UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

Organização e Arquitetura de Computadores 2

ANDREAS HUKUHARA CHRISTE - 13673110

NICOLAS PEREIRA RISSO VIEIRA - 13672262

PEDRO PALAZZI DE SOUZA - 13748012

Exercício Programa 1

São Paulo

2023

Introdução

O exercício programa 1 tem como objetivo a realização de um knn, no qual são lidos os arquivos “xtrain.txt”, “ytrain.txt” e “xtest.txt” e será gerado uma saída com a devida classificação no arquivo “ytest.txt”. Além disso, vale ressaltar que para esse knn o número de elementos que serão analisados “k” é sempre igual a um.

Como Compilar

A compilação é argumentavelmente simples. Basta executar o Mars4_5.jar (leve em consideração que é necessário ter o java instalado em seu computador) que já vem junto ao pacote do EP. Utilizando a ferramenta Mars, deve-se abrir a pasta “code” e então selecionar o arquivo .asm chamado “OAC2EP1”, ficando com o caminho no final de code/OAC2EP1.asm. Após estar com o código aberto, basta pressionar F3 (Ou ir na janela Run -> Assemble do Mars) e então pressionar o botão “Run the Current Program” pelo próprio editor.

O resultado será salvo na pasta output em um arquivo txt chamado ytest.txt, que também vem junto do EP (A saída será impressa também no terminal do MARS).

Para trocar a entrada do EP é necessário alterar os arquivos da pasta data, lá devem ser adicionados (ou modificados) os arquivos, xtrain.txt, ytrain.txt e xtest.txt.

IMPORTANTE: É necessário utilizar o Mars4_5.jar que vem junto ao Exercício Programa, tendo em vista que o compilador utiliza como ponto de referência o local em que o seu arquivo .jar está salvo e não onde o código .asm está. Caso não deseje utilizar o Mars que vem junto ao projeto, atualize o caminho dessas variáveis abaixo localizadas no .data do arquivo OAC2EP1.asm:

- pathXTrain: .asciiz "C:/caminho/xtrain.txt"
- pathYTrain: .asciiz "C:/caminho/ytrain.txt"
- pathXTest: .asciiz "C:/caminho/xtest.txt."
- ytestFileName: .asciiz "C:/caminho/ytest.txt"

MAIN

Primeiramente carrega-se os conteúdo dos arquivos “xtrain.txt”, “xtest.txt” e “ytrain.txt e seus respectivos números de linhas e de colunas, após isso, chama a função “chamaknn” para iniciar o processo de análise de cada um dos vetores presentes em “xtest.txt”. Após isso , chama-se a função “saída” para salvar o conteúdo da saída em “ytest.txt”.

PRINT

As funções da categoria print em si foram utilizadas somente para debugar o código. Segue uma breve explicação delas.

- ***print***

Tal função recebe um array, o index do primeiro elemento desse array que deseja printar (caso não deseje começar pelo 0) e o tamanho do array. Ela printa o primeiro valor

desejado e então atualiza a posição do array. Após isso, verifica se chegou ao final do array, caso tenha chegado, chama a função “printlinha” e após isso executa um jump return, se não, chama a si mesma novamente (chama a função “print” de novo).

- ***printlinha***

Printa uma linha em branco no terminal (ou uma divisória de prints: “========”, dependendo de como está salvo no .data do arquivo .asm) e retorna para a função print por meio de um jump return.

KNN

Abaixo serão explicadas algumas das funções utilizadas pelo código para gerar o knn, leve em conta que as funções que chamam o KNN e que definem variáveis de antemão para ajudar em seu desempenho também serão explicadas nesta sessão.

- ***chamaknn***

Tal função tem como principal objetivo inicializar o contador do primeiro loop (passar por todos os valores de ytrain) e carregar o array do ytrain.
OBS: Essa função também carrega o array “closerlist” utilizado durante o desenvolvimento do código para depuração de bugs e análise das distâncias mínimas de cada ponto.

- ***first_loop***

É o loop que passará por todos os valores de ytrain, nele também é inicializado o vetor do xtest, o vetor do xtrain e o contador do “second_loop” responsável por percorrerer todo o xtrain.

- ***second_loop***

O second loop fica rodando até que todos os valores de xtrain tenham sido analisados. Esse loop é responsável por chamar a função KNN em si, passando como argumento (em $a1, $a2 e $a3) xtrain, ytrain e xtest.

- ***second_loop_end***

É chamada quando o segundo loop (referente ao xtrain) terminar de percorrer todos os valores. Ele adiciona 1 ao contador do loop do ytrain, salva a classe do ponto analisado pelo knn com o retorno fornecido pela função em $v0, reinicia a menor distância para um número muito grande e chama o primeiro loop novamente.

- ***first_loop_end***

Essa função simplesmente volta para a main com um jump return (jr $ra).

- ***knn***

Tal função realiza o cálculo do KNN. Primeiramente ela salva o início do vetor do xtrain e xtest, então realiza o cálculo a seguir para calcular a distância do um vetor de n dimensões:

$$d = \sqrt{\sum_{i=0}^{n} (xtest[i] - xtrain[i])^2}$$

Após isso, a função knn compara o resultado da equação com a atual menor distância, se a distância nova for menor, ela sobrescreverá a classe de retorno anual para a classe da menor distância.
A função retorna um inteiro referente à classe (1 ou 0) para esse ponto em específico analisado.

SAÍDA

Abaixo serão explicadas todas as funções que tem envolvimento com a criação do arquivo "ytest.txt".

- ***saida***

Abre o arquivo para a escrita (ou cria um no diretório output, caso o "ytest.txt" não exista), prepara uma variável contadora para percorrer o array do ytest inteiro e chama a função de escrita em si.

- ***write_loop***

Carrega o valor do ytest atual já calculado pelo código e o compara com 1.0, se for igual, realiza um jump para a função "write1", se não for (ou seja, é 0.0) escreve o valor 0.0 no arquivo "ytest.txt" e chama a função "call_loop". Também verifica se acabou o array e, caso tenha acabado, dá um jump return.

- ***write1***

Escreve o valor 1.0 no arquivo de saída "ytest.txt" e chama a função "call_loop". Também verifica se acabou o array e, caso tenha acabado, dá um jump return.

- ***call_loop***

Escreve uma quebra de linha "\n" e chama a função "write_loop".

LEITURA

Quanto à forma com que a leitura fora feita, cabe explicar o contexto geral primeiramente, de forma abrangente e depois os principais Loops da função Carrega.

A forma com que a leitura fora feita consiste em converter o arquivo em uma string, armazenar o endereço da primeira posição em um registrador($t0) e em seguida ir carregando subsequentemente byte por byte em um outro registrador, o qual armazenará o valor daquele Char, conforme as especificidades da tabela ASCII, para que possam ser feitas transformações matemáticas, a fim de que o valor, antes armazenado como um char, possa ser o equivalente em int. É então que ocorre uma conversão para o Co-Processador 1, transferindo o valor para um registrador de float com base em sua posição.

O loop começa com um valor inteiro, e entra no primeiro loop onde toda vez que um valor subsequente é encontrado, o primeiro é multiplicado por 10 e somado com o próximo.

A função segue assim até que seja encontrado o valor ASCII equivalente ao '.', onde entrará num novo loop que reverte e o primeiro valor encontrado é dividido por 10 e cada valor

subsequente é dividido mais uma vez, a fim de andar uma casa decimal para a direita e depois é somado e armazenado no registrador final.

Tanto para a parte inteira quanto para a parte decimal do valor são utilizados 2 processadores, totalizando 4. Um para armazenar os resultados das somas subsequentes e um auxiliar para sempre carregar o próximo valor.

Quando for encontrado um valor correspondente a um dos símbolos que indiquem uma quebra de linha, o começo do texto ou uma vírgula, a função realiza uma soma do registrador que contém a parte inteira com a parte decimal, resultando no valor completo que antes estava em string e o armazena no array indicado e depois anda uma posição no valor apontado pelo registrador que aponta para o array. Os principais loops responsáveis são:

- ***analisa:***

Loop responsável por analisar o primeiro valor antes da vírgula ou da quebra de linha, convertê-lo para float e armazenar em $f2.

- ***continua:***

Aqui, a cada iteração, $f2 é multiplicado por 10 e somado a $f3, indicando que a cada número subsequente os valores que já foram encontrados devem andar uma casa para a esquerda.

- ***decimal:***

Aqui, semelhantemente ao “analisa”, é iniciado o primeiro valor após o ‘.’ e é dividido por 10 e salvo em $f4.

- ***Loop_decimal:***

Aqui é semelhante ao “continua”. A cada novo valor encontrado, um contador $t9 indicará quantas vezes o valor precisa ser dividido por 10 antes de ser somado ao $f4, sendo o registrador que armazenará o valor definitivo da parte decimal.

- ***Próximo:***

Aqui, $f4 e $f2, o primeiro contendo a parte decimal, e o segundo a inteira, serão somados em $f7, que armazenará em seguida o resultado no Array armazenado em $a0. $a0 aponta para 4 bytes após, e o loop fará um Jump para o início da leitura.

- ***Fim:***

Faz a mesma operação que o loop acima, porém retorna para o main, onde a função fora chamada, terminando, assim, a leitura do arquivo desejado.

As mesmas etapas podem ser consideradas nos loops da função responsável pelos arquivos y, porém aqui, de forma a otimizar a velocidade das operações, os valores em floats das classes foram carregadas antecipadamente, sendo ‘1.0’ e ‘0.0’, em dois registradores. Aqui, a diferença é que apenas o primeiro valor após a quebra de linha é analisado e, a depender de uma das duas possibilidades, a respectiva classe é atribuída.

Abaixo segue uma breve explicação sobre as funções de leitura de coluna do código. Seu único propósito é descobrir a dimensão do vetor que será utilizado no knn.

- ***carregaCol***

Abre o arquivo do xtrain e salva sua primeira linha para futuramente contar quantas vírgulas existem nela. Também inicializa o contador de vírgula

- ***count_loop***

Carrega o caractere atual da linha, verifica se ele é uma vírgula ou quebra de linha, se foi uma vírgula, soma 1 ao contador e chama novamente a si mesmo, se for uma quebra de linha faz um jump para “end_program”.

- ***found_comma***

Aumenta em 1 o contador de vírgulas.

- ***continue_loop***

Avança para o próximo caractere da linha, incrementa o índice e dá um jump para “count_loop”.

- ***end_program***

Soma 1 ao contador de linhas e faz um jump return para a main.
O motivo de somar 1 a mais no contador é que um vetor de 5 dimensões, por exemplo, possuirá 4 vírgulas (1.0,2.0,3.0,4.0,5.0), enquanto possui 5 colunas, logo, a fórmula do número de colunas dado o número de linhas é:

$$n_{col} = n_{vir} + 1$$

Assim, finalizando a leitura de colunas.