CS 91 USACO

Bronze Division

Unit 4: Basic Tree and Graphs

LECTURE 18: DEPTH-FIRST SEARCH

DR. ERIC CHOU

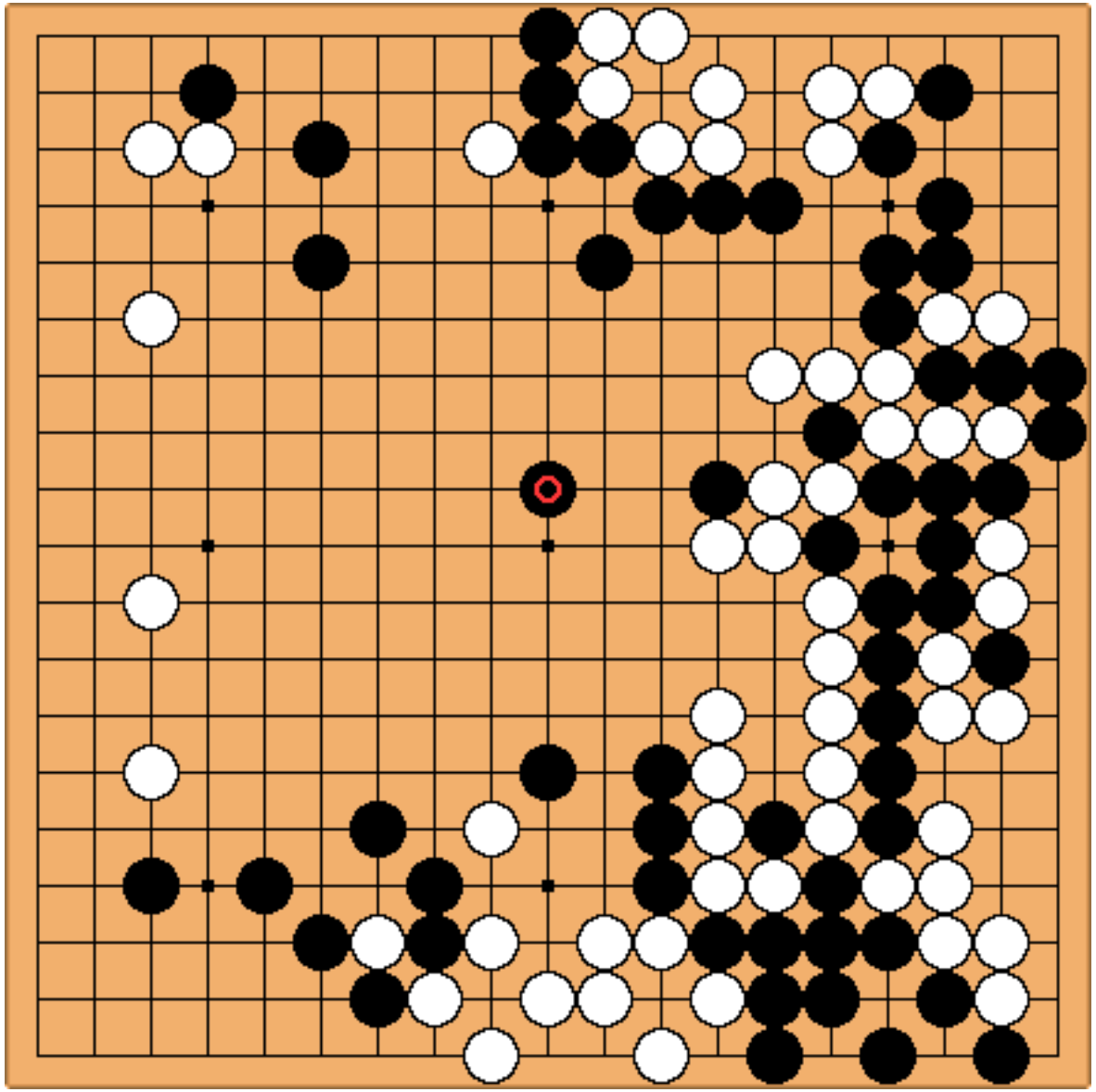
IEEE SENIOR MEMBER

Objectives

- Depth-first searching on 2D Map
- Connected Component Detection
- Exam: 2019 Jan. Silver P2, Perimeter

Connected Component

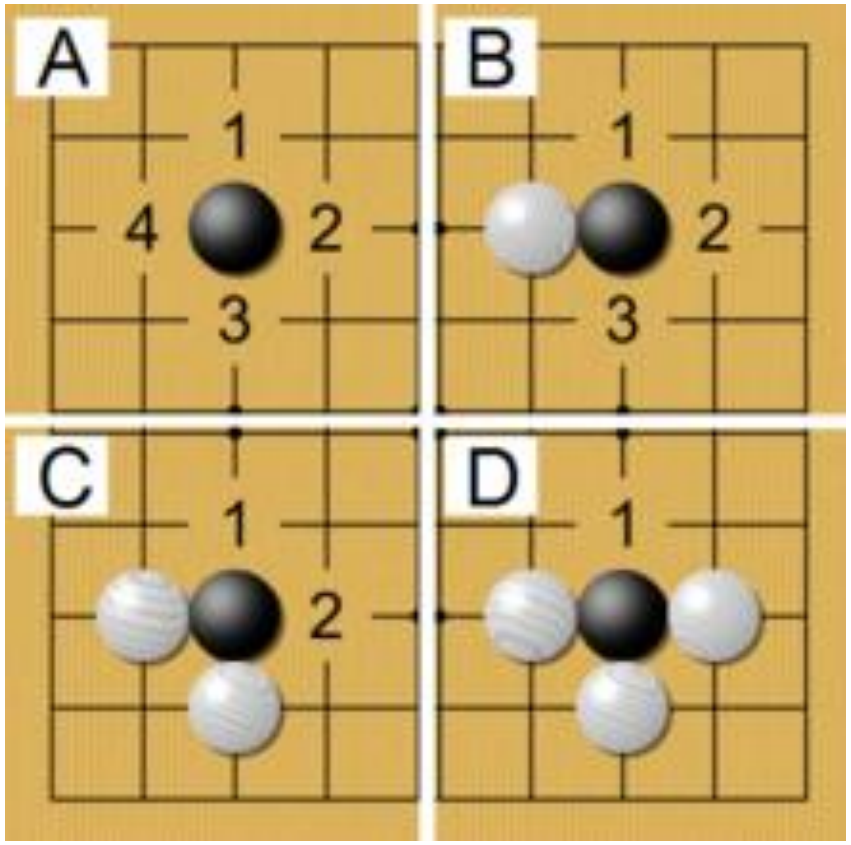
SECTION 1



Go Game

Depth First Search for Connected Component on 2D Map

1. 4 neighboring nodes
2. Find the largest connected component
3. Count the number of connected component



4 Direction Vectors

```
int[] dx = {1, 0, -1, 0};
int[] dy = {0, 1, 0, -1};
```

Use Group ID Matrix as the visited record

- Each visited node will be assigned with a group id.
- This group id should also be used as the visited matrix.

```java
import java.util.*;
import java.io.*;
public class Go
{
    static int[] dx = {1, 0, -1, 0};
    static int[] dy = {0, 1, 0, -1};

    public static void printMatrix(int[][] m){
        for (int i=0; i<m.length; i++){
            for (int j=0; j<m[i].length; j++){
                System.out.printf("%3d", m[i][j]);
            }
            System.out.println();
        }
    }
```

Go.java

```java
public static void visit(int[][] m, int r, int c, int stone, int group){
    m[r][c] = group;
    for (int i=0; i<4; i++){
       int y = r+dy[i];
       int x = c+dx[i];
       if (y<0 || y>=m.length || x<0 || x>=m.length) continue;
       if (m[y][x] >=0 || m[y][x]!= stone) continue;

       visit(m, y, x, stone, group);
     }
 }
```

```
public static void main(String[] args) throws Exception{
    System.out.print("\f");
    Scanner input = new Scanner(new File("go2.txt"));
    PrintWriter out = new PrintWriter(new File("go.out"));

    int N = Integer.parseInt(input.nextLine());
    int[][] m = new int[N][N];
    for (int i=0; i<N; i++){
       String line = input.nextLine();
       for (int j=0; j<line.length(); j++){
            if (line.charAt(j)=='#') m[i][j] = -2;
            if (line.charAt(j)=='_') m[i][j] = -1;
         }
     }
    System.out.println("Initial Map: ");
    printMatrix(m);
    System.out.println();
```

Go.java

```
        int count = 0;
        for (int r=0; r<N; r++){
            for (int c=0; c<N; c++){
                if (m[r][c]<0){
                    visit(m, r, c, m[r][c], count);
                    count++;
                 }
             }
         }

        printMatrix(m);
        System.out.println();
        System.out.println(count);
        out.close();
        input.close();
    }
}
```

```
Initial Map:
 -2 -1 -2 -2 -2 -2 -1 -2 -2 -2
 -1 -1 -1 -2 -2 -1 -1 -1 -2 -2
 -1 -2 -2 -2 -1 -1 -2 -2 -2 -1
 -1 -2 -2 -1 -1 -1 -2 -2 -1 -1
 -2 -1 -1 -1 -1 -2 -1 -1 -1 -1
 -2 -1 -2 -2 -2 -2 -1 -1 -1 -1
 -1 -1 -1 -2 -2 -1 -2 -2 -1 -1
 -1 -2 -2 -2 -1 -2 -1 -1 -1 -1
 -1 -2 -2 -1 -1 -2 -1 -2 -2 -2
 -2 -1 -1 -1 -1 -1 -2 -2 -1 -1
```

0	1	2	2	2	2	3	4	4	4
1	1	1	2	2	3	3	3	4	4
1	2	2	2	3	3	4	4	4	5
1	2	2	3	3	3	4	4	5	5
6	3	3	3	3	7	5	5	5	5
6	3	7	7	7	7	5	5	5	5
3	3	3	7	7	8	9	9	5	5
3	7	7	7	10	11	5	5	5	5
3	7	7	10	10	11	5	12	12	12
13	10	10	10	10	10	12	12	14	14

Exam: Jan. 2019 Silver P2: Icy Perimeter (perimeter)

SECTION 1

Problem Statement

- Farmer John is going into the ice cream business! He has built a machine that produces blobs of ice cream but unfortunately in somewhat irregular shapes, and he is hoping to optimize the machine to make the shapes produced as output more reasonable.

Problem Statement

- The configuration of ice cream output by the machine can be described using an $N \times N$ grid ($1 \leq N \leq 1000$) as follows:

```
##....
....#.
.#..#.
.#####
...###
....##
```

- Each '.' character represents empty space and each '#' character represents a 1×1 square cell of ice cream.

Problem Statement

- Unfortunately, the machine isn't working very well at the moment and might produce multiple disconnected blobs of ice cream (the figure above has two). A blob of ice cream is connected if you can reach any ice cream cell from every other ice cream cell in the blob by repeatedly stepping to adjacent ice cream cells in the north, south, east, and west directions.
- Farmer John would like to find the area and perimeter of the blob of ice cream having the largest area. The area of a blob is just the number of '#' characters that are part of the blob. If multiple blobs tie for the largest area, he wants to know the smallest perimeter among them. In the figure above, the smaller blob has area 2 and perimeter 6, and the larger blob has area 13 and perimeter 22.

Problem Statement

- Note that a blob could have a "hole" in the middle of it (empty space surrounded by ice cream). If so, the boundary with the hole also counts towards the perimeter of the blob. Blobs can also appear nested within other blobs, in which case they are treated as separate blobs. For example, this case has a blob of area 1 nested within a blob of area 16:

```
#####
#...#
#.#.#
#...#
#####
```

Problem Statement

- Knowing both the area and perimeter of a blob of ice cream is important, since Farmer John ultimately wants to minimize the ratio of perimeter to area, a quantity he calls the icyperimetric measure of his ice cream. When this ratio is small, the ice cream melts slower, since it has less surface area relative to its mass.

INPUT FORMAT (file perimeter.in):

- The first line of input contains N, and the next N lines describe the output of the machine. At least one '#' character will be present.

SAMPLE INPUT:

```
6
##....
....#.
.#..#.
.#####
...###
....##
```

OUTPUT FORMAT (perimeter.out):

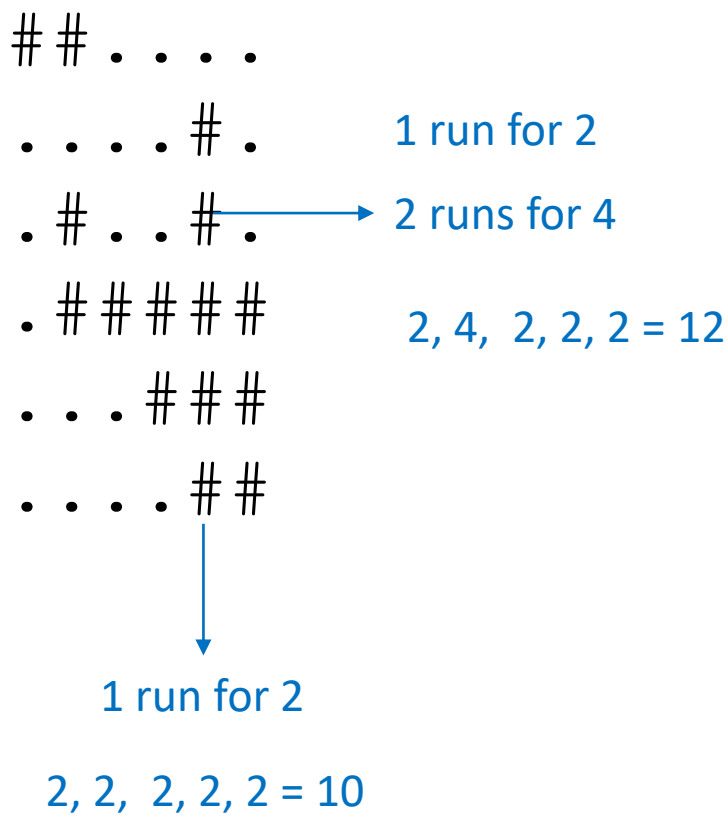
- Please output one line containing two space-separated integers, the first being the area of the largest blob, and the second being its perimeter. If multiple blobs are tied for largest area, print the information for whichever of these has the smallest perimeter.

SAMPLE OUTPUT:

```
13 22
```

Nature of Problem

- Find the largest connected component. Then, then count the area and the perimeter.
- Area, count the number of elements.
- Perimeter, count the number of connected run for each row and column and, then, multiply by 2.



Pairing Generation

SECTION 1

Pairing

- Pairing is the process to group elements into pairs. Let's assume there are even number of elements to be paired (2N).
- There will be totally, (2N-1) * (2N-3) * … * 1 possible pairings.

Pairing

- For example, there are 6 elements (ABCDEF).
- There will be totally 15 possible pairings: 5 * 3 * 1
- (A, B)(C, D)(E, F), (A, B)(C, E)(D, F), (A, B)(C, F)(D, E)
- (A, C)(B, D)(E, F), (A, C)(B, E)(D, F), (A, C)(B, F)(D, E)
- (A, D)(B, C)(E, F), (A, D)(B, E)(C, F), (A, D)(B, F)(C, D)
- (A, E)(B, C)(D, F), (A, E)(B, D)(C, F), (A, E)(B, F)(C, D)
- (A, F)(B, C)(D, E), (A, F)(B, D)(C, E), (A, F)(B, E)(C, F)

Basic Paring Algorithm

Demo Program: pairings.java

```
32  public static void pair(int[] partners){
33      int i=0;
34      while (i<alphabet.length && partners[i]!=-1) i++;
35      if (i==alphabet.length){
36          System.out.println(Arrays.toString(partners));
37          return;
38      }
39
40      for (int j=i+1; j<alphabet.length; j++){
41          if (partners[j]==-1) {
42              partners[i] = j;
43              partners[j] = i;
44              pair(partners);
45              partners[i] = -1;
46              partners[j] = -1;
47          }
48      }
49  }
```

Find the first unpaired element

If all paired, generate all paringing

Iterate through the second unpaired element

Generate pairing Strings with Alphabet

Demo Program: pairing.java

```
13  public static String generate(int[] partners, String[] alphabet){
14      Set s = new HashSet<Integer>();
15      boolean first = true;
16      String text = "";
17      for (int i=0; i<partners.length; i++){
18          if (!s.contains(i)){
19              s.add(i);
20              s.add(partners[i]);
21              if (first){
22                  text += ""+alphabet[i]+alphabet[partners[i]];
23                  first = false;
24              }
25              else {
26                  text += "-"+alphabet[i]+alphabet[partners[i]];
27              }
28          }
29      }
30      return text;
31  }
```

Generate pairing Strings with Alphabet

Demo Program: pairing.java

```
33  public static void pair(int[] partners){
34      //System.out.println(Arrays.toString(partners));
35      int i=0;
36      while (i<alphabet.length && partners[i]!=-1) i++;
37      if (i==alphabet.length){
38          //System.out.println(Arrays.toString(partners));
39          pairings.add(generate(partners, alphabet));
40          return;
41      }
42
43      for (int j=i+1; j<alphabet.length; j++){
44          if (partners[j]==-1) {
45              partners[i] = j;
46              partners[j] = i;
47              pair(partners);
48              partners[i] = -1;
49              partners[j] = -1;
50          }
51      }
52  }
```

```
AB-CD-EF
AB-CE-DF
AB-CF-DE
AC-BD-EF
AC-BE-DF
AC-BF-DE
AD-BC-EF
AD-BE-CF
AD-BF-CE
AE-BC-DF
AE-BD-CF
AE-BF-CD
AF-BC-DE
AF-BD-CE
AF-BE-CD
```