COL100 Assignment

Part 1

Magic List

Goal

The goal of this assignment is to get some practice with data structure implementation and usage in python. In Part (a) of the assignment you have to complete some functions of the Magic List data structure. In Part (b) of the assignment you have to use a Magic List to solve the given problem.

Part (a) - Magic List

Magic List is a special type of list L in which the elements are arranged according to the following properties :

1. The first element of the list is always 0 (L[0] = 0).

2. For each index i (>= 1), L[i] <= L[2i] and L[i] <= L[2i+1].

3. For each index i (>= 1), L[i/2] is called the parent of L[i], and L[2i], L[2i+1] are called the children of L[i].

4. L[1] has no parent.

In this list, all the elements besides the first one are called magic elements. The size of Magic List is the number of magic elements in the list. Observe that L[1] will be the smallest magic element in list (Due to point 2. above).

A class called MagicList has been given to you (inside MagicList.py). You have to complete the following functions inside the file :

1. findMin

Complete the findMin function and return the smallest magic element in the Magic List. If there are no magic elements (i.e. size of Magic List is zero), then return None.

2. insert

Complete the insert function and return the modified list. To insert an element in the Magic List, first append the element to the end of the list. Start at last element and keep comparing with parent L[i/2]. If parent is larger, swap the parent and the element. Keep doing this till parent is smaller, or the element has no parent.

3. deleteMin

Complete the deleteMin function, delete the smallest magic element in the Magic List, and return the modified list. To delete the minimum element, swap the minimum element (E1) with the last element (E2) and remove the last element. Keep comparing E2 with children L[2i] and L[2i+1] and swap with the smaller child. Keep doing this till both children are greater than E2 or E2 has no more children.

Part (b) - k-sum

In this part, you have to complete the function k-sum. The function takes a list L and an integer K as input, and you need to find the sum of smallest K elements of the list. You need to do this with the help of a Magic List.

To convert the input list L into a Magic list, you can start with an empty MagicList M and insert all the elements of L into M (Code given below). Then use M to find the sum of smallest K elements and return the sum. Note that K <= size of L

```
1. # Convert a list L to Magic List M
2. M = MagicList ()
3. for i in L :
4.     M. insert ( i )
5. # M now contains all elements of L
```

Testing

The given code contains a few test cases for each function. You need to ensure that each test case passes correctly. Do not change anything in the code except completing the given functions.

Submission

You need to submit completed MagicList.py on moodle. Grading would be done on the basis of a demo (viva). You should be able to run your code using python3 during the demo.

What is allowed? What is not?

1. This is an individual assignment. 2. Your code must be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems. 3. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.

Part 2

IMAGE EDITOR

For this assignment you will design a simple image editor which performs a few basic tasks on an input image. The particular tasks we will be interested in here are averaging and performing edge detection of an image. Given an image you will first read the image into an array, perform some basic operations and generate the output image.

Image files will be provided in pgm (portable graymap format). These image files have the following specifications. When you open images in pgm format using any text editor, you will see the following. The first line contains an identifier for the image which for this assignment will be P2. The next line contains dimension of the image with two numbers indicating the width and the height respectively. Next number specifies the maximum allowed pixel value of the image, for this assignment it will be 255. Next we have the actual pixel values of the image given in row major ordering. That is the first pixel value corresponds to location (0,0), the second for location (0, 1) and so on. For all pixels in the image, we have their value between 0 and 255. The number of pixel values equals width (W) times

height (H) of the image. You will be provided a function to read an image from a file and write an image to a file.

Write functions to perform the following tasks.

- **Averaging filter:** Create an image of the size WxH using an array of integers. Pixel value at location (i,j) of the image equals the average of the pixel values at locations (i − 1, j − 1), (i − 1, j), (i − 1, j + 1),(i,j−1),(i,j),(i,j+1),(i+1,j−1),(i+1,j),(i+1,j+1) of the input image.

  Formally pixel value at location (i, j) is

  (image[i-1][j-1]+image[i-1][j]+image[i-1][j+1]+image[i][j-1]+image[i][j]+image[i][j+1]+image[i+1][j-1]+ image[i+1][j]+image[i+1][j+1])/9

  Notice that the above formulation is undefined for i = 0 and i =H−1 and similarly for j = 0 and j =W−1. For these boundary conditions, assign pixel value at location (i, j) =image[i][j].

  Write this image to the file 'average.pgm'. You can view the file average.pgm in any image viewer.

- **Edge detection:** The idea here is to compute some function of pixel values among neighbouring cells in horizontal and vertical directions. This function approximates how close these pixel values are relative to each other. If pixel values in a neighbourhood are similar, then this function value would be very small or 0 where as if there is a significant change in neighbourhood pixel values, then this function would have non-zero values. We detect presence of an edge by noticing how this function value changes. The above is for intuition on how this works. Follow the steps mentioned below to detect edges of an image and output image to 'edge.pgm' file.

  Compute values of grad(i,j) = sqrt(hdif(i, j)*hdif(i,j) + vdif(i, j)*vdif(i,j)) for all i, jin the image, where hdiff and vdiff are the horizontal and vertical gradients and are computed as follows:

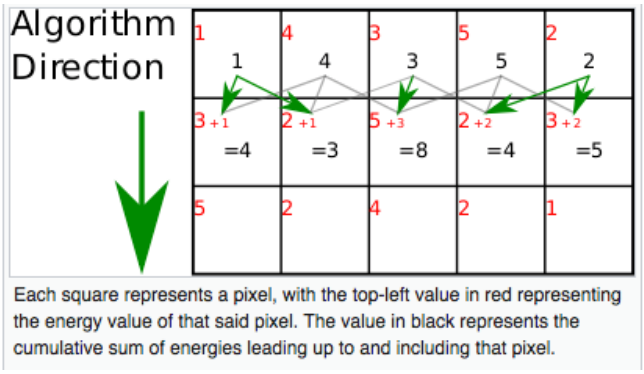
  hdif(i,j) = (image[i-1][j-1]-image[i-1][j+1]) + 2(image[i][j-1]-image[i][j+1]) + (image[i+1][j-1]-image[i+1][j+1])

  vdif(i,j) = (image[i-1][j-1]-image[i+1][j-1]) + 2(image[i-1][j]-image[i+1][j]) + (image[i-1][j+1]-image[i+1][j+1])

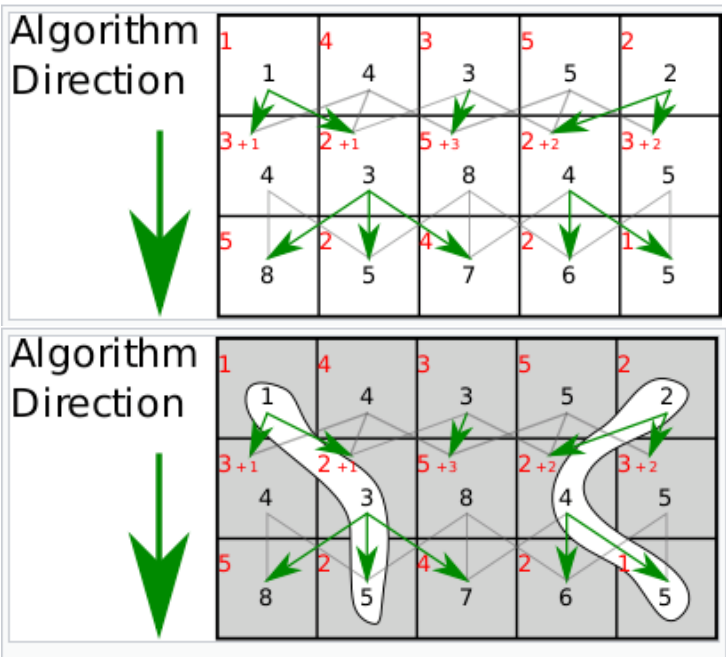
- **Path of least energy:** This part of the assignment requires you to implement an algorithm that finds a path from top of the image to the bottom, which has the least energy. Consider that your edge image as computed above gives the energy image. Now you have to compute the path of minimum energy and color that path with white (or 255 value). Start from the top row of the image and compute the energy of the pixels for the next row as follows. Please note that for the top row the pixels of MinEnergy image will have the same value as edge image.

  MinEnergy(i,j) = edge(i,j) + min(MinEnergy(i-1,j-1), MinEnergy(i-1,j), MinEnergy(i-1,j+1))

  As below is the pictorial view of the computation.



Each square represents a pixel, with the top-left value in red representing the energy value of that said pixel. The value in black represents the cumulative sum of energies leading up to and including that pixel.



The last row shows two min values (=5) and the paths with lowest energy are shown in white. Please note that you have to trace the path back from the min value in the last row all the way up and change the pixel value as 255 in the image.