**CS375 Assignment 3**

**Due on 10/15/2023 (by 11:59pm)**

**Objective:**

1. Design and analyze an algorithm using dynamic programming strategy
2. Enhance the concept of heap for sorting.

There are two parts in this assignment: (A) Theory part and (B) programming part

**[Part A] Theory [80%]:**

1. [10%] Given the following array:

a. [2%] Show the essential complete binary tree for the following array:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 11 | 7 | 15 | 9 | 3 | 12 | 10 | 22 | 8 |

**Answer:**

A drawing of a triangle with numbers

Description automatically generated

1. [4%] Convert the essential complete binary tree into a min heap using the fast-make-heap method. Draw the essential complete binary tree after each *siftDown*.

**Answer:**

A black and white image of a triangle

Description automatically generated

1. [4%] Convert the essential complete binary tree (from (a)) into a max heap. Using the max-heap sorting algorithm to sort the above sequence in the non-increasing order. Show the heap after the value “11” has been moved to the sorted sequence.

**Answer:**

A diagram of a diagram

Description automatically generated with medium confidence

A drawing of a triangle with numbers

Description automatically generated<- After “11” has been moved to the sorted sequence.

1. [10%] Find the longest common subsequence for the strings x= CACMYCCA, and y = YMCMAMYYCMA. Show the table with the lengths and arrows. (Optional extra points [6%]: write code and run your program to print out the LCS; explain time complexity and space complexity; print out the running time of the algorithm).

**Answer:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Y | M | C | M | A | M | Y | Y | C | M | A |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| C | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 |
| M | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 |
| Y | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| C | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 4 | 5 | 5 | 5 |
| C | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 4 | 5 | 5 | 5 |
| A | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 |

The longest common subsequence is CAMYCA.

1. (8%) Solve the all-pairs shortest-path problem for the following diagraph. Using the Floyd’s algorithm to calculate the values for matrices *D*0, *D*1, …, *D*5 and the corresponding P tables. So the shortest distance between all pairs of nodes can be found. Explain the time complexity and space complexity.

1

2

3

4

5

2

2

1

2

3

4

8

3

6

3

**Answer:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | INF | 1 | 8 |
| 2 | 6 | 0 | 3 | 2 | INF |
| 3 | INF | INF | 0 | 4 | INF |
| 4 | INF | INF | 2 | 0 | 3 |
| 5 | 3 | INF | INF | INF | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D1 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | INF | 1 | 8 |
| 2 | 6 | 0 | 3 | 2 | 14 |
| 3 | INF | INF | 0 | 4 | INF |
| 4 | INF | INF | 2 | 0 | 3 |
| 5 | 3 | 5 | INF | 11 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D2 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | 5 | 1 | 8 |
| 2 | 6 | 0 | 3 | 2 | 14 |
| 3 | INF | INF | 0 | 4 | INF |
| 4 | INF | INF | 2 | 0 | 3 |
| 5 | 3 | 5 | 8 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D3 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | 5 | 1 | 8 |
| 2 | 6 | 0 | 3 | 2 | 14 |
| 3 | INF | INF | 0 | 4 | INF |
| 4 | INF | INF | 2 | 0 | 3 |
| 5 | 3 | 5 | 8 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D4 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | 3 | 1 | 4 |
| 2 | 6 | 0 | 3 | 2 | 5 |
| 3 | INF | INF | 0 | 4 | 7 |
| 4 | INF | INF | 2 | 0 | 3 |
| 5 | 3 | 5 | 8 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D5 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 2 | 3 | 1 | 4 |
| 2 | 6 | 0 | 3 | 2 | 5 |
| 3 | 10 | 12 | 0 | 4 | 7 |
| 4 | 6 | 8 | 2 | 0 | 3 |
| 5 | 3 | 5 | 8 | 7 | 0 |

The table D5 represents the shortest distance between all pairs of nodes that can be found. The time complexity is O(V^3) = O(5^3), where V = 5 since there are 5 vertices, as there are three nested loops in Floyd’s algorithm. The space complexity is O(V^2) = O(5^2), where V = 5 since there are 5 vertices, for storing the 2D D and P matrices of size V\*V.

1. [10%] If we want to design an algorithm to merge two heaps of sizes *n*1 and *n*2. Assume that heap 1 is stored in bt1[1,…,n1] and heap 2 is stored in bt2[1,…,n2]. Assume that n2 >= n1. The two heaps do not overlap.

Is the following solution correct?

If yes, give the upper bound of the time complexity.

If not, give an example where it is wrong, describe a correct solution, and give the upper bound of the time complexity.

Copy the larger heap at the end of the smaller one. After the copy bt1 contains all n1 + n2 values. Now apply siftDown to all tree nodes from n1 down to 1.

**for** (i=1; i <=n2, i++)

bt1[n1+i] = bt2[i]

last = n1 + n2

**for** (i=n1; i > 0, i=i-1)

siftDown(bt1, i)

**Answer:** The solution is incorrect, because siftDown is only applied to all the nodes from n1 down to 1, which does not ensure that the heap property is maintained for all nodes in the merged heap. A correct solution would be to merge the two heaps into a single array and then build a new heap from this array. This can be done by copying the larger heap at the end of the smaller heap and applying a heapify operation to the entire array. The upper bound time complexity is O(n), where n is the total number of elements in the two heaps.

1. [6%] Compute the binomial value of using dynamic programming B table.

**Answer:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 1 |  |  |  |  |  |  |
| 1 | 1 | 1 |  |  |  |  |  |
| 2 | 1 | 2 | 1 |  |  |  |  |
| 3 | 1 | 3 | 3 | 1 |  |  |  |
| 4 | 1 | 4 | 6 | 4 | 1 |  |  |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 |  |
| 6 | 1 | 6 | 15 | 20 | 15 | 6 | 1 |
| 7 | 1 | 7 | 21 | 35 | 35 | 21 | 7 |
| 8 | 1 | 8 | 28 | 56 | 70 | 56 | 28 |

The binomial value is 28.

[4%] Can you make a small B table (less than 6 columns) to obtain the solution.

**Answer:** No, since a B table has dimensions (n+1) x (k+1), the minimum number of columns required is 7, where k = 6.

1. [12%]

Design a dynamic programming algorithm to solve the sum of subsets problem:

There are *n* positive integers, and a positive integer sum *S*. Is there a subset *A* of the *n* integers such that the sum of the integers in the subset is S, ()? If there is such a subset the output of the program is *true* otherwise it is *false*.

Example: In this case the output is *true* since *p*2+ *p*3 = 16.

Example: In this case there is no solution, and the output is *false*.

The problem can be solved with dynamic programming.

A **Boolean** matrix *B* with rows 0 to *n* and columns 0 to *S* is generated. For the examples above the matrix has rows 0 to 3, and columns 0 to 16.

*B*[*i*, *s*] is **true** if a subset of the first *i* integers sums to *s*, otherwise it is **false**.

The solution to the original problem is **true** if *B*[*n*, *S*]= **true**, otherwise it is **false.**

1. [5%] Write the recurrence relation for the solution.

**Answer:** The recurrence relation for the solution is B[i][s] = B[i – 1][s] OR B[i – 1][s - .

1. [4%] Write pseudo code to compute B. Explain the time complexity and space complexity.

**Answer:**

bool subsetSum(int A[], int n, int S) {

bool B[n+1][S+1];

// Initialize first column as true

for(int i = 0; i <= n; i++)

B[i][0] = true;

// Initialize top row, except for B[0][0], as false.

for(int i = 1; i <= S; i++)

B[0][i] = false;

// Fill rest of matrix using recurrence relation

for(int i = 1; i <= n; i++) {

for(int j = 1; j <= S; j++) {

if(A[i-1] <= j)

B[i][j] = B[i-1][j] || B[i-1][j-A[i-1]];

else

B[i][j] = B[i-1][j];

}

}

return B[n][S];

}

The time complexity is O(n\*S), where n is the number of integers and S is the target sum, because the algorithm fills up a 2D array of size n\*S. The space complexity is also O(n\*S) because the algorithm uses a 2d array of size n\*S.

1. [3%] Apply dynamic programming to the following problem: Show your matrix B. Each element of B has the value *true* or *false*. Also, use arrows to show the matrix elements that were ordered, or used.

**Answer:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | T | F | F | F | F | F | F |
| 1 | T | T | F | F | F | F | F |
| 2 | T | T | T | T | F | F | F |
| 3 | T | T | T | T | T | T | F |

7. (10%)

(a) (4%) Show that the number of distinct binary search tree b(n) that can be constructed for a set of n orderable keys satisfies the recurrence relation:

*for n>0, b(0)=1*

**Answer:** The recurrence relation is derived from the fact that for any given set of n keys, we can choose any key as the root of the binary search tree. The remaining keys are then divided into two subsets: one for keys that are less than the root and one for keys that are greater than the root. The number of binary search trees is the product of the number of binary search trees for the two subsets. So, the recurrence relation is satisfied.

(b) (6%) Constructing the optimal binary search tree, given

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| key | A | B | C | D |
| probability | 0.1 | 0.2 | 0.4 | 0.3 |

Show the dynamic programming tables of each step.

**Answer:**

**1. Initialization**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A |  |  |  |  |
| B |  |  |  |  |
| C |  |  |  |  |
| D |  |  |  |  |

**2. Diagonals**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0.1 |  |  |  |
| B |  | 0.2 |  |  |
| C |  |  | 0.4 |  |
| D |  |  |  | 0.3 |

**3. Subtrees**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0.1 | 0.3 |  |  |
| B |  | 0.2 | 0.6 |  |
| C |  |  | 0.4 | 0.8 |
| D |  |  |  | 0.3 |

**4. Optimal Tree**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0.1 | 0.3 | 0.7 |  |
| B |  | 0.2 | 0.6 | 1 |
| C |  |  | 0.4 | 0.8 |
| D |  |  |  | 0.3 |

Optimal binary search tree:

A black and white drawing of a triangle

Description automatically generated

8. (10%) Design an algorithm to solve a “ball-picking” problem.

Problem statement: there are a number of balls (n) placing in a circle, with all white except one red. Starting from any one of the balls, a robot travels along the circle in a clockwise direction, and pick a ball in every three balls (k=3 which means for every three balls, skip the first two balls, and picks the third one). The procedure is repeated with the remaining balls, starting with the next ball, the robot travels on the circle repeatedly, and pick balls in the same way until only one ball remains. In order to keep the red-ball remained at final, which ball has to be the first ball for robot to start with?

Example:

Design your algorithm by giving an iteration/recurrence relation for the solution. Explain the time complexity and space complexity.

Show the starting ball of the right figure (n=11, k=3) in order to have the red-ball #4 as the last ball to be remained.

**Answer:** The recurrence relation is s(n, k) = (s(n – 1, k) + k) % n, where n = total number of balls, k = number of balls the robot skips before picking one, and r = index of the red ball. To determine the starting ball index of n balls, we consider the index for n – 1 balls and add k to it, which in this case is 3. We take the modulo n to ensure the index wraps around in a circle.

Algorithm: First, initialize n, k and k. Then, use the recurrence relation to find s(n, k), which is the starting ball index for the desired result. The value of s(n, k) will be the ball that the robot should first pick up in order to keep the red-ball as the final one.

The time complexity is O(n), because the function is called recursively n times and the algorithm performs a single calculation for each n. The space complexity is O(1), as the algorithm uses a constant amount of memory to store input variables and calculated results.

Let n = 11, k = 3.

s(11, 3) = (s(10, 3) + 3) % 11

s(10, 3) = (s(9, 3) + 3) % 10

s(9, 3) = (s(8, 3) + 3) % 9

s(8, 3) = (s(7, 3) + 3) % 8

s(7, 3) = (s(6, 3) + 3) % 7

s(6, 3) = (s(5, 3) + 3) % 6

s(5, 3) = (s(4, 3) + 3) % 5

s(4, 3) = (s(3, 3) + 3) % 4

s(3, 3) = (s(2, 3) + 3) % 3

s(2, 3) = (s(1, 3) + 3) % 2

s(1, 3) = (s(0, 3) + 3) % 1

s(0, 3) = 0

So if ball 0 gets picked up first, then ball 4 will be the last one to remain. Hence, the robot should start at ball 8 so that it can pick up ball 0 first.

Extra point (10%): Given N balls, starting ball (P), traveling along the circle in the clockwise direction, (K-1) balls to be skipped, pick one ball for every K balls, repeat until only one ball remained. Write a program to implement your algorithm, and print out which ball is remained in the case of N=100 (balls) and K=7. (Print out your running time)

9. (Optional: Extra points 10%)

Suppose there is an apartment available for rent for a year. There are 6 requesters who want to rent the apartment for 6 different periods of time with 6 different offers:

Requester 1: starting January 1st for 3 months, and would like to pay $2k;

Requester 2: starting February 1st for 5 months, and would like to pay $4k;

Requester 3: starting May 1st for 5 months, and would like to pay $4k;

Requester 4: starting April 1st for 7 months, and would like to pay $7k;

Requester 5: starting October 1st for 2 months, and would like to pay $2k;

Requester 6: starting November 1st for 2 months, and would like to pay $1k;

Since the apartment cannot be rented for more than one requester at a same period of time, design a dynamic programming algorithm to select the subset of requesters in order to maximize the sum of rentals (values).

1. Write the recurrence relation for the solution;
2. Show the time complexity of your algorithm; Print out your running time.
3. Using the above instance, show the dynamic programming table of your solution from the 6 requesters;

(Hint: The problem can be generalized by *j* requesters (j=1, 2, …, n), and each requester would like to pay *v(j)* dollars for *t(j)* period of time. )

**[Part B] Programming [20%]:**

1. [20%] Given a following map, find the shortest distance between any two cities and plot their shortest paths as well. (Note: The number shown on a connected line of a pair of cities is the distance in miles between the two cities; the number shown in each node is the label (or index) of the city.)

*Note: Print out the final D table and P table; show one example which is the path from New York City to Toronto. What’s the running time of the algorithm?*

Toronto

Buffalo

Rochester

Kingston

Syracuse

Binghamton

Scranton

Allentown

Albany

Montreal

New York City

**140**

**130**

**160**

**180**

**100**

**60**

**70**

**70**

**65**

**60**

**65**

**70**

**100**

**180**

**100**



1

2

3

4

5

6

7

8

9

10

11

1. [Extra Points 10%] For this optional part, you need to design a three-string LCS algorithm for finding the longest common subsequence (LCS) of three strings. Test your algorithm using following three strings:

String 1: 6541254939322816220209974565477289648317

String 2: 3142522751761601737419090933147067701840

String 3: 28070305612903542595135701601 62463275171

The length of the LCS = 12, and an LCS =525410046771

**Print out the result by your program: length of LCS and the actual LCS. What’s the running time of the algorithm?**

Hint: To achieve this goal and to be able to run the large test data you need to generalize the recurrence equation given for the two strings LCS problem to work with three strings.

**3. Write-up:**

For each of above implementation:

1. Show your code (with comments and algorithm explanation)
2. Print out your results