Chapter 5

Dynamic Programming

Dynamic programming is a name, coined by Richard Bellman in 1955. Dynamic programming, as greedy method, is a powerful algorithm design technique that can be used when the solution to the problem may be viewed as the result of a sequence of decisions. In the greedy method we make irrevocable decisions one at a time, using a greedy criterion. However, in dynamic programming we examine the decision sequence to see whether an optimal decision sequence contains optimal decision subsequence.

When optimal decision sequences contain optimal decision subsequences, we can establish recurrence equations, called *dynamic-programming recurrence equations,* that enable us to solve the problem in an efficient way.

Dynamic programming is based on the principle of optimality (also coined by Bellman). The principle of optimality states that no matter whatever the initial state and initial decision are, the remaining decision sequence must constitute an optimal decision sequence with regard to the state resulting from the first decision. The principle implies that an optimal decision sequence is comprised of optimal decision subsequences. Since the principle of optimality may not hold for some formulations of some problems, it is necessary to verify that it does hold for the problem being solved. Dynamic programming cannot be applied when this principle does not hold.

The steps in a dynamic programming solution are:

- Verify that the principle of optimality holds
- Set up the dynamic-programming recurrence equations
- Solve the dynamic-programming recurrence equations for the value of the optimal solution.
- Perform a trace back step in which the solution itself is constructed.

Dynamic programming differs from the greedy method since the greedy method produces only one feasible solution, which may or may not be optimal, while dynamic programming produces all possible sub-problems at most once, one of which guaranteed to be optimal. Optimal solutions to sub-problems are retained in a table, thereby avoiding the work of recomputing the answer every time a sub-problem is encountered

The divide and conquer principle solve a large problem, by breaking it up into smaller problems which can be solved independently. In dynamic programming this principle is carried to an extreme: when we don't know exactly which smaller problems to solve, we simply solve them all, then store the answers away in a table to be used later in solving larger problems. Care is to be taken to avoid recomputing previously computed values, otherwise the recursive program will have prohibitive complexity. In some cases, the solution can be improved and in other cases, the dynamic programming technique is the best approach.

Two difficulties may arise in any application of dynamic programming:

1. It may not always be possible to combine the solutions of smaller problems to form the solution of a larger one.

2. The number of small problems to solve may be un-acceptably large.

There is no characterized precisely which problems can be effectively solved with dynamic programming; there are many hard problems for which it does not seen to be applicable, as well as many easy problems for which it is less efficient than standard algorithms.

5.1 MULTI STAGE GRAPHS

A multistage graph G = (V, E) is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$. In addition, if <u, v> is an edge in E, then $u \in V_i$ and $v \in V_{i+1}$ for some i, $1 \leq i < k$.

Let the vertex 's' is the source, and 't' the sink. Let c (i, j) be the cost of edge <i, j>. The cost of a path from 's' to 't' is the sum of the costs of the edges on the path. The multistage graph problem is to find a minimum cost path from 's' to 't'. Each set V_i defines a stage in the graph. Because of the constraints on E, every path from 's' to 't' starts in stage 1, goes to stage 2, then to stage 3, then to stage 4, and so on, and eventually terminates in stage k.

A dynamic programming formulation for a k-stage graph problem is obtained by first noticing that every s to t path is the result of a sequence of k – 2 decisions. The i^{th} decision involves determining which vertex in v_{i+1}, $1 \leq i \leq k - 2$, is to be on the path. Let c (i, j) be the cost of the path from source to destination. Then using the forward approach, we obtain:

$$\text{cost}(i, j) = \min_{\substack{l \,\varepsilon\, V_{i+1} \\ <j,\, l> \,\varepsilon\, E}} \{c(j, l) + \text{cost}(i + 1, l)\}$$

ALGORITHM:

```
Algorithm Fgraph (G, k, n, p)
// The input is a k-stage graph G = (V, E) with n vertices
// indexed in order or stages. E is a set of edges and c [i, j]
// is the cost of (i, j). p [1 : k] is a minimum cost path.
{
        cost [n] := 0.0;
        for j:= n - 1 to 1 step – 1 do
        {                                        // compute cost [j]
                let r be a vertex such that (j, r) is an edge
                of G and c [j, r] + cost [r] is minimum;
                cost [j] := c [j, r] + cost [r];
                d [j] := r:
        }
        p [1] := 1; p [k]  := n;                 // Find a minimum cost path.
        for j := 2 to k - 1 do p [j] := d [p [j - 1]];
}
```

The multistage graph problem can also be solved using the backward approach. Let bp(i, j) be a minimum cost path from vertex s to j vertex in V_i. Let Bcost(i, j) be the cost of bp(i, j). From the backward approach we obtain:

$$\text{Bcost}(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j\rangle \in E}} \{ \text{Bcost}(i-1, l) + c(l, j)\}$$

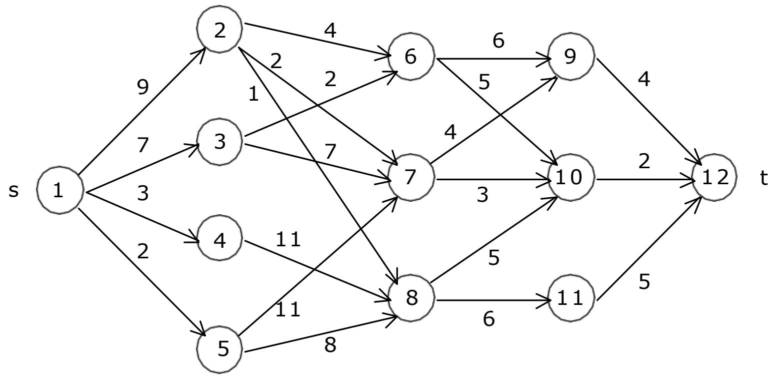
```
Algorithm Bgraph (G, k, n, p)
// Same function as Fgraph
{
        Bcost [1] := 0.0;
        for j := 2 to n do
        {                                            // Compute Bcost [j].
                Let r be such that (r, j) is an edge of
                G and Bcost [r] + c [r, j] is minimum;
                Bcost [j] := Bcost [r] + c [r, j];
                D [j] := r;
        }                                            //find a minimum cost path
        p [1] := 1; p [k] := n;
        for j:= k - 1 to 2 do p [j] := d [p [j + 1]];
}
```

Complexity Analysis:

The complexity analysis of the algorithm is fairly straightforward. Here, if G has $|E|$ edges, then the time for the first for loop is $\Phi(|V| + |E|)$.

EXAMPLE 1:

Find the minimum cost path from s to t in the multistage graph of five stages shown below. Do this first using forward approach and then using backward approach.



FORWARD APPROACH:

We use the following equation to find the minimum cost path from s to t:

$$\text{cost}(i, j) = \min_{l \in V_{i+1}} \{c(j, l) + \text{cost}(i + 1, l)\}$$

$$\text{cost}(1, 1) = \min_{\langle j, l\rangle \,\varepsilon\, E} \{c(1, 2) + \text{cost}(2, 2), c(1, 3) + \text{cost}(2, 3), c(1, 4) + \text{cost}(2,4), c(1, 5) + \text{cost}(2, 5)\}$$
$$= \min \{9 + \text{cost}(2, 2), 7 + \text{cost}(2, 3), 3 + \text{cost}(2, 4), 2 + \text{cost}(2, 5)\}$$

Now first starting with,

cost (2, 2) = min{c (2, 6) + cost (3, 6), c (2, 7) + cost (3, 7), c (2, 8) + cost (3,8)}
= min {4 + cost (3, 6), 2 + cost (3, 7), 1 + cost (3, 8)}

---

cost (3, 6) = min {c (6, 9) + cost (4, 9), c (6, 10) + cost (4, 10)}
= min {6 + cost (4, 9), 5 + cost (4, 10)}

cost (4, 9) = min {c (9, 12) + cost (5, 12)} = min {4 + 0) = 4

cost (4, 10) = min {c (10, 12) + cost (5, 12)} = 2

Therefore, cost (3, 6) = min {6 + 4, 5 + 2} = 7

---

cost (3, 7) = min {c (7, 9) + cost (4, 9) , c (7, 10) + cost (4, 10)}
= min {4 + cost (4, 9), 3 + cost (4, 10)}

cost (4, 9) = min {c (9, 12) + cost (5, 12)} = min {4 + 0} = 4

Cost (4, 10) = min {c (10, 2) + cost (5, 12)} = min {2 + 0} = 2

Therefore, cost (3, 7) = min {4 + 4, 3 + 2} = min {8, 5} = 5

---

cost (3, 8) = min {c (8, 10) + cost (4, 10), c (8, 11) + cost (4, 11)}
= min {5 + cost (4, 10), 6 + cost (4 + 11)}

cost (4, 11) = min {c (11, 12) + cost (5, 12)} = 5

Therefore, cost (3, 8) = min {5 + 2, 6 + 5} = min {7, 11} = 7

Therefore, cost (2, 2) = min {4 + 7, 2 + 5, 1 + 7} = min {11, 7, 8} = 7

Therefore, cost (2, 3) = min {c (3, 6) + cost (3, 6), c (3, 7) + cost (3, 7)}
= min {2 + cost (3, 6), 7 + cost (3, 7)}
= min {2 + 7, 7 + 5} = min {9, 12} = 9

cost (2, 4) = min {c (4, 8) + cost (3, 8)} = min {11 + 7} = 18
cost (2, 5) = min {c (5, 7) + cost (3, 7), c (5, 8) + cost (3, 8)}
= min {11 + 5, 8 + 7} = min {16, 15} = 15

Therefore, cost (1, 1) = min {9 + 7, 7 + 9, 3 + 18, 2 + 15}
= min {16, 16, 21, 17} = 16

The minimum cost path is 16.

The path is $$\begin{tikzcd} 1 \arrow[r, dash] & 2 \arrow[r, dash] & 7 \arrow[r, dash] & 10 \arrow[r, dash] & 12 \end{tikzcd}$$

or

$$\begin{tikzcd} 1 \arrow[r, dash] & 3 \arrow[r, dash] & 6 \arrow[r, dash] & 10 \arrow[r, dash] & 12 \end{tikzcd}$$

BACKWARD APPROACH:

We use the following equation to find the minimum cost path from t to s:

$$\text{Bcost}(i, J) = \min_{\substack{l \in v_{i-1} \\ <l, j> \in E}} \{\text{Bcost}(i-1, l) + c(l, J)\}$$

Bcost (5, 12) = min {Bcost (4, 9) + c (9, 12), Bcost (4, 10) + c (10, 12),
Bcost (4, 11) + c (11, 12)}
= min {Bcost (4, 9) + 4, Bcost (4, 10) + 2, Bcost (4, 11) + 5}

Bcost (4, 9) = min {Bcost (3, 6) + c (6, 9), Bcost (3, 7) + c (7, 9)}
= min {Bcost (3, 6) + 6, Bcost (3, 7) + 4}

Bcost (3, 6) = min {Bcost (2, 2) + c (2, 6), Bcost (2, 3) + c (3, 6)}
= min {Bcost (2, 2) + 4, Bcost (2, 3) + 2}

Bcost (2, 2) = min {Bcost (1, 1) + c (1, 2)} = min {0 + 9} = 9

Bcost (2, 3) = min {Bcost (1, 1) + c (1, 3)} = min {0 + 7} = 7

Bcost (3, 6) = min {9 + 4, 7 + 2} = min {13, 9} = 9

Bcost (3, 7) = min {Bcost (2, 2) + c (2, 7), Bcost (2, 3) + c (3, 7),
Bcost (2, 5) + c (5, 7)}

Bcost (2, 5) = min {Bcost (1, 1) + c (1, 5)} = 2

Bcost (3, 7) = min {9 + 2, 7 + 7, 2 + 11} = min {11, 14, 13} = 11

Bcost (4, 9) = min {9 + 6, 11 + 4} = min {15, 15} = 15

Bcost (4, 10) = min {Bcost (3, 6) + c (6, 10), Bcost (3, 7) + c (7, 10),
Bcost (3, 8) + c (8, 10)}

Bcost (3, 8) = min {Bcost (2, 2) + c (2, 8), Bcost (2, 4) + c (4, 8),
Bcost (2, 5) + c (5, 8)}
Bcost (2, 4) = min {Bcost (1, 1) + c (1, 4)} = 3

Bcost (3, 8) = min {9 + 1, 3 + 11, 2 + 8} = min {10, 14, 10} = 10

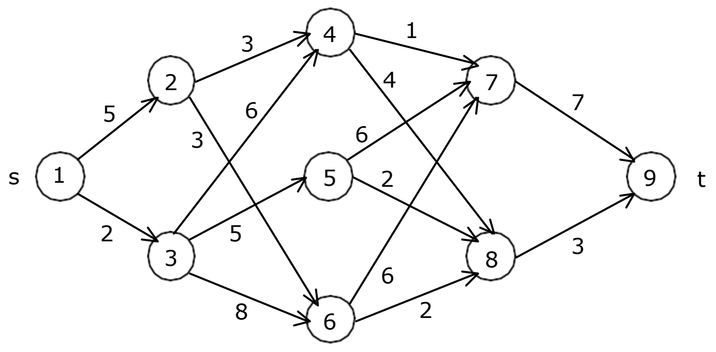
Bcost (4, 10) = min {9 + 5, 11 + 3, 10 + 5} = min {14, 14, 15) = 14

Bcost (4, 11) = min {Bcost (3, 8) + c (8, 11)} = min {Bcost (3, 8) + 6}
= min {10 + 6} = 16

Bcost (5, 12) = min {15 + 4, 14 + 2, 16 + 5} = min {19, 16, 21} = 16.

EXAMPLE 2:

Find the minimum cost path from s to t in the multistage graph of five stages shown below. Do this first using forward approach and then using backward approach.



SOLUTION:

FORWARD APPROACH:

$$cost\ (i, J) = \min_{\substack{l \in V_{i+1} \\ <J, l> \in E}} \{c\ (j, l) + cost\ (i + 1, l)\}$$

cost (1, 1) = min {c (1, 2) + cost (2, 2), c (1, 3) + cost (2, 3)}
= min {5 + cost (2, 2), 2 + cost (2, 3)}

cost (2, 2) = min {c (2, 4) + cost (3, 4), c (2, 6) + cost (3, 6)}
= min {3+ cost (3, 4), 3 + cost (3, 6)}

cost (3, 4) = min {c (4, 7) + cost (4, 7), c (4, 8) + cost (4, 8)}
= min {(1 + cost (4, 7), 4 + cost (4, 8)}

cost (4, 7) = min {c (7, 9) + cost (5, 9)} = min {7 + 0) = 7

cost (4, 8) = min {c (8, 9) + cost (5, 9)} = 3

Therefore, cost (3, 4) = min {8, 7} = 7

cost (3, 6) = min {c (6, 7) + cost (4, 7), c (6, 8) + cost (4, 8)}
= min {6 + cost (4, 7), 2 + cost (4, 8)} = min {6 + 7, 2 + 3} = 5

Therefore, cost (2, 2) = min {10, 8} = 8

cost (2, 3) = min {c (3, 4) + cost (3, 4), c (3, 5) + cost (3, 5), c (3, 6) + cost (3,6)}

cost (3, 5) = min {c (5, 7) + cost (4, 7), c (5, 8) + cost (4, 8)}= min {6 + 7, 2 + 3} = 5

Therefore, cost (2, 3) = min {13, 10, 13} = 10

cost (1, 1) = min {5 + 8, 2 + 10} = min {13, 12} = 12

BACKWARD APPROACH:

$$\text{Bcost}(i, J) = \min_{\substack{l \in v_{i-1} \\ <l, j> \in E}} \{\text{Bcost}(i - 1, l) = c(l, J)\}$$

Bcost (5, 9) = min {Bcost (4, 7) + c (7, 9), Bcost (4, 8) + c (8, 9)}
= min {Bcost (4, 7) + 7, Bcost (4, 8) + 3}

Bcost (4, 7) = min {Bcost (3, 4) + c (4, 7), Bcost (3, 5) + c (5, 7), Bcost (3, 6) + c (6, 7)}
= min {Bcost (3, 4) + 1, Bcost (3, 5) + 6, Bcost (3, 6) + 6}

Bcost (3, 4) = min {Bcost (2, 2) + c (2, 4), Bcost (2, 3) + c (3, 4)}
= min {Bcost (2, 2) + 3, Bcost (2, 3) + 6}

Bcost (2, 2) = min {Bcost (1, 1) + c (1, 2)} = min {0 + 5} = 5

Bcost (2, 3) = min (Bcost (1, 1) + c (1, 3)} = min {0 + 2} = 2

Therefore, Bcost (3, 4) = min {5 + 3, 2 + 6} = min {8, 8} = 8

Bcost (3, 5) = min {Bcost (2, 3) + c (3, 5)} = min {2 + 5} = 7

Bcost (3, 6) = min {Bcost (2, 2) + c (2, 6), Bcost (2, 3) + c (3, 6)}
= min {5 + 5, 2 + 8} = 10

Therefore, Bcost (4, 7) = min {8 + 1, 7 + 6, 10 + 6} = 9

Bcost (4, 8) = min {Bcost (3, 4) + c (4, 8), Bcost (3, 5) + c (5, 8), Bcost (3, 6) + c (6, 8)}
= min {8 + 4, 7 + 2, 10 + 2} = 9

Therefore, Bcost (5, 9) = min {9 + 7, 9 + 3} = 12

All pairs shortest paths

In the all pairs shortest path problem, we are to find a shortest path between every pair of vertices in a directed graph G. That is, for every pair of vertices (i, j), we are to find a shortest path from i to j as well as one from j to i. These two paths are the same when G is undirected.

When no edge has a negative length, the all-pairs shortest path problem may be solved by using Dijkstra's greedy single source algorithm n times, once with each of the n vertices as the source vertex.

The all pairs shortest path problem is to determine a matrix A such that A (i, j) is the length of a shortest path from i to j. The matrix A can be obtained by solving n single-source problems using the algorithm shortest Paths. Since each application of this procedure requires O (n^2) time, the matrix A can be obtained in O (n^3) time.

The dynamic programming solution, called Floyd's algorithm, runs in O (n^3) time. Floyd's algorithm works even when the graph has negative length edges (provided there are no negative length cycles).

The shortest i to j path in G, i ≠ j originates at vertex i and goes through some intermediate vertices (possibly none) and terminates at vertex j. If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively. Otherwise, the i to j path is not of minimum length. So, the principle of optimality holds. Let A^k (i, j) represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain:

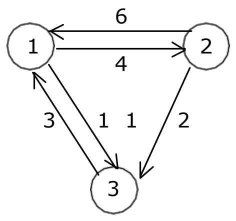
$$A^k (i, j) = \{\min \{\min_{1 \le k \le n} \{A^{k-1} (i, k) + A^{k-1} (k, j)\}, c (i, j)\}$$

Algorithm All Paths (Cost, A, n)
```
// cost [1:n, 1:n] is the cost adjacency matrix of a graph which
// n vertices; A [I, j] is the cost of a shortest path from vertex
// i to vertex j. cost [i, i] = 0.0, for 1 ≤ i ≤ n.
{
    for i := 1 to n do
        for j:= 1 to n do
            A [i, j] := cost  [i, j];                // copy cost into A.
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                A [i, j] := min (A [i, j], A [i, k] + A [k, j]);
}
```

Complexity Analysis: A Dynamic programming algorithm based on this recurrence involves in calculating n+1 matrices, each of size n x n. Therefore, the algorithm has a complexity of O (n^3).

Example 1:

Given a weighted digraph G = (V, E) with weight. Determine the length of the shortest path between all pairs of vertices in G. Here we assume that there are no cycles with zero or negative cost.



$$\text{Cost adjacency matrix } (A^0) = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

General formula: $\min_{1 \le k \le n} \{A^{k-1} (i, k) + A^{k-1} (k, j)\}, c (i, j)\}$

Solve the problem for different values of k = 1, 2 and 3

Step 1: Solving the equation for, k = 1;

$A^1 (1, 1) = \min \{(A^o (1, 1) + A^o (1, 1)), c (1, 1)\} = \min \{0 + 0, 0\} = 0$

$A^1 (1, 2) = \min \{(A^o (1, 1) + A^o (1, 2)), c (1, 2)\} = \min \{(0 + 4), 4\} = 4$

$A^1 (1, 3) = \min \{(A^o (1, 1) + A^o (1, 3)), c (1, 3)\} = \min \{(0 + 11), 11\} = 11$

$A^1 (2, 1) = \min \{(A^o (2, 1) + A^o (1, 1)), c (2, 1)\} = \min \{(6 + 0), 6\} = 6$

$A^1 (2, 2) = \min \{(A^o (2, 1) + A^o (1, 2)), c (2, 2)\} = \min \{(6 + 4), 0)\} = 0$

$A^1 (2, 3) = \min \{(A^o (2, 1) + A^o (1, 3)), c (2, 3)\} = \min \{(6 + 11), 2\} = 2$

$A^1 (3, 1) = \min \{(A^o (3, 1) + A^o (1, 1)), c (3, 1)\} = \min \{(3 + 0), 3\} = 3$

$A^1 (3, 2) = \min \{(A^o (3, 1) + A^o (1, 2)), c (3, 2)\} = \min \{(3 + 4), \propto\} = 7$

$A^1 (3, 3) = \min \{(A^o (3, 1) + A^o (1, 3)), c (3, 3)\} = \min \{(3 + 11), 0\} = 0$

$$A^{(1)} = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

Step 2: Solving the equation for, K = 2;

$A^2 (1, 1) = \min \{(A^1 (1, 2) + A^1 (2, 1), c (1, 1)\} = \min \{(4 + 6), 0\} = 0$

$A^2 (1, 2) = \min \{(A^1 (1, 2) + A^1 (2, 2), c (1, 2)\} = \min \{(4 + 0), 4\} = 4$

$A^2 (1, 3) = \min \{(A^1 (1, 2) + A^1 (2, 3), c (1, 3)\} = \min \{(4 + 2), 11\} = 6$

$A^2 (2, 1) = \min \{(A (2, 2) + A (2, 1), c (2, 1)\} = \min \{(0 + 6), 6\} = 6$

$A^2 (2, 2) = \min \{(A (2, 2) + A (2, 2), c (2, 2)\} = \min \{(0 + 0), 0\} = 0$

$A^2 (2, 3) = \min \{(A (2, 2) + A (2, 3), c (2, 3)\} = \min \{(0 + 2), 2\} = 2$

$A^2 (3, 1) = \min \{(A (3, 2) + A (2, 1), c (3, 1)\} = \min \{(7 + 6), 3\} = 3$

$A^2 (3, 2) = \min \{(A (3, 2) + A (2, 2), c (3, 2)\} = \min \{(7 + 0), 7\} = 7$

$A^2 (3, 3) = \min \{(A (3, 2) + A (2, 3), c (3, 3)\} = \min \{(7 + 2), 0\} = 0$

$$A^{(2)} = \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

Step 3: Solving the equation for, k = 3;

$A^3 (1, 1) = \min \{A^2 (1, 3) + A^2 (3, 1), c (1, 1)\} = \min \{(6 + 3), 0\} = 0$

$A^3 (1, 2) = \min \{A^2 (1, 3) + A^2 (3, 2), c (1, 2)\} = \min \{(6 + 7), 4\} = 4$

$A^3 (1, 3) = \min \{A^2 (1, 3) + A^2 (3, 3), c (1, 3)\} = \min \{(6 + 0), 6\} = 6$

$A^3 (2, 1) = \min \{A^2 (2, 3) + A^2 (3, 1), c (2, 1)\} = \min \{(2 + 3), 6\} = 5$

$A^3 (2, 2) = \min \{A^2 (2, 3) + A^2 (3, 2), c (2, 2)\} = \min \{(2 + 7), 0\} = 0$

$A^3 (2, 3) = \min \{A^2 (2, 3) + A^2 (3, 3), c (2, 3)\} = \min \{(2 + 0), 2\} = 2$

$A^3 (3, 1) = \min \{A^2 (3, 3) + A^2 (3, 1), c (3, 1)\} = \min \{(0 + 3), 3\} = 3$

$A^3 (3, 2) = \min \{A^2 (3, 3) + A^2 (3, 2), c (3, 2)\} = \min \{(0 + 7), 7\} = 7$

$A^3(3, 3) = \min \{A^2(3, 3) + A^2(3, 3), c(3, 3)\} = \min \{(0 + 0), 0\} = 0$

$$A^{(3)} = \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

TRAVELLING SALESPERSON PROBLEM

Let G = (V, E) be a directed graph with edge costs C_{ij}. The variable c_{ij} is defined such that $c_{ij} > 0$ for all I and j and $c_{ij} = \alpha$ if $< i, j> \notin E$. Let $|V| = n$ and assume $n > 1$. A tour of G is a directed simple cycle that includes every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The traveling sales person problem is to find a tour of minimum cost. The tour is to be a simple path that starts and ends at vertex 1.

Let g (i, S) be the length of shortest path starting at vertex i, going through all vertices in S, and terminating at vertex 1. The function g (1, V – {1}) is the length of an optimal salesperson tour. From the principal of optimality it follows that:

$$g(1, V - \{1\}) = \min_{2 \leq k \leq n} \{c_{1k} + g(k, V - \{1, k\})\} \qquad -- \qquad 1$$

Generalizing equation 1, we obtain (for $i \notin S$)

$$g(i, S) = \min_{j \in S} \{c_{ij} + g(i, S - \{j\})\} \qquad -- \qquad 2$$

The Equation can be solved for g (1, V – 1}) if we know g (k, V – {1, k}) for all choices of k.

Complexity Analysis:

For each value of |S| there are n – 1 choices for i. The number of distinct sets S of size k not including 1 and i is $\binom{n-2}{k}$.

Hence, the total number of g (i, S)'s to be computed before computing g (1, V – {1}) is:

$$\sum_{k=0}^{n-1} (n-1)\binom{n-2}{k}$$

To calculate this sum, we use the binominal theorem:

$$(n - 1) \left[\binom{(n-2)}{0} + \binom{(n-2)}{1} + \binom{(n-2)}{2} + ---- + \binom{(n-2)}{(n-2)}\right]$$

According to the binominal theorem:

$$\left[\binom{(n-2)}{0} + \binom{(n-2)}{1} + \binom{(n-2)}{2} + ---- + \binom{(n-2)}{(n-2)}\right] = 2^{n-2}$$

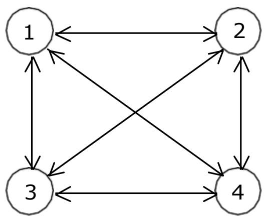
Therefore,

$$\sum_{k=0}^{n-1} (n-1)\binom{n-2}{k} = (n-1)\,2^{n-2}$$

This is Φ (n 2^{n-2}), so there are exponential number of calculate. Calculating one g (i, S) require finding the minimum of at most n quantities. Therefore, the entire algorithm is Φ (n^2 2^{n-2}). This is better than enumerating all n! different tours to find the best one. So, we have traded on exponential growth for a much smaller exponential growth. The most serious drawback of this dynamic programming solution is the space needed, which is O (n 2^n). This is too large even for modest values of n.

Example 1:

For the following graph find minimum cost tour for the traveling salesperson problem:



The cost adjacency matrix = $\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$

Let us start the tour from vertex 1:

$$g(1, V - \{1\}) = \min_{2 \le k \le n} \{c_{1k} + g(k, V - \{1, K\})\} \qquad - \qquad (1)$$

More generally writing:

$$g(i, s) = \min \{c_{ij} + g(J, s - \{J\})\} \qquad - \qquad (2)$$

Clearly, g (i, Φ) = c_{i1} , 1 ≤ i ≤ n. So,

$g(2, \Phi) = C_{21} = 5$

$g(3, \Phi) = C_{31} = 6$

$g(4, \Phi) = C_{41} = 8$

Using equation – (2) we obtain:

$g(1, \{2, 3, 4\}) = \min \{c_{12} + g(2, \{3, 4\}, c_{13} + g(3, \{2, 4\}), c_{14} + g(4, \{2, 3\})\}$

$g(2, \{3, 4\}) = \min \{c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\})\}$
$= \min \{9 + g(3, \{4\}), 10 + g(4, \{3\})\}$

$g(3, \{4\}) = \min \{c_{34} + g(4, \Phi)\} = 12 + 8 = 20$

$g(4, \{3\}) = \min \{c_{43} + g(3, \Phi)\} = 9 + 6 = 15$

Therefore, g (2, {3, 4}) = min {9 + 20, 10 + 15} = min {29, 25} = 25

g (3, {2, 4}) = min {(c_{32} + g (2, {4}), (c_{34} + g (4, {2})}

g (2, {4}) = min {c_{24} + g (4, Φ)} = 10 + 8 = 18

g (4, {2}) = min {c_{42} + g (2, Φ)} = 8 + 5 = 13

Therefore, g (3, {2, 4}) = min {13 + 18, 12 + 13} = min {41, 25} = 25

g (4, {2, 3}) = min {c_{42} + g (2, {3}), c_{43} + g (3, {2})}

g (2, {3}) = min {c_{23} + g (3, Φ} = 9 + 6 = 15

g (3, {2}) = min {c_{32} + g (2, Φ} = 13 + 5 = 18

Therefore, g (4, {2, 3}) = min {8 + 15, 9 + 18} = min {23, 27} = 23

g (1, {2, 3, 4}) = min {c_{12} + g (2, {3, 4}), c_{13} + g (3, {2, 4}), c_{14} + g (4, {2, 3})}
= min {10 + 25, 15 + 25, 20 + 23} = min {35, 40, 43} = 35

The optimal tour for the graph has length = 35

The optimal tour is: 1, 2, 4, 3, 1.

OPTIMAL BINARY SEARCH TREE

Let us assume that the given set of identifiers is {a_1, . . . , a_n} with $a_1 < a_2 < . . . . < a_n$. Let p (i) be the probability with which we search for a_i. Let q (i) be the probability that the identifier x being searched for is such that $a_i < x < a_{i+1}$, $0 \leq i \leq n$ (assume $a_0 = -\infty$ and $a_{n+1} = +\infty$). We have to arrange the identifiers in a binary search tree in a way that minimizes the expected total access time.
In a binary search tree, the number of comparisons needed to access an element at depth 'd' is d + 1, so if 'a_i' is placed at depth 'd_i', then we want to minimize:

$$\sum_{i=1}^{n} P_i \ (1 + d_i) \ .$$

Let P (i) be the probability with which we shall be searching for 'a_i'. Let Q (i) be the probability of an un-successful search. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

The expected cost contribution for the internal node for 'a_i' is:

$$P(i) \ * \ level \ (a_i) \ .$$

Unsuccessful search terminate with I = 0 (i.e at an external node). Hence the cost contribution for this node is:

$$Q \ (i) \ * \ \text{level} \ ((E_i) - 1)$$

The expected cost of binary search tree is:

$$\sum_{i=1}^{n} P(i) * level\ (a_i) + \sum_{i=0}^{n} Q(i) * level\ ((E_i) - 1)$$

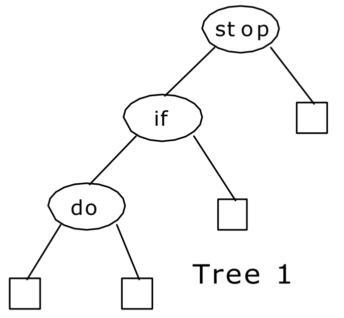
Given a fixed set of identifiers, we wish to create a binary search tree organization. We may expect different binary search trees for the same identifier set to have different performance characteristics.

The computation of each of these c(i, j)'s requires us to find the minimum of m quantities. Hence, each such c(i, j) can be computed in time O(m). The total time for all c(i, j)'s with j – i = m is therefore O(nm – m^2).

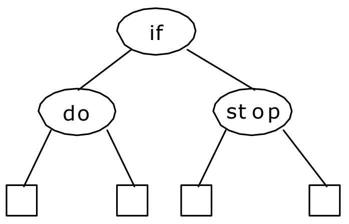
The total time to evaluate all the c(i, j)'s and r(i, j)'s is therefore:

$$\sum_{1 \le m \le n} \left(nm - m^2\right) = O\left(n^3\right)$$

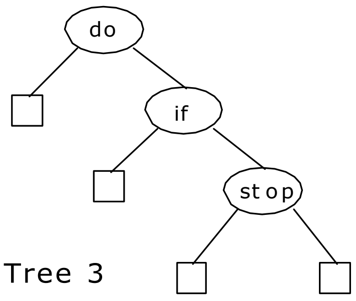
Example 1: The possible binary search trees for the identifier set (a_1, a_2, a_3) = (do, if, stop) are as follows. Given the equal probabilities p (i) = Q (i) = 1/7 for all i, we have:



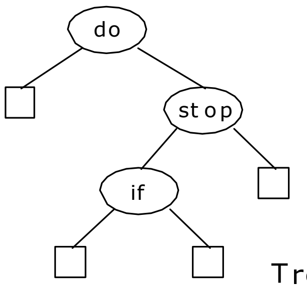
Tree 1



Tree 2



Tree 3



Tree 4

$$\text{Cost (tree \# 1)} = \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3\right) + \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3 + \frac{1}{7} x\ 3\right)$$

$$= \frac{1+2+3}{7} + \frac{1+2+3+3}{7} = \frac{6+9}{7} = \frac{15}{7}$$

$$\text{Cost (tree \# 2)} = \left(\frac{1}{7} x 1 + \frac{1}{7} x 2 + \frac{1}{7} x 2\right) + \left(\frac{1}{7} x 2 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 2 + \frac{1}{7} x 2\right)$$

$$= \frac{1+2+2}{7} + \frac{2+2+2+2}{7} = \frac{5+8}{7} = \frac{13}{7}$$

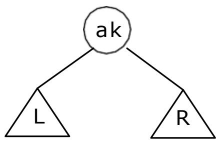
$$\text{Cost (tree \# 3)} = \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3\right) + \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3 + \frac{1}{7} x\ 3\right)$$

$$= \frac{1+2+3}{7} + \frac{1+2+3+3}{7} = \frac{6+9}{7} = \frac{15}{7}$$

$$\text{Cost (tree \# 4)} = \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3\right) + \left(\frac{1}{7} x\ 1 + \frac{1}{7} x\ 2 + \frac{1}{7} x\ 3 + \frac{1}{7} x\ 3\right)$$

$$= \frac{1+2+3}{7} + \frac{1+2+3+3}{7} = \frac{6+9}{7} = \frac{15}{7}$$

Huffman coding tree solved by a greedy algorithm has a limitation of having the data only at the leaves and it must not preserve the property that all nodes to the left of the root have keys, which are less etc. Construction of an optimal binary search tree is harder, because the data is not constrained to appear only at the leaves, and also because the tree must satisfy the binary search tree property and it must preserve the property that all nodes to the left of the root have keys, which are less.
A dynamic programming solution to the problem of obtaining an optimal binary search tree can be viewed by constructing a tree as a result of sequence of decisions by holding the principle of optimality. A possible approach to this is to make a decision as which of the a_i's be arraigned to the root node at 'T'. If we choose 'a_k' then is clear that the internal nodes for a_1, a_2, ... a_{k-1} as well as the external nodes for the classes E_0, E_1, ... E_{k-1} will lie in the left sub tree, L, of the root. The remaining nodes will be in the right subtree, R. The structure of an optimal binary search tree is:



$$\text{Cost (L)} = \sum_{i=1}^{K} P(i) * level\ (a_i) + \sum_{i=0}^{K} Q(i) * (level(E_i) - 1)$$

$$\text{Cost (R)} = \sum_{i=K}^{n} P(i) * level\ (a_i) + \sum_{i=K}^{n} Q(i) * (level(E_i) - 1)$$

The C (i, J) can be computed as:

$$C(i, J) = \min_{i<k\leq J} \{C(i, k-1) + C(k, J) + P(K) + w(i, K-1) + w(K, J)\}$$

$$= \min_{i<k\leq J} \{C(i, K-1) + C(K, J)\} + w\ (i, J) \qquad -- \qquad (1)$$

$$\text{Where } W(i, J) = P(J) + Q(J) + w\ (i, J-1) \qquad -- \qquad (2)$$

Initially C (i, i) = 0 and w (i, i) = Q (i) for $0 \leq i \leq n$.

Equation (1) may be solved for C (0, n) by first computing all C (i, J) such that J - i = 1
Next, we can compute all C (i, J) such that J - i = 2, Then all C (i, J) with J - i = 3 and so on.

C (i, J) is the cost of the optimal binary search tree 'T_{ij}' during computation we record the root R (i, J) of each tree 'T_{ij}'. Then an optimal binary search tree may be constructed from these R (i, J). R (i, J) is the value of 'K' that minimizes equation (1).

We solve the problem by knowing W (i, i+1), C (i, i+1) and R (i, i+1), $0 \leq i < 4$; Knowing W (i, i+2), C (i, i+2) and R (i, i+2), $0 \leq i < 3$ and repeating until W (0, n), C (0, n) and R (0, n) are obtained.

The results are tabulated to recover the actual tree.

Example 1:

Let n = 4, and (a_1, a_2, a_3, a_4) = (do, if, need, while) Let P (1: 4) = (3, 3, 1, 1) and Q (0: 4) = (2, 3, 1, 1, 1)

Solution:

Table for recording W (i, j), C (i, j) and R (i, j):

Column Row	**0**	**1**	**2**	**3**	**4**
0	2, 0, 0	3, 0, 0	1, 0, 0	1, 0, 0,	1, 0, 0
1	8, 8, 1	7, 7, 2	3, 3, 3	3, 3, 4	
2	12, 19, 1	9, 12, 2	5, 8, 3		
3	14, 25, 2	11, 19, 2			
4	16, 32, 2				

This computation is carried out row-wise from row 0 to row 4. Initially, W (i, i) = Q (i) and C (i, i) = 0 and R (i, i) = 0, $0 \leq i < 4$.

Solving for C (0, n):

First, computing all C (i, j) such that j - i = 1; j = i + 1 and as $0 \leq i < 4$; i = 0, 1, 2 and 3; $i < k \leq J$. Start with i = 0; so j = 1; as $i < k \leq j$, so the possible value for k = 1

W (0, 1) = P (1) + Q (1) + W (0, 0) = 3 + 3 + 2 = 8
C (0, 1) = W (0, 1) + min {C (0, 0) + C (1, 1)} = 8
R (0, 1) = 1 (value of 'K' that is minimum in the above equation).

Next with i = 1; so j = 2; as $i < k \leq j$, so the possible value for k = 2

W (1, 2) = P (2) + Q (2) + W (1, 1) = 3 + 1 + 3 = 7
C (1, 2) = W (1, 2) + min {C (1, 1) + C (2, 2)} = 7
R (1, 2) = 2

Next with i = 2; so j = 3; as $i < k \leq j$, so the possible value for k = 3

W (2, 3) = P (3) + Q (3) + W (2, 2) = 1 + 1 + 1 = 3
C (2, 3) = W (2, 3) + min {C (2, 2) + C (3, 3)} = 3 + [(0 + 0)] = 3
R (2, 3) = 3

Next with i = 3; so j = 4; as i < k ≤ j, so the possible value for k = 4

W (3, 4) = P (4) + Q (4) + W (3, 3) = 1 + 1 + 1 = 3
C (3, 4) = W (3, 4) + min {[C (3, 3) + C (4, 4)]} = 3 + [(0 + 0)] = 3
R (3, 4) = 4

Second, Computing all C (i, j) such that j - i = 2; j = i + 2 and as 0 ≤ i < 3; i = 0, 1, 2; i < k ≤ J. Start with i = 0; so j = 2; as i < k ≤ J, so the possible values for k = 1 and 2.

W (0, 2) = P (2) + Q (2) + W (0, 1) = 3 + 1 + 8 = 12
C (0, 2) = W (0, 2) + min {(C (0, 0) + C (1, 2)), (C (0, 1) + C (2, 2))}
= 12 + min {(0 + 7, 8 + 0)} = 19
R (0, 2) = 1
Next, with i = 1; so j = 3; as i < k ≤ j, so the possible value for k = 2 and 3.

W (1, 3) = P (3) + Q (3) + W (1, 2) = 1 + 1+ 7 = 9
C (1, 3) = W (1, 3) + min {[C (1, 1) + C (2, 3)], [C (1, 2) + C (3, 3)]}
= W (1, 3) + min {(0 + 3), (7 + 0)} = 9 + 3 = 12
R (1, 3) = 2

Next, with i = 2; so j = 4; as i < k ≤ j, so the possible value for k = 3 and 4.

W (2, 4) = P (4) + Q (4) + W (2, 3) = 1 + 1 + 3 = 5
C (2, 4) = W (2, 4) + min {[C (2, 2) + C (3, 4)], [C (2, 3) + C (4, 4)]
= 5 + min {(0 + 3), (3 + 0)} = 5 + 3 = 8
R (2, 4) = 3

Third, Computing all C (i, j) such that J - i = 3; j = i + 3 and as 0 ≤ i < 2; i = 0, 1; i < k ≤ J. Start with i = 0; so j = 3; as i < k ≤ j, so the possible values for k = 1, 2 and 3.

W (0, 3) = P (3) + Q (3) + W (0, 2) = 1 + 1 + 12 = 14
C (0, 3) = W (0, 3) + min {[C (0, 0) + C (1, 3)], [C (0, 1) + C (2, 3)],
[C (0, 2) + C (3, 3)]}
= 14 + min {(0 + 12), (8 + 3), (19 + 0)} = 14 + 11 = 25
R (0, 3) = 2

Start with i = 1; so j = 4; as i < k ≤ j, so the possible values for k = 2, 3 and 4.

W (1, 4) = P (4) + Q (4) + W (1, 3) = 1 + 1 + 9 = 11
C (1, 4) = W (1, 4) + min {[C (1, 1) + C (2, 4)], [C (1, 2) + C (3, 4)],
[C (1, 3) + C (4, 4)]}
= 11 + min {(0 + 8), (7 + 3), (12 + 0)} = 11 + 8 = 19
R (1, 4) = 2

Fourth, Computing all C (i, j) such that j - i = 4; j = i + 4 and as 0 ≤ i < 1; i = 0; i < k ≤ J.

Start with i = 0; so j = 4; as i < k ≤ j, so the possible values for k = 1, 2, 3 and 4.

W (0, 4) = P (4) + Q (4) + W (0, 3) = 1 + 1 + 14 = 16
C (0, 4) = W (0, 4) + min {[C (0, 0) + C (1, 4)], [C (0, 1) + C (2, 4)], [C (0, 2) + C (3, 4)], [C (0, 3) + C (4, 4)]}
= 16 + min [0 + 19, 8 + 8, 19+3, 25+0] = 16 + 16 = 32
R (0, 4) = 2

From the table we see that C (0, 4) = 32 is the minimum cost of a binary search tree for (a_1, a_2, a_3, a_4). The root of the tree 'T_{04}' is 'a_2'.

Hence the left sub tree is 'T_{01}' and right sub tree is T_{24}. The root of 'T_{01}' is 'a_1' and the root of 'T_{24}' is a_3.

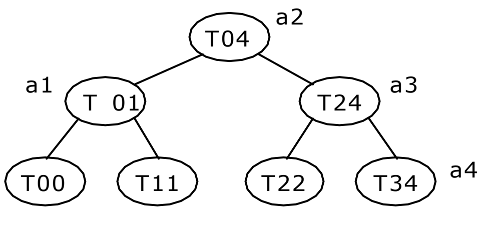
The left and right sub trees for 'T_{01}' are 'T_{00}' and 'T_{11}' respectively. The root of T_{01} is 'a_1'

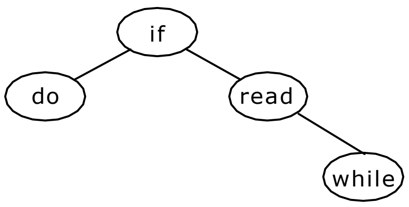
The left and right sub trees for T_{24} are T_{22} and T_{34} respectively.

The root of T_{24} is 'a_3'.

The root of T_{22} is null

The root of T_{34} is a_4.





Example 2:

Consider four elements a_1, a_2, a_3 and a_4 with Q_0 = 1/8, Q_1 = 3/16, Q_2 = Q_3 = Q_4 = 1/16 and p_1 = 1/4, p_2 = 1/8, p_3 = p_4 =1/16. Construct an optimal binary search tree. Solving for C (0, n):

First, computing all C (i, j) such that j - i = 1; j = i + 1 and as $0 \leq i < 4$; i = 0, 1, 2 and 3; $i < k \leq J$. Start with i = 0; so j = 1; as $i < k \leq j$, so the possible value for k = 1

W (0, 1) = P (1) + Q (1) + W (0, 0) = 4 + 3 + 2 = 9
C (0, 1) = W (0, 1) + min {C (0, 0) + C (1, 1)} = 9 + [(0 + 0)] = 9
R (0, 1) = 1 (value of 'K' that is minimum in the above equation).

Next with i = 1; so j = 2; as $i < k \leq j$, so the possible value for k = 2

W (1, 2) = P (2) + Q (2) + W (1, 1) = 2 + 1 + 3 = 6
C (1, 2) = W (1, 2) + min {C (1, 1) + C (2, 2)} = 6 + [(0 + 0)] = 6
R (1, 2) = 2

Next with i = 2; so j = 3; as $i < k \leq j$, so the possible value for k = 3

W (2, 3) = P (3) + Q (3) + W (2, 2) = 1 + 1 + 1 = 3
C (2, 3) = W (2, 3) + min {C (2, 2) + C (3, 3)} = 3 + [(0 + 0)] = 3

R (2, 3) = 3

Next with i = 3; so j = 4; as i < k ≤ j, so the possible value for k = 4

W (3, 4) = P (4) + Q (4) + W (3, 3) = 1 + 1 + 1 = 3
C (3, 4) = W (3, 4) + min {[C (3, 3) + C (4, 4)]} = 3 + [(0 + 0)] = 3
R (3, 4) = 4

Second, Computing all C (i, j) such that j - i = 2; j = i + 2 and as 0 ≤ i < 3; i = 0, 1, 2; i < k ≤ J

Start with i = 0; so j = 2; as i < k ≤ j, so the possible values for k = 1 and 2.

W (0, 2) = P (2) + Q (2) + W (0, 1) = 2 + 1 + 9 = 12
C (0, 2) = W (0, 2) + min {(C (0, 0) + C (1, 2)), (C (0, 1) + C (2, 2))}
= 12 + min {(0 + 6, 9 + 0)} = 12 + 6 = 18
R (0, 2) = 1
Next, with i = 1; so j = 3; as i < k ≤ j, so the possible value for k = 2 and 3.

W (1, 3) = P (3) + Q (3) + W (1, 2) = 1 + 1+ 6 = 8
C (1, 3) = W (1, 3) + min {[C (1, 1) + C (2, 3)], [C (1, 2) + C (3, 3)]}
= W (1, 3) + min {(0 + 3), (6 + 0)} = 8 + 3 = 11
R (1, 3) = 2

Next, with i = 2; so j = 4; as i < k ≤ j, so the possible value for k = 3 and 4.

W (2, 4) = P (4) + Q (4) + W (2, 3) = 1 + 1 + 3 = 5
C (2, 4) = W (2, 4) + min {[C (2, 2) + C (3, 4)], [C (2, 3) + C (4, 4)]
= 5 + min {(0 + 3), (3 + 0)} = 5 + 3 = 8
R (2, 4) = 3

Third, Computing all C (i, j) such that J - i = 3; j = i + 3 and as 0 ≤ i < 2; i = 0, 1; i < k ≤ J. Start with i = 0; so j = 3; as i < k ≤ j, so the possible values for k = 1, 2 and 3.

W (0, 3) = P (3) + Q (3) + W (0, 2) = 1 + 1 + 12 = 14
C (0, 3) = W (0, 3) + min {[C (0, 0) + C (1, 3)], [C (0, 1) + C (2, 3)],
[C (0, 2) + C (3, 3)]}
= 14 + min {(0 + 11), (9 + 3), (18 + 0)} = 14 + 11 = 25
R (0, 3) = 1

Start with i = 1; so j = 4; as i < k ≤ j, so the possible values for k = 2, 3 and 4.

W (1, 4) = P (4) + Q (4) + W (1, 3) = 1 + 1 + 8 = 10
C (1, 4) = W (1, 4) + min {[C (1, 1) + C (2, 4)], [C (1, 2) + C (3, 4)],
[C (1, 3) + C (4, 4)]}
= 10 + min {(0 + 8), (6 + 3), (11 + 0)} = 10 + 8 = 18
R (1, 4) = 2

Fourth, Computing all C (i, j) such that J - i = 4; j = i + 4 and as 0 ≤ i < 1; i = 0; i < k ≤ J. Start with i = 0; so j = 4; as i < k ≤ j, so the possible values for k = 1, 2, 3 and 4.

W (0, 4) = P (4) + Q (4) + W (0, 3) = 1 + 1 + 14 = 16
C (0, 4) = W (0, 4) + min {[C (0, 0) + C (1, 4)], [C (0, 1) + C (2, 4)],
[C (0, 2) + C (3, 4)], [C (0, 3) + C (4, 4)]}

= 16 + min [0 + 18, 9 + 8, 18 + 3, 25 + 0] = 16 + 17 = 33
R (0, 4) = 2

Table for recording W (i, j), C (i, j) and R (i, j)

Column Row	**0**	**1**	**2**	**3**	**4**
0	2, 0, 0	1, 0, 0	1, 0, 0	1, 0, 0,	1, 0, 0
1	9, 9, 1	6, 6, 2	3, 3, 3	3, 3, 4	
2	12, 18, 1	8, 11, 2	5, 8, 3		
3	14, 25, 2	11, 18, 2			
4	16, 33, 2				

From the table we see that C (0, 4) = 33 is the minimum cost of a binary search tree for (a_1, a_2, a_3, a_4)

The root of the tree 'T_{04}' is 'a_2'.

Hence the left sub tree is 'T_{01}' and right sub tree is T_{24}. The root of 'T_{01}' is 'a_1' and the root of 'T_{24}' is a_3.

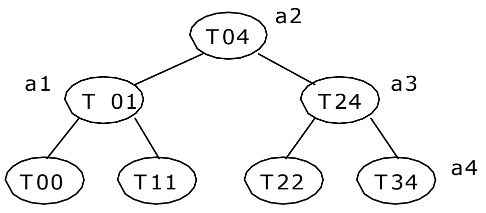
The left and right sub trees for 'T_{01}' are 'T_{00}' and 'T_{11}' respectively. The root of T_{01} is 'a_1'

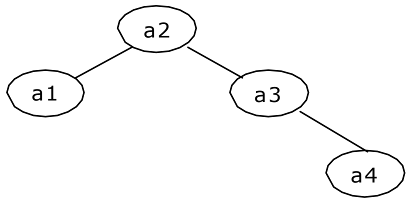
The left and right sub trees for T_{24} are T_{22} and T_{34} respectively.

The root of T_{24} is 'a_3'.

The root of T_{22} is null.

The root of T_{34} is a_4.





Example 3:

WORD	PROBABILITY
A	4
B	2
C	1
D	3
E	5
F	2
G	1

and all other elements have zero probability.

Solving c(0,n):

First computing all c(i, j) such that j- i = 1;j = i +1 and as 0 ≤ i < 7; i = 0, 1, 2, 3, 4, 5 and 6; i < k ≤ j. Start with i = 0 ; so j = 1; as i < k ≤ j, so the possible value for k = 1

W(0, 1) = P(1) + Q(1)+W(0, 0) = 4+0+0 = 4
C(0, 1) = W(0, 1)+ min {C (0, 0) + C(1, 1) }=4 + [(0 + 0)] = 4
R(0, 1) = 1

next with i = 1 ; so j = 2; as i < k ≤ j, so the possible value for k = 2

W(1, 2) = P(2) + Q(2)+W(1, 1) = 2+0+0 = 2
C(1, 2) = W(1, 2)+ min {C (1, 1) + C(2, 2) }=2 + [(0 + 0)] = 2
R(1, 2) = 2
next with i = 2 ; so j = 3; as i < k ≤ j, so the possible value for k = 3

W(2, 3) = P(3) + Q(3)+W(2, 2) = 1+0+0 = 1
C(2, 3) = W(2, 3)+ min {C (2, 2) + C(3, 3) }=1 + [(0 + 0)] = 1
R(2, 3) = 3

next with i = 3 ; so j = 4; as i < k ≤ j, so the possible value for k = 4

W(3, 4) = P(4) + Q(4)+W(3, 3) = 3+0+0 = 3
C(3, 4) = W(3, 4)+ min {C (3, 4) + C(4, 4) }=3 + [(0 + 0)] = 3
R(3, 4) = 4

next with i = 4 ; so j = 5; as i < k ≤ j, so the possible value for k = 5

W(4, 5) = P(5) + Q(5)+W(4, 4) = 5+0+0 = 5
C(4, 5) = W(4, 5)+ min {C (4, 4) + C(5, 5) }=5 + [(0 + 0)] = 5
R(4, 5) = 5

next with i = 5; so j = 6; as i < k ≤ j, so the possible value for k = 6

W(5, 6) = P(6) + Q(6)+W(5, 5) = 2+0+0 = 2
C(5, 6) = W(5, 6)+ min {C (5, 5) + C(6, 6) }=2 + [(0 + 0)] = 2
R(5, 6) = 6

next with i = 6; so j = 7; as i < k ≤ j, so the possible value for k = 7

W(6, 7) = P(7) + Q(7)+W(6, 6) = 1+0+0 = 1
C(6, 7) = W(6, 7)+ min {C (6, 6) + C(7, 7) }=1 + [(0 + 0)] = 1
R(6, 7) = 7

Second, computing all c(i, j) such that j - i = 2 ;j = i + 2 and as 0 ≤ i < 6; i = 0, 1, 2, 3, 4 and 5; i < k ≤ j; Start with i = 0 ; so j = 2; as i < k ≤ j, so the possible values for k = 1 and 2.

W(0, 2) = P(2) + Q(2)+W(0, 1) = 2 + 0 + 4 = 6
C(0, 2) = W(0, 2)+ min {C (0, 0) + C(1, 2) ,C(0, 1) + C(2, 2)}
= 6 +min{ 0 + 2, 4 + 0} = 8
R(0, 2) = 1

next with i = 1 ; so j = 3; as i < k ≤ j, so the possible values for k = 2 and 3.

W(1, 3) = P(3) + Q(3) +W(1, 2) = 1+ 0 + 2 = 3
C(1, 3) = W(1, 3)+ min {C (1, 1) + C(2,3) ,C(1, 2) + C(3, 3)}
= 3 +min{ 0 + 1, 2 + 0} = 4
R(1, 3) = 2

next with i = 2 ; so j = 4; as i < k ≤ j, so the possible values for k = 3 and 4.

W(2, 4) = P(4) + Q(4) +W(2, 3) = 3+ 0 + 1 = 4
C(2, 4) = W(2, 4)+ min {C (2, 2) + C(3,4) ,C(2, 3) + C(4, 4)}
= 4 +min{ 0 + 3, 1 + 0} = 5
R(2, 4) = 4
next with i = 3 ; so j = 5; as i < k ≤ j, so the possible values for k = 4 and 5.

W(3, 5) = P(5) + Q(5)+W(3, 4) = 5+ 0 + 3 =8
C(3, 5) = W(3, 5)+ min {C (3, 3) + C(4,5) ,C(3,4) + C(5, 5)}
= 8 +min{ 0 + 5, 3 + 0} = 11
R(3, 5) = 5
next with i = 4 ; so j = 6; as i < k ≤ j, so the possible values for k = 5 and 6.

W(4, 6) = P(6) + Q(6)+W(4, 5) = 2+ 0 + 5 = 7
C(4, 6) = W(4, 6)+ min {C (4, 4) + C(5,6) ,C(4, 5) + C(6, 6)}
= 7 +min{ 0 + 2, 5 + 0} = 9
R(4, 6) = 5

next with i = 5 ; so j = 7; as i < k ≤ j, so the possible values for k = 6 and 7.

W(5, 7) = P(7) + Q(7)+W(5, 6) = 1+ 0 + 2 = 3
C(5, 7) = W(5, 7)+ min {C (5, 5) + C(6,7) ,C(5, 6) + C(7, 7)}
= 3 +min{ 0 + 1, 2 + 0} = 4
R(5, 7) = 6

Third, computing all c(i, j) such that j – i = 3 ;j = i + 3 and as 0 ≤ i < 5 ; i = 0, 1, 2, 3, 4 and I < k ≤ j.

Start with i = 0 ; so j = 3; as i < k ≤ j, so the possible values for k = 1,2 and 3.

W(0, 3) = P(3) + Q(3)+W(0, 2) = 1+ 0 + 6 = 7
C(0, 3) = W(0, 3)+ min {C (0, 0) + C(1,3) ,C(0, 1) + C(2, 3),C(0, 2) + C(3, 3)}
= 7 +min{ 0 + 4, 4 + 1, 8 + 0} = 7
R(0, 3) = 1

next with i = 1 ; so j = 4; as i < k ≤ j, so the possible values for k = 2,3 and 4.

W(1, 4) = P(4) + Q(4)+W(1, 3) = 3+ 0 + 3 = 6
C(1, 4) = W(1, 4)+ min {C (1, 1) + C(2, 4) ,C(1, 2) + C(3, 4),C(1, 3) + C(4, 4)}
= 6 +min{ 0 + 5, 2 + 3, 4 + 0} = 10
R(1, 4) = 4

next with i = 2 ; so j = 5; as i < k ≤ j, so the possible values for k = 3, 4 and 5.

W(2, 5) = P(5) + Q(5)+W(2, 4) = 5+ 0 + 4 = 9
C(2, 5) = W(2, 5)+ min {C (2, 2) + C(3, 5) ,C(2, 3) + C(4, 5),C(2, 4) + C(5, 5)}
= 9 +min{ 0 + 11, 1 + 5 ,5 + 0} = 14
R(2, 5) = 5

next with i = 3 ; so j = 6; as i < k ≤ j, so the possible values for k = 4, 5 and 6.

W(3, 6) = P(6) + Q(6)+W(3, 5) = 2+ 0 + 8 = 10
C(3, 6) = W(3, 6)+ min {C (3, 3) + C(4, 6) ,C(3 ,4) + C(5, 6),C(3, 5) + C(6, 6)}
= 10 +min{ 0 + 9 , 3 + 2 ,11 + 0} = 15
R(3, 6) = 5

next with i = 4 ; so j = 7; as i < k ≤ j, so the possible values for k = 5, 6 and 7.

W(4, 7) = P(7) + Q(7)+W(4, 6) = 1+ 0 + 7 = 8
C(4, 7) = W(4, 7)+ min {C (4, 4) + C(5, 7) ,C(4 ,5) + C(6, 7),C(4, 6) + C(7, 7)}
= 8 +min{ 0 + 4 , 5 + 1 ,9 + 0} = 12
R(4, 7) = 5

Fourth, computing all c(i, j) such that j – i = 4 ;j = i + 4 and as 0 ≤ i < 4 ; i = 0, 1, 2, 3 for i < k ≤ j. Start with i = 0 ; so j = 4; as i < k ≤ j, so the possible values for k = 1,2 ,3 and 4.

W(0, 4) = P(4) + Q(4)+W(0, 3) = 3+ 0 + 7 = 10
C(0, 4) = W(0, 4)+ min {C (0, 0) + C(1,4) ,C(0, 1) + C(2, 4),C(0, 2) + C(3, 4),
C(0, 3) + C(4, 4)}
= 10 +min{ 0 + 10, 4 + 5,8 + 3,11 + 0} = 19
R(0, 4) = 2

next with i = 1 ; so j = 5; as i < k ≤ j, so the possible values for k = 2,3 ,4 and 5.

W(1, 5) = P(5) + Q(5)+W(1, 4) = 5+ 0 + 6 = 11
C(1, 5) = W(1, 5)+ min {C (1, 1) + C(2, 5) ,C(1, 2) + C(3, 5),C(1, 3) + C(4, 5)
C(1, 4) + C(5, 5)}
= 11 +min{ 0 + 14, 2 + 11,4 + 5,10 +0} = 20
R(1, 5) = 4

next with i = 2 ; so j = 6; as i < k ≤ j, so the possible values for k = 3,4,5 and 6.

W(2, 6) = P(6) + Q(6)+W(2, 5) = 2+ 0 + 9 = 11
C(2, 6) = W(2, 6)+ min {C (2, 2) + C(3, 6) ,C(2, 3) + C(4, 6),C(2, 4) + C(5, 6)
C(2, 5) + C (6, 6)} = 11 +min{ 0 + 15, 1 + 9 ,5 + 2,14 + 0} = 18
R(2, 6) = 5

next with i = 3 ; so j = 7; as i < k ≤ j, so the possible values for k = 4,5,6 and 7.

W(3, 7) = P(7) + Q(7)+W(3, 6) = 1+ 0 +11 = 12
C(3, 7) = W(3, 7)+ min {C (3, 3) + C(4, 7) ,C(3, 4) + C(5, 7),C(3, 5) + C(6, 7)
C(3, 6) + C (7, 7)} = 12 +min{ 0 + 12, 3 +4 ,11 +1,15 + 0} = 19
R(3, 7) = 5

Fifth, computing all c(i, j) such that j – i = 5; j = i + 5 and as 0 ≤ i < 3;
i = 0, 1, 2, i < k ≤ j. Start with i = 0 ; so j = 4; as i < k ≤ j, so the possible values for k = 1,2 ,3,4 and 5.

W(0, 5) = P(5) + Q(5)+W(0, 4) = 5+ 0 + 10 = 15
C(0, 5) = W(0, 5)+ min {C (0, 0) + C(1,5) ,C(0, 1) + C(2, 5),C(0, 2) + C(3, 5),
C(0, 3) + C(4, 5),C(0, 4) + C(5, 5)}
= 10 +min{ 0 + 20, 4 + 14, 8 + 11 ,19 + 0} = 28

R(0, 5) = 2

next with i = 1 ; so j = 6; as i < k ≤ j, so the possible values for k = 2, 3 ,4, 5 & 6.

W(1, 6) = P(6) + Q(6)+W(1, 5) = 2+ 0 + 11 = 13
C(1, 6) = W(1, 6)+ min {C (1, 1) + C(2, 6) ,C(1, 2) + C(3, 6),C(1, 3) + C(4, 6)
C(1, 4) + C(5, 6),C(1, 5)+C(6, 6)}
= 13 +min{ 0 + 18, 2 + 15, 4 + 9, 10 +2, 20 + 0} = 25
R(1, 6) = 5

next with i = 2 ; so j = 7; as i < k ≤ j, so the possible values for k = 3,4,5,6 and 7.

W(2, 7) = P(7) + Q(7)+W(2, 6) = 1+ 0 + 11 = 12
C(2, 7) = W(2, 7)+ min {C (2, 2) + C(3, 7) ,C(2, 3) + C(4, 7),C(2, 4) + C(5,7)
C(2, 5) + C (6, 7),C(2, 6) + C(7,7)}
= 12 +min{ 0 + 18, 1 + 12 , 5 + 4, 14 + 1, 18 + 0} = 21
R(2, 7) = 5

Sixth, computing all c(i, j) such that j – i = 6 ;j = i + 6 and as 0 ≤ i < 2 ; i = 0, 1
i < k ≤ j. Start with i = 0; so j = 6; as i < k ≤ j, so the possible values for k = 1, 2, 3, 4 5 & 6.
W(0, 6) = P(6) + Q(6)+W(0, 5) = 2+ 0 + 15 = 17
C(0, 6) = W(0,6)+ min {C (0, 0) + C(1,6) ,C(0, 1) + C(2, 6),C(0, 2) + C(3, 6),
C(0, 3) + C(4, 6),C(0, 4) + C(5, 6),C(0, 5) + C(6, 6)}
= 17 +min{ 0 + 25, 4 + 18, 8 + 15,19 + 2, 31 + 0} = 37
R(0, 6) = 4

next with i = 1 ; so j = 7; as i < k ≤ j, so the possible values for k = 2, 3, 4, 5, 6 and 7.

W(1, 7) = P(7) + Q(7)+W(1, 6) = 1+ 0 + 13 = 14
C(1, 7) = W(1, 7)+ min {C (1, 1) + C(2, 7) ,C(1, 2) + C(3, 7),C(1, 3) + C(4,7)
C(1, 4) + C(5, 7),C(1, 5)+C(6, 7),C(1, 6) +C(7, 7)}
= 14 +min{ 0 + 21, 2 + 18, 4 + 12, 10 + 4, 20 + 1, 21 + 0} = 28
R(1, 7) = 5

Seventh, computing all c(i, j) such that j – i = 7 ;j = i + 7 and as 0 ≤ i < 1 ; i = 0
i < k ≤ j. Start with i = 0 ; so j = 7; as i < k ≤ j, so the possible values for k = 1, 2, 3, 4, 5, 6 and 7.

W(0, 7) = P(7) + Q(7)+W(0, 6) = 1+ 0 + 17 = 18
C(0, 7) = W(0, 7)+ min {C (0, 0) + C(1, 7) ,C(0, 1) + C(2, 7),C(0, 2) + C(3,7),
C(0, 3) + C(4, 7),C(0, 4) + C(5, 6),C(0, 5) + C (6, 7),C(0, 6) + C(7, 7)}
= 18 +min{ 0 + 28, 4 + 21, 8 + 18,19 +4, 31 + 1, 37 + 0} = 41
R(0, 7) = 4

0/1 – KNAPSACK

We are given n objects and a knapsack. Each object i has a positive weight w_i and a positive value V_i. The knapsack can carry a weight not exceeding W. Fill the knapsack so that the value of objects in the knapsack is optimized.

A solution to the knapsack problem can be obtained by making a sequence of decisions on the variables $x_1, x_2, \ldots, x_n$. A decision on variable x_i involves determining which of the values 0 or 1 is to be assigned to it. Let us assume that decisions on the x_i are made in the order $x_n, x_{n-1}, \ldots . x_1$. Following a decision on x_n, we may be in one of two possible states: the capacity remaining in $m - w_n$ and a profit of p_n has accrued. It is clear that the remaining decisions $x_{n-1}, \ldots, x_1$ must be optimal with respect to the problem state resulting from the decision on x_n. Otherwise, $x_n, \ldots, x_1$ will not be optimal. Hence, the principal of optimality holds.

$$F_n(m) = \max \{f_{n-1}(m), f_{n-1}(m - w_n) + p_n\} \qquad -- \qquad 1$$

For arbitrary $f_i(y)$, $i > 0$, this equation generalizes to:

$$F_i(y) = \max \{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\} \qquad -- \qquad 2$$

Equation-2 can be solved for $f_n(m)$ by beginning with the knowledge $f_o(y) = 0$ for all y and $f_i(y) = -\propto$, $y < 0$. Then $f_1, f_2, \ldots f_n$ can be successively computed using equation–2.

When the w_i's are integer, we need to compute $f_i(y)$ for integer y, $0 \leq y \leq m$. Since $f_i(y) = -\propto$ for $y < 0$, these function values need not be computed explicitly. Since each f_i can be computed from $f_i - 1$ in $\mathbf{\Theta}(m)$ time, it takes $\mathbf{\Theta}(m\ n)$ time to compute f_n. When the wi's are real numbers, $f_i(y)$ is needed for real numbers y such that $0 \leq y \leq m$. So, f_i cannot be explicitly computed for all y in this range. Even when the w_i's are integer, the explicit $\mathbf{\Theta}(m\ n)$ computation of f_n may not be the most efficient computation. So, we explore **an alternative method for both cases.**

The $f_i(y)$ is an ascending step function; i.e., there are a finite number of y's, $0 = y_1 < y_2 < \ldots . < y_k$, such that $f_i(y_1) < f_i(y_2) < \ldots \ldots < f_i(y_k)$; $f_i(y) = -\propto$, $y < y_1$; $f_i(y) = f(y_k)$, $y \geq y_k$; and $f_i(y) = f_i(y_j)$, $y_j \leq y \leq y_{j+1}$. So, we need to compute only $f_i(y_j)$, $1 \leq j \leq k$. We use the ordered set $S^i = \{(f(y_j), y_j) \mid 1 \leq j \leq k\}$ to represent $f_i(y)$. Each number of S^i is a pair (P, W), where $P = f_i(y_j)$ and $W = y_j$. Notice that $S^0 = \{(0, 0)\}$. We can compute S^{i+1} from S_i by first computing:

$$S^i_1 = \{(P, W) \mid (P - p_i, W - w_i) \varepsilon S^i\}$$

Now, S^{i+1} can be computed by merging the pairs in S^i and S^i_1 together. Note that if S^{i+1} contains two pairs (P_j, W_j) and (P_k, W_k) with the property that $P_j \leq P_k$ and $W_j \geq W_k$, then the pair (P_j, W_j) can be discarded because of equation-2. Discarding or purging rules such as this one are also known as dominance rules. Dominated tuples get purged. In the above, (P_k, W_k) dominates (P_j, W_j).

Example 1:

Consider the knapsack instance $n = 3$, $(w_1, w_2, w_3) = (2, 3, 4)$, $(P_1, P_2, P_3) = (1, 2, 5)$ and $M = 6$.

Solution:

Initially, $f_o(x) = 0$, for all x and $f_i(x) = -\propto$ if $x < 0$.

$F_n(M) = \max \{f_{n-1}(M), f_{n-1}(M - w_n) + p_n\}$

$F_3(6) = \max (f_2(6), f_2(6 - 4) + 5\} = \max \{f_2(6), f_2(2) + 5\}$

$F_2(6) = \max (f_1(6), f_1(6 - 3) + 2\} = \max \{f_1(6), f_1(3) + 2\}$

$F_1(6) = \max(f_0(6), f_0(6-2)+1\} = \max\{0, 0+1\} = 1$

$F_1(3) = \max(f_0(3), f_0(3-2)+1\} = \max\{0, 0+1\} = 1$

Therefore, $F_2(6) = \max(1, 1+2\} = 3$

$F_2(2) = \max(f_1(2), f_1(2-3)+2\} = \max\{f_1(2), -\infty + 2\}$

$F_1(2) = \max(f_0(2), f_0(2-2)+1\} = \max\{0, 0+1\} = 1$

$F_2(2) = \max\{1, -\infty + 2\} = 1$

Finally, $f_3(6) = \max\{3, 1+5\} = 6$

Other Solution:

For the given data we have:

$S^0 = \{(0, 0)\}$; $S^0_1 = \{(1, 2)\}$

$S^1 = (S^0 \cup S^0_1) = \{(0, 0), (1, 2)\}$

$X - 2 = 0 \Rightarrow x = 2.$ $y - 3 = 0 \Rightarrow y = 3$
$X - 2 = 1 \Rightarrow x = 3.$ $y - 3 = 2 \Rightarrow y = 5$

$S^1_1 = \{(2, 3), (3, 5)\}$

$S^2 = (S^1 \cup S^1_1) = \{(0, 0), (1, 2), (2, 3), (3, 5)\}$

$X - 5 = 0 \Rightarrow x = 5.$ $y - 4 = 0 \Rightarrow y = 4$
$X - 5 = 1 \Rightarrow x = 6.$ $y - 4 = 2 \Rightarrow y = 6$
$X - 5 = 2 \Rightarrow x = 7.$ $y - 4 = 3 \Rightarrow y = 7$
$X - 5 = 3 \Rightarrow x = 8.$ $y - 4 = 5 \Rightarrow y = 9$

$S^2_1 = \{(5, 4), (6, 6), (7, 7), (8, 9)\}$

$S^3 = (S^2 \cup S^2_1) = \{(0, 0), (1, 2), (2, 3), (3, 5), (5, 4), (6, 6), (7, 7), (8, 9)\}$

By applying Dominance rule,

$S^3 = (S^2 \cup S^2_1) = \{(0, 0), (1, 2), (2, 3), (5, 4), (6, 6)\}$

From (6, 6) we can infer that the maximum Profit $\sum p_i x_i = 6$ and weight $\sum x_i w_i = 6$

Reliability Design

The problem is to design a system that is composed of several devices connected in series. Let r_i be the reliability of device D_i (that is r_i is the probability that device i will function properly) then the reliability of the entire system is $\Pi\ r_i$. Even if the individual devices are very reliable (the r_i's are very close to one), the reliability of the system may not be very good. For example, if $n = 10$ and $r_i = 0.99$, $i \leq i \leq 10$, then $\Pi\ r_i = .904$. Hence, it is desirable to duplicate devices. Multiply copies of the same device type are connected in parallel.

If stage i contains mi copies of device D_i. Then the probability that all m_i have a malfunction is $(1 - r_i)^{m_i}$. Hence the reliability of stage i becomes $1 - (1 - r_i)^{m_i}$.

The reliability of stage 'i' is given by a function $\phi_i\ (m_i)$.

Our problem is to use device duplication. This maximization is to be carried out under a cost constraint. Let c_i be the cost of each unit of device i and let c be the maximum allowable cost of the system being designed.

We wish to solve:

Maximize $\pi_{1 \leq i \leq n} \phi_i (m_i)$

Subject to $\sum_{1 \leq i \leq n} C_i m_i < C$

$m_i \geq 1$ and interger, $1 \leq i \leq n$

Assume each $C_i > 0$, each m_i must be in the range $1 \leq m_i \leq u_i$, where

$$u_i = \left\lfloor \left(C + C_i - \sum_{1}^{n} C_J \right) \Big/ C_i \right\rfloor$$

The upper bound u_i follows from the observation that $m_j \geq 1$

An optimal solution $m_1, m_2 \ldots\ldots m_n$ is the result of a sequence of decisions, one decision for each m_i.

Let $f_i(x)$ represent the maximum value of $\pi_{1 \leq j \leq i} \phi (m_J)$

Subject to the constrains:

$\sum_{1 \leq J \leq i} C_J m_J \leq x$ and $1 \leq m_j \leq u_J$, $1 \leq j \leq i$

The last decision made requires one to choose m_n from $\{1, 2, 3, \ldots\ldots u_n\}$

Once a value of m_n has been chosen, the remaining decisions must be such as to use the remaining funds $C - C_n m_n$ in an optimal way.

The principle of optimality holds on

$$f_n(C) = \max_{1 \leq m_n \leq u_n} \{\phi_n (m_n) f_{n-1} (C - C_n m_n)\}$$

for any $f_i\ (x_i)$, $i > 1$, this equation generalizes to

$$f_n(x) = \max_{1 \le m_i \le u_i} \{\phi_i(m_i)\, f_{i-1}(x - C_i m_i)\}$$

clearly, $f_0(x) = 1$ for all x, $0 \le x \le C$ and $f(x) = -\infty$ for all $x < 0$.

Let S^i consist of tuples of the form (f, x), where $f = f_i(x)$.

There is atmost one tuple for each different 'x', that result from a sequence of decisions on $m_1, m_2, \ldots m_n$. The dominance rule (f_1, x_1) dominate (f_2, x_2) if $f_1 \ge f_2$ and $x_1 \le x_2$. Hence, dominated tuples can be discarded from S^i.

Example 1:

Design a three stage system with device types D_1, D_2 and D_3. The costs are \$30, \$15 and \$20 respectively. The Cost of the system is to be no more than \$105. The reliability of each device is 0.9, 0.8 and 0.5 respectively.

Solution:

We assume that if if stage I has mi devices of type i in parallel, then $\phi_i(m_i) = 1 - (1-r_i)^{mi}$

Since, we can assume each $c_i > 0$, each m_i must be in the range $1 \le m_i \le u_i$. Where:

$$u_i = \left\lfloor \left(C + C_i - \sum_1^n C_J \right) \Big/ C_i \right\rfloor$$

Using the above equation compute u_1, u_2 and u_3.

$$u_1 = \frac{105 + 30 - (30 + 15 + 20)}{30} = \frac{70}{30} = 2$$

$$u_2 = \frac{105 + 15 - (30 + 15 + 20)}{15} = \frac{55}{15} = 3$$

$$u_3 = \frac{105 + 20 - (30 + 15 + 20)}{20} = \frac{60}{20} = 3$$

We use $S^i_j \rightarrow i$: *stage number and J: no. of devices in stage i* = m_i

$S^o = \{f_o(x), x\}$ *initially* $f_o(x) = 1$ *and* $x = 0$, *so*, $S^o = \{1, 0\}$

Compute S^1, S^2 and S^3 as follows:

S^1 = depends on u_1 value, as $u_1 = 2$, so

$$S^1 = \{S^1_1, S^1_2\}$$

S^2 = depends on u_2 value, as $u_2 = 3$, so

$$S^2 = \{S^2_1, S^2_2, S^2_3\}$$

S^3 = depends on u_3 value, as u_3 = 3, so

$$S^3 = \{S^3_1, S^3_2, S^3_3\}$$

Now find $S^1_1 = \{(f_1(x), x)\}$

$f_1(x) = \{\phi_1(1) f_o(\,), \phi_1(2) f_0(\,)\}$ With devices m_1 = 1 and m_2 = 2

Compute $\phi_1(1)$ and $\phi_1(2)$ using the formula: $\phi_i(mi)) = 1 - (1 - r_i)^{mi}$

$\phi_1(1) = 1 - (1 - r_1)^{m1} = 1 - (1 - 0.9)^1 = 0.9$

$\phi_1(2) = 1 - (1 - 0.9)^2 = 0.99$

$S^1_1 = \{f_1(x), x\} = = (0.9, 30)$

$S^1_2 = \{0.99, 30 + 30\} = (0.99, 60)$

Therefore, S^1 = {(0.9, 30), (0.99, 60)}

Next find $S^2_1 = \{(f_2(x), x)\}$

$f_2(x) = \{\phi_2(1) * f_1(\,), \phi_2(2) * f_1(\,), \phi_2(3) * f_1(\,)\}$

$\phi_2(1) = 1 - (1 - r_I)^{mi} = 1 - (1 - 0.8)^1 = 1 - 0.2 = 0.8$

$\phi_2(2) = 1 - (1 - 0.8)^2 = 0.96$

$\phi_2(3) = 1 - (1 - 0.8)^3 = 0.992$

S^2_1 = {(0.8(0.9),30+15), (0.8(0.99),60+15)} = {(0.72, 45), (0.792, 75)}

S^2_2 = {(0.96(0.9),30+15+15), (0.96(0.99),60+15+15)}
= {(0.864, 60), (0.9504, 90)}

S^2_3 = {(0.992(0.9),30+15+15+15), (0.992(0.99),60+15+15+15)}
= {(0.8928, 75), (0.98208, 105)}

$$S^2 = \{S^2_1, S^2_2, S^2_3\}$$

By applying Dominance rule to S^2:

Therefore, S^2 = {(0.72, 45), (0.864, 60), (0.8928, 75)}

<u>Dominance Rule</u>:

If S^i contains two pairs (f_1, x_1) and (f_2, x_2) with the property that $f_1 \geq f_2$ and $x_1 \leq x_2$, then (f_1, x_1) dominates (f_2, x_2), hence by dominance rule (f_2, x_2) can be discarded. Discarding or pruning rules such as the one above is known as dominance rule. Dominating tuples will be present in S^i and Dominated tuples has to be discarded from S^i.

Case 1: if $f_1 \leq f_2$ and $x_1 > x_2$ then discard (f_1, x_1)

Case 2: if $f_1 \underline{>} f_2$ and $x_1 < x_2$ the discard (f_2, x_2)

Case 3: otherwise simply write (f_1, x_1)

$S_2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}$

$\phi_3\ (1) = 1 - (1 - r_I)^{mi} = 1 - (1 - 0.5)^1 = 1 - 0.5 = 0.5$

$\phi_3\ (2) = 1 - (1 - 0.5)^2 = 0.75$

$\phi_3\ (3) = 1 - (1 - 0.5)^3 = 0.875$

$S_1^3 = \{(0.5\ (0.72), 45 + 20), (0.5\ (0.864), 60 + 20), (0.5\ (0.8928), 75 + 20)\}$

$S_1^3 = \{(0.36, 65), (0.437, 80), (0.4464, 95)\}$

$S_2^3 = \{(0.75\ (0.72), 45 + 20 + 20), (0.75\ (0.864), 60 + 20 + 20),$
$(0.75\ (0.8928), 75 + 20 + 20)\}$

$= \{(0.54, 85), (0.648, 100), (0.6696, 115)\}$

$S_3^3 = \{\ (0.875\ (0.72), 45 + 20 + 20 + 20), (0.875\ (0.864),\ 60 + 20 + 20 + 20),$
$(0.875\ (0.8928), 75 + 20 + 20 + 20)\ \}$

$S_3^3 = \{(0.63, 105), (1.756, 120), (0.7812, 135)\}$

If cost exceeds 105, remove that tuples

$S^3 = \{(0.36, 65), (0.437, 80), (0.54, 85), (0.648, 100)\}$

The best design has a reliability of 0.648 and a cost of 100. Tracing back for the solution through S^i's we can determine that $m_3 = 2$, $m_2 = 2$ and $m_1 = 1$.

Other Solution:

According to the principle of optimality:

$$f_n(C) = \max_{1 \leq m_n \leq u_n} \{\phi_n (m_n).\ f_{n-1} (C - C_n m_n) \text{ with } f_o (x) = 1 \text{ and } 0 \leq x \leq C;$$

Since, we can assume each $c_i > 0$, each mi must be in the range $1 \leq m_i \leq u_i$. Where:

$$u_i = \left\lfloor \left(C + C_i - \sum_{i}^{n} C_J \right) / C_i \right\rfloor$$

Using the above equation compute u_1, u_2 and u_3.

$$u_1 = \frac{105 + 30 - (30 + 15 + 20)}{30} = \frac{70}{30} = 2$$

$$u_2 = \frac{105 + 15 - (30 + 15 + 20)}{15} = \frac{55}{15} = 3$$

$$u_3 = \frac{105 + 20 - (30 + 15 + 20)}{20} = \frac{60}{20} = 3$$

$$f_3(105) = \max_{1 \le m3 \le u3} \{\phi_3(m_3).\ f_2(105 - 20m_3)\}$$

$= \max \{\phi_3(1)\ f_2(105 - 20),\ \phi_3(2)\ f_2(105 - 20\text{x}2),\ \phi_3(3)\ f_2(105 - 20\text{x}3)\}$

$= \max \{0.5\ f_2(85),\ 0.75\ f_2(65),\ 0.875\ f_2(45)\}$

$= \max \{0.5 \text{ x } 0.8928,\ 0.75 \text{ x } 0.864,\ 0.875 \text{ x } 0.72\} = 0.648.$

$$f_2\ (85) = \max_{1 \le m2 \le u2} \{\phi_2(m_2).\ f_1(85 - 15m_2)\}$$

$= \max \{\phi_2(1).f_1(85 - 15),\ \phi_2(2).f_1(85 - 15\text{x}2),\ \phi_2(3).f_1(85 - 15\text{x}3)\}$

$= \max \{0.8\ f_1(70),\ 0.96\ f_1(55),\ 0.992\ f_1(40)\}$

$= \max \{0.8 \text{ x } 0.99,\ 0.96 \text{ x } 0.9,\ 0.99 \text{ x } 0.9\} = 0.8928$

$$f_1\ (70) = \max_{1 \le m1 \le u1} \{\phi_1(m_1).\ f_0(70 - 30m_1)\}$$

$= \max \{\phi_1(1)\ f_0(70 - 30),\ \phi_1(2)\ f_0(70 - 30\text{x}2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } 1\} = \max \{0.9,\ 0.99\} = 0.99$

$$f_1\ (55) = \max_{1 \le m1 \le u1} \{\phi_1(m_1).\ f_0(55 - 30m_1)\}$$

$= \max \{\phi_1(1)\ f_0(50 - 30),\ \phi_1(2)\ f_0(50 - 30\text{x}2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } -\infty\} = \max \{0.9,\ -\infty\} = 0.9$

$$f_1\ (40) = \max_{1 \le m1 \le u1} \{\phi_1(m_1).\ f_0(40 - 30m_1)\}$$

$= \max \{\phi_1(1)\ f_0(40 - 30),\ \phi_1(2)\ f_0(40 - 30\text{x}2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } -\infty\} = \max\{0.9,\ -\infty\} = 0.9$

$f_2 \; (65) \; = \; \max\limits_{1 \le m2 \le u2} \; \{\phi_2(m_2).\ f_1(65 - 15m_2)\}$

$= \max \{\phi_2(1)\ f_1(65 - 15),\ \underline{\phi_2(2)\ f_1(65 - 15x2)},\ \phi_2(3)\ f_1(65 - 15x3)\}$

$= \max \{0.8\ \ f_1(50),\ 0.96\ \ f_1(35),\ 0.992\ \ f_1(20)\}$

$= \max \{0.8 \text{ x } 0.9,\ 0.96 \text{ x } 0.9,\ -\infty\} = 0.864$

$f_1 \; (50) \; = \max\limits_{1 \le m1 \le u1} \; \{\phi_1(m_1).\ f_0(50 - 30m_1)\}$

$= \max \{\phi_1(1)\ f_0(50 - 30),\ \phi_1(2)\ f_0(50 - 30x2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } -\infty\} = \max\{0.9, -\infty\} = 0.9$

$f_1 \; (35) \; = \max\limits_{1 \le m1 \le u1} \; \phi_1(m_1).\ f_0(35 - 30m_1)\}$

$= \max \{\underline{\phi_1(1).f_0(35\text{-}30)},\ \phi_1(2).f_0(35\text{-}30x2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } -\infty\} = \max\{0.9, -\infty\} = 0.9$

$f_1 \; (20) = \max\limits_{1 \le m1 \le u1} \; \{\phi_1(m_1).\ f_0(20 - 30m_1)\}$

$= \max \{\phi_1(1)\ f_0(20 - 30),\ \phi_1(2)\ f_0(20 - 30x2)\}$

$= \max \{\phi_1(1) \text{ x } -\infty,\ \phi_1(2) \text{ x } -\infty\} = \max\{-\infty, -\infty\} = -\infty$

$f_2 \; (45) = \max\limits_{1 \le m2 \le u2} \; \{\phi_2(m_2).\ f_1(45 - 15m_2)\}$

$= \max \{\phi_2(1)\ f_1(45 - 15),\ \phi_2(2)\ f_1(45 - 15x2),\ \phi_2(3)\ f_1(45 - 15x3)\}$

$= \max \{0.8\ f_1(30),\ 0.96\ f_1(15),\ 0.992\ f_1(0)\}$

$= \max \{0.8 \text{ x } 0.9,\ 0.96 \text{ x } -\infty,\ 0.99 \text{ x } -\infty\} = 0.72$

$f_1 \; (30) = \max\limits_{1 \le m1 \le u1} \; \{\phi_1(m_1).\ f_0(30 - 30m_1)\}$

$= \max \{\phi_1(1)\ f_0(30 - 30),\ \phi_1(2)\ f_0(30 - 30x2)\}$

$= \max \{\phi_1(1) \text{ x } 1,\ \phi_1(2) \text{ x } -\infty\} = \max\{0.9, -\infty\} = 0.9$

Similarly, $f_1\,(15) = -\infty$, $f_1\,(0) = -\infty$.

The best design has a reliability = 0.648 and

Cost = 30 x 1 + 15 x 2 + 20 x 2 = 100.

Tracing back for the solution through S^i's we can determine that:

$m_3 = 2$, $m_2 = 2$ and $m_1 = 1$.

Chapter 7

BACKTRACKING

General Method:

Backtracking is used to solve problem in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criterion. The desired solution is expressed as an n-tuple (x1, ... , x_n) where each x_i Є S, S being a finite set.

The solution is based on finding one or more vectors that maximize, minimize, or satisfy a criterion function P (x_1, ... , x_n). Form a solution and check at every step if this has any chance of success. If the solution at any point seems not promising, ignore it. All solutions requires a set of constraints divided into two categories: explicit and implicit constraints.

Definition 1: Explicit constraints are rules that restrict each x_i to take on values only from a given set. Explicit constraints depend on the particular instance I of problem being solved. All tuples that satisfy the explicit constraints define a possible solution space for I.

Definition 2: Implicit constraints are rules that determine which of the tuples in the solution space of I satisfy the criterion function. Thus, implicit constraints describe the way in which the x_i's must relate to each other.

- For 8-queens problem:

  Explicit constraints using 8-tuple formation, for this problem are S= {1, 2, 3, 4, 5, 6, 7, 8}.

  The implicit constraints for this problem are that no two queens can be the same (i.e., all queens must be on different columns) and no two queens can be on the same diagonal.

Backtracking is a modified depth first search of a tree. Backtracking algorithms determine problem solutions by systematically searching the solution space for the given problem instance. This search is facilitated by using a tree organization for the solution space.

Backtracking is the procedure where by, after determining that a node can lead to nothing but dead end, we go back (backtrack) to the nodes parent and proceed with the search on the next child.

A backtracking algorithm need not actually create a tree. Rather, it only needs to keep track of the values in the current branch being investigated. This is the way we implement backtracking algorithm. We say that the state space tree exists implicitly in the algorithm because it is not actually constructed.

Terminology:

Problem state is each node in the depth first search tree.

Solution states are the problem states 'S' for which the path from the root node to 'S' defines a tuple in the solution space.

Answer states are those solution states for which the path from root node to s defines a tuple that is a member of the set of solutions.

State space is the set of paths from root node to other nodes. *State space* tree is the tree organization of the solution space. The state space trees are called static trees. This terminology follows from the observation that the tree organizations are independent of the problem instance being solved. For some problems it is advantageous to use different tree organizations for different problem instance. In this case the tree organization is determined dynamically as the solution space is being searched. Tree organizations that are problem instance dependent are called dynamic trees.

Live node is a node that has been generated but whose children have not yet been generated.

E-node is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.

Dead node is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.

Branch and Bound refers to all state space search methods in which all children of an E-node are generated before any other live node can become the E-node.

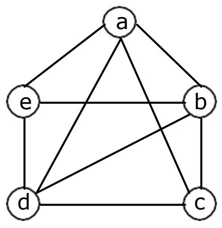
Depth first node generation with bounding functions is called ***backtracking***. State generation methods in which the E-node remains the E-node until it is dead, lead to branch and bound methods.

Planar Graphs:

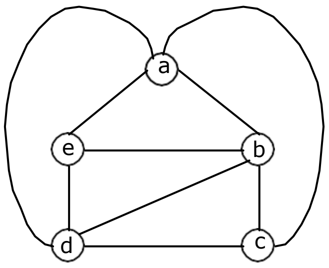
When drawing a graph on a piece of a paper, we often find it convenient to permit edges to intersect at points other than at vertices of the graph. These points of interactions are called crossovers.

A graph G is said to be planar if it can be drawn on a plane without any crossovers; otherwise G is said to be non-planar i.e., A graph is said to be planar iff it can be drawn in a plane in such a way that no two edges cross each other.

Example:



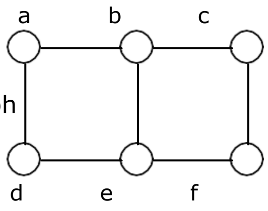
the following graph can be redrawn without crossovers as follows:

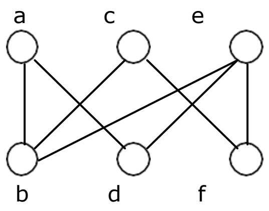


Bipartite Graph:

A bipartite graph is a non-directed graph whose set of vertices can be portioned into two sets V_1 and V_2 (i.e. $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$) so that every edge has one end in V_1 and the other in V_2. That is, vertices in V_1 are only adjacent to those in V_2 and vice- versa.

Example:

The graph 

is bipartite. We can redraw it as 

The vertex set $V = \{a, b, c, d, e, f\}$ has been partitioned into $V_1 = \{a, c, e\}$ and $V_2 = \{b, d, f\}$. The complete bipartite graph for which $V_1 = n$ and $V_2 = m$ is denoted $K_{n,m}$.

N-Queens Problem:

Let us consider, N = 8. Then 8-Queens Problem is to place eight queens on an 8 x 8 chessboard so that no two "attack", that is, no two of them are on the same row, column, or diagonal.

All solutions to the 8-queens problem can be represented as 8-tuples $(x_1, \ldots, x_8)$, where x_i is the column of the ith row where the ith queen is placed.

The explicit constraints using this formulation are $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $1 \leq i \leq 8$. Therefore the solution space consists of 8^8 8-tuples.

The implicit constraints for this problem are that no two x_i's can be the same (i.e., all queens must be on different columns) and no two queens can be on the same diagonal.

This realization reduces the size of the solution space from 8^8 tuples to 8! Tuples.

The promising function must check whether two queens are in the same column or diagonal:

Suppose two queens are placed at positions (i, j) and (k, l) Then:

- Column Conflicts: Two queens conflict if their x_i values are identical.
- Diag 45 conflict: Two queens i and j are on the same 45^0 diagonal if:

$$i - j = k - l.$$

  This implies, $j - l = i - k$

- Diag 135 conflict:

$$i + j = k + l.$$

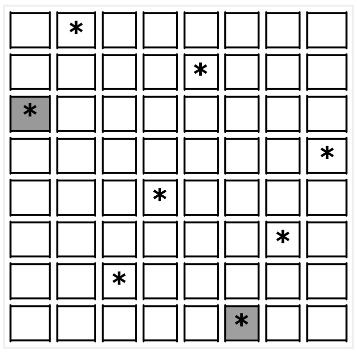
  This implies, $j - l = k - i$

Therefore, two queens lie on the same diagonal if and only if:

$$|j - l| = |i - k|$$

Where, j be the column of object in row i for the i^{th} queen and l be the column of object in row 'k' for the k^{th} queen.

To check the diagonal clashes, let us take the following tile configuration:



In this example, we have:

i	1	2	3	4	5	6	7	8
x_i	2	5	1	8	4	7	3	6

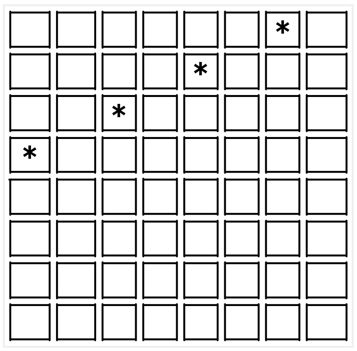
Let us consider for the case whether the queens on 3rd row and 8th row are conflicting or not. In this case (i, j) = (3, 1) and (k, l) = (8, 6). Therefore:

$$|j - l| = |i - k| \Rightarrow |1 - 6| = |3 - 8|$$
$$\Rightarrow 5 = 5$$

In the above example we have, $|j - l| = |i - k|$, so the two queens are attacking. This is not a solution.

Example:

Suppose we start with the feasible sequence 7, 5, 3, 1.



Step 1:
Add to the sequence the next number in the sequence 1, 2, . . . , 8 not yet used.

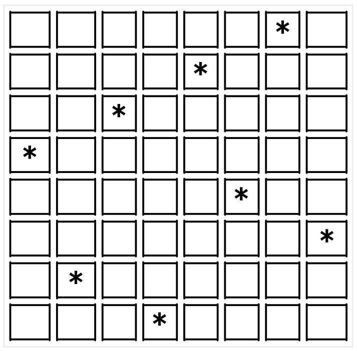
Step 2:
If this new sequence is feasible and has length 8 then STOP with a solution. If the new sequence is feasible and has length less then 8, repeat Step 1.

Step 3:
If the sequence is not feasible, then *backtrack* through the sequence until we find the *most recent* place at which we can exchange a value. Go back to Step 1.

1	2	3	4	5	6	7	8	Remarks
7	5	3	1					
7	5	3	1*	2*				|j - l| = |1 – 2| = 1 |i – k | = |4 – 5| = 1
7	5	3	1	4				
7*	5	3	1	4	2*			|j - l| = |7 – 2| = 5 |i – k | = |1 – 6| = 5
7	5	3*	1	4	6*			|j - l| = |3 – 6| = 3 |i – k | = |3 – 6| = 3
7	5	3	1	4	8			
7	5	3	1	4*	8	2*		|j - l| = |4 – 2| = 2 |i – k | = |5 – 7| = 2
7	5	3	1	4*	8	6*		|j - l| = |4 – 6| = 2 |i – k | = |5 – 7| = 2
7	5	3	1	4	8			*Backtrack*
7	5	3	1	4				*Backtrack*
7	5	3	1	6				
7*	5	3	1	6	2*			|j - l| = |1 – 2| = 1 |i – k | = |7 – 6| = 1
7	5	3	1	6	4			
7	5	3	1	6	4	2		
7	5	3*	1	6	4	2	8*	|j - l| = |3 – 8| = 5 |i – k| = |3 – 8| = 5
7	5	3	1	6	4	2		*Backtrack*
7	5	3	1	6	4			*Backtrack*
7	5	3	1	6	8			
7	5	3	1	6	8	2		
7	5	3	1	6	8	2	4	**SOLUTION**

* indicates conflicting queens.

On a chessboard, the **solution** will look like:



4 – Queens Problem:

Let us see how backtracking works on the 4-queens problem. We start with the root node as the only live node. This becomes the E-node. We generate one child. Let us assume that the children are generated in ascending order. Let us assume that the children are generated in ascending order. Thus node number 2 of figure is generated and the path is now (1). This corresponds to placing queen 1 on column 1. Node 2 becomes the E-node. Node 3 is generated and immediately killed. The next node generated is node 8 and the path becomes (1, 3). Node 8 becomes the E-node. However, it gets killed as all its children represent board configurations that cannot lead to an answer node. We backtrack to node 2 and generate another child, node 13. The path is now (1, 4). The board configurations as backtracking proceeds is as follows:

1			

(a)

1			
▪	▪	2	

(b)

1			
		2	
▪	▪	▪	▪

(c)

1			
			2
▪	3		

(d)

1			
			2
	3		
▪	▪	▪	▪

(e)

	1		

(f)

	1		
▪	▪	▪	2

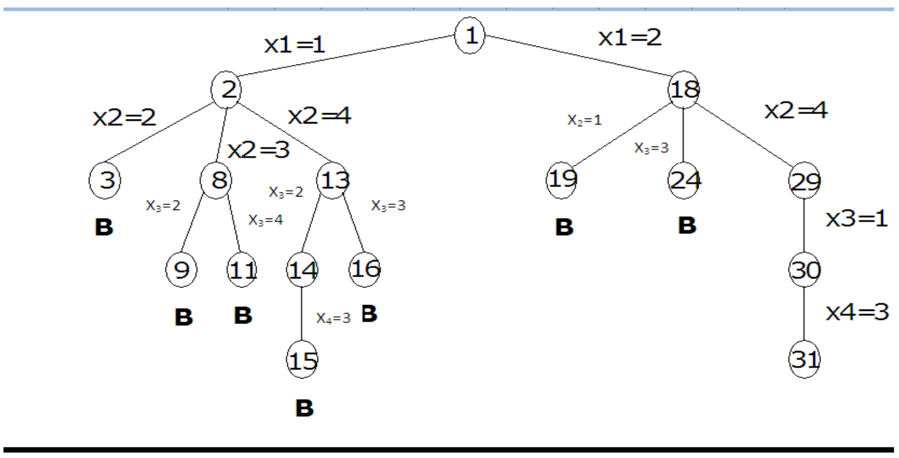
(g)

	1		
			2
3			
▪	▪	4	

(h)

The above figure shows graphically the steps that the backtracking algorithm goes through as it tries to find a solution. The dots indicate placements of a queen, which were tried and rejected because another queen was attacking.

In Figure (b) the second queen is placed on columns 1 and 2 and finally settles on column 3. In figure (c) the algorithm tries all four columns and is unable to place the next queen on a square. Backtracking now takes place. In figure (d) the second queen is moved to the next possible column, column 4 and the third queen is placed on column 2. The boards in Figure (e), (f), (g), and (h) show the remaining steps that the algorithm goes through until a solution is found.



Portion of the tree generated during backtracking

Complexity Analysis:

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

For the instance in which n = 8, the state space tree contains:

$$\frac{8^{8+1} - 1}{8 - 1} = 19,\ 173,\ 961 \text{ nodes}$$

Program for N-Queens Problem:

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>

int x[10] = {5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

place (int k)
{
        int i;
        for (i=1; i < k; i++)
        {
                if ((x [i] == x [k]) || (abs (x [i] – x [k]) == abs (i - k)))
                return (0);
        }
        return (1);
}
nqueen (int n)
{
        int m, k, i = 0;
        x [1] = 0;
        k = 1;
        while (k > 0)
        {
                x [k] = x [k] + 1;
                while ((x [k] <= n) && (!place (k)))
                        x [k] = x [k] +1;
                if(x [k] <= n)
                {
                        if (k == n)
                        {
                                i++;
                                printf ("\ncombination; %d\n",i);
                                for (m=1;m<=n; m++)
                                printf("row = %3d\t column=%3d\n", m, x[m]);
                                getch();
                        }
                        else
                        {
                                k++;
                                x [k]=0;
                        }
                }

                else
                        k--;
        }
        return (0);
```

```
 }
main ()
{
        int n;
        clrscr ();
        printf (“enter value for N: ”);
        scanf (“%d”, &n);
        nqueen (n);
 }
```

Output:

```
Enter the value for N: 4

Combination: 1                    Combination: 2

Row  =  1     column = 2                3
Row  =  2     column = 4                1
Row  =  3     column = 1                4
Row  =  4     column = 3                2
```

For N = 8, there will be 92 combinations.

Sum of Subsets:

Given positive numbers wi, $1 \leq i \leq n$, and m, this problem requires finding all subsets of w_i whose sums are ‘m’.

All solutions are k-tuples, $1 \leq k \leq n$.

Explicit constraints:

- $x_i \in \{j \mid j$ is an integer and $1 \leq j \leq n\}$.

Implicit constraints:

- No two x_i can be the same.
- The sum of the corresponding w_i’s be m.
- $x_i < x_{i+1}$, $1 \leq i < k$ (total order in indices) to avoid generating multiple instances of the same subset (for example, (1, 2, 4) and (1, 4, 2) represent the same subset).

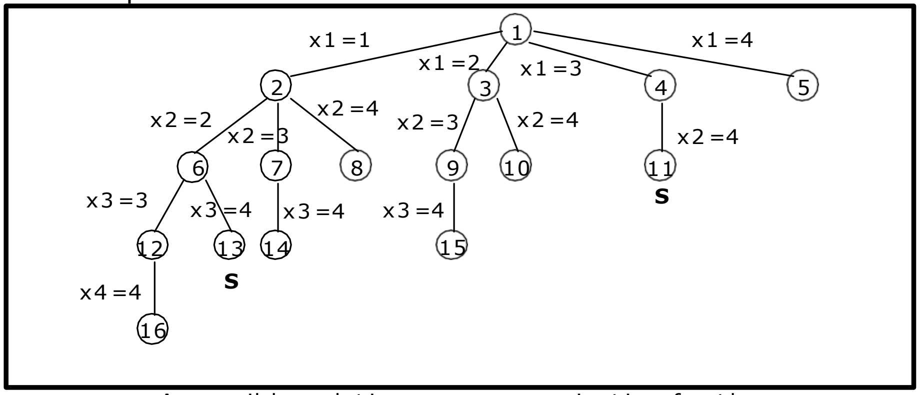
A better formulation of the problem is where the solution subset is represented by an n-tuple $(x_1, ... , x_n)$ such that $x_i \in \{0, 1\}$.

The above solutions are then represented by (1, 1, 0, 1) and (0, 0, 1, 1).

For both the above formulations, the solution space is 2^n distinct tuples.

For example, n = 4, w = (11, 13, 24, 7) and m = 31, the desired subsets are (11, 13, 7) and (24, 7).

The following figure shows a possible tree organization for two possible formulations of the solution space for the case n = 4.



A possible solution space organisation for the sum of the subsets problem.

The tree corresponds to the variable tuple size formulation. The edges are labeled such that an edge from a level i node to a level i+1 node represents a value for x_i. At each node, the solution space is partitioned into sub - solution spaces. All paths from the root node to any node in the tree define the solution space, since any such path corresponds to a subset satisfying the explicit constraints.

The possible paths are (1), (1, 2), (1, 2, 3), (1, 2, 3, 4), (1, 2, 4), (1, 3, 4), (2), (2, 3), and so on. Thus, the left mot sub-tree defines all subsets containing w_1, the next sub-tree defines all subsets containing w_2 but not w_1, and so on.

Graph Coloring (for planar graphs):

Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color, yet only m colors are used. This is termed the m-colorabiltiy decision problem. The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored.

Given any map, if the regions are to be colored in such a way that no two adjacent regions have the same color, only four colors are needed.

For many years it was known that five colors were sufficient to color any map, but no map that required more than four colors had ever been found. After several hundred years, this problem was solved by a group of mathematicians with the help of a computer. They showed that in fact four colors are sufficient for planar graphs.

The function m-coloring will begin by first assigning the graph to its adjacency matrix, setting the array x [] to zero. The colors are represented by the integers 1, 2, . . . , m and the solutions are given by the n-tuple (x_1, x_2, . . ., x_n), where x_i is the color of node i.

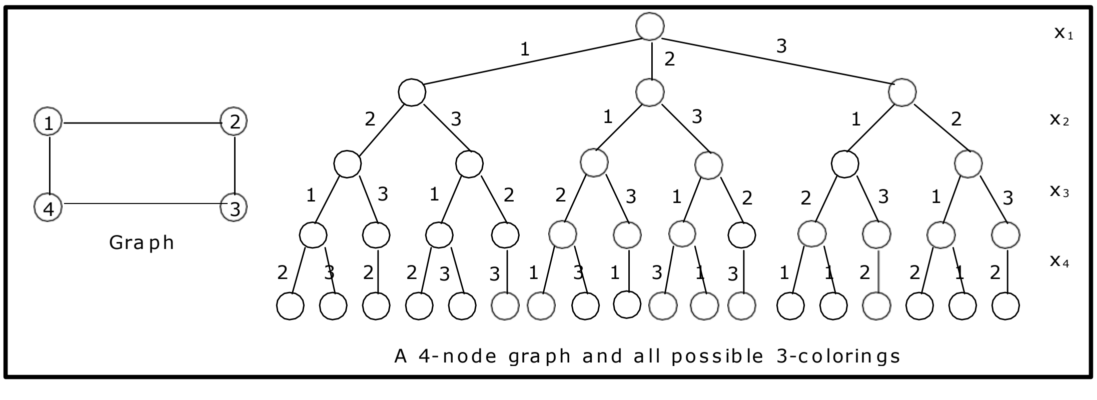
A recursive backtracking algorithm for graph coloring is carried out by invoking the statement mcoloring(1);

```
Algorithm mcoloring (k)
// This algorithm was formed using the recursive backtracking schema. The graph is
// represented by its Boolean adjacency matrix G [1: n, 1: n].  All assignments of
// 1, 2, . . . . . , m to the vertices of the graph such that adjacent vertices are assigned
// distinct integers are printed. k is the index  of the next vertex to color.
{
        repeat
        {                                        // Generate all legal assignments for x[k].
                NextValue   (k);           // Assign to x [k] a legal color.
                If (x [k] = 0)  then return;       // No new color possible
                If (k =   n)  then           // at most m colors have been
                                                 // used to color the n vertices.
                        write (x [1: n]);
                        else mcoloring (k+1);
                } until (false);
        }
```

```
Algorithm NextValue (k)
// x [1] , . . . . x [k-1] have been assigned integer values in the range [1, m] such that
// adjacent vertices have distinct integers. A value for x [k] is determined in the range
// [0, m].x[k] is assigned the next highest numbered color while maintaining distinctness
// from the adjacent vertices of vertex k. If no such color exists, then x [k] is 0.
{
        repeat
        {
                x [k]: = (x [k] +1)  mod (m+1)                  // Next highest color.
                If (x [k] = 0)  then return;                    // All colors have been used
                for j := 1 to n do
                {       // check if this color is distinct from adjacent colors
                        if ((G [k, j] ≠ 0) and (x [k] = x [j]))
                        // If (k, j) is and edge and if adj. vertices have the same color.
                        then break;
                }
                if (j = n+1)  then return;                  // New color found
        }  until  (false);                          // Otherwise try to find another color.
}
```

Example:

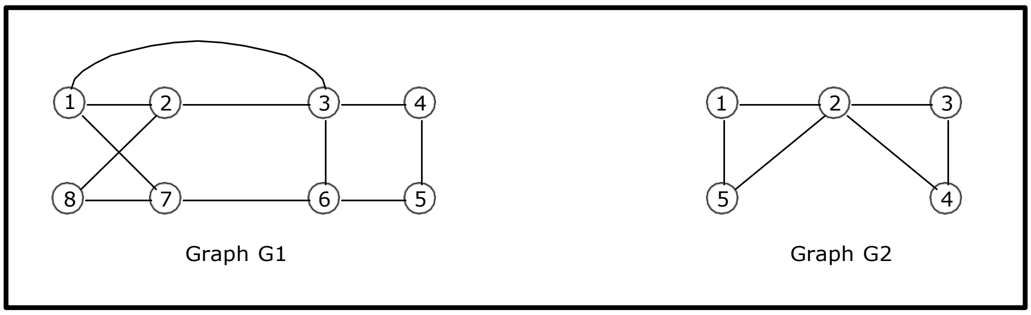
Color the graph given below with minimum number of colors by backtracking using state space tree.



A 4-node graph and all possible 3-colorings

Hamiltonian Cycles:

Let G = (V, E) be a connected graph with n vertices. A Hamiltonian cycle (suggested by William Hamilton) is a round-trip path along n edges of G that visits every vertex once and returns to its starting position. In other vertices of G are visited in the order $v_1, v_2, \ldots, v_{n+1}$, then the edges (v_i, v_{i+1}) are in E, $1 \leq i \leq n$, and the v_i are distinct expect for v_1 and v_{n+1}, which are equal. The graph G_1 contains the Hamiltonian cycle 1, 2, 8, 7, 6, 5, 4, 3, 1. The graph G_2 contains no Hamiltonian cycle.



Two graphs to illustrate Hamiltonian cycle

The backtracking solution vector $(x_1, \ldots\ldots x_n)$ is defined so that x_i represents the ith visited vertex of the proposed cycle. If k = 1, then x_1 can be any of the n vertices. To avoid printing the same cycle n times, we require that $x_1 = 1$. If $1 < k < n$, then x_k can be any vertex v that is distinct from $x_1, x_2, \ldots, x_{k-1}$ and v is connected by an edge to k_{x-1}. The vertex x_n can only be one remaining vertex and it must be connected to both x_{n-1} and x_1.

Using NextValue algorithm we can particularize the recursive backtracking schema to find all Hamiltonian cycles. This algorithm is started by first initializing the adjacency matrix G[1: n, 1: n], then setting x[2: n] to zero and x[1] to 1, and then executing Hamiltonian(2).

The traveling salesperson problem using dynamic programming asked for a tour that has minimum cost. This tour is a Hamiltonian cycles. For the simple case of a graph all of whose edge costs are identical, Hamiltonian will find a minimum-cost tour if a tour exists.

```
Algorithm NextValue (k)
// x [1: k-1] is a path of k – 1 distinct vertices . If x[k] = 0, then no vertex has as yet been
// assigned to x [k]. After execution, x[k] is assigned to the next highest numbered vertex
// which does not already appear in x [1 : k – 1] and is connected by an edge to x [k – 1].
// Otherwise x [k] = 0. If k = n, then in addition x [k] is connected to x [1].
{
        repeat
        {
                x [k] := (x [k] +1)  mod (n+1);         // Next vertex.
                If (x [k] = 0) then return;
                If (G [x [k – 1], x [k]] ≠ 0) then
                {                                       // Is there an edge?
                        for j := 1 to k – 1 do if (x [j] = x [k]) then break;
                                                        // check for distinctness.
                        If (j =  k) then                // If true, then the vertex is distinct.
                        If ((k < n) or ((k = n) and G [x [n], x [1]] ≠ 0))
                        then return;
                }
        } until (false);
}
```

```
Algorithm Hamiltonian (k)
// This algorithm uses the recursive formulation of backtracking to find all the Hamiltonian
// cycles of a graph. The graph is stored as an adjacency matrix G [1: n, 1: n]. All cycles begin
// at node 1.
{
        repeat
        {                                                   // Generate values for x [k].
                NextValue (k);                  //Assign a legal Next value to x [k].
        if (x [k] = 0) then return;
                if (k = n) then write (x [1: n]);
                else Hamiltonian (k + 1)
        } until (false);
}
```

0/1 Knapsack:

Given n positive weights w_i, n positive profits p_i, and a positive number m that is the knapsack capacity, the problem calls for choosing a subset of the weights such that:

$$\sum_{1 \leq i \leq n} w_i \; x_i \leq m \text{ and } \sum_{1 \leq i \leq n} p_i \; x_i \text{ is maximized.}$$

The x_i's constitute a zero–one-valued vector.

The solution space for this problem consists of the 2^n distinct ways to assign zero or one values to the x_i's.

Bounding functions are needed to kill some live nodes without expanding them. A good bounding function for this problem is obtained by using an upper bound on the value of the best feasible solution obtainable by expanding the given live node and any of its descendants. If this upper bound is not higher than the value of the best solution determined so far, than that live node can be killed.

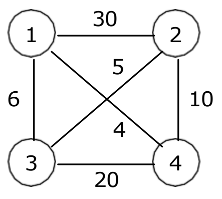
We continue the discussion using the fixed tuple size formulation. If at node Z the values of x_i, $1 \leq i \leq k$, have already been determined, then an upper bound for Z can be obtained by relaxing the requirements $x_i = 0$ or 1.

(Knapsack problem using backtracking is solved in branch and bound chapter)

7.8 Traveling Sale Person (TSP) using Backtracking:

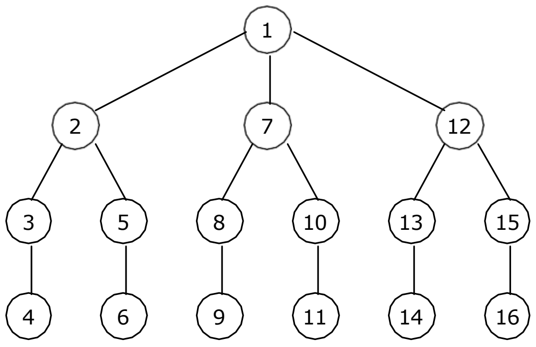
We have solved TSP problem using dynamic programming**.** In this section we shall solve the same problem using backtracking.

Consider the graph shown below with 4 vertices.



A graph for TSP

The solution space tree, similar to the n-queens problem is as follows:



We will assume that the starting node is 1 and the ending node is obviously 1. Then 1, {2, … ,4}, 1 forms a tour with some cost which should be minimum. The vertices shown as {2, 3, …. ,4} forms a permutation of vertices which constitutes a tour. We can also start from any vertex, but the tour should end with the same vertex.

Since, the starting vertex is 1, the tree has a root node R and the remaining nodes are numbered as depth-first order. As per the tree, from node 1, which is the live node, we generate 3 braches node 2, 7 and 12. We simply come down to the left most leaf node 4, which is a valid tour {1, 2, 3, 4, 1} with cost 30 + 5 + 20 + 4 = 59. Currently this is the best tour found so far and we backtrack to node 3 and to 2, because we do not have any children from node 3. When node 2 becomes the E-node, we generate node 5 and then node 6. This forms the tour {1, 2, 4, 3, 1} with cost 30 + 10 + 20 + 6 = 66 and is discarded, as the best tour so far is 59.

Similarly, all the paths from node 1 to every leaf node in the tree is searched in a depth first manner and the best tour is saved. In our example, the tour costs are shown adjacent to each leaf nodes. The optimal tour cost is therefore 25.