Disjoint Sets

2-28-2006

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the test?
- Do you have any questions about the assignment?

Disjoint Sets

- Our data structure for today is that of a disjoint set. The idea is that we have different elements and each element belongs to one set. We want to be able to efficiently determine if two elements are in the same set and also be able to merge sets together quickly. Each set is specified by a representative element.
- Typically we start off with all of our elements so each has its own set and perform a combination of unions and queries of sets from there.
- We will say that there are n elements total and the number of operations we do is m. Since we have to make all the individual sets to start with, $m \geq n$.

Using Linked Lists

- One way to implement this is with linked lists. Each set is a singly linked list where each node points back to the first element and that is the representative element.
- Making a list is O(1). Finding the representative is O(1). However, doing a union requires walking one of the lists to change the pointers to the new representative elements.
- In a simple implementation this gives $O(n^2)$ behavior for a full build and merge down to one set. That means each operation takes O(n) amortized time.

Improving the Linked Lists

- This can be improved if we make a slight alteration to the code. Make each list/set maintain how many elements are in it. Then when we do a union we reset the pointers on the shorter list.
- This improves the time for all m operations to O(m+n log n).

Using a Forest

- Instead of using linked lists, we can use trees. In this case, each node represents an individual element, and the nodes have parent pointers. The root node is the representative element.
- CLR has the root node point to itself though it I haven't seen a reason you couldn't make it point to null in their implementation.
- A simple implementation of this can be just as bad as the linked lists because while union is now fast, determining what set an element is in requires walking up the tree and the tree can degenerate into a linked list.

Union by Rank

- One step we can take to prevent things from degenerating into a linked list is to have each node keep track of a rank, which is an estimate of the height. When you do a union simply make the tree with the lower rank be the child.
- Rank can be easily maintained because when you union to trees with equal rank you increment the rank of the root by one. Otherwise they never change.
- This addition alone provides an O(m log n) total runtime.

Path Compression

- A second optimization that can be performed is to basically flatten out the tree to two levels every time we find the representative element.
- This can be easily done with a recursive function to find the representative element. This function recurses, then sets the pointer to what the recursion returned.
- This way you do O(log n) work the first time you get the representative elements, but you flatten out the tree so you will do less work the next time.

Order of Disjoint Set Forest

- Combining path compression with union by rank gives an algorithm with a rather interesting rank.
- The order of that data structure is $O(m\ \alpha(n))$ where α is the inverse Ackermann's function. This is a function that grows so incredibly slowly that for all practical applications it is $O(m)$.

Reminders

- We have no class on Thursday. Remember to turn in your test. Since everyone is typing these and this one is fairly simple, go ahead and e-mail me your answers when they are done.