CSSE 230 Day 22

Tree Variations
EditorTrees work time

Day 22 Announcements/Agenda

- WA 7: Due Tomorrow, 8 AM:
- EditorTrees Milestone 2: Due Friday, 8 AM
- WA 8: Due Tuesday Oct 30, 8 AM
- Exam 2: Thursday, Nov 1, 7-9 PM
- Agenda for today:
  - Tree variations
    - EBT reminder
    - Tries
  - Sorting overview
  - EditorTrees work time

What questions do you have?

Tree variations

- Expression Trees
- DAGs
- Tries

Q1

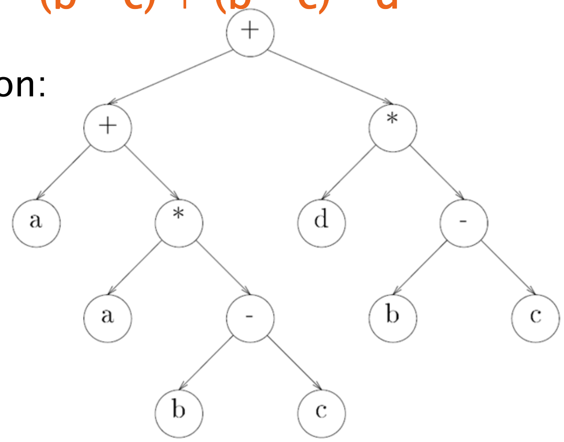
Expression Trees

- Could be used by a spreadsheet to store formula
- Used extensively by compilers and interpreters
- Each node represents an expression
- Child nodes represent sub-expressions
- Shape of the tree encodes:
  - Precedence
  - Associativity

- Consider:
  - 1+2
  - 2 + 3 * 4
  - 4 - 3 - 2
  - 2^3^4
  - IF(A1>1, 0, SUM(B1:G1))

Expression Tree Variation

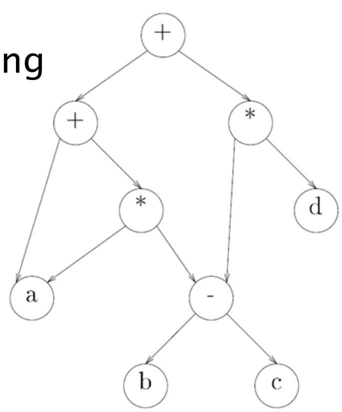
- Consider a tree that represents this expression: a + a * (b – c) + (b – c) * d
- Expression evaluation: Postorder
- Notice the common sub-expressions: a and (b – c)



Q2-3

Directed Acyclic Graph (DAG)

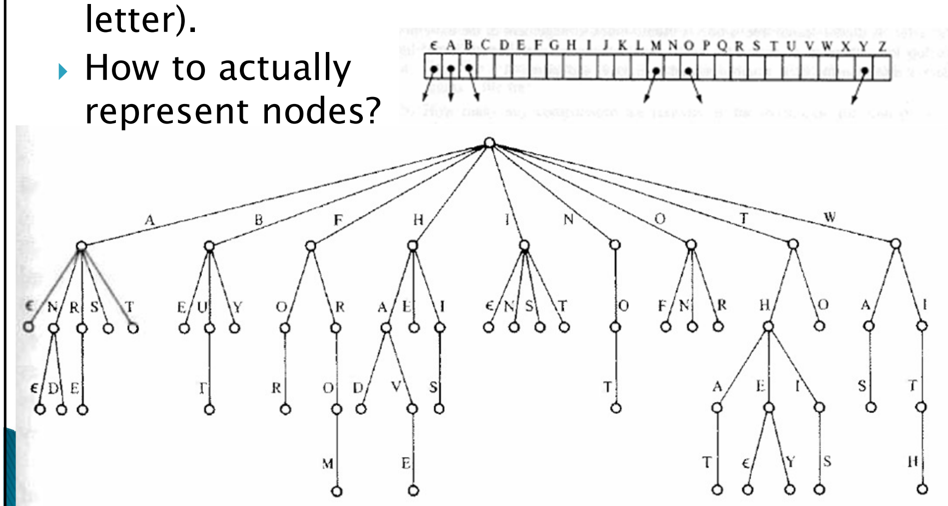
- A useful representation for common sub-expressions: : a + a * (b – c) + (b – c) * d
- A DAG is like a tree with sharing
  - Directed graph
  - No cycles
  - A distinguished root
  - Looks like a tree when doing a
  - traversal, but saves space.



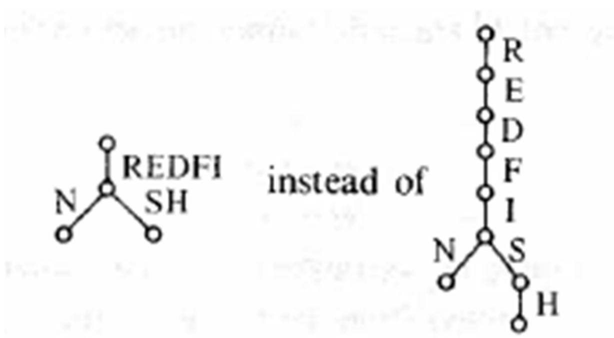
Q4-5

Another approach to search trees

- Digital search tree (trie).
- We store the data digit-by-digit (or letter by letter).
- How to actually represent nodes?

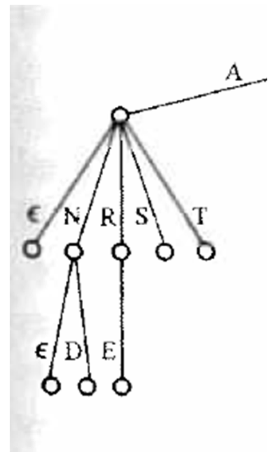


We can collapse single-branch paths to save space



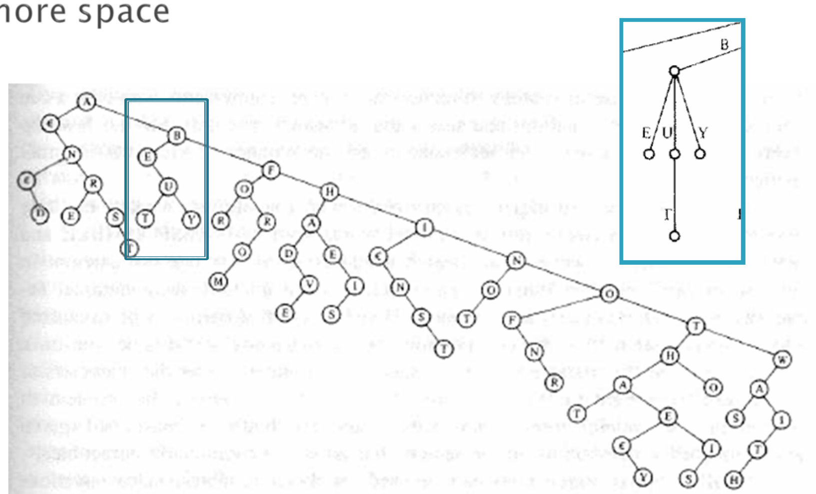
We can share a single static "ϵ-node“ to save space

- The epsilon nodes aren’t null; they just show the end of a word.
- There can still be null pointers at each level where there are missing letters



Representing a Trie as a binary tree saves even more space

Q6



For many more details on Tries, see http://en.wikipedia.org/wiki/Trie

You can trie to create an interesting trie using this applet

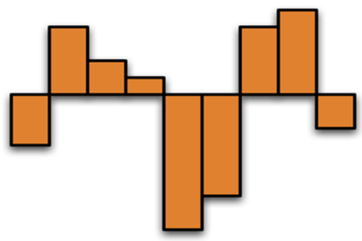
- http://blog.ivank.net/trie-in-as3.html

Introduction to Recurrence Relations

- A technique for analyzing recursive algorithms

Recap: Maximum Contiguous Subsequence Sum problem

Problem definition: Given a non-empty sequence of n (possibly negative) integers $A_1, A_2, \ldots, A_n$, find the maximum consecutive subsequence $S_{i,j} = \sum_{k=i}^{j} A_k$, and the corresponding values of i and j.

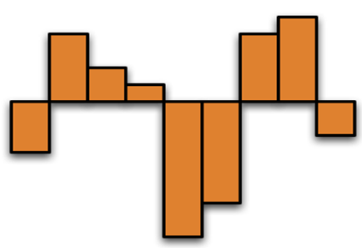


Q7

Divide and Conquer Approach

- Split the sequence in half
- Where can the maximum subsequence appear?

- Three possibilities :
  - entirely in the first half,
  - entirely in the second half, or
  - **begins** in the first half and **ends** in the second half



Overview of algorithm

1. Using recursion, find the maximum sum of **first** half of sequence
2. Using recursion, find the maximum sum of **second** half of sequence
3. Compute the max of all sums that begin in the first half and end in the second half
   - (Use a couple of loops for this)
4. Choose the largest of these three numbers

Q8-9

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
```

So, what's the run-time?

Q10

Analysis?

- Use a Recurrence Relation
  - A function of N, typically written T(N)
  - Gives the run-time as a function of N
  - Two (or more) part definition:
    - Base case, like T(1) = c
    - Recursive case, like T(N) = T(N/2)

So, what's the recurrence relation for the recursive MCSS algorithm?

Q11-12

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
```

What's N in the base case?

Recurrence Relation, Formally

- An equation (or inequality) that relates the n^{th} element of a sequence to certain of its predecessors (recursive case)
- Includes an initial condition (base case)
- **Solution**: A function of n.

- Similar to differential equation, but discrete instead of continuous
- Some solution techniques are similar to diff. eq. solution techniques

Q14-16: Skip 13 for now

Solve Simple Recurrence Relations

- One strategy: **guess and check**

- Examples:
  - $T(0) = 0$, $T(N) = 2 + T(N-1)$
  - $T(0) = 1$, $T(N) = 2\ T(N-1)$
  - $T(0) = T(1) = 1$, $T(N) = T(N-2) + T(N-1)$
  - $T(0) = 1$, $T(N) = N\ T(N-1)$
  - $T(0) = 0$, $T(N) = T(N-1) + N$
  - $T(1) = 1$, $T(N) = 2\ T(N/2) + N$

    (just consider the cases where $N=2^k$)

16

Another Strategy

- **Substitution**
- $T(1) = 1$, $T(N) = 2\ T(N/2) + N$
  (just consider $N=2^k$)
- Suppose we substitute N/2 for N in the recursive equation?
  - We can plug the result into the original equation!

Solution Strategies for Recurrence Relations

- Guess and check
- Substitution
- Telescoping and iteration
- The “master” method

Selection Sort

```
public static void selectionSort(int[] a) {
    //Sorts a non-empty array of integers.

    for (int last = a.length-1; last > 0; last--) {
        // find largest, and exchange with last
        int largest = a[0];
        int largePosition = 0;

        for (int j=1; j<=last; j++)
            if (largest < a[j]) {
                largest = a[j];
                largePosition = j;
        }
        a[largePosition] = a[last];
        a[last] = largest;
    }
}
```

What's N?

A Substitution Example

- Consider:
  - $T(1) = 1$
  - $T(N) = N + T(N/2)$, where $N = 2^k$ for some k
- Substitution:
  - Use recurrence relation repeatedly to expand T() on right-hand side of relation


Editor Trees
Work Time


Slides from Previous terms
Kept in case we want to do those things again some day