Amir Ali Yazdi-Nejad

CS 32

Carey Nachenberg

UID: 204846462

HW5

1a.

**50**

**75**

**80**

**65**

**70**

**25**

**35**

**30**

**40**

**15**

**10**

**60**

**20**

1b.

In-Order: 10 15 20 25 30 35 40 50 60 65 70 75 80

Pre-Order: 50 20 10 15 40 30 25 35 60 70 65 80 75

Post-Order: 15 10 25 35 30 40 20 65 75 80 70 60 50

1c.

Delete 30:

**50**

**60**

**25**

**35**

**40**

**20**

**70**

**10**

**80**

**65**

**15**

**75**

Delete 20:

**50**

**60**

**40**

**15**

**70**

**10**

**35**

**80**

**65**

**25**

**75**

2a.

struct Node

{

Node (int value, Node\* parent)

:m\_value (value), left (nullptr), right (nullptr), m\_parent (parent)

{}

int m\_value;

Node\* left;

Node\* right;

Node\* m\_parent;

}

2b.

insert (a pointer pointing at the root node, the integer value)

{

if (root node is equal to null pointer)

{

create a new node

set the new node’s value to the given value

set the parent node to null pointer

}

make a pointer that points to Node and set it equal to the root node

for ( ; ; )

{

if ( value is equal to the current node’s value)

return;

if (value is less than current node’s value)

{

if (current node’s left node pointer is not null pointer)

{

set current node equal to its left child

}

else

{

create a new node

set the new node’s value to the given value

set the parent node to current node

}

}

else if (value is greater than current node’s value)

{

if (current node’s right node pointer is not null pointer)

{

set current node equal to its right child

}

else

{

create a new node

set the new node’s value to the given value

set the parent node to current node

}

}

}

}

3a.

**8**

**6**

**3**

**4**

**2**

**0**

3b.

**8**

**3**

**6**

**0**

**2**

**4**

3c.

**6**

**3**

**4**

**0**

**2**

4a.

4b.

4c.

4d.

4e. 1)

4f.

4g.

4h.

4. Note: A pair<T1, T2> is a simple struct with two data members, one of type T1 and one of type T2. A set<K> and a map<K, V> are organized as binary search trees; an unordered\_set<K> and an unordered\_map<K, V> are organized as hash tables that never allow the load factor to exceed some constant, and visiting every item in a hash table of N items is O(N).

Suppose UCLA has **C** courses each of which has on average **S** students enrolled. For this problem, courses are represented by strings (e.g. "CS 32"), and students by their int UIDs. We will consider a variety of data structures, and for each determine the big-O time complexity of the appropriate way to use that data structure to determine whether a particular student *s* is enrolled in course *c*. For example, if the data structure were vector<pair<string, vector<int>>>, where each pair in the outer vector represents a course and all the students in that course, with those students being sorted in order, then if the pairs are in no particular order in the outer vector, the answer would be O(C + log S). (The reason is that we'd have to do a linear search through the outer vector to find the course, which is O(C), and then after that do a binary search of the S students in the sorted vector for that course, which is O(log S).) In these problems, we're just looking for the answer; you don't need to write the reason.

1. vector<pair<string, list<int>>>, where each pair in the outer vector represents a course and all the students in that class, with those students being sorted in order. The pairs are in no particular order in the outer vector. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
2. map<string, list<int>>, where the students in each list are in no particular order. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
3. map<string, set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
4. unordered\_map<string, set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
5. unordered\_map<string, unordered\_set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
6. Suppose we have the data structure map<string, set<int>> and we wish for a particular course *c* to write the id numbers of *all* the students in that course in sorted order. What is the big-O complexity?
7. Suppose we have the data structure unordered\_map<string, unordered\_set<int>> and we wish for a particular course *c* to write the id numbers of *all* the students in that course in sorted order (perhaps using an additional container to help with that). What is the big-O complexity?
8. Suppose we have the data structure unordered\_map<string, set<int>> and we wish for a particular student *s* to write *all* the courses that student is enrolled in, in no particular order. What is the big-O complexity?

**PROJECT 3**

#ifndef STUDENTWORLD\_H\_

#define STUDENTWORLD\_H\_

#include "GameWorld.h"

#include "GameConstants.h"

#include <list>

#include <string>

// Students: Add code to this file, StudentWorld.cpp, Actor.h, and Actor.cpp

class Actor;

class Compiler;

class StudentWorld : public GameWorld

{

public:

StudentWorld(std::string assetDir);

virtual ~StudentWorld() {}

virtual int init();

virtual int move();

virtual void cleanUp();

// Can an insect move to x,y?

bool canMoveTo(int x, int y) const;

// Add an actor to the world

void addActor(Actor\* a);

// If an item that can be picked up to be eaten is at x,y, return a

// pointer to it; otherwise, return a null pointer. (Edible items are

// only ever going be food.)

Actor\* getEdibleAt(int x, int y) const;

// If a pheromone of the indicated colony is at x,y, return a pointer

// to it; otherwise, return a null pointer.

Actor\* getPheromoneAt(int x, int y, int colony) const;

// Is an enemy of an ant of the indicated colony at x,y?

bool isEnemyAt(int x, int y, int colony) const;

// Is something dangerous to an ant of the indicated colony at x,y?

bool isDangerAt(int x, int y, int colony) const;

// Is the anthill of the indicated colony at x,y?

bool isAntHillAt(int x, int y, int colony) const;

// Bite an enemy of an ant of the indicated colony at me's location

// (other than me; insects don't bite themselves). Return true if an

// enemy was bitten.

bool biteEnemyAt(int x, int y, Actor\* me, int colony, int biteDamage);

//// Poison all poisonable actors at x,y.

void poisonAllPoisonableAt(int x, int y);

// Stun all stunnable actors at x,y.

void stunAllStunnableAt(int x, int y);

// Record another ant birth for the indicated colony.

void increaseScore(int colony);

int theWinnerColony();

void displayText();

private:

std::list<Actor\*> m\_actor[VIEW\_WIDTH][VIEW\_HEIGHT];

int tick;

int count0, count1, count2, count3;

std::vector<std::string> compilerProgram;

Compiler\* compiler[4];

int compilerSize;

};

#endif // STUDENTWORLD\_H\_

#include "StudentWorld.h"

#include "Field.h"

#include "Actor.h"

#include "Compiler.h"

#include <string>

#include <iostream>

#include <sstream>

using namespace std;

GameWorld\* createStudentWorld(string assetDir)

{

return new StudentWorld(assetDir);

}

// Students: Add code to this file (if you wish), StudentWorld.h, Actor.h and Actor.cpp

StudentWorld::StudentWorld(std::string assetDir)

: GameWorld(assetDir), count0(0), count1(0), count2(0), count3(0), compilerSize(0)

{

}

int StudentWorld::init()

{

tick = 0;

Field f;

string fieldFile = getFieldFilename();

string error;

if (f.loadField(fieldFile, error) != Field::LoadResult::load\_success)

{

setError(fieldFile + " " + error);

return false; // something bad happened!

}

compilerProgram = getFilenamesOfAntPrograms();

compilerSize = compilerProgram.size();

for (int i = 0; i < 4; i++)

{

compiler[i] = nullptr;

}

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

{

compiler[i] = new Compiler();

if (!compiler[i]->compile(compilerProgram[i], error))

{

setError(compilerProgram[i] + " " + error);

return GWSTATUS\_LEVEL\_ERROR;

}

}

// otherwise the load was successful and you can access the

// contents of the field – here’s an example

for (int x = 0; x < VIEW\_WIDTH; x++)

{

for (int y = 0; y < VIEW\_HEIGHT; y++)

{

Field::FieldItem item = f.getContentsOf(x, y); // note it’s x,y and not y,x!!!

switch (item)

{

case Field::FieldItem::rock:

m\_actor[y][x].push\_back(new Pebble(x, y, this));

//count++;

break;

case Field::FieldItem::grasshopper:

m\_actor[y][x].push\_back(new BabyGrasshopper(x, y, this));

//count++;

break;

case Field::FieldItem::food:

m\_actor[y][x].push\_back(new Food(x, y, this, 6000));

//count++;

break;

case Field::FieldItem::anthill0:

{

if (compiler[0] == nullptr)

break;

m\_actor[y][x].push\_back(new AntHill(x, y, this, 0, compiler[0]));

//count++;

break;

}

case Field::FieldItem::anthill1:

{

if (compiler[1] == nullptr)

break;

m\_actor[y][x].push\_back(new AntHill(x, y, this, 1, compiler[1]));

//count++;

break;

}

case Field::FieldItem::anthill2:

{

if (compiler[2] == nullptr)

break;

m\_actor[y][x].push\_back(new AntHill(x, y, this, 2, compiler[2]));

//count++;

break;

}

case Field::FieldItem::anthill3:

{

if (compiler[3] == nullptr)

break;

m\_actor[y][x].push\_back(new AntHill(x, y, this, 3, compiler[3]));

//count++;

break;

}

case Field::FieldItem::water:

m\_actor[y][x].push\_back(new Water(x, y, this));

//count++;

break;

case Field::FieldItem::poison:

m\_actor[y][x].push\_back(new Poison(x, y, this));

//count++;

break;

default:

break;

}

}

}

return GWSTATUS\_CONTINUE\_GAME;

}

int StudentWorld::move()

{

tick++; // update the current tick # in the simulation

// the term "actors" refers to all ants, anthills, poison, pebbles,

// baby and adult grasshoppers, food, pools of water, etc.

// give each actor a chance to do something

if (tick <= 2000)

{

for (int x = 0; x < VIEW\_WIDTH; x++)

{

for (int y = 0; y < VIEW\_HEIGHT; y++)

{

for (list<Actor\*>::iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

(\*it)->setMoved(false);

}

}

}

for (int x = 0; x < VIEW\_WIDTH; x++)

{

for (int y = 0; y < VIEW\_HEIGHT; y++)

{

//for (list<Actor\*>::iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++) // ERROR! List erase iterator

list<Actor\*>::iterator it = m\_actor[y][x].begin(); // outside range

while (it != m\_actor[y][x].end())

{

if ((\*it)->isDead())

{

delete \*it;

it = m\_actor[y][x].erase(it);

}

else

{

if ((\*it)->getMoved())

it++;

else

{

// get the actor’s current location

int oldx = (\*it)->getX();

int oldy = (\*it)->getY();

Actor\* tmp = \*it;

(\*it)->doSomething();

(\*it)->setMoved(true);

int newx = (\*it)->getX();

int newy = (\*it)->getY();

if (oldx != newx || oldy != newy)

{

m\_actor[newy][newx].push\_back(tmp);

it = m\_actor[oldy][oldx].erase(it);

}

else

it++;

}

}

}

}

}

displayText(); // update the ticks/ant stats text at screen top

//if the simulation’s over (ticks == 2000) then see if we have a winner

return GWSTATUS\_CONTINUE\_GAME;

}

if (count0 == 5 && count1 == 5 && count2 == 5 && count3 == 5)

return GWSTATUS\_NO\_WINNER;

else

{

int theWinner = theWinnerColony();

if(theWinner == count0)

setWinner(compilerProgram[0]);

if (theWinner == count1)

setWinner(compilerProgram[1]);

if (theWinner == count2)

setWinner(compilerProgram[2]);

if (theWinner == count3)

setWinner(compilerProgram[3]);

return GWSTATUS\_PLAYER\_WON;

}

}

int StudentWorld::theWinnerColony()

{

int winner1 = max(count0, count1);

int winner2 = max(count2, count3);

return max(winner1, winner2);

}

void StudentWorld::cleanUp()

{

for (int x = 0; x < VIEW\_WIDTH; x++)

{

for (int y = 0; y < VIEW\_HEIGHT; y++)

{

for (list <Actor\*>::iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end();)

{

delete \*it;

it = m\_actor[y][x].erase(it);

}

}

}

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

{

delete compiler[i];

}

}

void StudentWorld::displayText()

{

ostringstream display;

if (compilerSize == 1)

display << "Ticks: " << 2000 - tick << " - " << compilerProgram[0] << ": " << count0;

else if (compilerSize == 2)

{

display << "Ticks: " << 2000 - tick << " - " << compilerProgram[0] << ": " << count0 << " - " << compilerProgram[1]

<< ": " << count1;

}

else if (compilerSize == 3)

{

display << "Ticks: " << 2000 - tick << " - " << compilerProgram[0] << ": " << count0 << " - " << compilerProgram[1]

<< ": " << count1 << " - " << compilerProgram[2] << ": " << count2;

}

else if (compilerSize == 4)

{

display << "Ticks: " << 2000 - tick << " - " << compilerProgram[0] << ": " << count0 << " - " << compilerProgram[1]

<< ": " << count1 << " - " << compilerProgram[2] << ": " << count2 << " - " << compilerProgram[3] << ": " << count3;

}

string gameStart = display.str();

setGameStatText(gameStart);

}

void StudentWorld::addActor(Actor\* a)

{

int x = a->getX();

int y = a->getY();

m\_actor[y][x].push\_back(a);

}

bool StudentWorld::canMoveTo(int x, int y) const

{

if (x < 0 || x >= VIEW\_WIDTH || y < 0 || y >= VIEW\_HEIGHT)

return false;

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->blocksMovement())

return false;

}

return true;

}

Actor\* StudentWorld::getEdibleAt(int x, int y) const

{

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->isEdible())

return \*it;

}

return nullptr;

}

Actor\* StudentWorld::getPheromoneAt(int x, int y, int colony) const

{

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->isPheromone(colony))

return \*it;

}

return nullptr;

}

bool StudentWorld::isEnemyAt(int x, int y, int colony) const

{

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->isEnemy(colony))

return true;

}

return false;

}

bool StudentWorld::isDangerAt(int x, int y, int colony) const

{

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->isDangerous(colony))

return true;

}

return false;

}

bool StudentWorld::isAntHillAt(int x, int y, int colony) const

{

for (list<Actor\*>::const\_iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end(); it++)

{

if ((\*it)->isAntHill(colony))

return true;

}

return false;

}

bool StudentWorld::biteEnemyAt(int x, int y, Actor\* me, int colony, int biteDamage)

{

vector<Actor\*> enemies;

for (list<Actor\*>::iterator it = m\_actor[y][x].begin(); it != m\_actor[y][x].end();

it++)

{

if ((\*it) != me && (\*it)->isEnemy(colony))

enemies.push\_back(\*it);

}

if (!enemies.empty())

{

int randomNum = randInt(0, enemies.size() - 1);

enemies[randomNum]->getBitten(biteDamage);

return true;

}

return false;

}

void StudentWorld::poisonAllPoisonableAt(int x, int y)

{

list<Actor\*>::iterator it = m\_actor[y][x].begin();

if ((\*it) == nullptr)

return;

for (; it != m\_actor[y][x].end(); it++)

(\*it)->getPoisoned();

}

void StudentWorld::stunAllStunnableAt(int x, int y)

{

list<Actor\*>::iterator it = m\_actor[y][x].begin();

if ((\*it) == nullptr)

return;

for (; it != m\_actor[y][x].end(); it++)

(\*it)->getStunned();

}

void StudentWorld::increaseScore(int colony)

{

switch (colony)

{

case(0):

count0++;

break;

case(1):

count1++;

break;

case(2):

count2++;

break;

case(3):

count3++;

break;

}

}