# CS463 Spring 2017 Project#3

STUDENT NAME: Matthew Krieger FILE NAME P3cs463Krieger

DATE: April 27, 2017

|  |
| --- |
| 1. [ 1 point] Draw a binary tree with 9 nodes in such a way that the inorder traversal of the tree is 8,0,4,1,5,9,7,2,3 and the postorder traversal of the tree is 0,4,8,1,7,9,3,2,5. Explain how you created the resulting tree.   ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/0236F0C8-D044-40BC-8562-1287501CC1ED/CaptureTreeSnap.JPG |
| 2)[1 point] Figure 6.18 shows state-space graph for the peasant, wolf, goat, and cabbage puzzle. Draw state-space graph for four persons p1, p2, p5 and p10 that required 1, 2, 5, 10 minutes, respectively, to get to the other side of the bridge from project 1 in order to illustrate problem reduction to graph problems.  ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/6AFA6313-1C81-46E1-B4B7-296556E5CCDD/statespacecappy.JPG |
| 3) [ 2 points] Implement and test recursive algorithm for ternary search. Method returns the index where key was found in array arr with n occupied positions, or returns -1 if search is not successful. Perform worst case time analysis. If n=1 compare the search key with a single element of the array. Otherwise, search recursively by comparing key with arr[n/3], and if key is larger compare it with arr[2\*n/3] to determine in which third of the array to continue the search. Assume that n is a power of 3 ( n=3k) and that arr is sorted. Let first and last be indices of subarray so that c.  public int ternarySearch(int[] arr, int first, int last, int key)  Compare this algorithm's efficiency with that of binary search.   |  | | --- | | // Recursive Ternary Search. Returns the location of the key (x) in  // an array if it is present, -1 if not found  int ternarySearch(int arr[], int l, int r, int x){  if (r == 1){  if (arr[0] == x) return 0;  }  if (r >= l){  int mid1 = l + (r - l)/3;  int mid2 = mid1 + (r - l)/3;  // If key is at the mid1  if (arr[mid1] == x) return mid1;  // If key is at the mid2  if (arr[mid2] == x) return mid2;  // If key is in left one-third  if (arr[mid1] > x) return ternarySearch(arr, l, mid1-1, x);  // If key is in right one-third  if (arr[mid2] < x) return ternarySearch(arr, mid2+1, r, x);  // If key is in middle one-third  return ternarySearch(arr, mid1+1, mid2-1, x);  }  // Key not found  return -1;  } |  |  | | --- | | // Recursive Binary Search. Returns the location of the key (x) in  // an array if it is present, -1 if not found  int binarySearch(int arr[], int l, int r, int x){  if (r >= l){  int mid = l + (r - l)/2;  if (arr[mid] == x) return mid;  if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);  return binarySearch(arr, mid+1, r, x);  }  return -1;  } | |  |   At first it seems the ternary search does less number of comparisons as it makes Log3n recursive calls, but binary search makes Log2n recursive calls.  Worst Case Binary Search: T(n) = T(n/2) +2, T(1) = 1  Worst Case Ternary Search: T(n) = T(n/3) +4, T(1) = 1  In binary search, there are 2Log2n + 1 comparisons in worst case. In ternary search, there are 4Log3n + 1 comparisons in worst case.  Binary Search Time Complexity: 2clog2n + O(1)  Ternary Search Time Complexity: 4clog3n +O(1)  comparison of Ternary (2Log3n ) and Binary Searches(Log2n) results from comparison of expressions. The value of 2Log3n can be written as (2 / Log23) \* Log2n . As a result of the value of (2 / Log23) is more than one 🡺 Ternary Search does more comparisons than Binary Search in worst case. |
| 4) [ 1 point] For n=5 draw all binary trees that satisfy the balance requirements of AVL trees.   |  | | --- | | ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/C662C69F-029E-4036-A29A-14A7A5376C2B/bin5.JPG | |
| 5) [ 1 point] Solve the following system by applying Gaussian elimination. It must be exact application of the algorithm described in example 1 in the textbook page 210.  x + y + z = 6  3x – y + z = 2  x + 2y –2z = 3  1 1 1 6  3 -1 1 2 row2 – (6/2)row1  1 2 -2 3  1 1 1 6  0 -4 -2 -16  1 2 -2 3 row3 – (2/2)row1  1 1 1 6  0 -4 -2 -16  0 1 -3 -3 row3 – (-1/4)row2  1 1 1 6  0 -4 -2 -16  0 0 -7/2 -7  z = (-7 / (-7/2)) = 2  y = (-16 – (-2)z)/(-4)) = 3  x = ((6 – z – y ) = 1 |
| 6) [ 1 point] Solve the following system by Kramer rule.  x + y + z = 6  3x – y + z = 2  x +2 y –2z = 3  1 1 1 6  3 -1 1 2  1 2 -2 3  ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/EDA4D420-BCB9-45DA-831E-8468B9DDB234/cramer1.JPG  ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/A1330D52-7BCF-4175-83F6-396DC163FDE2/cramer2.JPG  ../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/1567BEE6-A864-4EEC-9DE4-D06C26C4AB9C/cramer3.JPG |
| 7) [2 points] Construct an AVL tree for the list 7, 4, 3, 5, 6, 2, 8, 9 by inserting the elements consecutively, starting with the empty tree. Show tree after each insert and/or rotation. Specify balancing factor for each node in every tree.../Library/Containers/com.apple.mail/Data/Library/Mail%20Downloads/272D665E-C19E-4DFD-BCD0-1B186D52F57F/Algocap.JPG |
| 8) [2 points] Implement and test decrease-by-one minimal change algorithm to generate all permutations of numbers {1,2,…,n}. User inputs positive integer n and program generates permutations of {1,2,…,n}. Provide code, and resulting permutations for the following two tests ( inputs n=3 and n=5.) |