June16

15 June 2020 23:14

Greedy Algorithms

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

Greedy Algorithm to find Minimum number of Coins

Given a value V, if we want to make a change for V Rs, and we have an infinite supply of each of the denominations in Indian currency, i.e., we have an infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?

```
Input: V = 70
Output: 2
We need a 50 Rs note and a 20 Rs note.

Input: V = 121
Output: 3
We need a 100 Rs note, a 20 Rs note and a 1 Rs coin.
```

Approach: A common intuition would be to take coins with greater value first. This can reduce the total number of coins needed. Start from the largest possible denomination and keep adding denominations while the remaining value is greater than 0.

Algorithm:

- Sort the array of coins in decreasing order.
- Initialize result as empty.
- Find the largest denomination that is smaller than current amount.
- Add found denomination to result. Subtract value of found denomination from amount.
- If amount becomes 0, then print result.
- Else repeat steps 3 and 4 for new value of V.

Complexity Analysis:
Time Complexity: O(N*logN).
Auxiliary Space: O(1) as no additional space is used.

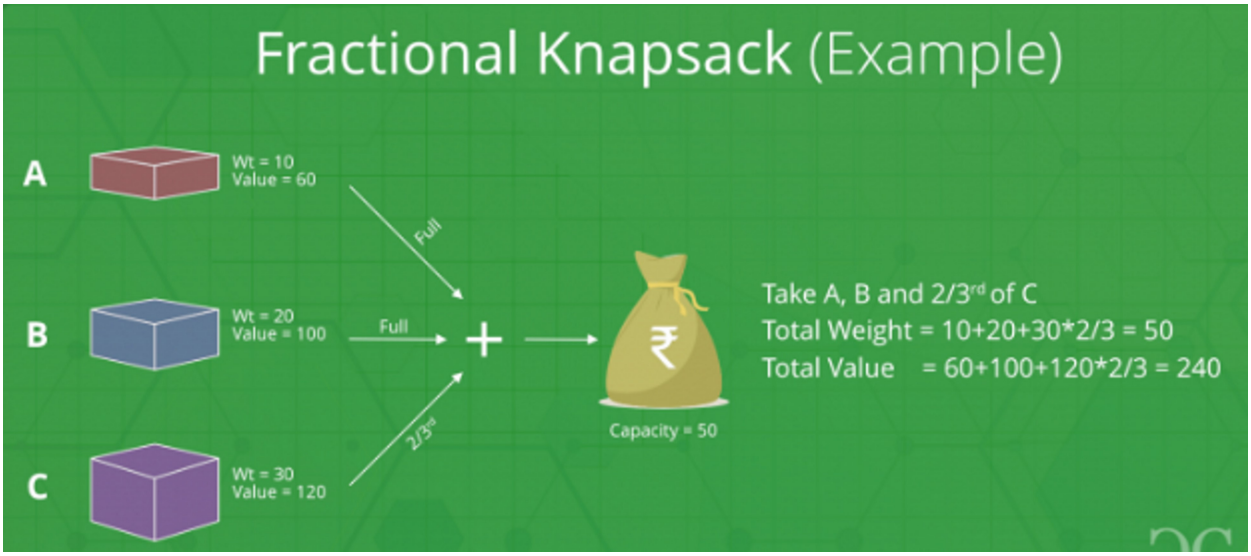
Complexity Analysis:
Time Complexity: O(N*logN).
Auxiliary Space: O(1) as no additional space is used.

Note: The above approach may not work for all denominations. For example, it doesn't work for denominations {9, 6, 5, 1} and V = 11. The above approach would print 9, 1 and 1. But we can use 2 denominations 5 and 6.

Fractional Knapsack Problem

Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

In the **0-1 Knapsack** problem, we are not allowed to break items. We either take the whole item or don't take it. In **Fractional Knapsack**, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.



An efficient solution is to use **Greedy approach.** The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.

Bin Packing Problem (Minimize number of used Bins)

Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.
Example:

```
Input:  wieght[]       = {4, 8, 1, 4, 2, 1}
        Bin Capacity c = 10
Output: 2
We need minimum 2 bins to accommodate all items
First bin contains {4, 4, 2} and second bin {8, 2}

Input:  wieght[]       = {9, 8, 2, 2, 5, 4}
        Bin Capacity c = 10
Output: 4
We need minimum 4 bins to accommodate all items.

Input:  wieght[]       = {2, 5, 4, 7, 1, 3, 8};
        Bin Capacity c = 10
Output: 3
```

Lower Bound

We can always find a lower bound on minimum number of bins required. The lower bound can be given as :

```
Min no. of bins  >=  Ceil ((Total Weight) / (Bin Capacity))
```

In the above examples, lower bound for first example is "ceil(4 + 8 + 1 + 4 + 2 + 1)/10" = 2 and lower bound in second example is "ceil(9 + 8 + 2 + 2 + 5 + 4)/10" = 3.

This problem is a NP Hard problem and finding an exact minimum number of bins takes exponential time. Following are approximate algorithms for this problem.

Online Algorithms

These algorithms are for Bin Packing problems where items arrive one at a time (in unknown order), each must be put in a bin, before considering the next item.

1. Next Fit:

When processing next item, check if it fits in the same bin as the last item. Use a new bin only if it does not.

Next Fit is a simple algorithm. It requires only O(n) time and O(1) extra space to process n items.

2. First Fit:

When processing the next item, scan the previous bins in order and place the item in the first bin that fits. Start a new bin only if it does not fit in any of the existing bins.

3. Best Fit:

The idea is to places the next item in the *tightest* spot. That is, put it in the bin so that smallest empty space is left.

Offline Algorithms

In the offline version, we have all items upfront. Unfortunately offline version is also NP Complete, but we have a better approximate algorithm for it. First Fit Decreasing uses at most (4M + 1)/3 bins if the optimal is M.

4. First Fit Decreasing:

A trouble with online algorithms is that packing large items is difficult, especially if they occur late in the sequence. We can circumvent this by *sorting* the input sequence, and placing the large items first. With sorting, we get First Fit Decreasing and Best Fit Decreasing, as offline analogs of online First Fit and Best Fit.

Applications

- Loading of containers like trucks.
- Placing data on multiple disks.
- Job scheduling.
- Packing advertisements in fixed length radio/TV station breaks.
- Storing a large collection of music onto tapes/CD's, etc.

Questions to implement

1.Program for Greedy Algorithm to find Minimum number of Coins
Given a value V, if we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?
Examples:
Input: V = 70
Output: 2
We need a 50 Rs note and a 20 Rs note.
Input: V = 121
Output: 3
We need a 100 Rs note, a 20 Rs note and a
1 Rs coin.
2.Implement fractional knapsack problem using greedy approach
Input: items[] = [[60, 10], [100, 20], [120, 30]]
Knapsack Capacity(capacity) = 50
Output: Maximum possible value = 240
Explanation: By taking full items of 10 kg, 20 kg and 2/3rd of last item of 30 kg. Total value = 60 + 100 +
120*2/3 = 240

3.Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.

Input: wieght[] = {4, 8, 1, 4, 2, 1}
Bin Capacity c = 10
Output: 2
We need minimum 2 bins to accommodate all items
First bin contains {4, 4, 2} and second bin {8, 2}