Assignment No.B3

Roll No: 4127

Title:

1.1 Knapsack problem using Branch and Bound.

Problem Statement:

Implementation of 0-1 knapsack problem using branch and bound approach.

Learning Objectives:

1. To study Branch and Bound approach.
2. To implement 0-1 Knapsack problem.

Learning Outcomes:

1. Able to study Branch and Bound approach.
2. Able to implement 0-1 Knapsack problem.

Software and Hardware Requirements:

1. 64 bit Machine i3/i5/i7
2. 64-bit open source Linux OS Fedora 20
3. Gedit editor and g++ compiler

Theory

0-1 Knapsack Problem

i. The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

ii. Consider a collection of N indivisible objects, labeled by the integers i = 1; 2; …; N. With each object is associated a positive real number w_i, the "weight" of the object, and real number v_i, the "value" of the object. It is desired to form a loading of the objects by selecting from among the N objects a subcollection which has a maximum total value but which does not exceed a total weight of say W units. The problem may be stated mathematically as follows: Find x_i, i = 1; 2; …; N such that

1. Maximize $\sum_{i\ 1}^{N} xiwi$ subject to
2. $\sum_{i\ 1}^{N} xiwi$ <= W
3. x_i = 0 or1; i = 1; 2; …; N:

iii. The problem, thus formulated, is an integer linear programming problem and may be solved as such. Further, since there are only a finite number of indivisible objects which are to be arranged into two groups (those loaded and those not loaded), there are only a finite (although possibly a very large) number of possible solutions, and an algorithm which proceeds by successive partitions of the class of all possible solutions will work, at least in theory.

Branch and Bound approach

i. Branch and bound (BB or B&B) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as general real valued problems.

ii. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root.

iii. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

iv. The algorithm depends on the efficient estimation of the lower and upper bounds of a region/branch of the search space and approaches exhaustive enumeration as the size (n-dimensional volume) of the region tends to zero.

v. The goal of a branch-and-bound algorithm is to find a value x that maximizes or minimizes the value of a real-valued function f(x), called an objective function, among some set S of admissible, or candidate solutions. The set S is called the search space, or feasible region.

vi. A B&B algorithm operates according to two principles:

1. It recursively splits the search space into smaller spaces, then minimizing f(x) on these smaller spaces; the splitting is called branching.
2. Branching alone would amount to brute-force enumeration of candidate solutions and testing them all. To improve on the performance of brute-force search, a B&B algorithm keeps track of bounds on the minimum that it is trying to nd, and uses these bounds to "prune" the search space, eliminating candidate solutions that it can prove will not contain an optimal solution.

vii. Turning these principles into a concrete algorithm for a specific optimization problem requires some kind of data structure that represents of sets of candidate solutions. Such a representation is called an instance of the problem.

viii. Several diffierent queue data structures can be used. A stack (LIFO queue) will yield a depth first algorithm. A best first branch and bound algorithm can be obtained by using a priority queue that sorts nodes on their g-value. The depth first variant is recommended when no good heuristic is available for producing an initial solution, because it quickly produces full solutions, and therefore upper bounds.

Mathematical Model:

Let S be the solution perspective of the class knapsack BB
such that,
S={s, e, X, Y, f, DD, NDD, Sc, Fc}
where,

s = Initial state of the System
= { ϕ }

e = End state or destructor of the class
= {Y}

X=Input of the system
X={ x1,x2,x3, x4}
x1={x11 | x11 ∈ I_+}
i.e Number of items
x2={ Wi | Wi ∈ I_+ } , i = {1, 2 ,…, x11}
i.e weight array for elements
x3={ Pi | Pi ∈ I_+ } , i = {1, 2 ,…, x11}
i.e profit array for elements
x4={Wc | Wc ∈ I_+ ^ ∑Wi <= Wc }
i.e. weight constraint for items

Y = Output of the system
Y = {y1, y2}
where,

y1 = Solution array of selected items
y1 = {0 | 1 } i.e. 0 -> selected , 1-> not selected
y2 = { Pmax } i.e.Maximum profit
 $P_{max} = \sum P_i$ for $i \in y1$

F = Set of functions
F = {f1, f2, f3, f4, f5}
where,
f1 : function to accept input
f1 : input -> X
f2 : function to sort the input elements
f2 : X -> y'
Ri = { Pi/Wi }, i = {1,2,…,x11}
y' = { $R_i \mid R_{i-1} > R_i > R_{i+1}$}
f3 : function to calculate the upper bound
f3 : y' -> y''
y'' : { cw = cw+ Wi && cp= cp + Pi | iff cw < Wc} for i= {1,…,x11}
f4 : function to calculate the lower bound
f4 : y' -> y'''
y'' : { cw = cw+ Wi && cp = cp – Pi | iff cw < Wc} for i= {1,…,x11}
f5 : function to calculate maximum prfiit
f5 : X -> y2

DD = Deterministic data
={}{Wc} i.e weight constraint

NDD=Non deterministic data
= {Y}

Sc : Success Case
Desired outcome generated

Fc : Failure Case
{Wi | $\sum$ Wi > Wc}
i.e. weight obtained is greater than weight constraint

Algorithm:

1. Start.
2. Initialise w[n], p[n], and the tuple structure.
3. Get no. of elements, weights, profits, and weight constraint from the user.
4. Sort the input in non-increasing order of pro t/weight ratio.
5. Calculate Upper-bound and Lower-bound.

6. Push the next value of weight and corresponding pro t onto the stack and revise the W, P and bound values at every node.
   if $x_i = 1$ i.e., the weight of x_i is accepted, and repeat step 5 else, $x_i = 0$ i.e., the weight of x_i is not feasible in the solution.

7. Repeat step 6 till all the leaf nodes in the state space tree are considered.

8. Print the solution and maximum profit value.

9. Stop.

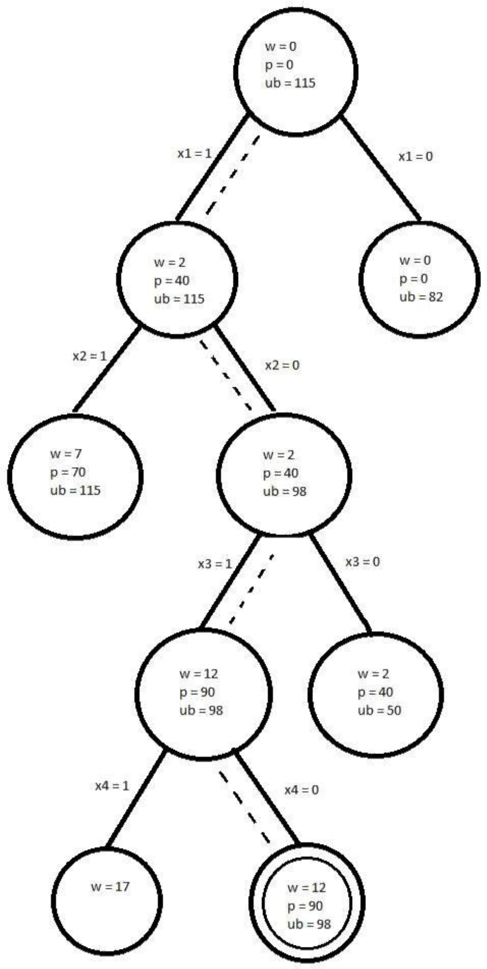

Figure 1: Here is the state space tree representation of the input given to the program. The dotted line traces the path of the final solution in the tree.

Conclusion:

Thus studied Branch and Bound approach using 0-1 Knapsack problem as an example.