Greedy Algorithms:
Fractional Knapsack Problem

Dept. of CSE
Varendra University

Course Instructor:

Sumaiya Tasnim

Lecturer, Department of CSE

Varendra University

Acknowledgement

Mosiur Rahman Sweet
Former Lecturer, Department of CSE
Varendra University

Md. Muktar Hossain
Lecturer, Department of CSE
Varendra University

Fractional Knapsack Problem

- You are given a list of products, their values and their weights.
- You have a bag of limited size.

- You need to choose products in such a way that the profit turns out to be **maximum**.

Fractional Knapsack Problem

Product No.	7	6	5	4	3	1	2
Value	120	88	60	36	7	12	12
Weight	12	11	10	9	3	4	6

Bag/Knapsack = 40

Profit = ?

Fractional Knapsack Problem

Product No.	7	6	5	4	3	1	2
Value	120	88	60	36	7	12	12
Weight	12	11	10	9	3	4	6
$\frac{Value}{Weight}$	10	8	6	4	3	3	2
After sorting by $\frac{Value}{Weight}$ in descending order							

Bag= 7-7 = 0

Profit: 268 + (4*7) = 296

Fractional Knapsack Problem

- While knapsack is not full.
  - Choose item i with maximum $v[i]/w[i]$.
  - If the item fits into the knapsack, take all of it.
  - Otherwise take so much as to fill the knapsack.
- Return total value and amounts taken.

- It is to be noted that the list needs to be sorted by $\boldsymbol{Value/Weight}$ in descending order before applying fractional knapsack.

Time Complexity: O(nlogn)

Pseudocode:

```
Function FractionalKnapsack(W, weights[], values[], n)
1. For i = 0 to n-1:
        a. Calculate ratio[i] = values[i] / weights[i]
2. Sort ratio[] in descending order.
3. Initialize totalValue = 0, currentWeight = 0.
4. For each item i in sorted ratio[]:
        a. If currentWeight + weights[i] <= W:
                i. Add values[i] to totalValue.
                ii. Update currentWeight.
        b. Else:
        i. Take fraction of item: totalValue += (W - currentWeight) * ratio[i]
        ii. Break loop.
5. Return totalValue.
End Function
```

Time Complexity

- **Sorting the items**: O(nlogn)
- **Iterating through items**: O(n)
- **Overall Complexity**: O(nlogn)

An Example

- A knapsack is available with a maximum weight capacity of 100 kg.
- There are 6 items, each with a given weight and value.
- The goal is to maximize the total value in the knapsack without exceeding its weight capacity.

Item	Weight (kg)	Value ($)
1	15	120
2	35	300
3	25	200
4	50	500
5	10	90
6	30	400

For i = 0 to n-1:
  Calculate ratio[i] = values[i] / weights[i]

Item	1	2	3	4	5	6
Weight (kg)	15	35	25	50	10	30
Value ($)	120	300	200	500	90	400
v_i/w_i	8.00	8.57	8.00	10.00	9.00	13.33

Sort ratio[] in descending order.

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

Initialize totalValue = 0, currentWeight = 0

totalValue = 0
currentWeight = 0

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 0
currentWeight = 0

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 0
currentWeight = 0

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 400
currentWeight = 30

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 400
currentWeight = 30

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 900
currentWeight = 80

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 900
currentWeight = 80

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	[redacted]	[redacted]	[redacted]
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 990
currentWeight = 90

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 990
currentWeight = 90

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

For each item i in sorted ratio[]:

a. If currentWeight + weights[i] <= W:

i. Add values[i] to totalValue.

ii. Update currentWeight.

b. Else:

i. Take fraction of item:

totalValue += (W - currentWeight) * ratio[i]

ii. Break loop.

W = 100
totalValue = 1075.7
currentWeight = 90

Item	6	4	5	2	1	3
Weight (kg)	30	50	10	35	15	25
Value ($)	400	500	90	300	120	200
v_i/w_i	13.33	10.00	9.00	8.57	8.00	8.00

Return totalValue.

totalValue = 1075.7

W = 100
totalValue = 1075.7
currentWeight = 90

Applications

- **Resource Allocation** – Optimizing project selection, investment planning.
- **Cargo Loading & Logistics** – Maximizing value while staying within weight limits in transport.
- **Manufacturing** – Reducing waste in material cutting (wood, textiles, metals).
- **CPU Scheduling & Memory Management** – Efficient job scheduling in operating systems.
- **Financial Portfolio Optimization** – Selecting the best assets under a budget.
- **Network Bandwidth Allocation** – Distributing bandwidth efficiently in telecommunications.
- **Cryptography** – Used in encryption algorithms (e.g., Merkle–Hellman Knapsack Cryptosystem).
- **Marketing & Ads** – Selecting cost-effective ads under a budget constraint.
- **Robotics & AI** – Optimizing object carrying and path planning.
- **Space Exploration** – Selecting scientific instruments for space missions with weight limits.

Components of Greedy

- **Candidate set:** A solution that is created from the set is known as a candidate set.
- **Selection function:** This function is used to choose the candidate or subset which can be added in the solution.
- **Feasibility function:** A function that is used to determine whether the candidate or subset can be used to contribute to the solution or not.
- **Objective function:** A function is used to assign the value to the solution or the partial solution.
- **Solution function:** This function is used to estimate whether the complete function has been reached or not.

An Example

Problem Statement (Fractional Knapsack) Given ***n*** items, each with a weight ***w[i]*** and a value ***v[i]***, and a knapsack that can carry a maximum weight ***W***, determine the maximum value that can be obtained by putting items (or fractions of them) into the knapsack.

Candidate set: The set of all items *{(w[i], v[i])}* that can be put into the knapsack.
Selection function: Select the item with the highest value-to-weight ratio *(v[i] / w[i])*.

Feasibility function: Before adding an item, check if adding the whole item exceeds the weight capacity W:
- If yes, take only a fraction of it.
- If no, add the whole item.

Objective function:
- Keep track of the total value accumulated in the knapsack.
- Stop when the knapsack is full (or all items are considered).

Solution function: The problem can be broken down into smaller subproblems:
- If we remove the most valuable item, the remaining problem is the same as before but with a smaller capacity.
- The optimal solution to the subproblem contributes to the optimal solution of the main problem.

Thank You