Assignment 11

Qn. R-5.1

Let S = {a, b, c, d, e, f, g} be a collection of objects with benefit-weight values as follows: a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1), g:(9,6). What is an optimal solution to the fractional knapsack problem for S assuming we have a knapsack that can hold objects with total weight 15? Show your work.

To solve the Fractional Knapsack Problem, we follow a greedy algorithm solution where we choose items based on their benefit-to-weight ratio, adding as much of the item as possible until the knapsack is full.

Step 1: Compute Benefit-to-Weight Ratios

Item	Benefit	Weight	Benefit/Weight
a	12	4	3.00
b	10	6	1.67
c	8	5	1.60
d	11	7	1.57
e	14	3	4.67
f	7	1	7.00
g	9	6	1.50

Step 2: Sort by Benefit/Weight Ratio (Descending)

Order:
f (7.00)
e (4.67)
a (3.00)
b (1.67)
c (1.60)
d (1.57)
g (1.50)

Step 3: Greedily Fill the Knapsack (Capacity = 15)

f: weight = 1 → take all → remaining = 14, benefit = 7
e: weight = 3 → take all → remaining = 11, benefit = 7 + 14 = 21
a: weight = 4 → take all → remaining = 7, benefit = 21 + 12 = 33
b: weight = 6 → take all → remaining = 1, benefit = 33 + 10 = 43
c: weight = 5 → can only take 1/5
 benefit = (1/5) × 8 = 1.6
 remaining = 0

total benefit = 43 + 1.6 = 44.6

Final Answer:

Take all of: f, e, a, b
Take 1/5 of c
Total benefit = 44.6 which is the optimal solution

Qn. R-5.3

Suppose we are given a set of tasks specified by pairs of the start times and finish times as T = {(1,2),(1,3),(1,4),(2,5),(3,7),(4,9),(5,6),(6,8),(7,9)}. Solve the task scheduling problem for this set of tasks.

To solve the Task Scheduling Problem, we want to select the maximum number of non-overlapping tasks (i.e., activities that do not conflict in time) and then use a greedy algorithm that selects tasks in increasing order of finish time.

Step 1: List and Sort Tasks by Finish Time

We are given:
T = {(1,2), (1,3), (1,4), (2,5), (3,7), (4,9), (5,6), (6,8), (7,9)}

Sort these by finish time:

Task	Start	Finish
T1	1	2
T2	1	3
T3	1	4
T4	2	5
T5	3	7
T7	5	6
T8	6	8
T6	4	9
T9	7	9

Step 2: Greedy Selection (by Earliest Finish Time)

We go through the sorted tasks and pick a task if its start time ≥ last_finish_time.

Iteration:
(1,2) → start = 1 ≥ 0 → select → selected = [(1,2)], last_finish_time = 2
(1,3) → start = 1 < 2 → skip
(1,4) → start = 1 < 2 → skip
(2,5) → start = 2 ≥ 2 → select → selected = [(1,2), (2,5)], last_finish_time = 5
(3,7) → start = 3 < 5 → skip
(5,6) → start = 5 ≥ 5 → select → selected = [(1,2), (2,5), (5,6)], last_finish_time = 6
(6,8) → start = 6 ≥ 6 → select → selected = [(1,2), (2,5), (5,6), (6,8)], last_finish_time = 8
(4,9) → start = 4 < 8 → skip
(7,9) → start = 7 < 8 → skip

Final Answer:

The optimal set of non-overlapping tasks is: [(1,2), (2,5), (5,6), (6,8)]
Total tasks selected: 4

Qn. R-5.11

Solve Exercise R-5.1 above for the 0-1 Knapsack Problem.

To solve this, we either take the entire item or don't take it at all. No fractional amounts allowed.

Step 1: 0-1 Knapsack with Dynamic Programming

Let's use Dynamic Programming (DP).
- Let n = 7 (number of items)
- Let W = 15 (capacity)
- Let dp[i][w] represent the maximum benefit using first i items with capacity w

We build the dp table from bottom up.

Step 2: Build the DP Table

Let's define arrays:
- weights = [4, 6, 5, 7, 3, 1, 6]
- benefits = [12, 10, 8, 11, 14, 7, 9]

We'll fill a dp[8][16] table (8 rows because we include 0-index row, and 16 columns for weight 0 to 15)

Calculation solution:
include item i if it fits: dp[i][w] = max(dp[i-1][w], dp[i-1][w - weight[i]] + benefit[i])

After filling the table, the maximum benefit is dp[7][15] = 45

Step 5: Trace Back to Find Which Items Were Chosen

We backtrack to find which items make up that 45:

Start at dp[7][15] = 45

Check if dp[7][15] == dp[6][15] → no → Item g was taken (index 6), w = 15 - 6 = 9
Check dp[6][9] == dp[5][9] → no → Item f was taken (index 5), w = 9 - 1 = 8
Check dp[5][8] == dp[4][8] → no → Item e was taken (index 4), w = 8 - 3 = 5
Check dp[4][5] == dp[3][5] → yes → Item d NOT taken
Check dp[3][5] == dp[2][5] → no → Item c was taken (index 2), w = 5 - 5 = 0

Any remaining items not taken.

Final Answer (0-1 Knapsack):

After a variation of different combinations

Items to take:
a:(12,4)
d:(11,7)
e:(14,3)
f:(7,1)

Total Weight = 4 + 7 + 3 + 1 = 15
Maximum Benefit = 44

items

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0								
A	0	0	0	0	12	12	12	12	12	12	12	12	12	12	12	12
B	0	0	0	0	12	12	12	12	12	12	12, 10	12,10	12,10	12,10	12,10	12,10
C	0	0	0	0	12	12	12	12	12	20	22	22	22	22	22	30
D	0	0	0	0	12	12	12	12	12	20	22	23	23	23	23	23
E	0	0	0	14	14	14	14	26	26	26	26	26	34			
F	0	7														
G	0	7	7	7	21	21	21	21	33	33	33	33	33	41	41	44

Qn. R-5.12

Sally is hosting an Internet auction to sell n widgets. She receives m bids, each of the form "I want ki widgets for di dollars," for i = 1, 2, ..., m. Characterize her optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?

Problem Description

- Sally has n widgets.
- She receives m bids.
- Each bid is: "I want k_i widgets for d_i dollars" — for $i = 1, 2, \ldots, m$.

Goal: Maximize Sally's revenue by choosing a subset of these bids without exceeding the n available widgets.

Knapsack Characterization

We can map this to the classic knapsack problem:

Knapsack Concept	Sally's Auction Equivalent
Capacity W	Total number of widgets n
Item i	Bid i
Item weight w_i	Number of widgets requested k_i
Item value v_i	Dollars offered for the widgets d_i
Objective	Maximize total value (total revenue)

So Sally's problem becomes:

Choose a subset of bids such that the total number of widgets sold does not exceed n, and the total revenue is maximized.

0-1 vs. Fractional Knapsack

Scenario	Description	Problem Type
0-1 Knapsack	Sally must either accept or reject each bid in full. She cannot partially fulfill a bid.	0-1 Knapsack
Fractional Knapsack	Sally can partially fulfill a bid (e.g., if someone asks for 10 widgets and she only gives 6, she gets a proportional amount of money).	Fractional Knapsack

In Detail:
If bidders are only willing to pay if they get exactly k_i widgets, then:

- Sally must either accept or reject the bid.
- This is a 0-1 Knapsack Problem.

If bidders accept partial fulfillment and pay proportionally (e.g., \$10 for 5 widgets → \$2 per widget), then:

- Sally can partially fulfill bids.
- This is a Fractional Knapsack Problem.

Final Answer:

Sally's optimization problem is a knapsack problem where:

- The capacity is the number of widgets n.
- Each bid is an item with:
  - weight = number of widgets requested (k_i)
  - value = dollars offered (d_i)

It is:
A 0-1 knapsack problem if Sally can only accept whole bids (no partial sales).
A fractional knapsack problem if partial bids (partial fulfillment) are allowed and paid proportionally.