Greedy algorithms

Wednesday, 1 November 2023 7:22 AM

① Introduction to Greedy

— used for optimization problems

Consider infinite supply of the following value coins:

10 5 2 1

If someone asks for an amount, how will you give this amount using minimum coins?

eg: Amount 52

$2 \times 10 + 1 \times 2$

```
def minCoins(coins, amount):
    coins.sort(reverse=True)
    res = 0
    for x in coins:
        if x <= amount:
            c = amount // x
            res += c
            amount -= c * x
        if amount == 0:
            break
    return res
```

```
amount = 57
coins = [5,10,2,1]
After sorting: [10,5,2,1]
res = 0
x = 10
c = 5, res = 5, amount = 7
x = 5
c = 1, res = 6, amount = 2
x = 2
c = 1, res = 7, amount = 0
```

General structure of Greedy Algo

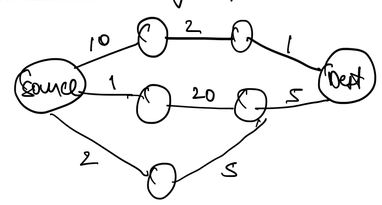
```
def getOptimal(arr):
    res = 0
    while (All items are not considered):
        i = selectAnItem()
        if feasible(i):
            res = res + i
    return res
```

Greedy algo may not work always!

consider: coins[] = {18, 1, 10}

amount = 20

Another ex: longest path



Applications

→ Finding optimal solutions

- Activity selection
- Fractional Knapsack
- Job sequencing
- Prim's Algo
- Kruskal's Algo
- Dijkstra's Algo
- Huffman Coding

→ Finding close to optimal solutions for NP Hard problems like travelling salesman problem

② Activity Selection Problem

I/P: {(2,3), (1,4), (5,8), (6,10)}

o/p: 2

— Maximum no. of activities that can happen on a single working machine

Naive: Consider every activity as first activity

— exponential time

Greedy:

1. Sort according to finish time
   {(1,3), (2,4), (5,8), (10,11)}
2. Initialize solution as first activity
3. Do following for remaining activities:
   (a) if current activity overlaps with last picked activity in solution, ignore current activity
   (b) Else add the current activity to the solution

```
def maxActivity(arr):
    n = len(arr)
    arr.sort(key=lambda x: x[1])
    prev = 0
    res = 1
    for curr in range(1,n):
        if arr[curr][0] >= arr[prev][1]:
            res += 1
            prev = curr
    return res
```

③ Fractional Knapsack Problem

	I_1	I_2	I_3
weights	50	20	30
values	600	500	400

Knapsack capacity = 70

o/p: 1140 → $I_2 + I_3 + I_1 \frac{20}{50}$

$= 500 + 400 + 600 \times \frac{20}{50}$

$= 1140$

① Calculate ratio (value/weight) for every item.
② Sort them in decreasing order of ratio.
③ Initialize, res = 0, curr_cap = given_cap
④ Do following:
   (a) if (I.weight <= curr_cap):
       curr_cap -= I.weight
       res += I.value
   (b) Else { res += (I.value) * (curr_cap / I.weight) }
       return res
⑤ return res

④ Job Sequencing Problem

⑤ Huffman Coding

— used for lossless compression
— variable length coding