Bradley Mont

Com Sci 32

Professor Smallberg

Lecture 2, Discussion 2C

UID: 804-993-030

Homework 4

2. It causes a compilation error because when the insert function is called, it calls the doInsertOrUpdate function, which calls the find function, which uses the “!=” operator for the KeyType (a Coord), but the Coord class does not have a defined != operator.

3b. We could not have implemented listAll recursively with just one parameter because there would’ve been no way to access the names of the super classes. With the path parameter (and strings being automatically passed by reference), this allows the recursive function to pass that as a parameter and build upon previous iterations of the recursive function.

4a.

const int N = *some value*;

bool hasCommunicatedWith[N][N];

...

int numIntermediaries[N][N];

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

{

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

// makes no sense in this case

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

{

if (i == j)

continue;

numIntermediaries[i][j] = 0;

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

{

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

continue;

if (hasCommunicatedWith[i][k] && hasCommunicatedWith[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.

4b.

const int N = some value;

bool hasCommunicatedWith[N][N];

...

int numIntermediaries[N][N];

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

{

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 N-1**

{

numIntermediaries[i][j] = 0;

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

{

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

continue;

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

numIntermediaries[i][j]++;

}

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

}

}

The time complexity is stil 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.

5.

bool combine(const Map& m1, const Map& m2, Map& result)

{

// For better performance, the bigger map should be the basis for

// the result, and we should iterate over the elements of the

// smaller one, adjusting the result as required.

const Map\* bigger;

const Map\* smaller;

if (m1.size() >= m2.size())

{

bigger = &m1;

smaller = &m2;

}

else

{

bigger = &m2;

smaller = &m1;

}

// Guard against the case that result is an alias for m1 or m2

// (i.e., that result is a reference to the same map that m1 or m2

// 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.

bool status = true;

Map res(\*bigger); // res starts as a copy of the bigger map

for (int n = 0; n < smaller->size(); n++) // for each pair in smaller **N**

{

KeyType k;

ValueType vsmall;

smaller->get(n, k, vsmall); **N**

ValueType vbig;

if (!res.get(k, vbig)) // key in smaller doesn't appear in **N** bigger

res.insert(k, vsmall); // so add it to res **N**

else if (vbig != vsmall) // same key, different value

{ // so pair shouldn't be in res

res.erase(k); **N**

status = false;

}

}

result.swap(res);

return status;

}

The time complexity is O(N^2) because the for loop runs N times (the size of smaller), and the time complexity of the get, insert, and erase functions are O(N), so the time complexity is O(N^2).