Dynamic Programming, Greedy - Practice

Last updated: 4/17/2022

Book (CLRS) problems:

1. Dynamic programming: 15.4-1, 15.4-2, 15.4-3 and 15.4-5 (here justify the complexity of all steps you take). (page 396)
2. Greedy: 16.1-3 (page 422)

Other sources: Problems/issues discussed in class, Homework problems, Canvas quizzes.

Types of problems:

1) Given solution table partially filled out, finish filling it out.
2) Given the gain/cost solution, recover the **solution choices** that gave this optimal value.
3) Time and space complexity for all covered algorithms.

Edit Distance (ED), Longest Common Subsequence (LCS), Longest Increasing Subsequence (LIS)

P1-MIX. Short answer:

a) True/False for any DP problem, we can only recover the choices that give the optimal solution AFTER we computed the actual optimal value.

b) (5 pts) Give the function for computing the longest common subsequence (LCS): D[i][j] = …. You can use max/min functions on however many arguments. You can write the function as a math function or as code. Assume i is the index for string $X = x_1 x_2 x_3 \ldots x_M$ and j is the index for string $Y = y_1 y_2 y_3 \ldots y_N$.

c) (6 pts) Give the math function (or code piece) for computing the cell Dist[i][j] for the Edit Distance between two strings X and Y. Assume that in the table, X will be vertical and Y will be horizontal. Assume the index of the first letter in a string is 0 (not 1). (Same question applies to **all** DP problems covered in class.)

d) (5 pts) This is meant to be an edit distance table, but something is wrong. What is the problem?

0	**1**	**2**	**3**
1	**1**	**3**	**4**
2	**2**	**2**	**4**
3	**3**	**3**	**3**

e) (4 pts) In the table below, the highlighted value should be 2 instead of 3, because it calculates the minimum of 2,3,4 when the letter is the same: o. True/False. EXPLAIN.

		s	o	m	e	t	h	i	n	g
	0	1	2	3	4	5	6	7	8	9
s	1	0	1	2	3	4	5	6	7	8
o	2	1	0	1	2	3	4	5	6	7
m	3	2	1	0	1	2	3	4	5	6
e	4	3	2	1	0	1	2	3	4	5
o	5	4	**3**	2	1	1	2	3	4	5
n	6	5	4	3	2	2	2	3	3	4

P2-LIS. (8pts) In class we discussed how we can solve the Longest Increasing Subsequence (LIS) problem by reducing it to another problem.

a) (3pts) What other type of problem did we reduce it to?

b) (5pts) Assume the specific instance you need to solve is 7, 1, 3, 6, 4, 1, 2, 3, 0, 6, and you allow repetitions in the increasing sequence (it is NOT strictly increasing: 3,3,6 is allowed). Give the reduction: produce a problem **instance** of the other problem type. You do NOT need to solve, it, just give the new instance problem corresponding to the given LIS problem.

P3-ED. (6pts) Fill out the last two rows in the table below:

		N	O	N	S	T	O	P
	0	1	2	3	4	5	6	7
R	1	1	2	3	4	5	6	7
O	2	2	1	2	3	4	5	6
U	3	3	2	2	3	4	5	6
N								
D								

P4-ED. (6pts) Fill-out the last two rows in the edit distance table below. (Here STEM is the complete second word. ME is the end of the first word. For example the first word could be NAME, TESTME).

		S	T	E	M
…	…	…	…	…	…
…	3	3	2	2	3
M					
E					

0/1 Knapsack, normal and fractional. Dynamic Programing & Greedy

NOTE: *Unless otherwise specified, any Knapsack problem is assumed to be normal, NOT be fractional: items can be picked up as the whole item or not at all.*

K1. Short answer:

a) (3pts) The time complexity for the ITERATIVE solution for the 0/1 Knapsack of max capacity N and d items is: is Θ(_______)

b) (3pts) What information did we use to recover the items from the optimal solution for the 0/1 Knapsack:

A) a star B) an arrow C) the Picked array D) None of these.

c) (5 pts) Give the function for computing the cell **sol[i][j]** for the 0/1 Knapsack. Assume that i is the index of the item. You can use value[i] and weight[i] to refer to the value and the weight of item i.

d) (5 pts) Is there any Knapsack version for which Greedy (with the ratio value to weight) gives an optimal solution for?

K5. (14 pts) Use a Greedy method based on the **largest value** to solve a **0/1** Knapsack problem.

a) (6pts) Give the pseudocode for the algorithm (what it computes, what actions it takes).

b) (8pts) Does it give an optimal solution or not? If Yes, justify, if No, prove it with a counter example.

Job Scheduling (or Weighted Interval Scheduling): DP & Greedy

JS1. Short answer

a) Assume that the data has been preprocessed to the point where you have all the pi values for a Job Scheduling problem with N jobs. What is the time complexity to solve this problem? Circle the tightest asymptotic bound (O, Θ or Ω) that applies and fill in the function as well. O / Θ / Ω (……….)

b)

JS2. (12 pts) a) (10 pts) Use Dynamic Programming to solve the Weighted Job Scheduling below for jobs 1-6 with job values given by v_i. Here p_i gives the last job that does not overlap with job j. When filling in the answer for m(i) show your work as done in class.

b) (2 pts) Recover the solution (fill in in the rightmost column).

i	v_i	p_i	m(i)	m(i) used i: Y/N	In opt Solution: Y/N
0					
1	4	0			
2	3	0			
3	4	1			
4	5	2			
5	3	2			
6	2	4			

JS3. (11pts) For a Job-Scheduling problem where each job has the same value (e.g. $10), your goal is to choose the maximum number of non-overlapping jobs. You take the following approach:

- Sort the jobs in increasing order of START TIME
- Repeat until no job left:
  - Pick the job,J, that starts first. Assume no two jobs have the same start time.
  - From the remaining jobs, remove all the jobs that overlap with job J.

a) (3 pts) Is this Greedy, Dynamic Programming or Brute Force Search?

b)(8 pts) Does it give an optimal solution or not? If Yes, justify, if No, prove it with a counter example.

JS4. () When we solved the Job Scheduling problem, the problem was already 'preprocessed' to lend itself to a Dynamic Programing (DP) solution.

a) (5 pts) Fill in the 2 sentences below to say what the 'preprocessing" does.

Step 1:____

p_i is ____

b) (6 pts) Given the jobs in the table to the right, do the preprocessing so that the problem can be solved with DP. Solve up to and including the p_i.

Start time	End time	Value
5	7	5
1	2	4
3	6	4
4	7	3
1	4	3
6	8	2

Draw the picture and fill in the table below:

Job	Start	End	Value	p_i
0				
1				
2				
3				
4				
5				
6				