Lecture 9

Tree

Abstract idea of a tree:

هيكل من هياكل البيانات تستخد

التخزين ليس خطي بل متفرع *

لا تشبه A tree is another data structure that you can use to store information. Unlike stacks and queues, which are linear data structures, **trees are hierarchical data structures.** Saying that the structure of a tree is hierarchical means that things are ordered above or below other things. ~~Here is an~~ example of a tree holding letters: بانا هرميه

```
          tree
          ----
           j    <-- root
          / \
Node <-- f    k
        / \    \
       a   h    z   <-- leaves
```

الاوراق

Tree Vocabulary

العنصر الموجود في أول هو root

Let's now introduce some vocabulary with our sample tree... The element at the top of the tree is called the **root**. The elements that are directly under an element are called its **children**. The element directly above something is called its **parent**. For example, **a** is a child of **f** and **f** is the parent of **a**. Finally, elements with no children are called **leaves**.

أي نود موجودة تحت نود آخر هي Children

الي مو طالع منها عناصر هي Leves node

Aside: If you were to draw the picture above upside down, it would look like a real tree, with the leaves at the top and the root at the bottom...However, we usually draw tree data structures as we've done above.

الرسمه الي فوق هي تشبه الحقيقة tree

Uses

لتخزين المعلومات بالطريقة الهرمية

We use trees you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

مثال على tree

```
file system
------------
     /    <-- root
   /   \
...   home
    /       \
Students     course
  /          /  |  \
 ...   cs101 cs112 cs113
```

هي عباره عن صفح tree

صفح tree

ارقام الملفات

بالرغم على التنظيم الهرمي للتري

Despite the hierarchical order of the structure of the tree, the order enforced on objects in the tree will depend on how we use the tree. This just means that unlike a stack whose operations are usually limited to push and pop, there are many different kinds of trees and ways to use them. Thus, this flexibility makes them more akin to linked lists.

فإن ترتيب العناصر يعتمد على استخدامي للتري.

فيه طرق لا يقتصر و لبنائه

* تشبه اللنكد لست بمرونتها

لا تشبه stack

Recursive Data Structure

هي عباره عن تكرار لبعض الاشياء لتكون الشجرة الكبيرة

A tree can be viewed as a recursive data structure. **Why? Remember that recursive means that something is defined in terms of itself.** Here this means that trees are made up of subtrees.

For example, let's look at our tree of letters and examine the part starting at **f** and everything under it...

```
     tree
     ----
      j
     / \
    f   k
   / \   \
  a   h   z
```

صف تري

صف تري (اشجار فرعية)

Doesn't it look like a tree itself? In this subtree, **f** is the root

ما تزيد عن 2

Binary Trees

عدد لا يزيد عن 2

We can talk about trees where the number of children that any element has is limited. In the tree above, no element has more than 2 children. For the rest of this example, we will enforce this to be the case.

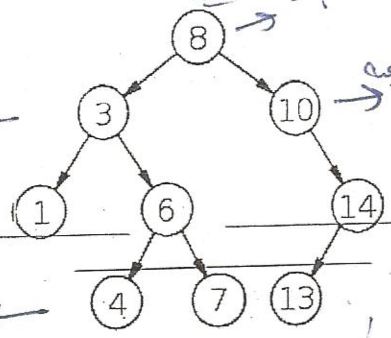
محدود

A tree whose elements have at most 2 children is called a binary tree.

Since each element in a binary tree can have only 2 children, we typically name them the *left* and *right* child.

Binary search tree

الشجرة الاصلية



شجرة فرعية (sub tree)

مستوى (1)

مستوى (2)

مستوى (3)

A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

A binary search tree (BST), which may sometimes also be called an ordered ~~or sorted binary tree, is a node-based binary tree data structure which has~~ the following properties:

كل اللي على يمين الجذر أصغر من الجذر مثل 1 أصغر من 3 و 4 أصغر من 6 - واللي على اليمين أكبر منه

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

المعلومات الموجودة في كل node ما يقدر يكون فقط data واحد بل نحتاج لتركيبة كاملة

Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records.

The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms can be very efficient.

فعالة جدا

هي بنية أساسية مستخدمة لبناء هياكل بيانات أخرى

قراءة

Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays.

أمثلة عليها

Tree operations: هذا الي راح نشتغل عليه

As mentioned, there are different kinds of trees (e.g., binary search trees, 2-3 trees, AVL trees, tries,.....).

What operations we will need for a tree, and how they work, depends on what kind of tree we use. However, there are some common operations we can mention:

1. **Add**: اضافة عنصر Tree

   تعتمد على

   Places an element in the tree (where elements end up depends on the kind of tree). For example, **Add(tree, i)** might give:

```
        tree
        ----
         j    <-- root
        / \
       f   k
      / \   \
     a   h   z
          \
           i    <-- new leaf
```

   هو الي يرتب على كيفه العنصر الأسبق أبجدياً يروح لليسار ـ إذا أرقام الرقم الأصغر يروح لليسار

   f حطها يسار لأنها f أبجدية قبل K

2. **Remove**: يخليني أعيد الترتيب بعد الحذف.

   Removes something from the tree (how the tree is reorganized after a removal depends on the kind of tree). For example, **Remove(tree, h)** might give:

```
        tree
        ----
         j    <-- root
        / \
       f   k
      / \   \
     a   i   z
```

   حذف h و i رفعها فوق

   Here, i moved up to take its place.

3. **IsMember**:

   Reports whether some element is in the tree.

For example, **IsMember(tree, a)** should give a true value and **IsMember(tree, y)** should give a false value.

Tree representation:

Since we want to be able to represent a tree in C++, how are we going to store this hierarchical structure?

Can we use an array?

Answer: Certainly! There are times when we can use an array to represent a tree.

However, we can also do something along the lines of a linked list. For example, just as linked list nodes hold one element and point to the next node...

```
 |
 v
---------    ---------    ---------
| a | --+---> | b | --+---> | c | 0 |
---------    ---------    ---------
```

we could have tree nodes that hold one element and point to their children...

```
          |
          v
        -----
        | j |
        |---|
        | | |
        /---\
       v     v
    -----   -----
    | f |   | k |
    |---|   |---|
    | | |   |0| |
    /---\   ----\
   v     v       v
 -----  -----   -----
 | a |  | h |   | z |
 |---|  |---|   |---|
 |0|0|  |0|0|   |0|0|
 -----  -----   -----
```

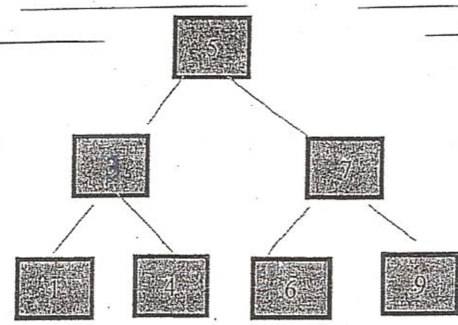
هي ال Pointer

Note that some nodes don't have a left and/or right child, so those pointers are NULL.

Also, just as we need a pointer to the first node to keep track of a linked list; here, we need a pointer to the root node to keep track of a tree.

Tree implementation in C++

The following program creats a tree, insert elements into it and then print it's elements




```
# include <iostream.h>

# include <stdio.h>

# include <stdlib.h>


struct btreenode
{
    int content;
    struct btreenode *left;
    struct btreenode *right;
};
```

مؤشر للفت

مؤشر right

```
typedef btreenode *p;
typedef p p1;
```

مؤشرات للـ struct

```
p1 insert(btreenode *nodePtr, int item)
{
    if(nodePtr == NULL)
    {
```

هقصد الترتيب الي قبل شوي عرفناه

```
        nodePtr = new btreenode;
        nodePtr->content=item;
        nodePtr->left=nodePtr->right=NULL;
    }
```

بناء العقده الجديد

مثلاً 3

تستدعي الدالة insert

```
    else if(item < nodePtr->content)
        nodePtr->left = insert(nodePtr->left, item);
```

اذا item أصغر يروح لليسار

استدعاء للدالة insert عشان نبني في اليسار

```
    else if (item > nodePtr->content)
        nodePtr->right = insert(nodePtr->right, item);
```

اذا أكبر يروحها لليمين

```
    return nodePtr;
}
```

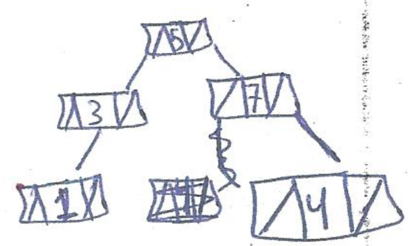
للطباعة ← void printpreoder(btreenode *nodeptr) ← ثابته

```
void printPreOrder(btreenode *nodePtr)
{
    if(nodePtr!=NULL)
    {
        cout<< nodePtr->content;
        printPreOrder(nodePtr->left);
        printPreOrder(nodePtr->right);
    }
}
```

بما ان root موجوده هذا معناه انها مبنية

انا اطبع

ASool



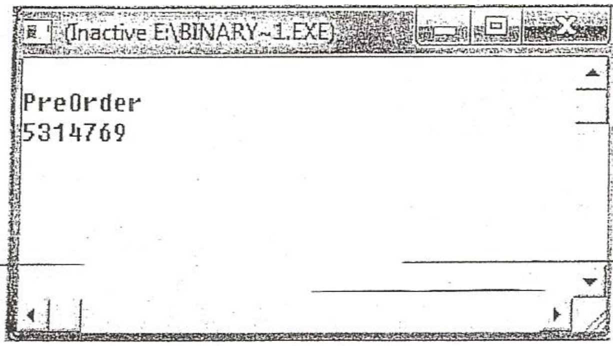
```
void main()
{
    int item;
    p1   rootPtr = NULL;
    rootPtr = insert(rootPtr, 5);
    rootPtr = insert(rootPtr, 3);
    rootPtr = insert(rootPtr, 7);
    rootPtr = insert(rootPtr, 1);
    rootPtr = insert(rootPtr, 4);
    rootPtr = insert(rootPtr, 6);
    rootPtr = insert(rootPtr, 9);
    // * Traversing the tree in Preorder *//
    cout<<"\nPreOrder\n";
    printPreOrder(rootPtr);

}
```

لأنه ماسك 5

يرجع

Run



Tree applications: (Expression Trees)

نخزنها للتطبيقات الرياضية

One application of trees is to store mathematical expressions such as 15*(x+y) or sqrt(42)+7 in a convenient form. Let's stick for the moment to expressions made up of numbers and the operators +, -, *, and /. Consider the expression 3*((7+1)/4)+(17-5). This expression is made up of two subexpressions, 3*((7+1)/4) and (17-5), combined with the operator "+". When the expression is represented as a binary tree, the root node holds the operator +, while the subtrees of the root node represent the subexpressions 3*((7+1)/4) and (17-5). Every node in the tree holds either a number or an operator. A node that holds a number is a leaf node of the tree. A node that holds an operator has two subtrees representing the operands to which the operator applies. The tree this type is refered to as an expression tree.

يتكون من تعبيرين

كل نود ممكن تكون رقم أو معامل

الارقام هي (Leaves node)

Given an expression tree, it's easy to find the value of the expression that it represents. Each node in the tree has an associated value. If the node is a leaf node, then its value is simply the number that the node contains. If the node contains an operator, then the associated value is computed by first finding the values of its child nodes and then applying the operator to those values. The process is shown by the red arrows in the illustration. The value computed for the root node is the value of the expression as a whole. There are other uses for expression trees. For example, a postorder traversal of the tree will output the postfix form of the expression.