Duncan Di Mauro

UID #805163177

3/01/2022

CS32 Homework 4 – Questions 2, 3e, 4b, 5a, 5b, 6a, and 6b

**2.**

When the insert function is called, it calls the doInsertOrUpdate function. This function uses the “==” operator, which is not defined for the Coord class, so we get an error.

**3e.**

In the execution of test case 3, a value is pushed to the back of the vector. When this happens during iteration, all iterators (except those who point to an element before the point of insertion) are invalidated, so when the iterator tries to go to the end of the list, it becomes invalidated and the program tries to extract a value from a nullptr.

**4b.**

This would not be possible given the constraints in part a, since in order to get the names of the subclasses, we need to call something like c->subclasses()[int i]->name(); where “i” would represent an integer. This could potentially allow us to get the names of all the subclasses, however then we would have no way of printing them properly. By using a two-parameter overload, we can pass in a string called path which we can then use with recursion to print the names in the proper order that we want.

**5a.**

const int N = *some value*;

bool hasContacted[N][N];

...

int numIntermediaries[N][N];

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

{

numIntermediaries[i][i] = -1; // the concept of intermediary

// makes no sense in this case

for (int j = 0; j < N; j++)

{

if (i == j)

continue;

numIntermediaries[i][j] = 0;

for (int k = 0; k < N; k++)

{

if (k == i || k == j)

continue;

if (hasContacted[i][k] && hasContacted[k][j])

numIntermediaries[i][j]++;

}

}

}

The time complexity in the above code would be O(N^3). There are three loops in the above code, each which have N-amount of iterations. The second loop is within the first loop, and the third loop is within the second loop. Therefor, we should multiply the amount of iterations, which is N^3.

**5b.**

const int N = *some value*;

bool hasContacted[N][N];

...

int numIntermediaries[N][N];

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

{

numIntermediaries[i][i] = -1; // the concept of intermediary

// makes no sense in this case

for (int j = 0; j < **i**; j++) **// loop limit is now i, not N**

{

numIntermediaries[i][j] = 0;

for (int k = 0; k < N; k++)

{

if (k == i || k == j)

continue;

if (hasContacted[i][k] && hasContacted[k][j])

numIntermediaries[i][j]++;

}

**numIntermediaries[j][i] = numIntermediaries[i][j];**

}

}

The time complexity in the above code would be O(N^3), since we only care about the high order term. In this version of the code, the 2nd loop has N-1 iterations, while the first and 3rd loop have N iterations. Therefor the total amount of iterations would be N \* (N-1) \* N = N^3 – N^2, and the time complexity would be O(N^3).

**6a.**

void reassign(const Map& m, Map& result)

{

// Guard against the case that result is an alias for m (i.e., that

// result is a reference to the same map that m refers to) by building

// the answer in a local variable res. When done, swap res with result;

// the old value of result (now in res) will be destroyed when res is

// destroyed.

Map res;

if (!m.empty())

{

KeyType prevKey;

ValueType value0;

// Get pair 0, which must succeed since m is not empty

m.get(0, prevKey, value0);

// For each pair i after pair 0, insert into res a pair with

// pair i-1's key and pair i's value. (This loop executes 0 times

// if m has only one pair.)

for (int i = 1; i < m.size(); i++)

{

KeyType k;

ValueType v;

m.get(i, k, v);

res.insert(prevKey, v);

prevKey = k;

}

// Insert a final pair with last pair's key and pair 0's value.

res.insert(prevKey, value0);

}

result.swap(res);

}

The above function’s time complexity is O(N^2). The for loop has N amount of iterations, and within the for loop, the get and insert functions call some other functions which eventually contain the findFirstAtLeast function. This function searches through potentially the entire list, going through N elements at maximum. Therefor, the time complexity would be [N \* (N + N)] + lower order Ns = O(N^2).

**6b.**

void Map::reassign()

{

Node\* p = m\_head->m\_next;

if (p != m\_head)

{

ValueType value0 = p->m\_value;

for ( ; p->m\_next != m\_head; p = p->m\_next)

p->m\_value = p->m\_next->m\_value;

p->m\_value = value0;

}

}

The time complexity of this function is O(N). The for loop has a maximum time complexity of N. This is better than the implementation in part a, which had a time complexity of O(N^2).