Pattern Recognition

ASSIGNMENT REPORT

劉　立坤

学籍番号：2IE19337P

Contents

[1. Assignment contents 2](#_Toc10124215)

[2. Constrains 2](#_Toc10124216)

[2.1 Explicit constrains 2](#_Toc10124217)

[2.2 Implicit constrains 2](#_Toc10124218)

[3. Design ideas 2](#_Toc10124219)

[2.1 Method 2](#_Toc10124220)

[2.2 Data structure 3](#_Toc10124221)

[4. Program design 4](#_Toc10124222)

[4.1 Program Diagram 4](#_Toc10124223)

[4.2 Modules’ functions 4](#_Toc10124224)

[5. Code details 5](#_Toc10124225)

[5.1 Constants and Variables 5](#_Toc10124226)

[5.1.1 Global Variables 5](#_Toc10124227)

[5.2 Program description 5](#_Toc10124228)

[5.2.1 Validation verification function 5](#_Toc10124229)

[5.2.2 Solution judgment function 5](#_Toc10124230)

[5.2.3 main 6](#_Toc10124231)

[6. Operating results 9](#_Toc10124232)

# Abstract

In this assignment, we’ve reimplemented step 01,02,04 and 05 provided by sample program in python. In step 01, we’ve changed the original read one process 1 into read all the images including both training samples and test samples into Numpy array. We then reshape the image into a single row vector so that the whole dataset became a We’ve combined step 02 and step 04 into a single python package ‘image\_processors’, in this package we’ve implemented 2 simple functions: intensity attenuator and binarizer which both directly return the result as a NumPy array. In step 5, we’ve changed the distance from Manhattan distance to Euclidian distance which greatly increase calculation complexity (To measure the time, we’ve also changed the sample code to calculate Euclidian distance via Numpy’s linear algebra package). We then store the result in a distance matrix and then sort each sample’s distance array to find the nearest neighbor or k nearest neighbor.

# Implementation details

## data\_loader

In this module, we firstly read each image in the datasets and reshape the image into a (1,784) Numpy array which then form a (datasize,784) matrix through append method. Meanwhile, we’ve also split the file name and extract the label from file name then store it in the label array. This module returns 4 Numpy arrays: Xtr for training images, Ytr for training images’ labels, Xte for test images and Yte for test images’ labels.

## intensity\_attenuator

This module accepts a Numpy array and an integer ranges from 0 to 100 as strength, then multiply to the Numpy array.

## binarizer

This module accepts a Numpy array and a threshold value. For any point in data array, if the value is greater than threshold then the value is set to 255 else set to 0.

## k\_nearest\_neighbor

This module takes data array and label array

# Design ideas

## 2.1 Method

In this assignment, we’re going to take a reverse back track method. Since the final state is all animals on the right side, we’re going to reverse the transportation process and start from the ending and try to move animals from the right side to the left side. Since the last transportation has to be a fully loaded one with 2 animals on board, in total, there’s only 3 nodes to start from. After right side is empty, the transportation process is complete.

## 2.2 Data structure

The whole process can be represented with a free tree structure as follow:

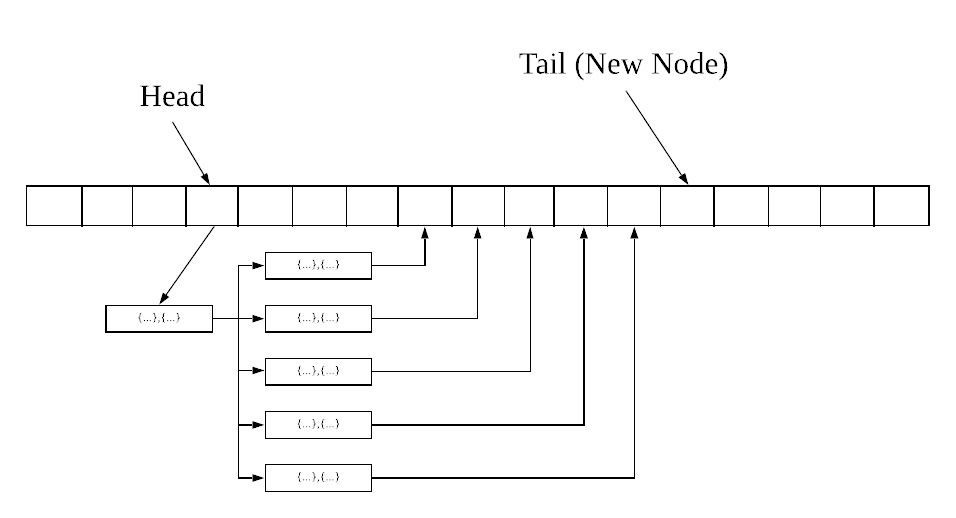
A screenshot of a cell phone

Description automatically generated

Before generating each node, we’ll first check the validation status of its parent node:

1. Whether there’s enough animal left to proceed with the designated operation.
2. Whether wolves on either side of the shore is greater than the rabbits.

In practice, we use queue to generate and store the tree:



For each node in the tree, we find out if it matches the condition to generate a new node, if so then we store the new node in the queue. We’ve setup a dead loop to generate this tree and after each new node, we’ll check if this node is the final state and if reaches end stage, will break this dead loop and print out (depth - 1) of the current node.

# Program design

## Program Diagram

A close up of a logo

Description automatically generated

## Modules’ functions

|  |  |
| --- | --- |
| Module | Function |
| is\_valid | Accept current node’s parent node’s index and the queue, judging whether parent is valid to have a child under the constrain that rabbits count on both sides is either 0 or greater than the wolves. |
| is\_found | Accept current node’s index and the queue to find out whether the condition is satisfied. |
| main | Driver function holds the entire loop |

# Code details

## Constants and Variables

### Global Variables

Global Variables

|  |  |  |
| --- | --- | --- |
| Variable Name | Type | Function |
| head | int | The head of the queue |
| tail | int | The tail of the queue |
| queue | struct node | Storing student data:  int RWo: wolves on the right side  int RRa: rabbits on the right side  int op\_flag: an integer with value of 0 or 1 indicate allowed movements  int depth: an integer indicate current tree’s depth. |

## Program description

### Validation verification function

Input: node’s index, pointer of the queue

Return: 0 or 1

Description:

This function takes one node as parameter and verify if the node itself is valid, if not, the function will return 0 and not allow this node to generate a child.

### Solution judgment function

Input: node’s index, pointer of the queue

Return: 0 or 1

Description:

This function takes one node as parameter and check if it satisfies the goal we’re trying to reach. If so, print out the step and return 1, if not, return 0.

### main

Input: none

Return: 0

Description:

This is the main driving function. It firstly set the root’s parameters, which in this case is queue[0]. Then execute a dead loop to iterate though a series of if statement to perform the operations.

This part of the code performs the move 2 wolves forward operation.

This part of code performs the move 2 rabbits forward operation.

This part of code performs move 1 wolf and 1 rabbit forward operation.

This part of code performs move 1 wolf backward operation.

This part of code performs move 1 rabbit backward operation.

 This part of code performs move 1 wolf and 1 rabbit backward operation.

This part of code performs move 2 wolves backward operation.

This part of code performs move 2 rabbits backward operation.

# Operating results