CMPT-225 Jan Manuch

Recommended Labs – Monday, July 31, 2006

The goal of this lab is to get familiar with hash tables.

Task. Modify implementation of the hash table using linear probing, so that it doesn't use the special value "AVAILABLE". Instead, the removal operation should rearrange the contents of the hash table so that it appears that the removed item was never inserted to the table.

Hint. One possible way how to do this is as follows. If we would just write the `null` value into the deleted position, the cluster gets split to 2 clusters resulting in possible failures to find an existing item in the hash table. To prevent this from happening, we have to fill in something to the deleted position. Scan the remaining (second) part of the cluster and find an item with a key which would be hashed to the first part of the cluster or to the deleted position. If such an item does not exist, then it's fine to split the cluster to 2 parts and we can finish. Otherwise, copy the item to the deleted position. The position of the copied item is now emptied, so we have the same problem as if the item in this position would be deleted. However, the cluster is now split to 2 parts such that the second part is smaller than it was originally. That it is, if we continue fixing the cluster recursively, the recursion has to stop (either at some moment, it will be ok to leave the cluster split to two parts or the second cluster will be empty, in both cases we can stop).

Example. Here is an example of the work of the cluster fixing algorithm. Assume that array has size 13 and it contains one cluster at positions 3-10 (the table shows the keys of the items; the empty positions contain value `null`):

0	1	2	3	4	5	6	7	8	9	10	11	12
			16	3	5	4	7	19	9			

And assume that element with key 5 is removed from the hash table (the position with 'x' contains `null` but marks the position where original cluster was split and need fixing):

0	1	2	3	4	5	6	7	8	9	10	11	12
			16	3	x	4	7	19	9			

We need to either check that it's ok to leave position 5 empty and fill something in there. Start scanning the second part of the cluster (positions 6-9). Item at position 6 (with key 4) is mapped to position 4, so it can be moved to the 'x' position:

0	1	2	3	4	5	6	7	8	9	10	11	12
			16	3	4	x	7	19	9			

Now, we need to fix the position 6. Start scanning the second part of the cluster (positions 7-9). Item at position 7 is fine, as it's mapped to the second part of the cluster. Item at position 8 (with key 19), would be mapped to position 6, so we can move it there:

0	1	2	3	4	5	6	7	8	9	10	11	12
			16	3	4	19	7	x	9			

Now, we need to fix the position 8. The second part of the cluster contains only one position: 9. Since the item there has key 9, it would be mapped to the second part, so we can stop fixing the cluster and leave it split to 2 parts:

0	1	2	3	4	5	6	7	8	9	10	11	12
			16	3	4	19	7		9			

Note that if the item at position 9 would have key 8 instead of 9, we would have to move it to position 8 and then the second part of the cluster would be empty, so we would stop fixing as well.