

1 Introduction & Instructions

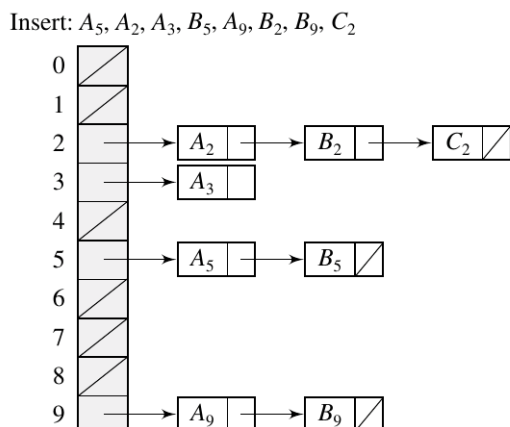
This lab is another representation conversion exercise. You will be taking the hash table code from the class example and converting the open addressing format it uses to a chaining format.

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab07, put each programming exercise into its own subdirectory of this directory, zip the entire Lab07 directory up into the file Lab07.zip, and then submit this zip file to Lab #7.

Make sure that you include a makefile, check the contents of the zip file before uploading it, and that the file was submitted correctly to MyClasses.

2 Exercise

Recall from the class discussion that chaining is where the array was an array of pointers that point to a linked list of all the data items that hashed to the same value.



So A_2 , B_2 , and C_2 all hashed to 2. These linked lists are not ordered (although they could be) so the values of A_2 , B_2 , and C_2 are arbitrary except for the fact that they all hashed to 2. Although we could use a linked list for this structure it would be easier to use a vector of vectors, that is our old friend the ListOfLists structure. What would come in handy with using this is that if our table (`tab`) was a ListOfLists then `tab[2]` is a vector of values that all hashed to 2. Since it is a vector, we can use all of our nifty functions that are supported by vectors and as a result the code to do everything we need in a hash table is reduced to just a few well-chosen lines.

1. Start with the `HashTable.h` file from the example code, the `ListOfLists.h` file from the graph examples and the test program `HashTableExample.cpp` from the hash table example code.
2. The `ListOfLists.h` file will need no alterations.
3. In the `HashTable.h` file,
 - (a) Include the `ListOfLists.h` file.
 - (b) Change the table structure `tab` to a `ListOfLists`.
 - (c) Keep the function pointer to the hash function.
 - (d) Because of the change in storage method and mode of the hash table, you will not need to store the size, empty, and removed member variables. So delete them.
 - (e) You will also not need the probe function so you can delete its prototype and implementation.

- (f) Keep all of the public functions as they are with the exception of the constructor. Here we do not need to have parameters for the flags for the empty position or the removed positions anymore.
- (g) Revise all of the functions in the class accordingly. That is, they should do exactly what the original functions did but with the chaining structure instead of the open addressing. The one exception to this is in the printing. In the print function, the function should display the linked list for each hash table entry. For example,

```
0:
1: 11111
2:
3: 437543 1103
4: 3284 234
5:
6:
7:
8:
9:
```

- (h) There is only one change that needs to be done to the HashTableExample.cpp file. The declaration of the hash table will now be `HashTable<int> table(10, hf)` since we do not need flags for empty and removed locations. This is the only alteration you will need for the main. The output of the test program should now be the following.

```
0:
1: 11111
2:
3: 437543 1103
4: 3284 234
5:
6:
7:
8:
9:

1
0
1
0

0

0:
1: 11111
2:
3: 437543
4: 3284 234
5:
6:
7:
8:
9:

0:
1: 437543 3284
2: 11111
```

3: 234

4:

5:

6:

1

0

0

A couple of quick notes here. Feel free to use the algorithm library if you wish. There were some functions there that made some of this reimplementation easier. If you use a function in the algorithm library that has the same name as a member function (such as `find`) remember to scope it to the `std` namespace to use the algorithm library version. In all, your `HashTable.h` file will probably be about 50 lines of code shorter, give or take.