Bulent Yesilyurt

Professor Smallberg

CS32

31 January 2023

Project 2 Report

My implementation used a doubly linked list with a tail node. It was not linked circularly. Each node is not in any particular order as they are simply inserted to the end as the list gets added on to in the insert function. Following the parameters of a doubly linked list, each node has a next pointer, a previous pointer, and a value. I declared the node constructor as a private member of the Set class. I also had a size tracker for the size of the list and had head and tail nodes as private Set class members.

**For Non-Trivial Algorithms:**

**Destructor:**

Initialize new node and point it to head

If the size is 1:

Delete head

Else If size is not 0

Loop through every node

Delete the previous node

Delete tail

Set size to 0

**Copy Constructor:**

If list is empty:

Set head and tail to nullptr

If list has one element:

Set this size equal to the other set’s size

Make this head a new node and initialize value

Set this head’s next and previous pointers to nullptr

Set tail equal to head

Else:

Initialize a Set loop variable and a new Set variable

Set sizes equal

Make this head equal a new node and initialize values and pointers

Set new Set variable equal to head

Starting from other’s head’s next node, loop through end of list:

Declare a new temporary node and initialize values and pointers

Increment the new set variable

End loop

Initialize tail and make its next pointer equal nullptr

**Assignment Operator:**

If the right side equals the left side already:

Return a pointer to the left

Declare and initialize a new Set variable using the copy constructor

Swap elements of this Set and new Set

Return pointer to the left

**insert:**

Check if value is contained in the list already

If list is empty:

Declare a new node and initialize value and pointers

Make tail’s next equal nullptr and head’s previous pointer equal nullptr

Increment size;

Else:

Declare a new node and initialize values

Set tail equal to the new node

Set tail’s next pointer to nullptr

Increment size

Return true

**erase:**

Declare a kill pointer and initialize it to head

Check if list is empty

if list has one element:

Delete head and decrement size

If the value of head equals the value to be killed

Make head point to the next element and delete kill

Decrement size

Loop through to end of list:

If the value of the node equals the value to be deleted and it's the last element:

Make tail point to the previous element and set previous node next to null

Decrement size

If the value is not the last in the list and values equal:

Change previous and next node’s pointers to point to each other

Delete kill and decrement size

End loop

Return false

**get:**

Make a counter and initialize a current and traversing Set variable to point to head

Check if size is one

Else:

Outer Loop through end of the list with current variable

Inner Loop with tail pointing to the last element to the start of the list

if current value greater than traversing

Increment counter

if *i* is in the bounds and if count equals *i*

Set value equal to the current node’s value

Exit function

Set count to 0

Increment the current variable

End inner loop

End outer loop

Return false

**swap:**

Swap sizes with temporary int variable

Swap head pointers with temporary Set variable

Swap tail pointers with another temporary Set variable

**unite:**

Construct a new temporary set variable

Loop through set1:

Get each value

Insert it into the temporary set variable

End loop

Loop through set2:

Get each value

Insert it into the temporary set variable

End loop

Set result equal to the temporary variable

**butNot:**

Construct a new temporary set variable

Loop through set1:

Get each value

Insert it into the temporary set variable

End loop

Loop through set2:

Get each value

If the value is already in the temporary set:

Erase that value from the set

End loop

Set result equal to the temporary variable

**contains:**

Make a temporary set variable and set to head

If size of list is 1:

Value is head’s value

Return true;

Loop through end of the list

if value is found

Set value to node’s value

Return true

End loop

Return false

**Test Conditions:**

**Using ItemType as unsigned long(For easier testing)**

Set s1;

Set s2;

Set s3;

unite (s1,s2,s3); *//Unite with empty sets*

assert(s3.size() == 0); *//Check if constructor works*

assert(s1.empty()); *//Check if empty works*

assert(!s1.erase(100)); *// Nothing to remove so returns false*

assert(s1.insert(1)); *//Inserts into an empty list*

assert(s1.size() == 1); *//Checks size*

assert(s1.erase(1)); *//checks if erase works for one element*

assert(s1.size() == 0); *//Checks size*

for (int i = 0; i < 20; i++)

assert(s1.insert(i)); *//check if insert works correctly*

for (int i = 0; i < 30; i++)

assert(s2.insert(i)); *//Check if Insert works correctly*

for (int i = 0; i < 40; i++)

assert(s3.insert(i)); *//Check if insert works correctly*

unite(s1,s2,s3);

for (int i = 0; i < 30; i++){

ItemType x = 0;

s3.get(i,x);

assert(x == i); *//Check if unite works properly;*

}

Checking Unite

unite (s1, s1, s3); *//Check if unite works with s1 == s2*

for (int i = 0; i < 20; i++){

ItemType x = 0;

s3.get(i,x);

assert(x == i);

}

unite (s1, s2, s2); *//Check if unite works with s2 == s3*

for (int i = 0; i < 30; i++){

ItemType x = 0;

s2.get(i,x);

assert(x == i);

}

unite (s1, s2, s1); *//check if unite works with s1 == s3*

for (int i = 0; i < 20; i++){

ItemType x = 0;

s1.get(i,x);

assert(x == i);

}

//Checking contains and erase

for (int i = 30; i < 40; i++)

assert(!s3.contains(i)); *//Check if last elements of s3 were erased*

assert(s3.erase(0) && !s3.contains(0)); *//Check if erase works for first item in list*

assert(!s3.erase(39) && !s3.contains(39)); *//Check if erase works for last item*

assert(s3.erase(11) && !s3.contains(11)); *//Check if erase works for middle item*

//Test Copy constructor, assignment operator:

Set s5 = s1;

Set s4(s2);

for (int i = 0; i < s4.size(); i++){

ItemType x = 0;

ItemType y = 0;

s4.get(i,x);

s2.get(i,y);

assert(x == y); *//Check if s4 = s2*

ItemType z = 0;

ItemType a = 0;

s5.get(i,z);

s1.get(i,a);

assert(z == a); *//Check if s5 equals s1*

}

//Testing get:

Set s6;

s6.insert(100);

s6.insert(200);

ItemType b = 0;

s6.get(0,b);

assert(b == 100); *//Check if get works for one element*

s6.get(1,b);

assert(b == 200); *//Check if get works for last element*

Set s7;

assert(!s7.erase(0)); *//check if erase works when value isn't in the list*

assert(s6.get(2,b) == false); *//check if get works for an out of range index*

//checking butNot

Set s8;

Set s9;

Set s10;

for (int i = 0; i < 10; i++)

s8.insert(i);

for (int i = 0; i < 20; i++)

s9.insert(i);

for (int i = 0; i < 30; i++)

s10.insert(i);

butNot(s8,s8,s10);

s10.dump(); *//Checks if s1 and s2 are equal, if butNot works properly*

butNot(s8,s9 ,s10);

assert(s10.size() == 0); *//Checks if butNot works if s1’s values are in s2*

butNot(s9,s8,s10);

assert(s10.contains(11)); *//Checks if butNot works normally*

butNot(s10,s9, s10); *//Checking if s1 == s3*

assert(s10.size() == 0); *//Checks if s1 = butNot;*

assert(s10.empty()); *//Asserts that empty works as intended*