Universidade Tecnológica Federal do Paraná – UTFPR, Campus Curitiba - Centro
Aluno: Amanda Jury Nakamura RA: 2582686 Curso: Sistemas de Informação
Disciplina: ICSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu, Profa. Leyza Baldo Dorini, Prof. Daniel Fernando Pigatto

Relatório – Projeto 01

Este relatório foi escrito, de certa forma, juntamente ao desenvolvimento dos códigos, uma vez que servia também como rascunho e guia para ideias a serem aplicadas. Após a leitura integral das instruções, o projeto foi dividido em 2 (uma parte por função) e feito separadamente. Cada função também foi separada em 2 partes.

1. Função 01 - `int calculaInterseccao (int n_retangulos);`

- Recebe: Número de retângulos
- Retorna: Área total do retângulo formado pela intersecção dos demais
- Funções inclusas: Recebem o número do retângulo e retornam suas coordenadas

** Importante: Otimização (parar o código caso 2 retângulos não se interseccionem)

Inicialmente, tive certa dificuldade para verificar se haveriam retângulos que não se intersecionavam. Por isso, essa parte foi resolvida mais para o fim do trabalho.

O objetivo dessa função é encontrar uma forma de calcular a área total do retângulo formado pela intersecção dos demais. Para isso fiz uma lista de possíveis formas e informações que seriam necessárias coletar:

- Coordenadas de todos os retângulos;

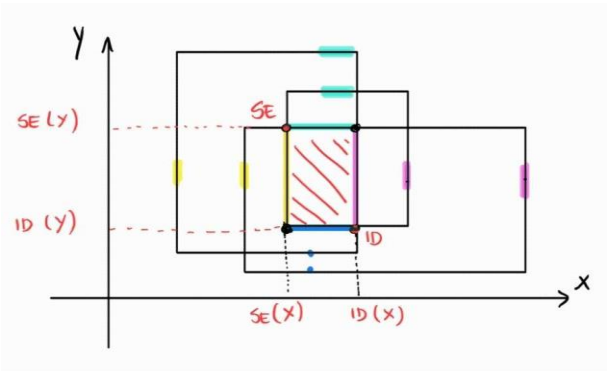
- Área de todos os retângulos (?);

- Coordenadas do retângulo formado pela intersecção dos demais;

- Como se calcula a área de um retângulo? Base x Altura;

Obter as coordenadas dos retângulos foi simplesmente aplicar as funções inclusas. Porém, o problema foi encontrar as coordenadas do retângulo específico que se buscava. O método que auxiliou nessa abstração foi utilizar ferramentas para esboçar graficamente o que se pretendia.

O desenho em que ficou claro quais seriam as informações necessárias para cumprir o objetivo da função foi o esboçado a seguir:



As conclusões disso foram: Independente do número de retângulos, aquele formado pela intersecção dos demais deve ter coordenadas correspondentes ao maior yID (lados direitos expressas em rosa), maior xSE (lados superiores em verde), menor ySE (inferiores em azul) e menor xID (laterais esquerdas em amarelo).

Sabendo disso, conforme as coordenadas dos retângulos eram coletadas, também eram salvos em variáveis. Tendo esses valores ao final do *loop*, calculou-se a base e a altura desse retângulo e então sua área.

2. Função 02 - `unsigned int encontraParMaisProximo (int n_retangulos)`

- Recebe: Número de retângulos
- Retorna: Número de 32 bits (correspondendo aos números de 2 retângulos cujos centros têm a menor distância)
- Funções inclusas: Recebem o número do retângulo e retornam suas coordenadas

Lista de possíveis informações úteis:

- Base, altura, diagonais;

- Achar centros (é um ponto com coordenadas X e Y) - 2 formas:

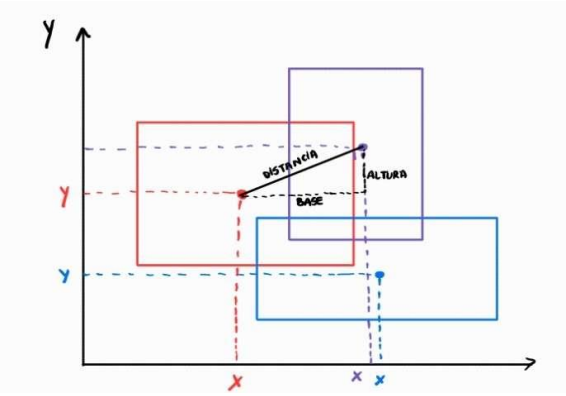
  –> base/2 e altura/2

  –> diagonal/2

- Distância entre todos os centros (distância entre pontos no ponto cartesiano forma triângulos – uso Pitágoras)

Novamente, as coordenadas eram recebidas pelas funções inclusas, e a partir disso calculei a base e altura dos retângulos e dividi por 2 achando que iria obter o centro. Inicialmente, usei o teorema de Pitágoras considerando esse “centro”, o que retornava diversos erros nos testes.

Após uma análise mais precisa e conversas com colegas, notei que ao informar esses valores, o que o programa recebia era somente a distância dos vértices até o centro. Os valores que deviam ser enviados em “coordenadaXcentro” de A e B correspondem à coordenada xSE acrescido do valor da (base/2). O mesmo para as coordenadas Y.



Foram usados 2 *loops* de forma a comparar a distância dos centros de uma retângulo A (regulado pelo *loop* exterior) e retângulo B (regulado pelo *loop* interior). Os valores salvos em variáveis foram feitos baseados na condição da menor distância. A menor distância foi inicializada como FLT_MAX da biblioteca “math.h”. Se essa condição for verdadeira, salva-se os números dos retângulos cujos centros formam a menor distância, e atualiza-se a menor distância.

A transformação dos números decimais para 32 bits foi feita utilizando operadores “bit a bit” e para ordená-los, condicionais.

Dificuldades:

As maiores dificuldades foram encontradas na verificação das intersecções. Inicialmente, meu pensamento estava voltado em um único ponto que resultava na intersecção de 2 retas. Depois, uma das ideias que ocorreu foi verificar se haviam pelo menos um vértice de um retângulo dentro da área de outro, porém dessa forma seriam necessárias diversas comparações e talvez não fosse o melhor caminho.

Ao recorrer a monitoria e conversar com colegas, ficou claro que era necessário realizar comparações entre os valores de x de todos os retângulos, e depois entre os valores y, separadamente.

Usando estruturas condicionais verifiquei casos tomando como referência os retângulos A e B, definidos pelos menores valores de X (retângulo mais à esquerda no plano) e maiores de Y e vice e versa. A situação em que os retângulos possuíam as mesmas coordenadas em algum dos pontos também foi levada em conta, utilizando “>=”.

Nas mesmas condicionais foram comparadas coordenadas dos retângulos quanto ao eixo X e quanto ao eixo Y de forma que um valor devia estar contido no outro para que houvesse intersecção e caso contrário o código já retornava 0 indicando que não havia intersecções.

Outro erro que demorou para ser notado foi a simples declaração das variáveis “base” e altura” que poderiam ser *float*. Com elas declaradas como *int*, o código compilava e rodava, mas apresentava erros nos testes. Uma das formas que auxiliaram na descoberta do erro foi “printando” os valores de retProx1 e retProx2, demonstrando que o problema não era na conversão dos números para binários.