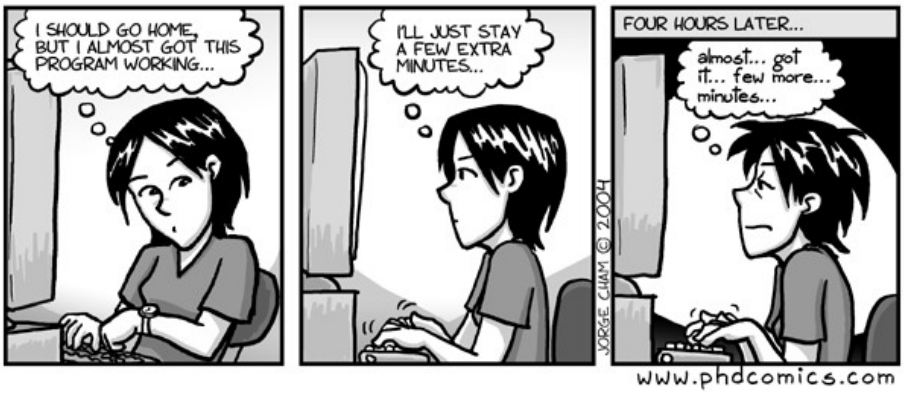
CS 302 – Assignment #09

Purpose: Learn concepts regarding disjoint sets data structure.
Due: Tuesday (3/28) → Must be submitted on-line before class.
Points: Part A → 75 pts, Part B → 25 pts

Assignment:

Part A:
Design and implement a C++ class, ***disjointSets***, to implement a disjoint sets[1] data structure. The disjoint sets class will use arrays for the parent links and sizes. A test program is provided to allow independent testing of the disjoint set class.



Once the disjoint sets class is working, create a new class, ***connectedComponents***, to perform some simple image manipulation. This will include reading a gray scale image in the portable gray map (PGM)[2] format. The ***connectedComponents*** object will identify all the connected components in the image and allow one of them to be changed to a user provided gray scale value (0-255). Objects will be considered connected if the set of gray scale values is within a user specific threshold. The threshold value, replacement value, input file name, and output file name arguments are provided on the command line.

A main will be provided. Refer to the UML descriptions for implementation details.

Part B:
Create and submit a brief write-up, not to exceed ~500 words, including the following:

- Name, Assignment, Section.
- Summary of data structures used.
- Note some other applications for the disjoint sets data structure.
- Big-O for disjoint sets functions (*setUnion* and *setFind*).

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- Submit a copy of the write-up (PDF format).

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make**. You must have a valid, working *makefile*.

1 For more information, refer to: http://en.wikipedia.org/wiki/Disjoint-set_data_structure
2 For more information, refer to: https://en.wikipedia.org/wiki/Netpbm_format

Class Descriptions

- Disjoint Sets Class
  The disjoint sets class will implement functions specified below.

disjointSets
-totalSize: int
-setsCount: int
-links: *int
-sizes: *int
+disjointSets()
+~disjointSets()
+createSets(int): void
+getTotalSets() const: int
+getSetCount() const: int
+getSetSize(const int) const: int
+printSets() const: void
+setUnion(int, int): int
+setFind(int): int

Function Descriptions

- The *disjointSets()* constructor should initialize the class variables to an empty state as appropriate (0, NULL, etc.).
- The *~disjointSets()* destructor should delete the dynamically allocated memory.
- The *createSets()* function should create the new sets of the passed size. The parameter must be checked to ensure it is > 0. If invalid, nothing should be changed.
- The *getTotalSets()* function should return total sets class variable which represents the number of total possible sets.
- The *getSetCount()* function should return the current number of unique sets (which will always be ≤ to the number of possible sets).
- The *getSetSize()* function should return the set size of the passed set.
- The *printSet()* function should print the current disjoint set status; index, links, and sizes. Refer to the example output for formatting.
- The *setUnion()* function should perform a *union-by-size* operation between the two passed sets, update the sizes, and return the parent. If the set sizes are equal, the parent should be the 1st argument.
- The *setFind()* functions should search for and return the parent of the passed set. *Note*, this may be the set itself. *Note*, the function ***must*** perform path compression.

- Connected Components Class
  The connected components class will implement functions specified below.

connectedComponents
-**image: unsigned char
-rows: int
-columns: int

-threshold: int
-imageSet: disjointSet
-MIN_SIZE=10: static constexpr int
-MAX_SIZE=1000000: static constexpr int
+connectedComponents()
+~connectedComponents()
+readImageFile(const string): bool
+identify(): void
+fillComponent(const int): void
+writeImageFile(const string): bool
+setTheshold(const int): void
+showStatus() const: void
-withinThreshold(const unsigned char, const unsigned char) const: bool

Function Descriptions

- The *connectedComponents()* constructor should initialize the class variables as appropriate (0, NULL, etc.). The default threshold value should be 0.
- The *~connectedComponents()* destructor should delete the dynamically allocated memory; links and sizes arrays. Additionally, reset all class variables as appropriate.
- The *readImageFile()* function should read the PGM image file. The final must be opened in binary format (e.g., **`inFile.open(fileName.c_str(), ios::binary);`**). Additionally, the function should verify that the format is P5, that the rows and columns values within the MIN_SIZE and MAX_SIZE (inclusive), and that the maximal gray value is 255. If there is a comment line, as denoted with a '#', the line should be ignored. The two-dimensional image array should be allocated and populated from the file. The image values are in binary and need to be read as char and then assigned to the unsigned char array (without typecasting). The header lines will be read in the usual manner (i.e., **`getline()`**). The image gray scale values will be read as a single character with no white space (i.e., **`inFile.get(ch)`**). When done, the file should be closed. If the file operations are successful, the function should return true or if there are any I/O errors return false.
- The *identify()* function will identify all connected components. The is accomplished by using a disjoint sets data structure and logically assigning each pixel to a set, initially all disjoint. Then, for every pixel, check each neighbor (up, down, left, right) and union each neighbor that is the within the threshold.
- The *writeImageFile()* function should write a PGM format file. The final must be opened in binary format (e.g., **`outFile.open(fileName.c_str(), ios::binary);`**). The P5, columns, rows, and 255 (in that order) should be written as the header, the then contents of the two-dimensional image array. The header information can be written in the usual manner. However, the each pixel of the image data must be written as a char, so the image value must be assigned to a char variable to write to the file (assigned without typecasting). When done, the file should be closed. If the file operations are successful, the function should return true or if there are any I/O errors return false.
- The *setThreshold()* function should set the class threshold variable to the passed value. The function must ensure the passed value is between 0 and 255 (inclusive). If the passed value is not valid, nothing should be changed.

- The *fillComponent()* function should identify and display the 1st and 2nd largest components. The size and parent values for both the 1st and 2nd largest components should be displayed. Refer to the example output for formatting. Then, the each pixel in the ***2nd largest component*** set should be set new (passed) value. *Note*, the can only be done after the *identify()* function is called (which is ensured by the provided main).
- The *showStatus()* function should display the total possible sets, current number of sets, and image size (rows, columns). Refer to the example output for formatting.
- The private *withinThreshold()* function should determine if the two passed char values are within the class set threshold. If so, the function should return true and false otherwise.

You may add additional private functions if desired.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

Portable GrayMap (PGM) File Format

A PGM image represents a gray scale graphic image.

A PGM file consists of two parts, a header and the image data. The header consists of at least three parts normally delineated by line feeds. The first line is a PPM identifier, it should be "P5" (not including the double quotes!). In addition to the above required line, optional comment line or lines can be placed with a "#" character, where the comment extends to the end of the line. The next line consists of the columns (width) and rows (height) of the image. The last part of the header gives the maximum value of the gray scale components for the pixels, this allows the format to describe more than single byte (0..255) values. Our assignment will require the maximal gray scale value to be 255.

For example, a small file, **`line.pgm`**, would look like the following:

```
P5
# A diagonal line 10 pixels wide and 10 pixels high
24 7
255
00 FF FF FF FF FF FF FF FF FF
FF 00 FF FF FF FF FF FF FF FF
FF FF 00 FF FF FF FF FF FF FF
FF FF FF 00 FF FF FF FF FF FF
FF FF FF FF 00 FF FF FF FF FF
FF FF FF FF FF 00 FF FF FF FF
FF FF FF FF FF FF 00 FF FF FF
FF FF FF FF FF FF FF 00 FF FF
FF FF FF FF FF FF FF FF 00 FF
FF FF FF FF FF FF FF FF FF 00
```

Most images will be larger. The P5 indicates the the values are stored in the file in binary format. As such, the file must be opened using the binary specifier. The header lines can be read in the usual manner (i.e., **`getine()`**). The image values will be read as a single character with no white space (i.e., **`inFile.get(ch)`**). In order to view the file, a binary editor, such a **GHex**, may be useful.

Example Execution:

This is example output for the disjoint sets test program.

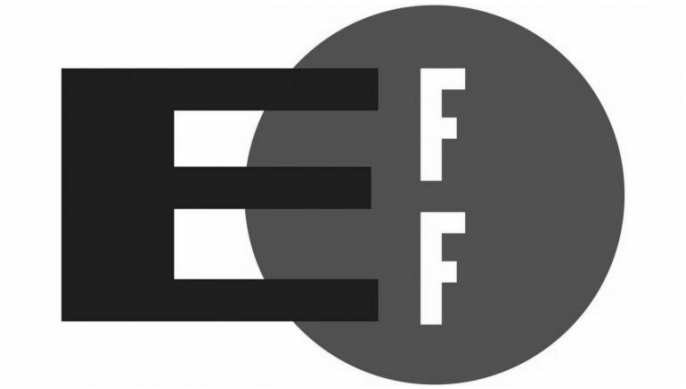
```
ed-vm% ./testDS
****************************************************************
CS 302 - Assignment #9
Disjoint Sets

---------------------------------------
Test Set 0

Initial State:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  sizes:  1  1  1  1  1  1  1  1  1  1

---------------------------------------
  union(0,1) -> 0
  union(2,3) -> 2
  union(4,5) -> 4

New State 1:
Size: 10
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1  0 -1  2 -1  4 -1 -1 -1 -1
  sizes:  2  0  2  0  2  0  1  1  1  1

---------------------------------------
  union(0,2) -> 0
  union(4,6) -> 4
  union(4,7) -> 4
  union(4,8) -> 4

New State 2:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1  0  0  2 -1  4  4  4  4 -1
  sizes:  4  0  0  0  5  0  0  0  0  1

---------------------------------------
  setFind(1): 0
  setFind(2): 0
  setFind(4): 4
  setFind(7): 4

New State 3:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1  0  0  2 -1  4  4  4  4 -1
  sizes:  4  0  0  0  5  0  0  0  0  1

---------------------------------------
m0123 = 0
m45678 = 4
  union(m0123,m45678) -> 4

New State 4:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  4  0  0  2 -1  4  4  4  4 -1
  sizes:  0  0  0  0  9  0  0  0  0  1

---------------------------------------
  setFind(3): 4
  setFind(5): 4
  setFind(7): 4

New State 5:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  4  0  4  4 -1  4  4  4  4 -1
  sizes:  0  0  0  0  9  0  0  0  0  1

---------------------------------------
  setFind(0): 4

New State 6:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  4  0  4  4 -1  4  4  4  4 -1
```

```
  sizes:  0  0  0  0  9  0  0  0  0  1

----------------------------------------
  setFind(4): 4
  setFind(6): 4
  setFind(8): 4

Final State:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  4  0  4  4 -1  4  4  4  4 -1
  sizes:  0  0  0  0  9  0  0  0  0  1

----------------------------------------
Test Set 1


Initial State:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  sizes:  1  1  1  1  1  1  1  1  1  1

Set Size: 9

Final State:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1 -1  1  1  1  1  1  1  1  1
  sizes:  1  9  0  0  0  0  0  0  0  0

testDS: Error 0
----------------------------------------
Quiz Test Set


Initial State:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  sizes:  1  1  1  1  1  1  1  1  1  1


Intermediate State 1:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1  0 -1  2 -1  4 -1 -1 -1 -1
  sizes:  2  0  2  0  2  0  1  1  1  1


Intermediate State 2:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1  0  0  2 -1  4  4  4  4 -1
  sizes:  4  0  0  0  5  0  0  0  0  1


Final State:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  4  0  0  0 -1  4  4  4  4 -1
  sizes:  0  0  0  0  9  0  0  0  0  1

testDS: error in quiz set. 4
find(0) = 4
find(4) = 4
find(7) = 4
find(6) = 4
find(8) = 4
find(9) = 9
----------------------------------------
Large Set Test

Large Set Test, Sets Count - OK
Large Set Test, Sets - OK

************************************************************
Game Over, thank you for playing.
ed-vm%
```

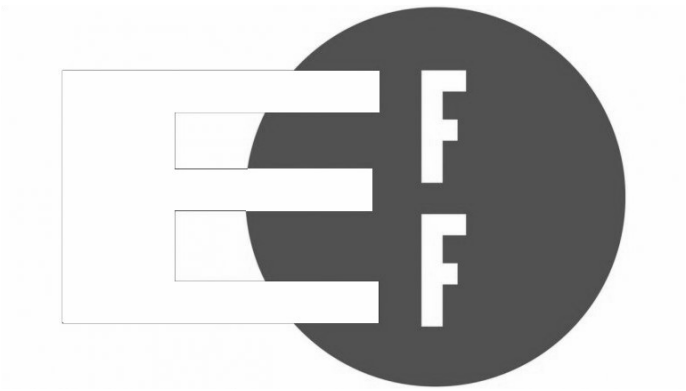
Example Execution:
Below are a couple of example executions. Given the following execution;

```
ed-vm%
ed-vm% ./identify -th 20 -fv 255 -if img2.pgm -of tmp.pgm
*****************************************************************
CS 302 - Assignment #9
Identify Connected Components

Total Possible Sets: 332544
Sets: 898
Image: 433 x 768 = 332544
1st Largest Set Size = 242015
1st Largest Set Prnt = 0
2nd Largest Set Size = 67850
2nd Largest Set Prnt = 57669
Total Possible Sets: 332544
Sets: 898
Image: 433 x 768 = 332544

*****************************************************************
Game Over, thank you for playing.
ed-vm%
```

and the input file of **img2.pgm**,



the output file, **tmp.pgm**, would appear as follows:



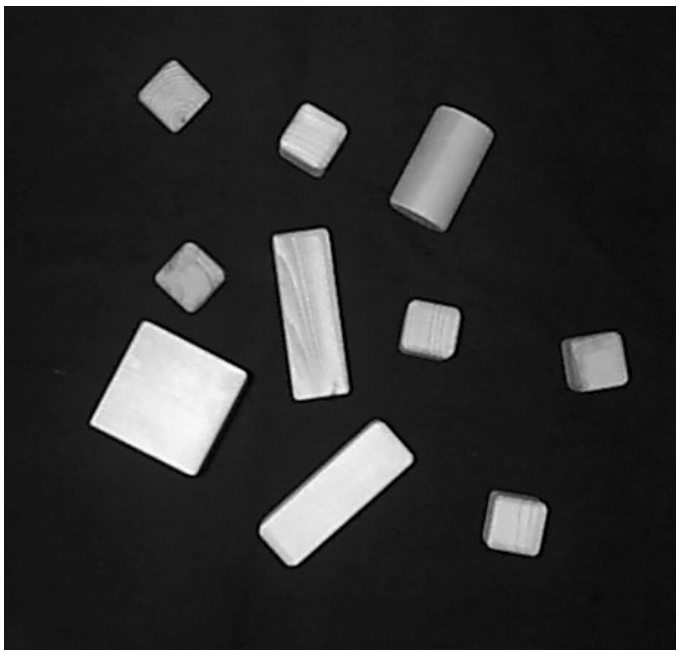
Note, in this example the 1st largest component is the background and the 2nd largest component is the stylized capital **E**.

Example Execution:

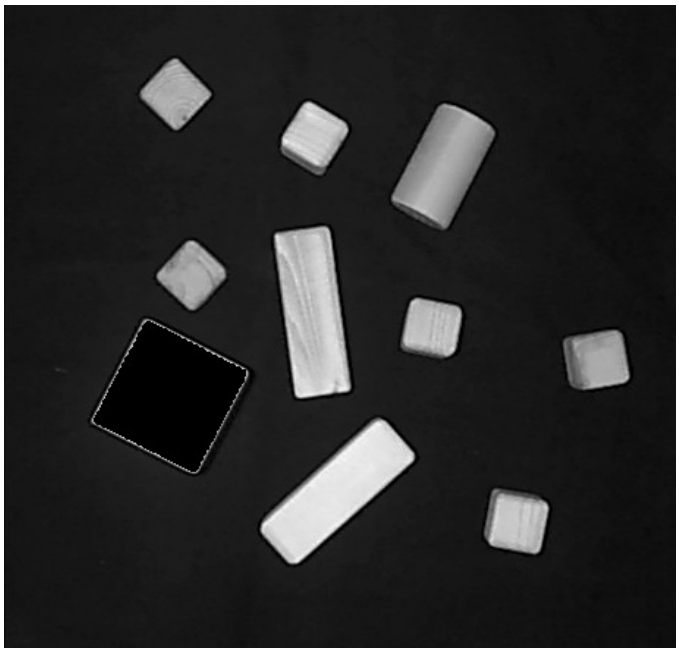
Below are a couple of example executions. Given the following execution;

```
ed-vm%
ed-vm% ./identify -th 20 -fv 1 -if test/img1.pgm -of tmp.pgm
*****************************************************************
CS 302 - Assignment #9
Identify Connected Components

Total Possible Sets: 201062
Sets: 2112
Image: 439 x 458 = 201062
1st Largest Set Size = 181574
1st Largest Set Prnt = 0
2nd Largest Set Size = 6185
2nd Largest Set Prnt = 99024
Total Possible Sets: 201062
Sets: 2112
Image: 439 x 458 = 201062

*****************************************************************
Game Over, thank you for playing.
ed-vm%
```

and the input file of **img1.pgm**,



the output file, **tmp.pgm**, would appear as follows:



Note, again in this example the 1st largest component is the background. The 2nd largest component is the largest block (lower left quadrant).