CS3510-A: Design and Analysis of Algorithms, Spring 2021

Homework-8

March 7, 2021

DUE DATE: Wednesday, March 17, 11:59pm

Note-1: Your homework solutions should be electronically formatted as a single PDF document that you will upload on Gradescope. If you have to include some handwritten parts, please make sure that they are very clearly written and that you include them as high resolution images.

Note-2: Please think twice before you copy a solution from another student or resource (book, web site, etc). It is not worth the risk and embarrassment.

Note-3: You need to **explain/justify** your answers. Do not expect full credit if you just state the correct answer. This includes their correctness and runtime.

Note-4: You will get 2 extra points if you submit electronically typed solutions instead of hand-written.

Problem-1 (30 points)

Using Huffman encoding of n symbols with frequencies $f_1, f_2, ...f_n$, what is the length of the longest possible code-word? Additionally, what values of $f_1, f_2, ...f_n$ lead to this length? Please justify why these values of f_i lead to the maximum possible length code-word.

Solution:

The longest possible code word will be $n - 1$.
The values of $f_1..f_n$ should be $f_x \leftarrow \frac{1}{2^x}$.

Explanation:

The Huffman algorithm for building an encoding tree makes a new merged node with the two nodes that have the lowest proabibilities. By having probabilities that are each half of the next highest probability, the algorithm will merge each node with a subnode that includes all nodes with lower probabilities. This will result in a tree that is the maximum possible depth and therefore have the longest possible code-word.

Problem-2 (35 points)

Suppose you're consulting for a company that manufactures PC equipment and ships it to distributors all over the country. For each of the next n weeks, they have a projected supply s_i of equipment (measured in pounds), which has to be shipped by an air freight carrier.
Each week's supply can be carried by one of two air freight companies, A or B.

- Company A charges a fixed rate r per pound (so it costs $r * s_i$ to ship a week's supply s_i).
- Company B makes contracts for a fixed amount c per week, independent of the weight. However, contracts with company B must be made in blocks of four consecutive weeks at a time.

A schedule, for the PC company, is a choice of air freight company (A or B) for each of the n weeks, with the restriction that company B, whenever it is chosen, must be chosen for blocks of four contiguous weeks at a time. The cost of the schedule is the total amount paid to company A and B, according to the description above.

Example: Suppose $r = 1, c = 10$, and the sequence of values is 11,9,9,12,12,12,12,9,9,11. Then the optimal schedule would be to choose company A for the first three weeks, then company B for a block of four consecutive weeks, and then company A for the final three weeks.

Give a **polynomial-time** algorithm that takes a sequence of supply values $s_1, s_2, ..., s_n$, which represent the supply for n weeks, and returns a schedule of minimum cost. Note that you must not only return the **minimum cost** itself, but also the **schedule**.

Please provide your answer in four sections:

1. Dynamic Programming Table: Must state the dimensions of the table, and explain what each element of this table represents.
2. Recurrence Equation: Express mathematically each value in the table as a function of previous values in the table (do not forget to write down the equation(s) for the base case of the recursion).
3. Pseudo-code: Construct your algorithm: takes $s_1, s_2, ..., s_n$, n, r, c as input and it should return the min cost as well as the schedule.
4. Run-Time Analysis: Explain why your algorithm is polynomial time.

Solution:

Dynamic Programming Table:

The table for the solution of this problem will be one dimensional. The table will essentially be an array with the optimal cost at each week for the given input data.

Recurrence Equation:

For a week i the optimal cost for that week can be calculated with: $min(rw_i + OPT_{i-1}, 4c + OPT_{i-4})$. With a base case of $OPT_0 = 0$.

Pseudo-code:

```
def solve(weights, r, c):
    OPT = {0:0}
    out = ""
    for i, w in enumerate(weights):
        i = i + 1
        a, b = float('inf'), float('inf')
        if i-1 in OPT:
            a = (r*w) + OPT[i-1]

        if i-4 in OPT:
            b = (c*4) + OPT[i-4]

        m = min(a, b)
        OPT[i] = m
        selection = "ab"[[a, b].index(m)]
        out += selection

    return out, m
```

Run-Time:

This algorithm will do $O(1)$ work for each of the n weeks given. Therefore, the overall runtime is $O(n)$.

Problem-3 (35 points)

Recall that in the basic Load Balancing Problem, we're interested in placing jobs on machines so as to minimize the makespan— the maximum load on any one machine. In a number of applications, it is natural to consider cases in which you have access to machines with different amounts of processing power, so that a given job may complete more quickly on one of your machines than on another. The question then becomes: How should you allocate jobs to machines in these more heterogeneous systems?

Here's a basic model that exposes these issues. Suppose you have a system that consists of m slow machines and k fast machines. The fast machines can perform twice as much work per unit time as the slow machines. Now you're given a set of n jobs; job i takes time t_i to process on a slow machine and time $(1/2)t_i$ to process on a fast machine. You want to assign each job to a machine; as before, the goal is to minimize the makespan – that is the maximum, over all machines, of the total processing time of jobs assigned to that machine.

Describe a **polynomial-time algorithm** that produces an assignment of jobs to machines with a makespan that is at most three times the optimum. Your solution should be of the form:

- Algorithm and runtime: Briefly describe how you will assign tasks to machines – and explain why your algorithm can run in poly-time. (This part can be done with 1-2 sentences only!)
- Proof of Upper Bound: Show why your greedy solution can have a makespan that is at most 3 times the optimal solution's makespan.

Solution:

Algorithm 1 Sorted-Balance for Heterogeneous setting

```
Start with no jobs assigned
T_i = 0 and A(i) = ∅ for all machines M_i
Sort jobs in decreasing order of processing times t_j
Assume that t_1 ≥ t_2 ≥ ... ≥ t_n
for j = 1, ..., n do
    F_i ← fast machine with the minimum min_k T_k
    S_i ← slow machine with the minimum min_k T_k
    if T_S + t_j > T_F + t_j/2 then
        M_i ← F_i
    else
        M_i ← S_i
    end if
    Assign job to M_i
    Set A(i) ← A(i) ∪ j
    if M_i is a fast machine then
        Set T_i ← T_i + t_j/2
    else
        Set T_i ← T_i + t_j
    end if
end for
```

Explanation:

This algorithm is the same as the one from section 11.1 of the book. However this algorithm accounts for faster machines being able to process double the amount of work by adding half of the job load to the appropriate T when a fast machine is chosen.

Runtime:

- Sorting the jobs in decreasing order can be done in $O(nlogn)$
- For n jobs:
  - The machine with minimum job load can be found and removed with a Fibonnaci heap in $O(logm)$ or $O(logk)$
  - The machine job load is then updated with the current job added as a new point in a Fibonnaci heap in $O(1)$.

These runtimes reduce down to $O(n)$ which is polynomial time.

Proof of Upper Bound:

Where t_j is the load of the largest job. If the largest job were placed on the fastest machine, then $t_j \leq 2T^*$ (where T^* is the makespan of the optimal solution).

Where T_i is the makespan of the machine that has recieved job j, $T_i - t_j \leq T^*$ (the load on the machine before it recieves job j will be less than the optimal makespan).
This can be rewritten as: $T_i - 2T^* \leq T^*$. And then reduced to: $T_i \leq 3T^*$.
Thus, the makespan of this greedy solution is at most 3 times the optimal solution's makespan.