Rodger Lee

UID: 204914719

CS32 : Homework 4

P2.

The compilation error occurs when mpi.insert calls doInsertorUpdate, which calls the find function. In the find function the program attempts to use the “!=” operator to compare the KeyType Coord. The Coord class does not have a defined “!=” operator, which creates this error message.

P3.

a)

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 is O(N^3) because the loop with i as the variable runs N times, the loop with j as the variable runs N \* N times, and the loop with k as the variable runs N \* N \* N times.

b)

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 is still O(N^3) because the loop with i as the variable runs N times, the loop with j as the variable runs N \* (N – 1) times, and the loop with k as the variable runs N \* (N – 1) \* N times.

P4.

a)

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()). //**O(1)**

{

KeyType prevKey;

ValueType value0;

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

m.get(0, prevKey, value0); //**O(N)** needs to find value in map

// 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++). //size N

{

KeyType k;

ValueType v;

m.get(i, k, v); //O(N)

res.insert(prevKey, v); //O(N)

prevKey = k;

}

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

res.insert(prevKey, value0); //O(logN)

}

result.swap(res); //O(1)

}

The time complexity is O(N^2) because the loop with i as the variable runs N times, and it inserts something each the time the loop runs. Insert for maps have time complexity of O(N). Running insert N times would produce O(N^2) time complexity, which is larger than N, logN, and 1. All other additional time complexities are nullified due to the magnitude of this complexity.

b)

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 is O(N), the for loop visits each node only once for N nodes.