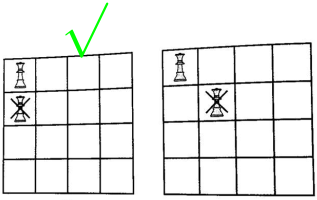
4-Queens problem (improving from DFS)





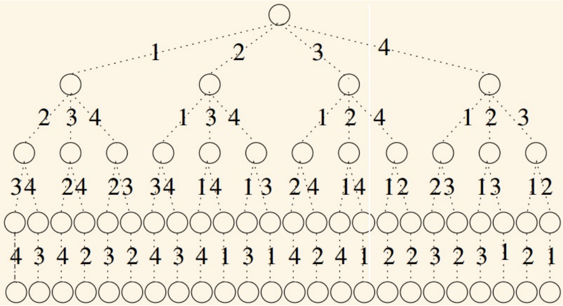
4-Queens problem (improving from DFS)

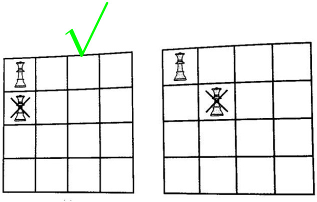




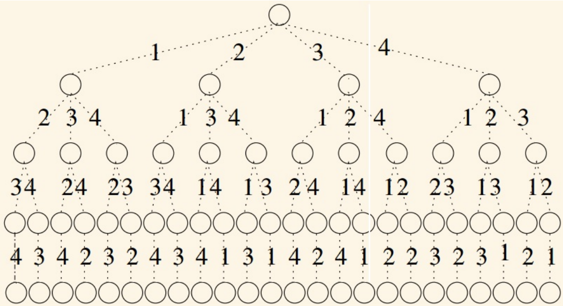
4-Queens problem (improving from DFS) (2)

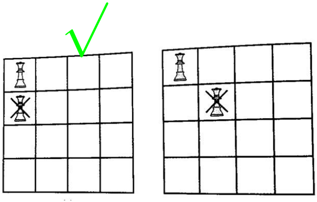
4-Queens problem (improving from DFS) (2)





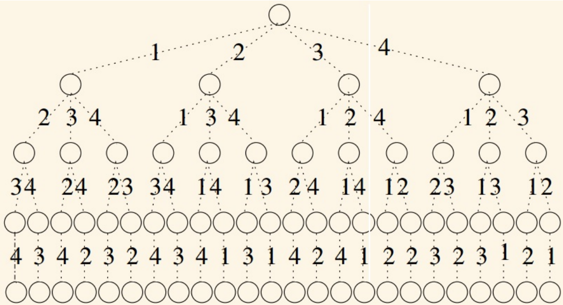
4-Queens problem (improving from DFS) (2)

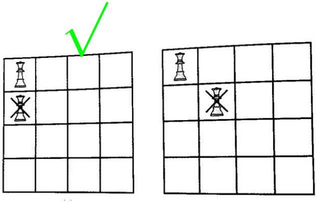




- What would be the total number of sequence in this 'State space tree'?

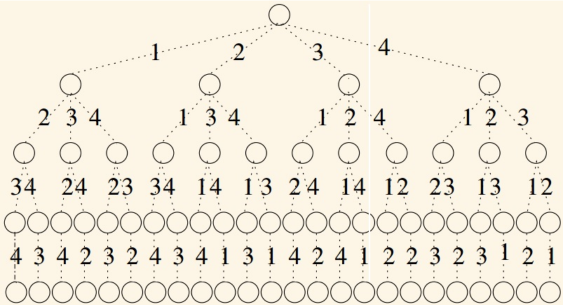
4-Queens problem (improving from DFS) (2)

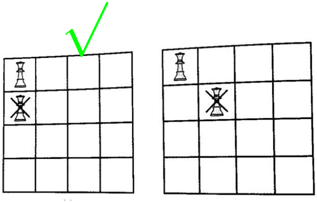




- What would be the total number of sequence in this 'State space tree'? $4 \times 3 \times 2 \times 1$

4-Queens problem (improving from DFS) (2)

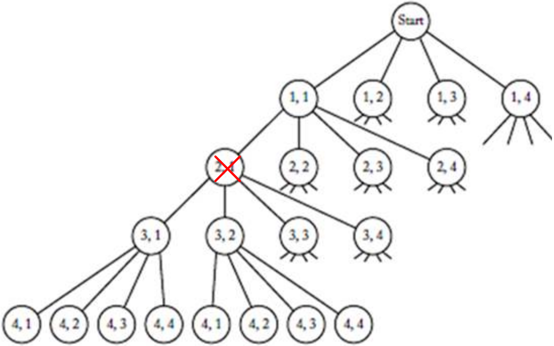


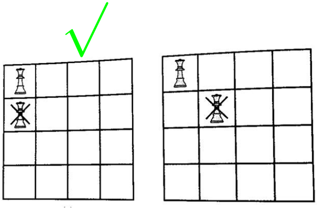


- What would be the total number of sequence in this 'State space tree'?
  $4 \times 3 \times 2 \times 1 \;= 4!$

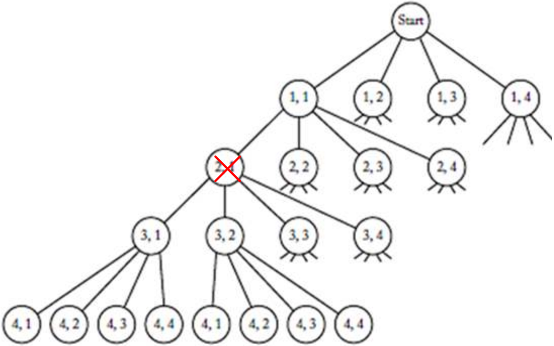
4-Queens problem (improving from DFS) (3)

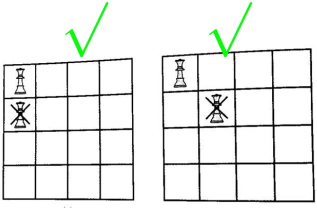
4-Queens problem (improving from DFS) (3)



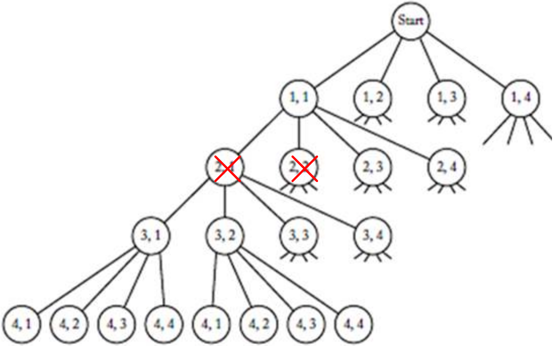


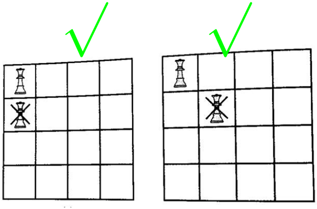
4-Queens problem (improving from DFS) (3)





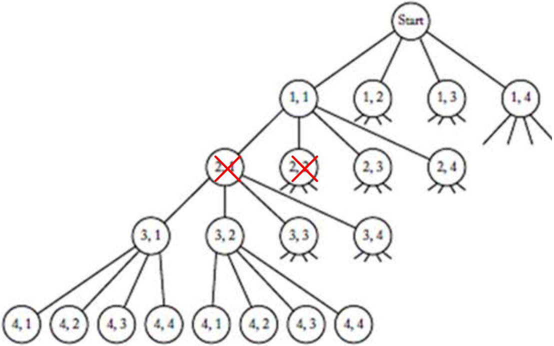
4-Queens problem (improving from DFS) (3)

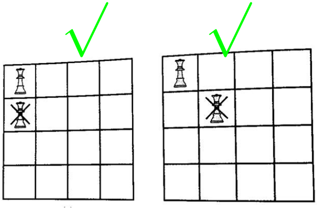




4-Queens problem (improving from DFS) (3)

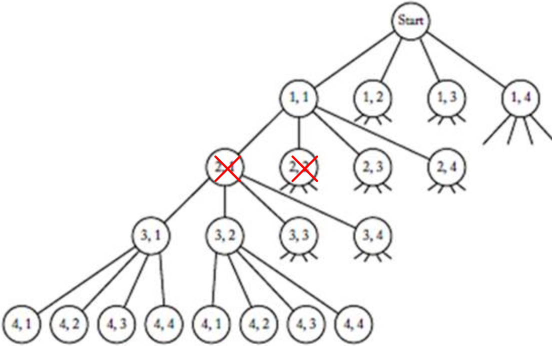
- After realizing a path is not promising

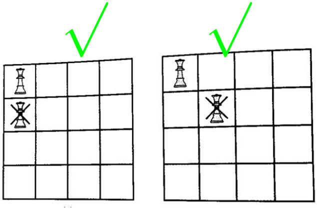




4-Queens problem (improving from DFS) (3)

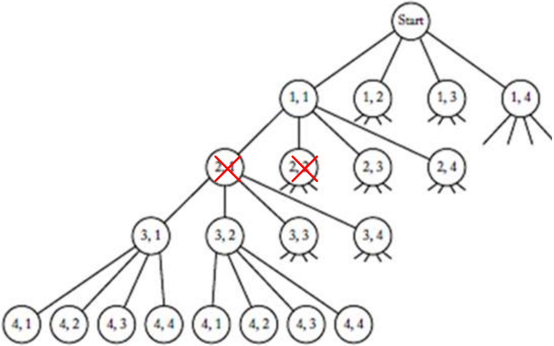
- After realizing a path is not promising
  - Backtrack to the parent

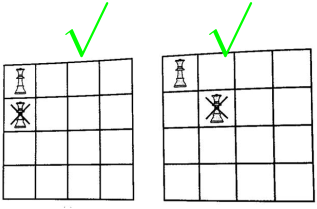




4-Queens problem (improving from DFS) (3)

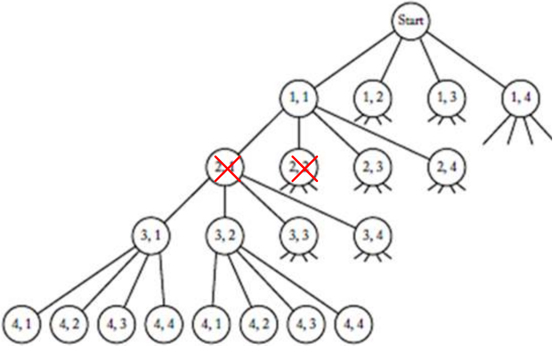
- After realizing a path is not promising
  - Backtrack to the parent
  - Proceed with the next child

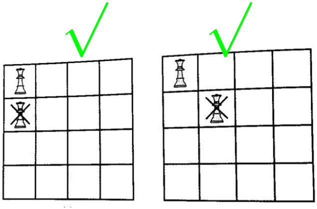




4-Queens problem (improving from DFS) (3)

- After realizing a path is not promising
  - Backtrack to the parent
  - Proceed with the next child





Promising Function

Promising Function

- Promising function is application dependent

Promising Function

- Promising function is application dependent
- Promising function *n*-queen problem:

Promising Function

- Promising function is application dependent
- Promising function n-queen problem:
  - Returns false if a node and any of the nodes ancestors place queens in the same column or diagonal

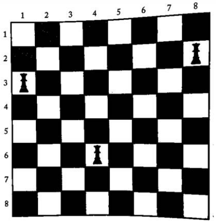
Backtracking Alg. for the n-Queens Problem

Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem

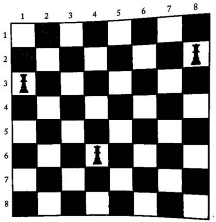
Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.



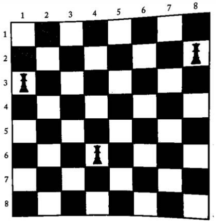
Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
    - Queens of the same row or the same columns challenge each other



Backtracking Alg. for the n-Queens Problem

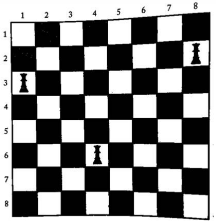
- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
  - Queens of the same row or the same columns challenge each other
  - No two queens have the same diagonal



Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
    - Queens of the same row or the same columns challenge each other
    - No two queens have the same diagonal

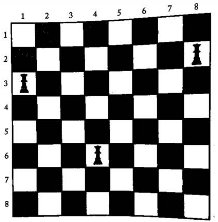
- Promising function:



Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
  - Queens of the same row or the same columns challenge each other
  - No two queens have the same diagonal

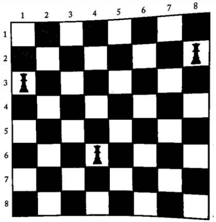
- Promising function:
  - $col(i) \neq col(k)$ and $row(i) \neq row(k)$



Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
  - Queens of the same row or the same columns challenge each other
  - No two queens have the same diagonal

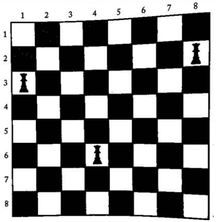
- Promising function:
  - $col(i) \neq col(k)$ and $row(i) \neq row(k)$
  - How about 2 queens in the same diagonal?



Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
  - Queens of the same row or the same columns challenge each other
  - No two queens have the same diagonal

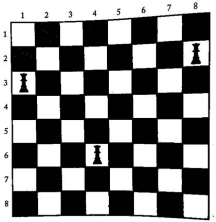
- Promising function:
  - $col(i) \neq col(k)$ and $row(i) \neq row(k)$
  - How about 2 queens in the same diagonal? $col(i) - col(k) \neq i - k$ (for left diagonal)



Backtracking Alg. for the n-Queens Problem

- All solutions to the n-Queens problem
- Apply the right constraints with the understanding of the problem.
  - Queens of the same row or the same columns challenge each other
  - No two queens have the same diagonal

- Promising function:
  - $col(i) \neq col(k)$ and $row(i) \neq row(k)$
  - How about 2 queens in the same diagonal?
    $col(i) - col(k) \neq i - k$ (for left diagonal) or
    $col(i) - col(k) \neq k - i$ (for right diagonal)



Backtracking Alg. for the n-Queens Problem (2)

Some observations about solutions:

Backtracking Alg. for the n-Queens Problem (2)

Some observations about solutions:

- Each Queen must be on a different row.

Backtracking Alg. for the n-Queens Problem (2)

Some observations about solutions:

- Each Queen must be on a different row.
- Assume Queen i is placed on row i.

Backtracking Alg. for the n-Queens Problem (2)

Some observations about solutions:

- Each Queen must be on a different row.
- Assume Queen i is placed on row i.
- So all we have to do is choose the columns.

Backtracking Alg. for the n-Queens Problem (2)

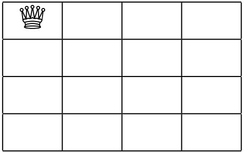
Some observations about solutions:

- Each Queen must be on a different row.
- Assume Queen i is placed on row i.
- So all we have to do is choose the columns.
- A candidate configuration can be represented by an n-tuple, $< x_1, x_2, \ldots, x_n >$, where x_i is the column on which Queen i is placed.

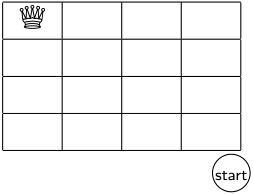
Example of How Backtracking works for 4-queen Problem

Example of How Backtracking works for 4-queen Problem

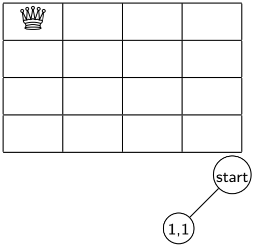
Example of How Backtracking works for 4-queen Problem



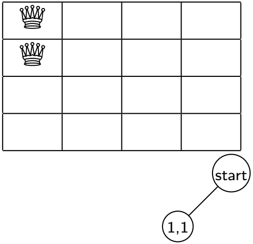
Example of How Backtracking works for 4-queen Problem



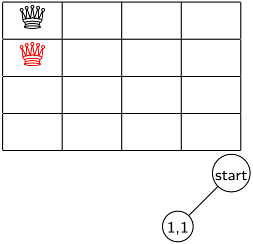
Example of How Backtracking works for 4-queen Problem



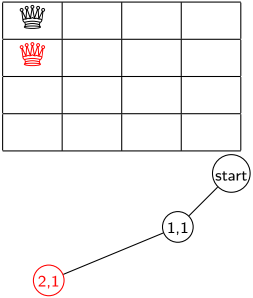
Example of How Backtracking works for 4-queen Problem



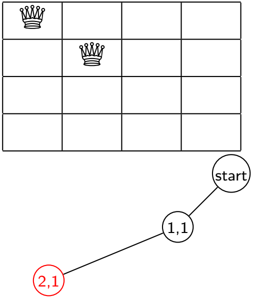
Example of How Backtracking works for 4-queen Problem



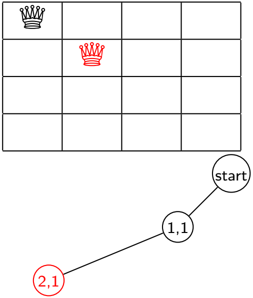
Example of How Backtracking works for 4-queen Problem



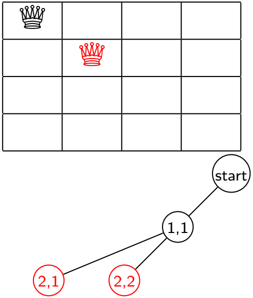
Example of How Backtracking works for 4-queen Problem



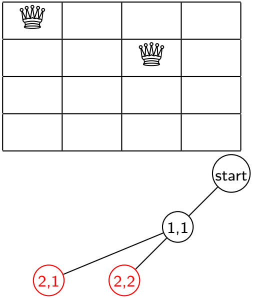
Example of How Backtracking works for 4-queen Problem



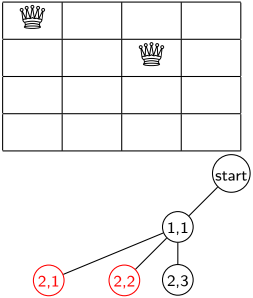
Example of How Backtracking works for 4-queen Problem



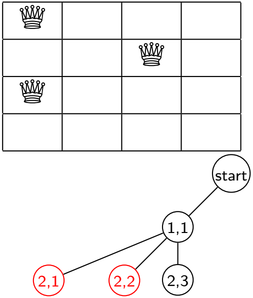
Example of How Backtracking works for 4-queen Problem



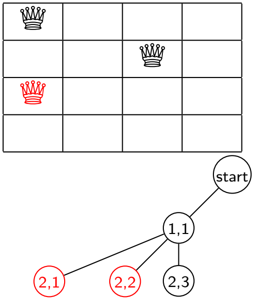
Example of How Backtracking works for 4-queen Problem



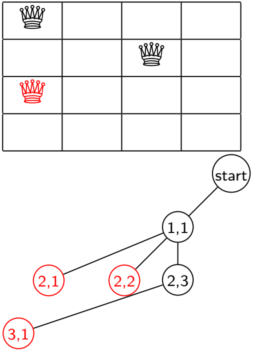
Example of How Backtracking works for 4-queen Problem



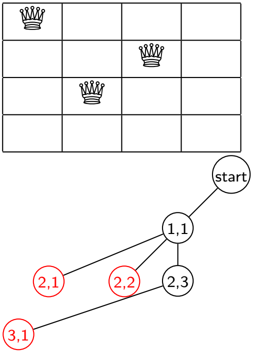
Example of How Backtracking works for 4-queen Problem



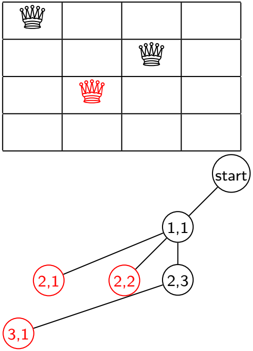
Example of How Backtracking works for 4-queen Problem



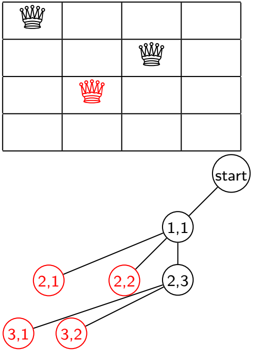
Example of How Backtracking works for 4-queen Problem



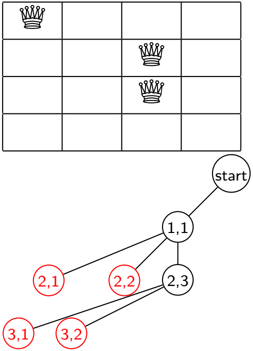
Example of How Backtracking works for 4-queen Problem



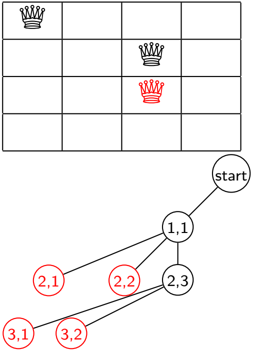
Example of How Backtracking works for 4-queen Problem



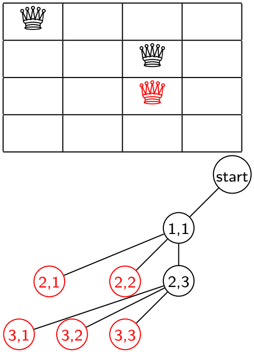
Example of How Backtracking works for 4-queen Problem



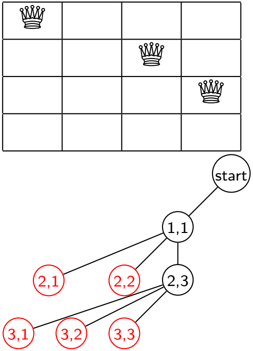
Example of How Backtracking works for 4-queen Problem



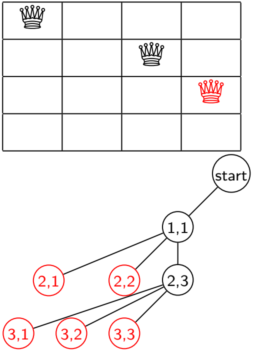
Example of How Backtracking works for 4-queen Problem



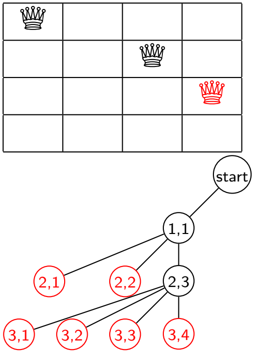
Example of How Backtracking works for 4-queen Problem



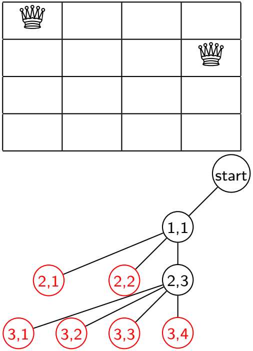
Example of How Backtracking works for 4-queen Problem



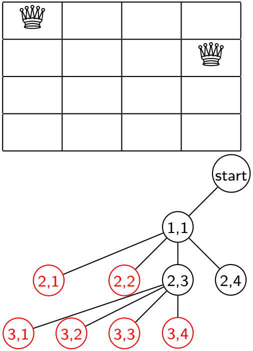
Example of How Backtracking works for 4-queen Problem



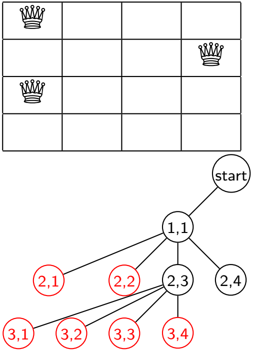
Example of How Backtracking works for 4-queen Problem



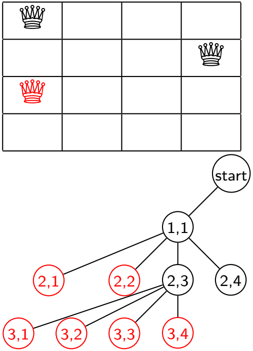
Example of How Backtracking works for 4-queen Problem



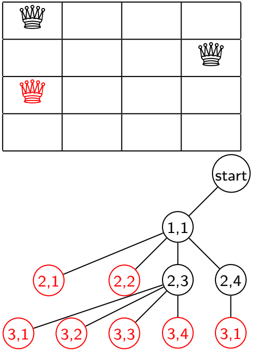
Example of How Backtracking works for 4-queen Problem



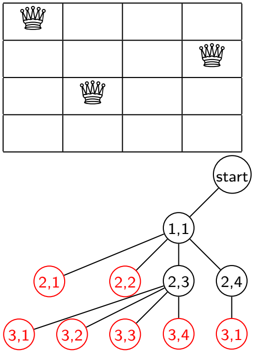
Example of How Backtracking works for 4-queen Problem



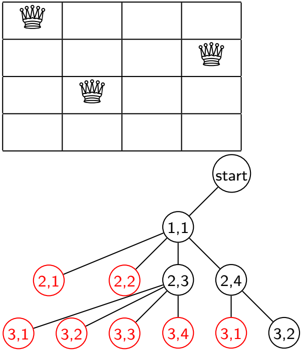
Example of How Backtracking works for 4-queen Problem



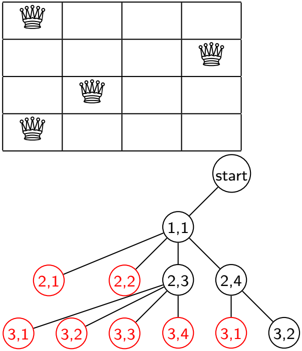
Example of How Backtracking works for 4-queen Problem



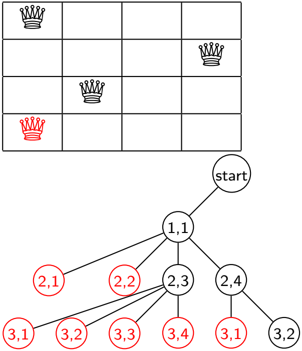
Example of How Backtracking works for 4-queen Problem



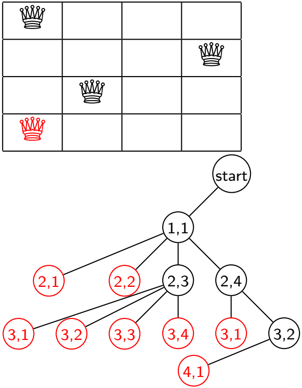
Example of How Backtracking works for 4-queen Problem



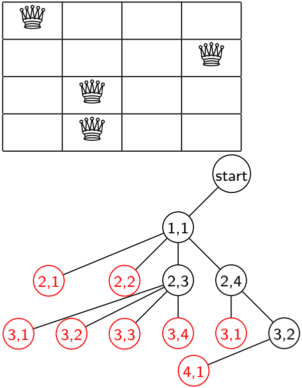
Example of How Backtracking works for 4-queen Problem



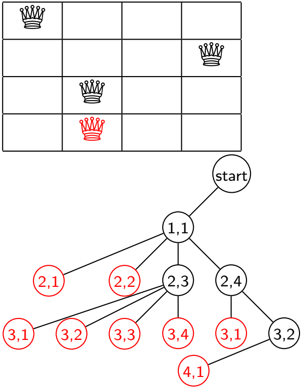
Example of How Backtracking works for 4-queen Problem



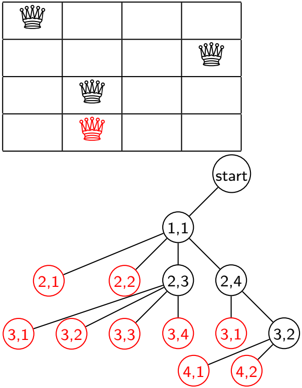
Example of How Backtracking works for 4-queen Problem



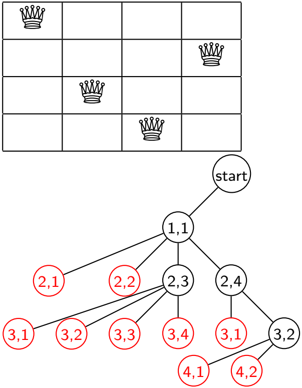
Example of How Backtracking works for 4-queen Problem



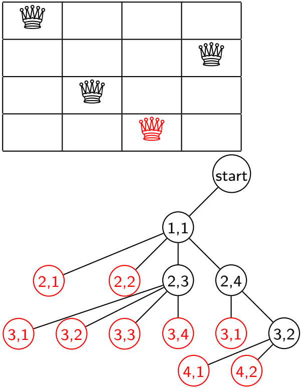
Example of How Backtracking works for 4-queen Problem



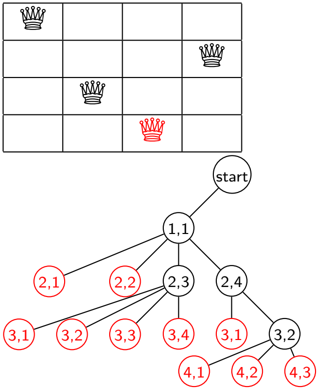
Example of How Backtracking works for 4-queen Problem



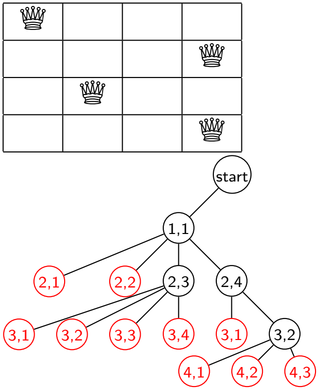
Example of How Backtracking works for 4-queen Problem



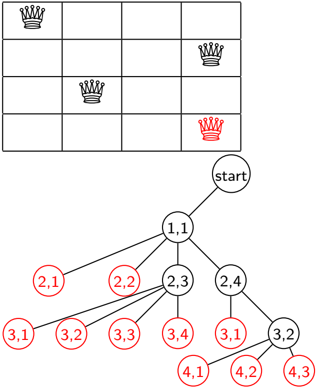
Example of How Backtracking works for 4-queen Problem



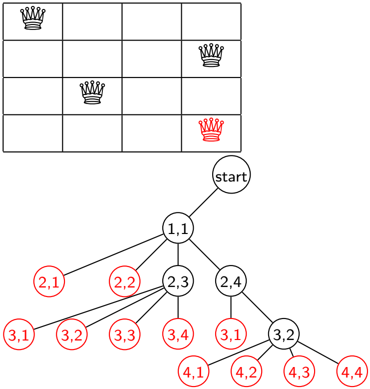
Example of How Backtracking works for 4-queen Problem



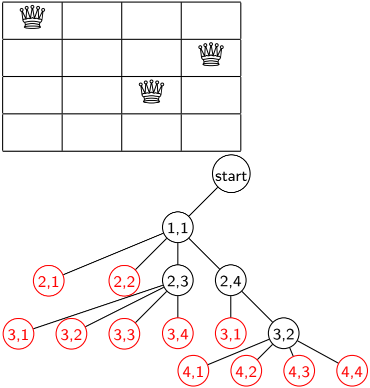
Example of How Backtracking works for 4-queen Problem



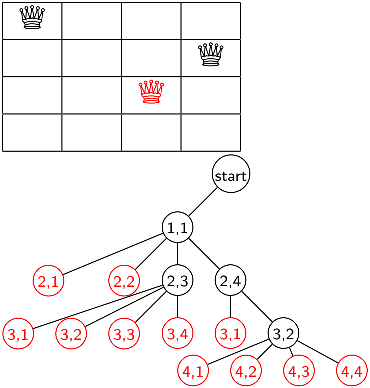
Example of How Backtracking works for 4-queen Problem



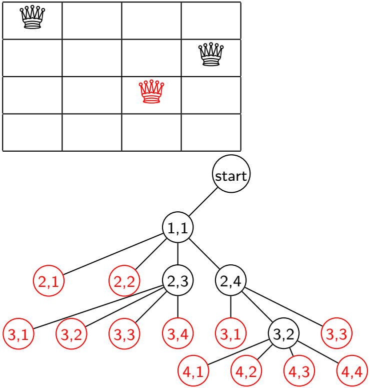
Example of How Backtracking works for 4-queen Problem



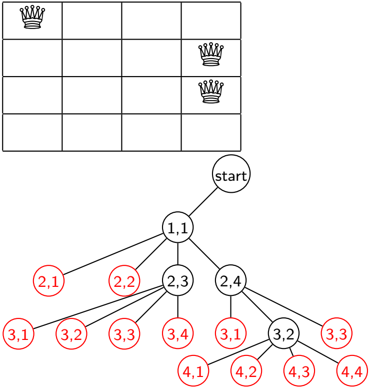
Example of How Backtracking works for 4-queen Problem



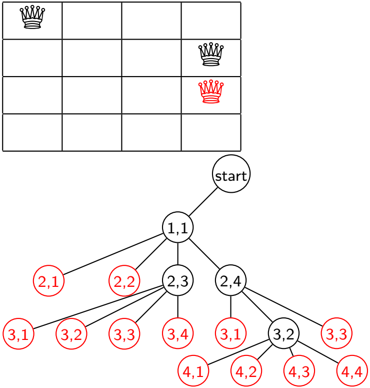
Example of How Backtracking works for 4-queen Problem



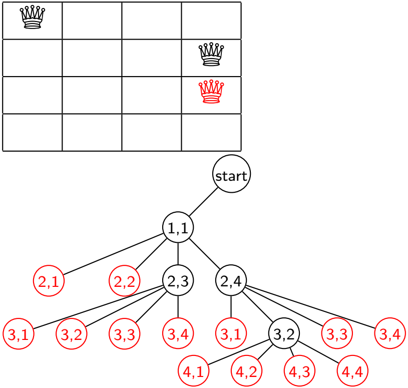
Example of How Backtracking works for 4-queen Problem



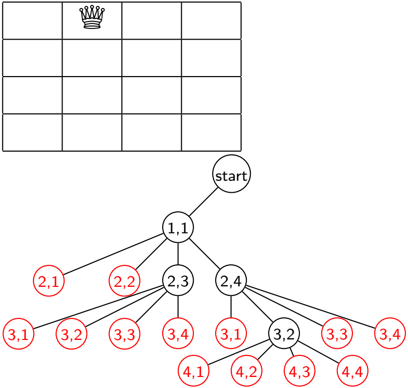
Example of How Backtracking works for 4-queen Problem



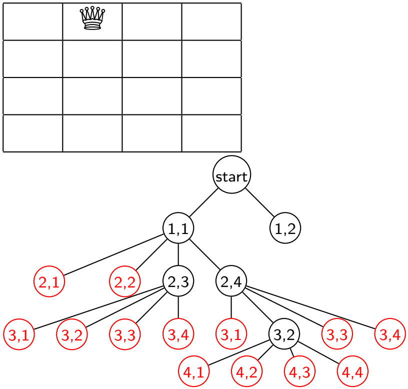
Example of How Backtracking works for 4-queen Problem



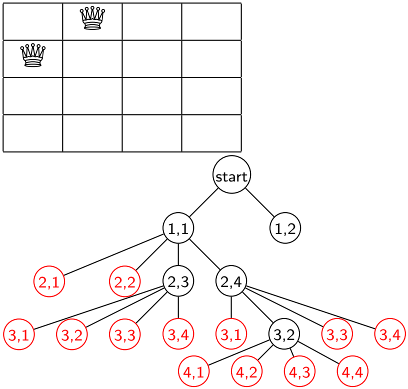
Example of How Backtracking works for 4-queen Problem



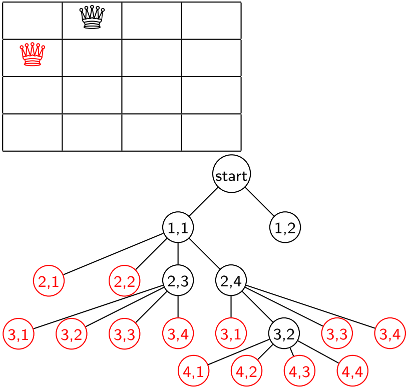
Example of How Backtracking works for 4-queen Problem



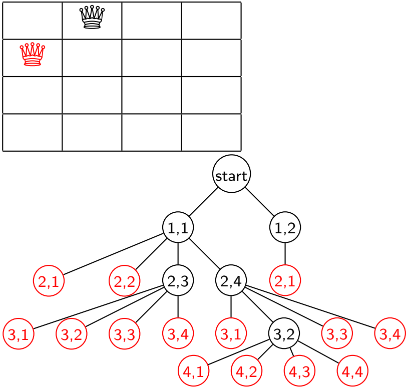
Example of How Backtracking works for 4-queen Problem



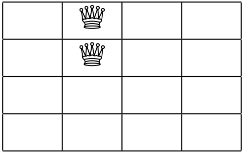
Example of How Backtracking works for 4-queen Problem

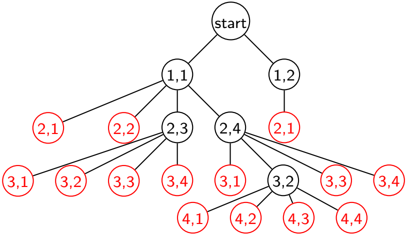


Example of How Backtracking works for 4-queen Problem

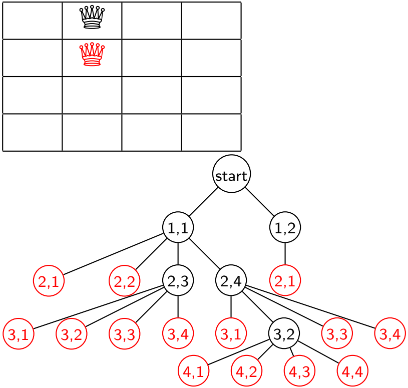


Example of How Backtracking works for 4-queen Problem

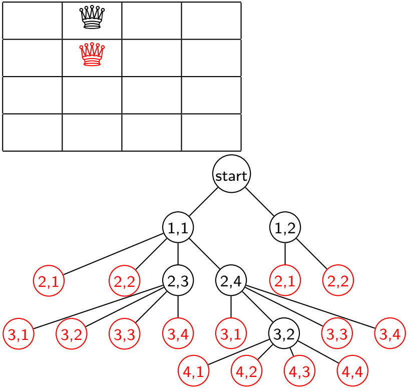




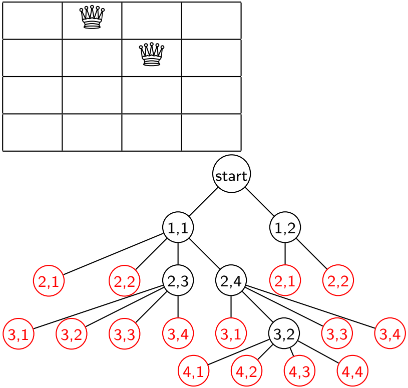
Example of How Backtracking works for 4-queen Problem



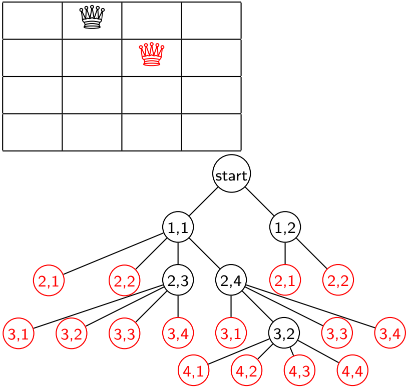
Example of How Backtracking works for 4-queen Problem



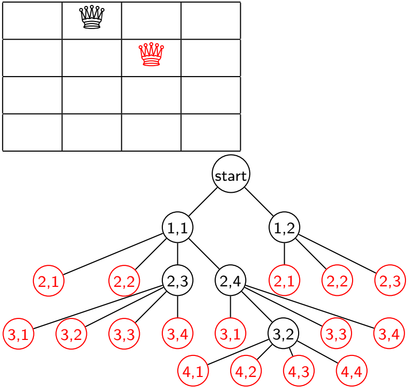
Example of How Backtracking works for 4-queen Problem



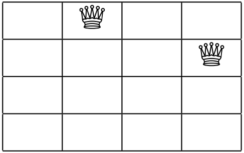
Example of How Backtracking works for 4-queen Problem

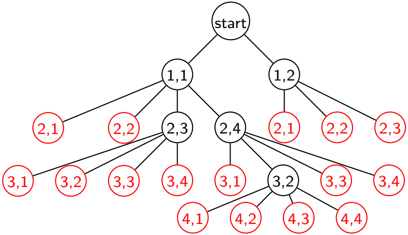


Example of How Backtracking works for 4-queen Problem

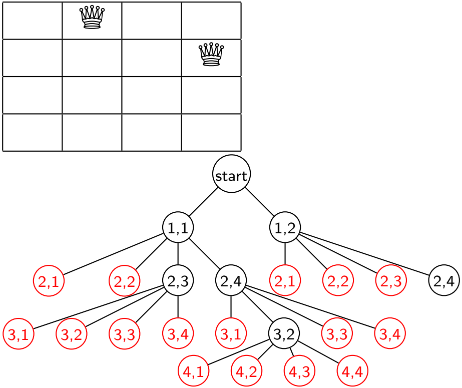


Example of How Backtracking works for 4-queen Problem

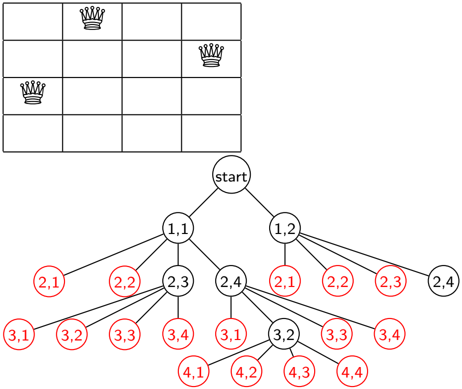




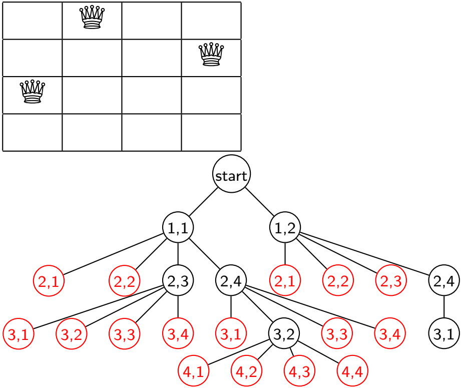
Example of How Backtracking works for 4-queen Problem



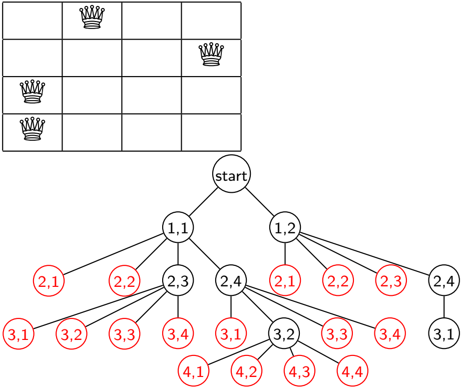
Example of How Backtracking works for 4-queen Problem



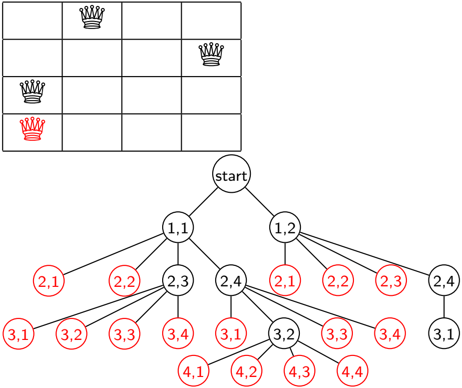
Example of How Backtracking works for 4-queen Problem



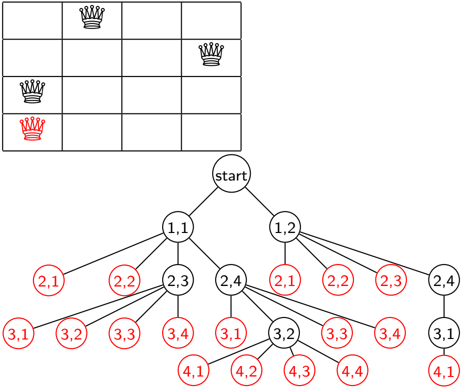
Example of How Backtracking works for 4-queen Problem



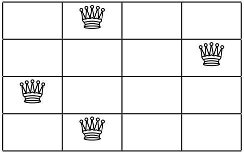
Example of How Backtracking works for 4-queen Problem

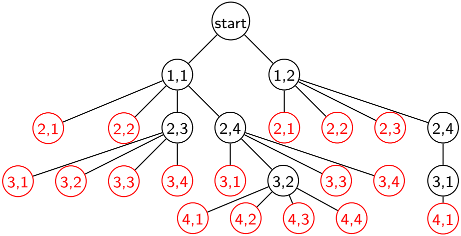


Example of How Backtracking works for 4-queen Problem

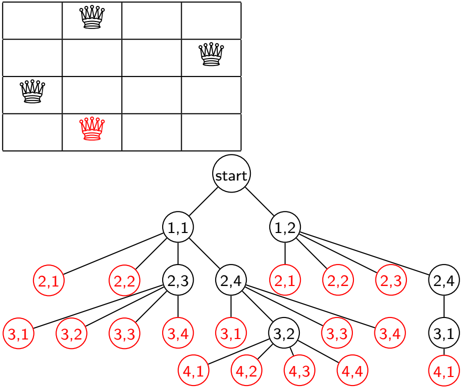


Example of How Backtracking works for 4-queen Problem

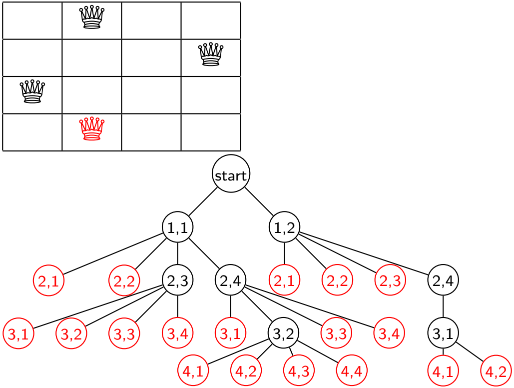




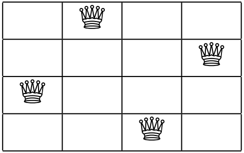
Example of How Backtracking works for 4-queen Problem

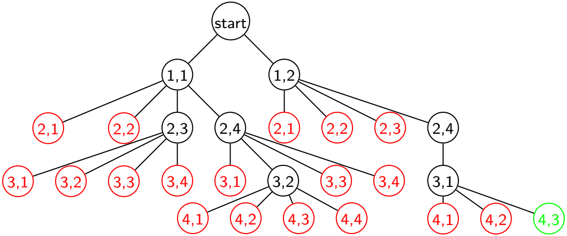


Example of How Backtracking works for 4-queen Problem

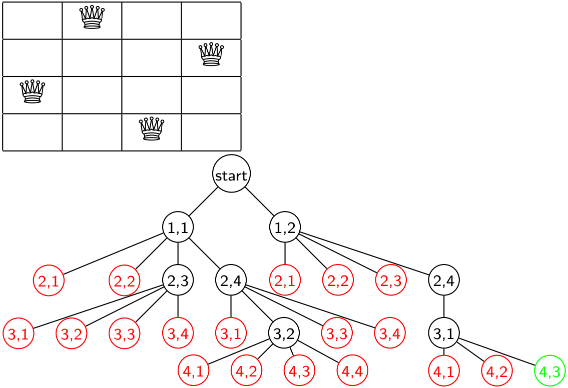


Example of How Backtracking works for 4-queen Problem



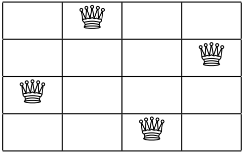


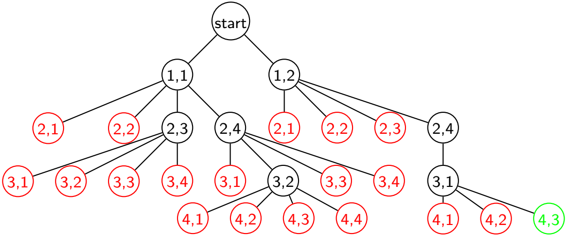
Example of How Backtracking works for 4-queen Problem



- The first solution is <2, 4, 1, 3>

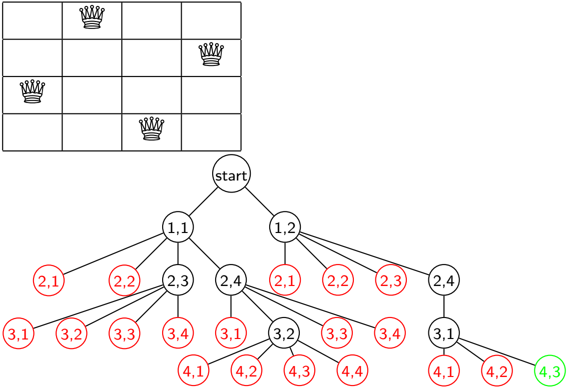
Example of How Backtracking works for 4-queen Problem





- The first solution is <2, 4, 1, 3>
- The backtracking algorithms checks 27 nodes before finding a solution.

Example of How Backtracking works for 4-queen Problem



- The first solution is $<2, 4, 1, 3>$
- The backtracking algorithms checks 27 nodes before finding a solution.
- Without backtracking, a depth-first search of the state space tree checks 155 nodes before finding the same solution.

Backtracking Alg. for the n-Queens problem (3)

Backtracking Alg. for the n-Queens problem (3)

```
void Queens (index i){
    index j;
    if (promising(i))
       if (i==n)
          cout<< col[1] through col[n]
       else
          for(j=1; j<=n; j++){
             col[i+1]=j;
             Queens(i+1);
          }
}
                             // Call the function with Queens(0);
bool promising (index i){
    index k=1;
    bool switch=true;
    while (k<i && switch){
       if (col[i]==col[k] || abs(col[i]-col[k])==i-k)
          switch = false;
       k++;
       }
    return switch;
}
```

Backtracking Procedure

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
  - Promising node: may lead to a solution

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
  - Promising node: may lead to a solution
- Backtracking:

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
  - Promising node: may lead to a solution
- Backtracking:
  - DFS of state space tree

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
  - Promising node: may lead to a solution
- Backtracking:
  - DFS of state space tree
  - Pruning state space tree:
    if a node is determined to be non-promising, back track to its parent

Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
  - Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
  - Promising node: may lead to a solution
- Backtracking:
  - DFS of state space tree
  - Pruning state space tree:
    if a node is determined to be non-promising, back track to its parent
  - Pruned State Space Tree:
    sub-tree consisting of visited nodes