Benha University
Faculty of Science
Dept. of Mathematics

Time: Two Hours
First Semester 2014-2015
Date : 5/1/2015

Special Topics in Computer Science (1) (MC355) for Third Level Students (Computer Science)

جامعة بنها – كلية العلوم – قسم الرياضيات

المستوي الثالث (علوم حاسب)

يوم الامتحان: الاثنين

تاريخ الامتحان: 5 / ١ / ٢٠١٥ م

المادة : موضوعات مختارة في علوم الحاسب(١)   (٣٥٥ رس)

الممتحن: د/ مصعب عبد الحميد محمد حسان

مدرس بقسم الرياضيات بكلية العلوم

الاسئلة و نموذج الإجابة

ورقة كاملة

Benha University
Faculty of Science
Dept. of Mathematics

Time: Two Hours
First Semester 2014-2015
Date : 5/1/2015

Special Topics in Computer Science (1) (MC355) for Third Level Students (Computer Science)

Answer the following questions:

Question 1. (12 marks)

A-Define balance factor, collision, multigraph, graph invariant. (4 marks)

B-Write a function to compute the balance factor of a given node. Show how can we compute the balance factor of all nodes in our tree. (8 marks)

Question 2. (14 marks)

A-Draw the binary search tree by inserting the following sequence into an initially empty tree:

23 14 24 13 18

Test if this tree is AVL tree or not? Suppose we insert a new node of data 16, test that the tree is AVL tree or not and if the tree is not AVL tree show how can we rebalance it. (4 marks)

B-Write a function to apply the right rotation round node p. (4 marks)

C-Consider the following hash function:

$$H(S)=H(\text{"}s_1s_2\ldots\, s_n\text{"})=s_1+p.s_2+p^2.s_3+\ldots+p^{n-1}.s_n,$$

where p is a prime number

. Suppose we construct our hash table using separate chaining idea, write a function to insert a string into the hash table and write a function to search for a string. (6 marks)

Question 3. (10 marks)

A-Discuss Ullman algorithm for graph isomorphism and subgraph isomorphism in details. (7 marks)

B-Compare between adjacency list representation and adjacency matrix representation of graphs. (3 marks)

Question 4. (12 marks)

Using adjacency matrix representation:

(A)Write a function to test if the given graph is complete or not. (2 marks)

(B)Write a function to delete a node. (2 marks)

(C)Write a function to calculate the two degree sequences of two graphs and check if the two graphs are not isomorphic based on degree sequence. (5 marks)

(D)Write a function to check if there is a path of length two from node i to node j. (3 marks)

Best Wishes
Dr. Mosab Abd El-Hameed

Benha University
Faculty of Science
Dept. of Mathematics

Time: Two Hours
First Semester 2014-2015
Date : 5/1/2015

Special Topics in Computer Science (1) (MC355) for Third Level Students (Computer Science)

Answer of Question 1

A-

Balance factor of a node x is the height of the left subtree of x minus the height of x's right subtree.

The items would hash into the same location, creating what we call a **"collision".**

If there is more than one edge between any two vertices the graph is called **multigraph**

A graph invariant is a function T such that if applied to two isomorphic graphs H and G, then T(H) = T(G). In other words, if T(H) ≠ T(G) then H is not isomorphic to G.

B-

```
int height(node *temp)
{
            int h = 0;
            if (temp != NULL)
            {
               int l_height = height (temp->left);
               int r_height = height (temp->right);
               int max_height = max (l_height, r_height);
               h = max_height + 1;
            }
            return h;
}

int diff(node *temp)
{
   int l_height = height (temp->left);
   int r_height = height (temp->right);
   int b_factor = l_height - r_height;
   return b_factor;
}
```

Note that we can compute the balance factor of all nodes in our tree by calling the function ***diff*** in traversing tree function (inorder).

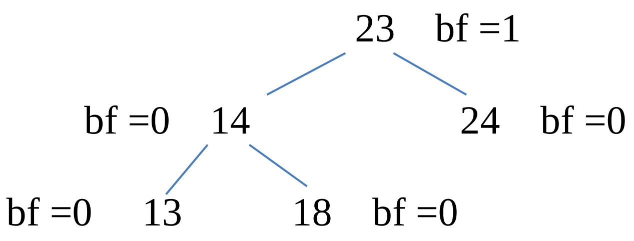
```
void inorder(node * root_node)
{
  if(root_node != NULL){

        inorder(root_node->left);
        root_node->balance_factor = diff(root_node);
        cout << "Data = " << root_node->info << ", bf = "
              << root_node->balance_factor << endl;
        inorder(root_node->right);

    }
  }
```

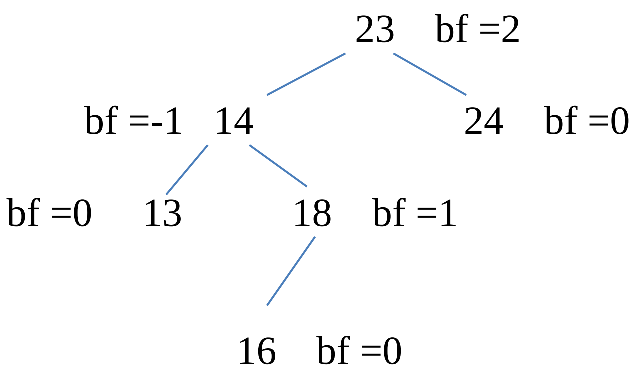
Answer of Question 2

A-



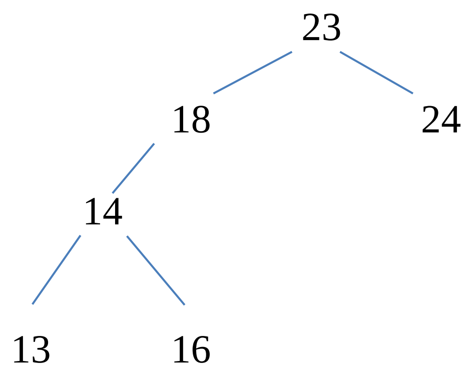
The Above tree is AVL tree.

After inserting a new node of data 16, the tree is not AVL tree

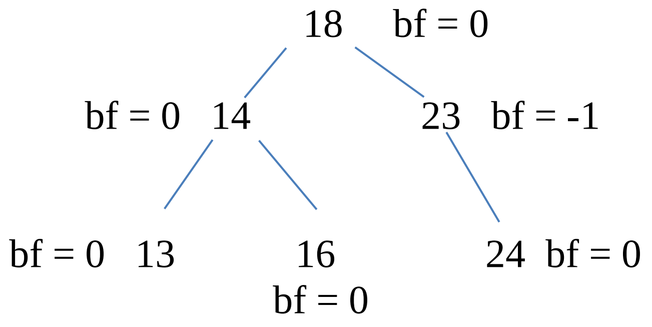


we can we rebalance it using left-right rotation.

First we perform a left rotation of the nodes in the left subtree of the nearest ancestor with balance factor 2



Now we apply a right rotation to the tree



B-

```
 node* rotateright(node* p) // the right rotation round p
{
   node* q = p->left;
   p->left = q->right;
   q->right = p;
   return q;
}
```

C-

```
void insert(char *s){
     int sum = A[s[strlen(s)-1]]; //The array A map each char in the string to
                                                   number
     int p = 3;
     for(int i =1; i <strlen(s); i++)
        sum = A[s[strlen(s)-i-1]] + p*sum;
     Hash_Table[sum].AddAtBeg(s);
   }
```

```
void search(char *s){
    int sum = A[s[strlen(s)-1]]; //The array A map each char in the string to
                                  number
    int p = 3;
    for(int i =1; i <strlen(s); i++)
      sum = A[s[strlen(s)-i-1]] + p*sum;
    Hash_Table[sum].Search(s); //in fun serach we must compare the two
    strings are equal or not
  }
```

<u>Answer of Question 3</u>

A-

Ullman algorithm is the earliest and highly-cited approach to the (sub)graph isomorphism problem. Given two graphs G1 and G2. To check if G1 is subgraph of G2, Ullman's basic approach is to enumerate all possible mappings of vertices in V_{G1} to those in V_{G2} using a depth-first tree-search algorithm. In order to cope with subgraph isomorphism problem efficiently, Ullman proposed a refinement procedure to prune the search space. It is based on the following three conditions:

1. *Label and degree condition.*
   A vertex $u \in V_{G1}$ can be mapped to $v \in V_{G2}$ under injective mapping f, i.e $v = f(u)$, if
      (i) $L_{G1}(u) = L_{G2}(v)$, and
      (ii) $\deg_{G1}(u) \leq \deg_{G2}(v)$.

2. *One-to-One mapping of vertices condition.*
   Once vertex $u \in V_{G1}$ is mapped to $v \in V_{G2}$, we cannot map any other vertex in V_{G1} to the vertex $v \in V_{G2}$.

3. *Neighbor condition.*
   By this condition Ullman algorithm examines the feasibility of mapping $u \in V_{G1}$ to $v \in V_{G2}$ by considering the preservation of structural connectivity. If there exist edges connecting u with previously explored vertices of G1 but there are no counterpart edges in G2, the mapping test simply fails.

Considering the graph isomorphism instead of the subgraph isomorphism, the two graphs must have the same number of vertices and the condition 1 is modified as the following :-

Benha University
Faculty of Science
Dept. of Mathematics

Time: Two Hours
First Semester 2014-2015
Date : 5/1/2015

Special Topics in Computer Science (1) (MC355) for Third Level Students (Computer Science)

1. *Label and degree condition.*
A vertex $u \in V_{G1}$ can be mapped to $v \in V_{G2}$ under bijective mapping f, i.e $v = f(u)$, if

(i) $L_{G1}(u) = L_{G2}(v)$, and
(ii) $\deg_{G1}(u) = \deg_{G2}(v)$.

B-

Comparison	**Winner**
Faster to test if (x, y) is in graph?	adjacency matrices
Faster to find the degree of a vertex?	adjacency lists
Less memory on small graphs?	adjacency lists
Less memory on big graphs?	adjacency matrices
Edge insertion or deletion?	adjacency matrices
Faster to traverse the graph?	adjacency lists
Better for most problems?	adjacency lists

Table : Relative advantages of adjacency lists and matrices.

Answer of Question 4

A-
```
Bool Test_Complete(int V, int E){
            int m = V * (V - 1) / 2 ;
            if(m == E)
              return true;
            else
              return false;
     }
```

B-
```
void Delete_Node(int u, int & V1)
{


   if(V1 == 0)
   {
     cout << "Graph is empty" << endl;
     return;
```

```
    }

    if ( u > V1 - 1 )
    {
      cout << "This node is not present in the graph" << endl ;
      return;
    }

for(int i = u   ; i < V1 - 1; i++)
       for(int j = 0; j < V1 ; j++)
       {
         if(i == j)
           continue;
         Adj_Matrix[j][i] = Adj_Matrix[j][i+1]; /* Shift columns left */
         Adj_Matrix[i][j] = Adj_Matrix[i+1][j]; /* Shift rows up */
       }

    V1--; /*Decrease the number of nodes in the graph */
}
```

C-

```
void Find_Degree_Sequence(int V, bool Adj_Matrix[][100], int deg[]){

      //Display the Adj_Matrix
     for(int l = 0; l < V; l++){
      int deg_vertex_l = 0;
      for(int k = 0; k < V; k++)
        if(Adj_Matrix[l][k] == 1)
          deg_vertex_l ++;

      deg[l] = deg_vertex_l;
     }

}
```

```
void insertion_sort(int V, int Arr[]){
   for(int j = 1; j < V; j++){
     int key = Arr[j];
     int i = j-1;
     while(i >= 0 && Arr[i] > key){
       Arr[i+1] = Arr[i];
       i = i-1;
     }
     Arr[i+1] = key;
   }
}


Void    Compare_Degree_Sequence_Two_Graphs(){
//Here we get the degree sequence of the two graphs


     int deg1[V1];
     int deg2[V2];


     Find_Degree_Sequence(V1, Adj_Matrix1, deg1);


     Find_Degree_Sequence(V2, Adj_Matrix2, deg2);


     cout << "Degree Seq of the first graph: " << endl;
     cout << "-------------------------------" << endl;
     for(int i = 0  ; i < V1; i++)
       cout << "Degree of vertex " << i << " = " << deg1[i] << endl;


     cout << endl;



     cout << "Degree Seq of the second graph: " << endl;
     cout << "-------------------------------" << endl;
     for(int i = 0  ; i < V2; i++)
       cout << "Degree of vertex " << i << " = " << deg2[i] << endl;



     //Two test the two degree sequences, we must sort them in increasing
order or decreasing order
     insertion_sort(V1, deg1);
     for(int i = 0  ; i < V1; i++)
       cout << deg1[i] << " ";
     cout << endl;

     insertion_sort(V2, deg2);
     for(int i = 0  ; i < V2; i++)
       cout << deg2[i] << " ";
     cout << endl;



     for(int i = 0  ; i < V1; i++)
      if(deg1[i] != deg2[i]){
          cout << "Two graphs are not isomorphic " << endl;
          return 0;
      }

  }
```

```
D- bool fun(int i, int j, int V){
       for(int k = 0; k < V; k++)
          if(Adj_Matrix[i][k] == 1 && Adj_Matrix[k][j] == 1)
              return true;
       return false;
     }
```