Report

CS32 Project 2

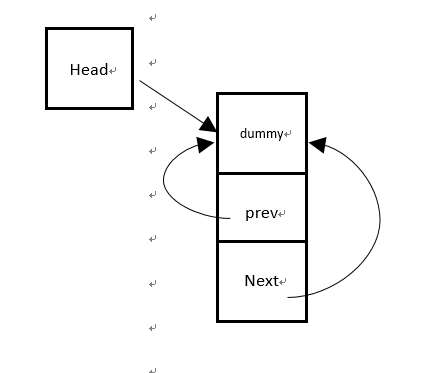
Xiaoshu Luo

904971914

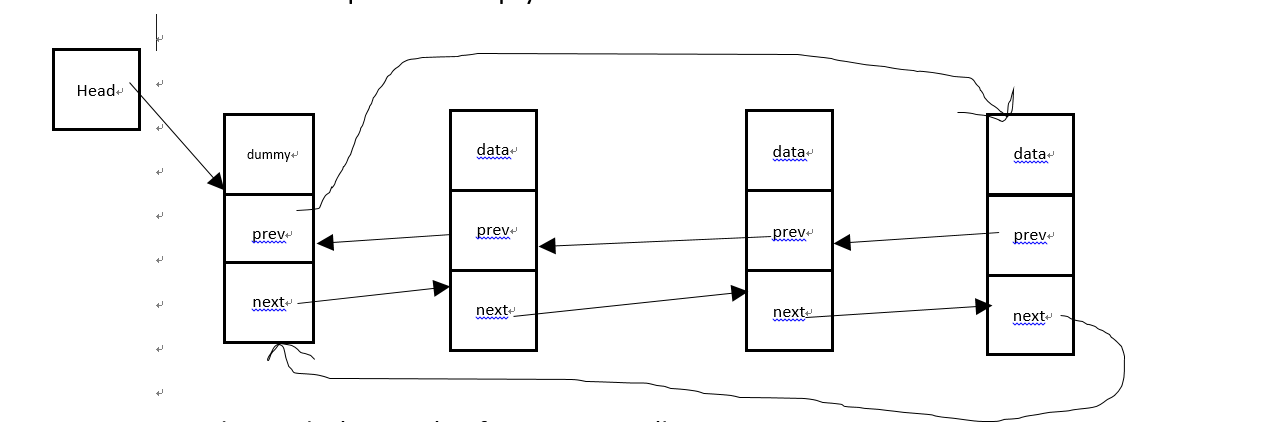
Design:

In this project I used a circular doubly linked list to store data. Every piece of data is stored in a node which is made up of three parts: the data, a next pointer, and a previous pointer. The list starts with a head pointer, which points to a dummy node created during initialization. The dummy node has no data in it. For each time when a new piece of data is added, a new node is created to hold it, and it is inserted after the last node in the list.

Here is an example of an empty list:



Here is a typical example of a non-empty list:



Pseudocode:

1. Constructor:

*Create a dummy node;*

1. Copy constructor:

*Create a dummy node;*

*Repeatedly*

*Insert every data in the target set;*

1. Operator =:

*If the set is not the same as the target set*

*Create a copy of the target;*

*Swap the set with the copy;*

*Return the pointer to the set;*

1. Destructor:

*Starting from the head;*

*Repeatedly:*

*Move to the next node;*

*Delete the last node;*

*Delete the dummy node;*

1. Insert:

*Check if the value already exists in the list;*

*If not, create a new node;*

*Modify the “old” last node and the dummy node;*

*Implement the new node;*

1. Check if the value exists in the list;

*If so, modify the two adjacent nodes;*

*Delete the node;*

1. Get:

*Beginning from the first node after the dummy node;*

*Repeatedly:*

*Record the number of data in the list is larger than the current value ;*

*If the number matches with I, copy this value and return;*

1. Swap:

*Swap the head pointer;*

*Swap the number of data in the list;*

1. Unite:

*Create a copy of s1;*

*Swap the result with s1;*

*Repeatedly:*

*Check if each element in s2 already exists in s1;*

*If not, insert it into result;*

1. Subtract:

*Create a copy of s1;*

*Swap the result with s1;*

*Repeatedly:*

*Check if each element in result already exists in s2;*

*If so, erase it from the result;*

Test:

Set uls;//test the default constructor, uls shoud be an empty set;

assert(uls.empty());//test empty;

assert(uls.size() == 0);//test size;

assert(!uls.contains(10));//test contains;

assert(!uls.erase(5));//test erase;

assert(uls.insert(10));//test insert;

assert(uls.insert(20));//test insert;

assert(uls.insert(30));//test insert;

assert(uls.insert(1));//test insert;

assert(uls.insert(2));//test insert;

assert(uls.insert(3));//test insert;

assert(!uls.insert(20));//test insert;

assert(uls.size() == 6);//test size;

assert(!uls.empty());//test empty;

assert(uls.contains(20));//test contains;

assert(!uls.contains(4));//test contains;

assert(uls.erase(1));//test erase;

assert(uls.erase(10));//test erase;

assert(!uls.erase(1));//test erase;

assert(uls.size() == 4);//test size;

assert(!uls.empty());//test empty;

assert(!uls.contains(1));//test contains;

assert(uls.contains(2));//test contains;

ItemType x = 4;

assert(uls.get(0, x) && x == 2);//test get;

assert(uls.get(1, x) && x == 3);//test get;

x = 4;

assert(!uls.get(4, x)&&x==4);//test get;

assert(!uls.get(5, x) && x == 4);//test get;

assert(uls.get(3, x) && x == 30);//test get;

Set a(uls);//test copy contructor. now a should have the same data as uls;

assert(a.contains(2) && a.contains(3) && a.contains(20) && a.contains(30));//test copy constructor;

assert(!a.contains(1) && !a.contains(4) && !a.contains(5));//test copy constructor;

assert(a.size() == 4 && !a.empty());//test copy constructor;

Set b = a;//test operator = for an empty set. now b should have the same data as a;

assert(b.contains(2) && b.contains(3) && b.contains(20) && b.contains(30));//test operator =;

assert(!b.contains(1) && !b.contains(4) && !b.contains(5));//test operator =;

assert(b.size() == 4 && !b.empty());//test operator =;

Set c;

assert(c.insert(5) && c.insert(6) && c.insert(7) && c.insert(8) && c.insert(9));//c is now not empty;

c = a;//test operator = for a non-empty set;now c should have the same data as a;

assert(c.contains(2) && c.contains(3) && c.contains(20) && c.contains(30));//test operator =;

assert(!c.contains(5) && !c.contains(6) && !c.contains(7)&&!c.contains(8)&&!c.contains(9));//test operator =;

assert(c.size() == 4 && !c.empty());//test operator =;

Set d;

assert(d.insert(5) && d.insert(6) && d.insert(7) && d.insert(8) && d.insert(9));

a.swap(d);//test swap for a non empty list;

assert(a.size() == 5 && !a.empty());//test swap;

assert(a.contains(5) && a.contains(6) && a.contains(7) && a.contains(8) && a.contains(9));//test swap;

assert(d.contains(2) && d.contains(3) && d.contains(20) && d.contains(30));//test swap;

assert(!a.contains(2) && !a.contains(3) && !a.contains(20) && !a.contains(30));//test swap;

assert(!d.contains(5) && !d.contains(6) && !d.contains(7) && !d.contains(8) && !d.contains(9));//test swap;

Set e;

e.swap(b);//test swap for an empty list;

assert(e.contains(2) && e.contains(3) && e.contains(20) && e.contains(30));//test swap;

assert(b.empty()&&b.size()==0&&e.size()==4);//test swap;

Set X;

Set Y;

Set result1;

unite(X, Y, result1);//test unite when s1 s2, and result are empty;

assert(result1.empty());

assert(result1.insert(1) && result1.insert(2));//test unite when s1 and s2 are empty and result isn't;

unite(X, Y, result1);

assert(result1.empty() && !result1.contains(1) && !result1.contains(2));

assert(X.insert(3) && X.insert(4));//test unite when s2 is empty and s1 isn't;

unite(X, Y, result1);

assert(result1.size() == 2 && result1.contains(3) && result1.contains(4));

assert(X.erase(3) && X.erase(4));

assert(Y.insert(5) && Y.insert(6) && Y.insert(7));//test unite when s1 is empty and s2 isn't;

unite(X,Y, result1);

assert(result1.contains(5) && result1.contains(6) && result1.contains(7) && result1.size() == 3);

assert(X.insert(8) && X.insert(9) && X.insert(10) && X.insert(11));//test unite when result, s1 and s2 are not empty and they have no intersection;

unite(X, Y, result1);

assert(result1.size() == 7 && result1.contains(5) && result1.contains(6) && result1.contains(7));

assert(result1.contains(8) && result1.contains(9) && result1.contains(10) && result1.contains(11));

assert(X.insert(5) && X.insert(6)&&X.insert(12));//test unite when result, s1 and s2 are not empty and s1 and s2 have some intersection;

assert(Y.insert(8) && Y.insert(11)&&Y.insert(13));

unite(X, Y, result1);

assert(result1.size() == 9 && result1.contains(5) && result1.contains(6) && result1.contains(7) && result1.contains(8));

assert(result1.contains(9) && result1.contains(10) && result1.contains(11) && result1.contains(12) && result1.contains(13));

result1 = X;//test unite when result is the same as s1;

unite(X, Y, result1);

assert(result1.size() == 9 && result1.contains(5) && result1.contains(6) && result1.contains(7) && result1.contains(8));

assert(result1.contains(9) && result1.contains(10) && result1.contains(11) && result1.contains(12) && result1.contains(13));

result1 = Y;//test unite when result is the same as s2;

unite(X, Y, result1);

assert(result1.size() == 9 && result1.contains(5) && result1.contains(6) && result1.contains(7) && result1.contains(8));

assert(result1.contains(9) && result1.contains(10) && result1.contains(11) && result1.contains(12) && result1.contains(13));

X = Y;//test unite when s1, s2 and result are the same;

unite(X, Y, result1);

assert(result1.size() == 6 && result1.contains(5) && result1.contains(6) && result1.contains(7) && result1.contains(8));

assert(result1.contains(11) && result1.contains(13)&&!result1.contains(4));

Set W;

Set Z;

Set result2;

subtract(W, Z, result2);//test subtract when s1, s2 and result are empty;

assert(result2.empty());

assert(result2.insert(5));//test subtract when s1, s2 are empty and result is not.

subtract(W, Z, result2);

assert(result2.empty());

assert(W.insert(1) && W.insert(3) && W.insert(5));//test subtract when s1 is not empty and s2 is.

subtract(W, Z, result2);

assert(result2.size() == 3 && result2.contains(1) && result2.contains(3)&&result2.contains(5));

assert(W.erase(1) && W.erase(3) && W.erase(5));//test subtract when s2 is not empty and s1 is.

assert(Z.insert(2) && Z.insert(4) && Z.insert(6));

subtract(W, Z, result2);

assert(result2.empty() && !result2.contains(2) && !result2.contains(4) && !result2.contains(6));

assert(W.insert(1) && W.insert(3) && W.insert(5));//test subtract when s1 and s2 are not empty and they have no intersection.

subtract(W, Z, result2);

assert(result2.size() == 3 && result2.contains(1) && result2.contains(3) && result2.contains(5));

assert(Z.insert(1) && Z.insert(5));//test subtract when s1 and s2 are not empty and they have intersection.

subtract(W, Z, result2);

assert(result2.size() == 1 && result2.contains(3) && !result2.contains(1) && !result2.contains(5));

assert(Z.insert(3));//test subtract when s1 is a subset of s2;

subtract(W, Z, result2);

assert(result2.empty());

W = Z;//test subtract when s1 is the same set as s2;

subtract(W, Z, result2);

assert(result2.empty());

assert(W.erase(2) && W.erase(4) && W.erase(6));//test subtract when s1 is the same as result;

assert(Z.erase(1) && Z.erase(3) && Z.erase(5));

result2 = W;

subtract(W, Z, result2);

assert(result2.size() == 3 && result2.contains(1) && result2.contains(3) && result2.contains(5));

result2 = Z;//test subtract when s2 is the same as result;

subtract(W, Z, result2);

assert(result2.size() == 3 && result2.contains(1) && result2.contains(3) && result2.contains(5));

W = Z;//test subtract when s1, s2 and result are all the same;

subtract(W, Z, result2);

assert(result2.empty());