Algorithms (6470) HW03b

Alex Darwiche

July 19, 2024

Answers

Q1

(a) Prove the greedy choice property for the fraction knapsack is optimal:

(b) For this proof, we will use contradiction.

(c) The proof begins with the following assumptions:

(1) The greedy selection criteria is $max(\frac{v_i}{w_i})$

(2) Knowing that, we can assume that there is an item i with with the highest $\frac{v_i}{w_i}$.

(d) For contradiction, we want to prove that an optimal solution exists that doesn't have the max amount of item i as possible.

(e) Assumption: There exists a solution where $V_2 > V_1$ given a lesser amount of item i.

(f) Step 1: Let's subtract $V_2 - V_1$

(1) With this subtraction, we will be left with:

(2) $V_2 - V_1 = (y - x)(v_i) + (x)(v_j) - (y)(v_i)$

(3) In this equation, y = the weight of item i in the V_1 and x = the weight of item j used to replace item i in V_2.

(4) Simplify, $V_2 - V_1 = y * v_i - x * v_i + x * v_j - y * v_i = x * v_j - x * v_i$

(5) So, with the assumption that item i has the $max(\frac{v_i}{w_i})$, then $x * v_j - x * v_i < 0$ given they both have the same amount x.

(6) So, $V_2 - V_1 < 0$ or $V_2 < V_1$. This contradicts are earlier statement, thus proving the greey choice property of the fractional knapsack problem.

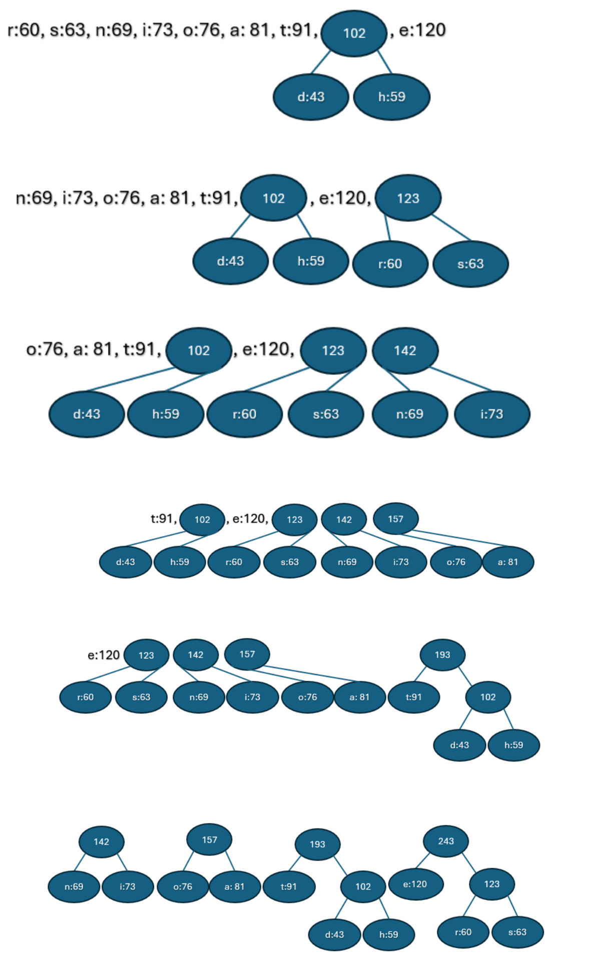
Q2

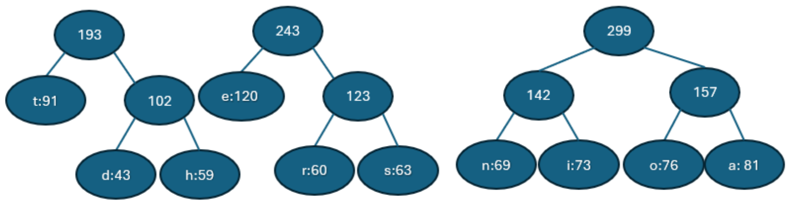
(a) Huffman Encoding:

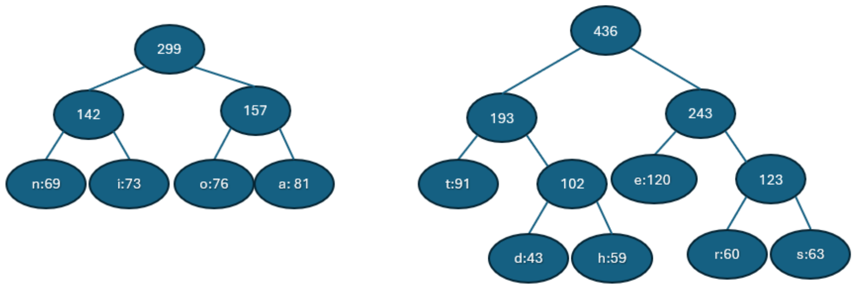
a: 81, d:43, e:120, h:59, i:73, n:69, o:76, r:60, s:63, t:91

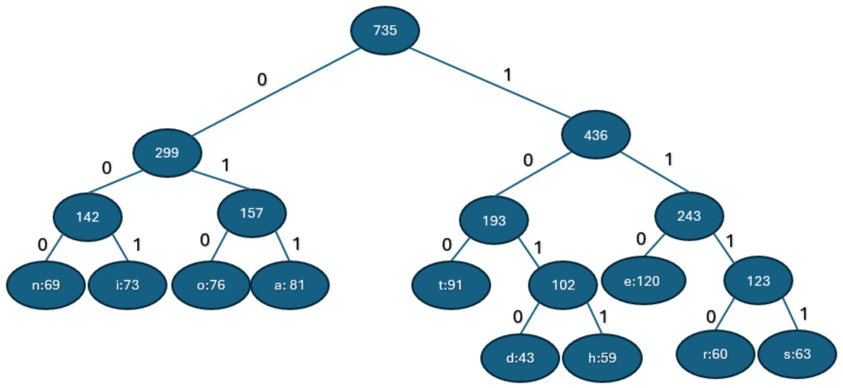
Sort the characters by frequency

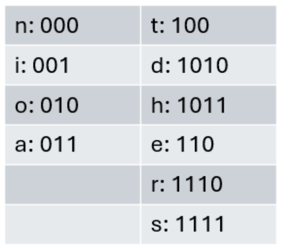
d:43, h:59, r:60, s:63, n:69, i:73, o:76, a: 81, t:91, e:120


r:60, s:63, n:69, i:73, o:76, a: 81, t:91, 102, e:120
d:43
h:59
n:69, i:73, o:76, a: 81, t:91, 102, e:120, 123
d:43
h:59
r:60
s:63
o:76, a: 81, t:91, 102, e:120, 123, 142
d:43
h:59
r:60
s:63
n:69
i:73
t:91, 102, e:120, 123, 142, 157
d:43
h:59
r:60
s:63
n:69
i:73
o:76
a: 81
e:120, 123, 142, 157, 193
r:60
s:63
n:69
i:73
o:76
a: 81
t:91
102
d:43
h:59
142
n:69
i:73
157
o:76
a: 81
193
t:91
102
d:43
h:59
243
e:120
123
r:60
s:63









n: 000	t: 100
i: 001	d: 1010
o: 010	h: 1011
a: 011	e: 110
	r: 1110
	s: 1111

Q3

(a) Find the greedy strategy for this problem:

(1) Assumption: Each day waited, incurs a penalty p_i for job j_i.

(2) Assumption: Each job j_i takes time d_i days to complete.

(3) Assumption: Only 1 job can be worked on at a time.

(4) The objective function is to Minimize Total penalty: $min(\sum_1^n P_i)$

(5) P = Total penalty incurred by each task

(5) P = The total time used to complete previous tasks multiplied by whatever the value of p_i is.

(6) Given the objective is to minimize total penalty, we need to look for a greedy selection criteria that minimizes this penalty. To do this, I worked out a few brute force problems and decided on the following selection criteria:

(7) Greedy Selection Criteria: $min\frac{d_i}{p_i}$

(8) This essentially means that we are looking for the jobs with the lowest days per point of penalty. This approach will minimize the total penalty in the problem.

Q1 (Graduate students only)

(a) This problem introduces value into the traditional "activity selector" problem. This value component requires that look at bit deeper when determining how exactly we will generate an optimal solution A.

(b) The solution to this problem will essentially have a few moving parts, but will be similar to some of the dynamic programming work that we've done in the previous homeworks.

(c) Given: activities = $< a_1, a_2, a_3, ..., a_n >$ values = $< v_1, v_2, v_3, ..., v_n >$

(d) Given: start = $< s_1, s_2, s_3, ..., s_n >$ finish = $< f_1, f_2, f_3, ..., f_n >$

(e) The below algorithm should be able to return the highest value that is achieveable with the current set of activities. This algorithm does not currently hold any of the actual assignment data, it simply returns the highest value. To extend it to hold assignment data, I would need to have some sort of assignment matrix that tracks whether an activity is part of the current working solution of not. This algorithm works by essentially breaking the problem down into smaller subparts. So at each time step, we loop through each of the activities and determine which of them we can assign.

(f) At each time step, we look at all tasks and determine if the value of the current task and the max value of the times before its start time, is greater than any value we've gotten before. If so, then that becomes the new max value, if not, we defer to the current max.

(g) Additional Assumptions:

(1) We sort the list by earliest finish date

(2) The list if sorted using an $O(nlog(n))$ algorithm

(3) This algorithm has nested for loops that each depend on n.

(4) Given these loops, this algorithm is $O(n^2)$

```
def valueActivitySelector(s, f, v):

    times = set(s,f) // find all time indices
    currentMax = 0

    # Set max values to 0 at each time step
    for i in times:
        maxVal[i] = 0

    # Iterate through time indices
    for i in times:
        for j in range(len(s)):
            start = s[j]
            finish = f[j]
            value = v[j]

            if finish <= i:
                maxValue = max(
                    value+maxVal[start],
                    currentMax
                    )
                maxVal[i] = maxValue
        currentMax = maxVal[i]

    return currentMax
```