CSE 241 Assigment #7 - Skip List Spring 2014

Due date: May. 23 20:00

Skip List

Write SkipList class that stores arbitrary objects via class templates. For this assignment, a static library, which contains correct working version of each method to allow for testing, has been provided. Any unimplemented methods in SkipList.cpp will use the corresponding method from the library, thus you can implement the methods in any order. Make sure to test each method you implement individually against the library for proper operation.

SkipList Class Declaration

Provided source files are available in /home/cse241/assign7. You are not allowed to make changes to any other file than the provided SkipList.cpp.

SkipList.h

class SkipList

```
template <class T>
class SkipList
{
  public:
    SkipList();
    ~SkipList();

    // Public interface methods
    bool find(const T & x) const;
    void insert(const T & x);
    void remove(const T & x);
    bool isEmpty() const;
    void makeEmpty();
    void printList();

  private:
    // Head array
    Node<T> *head;
    static const int maxHeight = 5;

    // utility methods
    int randomLevel();
    double getRandomNumber();
};
```

Constructor/Destructor

Since the list will grow dynamically as needed, the only purpose of the constructor is to create and initialize the sentinel node.

Tasks

1. Add code to SkipList() (in SkipList.cpp) to dynamically allocate head as a Node. The next array should be made maxHeight size (which can be done when the Node is created). Do not forget to set the height field appropriately. For this node, the height will represent the current highest level in the skiplist and NOT the maximum height of the skiplist (which is stored in the static member variable maxHeight).
2. Add code to SkipList() (in SkipList.cpp) to free all Node's in the list and then deallocate head.

Find()

The advantage of a skip list is the efficiency of the search operation. This is accomplished by the multi-level next pointer array and the search process which has three cases starting at the highest level of the head pointer:

1. If the value in the next node is the desired value then the value is found.
2. If the value in the next node is greater than the desired value, then drop a level in the current node and repeat the search process.
3. If the value in the next node is less than the desired value, then move the current node to the next node and repeat the search process.

Tasks

1. Add a method named find() that returns a bool (do not forget to qualify it with the class name) that takes a single const reference to a T object parameter and determines if the value is in the list.

   Pseudocode for the Skip List find algorithm is as follows:

SkipList Search

```
Search(list, searchKey)
    x := list->header
    // loop invariant: x->key < searchKey
    for i := list->level downto 1 do
        while x->forward[i]->key < searchKey do
            x := x->forward[i]
        end while
        // x->key < searchKey <= x->forward[1]->key
    end for
    x := x->forward[1]
    if x->key = searchKey then
        return x->value
    else
        return failure
    end if
```

Insert()

Since a skip list maintains elements in sorted order, to insert an element requires determining where the element belongs in the list, i.e. searching for the correct location. To accomplish an insert, we must first determine what level the new node will be inserted at up to one level greater than the current max level of the skip list. Then a similar procedure as search is performed, but since the data structure is implemented with a singly linked list, a local "array of pointers" is needed to ensure all links are updated correctly when the new node is inserted.

Tasks

1. Add a void method named insert() that takes a parameter of type T and inserts a Node containing the data at the appropriate place in the list. You will need to use a local update array of pointers during the search that remembers the current node when the process drops levels.

   Pseudocode for the Skip List insertion algorithm is as follows:

SkipList Insert

```
Insert(list, searchKey, newValue)
    local update[1..MaxLevel]
    x := list->header

    for i := list->level downto 1 do
        while x->forward[i]->key < searchKey do
            x := x->[i]
        end while
        //post condition: x->key < searchKey <= x->forward[i]->key
```

```
        update[i] := x
    end for

    x := x->forward[1]

    if x->key = searchKey then
        x->value := newValue
    else
        lvl := randomLevel()
        if lvl > list->level then
            for i := list->level + 1 to lvl do
                update[i] := list->header
            end for
            list->level := lvl
        end if
        x := makeNode(lvl, searchKey, value)
        for i := 1 to level do
            x->forward[i] := update[i]->forward[i]
            update[i]->forward[i] := x
        end for
    end if
```

2. Add a method named randomLevel() that takes no parameters and returns an int between 1 and the current highest level of the skip list plus 1 (and no greater than the maxHeight of the skip list). Use the provided getRandomNumber() function to generate (deterministic) random numbers and a threshold of 0.5 for continuing to increment the level. Use the following pseudo code for this method.

Deterministic Random Number Generator

```
randomLevel()
  lvl := 1
  // getRandomNumber() that returns a random value in [0...1)
  while getRandomNumber() < 0.5 and lvl < MaxLevel do
      lvl := lvl + 1
  return lvl
```

Remove()

This operation will remove a node from the list (if it exists). Like the insert process, a local array of pointers will need to be created to appropriately update the other nodes in the list when the desired one is deleted. Also if the node is the only one currently at the highest level, then the highest level of the skip list should correspondingly be decremented.

Tasks

1. Add a void method named remove() that takes a const reference to a T object parameter and removes the node that contains the given value (or does nothing if the value does not exist in the list). You will need to use a local update array of pointers during the search that remembers the current node when the process drops levels.

   Pseudocode for the Skip List deletion algorithm is as follows:

SkipList Delete

```
Delete(list, searchKey)
    local update[1..MaxLevel]
    x := list->header
    for i := list->level downto 1 do
        while x->forward[i]->key < searchKey do
            x := x->forward[i]
        end while
        update[i] := x
```

```
    end for
    x := x->forward[1]
    if x->key = searchKey then
        for i := 1 to list->level do
            if update[i]->forward[i] != x then
                break
            end if
            update[i]->forward[i] := x->forward[i]
        end for
        free(x)
        while list->level > 1 and list->header->forward[list->level] = NIL do
            list->level := list->level - 1
        end while
    end if
```

IsEmpty()

A private method which simply returns a boolean indicating whether or not the current list contains any valid, i.e. non-head, nodes.

Tasks

1. Add a method named isEmpty() (do not forget to qualify it with the class name) that takes no parameters and returns a bool indicating true if the list contains no non-head nodes, i.e. when the list is empty.

MakeEmpty()

This method should deallocate all the nodes in the list and reallocate a new head node.

Tasks

Add a void method named makeEmpty() that takes no parameters. It should traverse the list removing each non-head node individually and then reallocate a new head node. Hint: Level 0 is a continuous linked list with all the nodes (and thus the entire pointer array for a particular node can be deallocated once the level 0 next pointer is placed in a temp pointer variable.)

How to Test

Once you have completed implementing any of the above methods (the remaining unimplemented ones will be drawn from the static library), type "make". To run the program, type "./SkipList.x". The console window should generate output similar to:

```
Output
Empty List

List Insert Tests
Node 1 height 3: Level 2 -> NULL,Level 1 -> 4,Level 0 -> 2,
Node 2 height 1: Level 0 -> 4,
Node 4 height 2: Level 1 -> NULL,Level 0 -> NULL,


List Find Tests

List Remove Tests
Node 1 height 3: Level 2 -> NULL,Level 1 -> 4,Level 0 -> 4,
Node 4 height 2: Level 1 -> NULL,Level 0 -> NULL,


List Empty Tests
```

```
Empty List
```

If you are done with implementing any method, modify Flags.h and run to test your code. For example, if you are going writing insert() function, comment "#define ALL 1" and uncomment "#define INSERT 1" in Flags.h, and type "make". If your implementation is correct, you should see output similar to the above output. For the functions that you didn't uncomment the define directives, the functions provided in the static library will be used.

Submitting Your Code

Be sure to remove all DEBUG output from your implemention before submission. You should submit SkipList.cpp file only. Please do not implement your code in other provided files. Turn in your project using the "oopsubmit" command as follows:

```
$ oopsubmit assign7 SkipList.cpp
```

If you do not follow this submission guideline, you will lose 10 points out of 100.

You should also submit a hard copy of your code to TA. Your report must have a cover page with your student ID and name. In the report, your code must be well commented to explain your class.