Greedy Method

Introduction

- Greedy algorithms are typically used to solve an optimization problem.
- An Optimization problem is one in which, we are given a set of input values, which are required to be either maximized or minimized w. r. t. some constraints or conditions.
- Generally an optimization problem has n inputs (call this set as input domain or Candidate set, C), we are required to obtain a subset of C (call it solution set, S where S) that satisfies the given constraints or conditions.
- Any subset S, which satisfies the given constraints, is called a feasible solution.
- We need to find a feasible solution that maximizes or minimizes a given objective function.
- The feasible solution that does this is called an optimal solution.

Introduction

- A greedy algorithm proceeds step–by-step, by considering one input at a time.
- At each stage, the decision is made regarding whether a particular input (say x) chosen gives an optimal solution or not.
- Our choice of selecting input x is being guided by the selection function (say select). If the inclusion of x gives an optimal solution, then this input x is added into the partial solution set.
- On the other hand, if the inclusion of that input x results in an infeasible solution, then this input x is not added to the partial solution.
- The input we tried and rejected is never considered again.
- When a greedy algorithm works correctly, the first solution found in this way is always optimal.

Steps:

1. First we select an element, say, from input domain C.

2. Then we check whether the solution set S is feasible or not. That is we check whether x can be included into the solution set S or not. If yes, then solution set. If no, then this input x is discarded and not added to the partial solution set S. Initially S is set to empty.

3. Continue until S is filled up (i.e. optimal solution found) or C is exhausted whichever is earlier.

- (Note: From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function (either maximize or minimize, as the case may be), is called ***optimal solution***

Formalization of Greedy Technique

- In order to solve optimization problem using greedy technique, we need the following data structures and functions:

1) A candidate set from which a solution is created. It may be set of nodes, edges in a graph etc. call this set as: C: Set of given values or set of candidates

2) A solution set S (where S , in which we build up a solution. This structure contains those candidate values, which are considered and chosen by the greedy technique to reach a solution. Call this set as: S: Set of selected candidates (or input) which is used to give optimal solution.

3) A function (say solution) to test whether a given set of candidates give a solution (not necessarily optimal).

4) A selection function (say select) which chooses the best candidate form C to be added to the solution set S,

5) A function (say feasible) to test if a set S can be extended to a solution (not necessarily optimal)

6) An objective function (say ObjF) which assigns a value to a solution, or a partial solution.

General Greedy Method

A general form for greedy technique can be illustrated as:

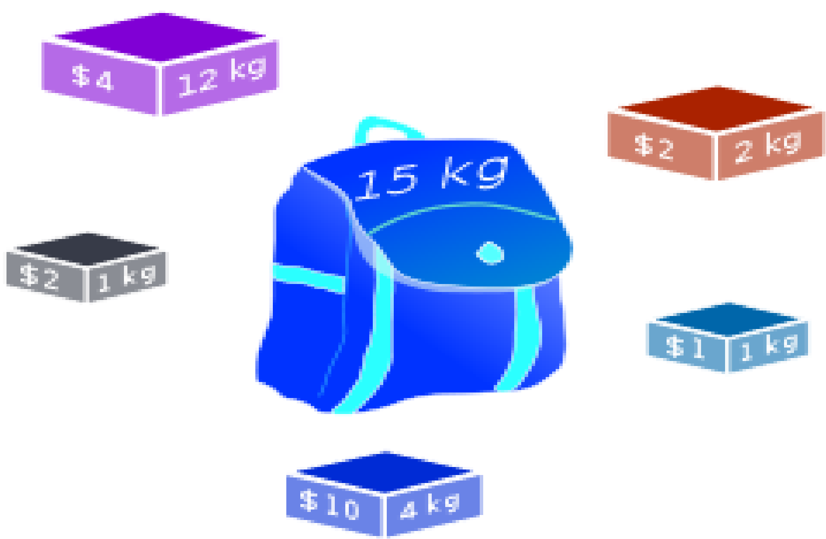
```
Algorithm Greedy(C, n)

   /* Input: A input domain (or Candidate set ) C of size n, from which solution is to be
              Obtained. */

  // function select (C: candidate_set)  return an element (or candidate).
  // function solution (S: candidate_set) return Boolean
  // function feasible (S: candidate_set) return Boolean
  /* Output: A solution set S, where S ⊆ C, which maximize or minimize the selection
              criteria w. r. t. given constraints */

{
        S ← ϕ                              // Initially a solution set S is empty.
        While ( not solution(S) and C ≠ ϕ)
          {
            x ← select(C)              /* A "best" element x is selected from C which
                                          maximize or minimize the selection criteria. */
            C ← C − {x}                    /* once x is selected , it is removed from C
            if ( feasible(S ∪ {x})   then  /* x is now checked for feasibility
                S ← S ∪ {x}
          }
        If (solution (S))
            return S;
         else
           return " No Solution"
    } // end of while
```

Characteristics of Greedy Algorithm

- Used to solve optimization problem
- Most general, straightforward method to solve a problem.
- Easy to implement, and if exist, are efficient.
- Always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a overall globally optimal solution.
- Once any choice of input from C is rejected then it never considered again.
- Do not always yield an optimal solution; but for many problems they do.

Knapsack Problem

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



Knapsack Problem

The Knapsack Problem:

➢Given *n* objects each have a *weight* w_i and a *Profit* p_i, and given a knapsack of total *capacity M*.

➢The problem is to pack the knapsack with these objects in order to maximize the total value of those objects packed without exceeding the knapsack's capacity.

➢More formally, let x_i denote the fraction of the object *i* to be included in the knapsack, $0 \le x_i \le 1$, for $1 \le i \le n$. The problem is to find values for the x_i such that

$$\sum_{i=1}^{n} x_i w_i \le M \text{ and } \sum_{i=1}^{n} x_i p_i \text{ is maximized.}$$

➢Note that we may assume $\sum_{i=1}^{n} w_i > M$ because otherwise, we would choose $x_i = 1$ for each *i* which would be an obvious optimal solution.

The knapsack Problem

- n objects, each with a weight $w_i > 0$
  a profit $p_i > 0$
  capacity of knapsack: M

Maximize $\sum_{1 \le i \le n} p_i x_i$

Subject to $\sum_{1 \le i \le n} w_i x_i \le M$

$0 \le x_i \le 1, 1 \le i \le n$

- The knapsack problem is different from the 0/1 knapsack problem. In the 0/1 knapsack problem, x_i is either 0 or 1 while in the knapsack problem, $0 \le x_i \le 1$.

The knapsack algorithm

- The greedy algorithm:

Step 1: Sort p_i/w_i into non increasing order.

Step 2: Put the objects into the knapsack according to the sorted sequence as far as possible.

- e.g.

$n = 3, M = 20, (p_1, p_2, p_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

Sol: $p_1/w_1 = 25/18 = 1.32$

$p_2/w_2 = 24/15 = 1.6$

$p_3/w_3 = 15/10 = 1.5$

Optimal solution: $x_1 = 0, x_2 = 1, x_3 = 1/2$

The Optimal Knapsack Algorithm:

Input: an integer n, positive profits w_i and p_i , for $1 \le i \le n$, and another positive value M.

$$\sum_{i=1}^{n} x_i w_i \le M \text{ and } \sum_{i=1}^{n} x_i p_i \text{ is maximized.}$$

Output: n values x_i such that $0 \le x_i \le 1$ and

Algorithm (of time complexity $O(n \lg n)$)

(1) Sort the *n* objects from large to small based on the ratios p_i/w_i .

We assume the arrays $w[1..n]$ and $p[1..n]$ store the respective weights and values after sorting.

(2) initialize array $x[1..n]$ to zeros.

(3) weight = 0; i = 1; profit=0;

(4) while ($i \le n$ and weight < M) do

(4.1) if weight + $w[i] \le M$ then $x[i]$ = 1

(4.2) else $x[i]$ = (M – weight) / w[i]

(4.3) weight = weight + x[i] * w[i]

*(4.4)profit=profit + x[i]*p[i]*

(4.5) i++