Süre: 75 Dakika

Manisa Celal Bayar Üniversitesi Yazılım Mühendisliği Bölümü

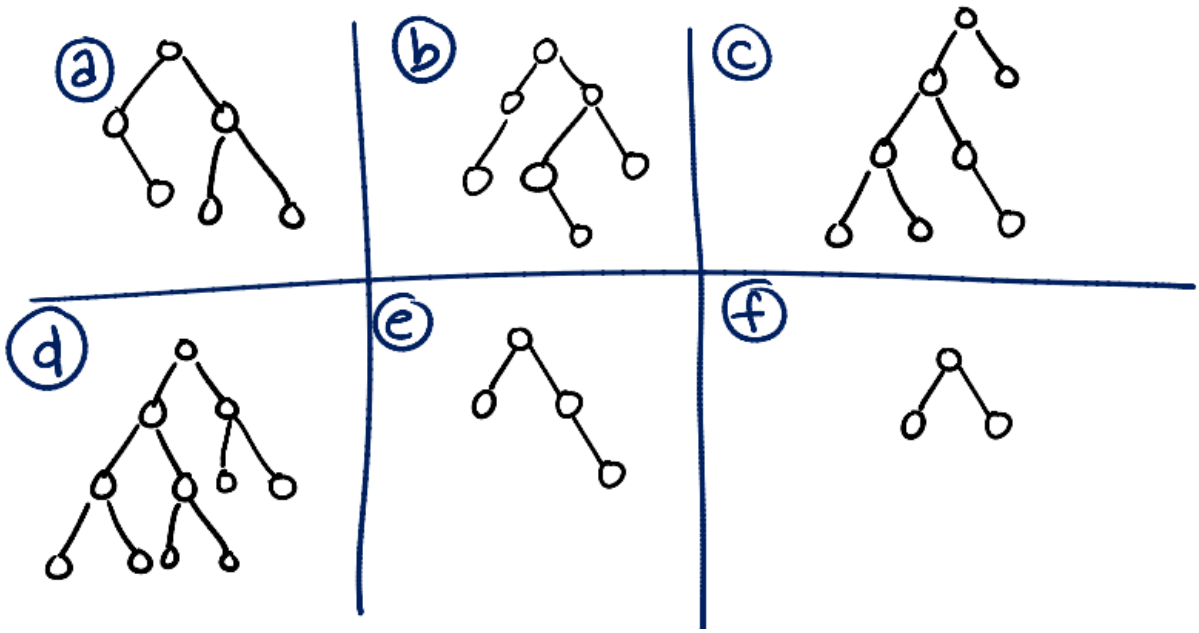
YZM 2116 – Veri Yapıları Dersi

Arasınav Soruları

Bahar 2018

Adı ve Soyadı	YANIT ANAHTARI	Öğrenci Numarası	
Grubu		İmza	
Tarih	02.04.2018	Not	/100

Soru#1 (10 puan): Aşağıda **a, b, c, d, e ve f** şeklinde isimlendirilmiş **altı adet ağaç** verilmiştir:



Buna göre, yukarıdaki "a, b, c, d, e ve f" **ağaçlarını, tam ikili ağaç** *(full binary tree),* **eksiksiz ikili ağaç** *(complete binary tree)* **ve katı ikili ağaç** *(strict binary tree)* **özellikleri taşımalarına göre gruplandırıp aşağıdaki tabloyu doldurunuz.** Not: Herhangi bir ağaç, birden fazla özelliği taşıyor olabilir.

İkili Ağaç Özelliği	İlgili Özelliği Örnekleyen/Taşıyan Ağaçlar
Tam İkili Ağaç	f
Eksiksiz İkili Ağaç	d, f
Katı İkili Ağaç	d, f

Soru#2 (15 puan): Aşağıdaki kod parçalarının hesaplama karmaşıklıklarını (koşma süreleri bakımından), *Big-O* hesaplama kuralları doğrultusunda değerlendirerek, en kötü durum senaryoları için *Big-O* değerlerini belirleyiniz. Ara işlemlerin yazılması zorunlu değildir.

Kod Parçası		Karmaşıklık Sınıfı (Big-O)

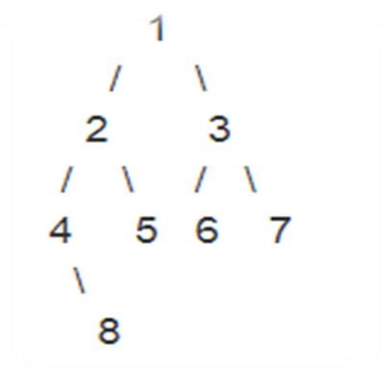
i.	`void silly(int n, int x, int y) {`
	`  if (x < y) {`
	`  for (int i = 0; i < n; ++i)`
	`  for (int j = 0; j < n * i; ++j)`
	`  System.out.println("y = " + y);`
	`  } else {`
	`  System.out.println("x = " + x);`
	`  }`
	`}`
		$O(n^3)$

ii.	`void silly(int n) {`
	`  j = 0;`
	`  while (j < n) {`
	`  for (int i = 0; i < n; ++i) {`
	`  System.out.println("j = " + j);`
	`  }`
	`  j = j + 5;`
	`  }`
	`}`
		$O(n^2)$

iii.	`void silly(int n) {`
	`  for (int i = 0; i < n * n; ++i) {`
	`  for (int j = 0; j < n; ++j) {`
	`  for (int k = 0; k < i; ++k)`
	`  System.out.println("k = " + k);`
	`  for (int m = 0; m < 100; ++m)`
	`  System.out.println("m = " + m);`
	`  }`
	`  }`
	`}`
		$O(n^5)$

iv.	`void f2(int n) {`
	`  for(int i=0; i < n; i++) {`
	`  for(int j=0; j < 10; j++) {`
	`  for(int k=0; k < n; k++) {`
	`  for(int m=0; m < 10; m++) {`
	`  System.out.println("!");`
	`  } } } }`
	`}`
		$O(n^2)$

Soru#3 (20 puan): Ağaç üzerinde gezinme (*traversal*) temelde "*inorder*", "*postorder*" ve "*preorder*" şeklinde üç farklı yöntemle yapılabilmektedir. Örneğin, inorder gezinme sonucu {4, 8, 2, 5, 1, 6, 3, 7} ve postorder gezinme sonucu {8, 4, 5, 2, 6, 7, 3, 1} sıralamaları elde edilen bir ikili ağaç *(binary tree)* veri yapısı için aşağıda verilmiş olan ağaç yapısı oluşturulabilir:



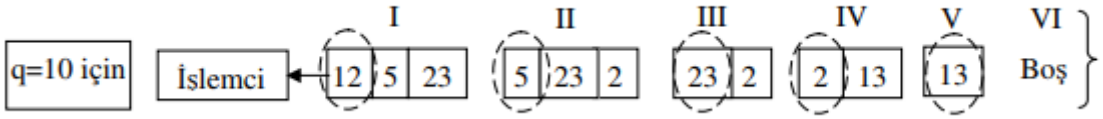
Buna göre, "Inorder" ve "Postorder" gezinme sıralamaları verilen bir ikili ağacı oluşturan programı yazınız. Algoritmanızın etkinlik sınıfını belirleyiniz. [Gerçekleştirimde, C#, Java, C vb. diller kullanılabilir. Çözüm, sözde kod ile algoritmik olarak da ifade edilebilir.]

$O(n^2)$

```
// Java program to construct a tree using inorder
// and postorder traversals

/* A binary tree node has data, pointer to left
   child and a pointer to right child */
class Node
{
    int data;
    Node left, right;

    public Node(int data)
    {
        this.data = data;
        left = right = null;
    }
}

// Class Index created to implement pass by reference of Index
class Index
{
    int index;
}

class BinaryTree
{
    /* Recursive function to construct binary of size n
       from  Inorder traversal in[] and Preorder traversal
       post[].  Initial values of inStrt and inEnd should
       be 0 and n -1.  The function doesn't do any error
       checking for cases where inorder and postorder
       do not form a tree */
    Node buildUtil(int in[], int post[], int inStrt,
            int inEnd, Index pIndex)
    {
        // Base case
        if (inStrt > inEnd)
            return null;
```

```
        /* Pick current node from Preorder traversal using
           postIndex and decrement postIndex */
        Node node = new Node(post[pIndex.index]);
        (pIndex.index)--;

        /* If this node has no children then return */
        if (inStrt == inEnd)
            return node;

        /* Else find the index of this node in Inorder
           traversal */
        int iIndex = search(in, inStrt, inEnd, node.data);

        /* Using index in Inorder traversal, construct left and
           right subtress */
        node.right = buildUtil(in, post, iIndex + 1, inEnd, pIndex);
        node.left = buildUtil(in, post, inStrt, iIndex - 1, pIndex);

        return node;
    }

    // This function mainly initializes index of root
    // and calls buildUtil()
    Node buildTree(int in[], int post[], int n)
    {
        Index pIndex = new Index();
        pIndex.index = n - 1;
        return buildUtil(in, post, 0, n - 1, pIndex);
    }

    /* Function to find index of value in arr[start...end]
       The function assumes that value is postsent in in[] */
    int search(int arr[], int strt, int end, int value)
    {
        int i;
        for (i = strt; i <= end; i++)
        {
            if (arr[i] == value)
                break;
        }
        return i;
    }

    /* This funtcion is here just to test  */
    void preOrder(Node node)
    {
        if (node == null)
            return;
        System.out.print(node.data + " ");
        preOrder(node.left);
        preOrder(node.right);
    }

    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        int in[] = new int[]{4, 8, 2, 5, 1, 6, 3, 7};
        int post[] = new int[]{8, 4, 5, 2, 6, 7, 3, 1};
        int n = in.length;
        Node root = tree.buildTree(in, post, n);
        System.out.println("Preorder of the constructed tree : ");
        tree.preOrder(root);
    }
}
```

Soru#4 (15 puan): **"Round robin iş zamanlaması"** algoritmasında, işlerin boyutları (büyüklüğü) değişken olabilmektedir. Quantum (q) adı verilen belli bir boyuttan büyük işlerin q boyutu kadarlık kısmı tamamlanıp, kalanı kuyruğun sonuna tekrar eklenmektedir. q boyutundan küçük (veya eşit) işler, boyutları kadar sürede tamamlanıp kuyruktan tamamen silinmektedir:



Bu işlemi gerçekleştiren kuyruk sınıfını, enque (ekle) ve deque (sil) metotlarını yazınız. Deque içerisinde, tamamı bitirilemeyen işlerin kuyruğa belirtilen şekilde eklenmesini de gerçekleştiriniz. Tüm işlemlerin, yaklaşık aynı anda geldiklerini varsayabilirsiniz. Sil metodu içinde, etkin işin ve (o ana kadar tamamlanan işler için) ortalama tamamlanma süresini de hesaplatıp yazdırınız. Kuyruk altyapısında Vector kullanınız. [Gerçekleştirimde, C#, Java, C vb. diller kullanılabilir. Çözüm, sözde kod ile algoritmik olarak da ifade edilebilir.]

```
class RoundRobin
{ Vector<Integer> v; int q = 10, sayac, time, toplam;

 RoundRobin() { v = new Vector<Integer>(); sayac = 0; time = 0; toplam = 0; }

 public void enque(int sayi) { v.add(sayi); }

 public int deque() {
 int prosesSize = v.get(0); v.removeElementAt(0);
 if (prosesSize>q && (v.size()!=0)) {
 time+=q; enque(prosesSize-q); }
 else {
 time+=prosesSize;
 System.out.println("Etkin Is Tamamlanma Suresi : "+time);
 toplam+=time; sayac++;
 System.out.println("Ortalama Tamamlanma Suresi : "+(toplam/sayac));
 }
 return prosesSize;
 }
 }
```

Soru#5 (30 puan): Verilen bir string ifadesi "açılış" ve "kapanış" parantezleri içermektedir. Bu string ifade içerisindeki açılış ve kapanış parantezi eşleşen en uzun parça, ***"en uzun geçerli parantez altstringi"*** olarak tanımlanmaktadır. Aşağıda farklı girdiler için algoritmanın oluşturması gereken çıktı ve en uzun geçerli parantez altstringi sunulmaktadır:

Girdi : (((
Çıktı : 2
En Uzun Geçerli Parantez Altstring : ()

Girdi:)()())
Çıktı : 4
En Uzun Geçerli Parantez Altstring: ()()

Girdi: ()((((((
Çıktı: 6
En Uzun Geçerli Parantez Altstring: ()(())

Buna göre, verilen bir string ifadenin en uzun geçerli parantez altstringini ve altstring uzunluğunu yığın (stack) veri yapısı kullanarak, *O(n)* zaman karmaşıklığı ile çözünüz. [Gerçekleştirimde, C#, Java, C vb. diller kullanılabilir. Temel yığın operasyonları kullanılarak çözüm, sözde kod ile algoritmik olarak da ifade edilebilir.]

```
import java.util.Stack;

class Test
{
    // method to get length of the longest valid
    static int findMaxLen(String str)
    {
        int n = str.length();

        // Create a stack and push -1 as initial index to it.
        Stack<Integer> stk = new Stack<>();
        stk.push(-1);

        // Initialize result
        int result = 0;

        // Traverse all characters of given string
        for (int i=0; i<n; i++)
        {
            // If opening bracket, push index of it
            if (str.charAt(i) == '(')
              stk.push(i);

            else // If closing bracket, i.e.,str[i] = ')'
            {
                // Pop the previous opening bracket's index
                stk.pop();

                // Check if this length formed with base of
                // current valid substring is more than max
                // so far
                if (!stk.empty())
                    result = Math.max(result, i - stk.peek());
```

```
                // If stack is empty. push current index as
                // base for next valid substring (if any)
                else stk.push(i);
            }
        }

        return result;
    }

    // Driver method
    public static void main(String[] args)
    {
        String str = "((()()";
        System.out.println(findMaxLen(str));

        str = "()(()))))";
        System.out.println(findMaxLen(str));

    }
}
```

Soru#6 (10 puan): A=[23, 3, 87, 34, 39, 1] dizisi için aşağıdaki sıralama algoritmalarını belirtilen adım sayısı kadar işleterek her adım sonunda dizinin son halini yazınız.

Adım Sayısı	Bubble Sort (Kabarcık Sıralama)	Selection Sort (Seçimli Sıralama)
1. İşletim Adımı	3 23 87 34 39 1	1 3 87 34 39 23
2. İşletim Adımı	3 23 87 34 39 1	1 3 87 34 39 23
3. İşletim Adımı	3 23 34 87 39 1	1 3 23 34 39 87