CSCI-4470 HW-3b

Due: Oct 17 2023, Monday

Homework 3b

Jun Wang
ID: 811574679

Upload a soft copy of your answers **(single pdf file)** to the submission link on elc before the due date. The answers to the homework assignment should be your own individual work. Make sure to show all the work/steps in your answer to get full credit. Answers without steps or explanation will be given zero.

Extra credit: There is 5 percentage extra credit if you don't submit hand written homework including the figures. You can use latex or any other tool to write your homework. For figures you can use any drawing tool and include the figure as a jpeg or a png file in your latex file.

1. (20 points) In class we have talked about greedy solution to the fractional knapsack problem. Prove that the greedy choice property for the fractional knapsack problem is optimal.

Greedy Choice Property, For ***the fractional knapsack problem***, always choose the item with the highest value-to-weight ratio first. If the item doesn't fit entirely, take the fraction that fits.

$$\frac{v_i}{w_i}$$

Using the Example from PPT, Given items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$, and a knapsack of capacity $W = 14$, the goal is to maximize the total value of the knapsack. We can take fractions of items.

Proof of Optimality, the value-to-weight ratios for items are calculated as:

$$\frac{v_1}{w_1} = \frac{72}{6} = 12, \frac{v_2}{w_2} = \frac{90}{9} = 10, \frac{v_3}{w_3} = \frac{15}{5} = 3$$

Following the greedy choice property to fill knapsack:
Greedy Strategy:, Sort the items based on their value-to-weight ratio, $\frac{v_i}{w_i}$, in decreasing order. Start filling the knapsack with the item of highest ratio. If the item doesn't fit entirely, take the fraction that fits.

1. Item 1 is chosen entirely because it has the highest ratio (12), it takes 6 out of the total capacity of knapsack $14 - 6 = 8$

2. Item 2 is chosen next. However, only a fraction of it can fit because the knapsack has only 8 units of space left. So, 8 units of weight from Item 2 are added, giving a value of $10 \times 8 = 80$. $8 - 8 = 0$, the capacity of knapsack runs out when the fraction of item 2 is added.

3. Item 3 is not chosen because its ratio (3) is the lowest, and there's no space left in the knapsack.

4. Result: The ***total value*** in the knapsack becomes $72 + 80 = 152$.

Conclusion, by always choosing the item with the highest value-to-weight ratio, we can consider that the knapsack is filled in a way that maximizes its total value without exceeding its weight capacity. Thus, The greedy choice property for the fractional knapsack problem is optimal.

HW-Key:

Assume that using greedy selection property, fraction f_i of i^{th} item is selected followed by j^{th} We will assume that weights total and items selected before and after these items are identical so the total cost is

$$C_1 = ... + f_i \frac{v_i}{w_i} + f_j \frac{v_j}{w_j} + ...$$

We will argue that by changing the order of i^{th} and j^{th} items the cost function actually decreases. We will prove it by contradiction. Lets say we pick j^{th} item before i^{th} item (greedy solution) and let the fraction of weights selected of each item are $f_{i'}$ and $f_{j'}$. Lets say this is an optimal cost and given by

$$C_2 = ... + f_{j'} \frac{v_j}{w_j} + f_{i'} \frac{v_i}{w_i} + ...$$

$$f_i + f_j = f_{i'} + f_{j'}$$

$$f_i - f_{i'} = f_{i'} - f_{j'}$$

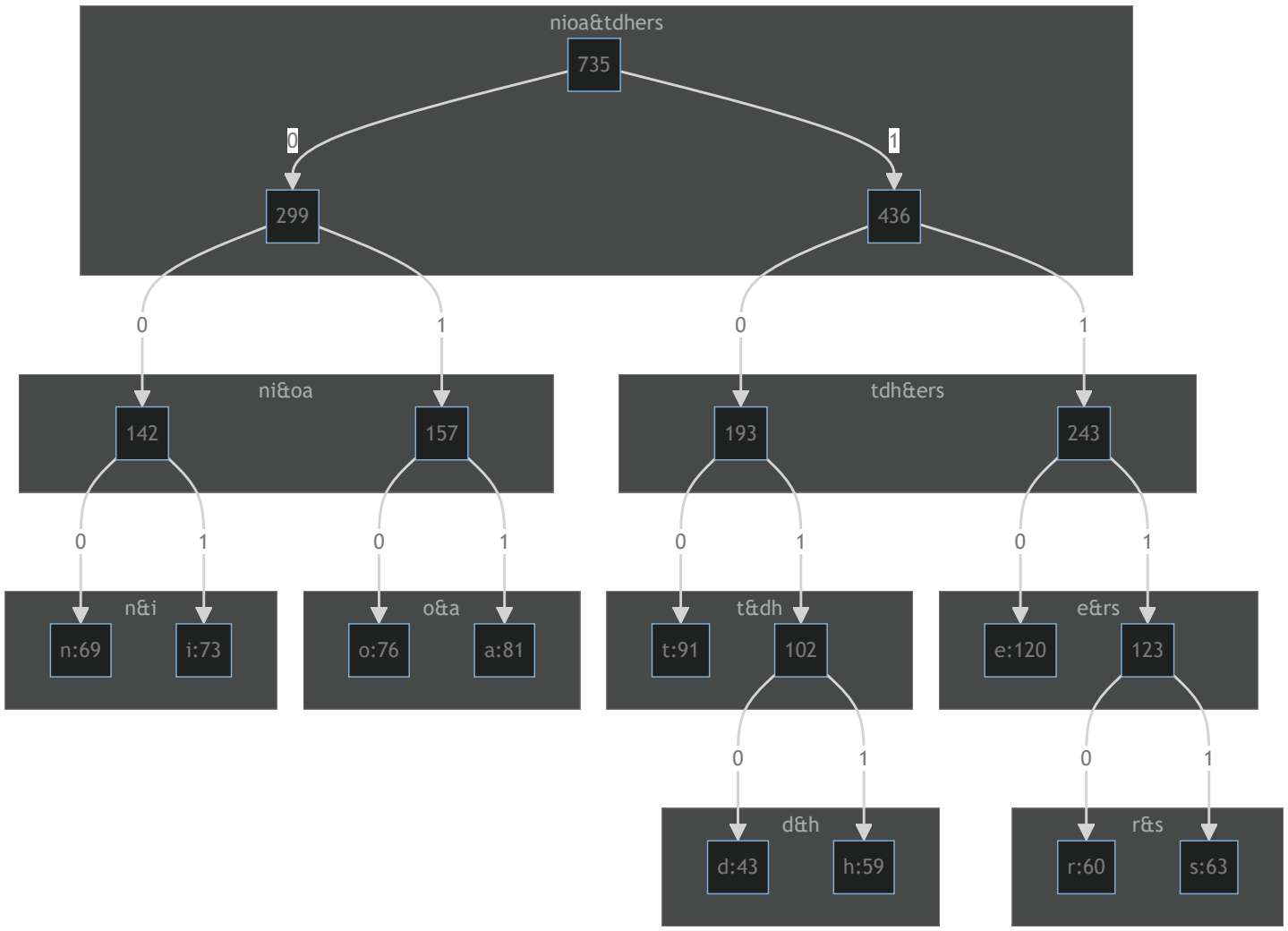
Subtracting C_2 and C_1 we get

$$C_2 - C_1 = f_{j'} \frac{v_j}{w_j} + f_{i'} \frac{v_i}{w_i} - \left(f_i \frac{v_i}{w_i} + f_j \frac{v_j}{w_j} \right)$$

$$\frac{v_j}{w_j} (f_{j'} - f_j) - \frac{v_i}{w_i} (f_i - f_{i'})$$

Using the above $C_2 < C_1$ hence contradicts our assumption.

2. (15 points) What is an optimal Huffman code for the following set of frequencies `a:81 d:43 e:120 h:59 i:73 n:69 o:76 r:60 s:63 t:91` Character "a" has a frequency of 81, character "d" has a frequency of 43 and so on.

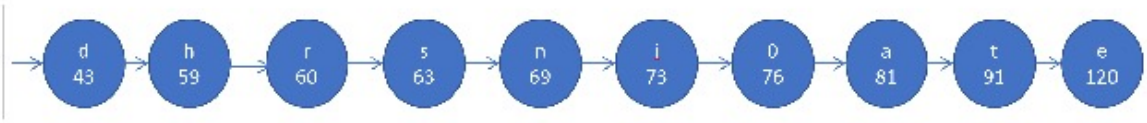


> `a = 011`, `d = 1010`, `e = 110`, `h = 1011`, `i = 001`, `n = 000`, `o = 010`, `r = 1110`, `s = 1111`, and `t = 100`.

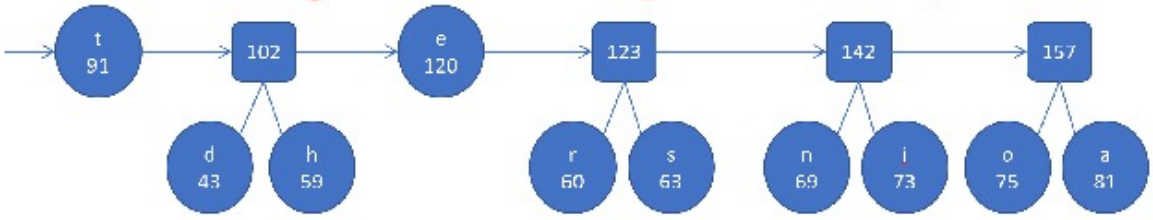
Answer:

e	t	a	o	i	n	s	r	h	d
120	91	81	76	73	69	63	60	59	43

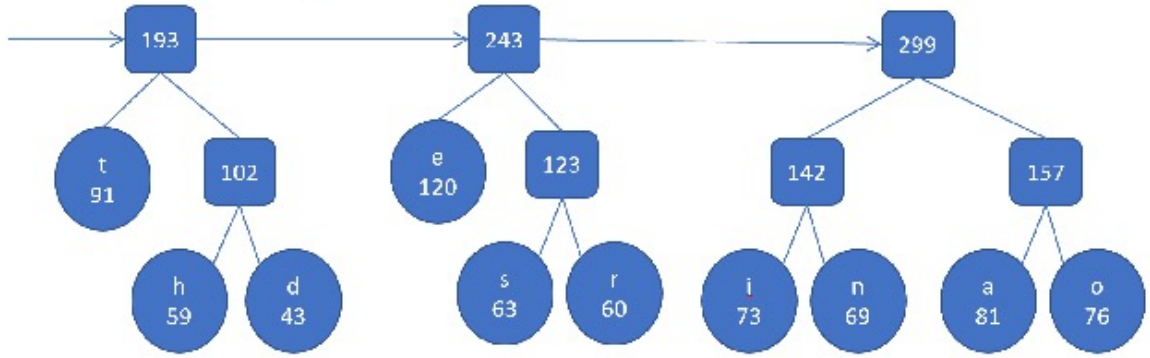
We start with a sorted list of nodes as follows:



After combining the 4 smallest pairs of nodes, the list looks like this:



After combining the next 3 pairs of nodes, the list looks like this:



Finally, after the last 2 steps combining these nodes, we have our Huffman tree:

3. (15 points) There are set of jobs to perform $J_1, \ldots, J_n$. Each job J_i takes d_i days to complete. For each day J_i waits, you incur a penalty of p_i. You can only work on one job at a time. You want to determine which order to perform the jobs with the aim of minimizing your penalties. Find a greedy strategy for this problem. Include selection function, objective function etc in your solution. Hint: You can do a brute force technique with a small example and try to identify the greedy solution.

Initialization and Understanding the Problem:

- We have n jobs, each with a duration d_i and a penalty p_i for each day it waits.
- The objective is to find an order to perform these jobs to minimize the total penalty.
- Let S be the sequence of scheduled jobs, initially empty.
- Let $J = \{J_1, J_2, \ldots, J_n\}$ be the set of jobs.

- Let $d = \{d_1, d_2, ..., d_n\}$ be the set of durations for each job.
- Let $p = \{p_1, p_2, ..., p_n\}$ be the set of penalties for each job.

Greedy Strategy:

Greedy Choice Property:

The greedy choice property emphasizes making a locally optimal decision in anticipation of achieving a globally optimal solution. In this context, the strategy is to prioritize unscheduled jobs based on their penalty-to-day ratio, $\frac{p_i}{d_i}$. A higher ratio indicates that postponing the job would substantially increase the overall penalty.

HW-Key:

If all the penalties are 1, then this problem becomes the “minimizing time in the system” problem, so you’d select in increasing order of d_i. On the other hand, if all the times d_i are 1, you minimize penalty by selecting in increasing order of p_i. This shows you need to do something that combines penalty and time in your solution.

This solution is similar to the knapsack problem. In the knapsack problem you wanted to maximize the profit per weight ratio. In this case, you want to minimize your penalty per wait-time ratio $\frac{p_i}{d_i}$. So you select jobs in decreasing order by $\frac{pi}{di}$.

Selection function: $max(\frac{p_i}{d_i})$

Objective function: $min \sum_{i=1}^{n} \left(\sum_{j=1}^{i} d_j \right) p_i$

For each job, calculate the Penalty-to-Day Ratio:

$$r_i = \frac{p_i}{d_i}$$

The objective is to select the job with the highest r_i among the unscheduled jobs.

Step 1 Selection Function: At each step, select the job with the highest penalty-to-day ratio, i.e., $\frac{p_i}{d_i}$. This ensures that jobs which have a high penalty but take less time are prioritized.

Rationale: Jobs with a high penalty-to-day ratio, $\frac{p_i}{d_i}$, indicate that they incur a significant penalty even if they take a short duration. Thus, prioritizing such jobs can help in minimizing the overall penalty.

1. **Calculate the Penalty-to-Day Ratio for Each Unscheduled Job:** For every job J_k in the set of unscheduled jobs U, compute the ratio:

$$r_k = \frac{p_k}{d_k}$$

2. **Select the Job with the Maximum Ratio:** Identify the job J_i that has the highest penalty-to-day ratio among all unscheduled jobs. Mathematically, this can be represented as:

$$J_i = \max_{J_k \in U} \left(\frac{p_k}{d_k} \right)$$

3. **Update the Sets:** Once J_i is identified:
   - Remove J_i from the set of unscheduled jobs U.
   - Add J_i to the sequence of scheduled jobs S.
4. **Repeat Until All Jobs are Scheduled:** Continue this process until the set U is empty, ensuring that all jobs are scheduled based on the defined criterion.

Step 2: Objective Function Calculating Total Penalty
The objective function quantifies the goal of our problem. In this context, our aim is to minimize the total penalty incurred due to the waiting time of jobs.

Rationale: Each job incurs a penalty for every day it waits before being executed. The longer a job waits, the higher the accumulated penalty. Therefore, the sequence in which jobs are executed directly impacts the total penalty.

1. **Calculate Penalty for Each Job:** For every job J_{ij} in the sequence S, the penalty is determined by the product of its penalty rate p_{ij} and the total days it waited before execution. The waiting time is the sum of durations of all preceding jobs minus its own duration. Mathematically:

$$\text{Penalty}(J_{ij}) = p_{ij} \times \left(\sum_{k=1}^{j} d_{ik} - d_{ij} \right)$$

2. **Compute Total Penalty for the Sequence:** Sum up the penalties of all jobs in the sequence S to get the total penalty:

$$P(S) = \sum_{j=1}^{n} \text{Penalty}(J_{ij})$$

3. **Determine Optimal Sequence:** The optimal sequence S^* is the one that results in the minimum total penalty among all possible sequences:

$$S^* = \min_{S} P(S)$$

Goal: Find the sequence S^* that minimizes the total penalty P.

Brute Force Technique: The brute force approach involves evaluating all possible job sequences to determine the one that results in the minimum penalty.

1. **Enumerate All Possible Sequences**:
   For a given set of jobs, generate all possible sequences S.

2. **Compute Penalty for Each Sequence**:
   For each sequence S, calculate the total penalty $P(S)$.
3. **Identify the Optimal Sequence**:
   The sequence S^* with the least penalty is the optimal solution:

$$S^* = \min_S P(S)$$

Limitations: While this method is straightforward for a small number of jobs, its computational complexity grows factorially with the number of jobs, making it impractical for larger datasets.

Example: Consider three jobs with the following properties:

Job	Duration (days)	Penalty
J1	2	5
J2	4	1
J3	1	10

Step 1: Calculate the penalty-to-day ratio for each job:

$$r_1 = 2.5, r_2 = 0.25, r_3 = 10$$

Step 2: Sort jobs based on the ratio:

- The order based on descending ratios is J_3, J_1, J_2.

Step 3: Schedule jobs and compute penalties:

- J_3: Scheduled first with a penalty of 0.
- J_1: Scheduled next. During its execution, J_2 incurs a penalty of 2 (2 days x p_2).
- J_2: Scheduled last. By its start time, 3 days have elapsed, incurring an additional penalty of 3 (3 days x p_2).

Total penalty = 2 + 3 = 5.

Step 3: Feasibility Check The feasibility check ensures that jobs are scheduled without overlap, meaning only one job is executed at any given time.

1. **Define Start and Finish Times**: For each job J_i in sequence S:
   - The start time of J_i is the finish time of J_{i-1}.
   - The finish time of J_i is its start time plus its duration d_i.
2. **Ensure No Overlap**: For each job J_i in S, ensure that the start time of the next job J_{i+1} is not earlier than the finish time of J_i. Mathematically:

$$\text{Start time of } J_{i+1} \geq \text{Finish time of } J_i$$

3. **Express Start and Finish Times in Terms of Durations**: The finish time of job J_i is the cumulative duration of all preceding jobs including itself:

$$\text{Finish time of } J_i = \sum_{k=1}^{i} d_k$$

Given that each job starts immediately after the previous one finishes:

$$\text{Start time of } J_{i+1} = \text{Finish time of } J_i$$

Goal: Ensure that the scheduling of jobs adheres to the constraint that jobs do not overlap, preserving the integrity of the solution.

Step 4: Solution Function Output, Return the sequence S and the total penalty P.

This series of steps, expressed mainly in mathematical terms, provides a detailed solution to the problem using the greedy strategy. The order of jobs that results in the minimum total penalty.

Greedy Solution Procedure:

1. **Calculate Penalty-to-Day Ratio** For each job J_i, compute the ratio:

$$r_i = \frac{p_i}{d_i}$$

This ratio represents the penalty incurred for each day the job is delayed.
2. **Sort Jobs by Ratio:** Arrange the jobs in descending order based on their penalty-to-day ratios. This ensures that jobs with higher penalties relative to their durations are prioritized.
3. **Schedule Jobs:** Initialize two sets: $S = \emptyset$ (scheduled jobs) and $U = J$ (unscheduled jobs).
While U is not empty:
   - Select J_i from U with the highest r_i.
   - Add J_i to S and remove it from U.

Conclusion:

The greedy approach, based on the penalty-to-day ratio, offers a heuristic solution to the problem. While it's a logical strategy, it's crucial to validate its effectiveness across various scenarios. It may not always guarantee the optimal solution, and in such cases, alternative optimization techniques might be more appropriate.

Note: Validating the correctness of this greedy strategy is essential. Testing it against a brute force approach, especially for smaller datasets, can help in this validation.