Volume 5, Issue 11, November 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Case Study on Shifting Items with Max Profit in KNAPSACK with GREEDY Approach

Rajinder Singh
Panjab University S.S.G. Regional Centre,
Hoshiarpur, Punjab, India

***Abstract:** The paper describes a case study of Knapsack in which we have a set of items, each item has a profit and a weight, we have to determine the number of each item to include in a Knapsack (bag) so that the total weight of selected items is less than or equal to the weight that carried by Knapsack and the total profit will be as large as possible. We have to select only that items for our solution which has maximum profit as well as that item's weight can't be large than bag capacity. If any item has more profit but weight is larger than enough then we need to take part of that item rather than whole item.*

***Keywords:** Knapsack class, Weight (P[i]/W[i]), profit matrix, Greedy Policy, Highest Profit*

I. INTRODUCTION

Before entering into the problem, it becomes important to know about the Greedy method and knapsack problem, that is why, here, a brief introduction on Greedy method: -
Greedy Selection policy: -Three natural possibilities

•**Policy 1**:
Choose the lightest remaining item, and take as much of it as can fit.

•**Policy 2**:
Choose the most profitable remaining item, and take as much of it as can fit.

•**Policy 3**:
Choose the item with the highest price per unit weight (P[i]/W[i]) and take as much of it as can fit.

II. KNAPSACK PROBLEM

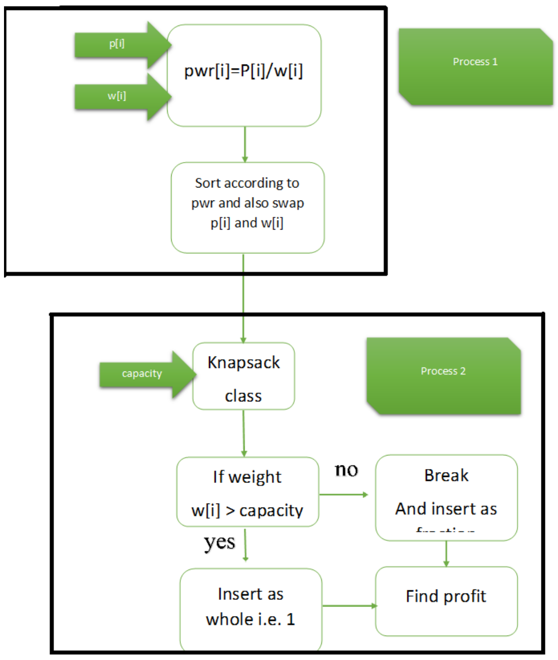
- **Input: -**A weight capacity C, and n items of weights W [1:n] and profit value P[1:n].
- **Formulation of the problem**: Let x[i] be the fraction taken from item i.
  0 <= x[i] <= 1.

The weight of the part taken from item i is x[i]*w[i] The corresponding profit is x[i]*P[i].
The problem is then to find the values of the array x[1:n].
The case study with name" **Shifting Items with Max profit in KNAPSACK**" is implemented with "**GREEDY Approach**" in which we have a set of items, Profits and Weights are associated with each item and we have to determine the number of each item to be include in a Knapsack (bag) so that the total weight of the selected items is less than or equal to the weight, that is carried by Knapsack and we have to get the maximum profit that we can earn from these items. But we have to select only that items for our solution which have maximum profit as well as that item's weight can't be large than bag capacity.

If any item has more profit but weight is larger than enough then we need to take part of that item rather than whole item. We have three possibilities to achieve maximum profit. These are:
1. Choose the lightest remaining item, and take as much of it, in which it can fit.
2. Choose the most profitable remaining items and take as much part of it, in which it can fit.
3. Choose the item with the highest price per unit weight (P[i]/W[i]) and take as much of it, in which it can fit.

Fractional Knapsack has time complexity **O (N log N)** where N is the number of items. It has been observed that a greedy strategy does not generally produce an optimal solution, but a greedy algorithm may locally yield optimal solutions that approximate a global optimal solution in a reasonable time. It is a way of making the locally optimal choice at each stage with the hope of finding an optimal solution with maximum profit.

DFD


p[i]
w[i]
pwr[i]=P[i]/w[i]
Process 1
Sort according to pwr and also swap p[i] and w[i]
capacity
Knapsack class
Process 2
If weight w[i] > capacity
no
Break And insert as fraction
yes
Insert as whole i.e. 1
Find profit

Construct in C

```
#include<iostream.h>
#include<conio.h>
template <class T>
    void swap(T *a,int i,int j)
    {
      T temp;
      temp=a[j];
      a[j]=a[i];
      a[i]=temp;
      return;
    }
class knapsack
{
    int cap;
    float x[10];
  public:
    knapsack();
    void select(int,int []);
    void objective(int,int [],int []);
    void display(int,int [],int [],int [],float []);
    void line();
};

knapsack :: knapsack()
{
  cout<<"\n Enter the bag capacity :\t";
  cin>>cap;
}

void knapsack :: select(int n,int w[])
{
  int u,i;
  u=cap;
  for(i=0;i<n;i++)
    x[i]=0;
  for(i=0;i<n;i++)
  {
    if(w[i]>u)
      break;
    else
    {
      x[i]=1;
      u=u-w[i];
    }
  }
  if(i<=n)
    x[i]=(float)u/w[i];
}

void knapsack :: objective(int n,int no[],int p[])
{
  int i,j;
  for(i=0;i<n;i++)
  {
    for(int j=i+1;j<n;j++)
    {
      if(no[i]>no[j])
      {
          swap(no,i,j);
          swap(x,i,j);
          swap(p,i,j);
      }
    }
  }
  float opt_solution=0;
  cout<<"\nThe Order is\n\n";
  cout<<"(";
  for(i=0;i<n;i++)
    cout<<no[i]<<"  ";
  cout<<")";
  cout<<"\t\t(";
  for(i=0;i<n;i++)
  {
    cout<<x[i]<<"  ";
    opt_solution=opt_solution+(p[i]*x[i]);
  }
  cout<<")";
  cout<<"\n\nThe optimal solution is\t"<<opt_solution;
}
void knapsack :: display(int n,int no[],int w[], int p[],float pwr[])
{
  line();
  cout<<"\n\t\t\t\tKnapsack\n";
  line();
  cout<<"\nNumber Of Products:\t"<<n<<"\nBag Max Capacity:\t"<<cap<<"\n";
  line();
  cout<<"\nSl.No\t\tWeight\t\tProfit\t\tPWR\n";
  line();
  for(int i=0;i<n;i++)
    cout<<no[i]<<"\t\t"<<w[i]<<"\t\t"<<p[i]<<"\t\t"<<pwr[i]<<"\n";
  line();
}
```

III. CONCLUSION

Fractional Knapsack has time complexity O (N logN) where N is the number of items. I observe that a greedy strategy does not in general produce an optimal solution, but a greedy algorithm may yield locally optimal solutions that approximate a global optimal solution in a reasonable time. It is a way of making the locally optimal choice at each stage with the hope of finding an optimal solution.

1) Which Algorithm we can use to solve this problem?
2) In how many processes we can complete our task?
3) What is the feasible solution for this problem?
4) What is optimal outcome of problem solving Algorithm?

REFERENCES

[1] Multitasking/multithreading in C on Arduino, **http://playwithmyled.com/2009/10/multitasking-multithreading-in-c-on-arduino/, 2009**

[2] Multithreaded C Program on the Raspberry PI, **http://stackoverflow.com/questions/18423885/multithreaded-c-program-on-the-raspberry-pi, 2015**

[3] Process (computing), **https://en.wikipedia.org/wiki/Process_(computing), 2015**