//goes down ALL paths and can reach the base case multiple times

//will eventually end on its own

void Boggle::lookforWord(const string& str, int x, int y) {

if (str == "") {

trie.insert(str)

}

isUsed[x][y] = true;

//lookThroughNeighbors

for (int row = LEFT; row <= RIGHT; row++) {

for (int col = TOP; col <= BOTTOM; col++) {

//for (each valid neighbor [no repeats and in bounds]) {

if (isValidNeighbor(x + row, y + col) && myBoard[x+row][y+col] == str[0]) {

//findWords(tmp, coordinates of neighbor)

lookforWord(str.substr(1), x + row, y + col);

}

}

}

isUsed[x][y] = false;

}

//private helper function for isWordPresent

//function goes down all paths until a SINGLE solution is found

//if no solution is found, returns false

bool Boggle::lookforWord(const string& str, int x, int y) {

if (str == "") {

return true;

}

isUsed[x][y] = true;

//lookThroughNeighbors

for (int row = LEFT; row <= RIGHT; row++) {

for (int col = TOP; col <= BOTTOM; col++) {

//for (each valid neighbor [no repeats and in bounds]) {

if (isValidNeighbor(x + row, y + col) && myBoard[x+row][y+col] == str[0]) {

//findWords(tmp, coordinates of neighbor)

if (lookforWord(str.substr(1), x + row, y + col)) {

isUsed[x][y] = false;

return true;

}

}

}

}

isUsed[x][y] = false;

return false;

}

//private helper function for isWordPresent

//function goes down ALL paths and remembers if any path returned true

//if no solution is found, returns false

bool Boggle::lookforWord(const string& str, int x, int y) {

bool isFound = false;

if (str == "") {

return true;

}

//lookThroughNeighbors

for (int row = LEFT; row <= RIGHT; row++) {

for (int col = TOP; col <= BOTTOM; col++) {

//for (each valid neighbor [no repeats and in bounds]) {

if (isValidNeighbor(x + row, y + col) && myBoard[x+row][y+col] == str[0]) {

//findWords(tmp, coordinates of neighbor)

if (lookforWord(str.substr(1), x + row, y + col))

isFound = true;

}

}

}

return isFound;

}

//NOT a backtracking algorithm (also doesn’t work since board positions are not reset to false

//function only goes down ONE path (the first valid one)

//returns true if that path resulted in success, else returns false

bool Boggle::lookforWord(const string& str, int x, int y) {

if (str == "") {

return true;

}

isUsed[x][y] = true;

//lookThroughNeighbors

for (int row = LEFT; row <= RIGHT; row++) {

for (int col = TOP; col <= BOTTOM; col++) {

//for (each valid neighbor [no repeats and in bounds]) {

if (isValidNeighbor(x + row, y + col) && myBoard[x+row][y+col] == str[0]) {

//findWords(tmp, coordinates of neighbor)

return lookforWord(str.substr(1), x + row, y + col);

}

}

}

isUsed[x][y] = false;

Return false;

}

//attempt to do lookforWord another way-Project 10 copy version, doesn’t ever call lookforWord(“”)

// str is only lowercase letters && at least MIN\_WORD\_LENGTH long

//post: returns true if str is on the Board, else false

bool Boggle::lookforWord(const string& str, int x, int y) {

if (str == "") {

return true;

}

if (myBoard[x][y] == str[0]) {

isUsed[x][y] = true;

//lookThroughNeighbors

for (int row = LEFT; row <= RIGHT; row++) {

for (int col = TOP; col <= BOTTOM; col++) {

//for (each valid neighbor [no repeats and in bounds]) {

if (isValidNeighbor(x + row, y + col)) {

//findWords(tmp, coordinates of neighbor)

if (lookforWord(str.substr(1), x + row, y + col)) {

isUsed[x][y] = false;

return true;

}

}

}

}

isUsed[x][y] = false;

}

return false;

}