M16 - Algorithms and Data Structures

- **12 Coping with the Limitations of Algorithm Power**
  - introduce two algorithm design techniques —**backtracking** and **branch-and-bound**— that often make it possible to solve at least some large instances of difficult combinatorial problems.
    - Both strategies can be considered an improvement over exhaustive search. Unlike exhaustive search, they construct candidate solutions one component at a time and evaluate the partially constructed solutions
      - Both backtracking and branch-and-bound are based on the construction of a state-space tree whose nodes reflect specific choices made for a solution's components
  - **12.1 Backtracking**
    - Backtracking is a more intelligent variation of this approach (exhaustive-search).
    - the principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows
      - If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component.
      - If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered
        - It is convenient to implement this kind of processing by constructing a tree of choices being made, called the **state-space tree**.
          - A node in a state-space tree is said to be **promising** if it corresponds to a partially constructed solution that may still lead to a complete solution
            - otherwise, it is called **nonpromising**
            - Leaves represent either nonpromising dead ends or complete solutions found by the algorithm
              - In the majority of cases, a state-space tree for a backtracking algorithm is constructed in the manner of depth-first search
    - **n-Queens Problem**
      - place n queens on an n × n chessboard so that no two queens attack each other
        - it should be pointed out that a single solution to the n-queens problem for any n ≥ 4 can be found in **linear time**
    - **Hamiltonian Circuit Problem**
    - **Subset-Sum Problem**
    - In conclusion, three things on behalf of backtracking need to be said.
      1. it is typically applied to difficult combinatorial problems for which no efficient algorithms for finding exact solutions possibly exist
      2. unlike the exhaustive-search approach, which is doomed to be extremely slow for all instances of a problem, backtracking at least holds a hope for solving some instances of nontrivial sizes in an acceptable amount of time
         - This is especially true for optimization problems, for which the idea of backtracking can be further enhanced by evaluating the quality of partially constructed solutions
      3. even if backtracking does not eliminate any elements of a problem's state space and ends up generating all its elements, it provides a specific technique for doing so, which can be of value in its own right.
  - **12.2 Branch-and-Bound**
    - This idea (backtracking) can be strengthened further if we deal with an optimization problem.
      - An optimization problem seeks to minimize or maximize some objective function
      - Compared to backtracking, branch-and-bound requires two additional items:
        1. a way to provide, for every node of a state-space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partially constructed solution represented by the node
           - This bound should be a lower bound for a minimization problem and an upper bound for a maximization problem
             - If this information is available, we can compare a node's bound value with the value of the best solution seen so far
        2. the value of the best solution seen so far
    - In general, we terminate a search path at the current node in a state-space tree of a branch-and-bound algorithm for any one of the following three reasons:
      1. The value of the node's bound is not better than the value of the best solution seen so far
      2. The node represents no feasible solutions because the constraints of the problem are already violated.
      3. The subset of feasible solutions represented by the node consists of a single point t (and hence no further choices can be made)
    - **Assignment Problem**
      - assigning n people to n jobs so that the total cost of the assignment is as small as possible.
        - **best-first branch-and-bound**
    - **Knapsack Problem**
      - given n items of known weights wi and values vi, i = 1, 2, . . . , n, and a knapsack of capacity W, find the most valuable subset of the items that fit in the knapsack
    - **Traveling Salesman Problem**