EEL 4851-001 Data Structures

MARS PATHFINDER IMAGE ANALYSIS PACKAGE

Programming Assignments (Five)

Prof. Sudeep Sarkar

Contents

1. Introduction

1.1 Why is this document important?

The goal of this set of assignments is to provide you with hands-on experience with handling various kinds of data structures. In the lectures you will be introduced to data structures at an *abstract* level and in the assignments you will write code to *implement* and *use* these data structures. The set of tasks chosen for your assignments all relate to *images*. You will design algorithms to perform simple image processing tasks using data structures such as arrays, stacks, queues, trees, and hash tables. These exercises are not only important from a grading standpoint but also are crucial learning tools.

The assignments are to be completed in *groups of three*. The main reason behind this dictum is to expose you to the experience of team work. In the real world you will definitely be called upon to work with others towards a common end. It is very difficult to convey all aspects of team work in a classroom setting. One has to learn from experience.

You are welcome to discuss with other groups *but* the implementation should be your own. Please be sincere regarding this. Contrary to what you might expect, it is relatively easy to find out if codes have been copied! **Any case of unethical conduct will result in an F in the class.**

1.2 What should I submit?

For each assignment you should submit the following by the due dates and times.

1. *Demonstration:* You are expected to demonstrate your working code to Lisa Whitcomb. *The demonstrations should be done during his office hours: 10 am to 12 noon on a Tuesday or Thursday.* (If you have problems with the days then let me or Lisa know.) You should sign up for a demonstration. Lisa will put up sign up sheets outside ENB 329 two days prior to the due time. For any demonstration with out sign up, Lisa reserves the right to refuse appointments and to consider the demo part of the assignment as being turned in late.

2. *On Paper:* Submit, on paper (to me) by the due date and time:

(a) A written five (maximum) page report discussing your results and experience. The report should include the following: (i) introduction, (ii) flowchart of the algorithm, (iii) description of how would one run and use your code, (iv) some typical results or outputs, (v) a section describing the programming bugs and errors you ran into (vi) a section describing what you learned from the assignment and (vii) a statement of division of work between the partners.

Note that you are expected to submit a professionally appearing report and it should be well organized, clear, neat, with adequate visual aids such as figures and images. Writing is a very important part of the scientific process and should not be ignored.

(b) A print out of the source code.

1.3 Grading:

The grading of the assignment will be along the following lines.

1. The writeup is out of 5 points. One points each for the following:
   (a) Flowchart
   (b) Description of usage
   (c) Results
   (d) Programming bugs and errors
   (e) Overall presentation
2. The source code is out of 5 points. One point each for the following:
   (a) The existence of the code itself
   (b) Proper indentation of the code
   (c) Comments
   (d) Proper naming of the procedures, functions, and variables.
   (e) Modularity of the code.

   For proper program style, formatting, and documentation see the guidelines in Appendix H of your text book. You should adhere to these guidelines.
3. The demonstration is out of 5 points. You get three points if the program does *exactly* what it is supposed to do. Two points are reserved for the ease of use, the type of user interface, the ability to handle various user input errors, or any extra features that your system might have.

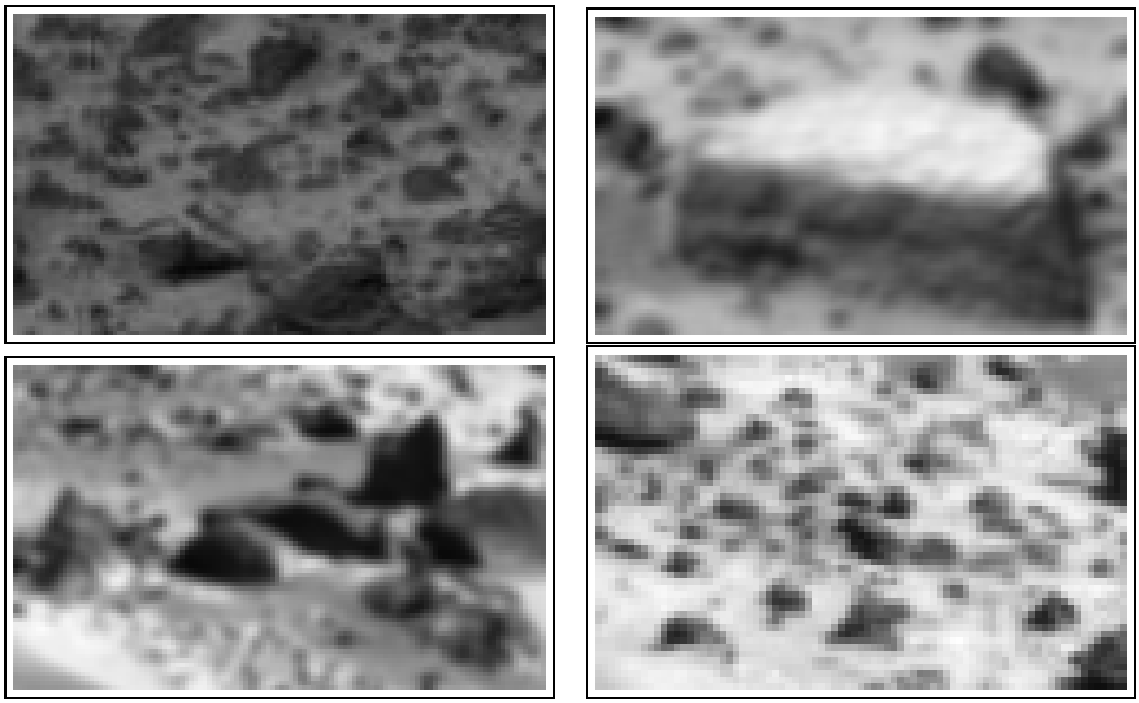


Figure 1.1: Samples of images to be used in the assignments.

Late assignments (or parts thereof) will be penalized 20% of the maximum possible grade for each extra day. Note that computer systems have a tendency go down unexpectedly. So, plan accordingly and do not wait for the last minute.

1.4 Data

All image data for the assignments will be available from the class homepage on Netscape. Figure 1.1 shows a sample of images that are available.

1.5 How are images represented in the computer?

An image as defined on, say a photographic film, is a continuous function of brightness values. Each point on the developed film can be associated with a gray level value representing how bright that particular point is. To store images in a computer

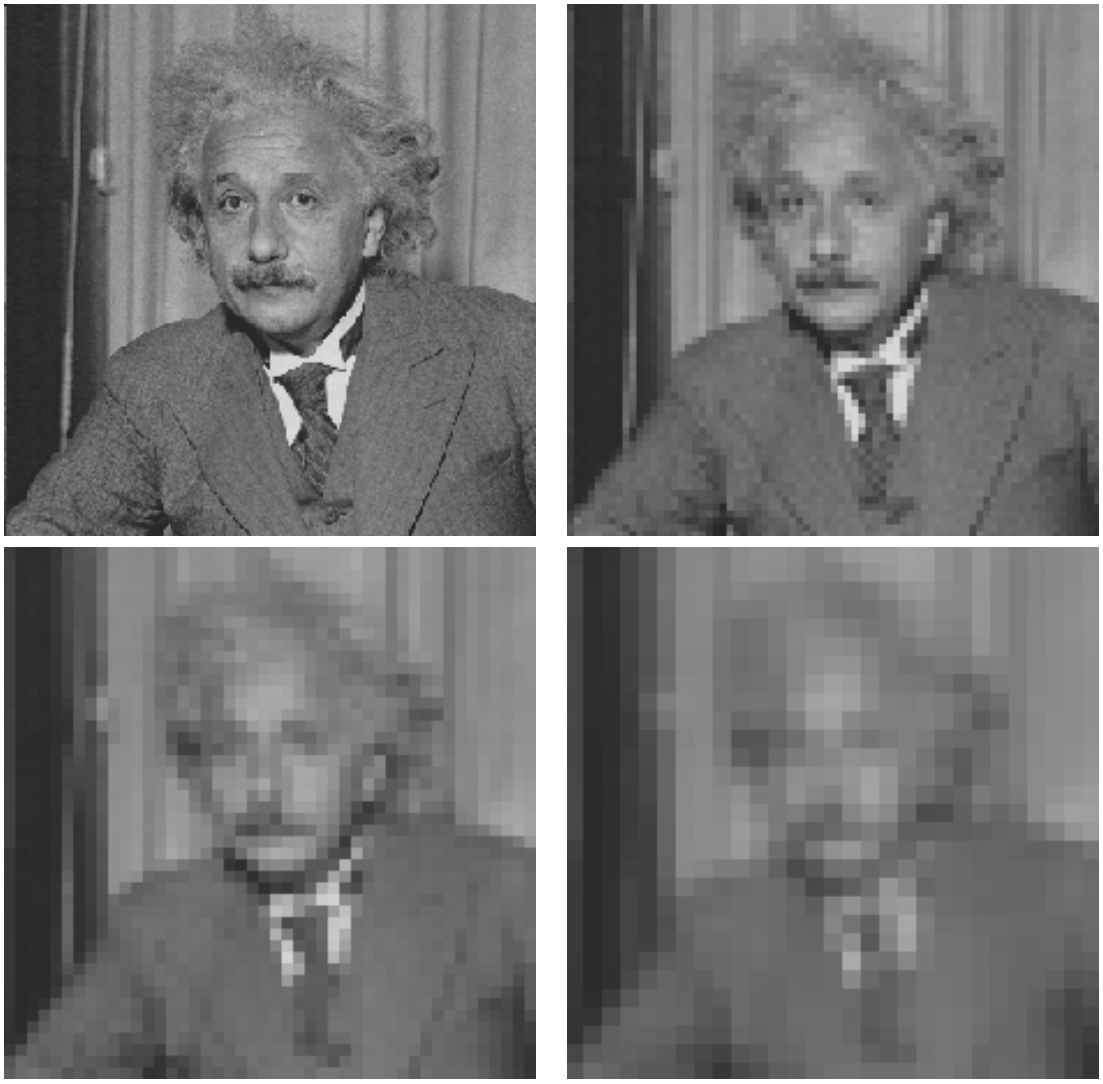


Figure 1.2: Images at different resolutions. Note the pixelization effect.

we have to sample and quantize (digitize) the image function. *Sampling* refers to considering the image only at a finite number of points. And *quantization* refers to the representation of the gray level value at the sampling point using finite number of bits. Each image sample is called a *pixel*. Your typical desktop image scanner does sampling and quantization for you.

One of the simpler scheme is sampling on a *regular* grid of squares. We can visualize the process as overlaying an uniform grid on the image and sampling the image function at the center of each grid square. Finer the grid, better the resolution of the image; coarser the grid, the more is the observed "pixelization" (see Fig. 1.2).

At each pixel (or at each grid square) we usually represent the gray level value using an integer ranging from 0 for black to 255 for fully white.

1.6 In what format are the images stored in a computer?

A digital image is essentially a collection of pixel values. There are various formats of storing an image in a file. Essentially they all involve storing the pixels values along with other relevant information about the image. The image format we will be using is called PGM or Portable Gray Map.

The extended format for a PGM file is given in Appendix of this document. For our example images we have a simplified PGM format. A typical image file will have data as shown below

```
P2
123 74
255
111 114 118 119 118 105  85  71  73  72  66  69  80  89  92  90  92
 93  82  78  92  92  95 100 104 107 108 109 109 112 108 100  92  84
 73  60  50  50  54  54  46  39  42  57  72  86  87  90  94  99 105
109 112 115 113 108 101  92  87  89  93 100 102  91  76  68  55  48
 56  72  76  76  66  54  53  67  81  94  78  56  42  48  68  85  92
 96  97 104 114 115 100  79  64  60  61  66  77  84  78  60  45  49
 42  50  72  83  71  55  49  42  41  39  40  43  43  35  24  45  49
 54  57  58  58 110 114 122 124 121  97  72  53  52  53  49  52  62
```

The first line is a "magic number" for identifying the file type. A pgm file's magic number is the two characters "P2".

The second line contains two numbers denoting the image width and image height, respectively.

The third line has the maximum number of gray levels which is, in our case, 255.

The numbers in fourth line onwards are the pixel values. The number 255 denotes a white pixel and 0 denotes a black pixel. There are a total of (width * height) gray values. The pixels are stored in row-scanned order, starting at the top-left corner of the image, proceeding in normal English reading order.

1.7 How do I "see" images on the computer?

You would be using the PCs in ENB 118 or 116 to see the images on screen. The software on the PCs use a different format to store images. So first you have to convert the PGM image files into PC format and then FTP them over to the PCs and view them. The steps are:

1. Type on suntan:

   ```
   /home/sunburn/cs.dept/sarkar/EEL_4851/convert block.pgm block.pcx
   ```

   This will convert the PGM image `block.pgm` into a PCX formatted image file `block.pcx`

2. ftp `block.pcx` over to the local PC.

3. Use the Paintbrush software to read in `block.pcx` and view it.

1.8 How do I get a print out of the image?

The `.pcx` formatted image can be imported into a Microsoft Word or a WordPerfect document and printed along with the document.

If you want independent print out of the images, then you can again use suntan to convert the PGM file into a Postscript file which can be sent to the printer. The steps are (all on suntan)

```
convert block.pgm block.eps
```

```
lp -d lw118 block.eps
```

2. Assignment1: Find the rocks (Due at 5 pm on Sept 22)

Goal? The goal of this assignment is to familiarize you with manipulating arrays. You will have to write Ada code to read, write and manipulate images, specifically you have to write code to find rocks in the Mars Pathfinder images.

Coding: Your code writing task can be divided into two parts:

1. You have to write a package to implement the image abstract data type using arrays. The ADT should conform to the specifications in Fig. 2.2. The Image ADT should allow you

   (a) to read images from a file and to save images to a file.

   (b) to threshold a image based on an user specified number, N. The thresholding operation is really simple. Given a gray level image, I, you have to create an output image, O, such that for each pixel location (i, j):

   $$O(i,j) = \begin{cases} 255 & \text{if } I(i,j) > N \\ 0 & \text{if } I(i,j) \leq N \end{cases}$$

   (c) to perform the morphological *opening* operation defined by the following sequence of two operations:

   $$O(i,j) = \begin{cases} 0 & \text{if } \mathbf{ALL}\ m \text{ by } m \text{ neighbors of I(i, j) == 0} \\ 255 & \text{otherwise} \leq N \end{cases}$$

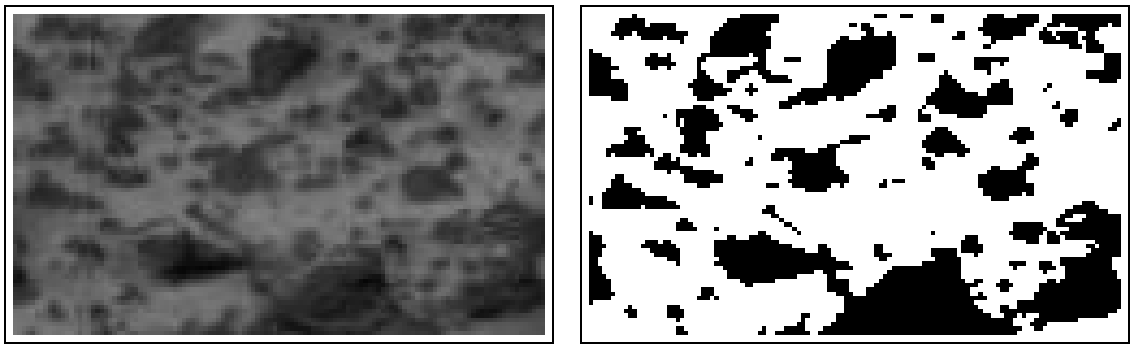
   $$O(i,j) = \begin{cases} 0 & \text{if } \mathbf{ANY}\ m \text{ by } m \text{ neighbors of I(i, j) == 0} \\ 255 & \text{otherwise} \leq N \end{cases}$$

   (d) to return the size of the image.

2. Write a driver program which interacts with the above Image ADT and the user. The user should have options to read in an image file, threshold the image based on an user specified number, and save the thresholded image to a file.

What should I submit? For details about what and how to submit see Chapter 1.

Experiment with thresholding followed by opening on the Mars Pathfinder images so that you can segment out the rocks. The rocks will appear in black in the thresholded image. You will also notice that not all choices of the threshold and the opening neighborhood size (typically $m = 3 or 5$) do a proper job of finding the rocks in the image. In your report, include the original and best thresholded images that show



(**a**) (**b**)

Figure 2.1: (a) Gray level image of Mars. (b) Thresholded image showing the rocks.

the rocks. For, example for the image in Fig. 2.1(a), a good choice of the threshold results in Fig. 2.1(b). Note that the rocks are in black and the background is in white.

```
with text_io;

package Image_Data_Type is

-- This package implements an ADT to represent images

-- Structure:  The image is represented as an array of pixel values

   OUT_OF_BOUNDS : Exception;    -- Raised if an attempt is made to access
                                 -- image pixels outside the image

   type Image_Array is private;
-------------------------------------------------------------------------------
   procedure Read (File : in  Text_IO.File_Type;
                   Image: out Image_Array);
   -- Function:        Reads in the Image from File
   -- Preconditions:   None
   -- Postconditions:  All the image pixels are read in and the image
   --                  width and height recorded in the appropriate fields.
-------------------------------------------------------------------------------
   procedure Save (File : in  Text_IO.File_Type;
                   Image: in Image_Array);
   -- Function:        Write out the Image to the File in PGM format
   -- Preconditions:   None
   -- Postconditions:  The written image file is in PGM format

-------------------------------------------------------------------------------
   procedure Get_Size (Image:in Image_Array;
                       Row_Num: out positive;
                       Col_Num: out positive);

   -- Function:   Returns the total number of Rows and Columns in the Image
   -- Preconditions:   None
   -- Postconditions:  The values are returned through the procedure parameters
```

```
-------------------------------------------------------------------------
   procedure Morphological_Open (In_Image:in Image_Array;
                                 Out_Image: out Image_Array;
                                 m: in positive);

   -- Function: Performs the morphological open operation on (thresholded)
   --           In_Image. The opening neighborhood is of size m by m
   -- Preconditions:   None
   -- Postconditions:  The opened image is returned through the
      procedure parameter.
-------------------------------------------------------------------------
   procedure Threshold (In_Image:in Image_Array;
                        Out_Image: out Image_Array;
                        Threshold_Value: in positive);

   -- Function:Thresholds the In_Image based on the Threshold_Value to produce
   --          an image with two gray levels 0 an 255. Any pixel in
   --          In_Image with gray  level above Threshold_Value is assigned
   --          255 in Out_image and the rest of the pixels in Out_Image is
   --          assigned 0.
   -- Preconditions:   None
   -- Postconditions:  The thresholded image is returned through the
      procedure parameter.
-------------------------------------------------------------------------

private

  type Image_array_type is array(1..250,1..250) of integer;

  type Image_Array is record
    Rows: positive; -- stores the total number of rows in the image
    Cols: positive; -- stores the total number of cols in the image
    Image: Image_array_type;
  end record;

end Image_Data_Type;
```

Figure 2.2: Specification of the Image ADT.

3. Assignment 2: Count the rocks (Due 5 pm on Oct. 13)

Goal: The goal of this assignment is to use stack and queues to automatically count the rocks in the thresholded Mars Pathfinder images. The output images from your first assignment will be the input images for this assignment. You have to basically count the number of connected black regions.

For example, consider the 16 by 16 image shown in Fig 3.1. There are two connected black components. You will mark these pixels in a new blank image with the component number. Thus for the image Fig 3.1, your output will be another image with the pixels in the top-left connected component marked with a value of 1 and the pixels in the bottom-right marked with a value of 2.

Algorithm: You will build the rock labeling algorithm around a seed region growing algorithm. Given a starting point (the seed) the region growing algorithm finds all pixels which have the same gray level value as the seed pixel and connected to it. The idea is to iteratively invoke this seed region growing algorithm for all the rocks. The region growing algorithm can be described as follows.

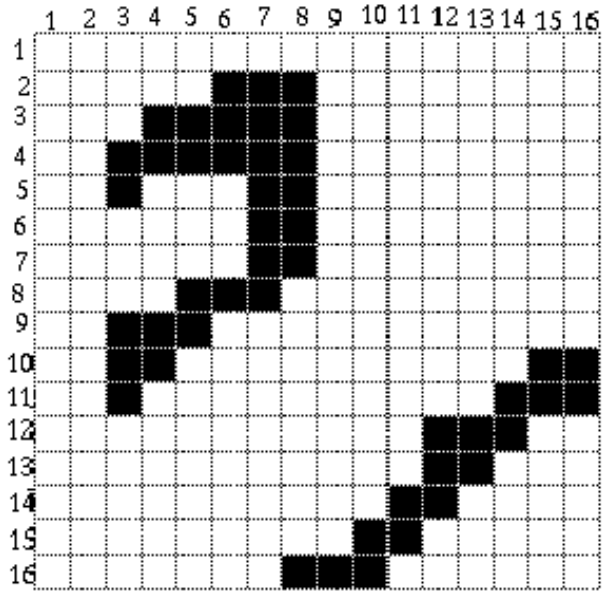


Figure 3.1: An example image with two connected components. The image size is 16 by 16.

Each pixel in an image has 8 neighbors. For example, the pixel (4, 6) has the pixels at locations {(4, 5), (4, 7), (5, 5), (5, 6), (5, 7), (3, 5), (3, 6), (3, 7) } as neighbors. Not all the neighbors have the same pixel value. We start by considering the neighbors which have the same value as the starting pixels. There are two ways to proceed: depth first or breadth first. In depth first search, you recursively follow one neighbor of a pixel. In breadth first search you process all the neighbors of one pixel before proceeding to the next one. The pseudo code of the algorithms are as follows:

```
Breadth_First_Search (Input_Image, Output_Image, Start_x, Start_y, Value)
  Q: Queue of (x, y) coordinates of a pixel
  px, py: integers;
  Initialize Q to null.
  Set all pixels of the Output_Image to 255 (white).
  Enqueue(Q, (Start_x, Start_y));
  while (Q is not null) {
    (px, py) = Dequeue(Q);
    Output_Image(px, py) = Value;
    for each neighbor, (nx, ny), of (px, py) {
       if (Input_Image(nx, ny) == Input_Image(px,  py)) and
          Output_Image(nx, ny) == 255 {
         Output_Image(nx, ny) == -120;
         Enqueue(Q, nx, ny);
        }}}
end Breadth_First_Search;


Depth_First_Search (Input_Image, Output_Image, Start_x, Start_y, Value)
  S: Stack of (x, y) coordinates of a pixel
  px, py: integers;
  Initialize S to null.
  Set all pixels of the Output_Image to 255 (white).
  Push(S, (Start_x, Start_y));
  while (S is not null) {
    (px, py) = Pop(S);
    Output_Image(px, py) = Value;
    for each neighbor, (nx, ny), of (px, py) {
       if (Input_Image(nx, ny) ==  Input_Image(px,  py)) and
          Output_Image(nx, ny) == 255 {
         Output_Image(nx, ny) == -120;
         Push(S, nx, ny);
       }}}
end Depth_First_Search;
```

The above algorithms will mark with "`Value`" all pixels with the same gray level and connected to the pixel at (`Start_x, Start_y`). You have to iteratively invoke the above subroutines to process all the rocks in the image.

What to code? You have to implement the above two search algorithms as a part of the image ADT package. The application should use the generic queue and stack packages from the text book and the image package you developed in the first assignment. The application should have an menu based interface which allows the user to select the search algorithm, to specify the image, to save the marked-up image, and to output the number of connected components, with their sizes.

What to submit? For details about what and how to submit see Chapter 1. Make sure you include outputs which show the states of the Output_Image after 5, 20, 50, 100, and 300 iterations of the depth first search and the breadth first search on the flat-top rock image (`mars2.pgm`) for the big rock in the image.

4. Assignment3: Sort the rocks (Due 5 pm on Oct 27)

Goal: In this assignment you will use linked lists to sort the rocks in the Mars image based on their sizes. Thus, the output of your second assignment is the input for this assignment.

Algorithm: First, store the following information about each connected component you have found in a record: size of the component, centroid of the component, and the individual pixels in the component. Store the individual pixels in each component as a linked list. Next, insert these component records in another linked list so that the components are in sorted order with the fist element being the largest.

What to code? Your task is to design an ADT, let us call it the ComponentADT, with the appropriate structure to store the above information. The structure should be a linked list of records (see Fig. 4.1. One of the field of these records is another linked list. Reuse the linked list package from the book. You might have to modify the list package in the book so that you can also insert in sorted order. Design the ComponentADT so that you can interact with the ImageADT to construct the sorted component list and to save the components. Your program should be able to save the user specified number of largest rocks in an image. Remember to hide your data structure from the driver or user.

What to submit? For details about what and how to submit see Chapter 1.

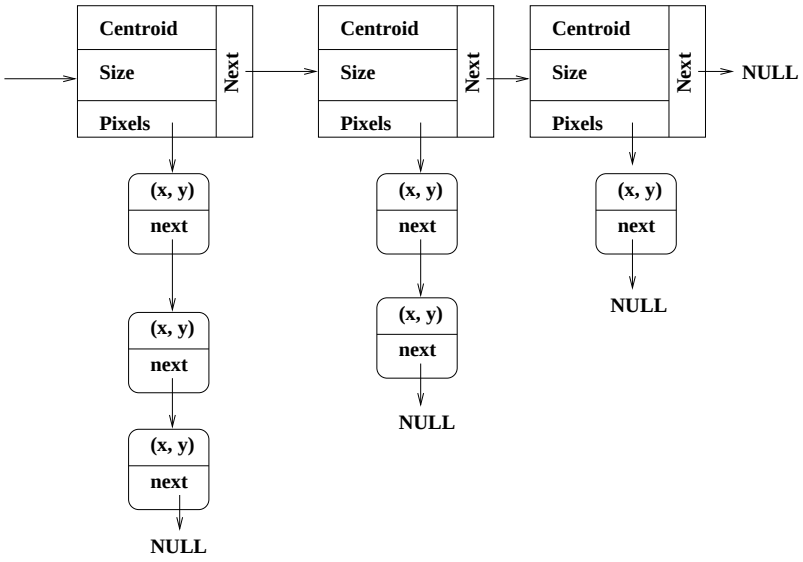


Figure 4.1: Schematic of the ComponentADT structure.

5. Assignment 4: Is there a rock nearby? (Due 5 pm on Nov 17)

Goal: The goal of this assignment is to introduce you to spatial reasoning about the rocks. You would be able to respond to queries like: Is a there a rock within N pixels of location (x, y)?

Algorithm: To efficiently answer such spatial queries you would need a *Point Quad Tree structure*. Nodes in the quad tree represent rocks in the image. Each node has an image coordinate associated with it corresponding to the centroid of the rock represented by the node. (You may, of course, store other information about the rocks such as their sizes and constituent pixels along with coordinate information.) In addition, each nodes has four pointers corresponding to the four directions: North-East (NE), North-West (NW), South-West (SW), and South-East (SE). The children are placed in the tree according to the spatial relationship with respect to the parent.

As an example consider the arrangement of rocks shown in Fig. 5.1. The construction of the point quad tree is shown in Fig. 5.2. The nodes are inserted in order of their

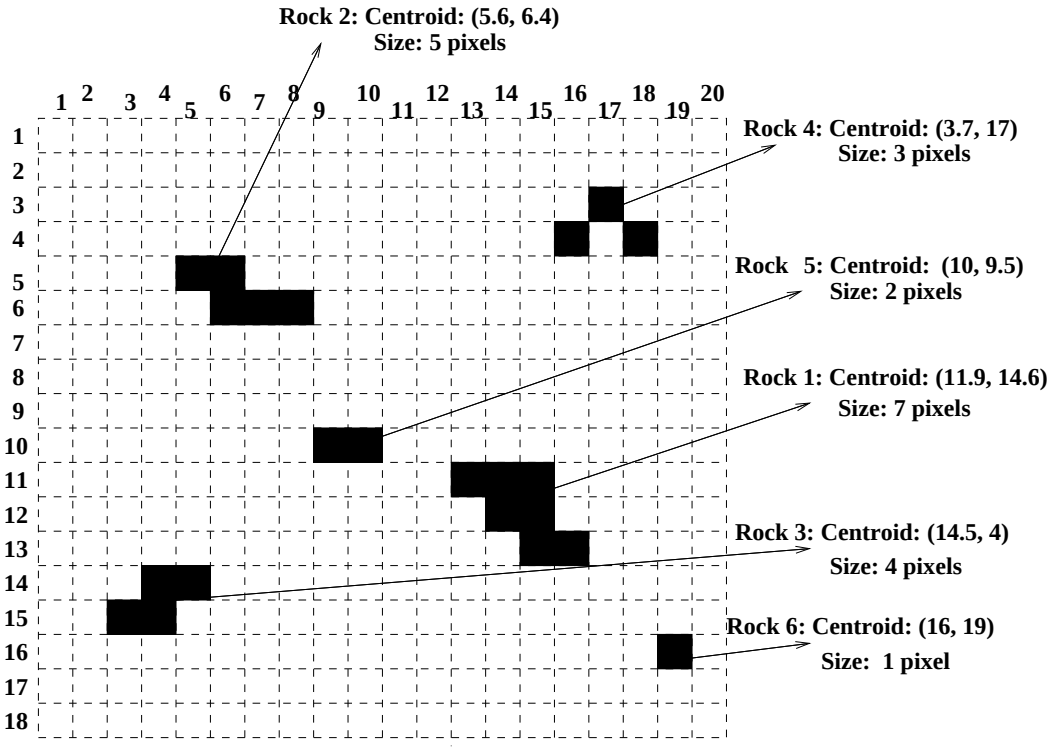


Figure 5.1: An example arrangement of rocks.

size which is the order of the rocks in the ComponentADT of Assignment 3. To search of rock which is with N pixels of a given location, you would just traverse the tree links until you come across a rock whose centroid is within N pixels. As an example, consider a search for a rock within 1 pixel of location (9, 9). You would start at the root (Rock 1) and follow the NW link since the location (9, 9) is North-West of Rock 1. Subsequently you would following the SE link of Rock 2 to arrive at Rock 5 which is the destination.

What to code? You will have to design an point quad tree ADT which support the operations of insert, search, traverse, and print node information.

You should provide a menu driven interface which allows the user to specify an image location and a distance threshold. The output of the program would a "Yes" or "No" with the information about the rock at that location. The user should also be able to save that particular rock as an image file.

What to submit? For details about what and how to submit see Chapter 1.

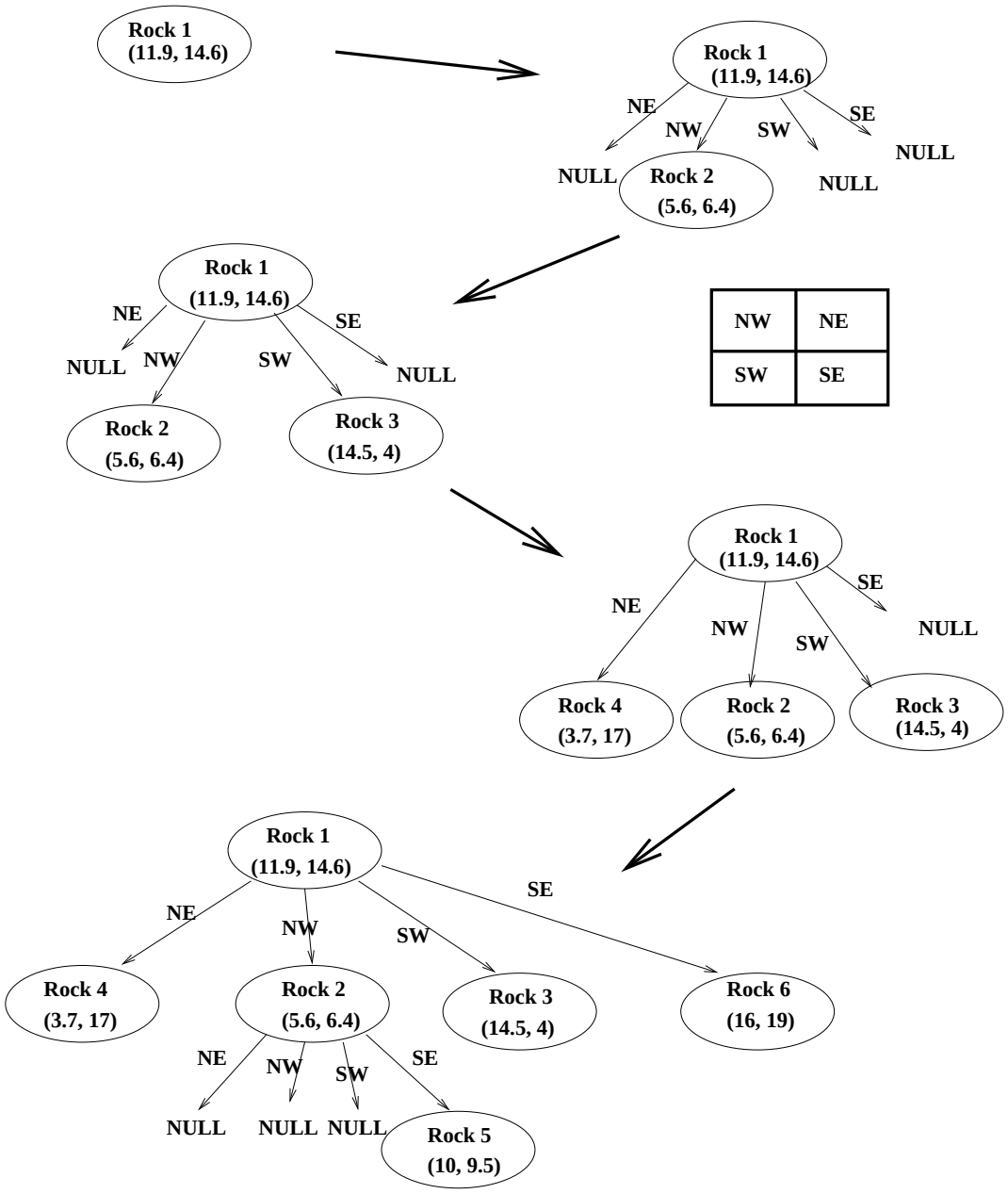


Figure 5.2: Stages through the building of a quad tree.

6. Assignment 5: Can I Jump? (Due 5 pm on Dec 1)

Goal: The goal of this assignment is to introduce you to graph manipulation and spatial reasoning. This assignment will allow you to answer queries about the ability to reach a particular rock from another rock by jumping through intermediate rocks.

Algorithm: We will decide on the ability to go from one rock to the other using graph theory. We will assume that in one jump we can cover at most N pixels (user specified). Let each node in a graph represent a rock. The links between the nodes denote whether we can go from one rock to the other in one jump. Each connected component of the graph determine the subset of nodes amongst whom we can travel.

What to code? Implement the GraphADT with the ability to construct a graph from the ComponentADT of Assignment 3. The GraphADT should support connected component labeling using depth first and breadth first search. You should also be able to save nodes of the graph as an image.

The user would specify the maximum jump distance and your program should be able to print out information about the rocks between whom we can travel. The user should also be able to save these subsets as images in separate files.

What to submit? For details about what and how to submit see Chapter 1.

7. Appendix

7.1 PGM format for images

```
pgm(5)              Headers, Tables, and Macros              pgm(5)


NAME
     pgm - portable gray-map file format

DESCRIPTION
     The portable gray-map format is a lowest  common  denominator
     gray-scale file format.  The definition is as follows:

     - A "magic number" for identifying the  file  type.   A  pgm
       file's magic number is the two characters "P2".

     - Whitespace (blanks, TABs, CRs, LFs).

     - A width, formatted as ASCII characters in decimal.

     - Whitespace.

     - A height, again in ASCII decimal.

     - Whitespace.

     - The maximum gray value, again in ASCII decimal.

     - Whitespace.

     - Width * height gray values, each in ASCII decimal, between
       0  and  the  specified  maximum  value,  separated by whi-
       tespace, starting at the top-left corner of  the  graymap,
       proceding  in  normal English reading order.  A value of 0
       means black, and the maximum value means white.

     - Characters from a "#" to the next end-of-line are  ignored
```

(comments).

- No line should be longer than 70 characters.

Here is an example of a small graymap in this format:

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a graymap.

There is also a variant on the format, available by setting the RAWBITS option at compile time. This variant is different in the following ways:

- The "magic number" is "P5" instead of "P2".

- The gray values are stored as plain bytes, instead of ASCII decimal.

- No whitespace is allowed in the grays section, and only a single character of whitespace (typically a newline) is allowed after the maxval.

- The files are smaller and many times faster to read and write.

Note that this raw format can only be used for maxvals less than or equal to 255. If you use the _ϕp_ϕg_ϕm library and try to write a file with a larger maxval, it will automatically

fall back on the slower but more general plain format.

SEE ALSO

fitstopgm(1), fstopgm(1), hipstopgm(1), lispmtopgm(1), psidtopgm(1), rawtopgm(1), pgmbentley(1), pgmedge(1), pgmenhance(1), pgmhist(1), pgmnorm(1), pgmoil(1), pgmramp(1), pgmtofits(1), pgmtofs(1), pgmtolispm(1), pgmtopbm(1), pgmtops(1), pnm(5), pbm(5), ppm(5)

AUTHOR