Trabalho Prático 1 de AEDS 3

Departamento de Ciências da Computação - UFSJ

Felipe Caldas de Paiva - 212050016

Resumo

Resumo: *Esse trabalho tem como objetivo a resolução de um problema envolvendo hipercampos proposto pelo professor Léo Rocha. A documentação a seguir explica o algoritmo implementado e as estratégias adotados para solucionar o problema proposto, além da análise de sua complexidade e exemplos de teste usados durante o processo de construção do algoritmo.*

1. Introdução

O problema proposto se baseia em duas âncoras (Xa,0) e (Xb,0) e um conjunto P de pontos e espera como resultado o número máximo de pontos que podem ser interligados, possuindo interseção apenas nas âncoras, em ouras palavras, os segmentos de reta que ligam os pontos não podem se cruzar.

2.Solução proposta

Para solucionar a questão proposta, foram adotados conceitos básicos de geometria, alinhados com estruturas simples de programação em C, como alocação dinâmica, structs e algoritmos de ordenação(*quicksort*). O algoritmo criado recebe o número de pontos, os valores x das âncoras e as coordenadas dos pontos do conjunto P através de um arquivo, aloca espaço na memória pra tais pontos, ordena-os por ordem crescente de coordenada y e verifica se há interseção entre os segmentos que ligam os pontos, usando como parâmetro o coeficiente da reta que os liga, que define a orientação da reta e permite saber se um ponto está dentro ou fora do triangulo que se forma quando se ligam os pontos. Por fim ele retorna o número máximo de pontos que podem ser interligados com interseção apenas nas âncoras, sendo este valor o tamanho do maior conjunto de pontos ligados durante a execução.

3.Implementação

Usando uma struct ‘Ponto’ para guardar as coordenadas dos pontos recebidos e variáveis globais externas para guardar o valor da âncoras, a quantidade n de pontos e o valor a ser usado como resultado.

```
typedef struct {
    float x,y;
}Ponto;



extern Ponto *pontos;
extern int num_pontos, maior_tamanho_conjunto;
extern Ponto ponto_ancora_a, ponto_ancora_b;
```

A partir daí o programa recebe 2 arquivos como parâmetro (entrada e saída), como o arquivo de entrada contendo na primeira linha o valor n de pontos a ser lidos para o problema e o valor X das duas âncoras. Nas demais linhas são passados o restante dos pontos por coordenada X e Y.

Obs: Há condições para a quantidade máxima de pontos e para a grandeza dos valores dos pontos, e em caso dessas condições não serem atendidas, o programa retorna uma mensagem de erro ao usuário.

Após alocar memória para as entradas de ‘input’ e output’, a função ‘openFile’ abre o arquivo a partir dos comandos passados no momento da execução e através da diretiva ‘getopt’ identifica os arquivos de entrada e saída (input e output, respectivamente).

```
int openFILE(int argc, char* argv[],char* input,char* output){
    int op;
    if(argc < 2){
        printf("ENTRADA INVALIDA!\n");
    }



    while((op    getopt(argc, argv, "i:o:")) != -1){
//lê os comandos e usa o "-i" como arquvio de entrada e o "-o" como arquivo de saida
        switch (op) {
            case 'i':
                strcpy(input, optarg);
                break;
            case 'o':
                strcpy(output, optarg);
                break;
            default:
                fprintf(stderr, "ENTRADA INVALIDA!: `%c'\n", optopt);
//se não foi informado -i ou -o, fecha o programa
                return -1;
                break;
        }
    }
    return 0;
```

```
}
```

Após isso chama a função ‘readFILE’ que realiza a leitura dos parâmetros do problema e armazena nas variáveis globais citadas anteriormente, alocando memória dinamicamente.

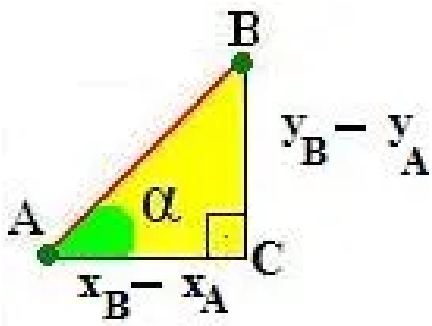
```
void readFILE(char *input) {

    FILE *fp    fopen(input, "r");



    fscanf(fp, "%d %f %f \n", &num_pontos, &ponto_ancora_a.x, &ponto_ancora_b.x);

    ponto_ancora_a.y    0;

    ponto_ancora_b.y  0;

    if((num_pontos >1000000) || ponto_ancora_b.x <= ponto_ancora_a.x || ponto_ancora_b.x >10000 ||
ponto_ancora_a.x < 0) {                    //testa se os valores estão dentro do limite

        printf("ERRO: entrada do usuário não corresponde aos requisitos do programa!\n");

        exit(0);

    }



    pontos    (Ponto *)malloc(num_pontos * sizeof(Ponto));



    for(int i    0; i < num_pontos; i++)

        fscanf(fp, "%f %f \n", &(pontos[i].x), &(pontos[i]).y);



    fclose(fp);

}
```

Tendo as entradas devidamente lidas, o módulo principal chama a função ‘encontrar_maior_conjunto’ que primeiramente, ordena os pontos por coordenada y, aloca espaço para a variável ‘tamanho_conjunto’ e define todos os seus valores como 1. Logo em seguida chama a função resolver que será responsável por retornar o valor esperado para o resultado.

```
void encontrar_maior_conjunto(){

    qsort(pontos, num_pontos, sizeof(Ponto), cmp_Pontos);



    int *tamanho_conjunto    malloc(sizeof(int) * (num_pontos + 1));
```

```
    for(int i     0; i < num_pontos; i++)
        tamanho_conjunto[i]     1;

    resolver(tamanho_conjunto);
    free(tamanho_conjunto);
}
```

A função ‘resolver’ em seguida calcula o coeficiente das retas que ligam as âncoras A e B a cada um dos pontos passados (usando de um laço for para verificar cada um dos pontos). O coeficiente angular da reta por si define a orientação e sentido da reta, permitindo saber se ela é inclinada para a direita ou para a esquerda. O cálculo desse coeficiente pode ser feito através da tangente do ângulo do triângulo formado (cateto oposto sobre cateto adjacente), o que em outras palavras seria a diferença entre as coordenadas Y sobre a diferença entre as coordenadas X, como mostrado na imagem a seguir:



Através do deste coeficiente de reta, podemos verificar se há interseção entres os segmentos que ligam os pontos às âncoras, já que sabendo a orientação da reta e o ângulo que ela forma com o eixo das âncoras, fica fácil descobrir se um ponto está dentro ou não das retas formadas e a existência ou não de interseções entre os segmentos de reta.

```
int verificar_interseccao(float coef_a, float coef_b, Ponto pa, Ponto pb) {
    float coef     calcular_coeficiente_reta(pa, pb);
    if (coef_a < 0) {
        return (pa.x > pb.x && coef < coef_b && coef > coef_a) ? 1 : 0;
    } else if (coef_b > 0) {
        return (pa.x > pb.x && coef < coef_b && coef > coef_a) ? 1 : 0;
    } else {
        if(pa.x > pb.x){
            return (coef > coef_a) ? 1 : 0;
            return (coef < coef_b) ? 1 : 0;
        }
```

```
    }
}
```

Por fim, após verificar as interseções, a função ‘encontrar_tamanho_conjunto’ retorna o o valor referente ao tamanho de cada conjunto de pontos ligados e junto da função ‘atualizar_maior_conjunto’ define qual o conjunto com a maior quantidade de pontos ligados dentro das condições propostas no problema, retornando o valor ao usuário e guardando este mesmo valor no arquivo de saída passado como parâmetro.

Importante dizer que durante a execução do algoritmo foi feito uso das funções ‘getrusage()’ e ‘gettimeofday()’ para fins de comparação entre os tempos de execução (que também são retornados ao usuário ao fim do programa).

4.Compilação

Quanto a compilação foi criado um arquivo ***makefile*** que compila e roda o código com os parâmetros de entrada e saída, além de remover os objetos criados através do comando ***make clean.***

Make: *compila o programa todo.*

Make run*: roda o programa passando os parâmetros input.txt e output.txt de uma vez.*

./tp1 -i input.txt -o output.txt

Make clean*: limpa e remove os objetos criados para a execução do programa.*

```
IDIR=../includes
CFLAGS=-I$(IDIR)

ODIR=obj
LIBS=-lm
DEPS= pontos.h file.h

_DEPS = pontos.h file.h
DEPS = $(patsubst %,$(IDIR)/%,$(_DEPS))

_OBJ = pontos.o file.o main.o
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))


$(ODIR)/%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

tp1: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

.PHONY: clean

clean:
    rm -f $(ODIR)/*.o *~ core $(INCDIR)/*~

run:
    ./tp1 -i input.txt -o output.txt
```

5.Análise

Durante os testes do algoritmo, foi usado um gerador de teste para este caso específico que me foi fornecido por um colega do curso de computação.

Para esta análise, foi considerado o uso de tempo para o programa ser executado.

O programa foi executado em um notebook Acer Nitro 5, com processador i5-9300H CPU @ 2.40GHz e GPU GTX 1650. O algoritmo foi testado principalmente no sistema operacional Linux, conforme pedido do professor, porém também foi testado no sistema Windows através de uma máquina virtual, tendo resultados bastantes similares nos dois SOs.

Cada teste foi realizado diversas vezes, usando primeiramente os valores passados como exemplo na documentação referente ao enunciado do trabalho e depois com valores N de pontos de 50,100,1000,5000 e 9999. A seguir estão os melhores resultados de tempo de cada teste, levando em consideração que entre os melhores e piores tempos houve uma diferença bem mínima.

Entrada de exemplo (2 pontos) :

```
./tp1 -i input.txt -o output.txt
Resultado = 1

Tempo de Sistema = 0.000000
Tempo de Usuario = 0.002612
gettimeofday = 0.005221
```

Teste com 50 pontos:

```
./tp1 -i input.txt -o output.txt
Resultado = 15

Tempo de Sistema = 0.000000
Tempo de Usuario = 0.003183
gettimeofday = 0.005948
```

Teste com 100 pontos:

```
./tp1 -i input.txt -o output.txt
Resultado = 9

Tempo de Sistema = 0.000000
Tempo de Usuario = 0.002385
gettimeofday = 0.005561
```

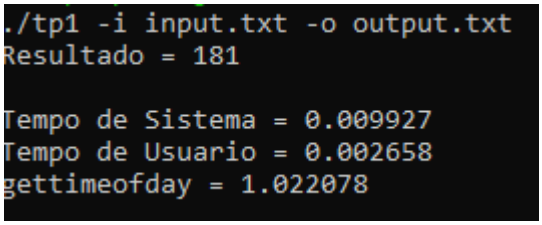
Teste com 1000 pontos:

```
./tp1 -i input.txt -o output.txt
Resultado = 76

Tempo de Sistema = 0.000000
Tempo de Usuario = 0.006262
gettimeofday = 0.013377
```

Teste com 5000 pontos:

```
./tp1 -i input.txt -o output.txt
Resultado = 2740

Tempo de Sistema = 0.010033
Tempo de Usuario = 0.602028
gettimeofday = 0.619326
```

Teste com 9999 pontos:



Através desses resultados podemos ver que o tempo de execução cresce muito pouco de acordo com o crescimento de ‘n’. Isso acontece devido à ordenação usada no algoritmo, facilitando o processo de verificação, que permite uma complexidade menor favorecendo entradas muito grandes.

7.Referências

<https://users.cs.utah.edu/~germain/PPS/Topics/C_Language/file_IO.html/>

<https://pt.stackoverflow.com/questions/320400/geometria-computacional-como-verificar-se-duas-retas-se-interceptam-apenas-na>

<https://pt.khanacademy.org/math/algebra/x2f8bb11595b61c86:linear-equations-graphs/x2f8bb11595b61c86:slope/a/slope-review#:~:text=O%20coeficiente%20angular%20de%20uma,dividida%20pela%20varia%C3%A7%C3%A3o%20em%20x).>

<https://stackoverflow.com/questions/39422703/optopt-and-getopt-in-c>

DE FIGUEIREDO, Luiz Henrique. PINTO, Paulo César.” Introdução à geometria computacional” - Instituto de Matemática Pura e Aplicada

<https://impa.br/wp-content/uploads/2017/04/18_CBM_91_06.pdf>