ARK Task Round Documentation

Millon Madhur Das

***Abstract*— This document compiles my attempts for the task round of ARK.**

In the last 40 days, I went from learning how to print in C++(ofcourse using cout, bkz C styled printf is boring) to optimizing a program running in NP time to polynomial time, making a face detection game in python, and exploring the beauties of graph theory.

I also learnt a lot about Recurrent Learning, DQN, Greedy Algo. Although I didn't get sufficient time to implement, it has given me ample motivation to keep exploring this field.

I. TASK 2.1 Optimise Me

INTRODUCTION

The task involved optimizing code with recursive functions and matrix multiplications. I tried both recursive and iterative methods of which the latter failed to scaling.

Problem Statement

costMatrixA and costMatrixB are two n x n matrices given to you. Each cell in these matrices contains the cost you have to pay on landing on that cell.

There are basically 3 parts to the problem statement:

1. FindMinCostA() and FindMaxCostB(): FindMinCostA() which returns the minimum cost of going from cell i,j to cell n,n in costMatrixA. This cost is the sum of the costs of the cells of costMatrixA which will be on your path from i,j to n,n. Similarly, FindMaxCostB().

2. productMat: where, productMat[i][j] stores the value of FindMinCostA(i,0)*FindMaxCostB(0,j,n) + FindMinCostA(i,1,n)*FindMaxCostB(1,j,n).... + FindMinCostA(i,n-1,n)*FindMaxCostB(n-1,j,n)

3. Filter: Filter's dimension is c x n, then we replace the dot product of this filter corresponding c rows of prductMat with a single element in a new matrix whose dimension is (n/c) x 1.

INITIAL ATTEMPTS

Initially, I tried the iterative approach. I used dynamic programming, i.e. DP(i,j) = min(DP(i-1,j), DP(i,j-1)) or max(DP(i-1,j), DP(i,j-1)) for the functions `FindMinCostA()` and `FindMaxCostB()`, respectively. DP(i,j) denotes the min/max cost to reach the (n,n) element of the costMatrix starting from (i,j) position.

Using this logic I called the functions repeatedly to return the min/max cost which was stored in another matrix, i.e. `proMatrixA` and `proMatrixB`.

In the template code provided, productMatrix involved calling the functions multiple times to compute the product. Storing the values in `proMatrix` helped speed up the process because accessing the value from memory is much more efficient than computing it using these expensive functions. Also each element was computed more than once. Notice the optimized version of the code needs only one function call to FindCost.

```
//creating productMat as explained in the beginning
for (i = 0; i < sizen; i++)
    {
        for (j = 0; j < sizen; j++)
        {
            for (k = 0; k < sizen; k++)
                productMat[i][j] += FindMinCostA(i,
   k, sizen) * FindMaxCostB(k, j, sizen);
        }
    }
```

Listing 1. Original Implementation

```
//Filling proMatrices
    FindMinCostA(i,j,sizen);
    FindMaxCostB(i,j,sizen);

    //creating productMat from already computed
    proMatrices
    for (i = 0; i < sizen; i++)
    {
        for (j = 0; j < sizen; j++)
        {
            for (k = 0; k < sizen; k++)
                productMat[i][j] += proMatrixA[i][k
    ] * proMatrixB[k][j];

        }
    }
```

Listing 2. Optimized Implementation

The time complexity of the most expensive functions are as follows:

`FindMinCost`: Reduced from O(2^{n^2}) to O(n^2)

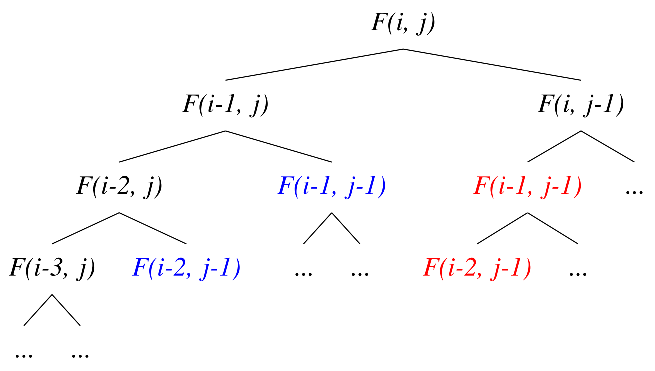
`proMatrix`: Reduced from O(n^3) to O(n^2)

Final Approach

1. Optimizing FindMinCost

Snipping the recursion tree by storing values as we find them reduced the execution time significantly. Using multiple threads to compute FindMinCostA() and FindMaxCostB() simultaneously further improved it.

Let call FindMinCostA(i,j,n) as F(i,j). Notice the leaves marked in red, which are already computed by previous recursive function calls(in blue).



Recursion tree for Cost function

```
//all elements of proMatrix are initialized as -1
FindMinCostA(int i, int j, int n)
{
    //going out of bounds
    if (i >= n)
        return 0;
    //going out of bounds
    if (j >= n)
        return 0;

    if (proMatrixA[i][j] != -1)
        return proMatrixA[i][j];

    //going down or right
    proMatrixA[i][j] = costMatrixA[i][j] + min(
    FindMinCostA(i + 1, j, n), FindMinCostA(i, j +
    1, n));
}
```

Listing 3. Optimized Implementation

2. Optimizing proMatrix

Same as initial approach.

2. Optimizing filterArray

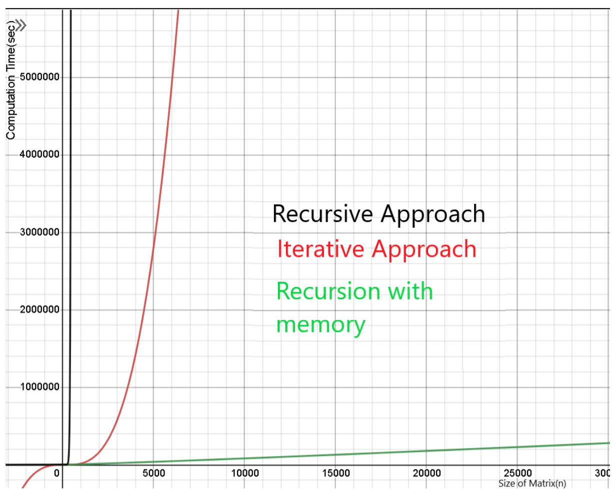
For this part I could only write the trivial solution.

```
for (i = 0; i <= sizen - 4; i += 4)
    {
        long long sum = 0;
        // dot product of 4xn portion of productMat
        for (j = 0; j < sizen; j++)
            for (int filterRow = 0; filterRow < 4;
    filterRow++)
                sum += productMat[i + filterRow][j
    ]*filterArray[filterRow][j];
        finalMat[i / 4] = sum;
    }
```

Listing 4. Filter

Results and Observation

The runtime of the approaches is compared in the graph below.



Future Work

The major bottleneck is the computation of productMatrix. I tried to look into CUDA code to vectorize matrix multiplication. I was not able to learn due to time limitations. I suspect it can further optimize the code.

I also tried parallelization using OpenMP [1] . It basically uses multiple threads in a CPU to process more than one unrelated chunks of code.

```
 #pragma omp parallel
{
  #pragma omp sections
  {
      #pragma omp section
      FindMinCostA(i,j,sizen);

      #pragma omp section
      FindMaxCostB(i,j,sizen);
    }
 }
```

Listing 5. OpenMP parallel processing code to compute both FindCost functions

I also tried to use a different random number generator function(Permuted Congruential Random Number Generator) [2] which is about 5x faster than our good old rand() function. Although I have implemented it I didn't include it because I don't understand it completely.

CONCLUSION

Optimisation is absolutely critical in real-time applications used in ARK. This task also made me explore how code is actually run on hardware. As image processing and decision trees can get very resource intensive, it is necessary to optimize the code to run on embedded system architectures.

REFERENCES

[1] Parallelisation using OpenMP, https://people.sc.fsu.edu/~jburkardt/cpp_src/multitask_openmp/multitask_openmp.html
[2] Permuted Congruential Random Number Generator(PCG32fast), https://en.wikipedia.org/wiki/Permuted_congruential_generator
[3] Diagonal traversal of matrix,used in Iterative approach, https://www.geeksforgeeks.org/zigzag-or-diagonal-traversal-of-matrix/

II. TASK 2.2 Face Detection Game

INTRODUCTION

For this task we had to design a game environment similar to single-player pong, with the paddle is replaced with our head.

INITIAL ATTEMPTS

My initial approach is same as final for this task.

Final Approach

Let's begin with some pseudo code:

```
initialisations

def playerCollision(head):
  code...

def wallCollision():
  code...

def frameUpdate(img, dt, head):
  code...

faceDetection(haar cascades)

main loop
```

Listing 6. Pseudo Code

initialisations : This contains the various initial conditions like the position of the ball, its velocity, time frame, etc.

```
#initial contitions
pos = [36,36]
vel = [8,12]
dt = 1

frame = 0
frame_col = 0

game_stat = False
```

Listing 7. initialisations

playerCollision(): This function is responsible for detecting the collision b/w the ball and head. It also changes the velocity of the ball as required. It takes position of the head as input.

```
def playerCollision(head):
    dist = math.sqrt((head[0]-pos[0])*(head[0]-pos
    [0])+ (head[0]-pos[0])*(head[0]-pos[0])) #
    distance bw player and ball
    global frame_col
    if (dist <= 35 + head[2]):
        frame_col = frame
        vel[1] = -vel[1]
```

Listing 8. playerCollision

wallCollision(): This function detects the wall collisions and and changes the velocity of the ball accordingly. It also checks the game status, i.e. when the ball crosses the lower wall, it exits the game.

```
def wallCollision():
    global game_stat
    if (pos[0] + 35 >= 640) or (pos[0]-35 <= 0):
        vel[0] = -vel[0]
    if (pos[1]-35 <= 0):
        vel[1] = -vel[1]
    if (pos[1] + 35 >= 480):
        game_stat = True
```

Listing 9. wallCollision

frameUpdate(img, dt, head): The main loop calls this function for each frame of the game, which in turn calls playerCollision() and wallCollision().

```
def wallCollision():
    def frame_update(img, dt, head):
    global pos
    global frame
    global frame_col
    frame += 1
    wallCollision()
    if((frame - frame_col) > 25):
        playerCollision(head)
    pos[0] = pos[0] + vel[0]*dt
    pos[1] = pos[1] + vel[1]*dt
    cv2.circle(img, (pos[0],pos[1]), 35, (0, 0,
    255), -1) #the ball
```

Listing 10. wallCollision

faceDetection: I used OpenCV haar cascades [1] to detect the face of the played. It is basically a ML based algorithm which detects certain features sequentially to detect faces.

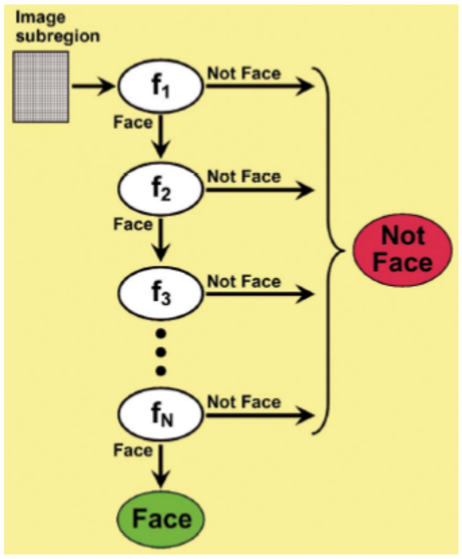


Fig. 1. 'cascades' of feature matching [1]

main loop: This combines all the data and renders the frames using OpenCV.

```
# Load the cascade
face_cascade = cv2.CascadeClassifier(cv2.data.
    haarcascades + 'haarcascade_frontalface_default
    .xml')

# To capture video from webcam.
cap = cv2.VideoCapture(0)

while True:
    # Read the frame
    _, img = cap.read()
    img = cv2.flip(img, 1)

    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect the faces

    faces = face_cascade.detectMultiScale(gray)

    # marking detected face with circle
    for (x, y, w, h) in faces:
        cv2.circle(img, (x+w//2, y + h//2), int((w+
    h)/3.5), (255, 0, 0), 2)

    # coordinates of centre of face
    head = [x+w//2, y + h//2, (int((w+h)/3.5))]

    # updating frame
    frame_update(img, dt, head)

    if(game_stat == True):
        exit(-1)

    # display
    cv2.imshow('img', img)

    # Stop if escape key is pressed
    k = cv2.waitKey(30) & 0xff
    if k==27:
        break

# Release the VideoCapture object

cap.release()
```

Listing 11. faceDetection and main loop

Results and Observation

Although the game works fine most of the time, it sort of gets confused when there is faulty face detection the ball jitters around.

Future Work As evident from the above discussion, the collision response is not very accurate. Although I tried to use actual physics equations, it had worser performance due to tunneling(missing the collision between two frames). A better approach would be to use Continuous Collision Detection instead of Discrete Collision Detection.

Also, due to lack of time there is no used interface like start button, score, game over, etc, which I would love to add in my free time.

CONCLUSION

Being able to manipulate images would be very important for controlling the aerial robot.

REFERENCES

[1] Face Detection using CV haarcascades, https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-9

[2] Drawing basic shapes in CV, https://docs.opencv.org/3.4/d3/d96/tutorial_basic_geometric_drawing.html https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#gaf10604b069374903dbd0f0488cb43670

[3] Collision Detection and response, https://www.youtube.com/watch?v=eED4bSkYCB8, Building Collision Simulations: An Introduction to Computer Graphics

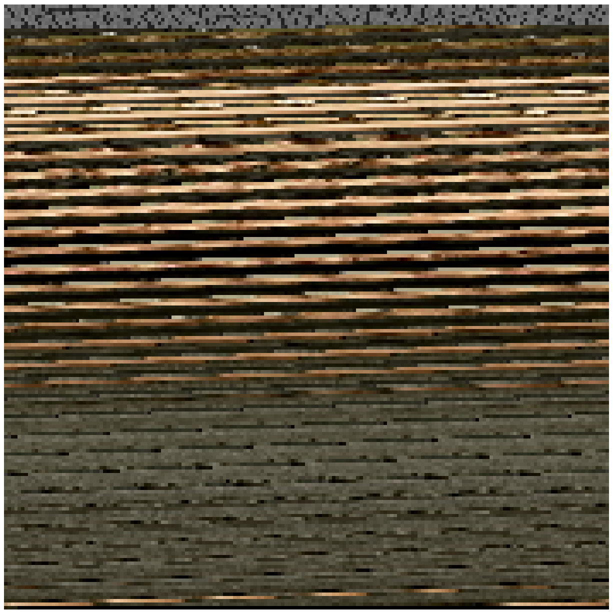
III. TASK 3.1 Path Planning and CV Puzzle

INITIAL ATTEMPTS

My initial approach is same as final for this task.

Final Approach

This task was quite unique in which one thing lead to another, so it's better that I narrate it like a story.



Reading Level1.png in grayscale and converting the pixel values to ASCII reveals the instructions for the next levels.

```
img = cv2.imread('Level1.png', cv2.IMREAD_GRAYSCALE
    )
for x in range(0, 177):
    for y in range(0, 177):
        file1.write(chr(img[x][y]))
```

Listing 12. level1

Looping through all the pixels and writing the found instructions in a file.

```
'Congrats on solving the first level of this task!
    You were able to figure out the ASCII text from
     the image, but wait! Your journey is not yet
    over. After the end of the text you will find a
     (200, 150, 3) coloured image. This image is a
    part of the bigger image called "zucky_elon.png
    ". Find the top left coordinate (Image
    convention) from where this image was taken.
    The x coordinate represents the colour of a
    monochrome maze hidden in an image with
    coloured noise. Find the maze and solve the
    maze using any algothrim like dfs but better.
    Try comparing them and seeing how they perform
    like A*, RRT, RRT* for example. Once the maze
    is solved you will see a word. This word is a
    password to a password protected zip file which
     contains a png. Note that the password is case
     sensitive and all the aplhabets in the
    password will be capital letters This is your
    treasure. To open the treasure you need to
    convert the image in to an audio file in a
    simple way like you did for this ASCII text.
    Once converted, open the .mp3 file and enjoy
    your treasure, you deserved it! A part of the
    image "zucky_elon.png" will begin immediatly
    after the colon, image-lv2 :'
```

Listing 13. Instructions

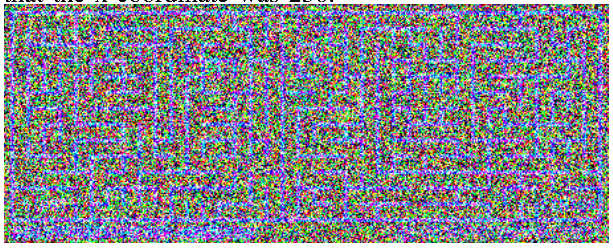
On masking the instructions out, and reshaping the image to a (200, 150, 3) image, I found a cropped out photo of Mark Zuckerberg, which is part of a bigger image 'zucky-elon.png'

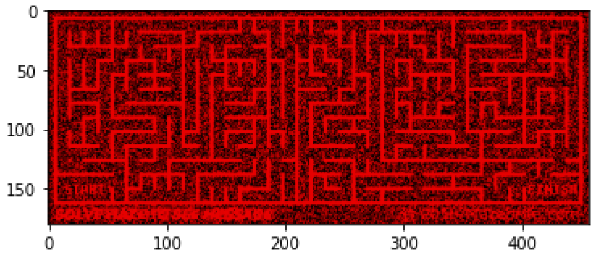
```
level2 = img.copy()
#MASK
```

```
mask = np.zeros((177*177, 3), dtype = bool)
for x in range(leng + 1, leng + 200*150 + 1):
    mask[x][0] = True
    mask[x][1] = True
    mask[x][2] = True

level2 = level2[mask,...]

level2Final = np.reshape(level2, (200, 150, 3))
```

Listing 14. level1

On matching the level2 image with zucky-elon.png I found that the x-coordinate was 230.



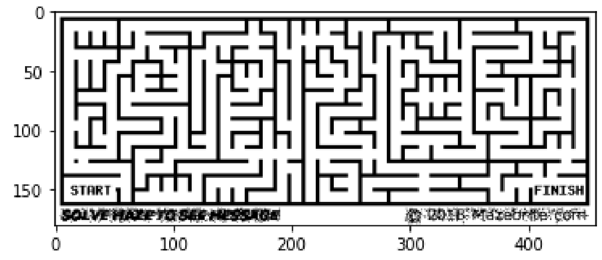


I solved the rest part of the problem on Google Colab because my Linux VM bailed out on me. For some reason(which I am not aware), OpenCV on Colab had image matrix as RGB instead of BGR.

By applying basic RGB filters, I was able to get a rough image of the maze. On filtering it using the obtained x-coordinate value, a cleaner image of the maze was obtained.

```
for x in range(0,180):
  for y in range(0,457):
    if (img[x][y][0] != 230):
      img[x][y][0] = 255
      img[x][y][1] = 255
      img[x][y][2] = 255
    if (img[x][y][0] == 230):
      img[x][y][0] = 0
      img[x][y][1] = 0
      img[x][y][2] = 0
```

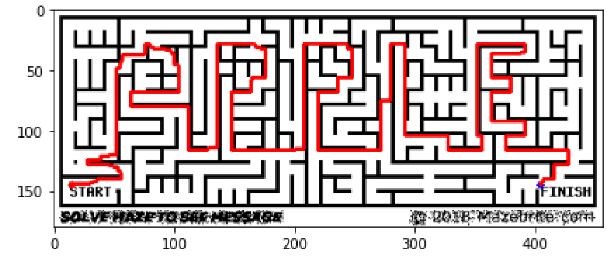
Listing 15. level1



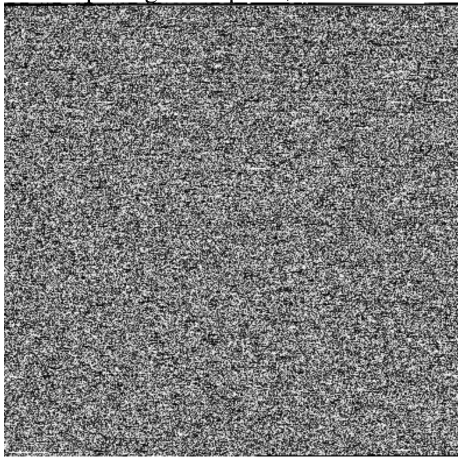
Due to lack of time, I wasn't able to implement path finding algorithms properly. So, I just solved the maze bruteforcing.

Later I was able to find an implementation of Dijkstra's algorithm online [1] which I tweaked a bit to work for this maze image.

The password is 'APPLE'.



On opening the zip file, I found this image.



On converting the pixels of the image to ascii, I got a lot of garbage text beginning with ID3.

I also noticed Rick Rolled!!!

I read a lot about ID3 tags, mp3 file structure, etc. Still I couldn't figure out a way to convert the image to mp3. I even tried opening other mp3 files in Sublime text to find patterns. As binary files open as hex in Sublime text, I even tried to convert the image to hex and write it to a file. Still it didn't match with mp3 files(there was a certain pattern of spacing in these files, which ofcourse wasn't present in the treasure mp3.)

Finally I converted the image to an array and fed it to bytearray and wrote it to a binary file.

Also, I think its worth mentioning that this task was a very cool bait that led me to Rick Astley's music video.

Future Work

I couldn't implement a few path finding algos and compare them, but I would love to implement them after this overloaded semester.

CONCLUSION

Being able to manipulate images would be very important for controlling the aerial robot. Path finding has many uses cases too. It was really cool to know that we use graph traversal algorithms like A*, BFS in Google Maps, etc,

References

[1] Dijkstra's Maze solver,
`https://towardsdatascience.com/solving-mazes-with-python-f7a412f2493f`

[2] Sebastian Lague A* Pathfinding Playlist,
`https://www.youtube.com/watch?v=-L-WgKMFuhE`

[3] , Introduction to Graph Theory Playlist(by Reducible),
`https://www.youtube.com/watch?v=LFKZLXVO-Dg`

[4] MP3 File Structure,
`https://en.wikipedia.org/wiki/MP3`

[5] ID3 Tags,
`https://en.wikipedia.org/wiki/ID3`