CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 12/6*

ROADMAP

Last time

- equality testing
- exercise 08

Today

- exercise 08 solution
- exercise 09

Coming up

- exercise 10
- search (linear and binary)

ANNOUNCEMENTS

FINAL EXAM NOTICE

CHECK YOUR ENTIRE FINAL EXAM SCHEDULE!

http://blogs.advising.buffalo.edu/beadvised/posts/have-you-checked-your-final-exam-schedule-4/

Room assignments will be announced at a later date.

Lab 11

Due 9:00 PM on last day of classes for everyone.

This week – regular recitations and UTA office hours.

Next week – recitations are office hours.

Required functionality 80 pts.
Optional functionality – pick and choose.

100 points is full credit.
You can get more than 100 points.

Lecture exercises will give hints for lab 11 functionality.

EXERCISE 08
solution

EXERCISE 08

Define a method a method with this header in a class named quiz.Question:

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>>
board)
```

Define the method so that it returns a HashSet containing those points on the board whose positions are adjacent to p and have the same contents.

Consider this board

```
aabbc
abbcd
ddddd
efffd
```

Answer for null is { }

Answer for (0,0) is { (1,0) , (0,1) }

Answer for (2,4) is { (1,4) , (2,3) , (3,4) }

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {



}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();









  return result;
}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();
  if (p != null) {
    Point check;
    check = new Point(p.x-1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }









  }
  return result;
}

private boolean matches(Point p, Point q, ArrayList<ArrayList<String>> board) {
  return board.get(p.x).get(p.y).equals(board.get(q.x).get(q.y));;
}

private boolean inBounds(Point p, ArrayList<ArrayList<String>> board) {
  return p.x >=0 && p.x < board.size() && p.y >= 0 && p.y < board.get(0).size();
}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();
  if (p != null) {
    Point check;
    check = new Point(p.x-1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x+1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }




  }
  return result;
}

private boolean matches(Point p, Point q, ArrayList<ArrayList<String>> board) {
  return board.get(p.x).get(p.y).equals(board.get(q.x).get(q.y));;
}

private boolean inBounds(Point p, ArrayList<ArrayList<String>> board) {
  return p.x >=0 && p.x < board.size() && p.y >= 0 && p.y < board.get(0).size();
}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();
  if (p != null) {
    Point check;
    check = new Point(p.x-1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x+1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x,p.y-1);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x,p.y+1);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
  }
  return result;
}

private boolean matches(Point p, Point q, ArrayList<ArrayList<String>> board) {
  return board.get(p.x).get(p.y).equals(board.get(q.x).get(q.y));;
}

private boolean inBounds(Point p, ArrayList<ArrayList<String>> board) {
  return p.x >=0 && p.x < board.size() && p.y >= 0 && p.y < board.get(0).size();
}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();
  if (p != null) {
    Point check;
    check = new Point(p.x-1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x+1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x,p.y-1);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }


  }
  return result;
}

private boolean matches(Point p, Point q, ArrayList<ArrayList<String>> board) {
  return board.get(p.x).get(p.y).equals(board.get(q.x).get(q.y));;
}

private boolean inBounds(Point p, ArrayList<ArrayList<String>> board) {
  return p.x >=0 && p.x < board.size() && p.y >= 0 && p.y < board.get(0).size();
}
```

```
public HashSet<Point> answer(Point p, ArrayList<ArrayList<String>> board) {
  HashSet<Point> result = new HashSet<Point>();
  if (p != null) {
    Point check;
    check = new Point(p.x-1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x+1,p.y);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x,p.y-1);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
    check = new Point(p.x,p.y+1);
    if (inBounds(check, board) && matches(p,check,board)) { result.add(check); }
  }
  return result;
}

private boolean matches(Point p, Point q, ArrayList<ArrayList<String>> board) {
  return board.get(p.x).get(p.y).equals(board.get(q.x).get(q.y));;
}

private boolean inBounds(Point p, ArrayList<ArrayList<String>> board) {
  return p.x >=0 && p.x < board.size() && p.y >= 0 && p.y < board.get(0).size();
}
```

EXERCISE 09

"Flood Fill" Algorithm

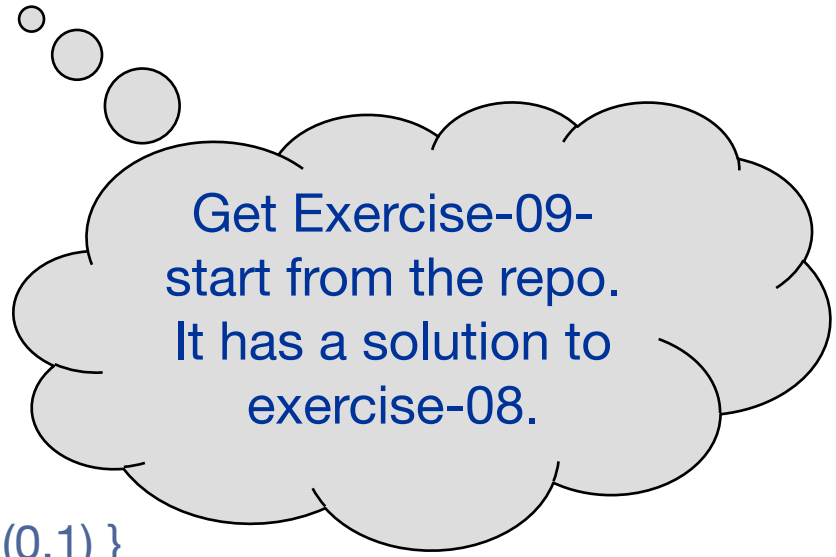
Define a method a method with this header in a class named quiz.Question:

```
public HashSet<Point> maximalMatchedRegion(Point p, ArrayList<ArrayList<String>> board)
```

Define the method so that it returns a HashSet containing those points of the largest contiguous region matching position p.

Consider this board

```
aabbc
abbcd
ddddd
efffd
```



Answer for null is { }

Answer for (0,0) is { (0,0) , (1,0) , (0,1) }

Answer for (0,3) is { (0,2) , (0,3) , (1,1) , (1,2) }

Answer for (2,4) is { (1,4) , (2,0) , (2,1) , (2,2) , (2,3) , (2,4) , (3,4) }

"Flood Fill" Algorithm

A	A	B	B	C
A	B	B	C	D
D	D	D	D	D
E	F	F	F	D

---

candidates: { (0,3) } matches: { } region: { }
candidates: { (0,3) } matches: { (0,2) } region: { }

---

candidates: { (0,2) } matches: { } region: { (0,3) }
candidates: { (0,2) } matches: { (1,2) } region: { (0,3) }

---

candidates: { (1,2) } matches: { } region: { (0,2) (0,3) }
candidates: { (1,2) } matches: { (1,1) } region: { (0,2) (0,3) }

---

candidates: { (1,1) } matches: { } region: { (0,2) (1,2) (0,3) }
candidates: { (1,1) } matches: { } region: { (0,2) (1,2) (0,3) }

---

candidates: { } matches: { } region: { (0,2) (1,2) (1,1) (0,3) }