CS32 Discussion
Week 10

Muhao Chen

muhaochen@ucla.edu

http://yellowstone.cs.ucla.edu/~muhao/
Outline

• Topics for final
Trees

• How to traverse through a Tree?
  • Recursive orders?
  • Level order? (Use a queue)
• Reconstruct a binary tree from traversal results (*)
• Variation of traversal
  • Count nodes
  • Count edges
  • Count leaf nodes
  • Calculate the height of the tree
• Properties of Trees
  • A tree with $n$ nodes is with $n-1$ edges
  • A full binary tree has $2^{h+1}-1$ nodes (if $h$ starts at 0)
Problem: reconstruct a binary tree

• How’s the **full binary tree** look for which the **pre-order** traversal is:
  • U C N L A G E

• How’s the **full binary tree** look for which **post-order** traversal is:
  • U C N L A G E
Problem: Traverse a Regular Tree

• Given a regular tree node definition as below, write the function `int nodeCount(Node *root)` to count number of nodes in the tree.
  • Consider: recursive traversal or non-recursive traversal?

```cpp
struct Node {
    int val;
    vector<Node *> children;
};
```
Solution 1: recursive traversal

```cpp
int nodeCount(Node *root) {
    if (root -> children.empty()) return 1; //optional
    int num = 1;
    for (int i = 0; i < root -> children.size(); ++i)
        num += nodeCount(root -> children[i]);
    return num;
}
```
Solution 2: level-order traversal (non-recursive)

```cpp
int nodeCount(Node *root) {
    queue<Node *> q;
    q.push_back(root);
    int num = 0;
    Node *tmp;
    while (!q->empty()) {
        tmp = q.front();
        q.pop_front();
        ++num;
        for (int i=0; i<tmp->children.size(); ++i)
            q.push_back(tmp->children[i]);
    }
    return num;
}
```
Question: Time complexity of nodeCount?

• Recursive traversal?
• Non-recursive traversal?

• $O(n)$ as we visit each node for once.
int edgeCount(Node *root) {

    //solution 1
    int num = root->children.size();
    for (int i = 0; i < root->children.size(); ++i)
        num += edgeCount(root->children[i]);
    return num;

    //solution 2
    return nodeCount(root) – 1;
}

Question: Edge count?
Problem: Leaf count?

• Write a function `int leafCount(Node *root)` that counts number of leaf nodes in the tree.
  • How do we decide whether a node is a leaf node?
• How do we modify `nodeCount` into `leafCount`?

```c
struct Node {
    int val;
    vector<Node *> children;
};
```
Solution 1: recursive traversal

```c++
int leafCount(Node *root) {
    if (root -> children.empty()) return 1;
    int num = 0;
    for (int i = 0; i < root -> children.size(); ++i)
        num += leafCount(root -> children[i]);
    return num;
}
```
Solution 2: level-order traversal (non-recursive)

```cpp
int leafCount(Node *root) {
    queue<Node *> q;
    q.push_back(root);
    int num = 0;
    Node *tmp;
    while (!q->empty) {
        tmp = q.front();
        q.pop_front();
        if (tmp->children.empty()) ++num;
        else {
            for (int i=0; i<tmp->children.size(); ++i) {
                q.push_back(tmp->children[i]);
            }
        }
    }
    return num;
}
```
Heap

• Definition of heap? maxHeap: complete binary tree, parent ≥ children

• How a heap is fitted into an array? From up to down, left to right.

• findMax?
  • root value

• Insertion?
  • Append newVal after the array. While it’s larger than its parent ( (i-1)/2), keep popping it up (swap with the current parent) until it finds a larger parent.

• deleteMax
  • Remove the first element from the array. Move the last element to the first. While the new root is smaller than either of its children ( i * 2 + 1, i * 2 + 2), keeping swapping it with the larger child.
Problem: Heap

• Suppose we have a maxheap as below.

  15 10 14 7 9 8 11 4 3 5 6

• How does it look after one `deleteMax()`?

  14 10 11 7 9 8 6 4 3 5

• Then how does it look after `insert(12)`?

  14 12 11 7 10 8 6 4 3 5 9
Stack, Queue

• Definition of stack and queue:
  • Stack: last-in-first-out (LIFO)
  • Queue: first-in-first-out (FIFO)

• Problems on stack:
  • Record the paths during DFS (e.g. mazes, routes in trees)
  • Infix, postfix expression

• Problems on queue:
  • Sliding window
Problem: find the least common ancestor in a binary tree

• The **least common ancestor (LCA)** of two nodes $n1$ and $n2$ is the closest ancestor node of $n1$ and $n2$ (and it’s the furthest ancestor from the root).
  
  • E.g. in the example tree, LCA of node 4 and node 3 is node 1; that of node 5 and node 7 is node 3.

Node* LCA(Node *root, Node *n1, Node *n2)

Hint: what’s common on the two paths from the root to n1 or n2? Where does LCA appear on those two routes

```c
struct Node {
    int val;
    Node *left;
    Node *right;
};
```
Problem: find the least common ancestor in a binary tree

• The paths from the root to $n_1$ and $n_2$ always have the same prefix. LCA appears at the last of the prefix.

• We use two stacks, either of them records the path from the root to $n_1$ or $n_2$. Then we pop the two stacks towards the LCA.
Node *LCA(Node *root, Node *n1, Node *n2) {
    stack<Node *> route1, route2;
    if (!DFS_search(root, n1, route1)) return NULL; //route1: root ===⇒ n1
    if (!DFS_search(root, n2, route2)) return NULL; //route2: root ===⇒ n2
    while (route1.size() > route2.size()) route1.pop();
    while (route1.size() < route2.size()) route2.pop();
    while (route1.top() != route2.top()) {route1.pop(); route2.pop();}
    return route1.top();
}

bool DFS_search(Node *current, Node *n, stack<Node *> &S) {
    if (current == NULL) return false;
    S.push_back(current);
    if (current == n) return true; //found
    if (DFS_search(current->left, n, S) == true) return true; //found in left
    else if (DFS_search(current->right, n, S) == true) return true; //found in right
    else { //current node is not in the route towards n, pop it and trace back
        S.pop_back();
        return false;
    }
}
Problem: Infix to postfix

• Given infix expressions as below, what are their postfix expressions:
  • $4 + 13 / 5$
  • $3 + 6 * 7 * 8 – 3$
  • $(3 + 5) * (4 + 3 / 2) – 5$

  $4 \ 13 \ 5 \ / +$
  $3 \ 6 \ 7 \ * \ 8 \ * \ + \ 3 \ -$
  $3 \ 5 \ + \ 4 \ 3 \ 2 \ / \ + \ * \ 5 \ -$
Problem: evaluate a postfix expression

1. Start with the left-most token.
2. If the token is a **number**: 
   a. Push it onto the stack
3. If the token is an **operator**: 
   a. Pop the top two #s and calculate them, then push the result back
4. Repeat until all tokens are pushed into the stack, then return the top of the stack.
int evalRPN(vector<string>& tokens) {
    stack<int> nums;
    int a;
    for (auto &s : tokens) {
        if (s == "+") {
            a = nums.top(); nums.pop();
            nums.top() += a;
        } else if (s == "-") {
            a = nums.top(); nums.pop();
            nums.top() -= a;
        } else if (s == "/") {
            a = nums.top(); nums.pop();
            nums.top() /= a;
        } else if (s == "*") {
            a = nums.top(); nums.pop();
            nums.top() *= a;
        } else {
            nums.push(stoi(s));
        }
    }
    return nums.top();
}
Problem: Valid brackets

Given a mathematical expression containing parentheses “(“, “)”“, curly braces “{“, “}”“, and square brackets “[“, “]”“. Decide if three kinds of brackets are closed validly.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "()" and "()[\]" are not.

All brackets must close.

More examples:

(2 + 4) * 6 valid
[(2 + 4) * {15 - 20}] valid
{{12+30]} not valid
(((({{[[<<<*_>>>]]}]]})) valid
(((((())))))) not valid
Solution

• Use a stack to record brackets.

• Each time read a character from the expression.
  • If it’s any of ‘(’, ‘[’, or ‘{’, push it into the stack.
  • If it’s any of ‘)’, ‘]’, or ‘}’, remove the top from the stack and check if these two closes validly.
  • If it’s any other character, let it pass.

• The expression is valid iff the stack is empty in the end, all characters are processed, and no invalid pair of brackets are found.
bool isValid(const string &exp) {
    stack<char> S;
    for (int i=0; i<exp.size(); ++i) {
        char ch = exp[i];
        if (ch == '(' || ch == '{' || ch == '[') S.push_back(ch);
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (S.empty()) return false; //Extra closed bracket
            if (ch == ')' && S.top() == '(' && S.top() == '}' || ch == ']' && S.top() == '[')
                S.pop_back();
            else return false; //Miss match
        }
    }
    return S.empty(); //return whether no bracket is left in the stack
}
Hash Table

• Given hash function, insert several items to hash table, how will it look?
  • Close addressing? (Separate chaining)
  • Open addressing?
    • Linear Probing
    • Quadratic Probing

• Is a hash function good?
  • Does it always separate keys into different buckets with the same *probability*?
  • Does it require very little time to calculate the hash value?
Problem: Chaining

Is it a good hash function?

No. Because odd # buckets are never used.

```c
int hashFunc(int x) {
    return (x * 2) % HASH_SIZE;
}
```

Assume HASH_SIZE = 10. Here is the hash table’s insert function:

```c
void insert(int key) {
    int index = hashFunc(key);
    hash_array[index].push_back(key);
}
```

where hash_array is an array of list<int>’s.

```
insert(7);
insert(1);
insert(23);
insert(14);
insert(19);
insert(53);
insert(37);
insert(83);
```
Inheritance & Polymorphism

• Inheritance:
  • Be careful about orders of construction and destruction
  • class B : A
    • construct B <= construct A then construct B
    • delete B <= destruct B then destruct A

• Polymorphism
  • We have both member functions \textit{msg()} in A and B
  • \texttt{A *ptr = new B();}
  • If \textit{msg()} in A is not \textit{virtual}
    • \texttt{ptr -> msg()} runs the \textit{msg()} defined in A
  • If \textit{msg()} in A is \textit{virtual}
    • \texttt{ptr -> msg()} runs the \textit{msg()} defined in B
Problem: Polymorphism and Inheritance

```cpp
class A {
  public:
    A() : m_msg("Apple") {}
    A(string msg) : m_msg(msg) {}
    virtual ~A() { message(); }
    void message() const {
      cout << m_msg << endl;
    }
  private:
    string m_msg;
};

class B : public A {
  public:
    B() : A("Orange") {}
    B(string msg) : A(msg), m_a(msg) {}
    void message() const {
      m_a.message();
    }
  private:
    A m_a;
};

int main() {
  A *b1 = new B;
  B *b2 = new B;
  A *b3 = new B("Apple");
  b1->message();
  b2->message();
  b3->message();
  delete b1;
  delete b2;
  delete b3;
}
```

(a) How many **Apple** and **Orange** do you see in the output?
(b) If we change message() in A into a virtual function. `virtual void message() const;` How many **Apple** and **Orange** do you see in the output?
If `A::message()` is not `virtual`, you get:

- Orange  // `b1->message()`
- Apple   // `b2->message()`
- Apple   // `b3->message()`
- Apple   // destroying `m_a` of `b1`
- Orange  // destroying `b1`
- Apple   // destroying `m_a` of `b2`
- Orange  // destroying `b2`
- Apple   // destroying `m_a` of `b3`
- Apple   // destroying `b3`

6 Apple’s and 3 Orange’s

If `A::message()` is `virtual`, you get:

- Apple
- Apple
- Apple
- Apple
- Apple
- Apple
- Apple

7 Apple’s and 2 Orange’s
Other topics

• Big-O notation
• Binary Search Tree
• How does each sorting algorithm work?
• Complexity of all the algorithms you have learnt in CS32
Bugs in your software are actually special features :)

if ($thirsty==TRUE) {
} else {
}