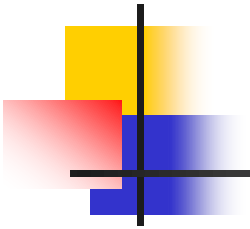
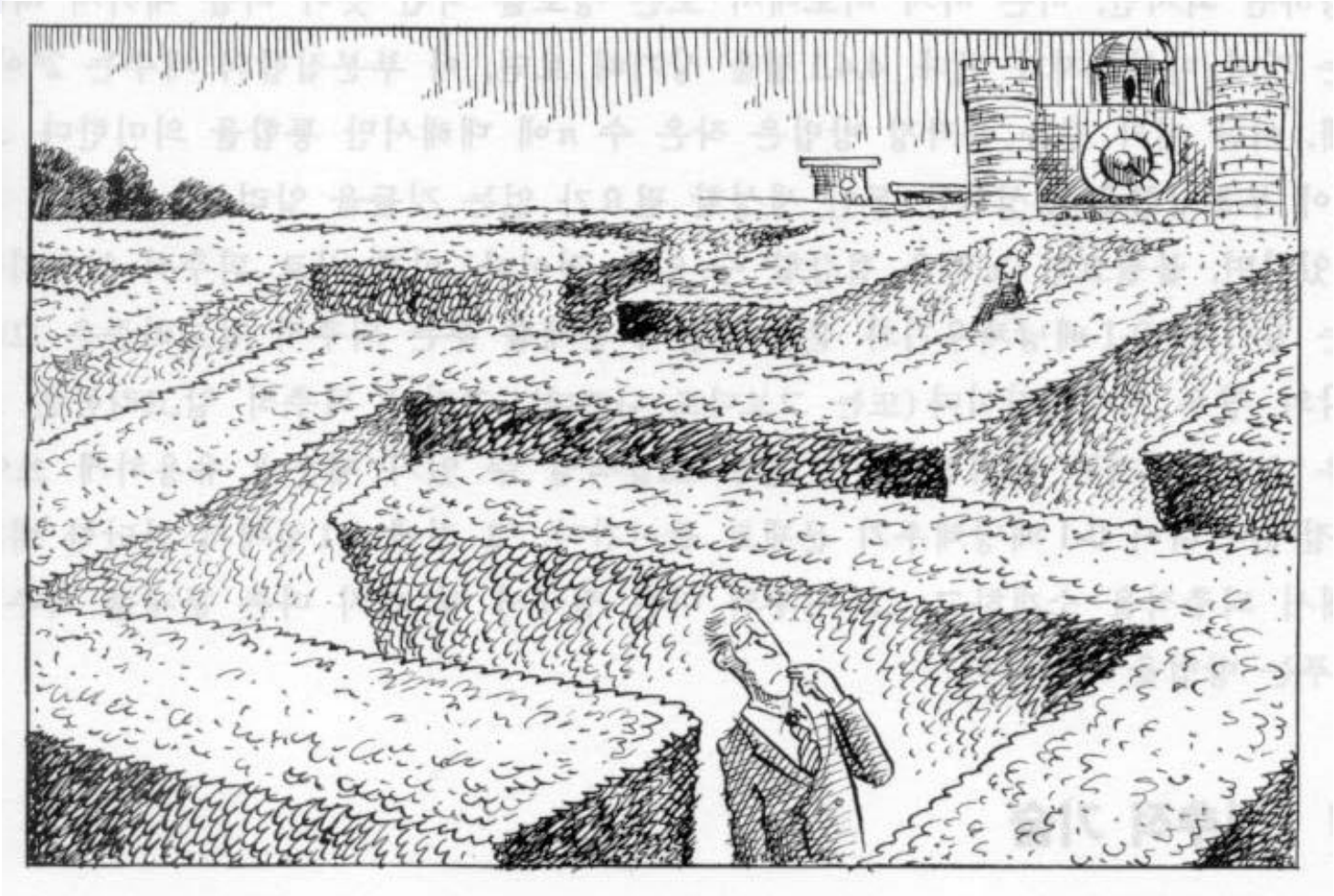
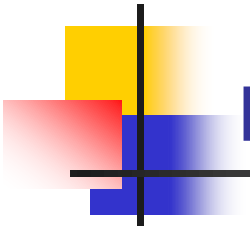


[Chapter 5]

Backtracking

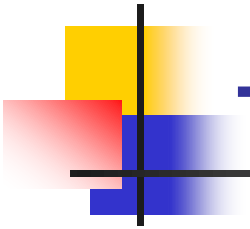
Labyrinth





Backtracking Technique

- Think about a well-known maze of hedges
  - Hitting dead ends may frustrate you.
  - Any sign for dead ends?
- They do exist in backtracking algorithms.
  - *Choose* a sequence of objects from a specified set.
    - A modified ***depth-first-search*** of a tree
  - *Devise* your own criterion to maintain such sequence.
    - We call it the ***promising function*** for a backtracking algorithm

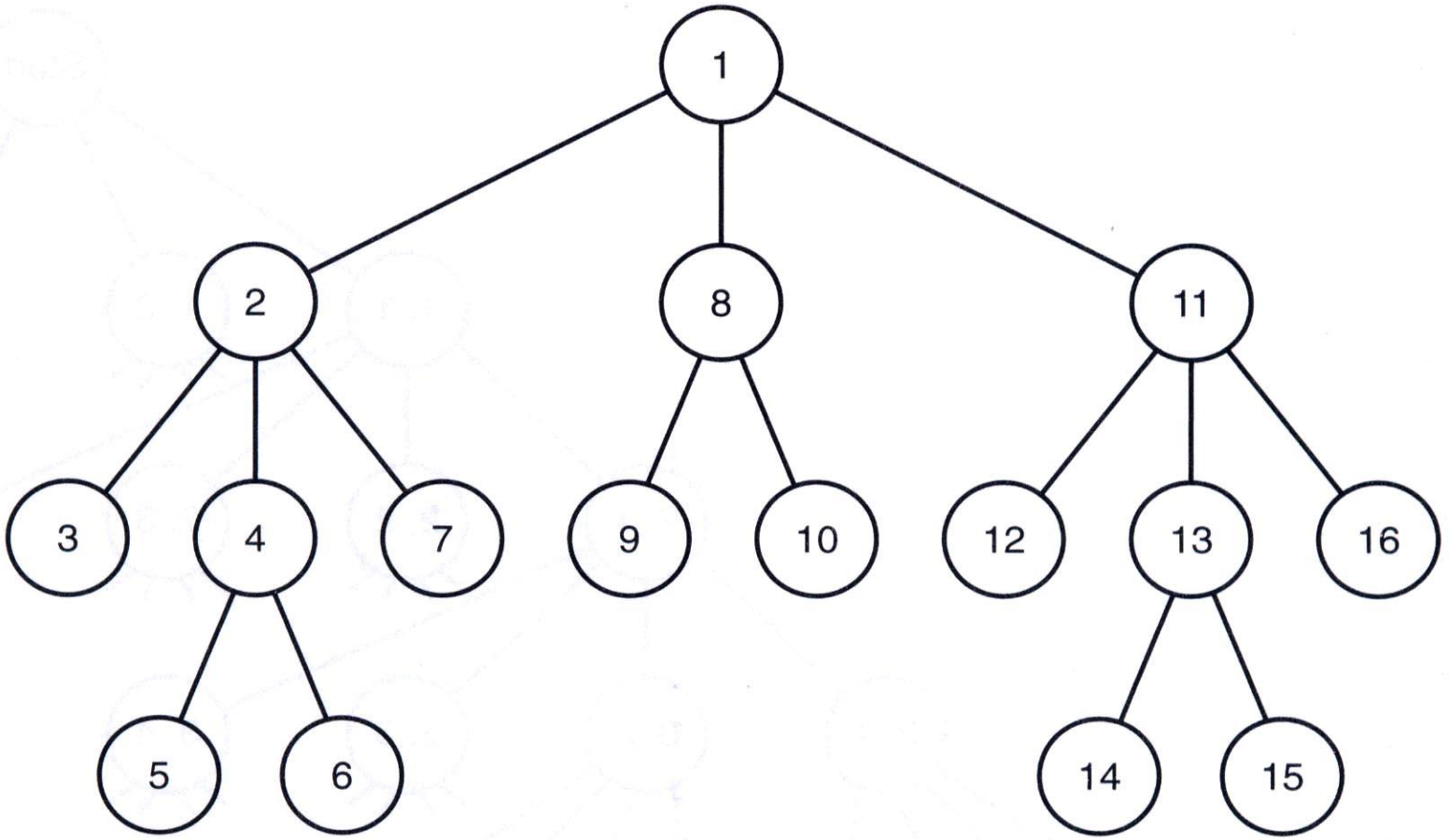


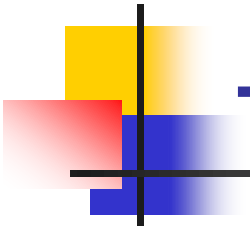
The Depth-First Search

- Search a path as deeply as possible until a dead end is reached.
  - Visit a root first.
  - And then, visit one and each of its descendant nodes.
  - Generally, from left to right.

```
void depth_first_tree_search (node v) {
        node u;
        visit v;
        for (each child u of v)
            depth_first_tree_search(u)
}
```

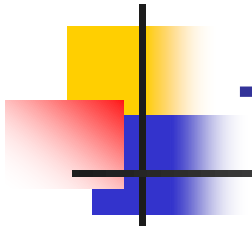
Example: The Depth-First Search





The General Backtracking Algorithm

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



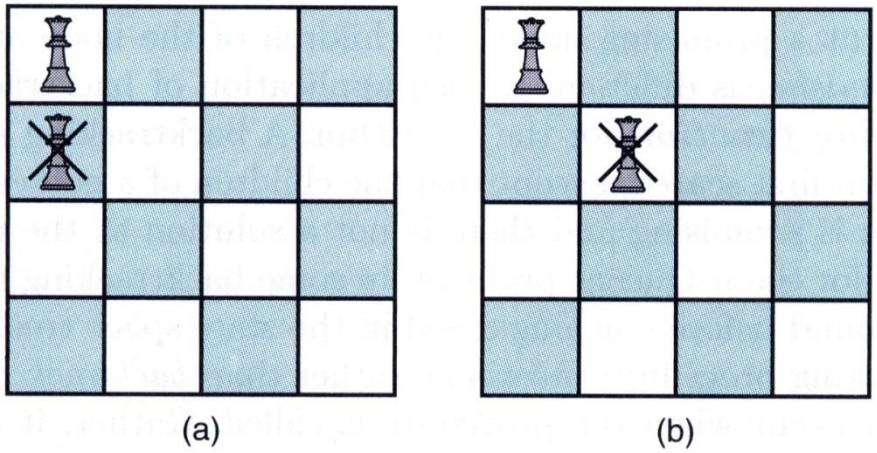
The Revised Backtracking Algorithm

```
void expand(node v) {
        node u;

        for (each child u of v)
            if (promising(u))
                if (there is a solution at u)
                    write the solution;
                else
                    expand(u);
}
```

4 Queens Problem

- To position 4 queens on 4 x 4 chessboard
- No two queens should threaten each other.
  - No two queens in the same row.
  - No two queens in the same column.
  - No two queens in the same diagonal.



The state space tree for the 4-Queens problem

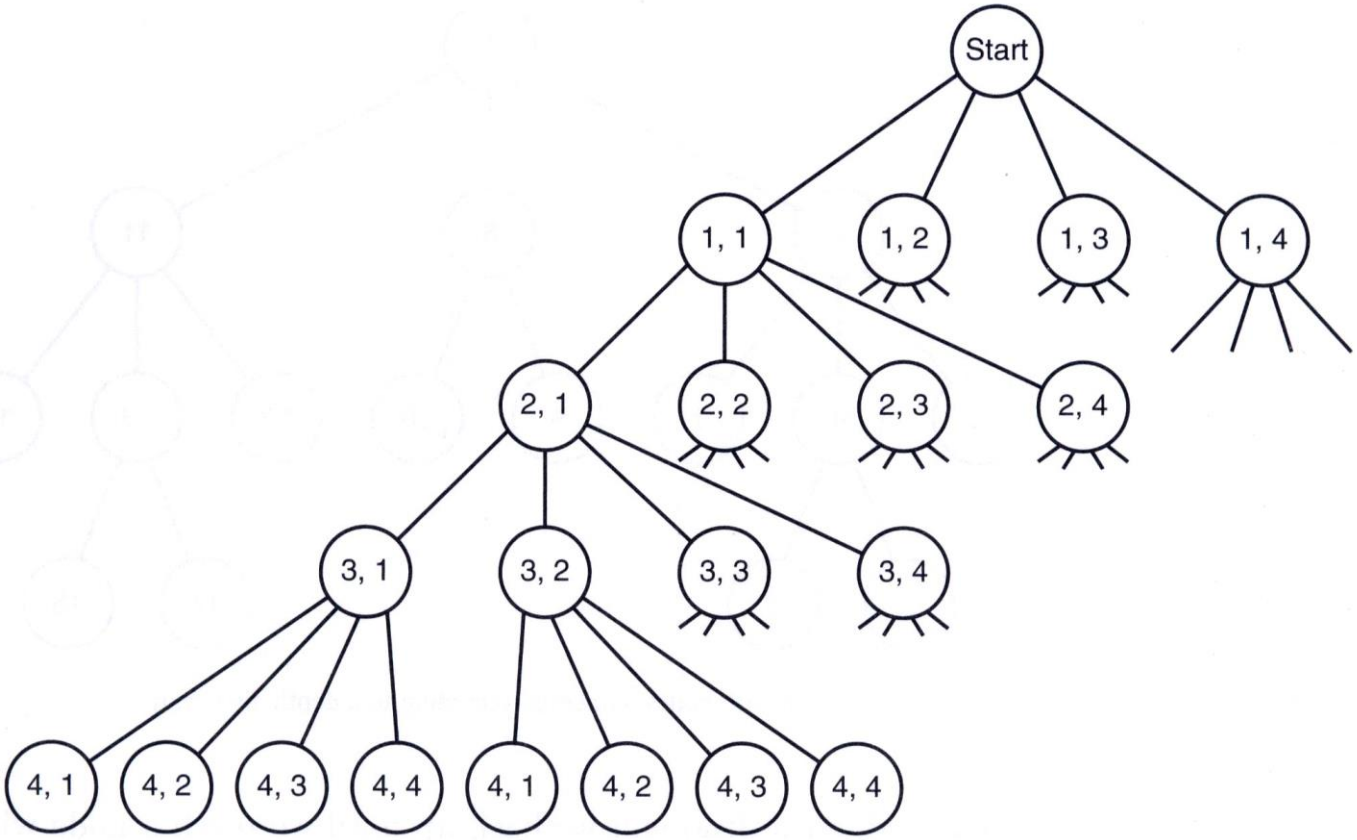


Figure 5.4 A portion of the pruned state space tree produced when backtracking is used to solve the instance of the *n*-Queens problems in which n = 4. Only the nodes checked to find the first the solution are shown. That solution is found at the color-shaded node. Each non-promising node is marked with a cross.

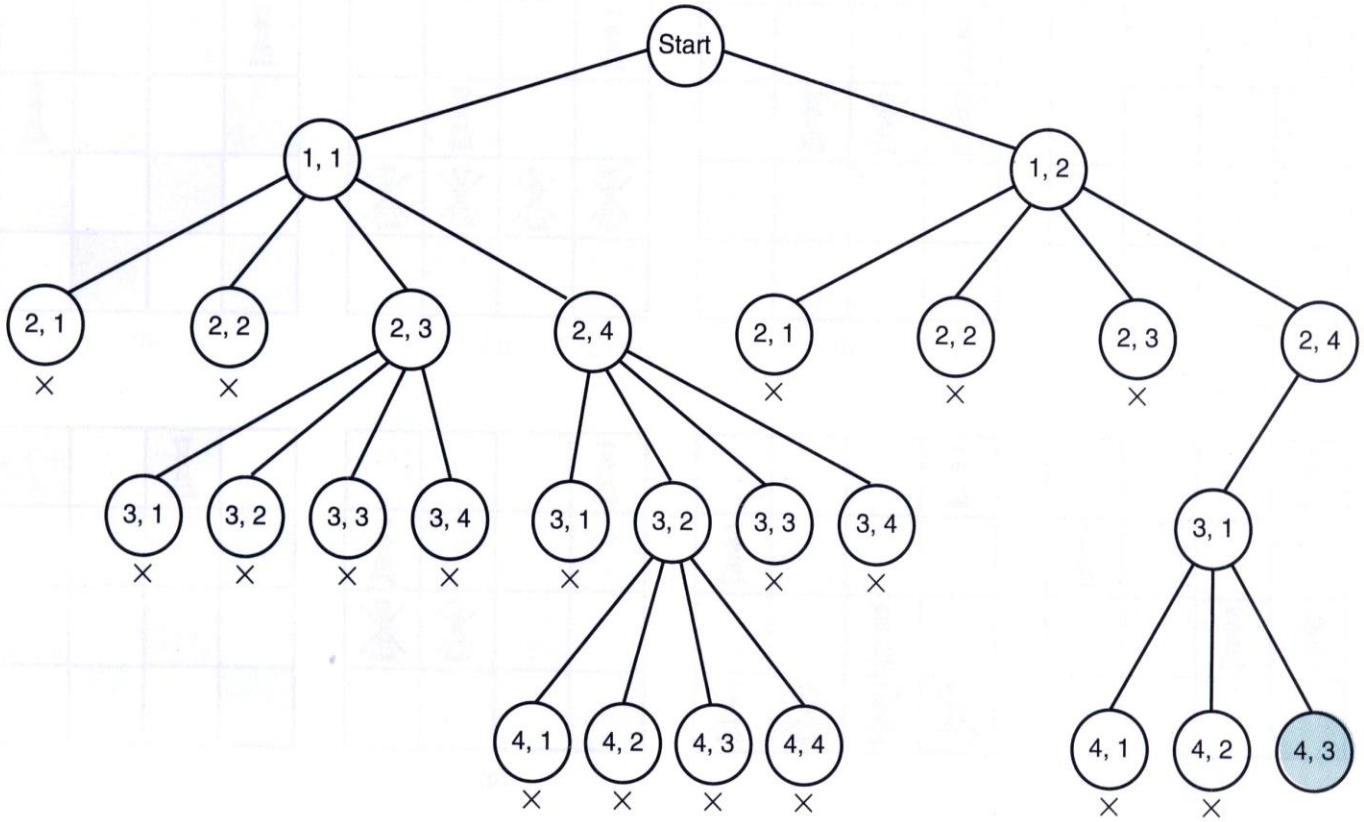
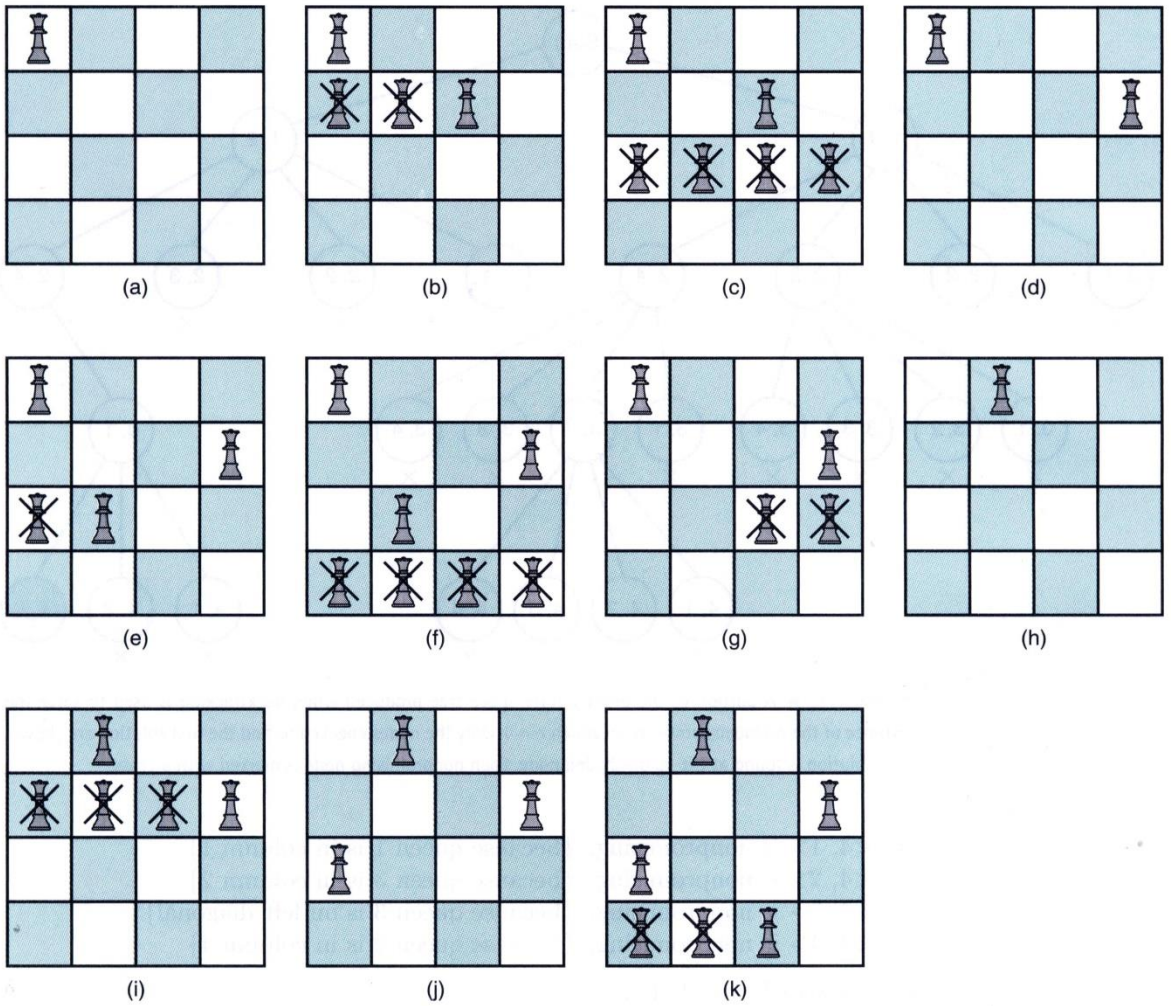
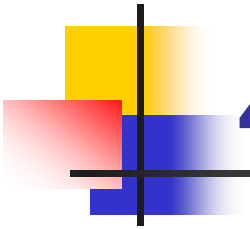


Figure 5.5 The actual chessboard positions that are tried when backtracking is used to solve the instance of the *n*-Queens problem in which n = 4. Each nonpromising position of marked with a cross.





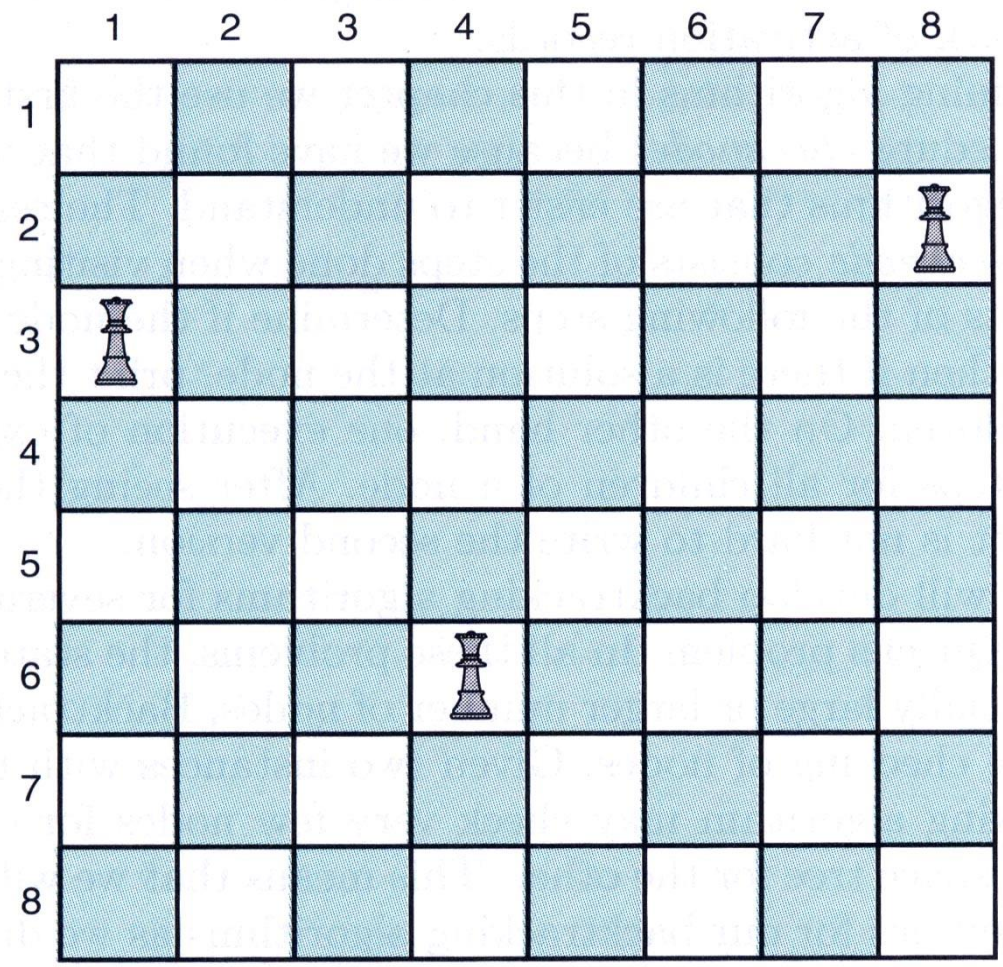
4 Queens Problem (Again)

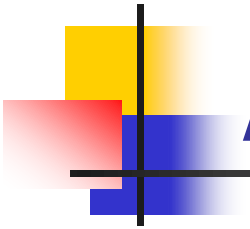
- Quick overview on the problem
  - The size of the state space tree is

$$1 + 4 + 4^2 + 4^3 + 4^4 = 341$$

  - There are 256 candidate solutions
    - From the start node to each of leaf nodes
  - The number of nodes to visit using DFS is 155
    - Count and see if it is truly 155 by yourself!!
  - But if we use a proper promising function, the number can be reduced to 27.

Figure 5.6 The queen in row 6 is being threatened in its left diagonal by queen in row 3 and in its right diagonal by the queen in row 2.



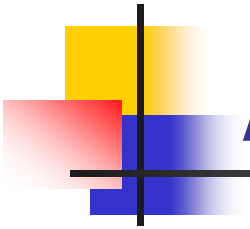


n Queens Problem

- Quick overview on the problem
  - The size of the state space tree is

$$1+n+n^2+n^3+\cdots+n^n=\frac{n^{n+1}-1}{n-1}$$

  - For, 8-queens problem, the tree has 19,173,961 nodes
  - There are n^n candidate solutions
  - The promising function is **col(i)-col(k) = i – k or k - i**



n Queens Problem (Cont'd)

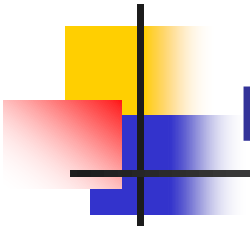
- But the size of the tree is of limited value
  - Because the backtracking may prune many unnecessary nodes
- Upper bound of the number of promising nodes
  - $1 + 8 + 8 \times 7 + 8 \times 7 \times 6 + \ldots + 8! = 109{,}601$
  - In general,

$$1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n!$$

  - Still, not a good idea, because
    - The promising function can be more accurately designed.
    - The number of non-promising nodes can be substantially larger.
- A straightforward way to determine the efficiency of the algorithm is to actually run.

Table 5.1 An illustrations of how much checking is saved by backtracking in the *n*-Queens problems*

n	Number of Nodes Checked by Algorithm 1†	Number of Candidate Solutions Checked by Algorithm 2‡	Number of Nodes Checked by Backtracking	Number of Nodes Found Promising by Backtracking
4	341	24	61	17
8	19,173,961	40,320	15,721	2057
12	9.73×10^{12}	4.79×10^{8}	1.01×10^{7}	8.56×10^{5}
14	1.20×10^{16}	8.72×10^{10}	3.78×10^{8}	2.74×10^{7}



Monte Carlo Algorithm

- To estimate the efficiency of a backtracking algorithm
- Probabilistic algorithm
  - The next instruction executed is determined randomly according to some probabilistic distribution
  - The same promising function must be used on all nodes at the same level in the state space tree.
  - Nodes at the same level in the tree must have the same number of children.
- *N* queens problem

$$1 + t_0 + m_0 t_1 + m_0 m_1 t_2 + \cdots m_0 m_1 \cdots m_{i-1} t_i + \cdots$$

Sum-of-Subsets Problem

- Input
  - *n* items with positive integers of w_i
  - A positive integer *W*
- Problem
  - Find all sets of $\sum_{A \subseteq S} w = W$
  - A node is non-promising if $weight + w_{i+1} > W$
  - A node is non-promising if $weight + total < W$
  - w_{i+1} is the lightest weight remaining when we are at level *i*

Figure 5.7 A state space tree for instances of the Sum-of-Subsets problems in which $n = 3$

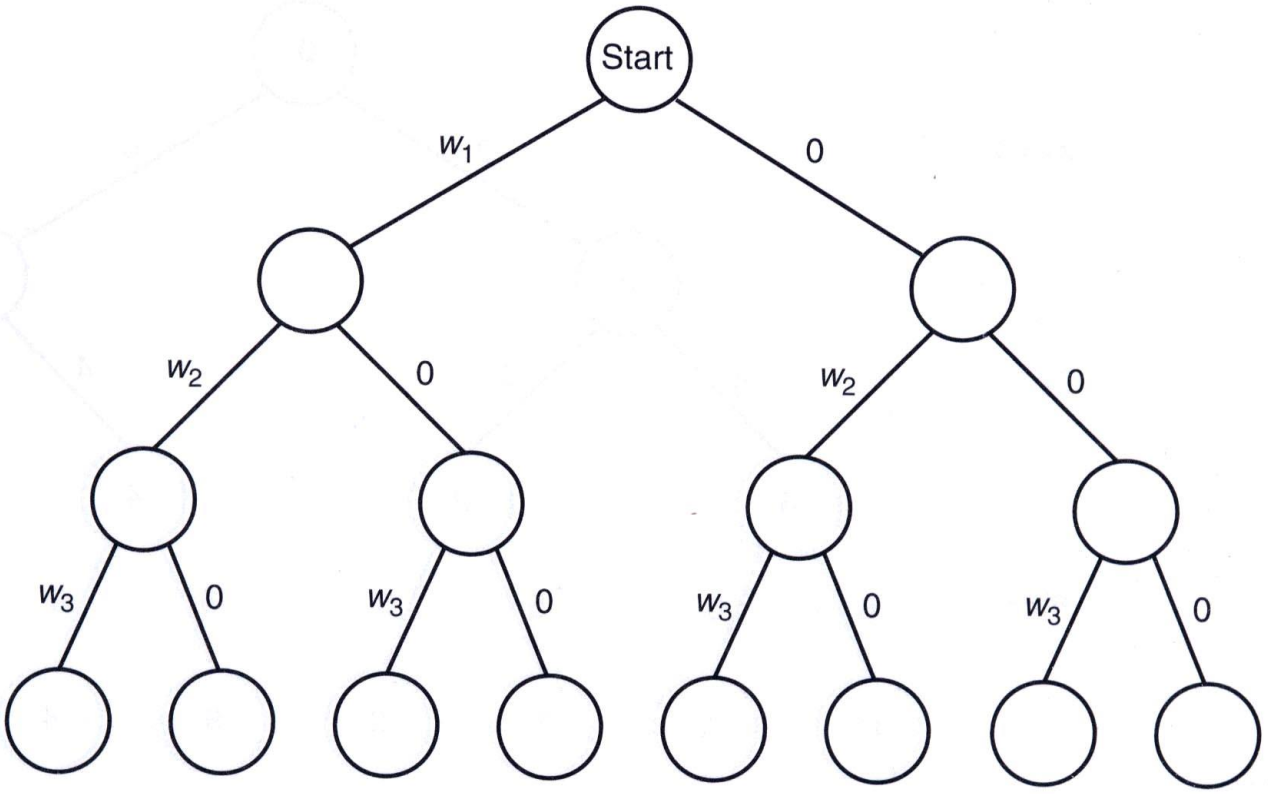


Figure 5.8 A state space tree for the Sum-of-Subsets problem for the instance in Example 5.3. Stored at each node is the total weight included up to that node. (W=6)

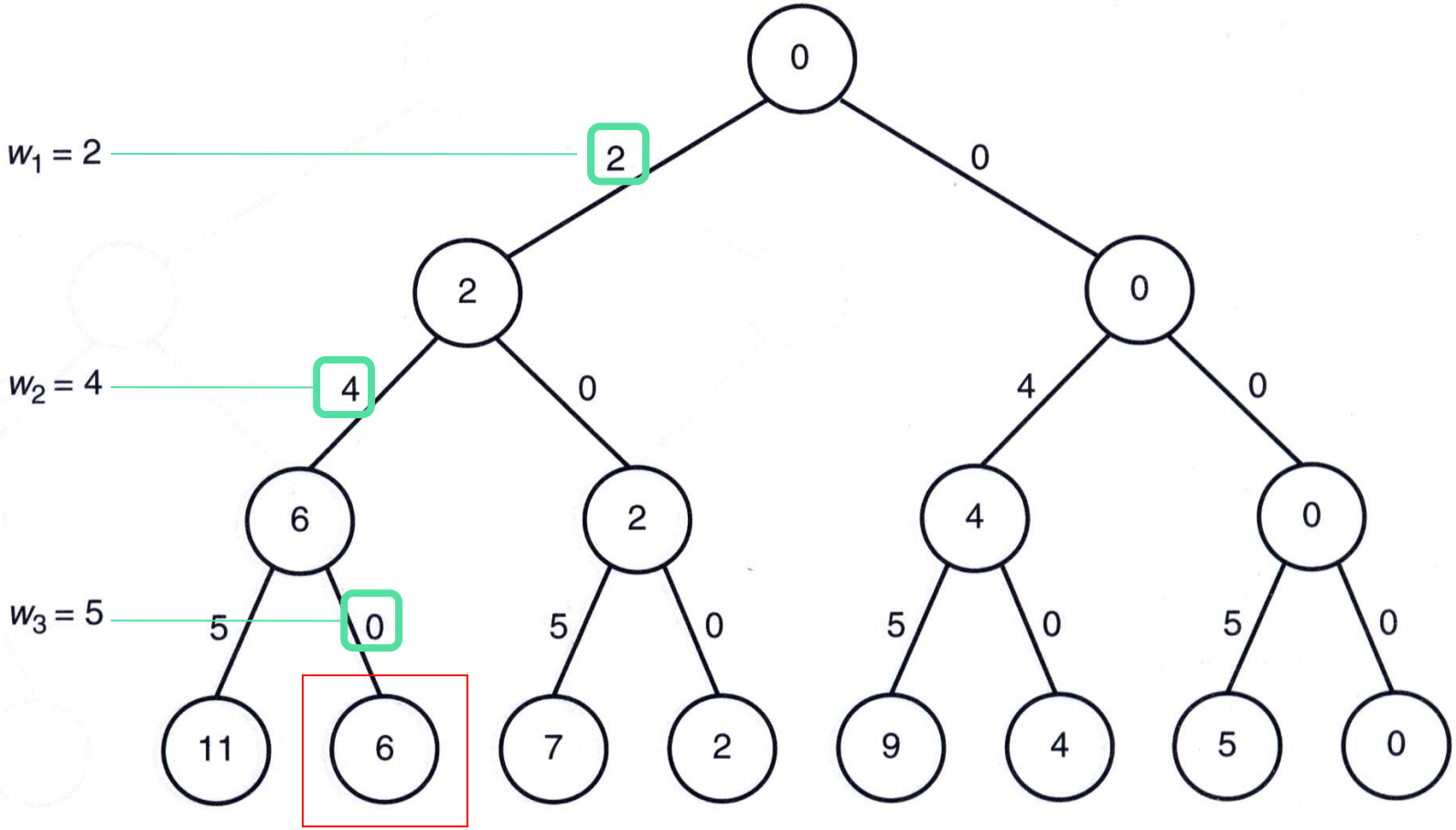
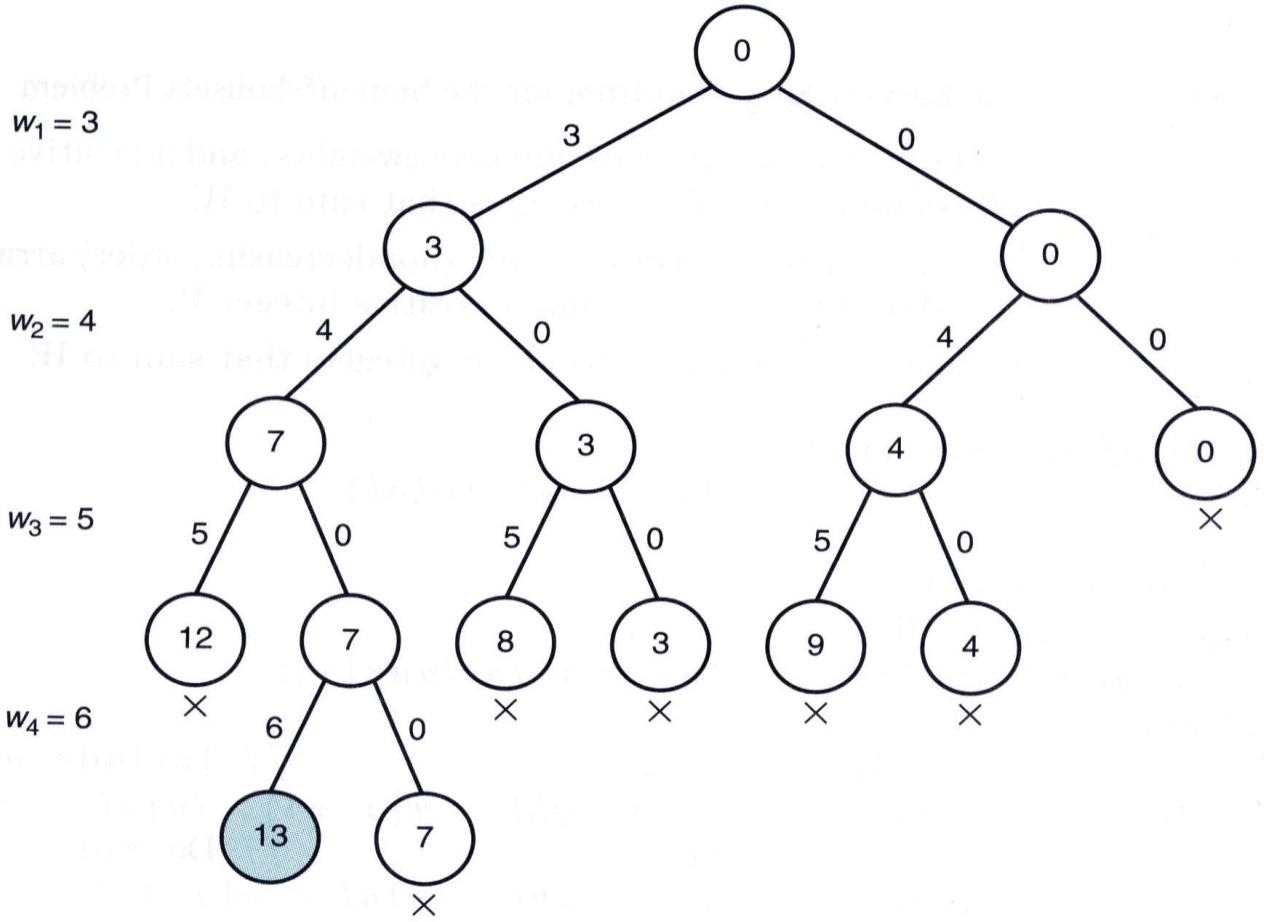
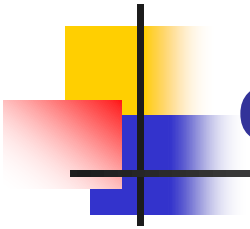


Figure 5.9 The pruned state space tree produced using backtracking in Example 5.4. Stored at each node is the total weight included up to that node. The only solution is found at the shaded node. Each nonpromising node is marked with a cross





Graph Coloring Problem

- Input
  - *n* and *m*, the number of vertices and colors
  - An undirected graph containing *n* vertices
  - A graph is called *planar*, if it can be drawn in a plane in such a way that no two edges cross each other
- Problem
  - Find all possible colorings of the graph using at most *m* colors
  - A node is non-promising if a vertex that is adjacent to the vertex being colored at the node has already been colored the color that is being used.

Figure 5.10 Graph for which there is no solution to the 2-Coloring problem. A solution to the 3- Coloring problems for this graph is shown in Example 5.5.

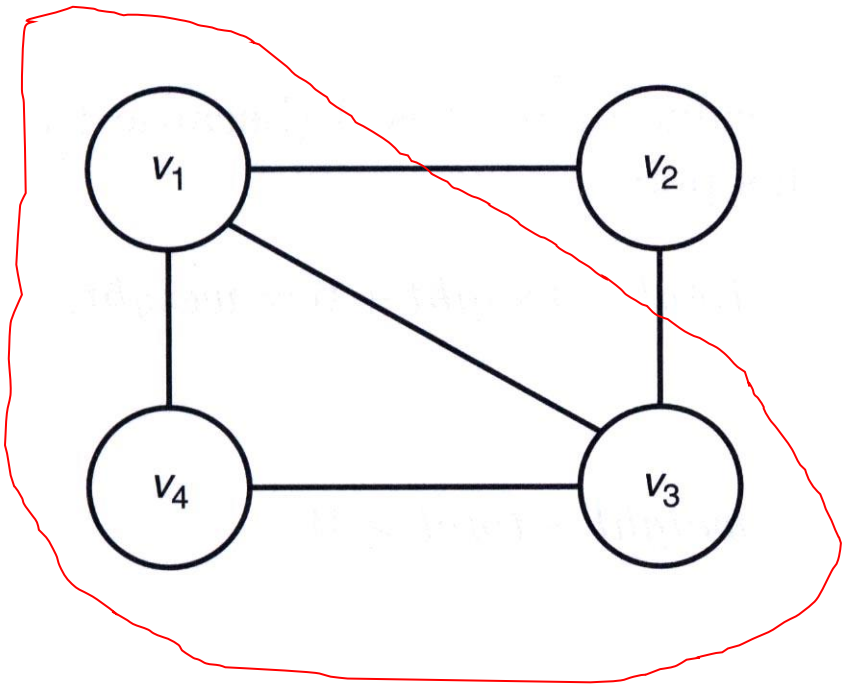


Figure 5.11 Map (top) and its planar graph representation (bottom).

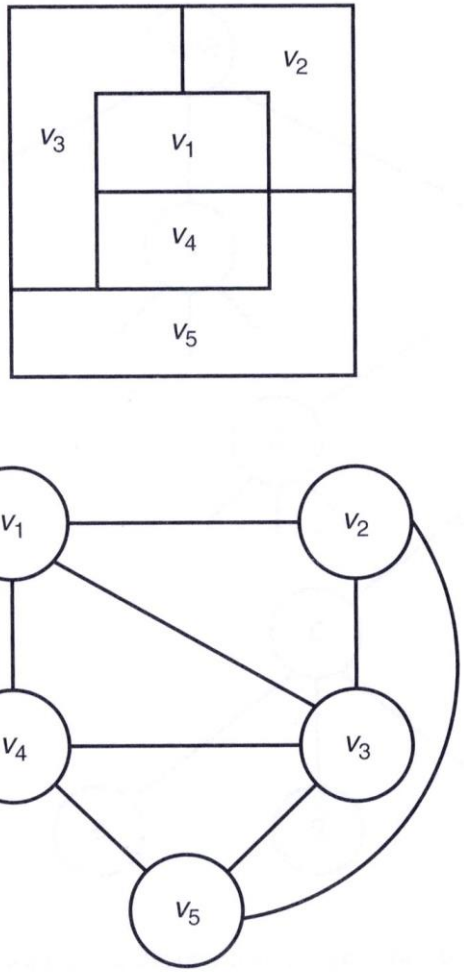
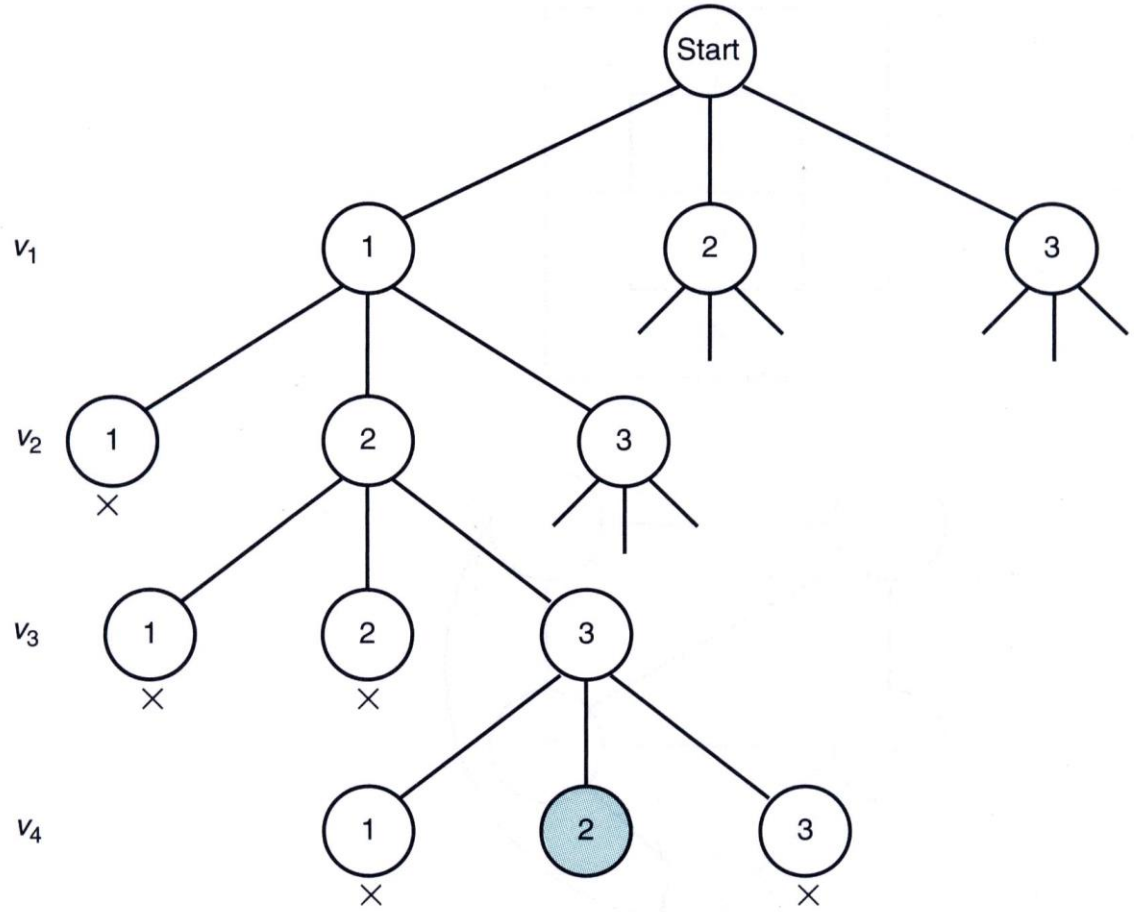
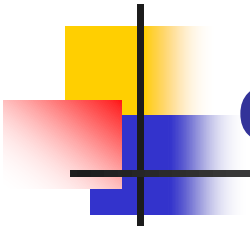


Figure 5.12 A portion of the pruned state tree produced using backtracking to do a 3-coloring of the graph in Figure 5.10. The first solution is found at the shaded node. Each nonpromising node is marked with a cross.



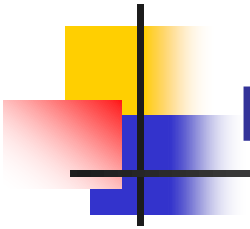


Graph Coloring Problem (Analysis)

- The total number of nodes in a state space tree is

$$1 + m + m^2 + \cdots + m^n = \frac{m^{n+1} - 1}{m - 1}$$

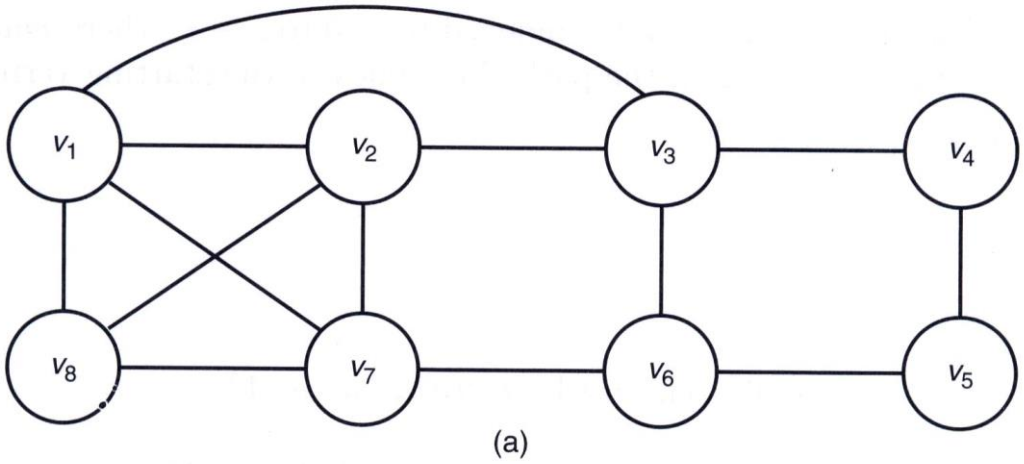
- The Monte Carlo technique can be applied to any backtracking algorithm to estimate the efficiency.

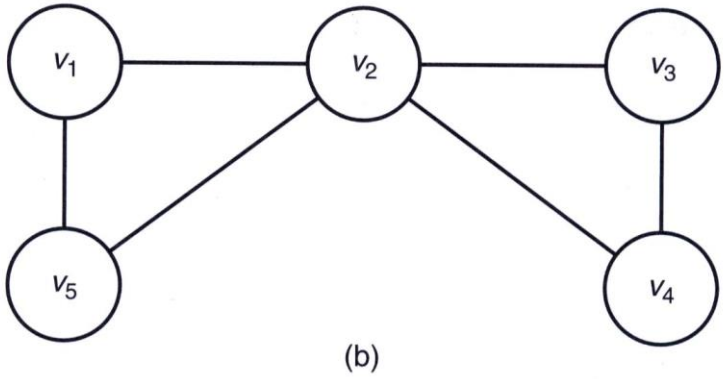


Hamiltonian Circuits Problem

- Input
  - An undirected graph containing n vertices.
  - A starting vertex v_i
- Problem
  - Find all paths that
    - start at a given vertex,
    - visit each vertex in the graph exactly once,
    - and end up at the starting vertex.
- *Think how to construct a state space tree!*

Figure 5.13 The graph in (a) contains the Hamiltonian Circuit [V1,V2,V8,V7,V6,V5,V4,V3,V2]; the graph in (b) contains no Hamiltonian Circuit.





Hamiltonian Circuits Problem (Cont'd)

- The total number of nodes in a state space tree is

$$1+(n-1)+(n-1)^2+\cdots+(n-1)^{(n-1)}=\frac{(n-1)^n-1}{n-2}$$