Arup's Exam Review

Wednesday, April 18, 2018 11:37 PM

Travis-Questions-...

COP 3503: Questions from Old Final Exams (for 4/19/2018 Final Exam Review)

1) Dynamic Programming (Matrix Chain Multiplication)

Using the dynamic programming algorithm shown in class, determine the minimum number of multiplications to calculate the matrix product ABCD, where the dimensions of A, B, C and D are given below:

A: 2 x 5, B: 5 x 4, C: 4 x 1, D: 1 x 5

	A	B	C	D
A	0	40	30	40
B	X	0	20	45
C	X	X	0	20
D	X	X	X	0

(ABC) → (A)(BC) → 0 + 20 + 2x5x1 = 30
(AB)(C) → 40 + 0 + ? = 48

(BCD) → (B)(CD) → 0 + 20 + 5x5x4 = 120
(BC)(D) → 20 + 0 + 5x5x1 = 45

(ABCD) → (A)(BCD) → 0 + 45 + 50 = 95
(AB)(CD) → 40 + 20 + 40 = 100
(ABC)(D) → 30 + 0 + 10 = 40

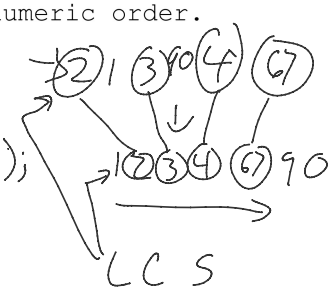
2) Dynamic Programming – Longest Increasing Sequence

Assume that you've already written a method LCS, that calculates the length of the longest common subsequence between two sequences of integers. (Its prototype is given below.) Write a method that takes in 1 sequence of integers and calculates its longest increasing sequence. For ease of implementation, assume that the input only contains distinct integers. Note: You may use both the Arrays.copyOf and Arrays.sort methods. These are listed below. (Note: The code is pretty short.)

```
// Returns an array storing the first newLength elements of original.
int[] copyOf(int[] original, int newLength);

// Sorts the specified array into ascending numeric order.
void sort(int[] a);


public static int lis(int[] seq) {
    int[] ref = copyOf(seq, seq.length);
    sort(seq);
    return lcs(seq, ref);
}
```



```
public static int lcs(int[] x, int[] y) {

    int[][] table = new int[x.length+1][y.length+1];

    for (int i = 1; i<=x.length; i++) {
        for (int j = 1; j<=y.length; j++) {
            if (x[i-1] == y[j-1])
                table[i][j] = 1+table[i-1][j-1];
            else
                table[i][j] = Math.max(table[i][j-1], table[i-1][j]);
        }
    }
    return table[x.length][y.length];
}
```

3) Dynamic Programming – Zero/One Knapsack Problem

Find the maximum valued knapsack of size 12 or less choosing from the following items (only 1 copy of each item is available). To get credit, please use the algorithm shown in class. Here are the items from which you are choosing:

Item	Weight	Value
Apple	3	4
Banana	4 X	6 ←
Cantaloupe	7	13
Durian	5 X	9 ∈
Emblic	2 X	5 ∈
Fig	1 X	3 ∈

12 - 4 = 8 NOT 9

Show the algorithm by filling in the following table:

Item 0	1 0	2 0	3 0	4 0	5 0	6 0	7 0	8 0	9 0	10 0	11 0	12 0	
Apple 0	0	0	4	4	4	4	4	4	4	4	4	4	
Banana 0	0	0	4	6	6	6	10	10	10	10	10	10	X
Cantaloupe 0	0	0	4	6	6	6	13	13	13	17	19	19	
Durian 6	0	0	4	6	9	9	13	13	15	17	19	22	X
Emblic 0	0	5	5	6	9	11	14	14	18	18	20	22	X
Fig 0	3	5	8	8	9	12	14	17	18	21	21	23	X

The correct answer to the query should be in the bottom right square of the table.

4) Graphs – Topological Sort

Alice has to complete items 1 through 10. The following ordered pairs show the dependency between the items she must complete. Namely, for each ordered pair (a, b) shown below, she must complete item a before item b.

(2, 7), (8, 3), (9, 2), (4, 5), (4, 1), (4, 7) and (6, 10).

Show the ordering of the items produced by the algorithm shown in class that builds the list from the front and always adds "safe" nodes iteratively. When choosing between multiple possible "safe" nodes, always add the lowest numbered one first.

4, 1, 5, 6, 8, 3, 9, 2, 7, 10

5) Algorithm Design – Dynamic Programming

Describe a dynamic programming algorithm (in words) to solve the following problem:

Soccer players are ordered 0, 1, 2, …, n – 1, where n is a positive integers less than or equal to 100. Each player i ($0 \le i \le n-2$) can pass to any player j such that $j > i$. The probability the pass will succeed is prob[i][j]. (Assume that this information is given in the input.) As with all probabilities, note that $0 \le \text{prob}[i][j] \le 1$, for all $0 \le i < j \le n - 1$.

We are allowed to choose any sequences of passes in order to move the ball from player 0 to player n – 1. (This sequence will necessarily be some subsequence of 0, 1, 2, …, n – 1, since all the players can only pass the ball in "one direction.") If all players act optimally, design an algorithm that calculates the probability that player n – 1 successfully receives the ball.

I used floyd's and turned addition into multiplication and we look for maximal values not minimums.

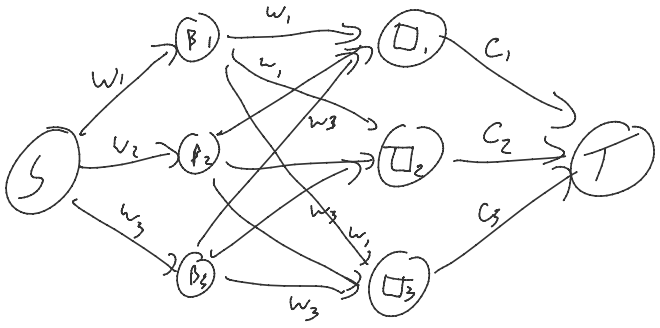
```
double[] p = new double[n];
for(i=n-1; i>=0; i--)
    p[i]= (i==n-1)? 1:0;
    for(int j=i+1; j<n; j++){
        if(p[i] < p[j]*prob[i][j]){
            p[i] = p[j]*prob[i][j]
        }
    }
}return p[0];
```

6) Consider the following problem:

You have a set of n books with weights w_1, w_2, … , w_n (in pounds). You must pack them into k boxes. The capacity of each of these boxes (in pounds) is c_1, c_2, …, c_k. Our goal is to determine whether or not there exists a way to pack all of the n books in the k boxes without exceeding the capacity of any of the boxes. Assume that all of the book weights are positive integers less than 50 and all of the box capacities are less than 200. Furthermore, assume that both n and k are less than 100.

A proposed solution to this problem is to set up a Network Flow graph. Each book would have a vertex and each box would have one also. A source vertex, s, would be set up and a sink vertex t would also be added to the graph. An edge from s to each book would be set up with the capacity of that book. An edge from each book to each box would be set up with the capacity of the corresponding book. Finally, an edge from each box would be set up to the sink with the capacity equal to the maximum weight the box could handle. Determine the maximum flow of this network. If it is equal to the sum of the weights of the books, the books can be packed. Otherwise they can not be.

What is the flaw in this solution? Create a simple example that shows that this solution does NOT work in all cases.



$B_1 = 49$ → $\square_1 = 50$

$B_2 = 49$ → $\square_2 = 50$

$B_3 = 2$

The solution suggests we tear a book in half!

7) In the jumble problem, you are given a set of letters, in random order, and you have to determine whether or not there exists a way to reorder those letters to form a valid word. One way to solve this problem is to generate all permutations of the input letters and check each permutation against a dictionary of words. Given that you have access to a function which tells you whether or not any words start with a set of letters, (for example startswith("jx") would return false while startswith("po") would return true), explain how you can solve the jumble problem using backtracking. Furthermore, explain why this solution is more efficient than the permutation solution.

```
boolean back(String prefix, string remainings)
    if(! starts with (prefix))
        return false;
    loop through letters in remainings strings {
        new prefix = prefix + cur letter;
        new rem = remains - curletter;
        if(back (new prefix, new rem))
            return true;
    return false;
```

Recursions $\overset{3}{}$ coffee → cf

by breaking out early with "cf"

we eliminate a large piece of the search tree (4!)

8) The following list is a list of all the TV shows you want to watch and when they are on. Determine the most number of shows you can watch, assuming that you can only watch one show at a time and that when you watch a show, you watch it in its entirety. In doing so, utilize the greedy algorithm shown in class and fill out your TV watching schedule. (Note: You may not use all the slots given.)

Show	Time	Show	Time	Show	Time
Idol	6:00-9:15pm	Survivor	8:45-9:45pm	24	7:30-10:30pm
CSI	7:30-8:30pm	Office	8:00-9:00pm	Law&Order	9:50-10:50pm
Cops	6:30-7:30pm	30 Rock	7:00-9:00pm	SportsCenter	9:15-11:00pm

6:30 - 7:30 Cops

7:30 - 8:30 CSI

~~7:00 - 9:00~~

~~8:00 - 9:00~~

~~6:00 - 9:15~~

8:45 - 9:45 Survivor

~~7:30 - 10:30~~

9:50 - 10:50 Law & Order

~~9:15 - 11:00~~