Universidade Tecnológica Federal do Paraná
Campus Curitiba

CSF13 - 2021-1 - PROJETO 2

Rafael Eijy Ishikawa Rasoto – rafaelrasoto@alunos.utfpr.edu.br
RA: 2004585

Gabriel de Almeida Spadafora - gspadafora@alunos.utfpr.edu.br
RA: 2377608

João Pedro Dos Reis Mendes - mendes.2021@alunos.utfpr.edu.br
RA: 2377616

1. Divisão do Trabalho

Nós decidimos que para conveniência da equipe, todos os integrantes estariam presentes no desenvolvimento da lógica, dessa forma todos estariam envolvidos em cada etapa do desenvolvimento além de facilitar o entendimento homogêneo do código pela equipe.

Sendo assim, o participante Rafael Eijy Ishikawa Rasoto ficou responsável pela escrita do código por ter experiência prévia ao curso em C, mas ao mesmo tempo, os alunos Gabriel Spadafora e João Mendes participaram da escrita pela extensão live share do Visual Studio Code. Todos os códigos foram compilados no Code Blocks IDE e editados no Visual Studio devido a praticidade da ferramenta da Microsoft.

2. Desenvolvimento

2.1. Início.

Inicialmente, uma análise da main.c e da imagem.c foram realizadas para entender exatamente o que o código solicitado deveria fazer, como seria calculado o score, como a imagem foi transformada em tonalidades de cinza além de como as structs foram declaradas dentro do projeto.

Como o material de structs foi liberado no mesmo dia, cada aluno ficou responsável em ver o material e dominá-lo para que então fosse possível discutir o código. Ficou claro para os integrantes que o trabalho exigiria um código para tratamento da imagem, separando o preto do branco, além de um código para achar o caminho mais curto.

Inicialmente discutimos qual forma de descobrir caminho usaríamos. A ideia utilizada na tarefa de segunda-feira (16/08) de preencher uma matriz de custo apresentou-se como uma solução simples e que apresentava o resultado necessário. Também foram considerados usar algoritmos de busca como o Algoritmo de Dijkstra, mas como demonstrava-se mais complicado, a equipe optou pelo primeiro devido a familiaridade.

Com isso em mente, a equipe começou a pensar como seria feito o tratamento da imagem. A princípio pensamos que provavelmente seria uma diferença clara, tendo em vista que para os olhos humanos, a diferença do branco para o preto era nítida, então provavelmente a diferença entre os valores seria claro. Contudo, ao criarmos um projeto de testes, no qual modificamos profundamente o código enviado pelos professores, percebemos que seria muito mais complicado que o previsto.

2.2 Tratamento da Imagem.

Compilamos um código que imprimia todos os pixels da imagem. Observamos que o limiar era bem sutil. De início, pensamos em fazer apenas um código semi funcional, capaz de pelo menos imprimir o risco pela tela. Começamos o desenvolvimento do código e vimos que ia ser complicado. A primeira versão do código estava cheia de problemas, e não entendíamos direito o porquê. O código compilava, porém, parava no meio e retornava um valor estranho. Fui atrás desse valor pelos .c e não encontrei em local algum.

Pedimos ajuda a um veterano, o qual ao bater o olho já indicou que o problema era acesso da memória. Após sua ajuda, observamos que nosso código avançava bem além do que ia no início, porém ainda não conseguia finalizar a imagem. Após inserirmos uma série de printfs no código para analisá-lo, descobrimos um grande problema. O código não conseguiu encontrar um caminho para o outro lado, alocando de maneira errônea a struct caminho e dando vários acessos indevidos.

Estávamos determinados a pelo menos conseguir imprimir uma linha reta na tela., então escrevemos um código super básico que imprimiria uma linha reta exatamente no meio da imagem, e mesmo assim observamos um problema de acesso. Esse erro foi muito importante para o desenvolvimento, pois observamos que estávamos alocando de maneira errônea “caminho” e sendo assim estava faltando exatamente um espaço. Ao corrigir o erro, conseguimos fazer a linha reta, e vimos como funcionava a devolução das imagens na pasta e o score pelo out.txt.

Depois disso, voltamos ao nosso código, e reescrevemos ele, utilizando partes do passado, usando uma lógica proposta pelo aluno Gabriel, botar um percentual da média da imagem. Então a cada imagem o código calculava a média e usávamos uma porcentagem para definir o preto do branco. O problema é que ainda várias imagens davam erro, principalmente a 2, 5 e 7. Para corrigir, fomos aumentando diversas vezes esse limiar, até que um momento todas compilaram, mas para nossa tristeza, as mais escuras e intermediárias apresentaram erros de cortes grandes, tendo vários aleatórios e diversas linhas retas atravessando a imagem, era claro que tínhamos muito trabalho a desenvolver.

Concentramos em estudar nosso código desenvolvido até o momento, tirar todos erros e variáveis redundantes, e assim, começarmos a discutir possíveis soluções. Pensamos em um código que faria a leitura de média de regiões do código para determinamos se é branco ou preto, um código que definiríamos apenas uma casa preta inicial e baseado na diferença com os pixels adjacentes, determinaria se o próximo seria branco ou preto, ou então um código que realizaria a leitura da

quantidade de pixels com um determinado valor e a partir de uma ideia de derivação conseguiríamos achar um bom limiar para branco. Porém, ao pensarmos mais a fundo, vimos que essas possibilidades trariam ou novos problemas ou então teriam uma lógica muito complexa. Seguindo o conselho da professora Leyza, pensamos na ideia de saturação da imagem, e a partir disso, veio em mente a possibilidade de elevar os valores dos pixels por alguma potência, pois aumentando drasticamente a diferença entre o preto e o branco, facilitava a criação de limiares para isso.

Inicialmente dependendo dos resultados íamos ajustando se seria ao quadrado ou ao cubo (para evitar problemas de overflow na matriz nova), e usávamos ainda a ideia de multiplicar a média, dessa vez elevada, por uma potência. Como estava algo muito aleatório e ainda não tínhamos a garantia que o código seria executado em qualquer código, o aluno Rafael Rasoto implementou um sistema de retroalimentação do código que indicaria se há ou não uma resposta, e caso não exista, a porcentagem era aumentada até achar uma saída válida. Inicialmente aumentava de 1% em 1%, mas vimos que de 5% em 5% era melhor pois assim diminuía a chance de pegar logo a primeira saída, mas sim, a mais curta, além disso, sendo um código que se retroalimenta, seria de alta adaptabilidade para cada imagem.

Então antes de implementar essa lógica, os alunos reescreveram o código, criando funções para organização, o que garantiu fácil modificações na estrutura do código, pois inicialmente o aluno Rafael tentou fazer várias partes copiadas do código reestruturando nessa ideia de realimentação e ocorreram vários erros, é notório que as funções auxiliaram muito na mudança da estrutura do código de uma maneira simples, funcional e organizada.