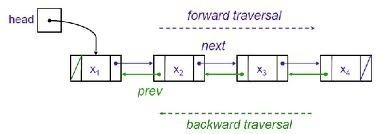
For this implementation of Sequence, a doubly-linked list was used. The doubly-linked list consists of a head pointer and Node structs. Each Node consists of a value, a pointer to the next item, and a pointer to the previous item. The list is considered empty when head points to the null pointer.

Empty:

head

Typical:



**Pseudocode:**

**Sequence::remove**

Loop through sequence

When a matching value is found

Erase the position

Increment the count

Reset the loop

Increment the position

Return the count

**Sequence::subsequence**

Repeatedly:

Get the first element of seq2

If the element is not found in seq1

Return -1

Otherwise

Iterate through and compare consecutive members of seq1

If anything doesn’t match

Erase the false subsequence start originally found in seq1

Increment the delete count

Remember there was a non-match

Go back to the outermost loop

If only matches were found

Return where the subsequence start was found

**Sequence::interleave**

Make copies of seq1 and seq2

Remove everything from result

Find which sequence is smaller and remember its size

Iterate through each sequence until the small one ends

Insert values from seq1 and seq2 into the end of result

Insert the remainder of the larger sequence to the end of result

**Test cases (given as code):**

Using ItemType unsigned long:

Sequence empty;

Sequence s;

s.insert(0, 63);

s.insert(1, 17);

s.insert(2, 63);

s.insert(3, 42);

s.insert(4, 17);

s.insert(5, 63);

s.insert(6, 17);

s.insert(7, 29);

s.insert(8, 8);

s.insert(9, 32);

Sequence s1;

s1.insert(0, 63);

s1.insert(1, 17);

s1.insert(2, 29);

assert(subsequence(s, empty) == -1); //tests that when the second argument is empty, returns -1

assert(subsequence(s1, s) == -1); //tests that when the second argument > the first argument, reutrns -1

assert(subsequence(s, s1)==5); //tests that sequence is appropriately found

Sequence s2;

s2.insert(0, 63);

s2.insert(1, 17);

assert(subsequence(s, s2) == 0); //tests that sequence of different length is appropriately found

s1.set(2, 63);

assert(subsequence(s, s1) == 0); //tests for repetition of a number in subsequence

s1.set(2, 32);

assert(subsequence(s, s1) == -1); //tests that failure is correctly detected

Sequence s3;

s3.insert(0, 8);

s3.insert(1, 32);

s3.insert(2, 32);

assert(subsequence(s, s3) == -1); //tests for when subsequence would run off of the sequence

Sequence a;

a.insert(0, 30);

a.insert(1, 21);

a.insert(2, 63);

a.insert(3, 42);

a.insert(4, 17);

a.insert(5, 63);

Sequence b;

b.insert(0, 42);

b.insert(1, 63);

b.insert(2, 84);

b.insert(3, 19);

Sequence c;

interleave(a, b, c); //tests when first argument is larger than second

ItemType x = 999;

assert(c.get(0, x) && x == 30);

assert(c.get(1, x) && x == 42);

assert(c.get(2, x) && x == 21);

assert(c.get(3, x) && x == 63);

assert(c.get(4, x) && x == 63);

assert(c.get(5, x) && x == 84);

assert(c.get(6, x) && x == 42);

assert(c.get(7, x) && x == 19);

assert(c.get(8, x) && x == 17);

assert(c.get(9, x) && x == 63);

interleave(b, a, c); //tests when second argument is larger and the case when the last argument contains something already

assert(c.get(0, x) && x == 42);

assert(c.get(1, x) && x == 30);

assert(c.get(2, x) && x == 63);

assert(c.get(3, x) && x == 21);

assert(c.get(4, x) && x == 84);

assert(c.get(5, x) && x == 63);

assert(c.get(6, x) && x == 19);

assert(c.get(7, x) && x == 42);

assert(c.get(8, x) && x == 17);

assert(c.get(9, x) && x == 63);

interleave(empty, a, c); //tests when first argument is empty

assert(c.get(0, x) && x == 30);

assert(c.get(1, x) && x == 21);

assert(c.get(2, x) && x == 63);

assert(c.get(3, x) && x == 42);

assert(c.get(4, x) && x == 17);

assert(c.get(5, x) && x == 63);

interleave(b, empty, c); //tests when second argument is empty

assert(c.get(0, x) && x == 42);

assert(c.get(1, x) && x == 63);

assert(c.get(2, x) && x == 84);

assert(c.get(3, x) && x == 19);

interleave(a, b, b); //tests aliasing with second argument and result

assert(b.get(0, x) && x == 30);

assert(b.get(1, x) && x == 42);

assert(b.get(2, x) && x == 21);

assert(b.get(3, x) && x == 63);

assert(b.get(4, x) && x == 63);

assert(b.get(5, x) && x == 84);

assert(b.get(6, x) && x == 42);

assert(b.get(7, x) && x == 19);

assert(b.get(8, x) && x == 17);

assert(b.get(9, x) && x == 63);

interleave(a, a, a); //tests aliasing with all three arguments

assert(a.get(0, x) && x == 30);

assert(a.get(1, x) && x == 30);

assert(a.get(2, x) && x == 21);

assert(a.get(3, x) && x == 21);

assert(a.get(4, x) && x == 63);

assert(a.get(5, x) && x == 63);

assert(a.get(6, x) && x == 42);

assert(a.get(7, x) && x == 42);

assert(a.get(8, x) && x == 17);

assert(a.get(9, x) && x == 17);

assert(a.get(10, x) && x == 63);

assert(a.get(11, x) && x == 63);

interleave(empty, empty, c); //tests interleave with two empty sequences

assert(c.size() == 0);

Using ItemType std::string:

Sequence b;

assert(b.size() == 0); //tests size for empty sequence

assert(b.insert(0, "a")); //tests inserting to an empty sequence

assert(b.size() == 1); //tests size for nonempty sequence

assert(b.erase(0) && b.size() == 0); //tests erase on single item sequence

string x;

Sequence s;

assert(s.empty()); //tests empty for empty sequence

assert(!s.erase(0)); //tests erase for empty sequence

s.insert("e");

assert(!s.empty()); //tests empty for non-empty sequence

s.insert("b");

s.insert("c");

s.insert("b");

s.insert("a");

assert(s.get(0, x) && x == "a"); //tests one argument insert (and two argument by proxy)

assert(s.get(1, x) && x == "b");

assert(s.get(2, x) && x == "b");

assert(s.get(3, x) && x == "c");

assert(s.get(4, x) && x == "e");

assert(s.remove("b") == 2); //tests remove

assert(s.size() == 3);

assert(s.get(0, x) && x == "a");

assert(s.get(1, x) && x == "c");

assert(s.get(2, x) && x == "e");

assert(s.set(2, "z") && s.get(2, x) && x == "z"); //tests set

Sequence a(s); //tests copy constructor

assert(a.get(0, x) && x == "a");

assert(a.get(1, x) && x == "c");

assert(a.get(2, x) && x == "z");

assert(a.size() == 3);

assert(a.insert(0, "d") && a.size() == 4); //tests inserting at the beginning

assert(a.insert(4, "e") && a.size() == 5); //tests inserting at the end

assert(a.get(0, x) && x == "d");

assert(a.get(4, x) && x == "e");

assert(a.erase(0) && a.size() == 4); //tests erase for beginning, end, and general cases

assert(a.erase(3) && a.size() == 3);

assert(a.erase(1) && a.size() == 2);

assert(a.get(0, x) && x == "a");

assert(a.get(1, x) && x == "z");

b = s; //tests assignment operator

assert(b.get(0, x) && x == "a");

assert(b.get(1, x) && x == "c");

assert(b.get(2, x) && x == "z");

assert(b.insert(2,"a") && b.size() == 4); //tests general insert case

assert(b.get(2, x) && x == "a");

Sequence s1;

s1.insert(0, "paratha");

s1.insert(0, "focaccia");

Sequence s2;

s2.insert(0, "roti");

s1.swap(s2);

assert(s1.size() == 1 && s1.find("roti") == 0 && s2.size() == 2 &&

s2.find("focaccia") == 0 && s2.find("paratha") == 1); //tests swap

Sequence s3;

s3.insert(0, "dosa");

s3.insert(1, "pita");

s3.insert(2, "");

s3.insert(3, "matzo");

assert(s3.find("") == 2);

s3.remove("dosa"); //tests remove

assert(s3.size() == 3 && s3.find("pita") == 0 && s3.find("") == 1 &&

s3.find("matzo") == 2);

assert(s3.remove("") == 1);

assert(s3.remove("matzo") == 1);

assert(s3.size() == 1);

assert(s3.remove("pita") == 1); //tests remove for sequence of size 1

assert(s3.remove("pita") == 0); //tests remove for empty sequence

Sequence s4;

assert(s4.empty());

assert(s4.find("laobing") == -1); //tests find on empty sequence

s4.insert("laobing");

assert(s4.size() == 1 && s4.find("laobing") == 0); //tests find

Two argument insert, get, and find functions are used prolifically, and can be thought of as being tested throughout.