The Greedy Method

Joy Mukherjee

The Greedy Paradigm

- Adopting a locally optimum solution leads to a globally optimal solution.
- Optimal Substructure property is maintained

Minimum Coin Change Problem

- Given an infinite supply of coins of different denominations {1, 2, 5, 10}, what is the minimum number of coins needed to have a change for Rs. R?

- R = 98
- 9 Rs 10 coins
- 1 Rs 5 coin
- 1 Rs 2 coin
- 1 Rs 1 coin

Minimum Coin Change Problem: Algorithm

1. Sort the denominations *denom* in decreasing order.
2. For (i = 0; i < n; i++)
    a. T = R / denom[i]
    b. R = R – T*denom[i]
    c. If(R == 0) break;

Time complexity = O(n)

Fractional Knapsack Problem

- Given n items with volume vi and price pi for the i-th item, and given a knapsack of capacity V, our objective is to collect items such that the total price is maximum.
- Unlike 0/1 knapsack, fractional knapsack allows to take a fraction of an item in it.
- V = 50
- Total Cost = 60 + 100 + 20 * (120/30) = 240

Item	Volume	Price	Price/Volume
1	20	100	5
2	30	120	4
3	10	60	6 (Costliest)

Fractional Knapsack Problem: Algorithm

1. Sort the items in the non-increasing order of price/volume.
2. Collect the items in the sorted order unless knapsack is full or the items got exhausted.

- Time Complexity = O(n log n)

Task Selection Problem

- Given n tasks each having a start time and a finish time. Tour goal is to complete as many tasks as possible while respecting the following constraint:
- You can execute at most one task at a time.
- A task i is non-overlapping with task j, if either s(i) > f(j) or s(j) > f(i)

Task	Start Time	Finish Time
1	5	9
2	2	4
3	3	8
4	13	14
5	10	11
6	10	12
7	1	4

Task	Start Time	Finish Time
2	2	4
7	1	4
3	3	8
1	5	9
5	10	11
6	10	12
4	13	14

Task	Start Time	Finish Time
2	2	4
7	1	4
3	3	8
1	5	9
5	10	11
6	10	12
4	13	14

Task Selection Problem: Algorithm

1. Sort the tasks in the non-decreasing order of their finish times.
2. Select the first task.
3. For the remaining tasks in the sorted order
4. If the current task is non-overlapping with the earlier selected one
    a. Select the current task

Time Complexity O(n log n)

Task Selection Problem: Proof of Correctness

T = {t1, t2, …., tn} is the set of tasks sorted by increasing finish times.

Say, A is the solution returned by our algorithm. Task t1 is selected.

Let, B be the optimal solution for the problem. We sort the tasks in B in increasing order of their finish times. Suppose, tk ≠ t1 is the first task in the sorted list of B.

Now I can always replace tk by t1 to get an optimal schedule. Why?

Finish time(t1) <= Finish time(tk)

Egyptian Fraction

- A unit fraction is one having numerator as 1 and denominator is some positive integer.
- Every positive fraction can be represented by a sum of unique unit fractions.
- 2/3 = 1/2 + 1/6
- 12/13 = 1/2 + 1/3 + 1/12 + 1/156

- Input: Numerator 12 Denominator 13
- Output: 1/2, 1/3, 1/12, 1/156

Egyptian Fraction: Algorithm

- Every positive fraction can be represented by a sum of unique unit fractions.
- Input: Numerator 12 Denominator 13
- Output: 1/2, 1/3, 1/12, 1/156

Numerator	Denominator	D/N	D%N	Unit Fraction	N/D – U.F.
12	13	1	1	1/2	12/13 – 1/2
11	26	2	4	1/3	11/26 – 1/3
7	78	11	1	1/12	7/78 – 1/12
7*12 – 78 = 6	78*12 = 936	156	0	1/156	Terminated

Maximum Profit Job Sequencing Problem

- Given n jobs, where each job i has a unit execution time, a deadline di, and an associated profit pi. Given a uniprocessor machine, our task is to execute the jobs such that the total profit is maximized.
- A uniprocessor machine can execute one job at a time.

Maximum Profit Job Sequencing Problem

Job	Deadline	Profit
1	2	100
2	1	20
3	2	30
4	1	25
5	3	50
6	3	100
7	4	5

Job	Deadline	Profit
1	2	100
6	3	100
5	3	50
3	2	30
4	1	25
2	1	20
7	4	5

T1	T2	T3	T4
5	1	6	7

Total Maximum Profit = 255

Maximum Profit Job Sequencing Problem

Job	Deadline	Profit
1	2	100
2	1	120
3	2	30
4	1	25
5	4	50
6	4	100
7	4	5

Job	Deadline	Profit
2	1	120
1	2	100
6	4	100
5	4	50
3	2	30
4	1	25
7	4	5

T1	T2	T3	T4
2	1	5	6

Total Maximum Profit = 370

Maximum Profit Job Sequencing Problem

1. Sort the jobs in non-increasing order of profits.
2. Create an array slot of size maximum deadline+1, and initialize them to -1
3. For(i = 0; i < n; i++) {

```
        For(j = deadline[i]; j >= 1; j--) {
                If(slot[j] == -1)
                        Assign job i to slot j
        }
}
```

Time Complexity = $O(n^2)$