

Computer Science II — CSci 1200

Lecture 18

Recursion Revisited

Completing Lecture 17

- Reviewing the erase function and its effect on the tree
- Iterating through the tree:
 - Parent pointer at each node
 - `operator++` requires finding the in-order successor to a node:
 - * If the node has a right child, the successor is the left-most descendent of the right child (which could just be the right child itself).
 - * If the node does not have a right child, the successor is found by walking back up the tree as long as it continues up right branches. As soon as a left branch is traversed, the in-order successor is found.

Recursion

- Reminder: rules for writing recursive functions
 1. Define the recursion — the relationship between solutions to subproblems and solutions to the complete problem.
 2. Handle the base case or cases first in the function
 3. What happens before recursive call or calls
 4. What happens after
 5. Assume the recursion works (if everything else is correct), but make sure the recursion is proceeding toward the base case.
- Examples:
 - Binary search
 - Tree height computation
 - Accumulate function from Lab 9.
- In our earliest examples — factorial, fibonacci and even binary search — the problems are easily solved without recursion. The same is not true for tree height computation, accumulate, and the functions we will consider here.
- Today's problems:
 - Merge sort
 - Non-linear maze search

Merge Sort

- Idea:
 - Split a vector in half
 - Recursively sort each half
 - Merge the two sorted halves into a single sorted vector
- We will work an example first to see how it is done.
- The recursive code, with everything provided except the merging function, is attached to the notes.

Merge — The Idea

We have already considered example problems involving merging of two ordered sequences, but here is a complete discussion of the idea:

- Suppose we have a vector called `values` having two halves that are each already sorted. In particular, the values in subscript ranges `[low..mid]` (the lower interval) and `[mid1..high]+` (the upper interval) are each in increasing order.
- Ask yourself, which value can be first if the two intervals are sorted? Which value could be next?
- In a loop, the merging algorithm repeatedly chooses one value to copy to `scratch`.
 - At each step of the loop, there are only two possibilities: the first uncopied value from the lower interval and the first uncopied value from the upper interval.
- The copying ends when one of the two intervals is exhausted. Then the remainder of the other interval is copied into the scratch vector. Finally, the entire scratch vector is copied back.
- We will complete the code during lecture.

Thinking About Merge Sort

- It exploits the power of recursion! We only need to think about
 - Base case (intervals of size 1)
 - Splitting the vector
 - Merging the results

(Recall the five rules of recursion.)

- We will insert `cout` statements into the algorithm and use this to try to understand what is happening.

Example: Word Search

- Take a look at the following grid of characters.

```
h e a n f u y a a d f j
c r a r n e r a d f a d
c h e n e n s s a r t r
k d f t h i l e e r d r
c h a d u f j a v c z e
d f h o e p r a d l f c
n e i c p e m r t l k f
p a e r m e r o h t r r
d i o f e t a y c r h g
d a l d r u e t r y r t
```

- The usual problem associated with a grid like this is to find words going forward, backward, up, down, or along a diagonal.

- Can you find “computer”?

- A sketch of the solution is as follows:

- The grid of letters is represented as

```
vector< string > grid;
```

Each string represents a row. We can treat this as a two-dimensional array, however.

- A word to be sought, such as “computer”, is read as a string.

- A pair of nested for loops searches the grid for occurrences of the first letter in the string. Call such a location (r, c)

- At each such location, the occurrences of the second letter are sought in the 8 locations surrounding (r, c) .

- At each location where the second letter is found, a search is initiated in the direction indicated.

- * For example, if the second letter is at $(r, c - 1)$, the search for the remaining letters proceeds up the grid.

- At this point, you should not have too much difficulty implementing such an algorithm.

- We are going to spend the rest of today’s lecture on a different, but somewhat harder problem.

Nonlinear Word Search

- Here is the same grid of letters as above.

```
h e a n f u y a a d f j
c r a r n e r a d f a d
c h e n e n s s a r t r
k d f t h i l e e r d r
c h a d u f j a v c z e
d f h o e p r a d l f c
n e i c p e m r t l k f
p a e r m e r o h t r r
d i o f e t a y c r h g
d a l d r u e t r y r t
```

- What happens when we no longer require the locations to be along the same row, column or diagonal of the grid, but instead allow the locations to snake through the grid? The only requirements are that
 1. the locations of adjacent letters are connected along the same row, column or diagonal, and
 2. a location can not be used more than once in each word
- Can you find `rensselaer`? It is there. How about `temperature`? Close, but nope!
- The implementation of this is very similar to the implementation described above until after the first letter of a word is found.
- We will look at the code during lecture, and then consider how to write the recursive function.

Summary of Idea for Recursion

- Recursion starts at each location where the first letter is found
- Each recursive call attempts to find the next letter by searching around the current position. When it is found, a recursive call is made.
- The current path is maintained at all steps of the recursion.
- The “base case” occurs when the path is full **or** all positions around the current position have been tried.

We will complete the implementation during lecture.