C. Warren Hammock

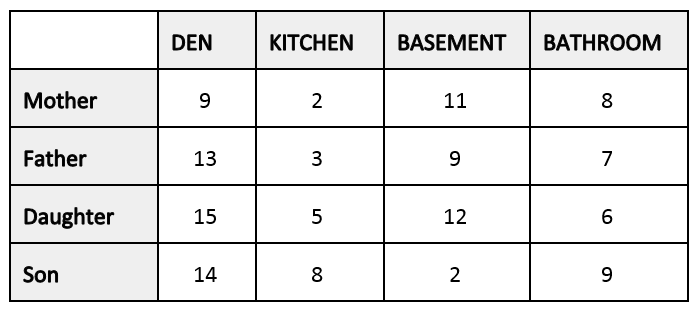
CS520

HW 4

April 20, 2018

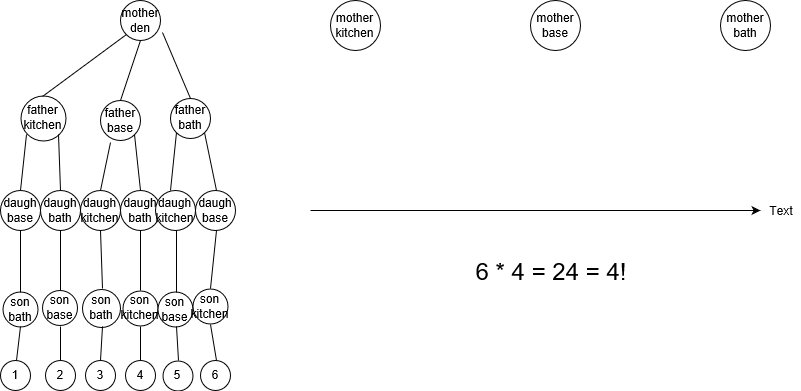
1. A family, decides to repaint their home. There are four tasks to be performed in their house: paint the kitchen, den, bathroom and basement. The family members hold jobs, go to school etc, so each painting task has an associated opportunity cost (i.e. cost of painting a room instead of earning money from working or, going to school). The table below contains the tasks, family members and the associated opportunity costs.

DEN KITCHEN BASEMENT BATHROOM



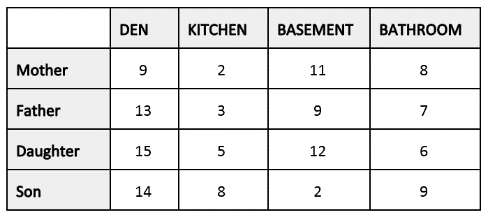
We wish to assign one task to each family member.

a. (5 pts) How many possible solutions exist? Explain your answer. Answer is calculated as n! (factorial of n) Total = 4! = 24,

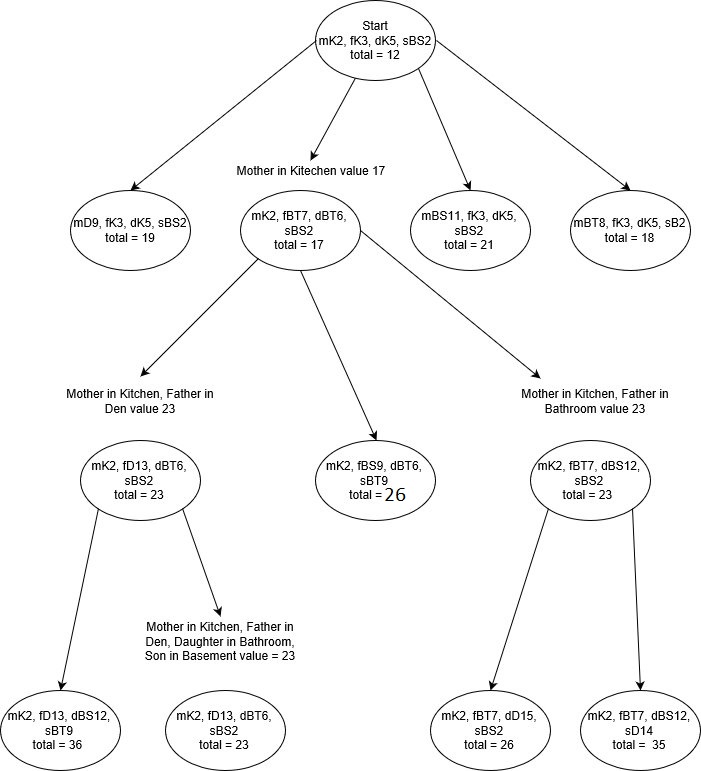


Answer = 24

b. (25 pts) Using the method presented in class provide an optimal solution (a set of task assignments) that assigns one task to each person, and minimizes the sum of the costs. You must represent your solution as a state space diagram with information such as the lower bound and each cost element that is added to generate the cost for the node.



Optimal cost = [Mother, Kitchen] = 2 + [Father, Kitchen] = 3 + [Daughter, Kitchen] = 5 + [Son, Basement] = 2 = 12 aka lowest bound.



Optimal jobs would be Mother in the Kitchen, Father in the Den, Daughter in the Bathroom, and Son in the Basement for a value of 23.

c. (20 pts) How many partial solutions did you generate? Was it worth using this method? Explain your answer. Hint: It is essential to demonstrate how the algorithm works. First, don’t waste your time by generating the entire space tree. Second, please refrain from picking (or steering) your solution to the lowest task assignment sequence (if you can visually identify it). While this question is reduced in terms of possible parameters to make it manageable to solve on paper, the problem -and the associated technique- are regularly encountered in real logistics problems with hundreds of persons and hundreds of tasks. Therefore, it would be impossible to visually identify the lowest task assignment sequence.

Not counting the optimal illegal state, there were 11 partial solutions generated. This was also due to two of the partial solutions being of the same value. If there were no ties in value, each level would have a number of n – 1 partial solutions from its parent level.

This algorithm could be used with any x \* x or x \* y matrix representation of “jobs” and “tasks.” Using the illegal optimal state, the algorithm starts at its perfect state. The beauty of the algorithm is its ability to always obtain an illegal optimal state and generate partial answers/states to “build” an optimal legal state. The goal is to use the lowest bound on each level of node comparison. Each level will a player a static position and create permutations of possible optimal outcomes given the introduction of the static legal position of the player placed. Each level adds n + 1 legal positioning of 1 player. Once The level calculates the lowest value of the nodes, and branches from that node. The next level repeats the process by using all the legal placements established by previous levels. This process continues until the legal placement of players is as close as it could possibly be to the illegal optimal positions, thus, giving the optimal legal positioning of jobs/players.

Whereas an exhaustive search would have taken us down each permutation of a possible answer, which could have given us 4! or 24 states at worst case. Branch and Bound tries to exclude the branches that don’t have a chance of producing a correct answer. Our answer gave us 4+3+4+1 = 12 node creations. This was half the number of states that had to be created in the worst-case scenario for this problem. For this problem, branch and bound was worth using. However, there could be a case which would have several tie scenarios throughout the levels. This would make some of the branches extremely large. Branch and Bound does find the optimal solution, but it could become inefficient from a “number of calculations” standpoint. Branch and Bound seems to have limitations on the size of the problem they are applied and the cost of time needed for the optimal solution.

2. (10 pts) Given an O(n2​) algorithm, a data set with ​n = 16 takes 24 seconds. What is the largest size data set that can be executed by this algorithm in 60 seconds? Explain your answer.

Answer: An O(n​2​) algorithm would double itself every iteration, thus the example would have been built by 2 -> 4 -> 8 -> 16. The time taken for this build would also have gotten to 24 seconds by the same number of iterations 3 -> 6 -> 12 -> 24. The number of iterations needed for the time to continue the same path to reach 60 is 3 -> 6 -> 12 -> 24 -> 48 -> 60(.25 = 12(number to 60)/96-48(number is 48, but wanted to show that it was taking into account next iteration) = 5 iterations + .25 of an iteration. Now to apply this number to the power of 2: 2 -> 4 -> 8 -> 16 -> 32 -> 40 = ((5 iterations = 32) + ((32 \* .25) = 8)). If a dataset with n=16 is done in 24 seconds, the answer is: 40 is the largest dataset you can execute in 60 seconds.

3. Select ONE of the topics below.

a. (15 pts) Provide a one-page synopsis of the topic, explaining the problem(s) it solves, the method, asymptotic complexity, advantages/disadvantages or limitations.

See HornerAlgorithm.xdoc

b. (25 pts) Using the programming language of your choice implement the algorithm and provide 4 sample input (if necessary).

Subject: Horner’s Rule algorithm (polynomial evaluation)

See attached Java program.

​Your implementation should read a list of polynomial coefficients (the order should be in the increasing powers of x) along with a value of x to compute its value at, and return the value of the polynomial at that value.