Chapter 5
Backtracking

단계적으로 선택을 해나가다가 조건이 만족되지 않으면 이전 상태로 복귀하여 남은 것 중에 선택을 해나간다.

Worst case time complexity는 나쁘지만
average case time complexity는 효율적이다.

Backtracking

- A typical example
  - The maze of hedges by Hampton Court Palace
  - The maze of Micro-mouse
- Applicable problems
  - The solution is a sequence of objects satisfying some criterion.
- General approach
  - Basically generate all possible candidates
  - *Prune* the state space tree due to a decision function
  - Still exponential time in the worst case, but efficient for many large instances.

The 8-Queens Problem

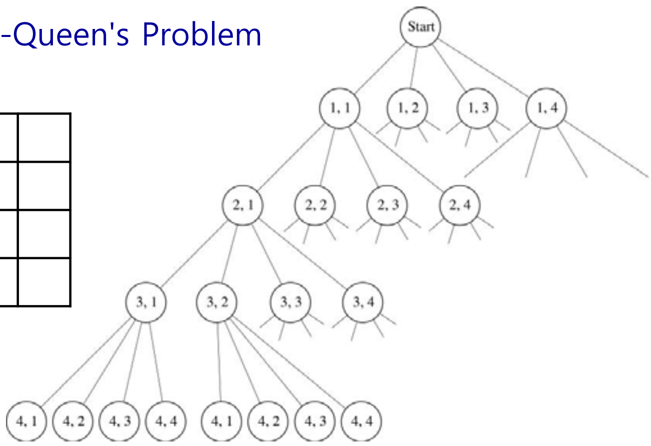
- ✓**Horse**: King, Queen, 2 Rooks, 2 Bishops, 2 Knights
- ✓Every horse threatens others on the same row, on the same column, or on the same diagonals.

			Q				
					Q		
							Q
	Q						
						Q	
Q							
		Q					
				Q			

State Space Tree

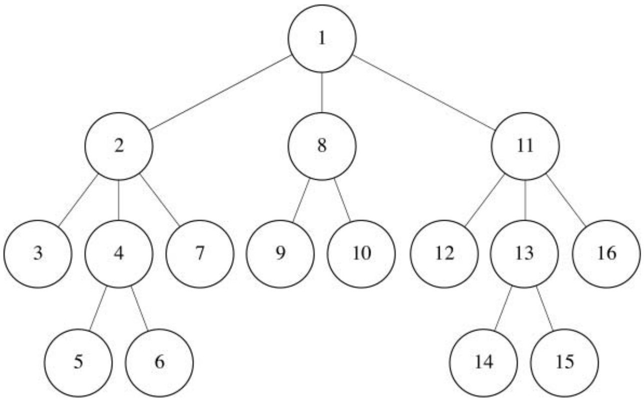
- ✓The tree such that each path from the root to a leaf node is a candidate solution
- ✓Example) 4-Queen's Problem

Q			
	Q		



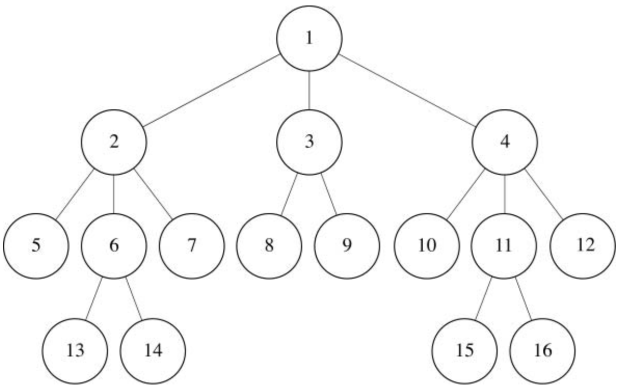
Tree Traversal

- ✓Search sequence of possible candidates
- ✓Depth-First Search (DFS)



Tree Traversal

- ✓Breadth-First Search (BFS)



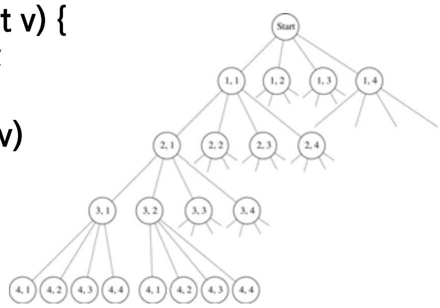
- ✓Best-First Search (Branch-and-Bound Algorithm)

Backtracking Algorithm

- ✓**Non-promising** node (in a state space tree)
  - ✓the node cannot lead to a solution
- ✓**Promising** node
  - ✓all nodes except non-promising nodes
- ✓**Pruning of a state space tree**
  - ✓If the visited node is non-promising, backtrack to the parent node.

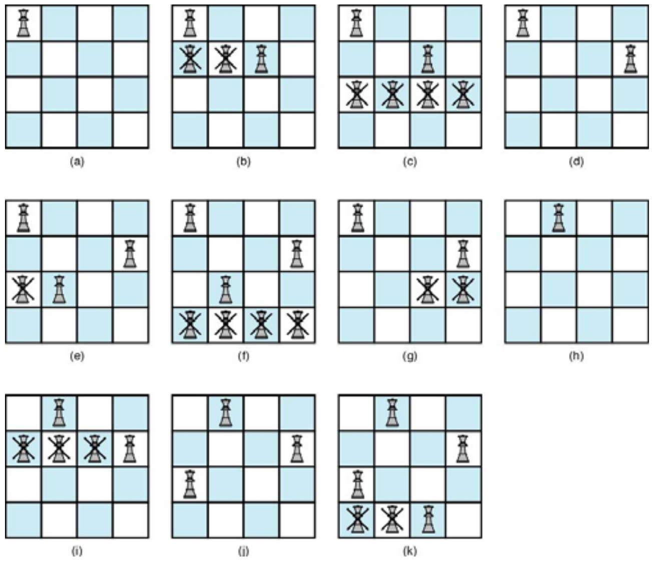
Outline of the algorithm

```
void checknode(node v)
{
   node u;
   if (promising(v)) {
      if (there is a solution at v) {
         output the solution;
      } else {
         for (each child u of v)
            checknode(u);
      }
   }
}
```

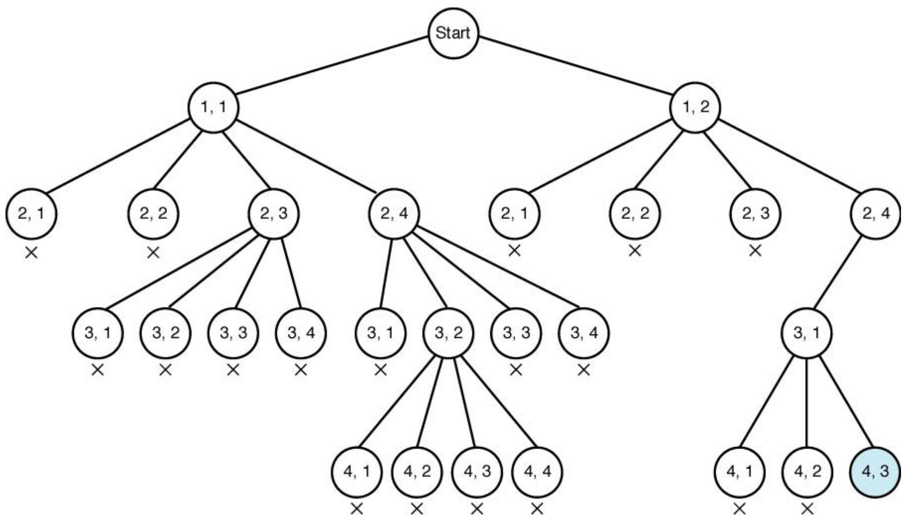


✓*(Note) The state space tree exists implicitly.*

4 Queen's Problem



State Space Tree of 4 Queen's Problem

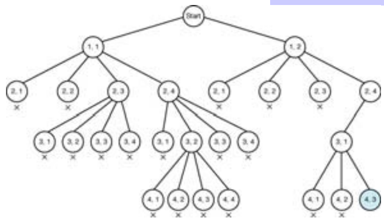


The n-Queens Problem

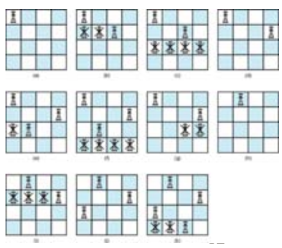
- ✓Promising Function (or *bounding function*)
  - ✓the same row : implicitly impossible
  - ✓the same column : col(i) = col(k)
  - ✓the same right diagonal : col(i) - col(k) = i-k
  - ✓the same left diagonal : col(i) - col(k) = k-i
- ✓Algorithm 5.1

● The algorithm (print all solutions)

- Algorithm 5.1 ***queens***
- Top level call : queens(0)



```
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);
            }
    }
}
```



Analysis of the Algorithm

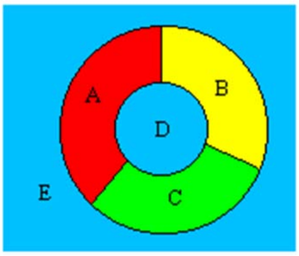
- Comparison from some instances (Table. 5.1)
  - (# of checked nodes of the state space tree)
  - Algorithm 1: DFS without backtracking ($\sim n^n$)
  - Algorithm 2: avoid same column (n!)
  - Algorithm 3: apply checknode algorithm
  - Algorithm 4: apply expand algorithm
- Theoretical analysis
  - Difficult
- Measure real execution time
  - Not-reasonable
- Estimate the efficiency by probabilistic analysis
  - may use Monte Carlo Algorithm

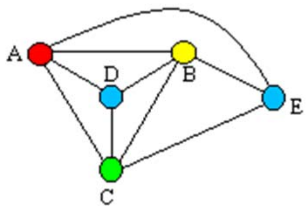
(참고) Monte Carlo Algorithm

- Characteristics
  - a probabilistic algorithm (c.f. deterministic algorithm)
  - the next instruction executed is determined at random
  - no guarantee that the estimate is close to the true expected value, but the probability is high
- Necessary condition
  - 1. the same promising function for all nodes at the same level
  - 2. nodes at the same level have the same number of children
- Outline of the Monte Carlo estimate algorithm
  - Algorithm 5.2 *Estimate*

Graph Coloring

- ✓**Graph coloring problem** (*m*-coloring)
  - ✓Color vertices of an undirected graph such that
  - ✓1. no two adjacent vertices are the same color
  - ✓2. and, use at most *m* colors
- ✓**Map coloring**
  - ✓Map and its ***planar graph*** representation

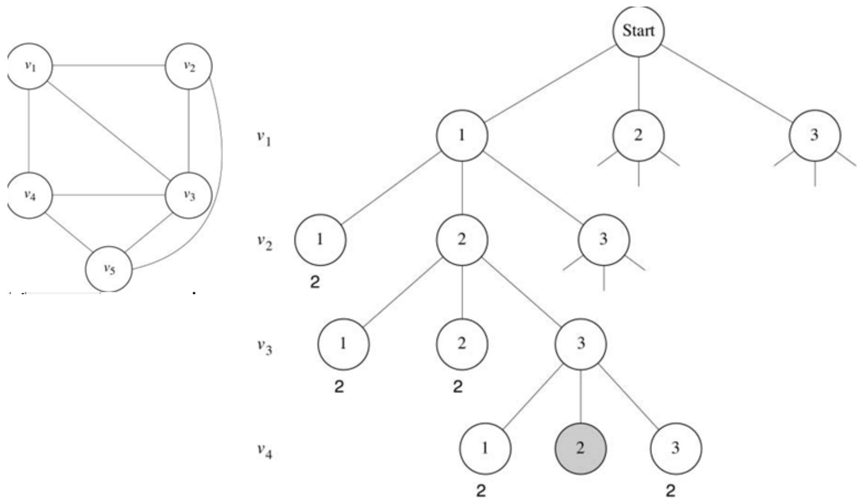




Graph Coloring

- ✓The problem
  - ✓Find all different *m*-colorings for a given graph
- ✓Promising function
  - ✓A node of the state space tree is non-promising if two adjacent vertices are the same color
- ✓State space tree
  - ✓3-Coloring of Fig. 5.10
- ✓Algorithm 5.5 *m_coloring*
  - ✓adjacency matrix W[i][j] : 1/0 represents that v_i and v_j are adjacent or not
  - ✓output vcolor[1..n] : color number (1..m)
  - ✓top-level call : m_coloring(0)

3-Coloring Example



Algorithm 5.5

```
void m_coloring (index i)
{
  int color;

  if (promising(i))
     if (i == n)
        cout << vcolor[1] through vcolor[n];
     else
        for (color = 1; color <= m; color++){
           vcolor[i + 1] = color;
           m_coloring(i + 1);
        }
}
```

```
bool promising (index i)
{
  index j;
  bool switch;

  switch = true;
  j = 1;
  while (j < i && switch){
     if (W[i][j] && vcolor[i] == vcolor[j])
        switch = false;
     j++;
  }
  return switch;
}
```

Promising of Algorithm 5.5

(참고) More Considerations

- ✓2-Coloring Problem : exist efficient algorithms
- ✓m-Coloring Problem for m >= 3 : NP-Hard

- ✓4-colorability conjecture for planar graph
  - ✓first noted by August Ferdinand Mobius, 1840
  - ✓proven to be true by Appel and Haken, 1976, with the help of computers
  - ✓2005, proven by Georges Gonthier with theorem proving software

0-1 Knapsack Problem

- Assumption
  - objects are ordered in p_i/w_i order
- Outline of the algorithm

```
void checknode(node v)
{
    node u;
    if (value(v) is better than best)
        best = value(v);
    if (promising(v))
        for (each child u of v)
            checknode(u);
}
```


- Promising function (bounding function)

  (1) *weight <= W*

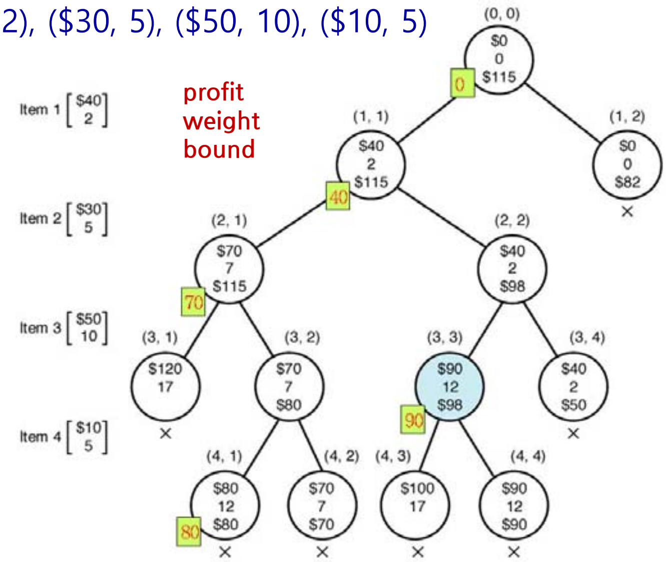
  (2) *bound > the known best solution,* where *bound* is the profit bound of solutions by expanding the node.

  Note) the profit bound might be the output of the Fractional Knapsack Problem

Example 5.6, Fig. 5.14

(p,w) : ($40, 2), ($30, 5), ($50, 10), ($10, 5)

W = 16



Algorithm 5.7 knapsack

The top level call

```
numbest = 0;        // # of items considered
maxprofit = 0;      // the known max profit
knapsack(0, 0, 0);
printResult();      // maxprofit, bestset[];
```

Algorithm 5.7

```
void knapsack (index i,
               int profit, int weight)
{
  if (weight <= W && profit > maxprofit){         // This s
     maxprofit = profit;                          // so fa
     numbest = i;                                 // Set n
     bestset = include;                           // numbe
  }                                               // consi
                     profit, weight               // bests
                                                  // solut
  if (promising(i)){
     include[i + 1] = "yes";                      // Inclu
     knapsack(i + 1, profit + p[i + 1], weight + w[i + 1]);
     include[i + 1] = "no";                       // Do no
     knapsack(i + 1, profit, weight);             // w[i +
  }
}
```


Promising of Algorithm 5.7

```
                              profit, weight
bool promising (index i)
{
  index j, k;
  int totweight;
  float bound;

  if (weight >= W)                                 // Node is
     return false;                                 // if we s
  else{                                            // its chi
     j = i + 1;                                    // be some
     bound = profit;                               // the chi
     totweight = weight;
     while (j <= n && totweight + w[j] < = W){
                                                   // Grab as
       totweight = totweight + w[j];               // possib
       bound = bound + p[j];
       j++;
     }
     k = j;                                        // Use k
     if (k <=n)                                    // with f
       bound = bound + (W - totweight) * p[k]/w[k];
                                                   // Grab f
     return bound > maxprofit;                     // item.
  }
}
```

Worst-case time complexity

- # of visited nodes in the state space tree

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \in \Theta(2^n)$$

- time complexity of ***promising***

$$n - i \in O(n)$$

- worst-case time complexity

$$W(n) = (2^{n+1} - 1)(n - i) \leq 2n \cdot 2^n$$

Therefore, $W(n) \in O(n2^n)$

- Simple derivation

$$W(n) \in \Theta(2^n) \cdot O(n) = O(n2^n)$$

정리

- ✓Backtracking Algorithm
- ✓M-Coloring Problem
- ✓0-1 Knapsack Problem
- ✓Chapter 6: Branch and Bound Algorithm
  - ✓0-1 Knapsack Problem
  - ✓Traveling Salesperson Problem