Projeto 2 - Implementação dos Algoritmos de Ordenação
Prof. Me. Edmar Roberto Santana de Rezende

Alcides Gomes Beato Neto	19060987
Henrique Sartori Siqueira	19240472
Rafael Silva Barbon	19243633

Introdução

O projeto tem como objetivo implementar os conhecimentos adquiridos de grafos, por meio do desenvolvimento de um programa que realiza a otimização de vôos com destino a vários aeroportos, sendo cada aeroporto um vértice e cada vôo uma aresta. O cálculo dos valores utiliza a busca em largura e profundidade, e também a utilização do algoritmo de Dijkstra (em conjunto com a busca em largura) para encontrar os caminhos mais curtos. O programa recebe como parâmetro da chamada de execução o nome de um arquivo texto que contém inicialmente a quantidade de aeroportos em seguida da quantidade de conexões e, por fim, uma lista de adjacência relacionando os aeroportos com seus destinos e custos. O arquivo é lido dentro do programa e o grafo é criado com base nesses dados.

Descrição

A implementação foi realizada em ambiente Linux - Ubuntu 20.04 utilizando o editor de texto Visual Studio Code em conjunto com o GitHub para o versionamento e compartilhamento de código.

Inicialmente foi realizada uma checagem de formatos e a quantidade de parâmetros recebidos pela função main:

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ make
gcc -o proj2 arestas.c main.c pilha.c
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voos.txt VCP

Argumentos: 3
VCP
Voos.txt
./proj2henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$
```

Imagem 1. Teste de parâmetros.

Para os testes de arquivo, foi realizado o teste de coleta das informações do arquivo, bem como a impressão das mesmas:

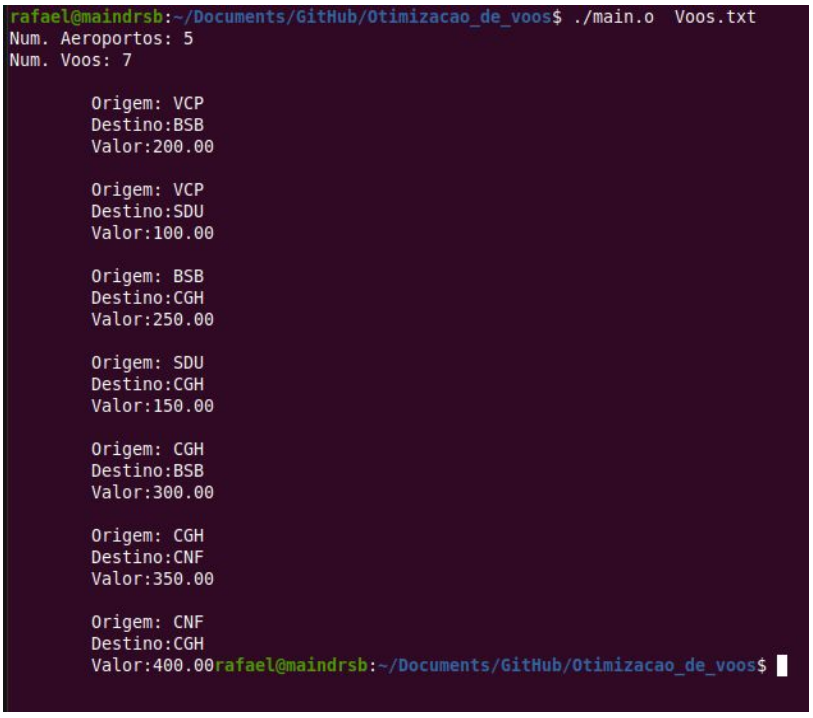

Imagem 2. Teste de leitura e coleta de dados do arquivo.

Com as informações sendo coletadas corretamente pelo programa, houve a criação e testes do grafo, em que uma função de exibição foi criada para comprovar o funcionamento, entretanto, foi notado que houve repetições de aeroportos como demonstrado a seguir:

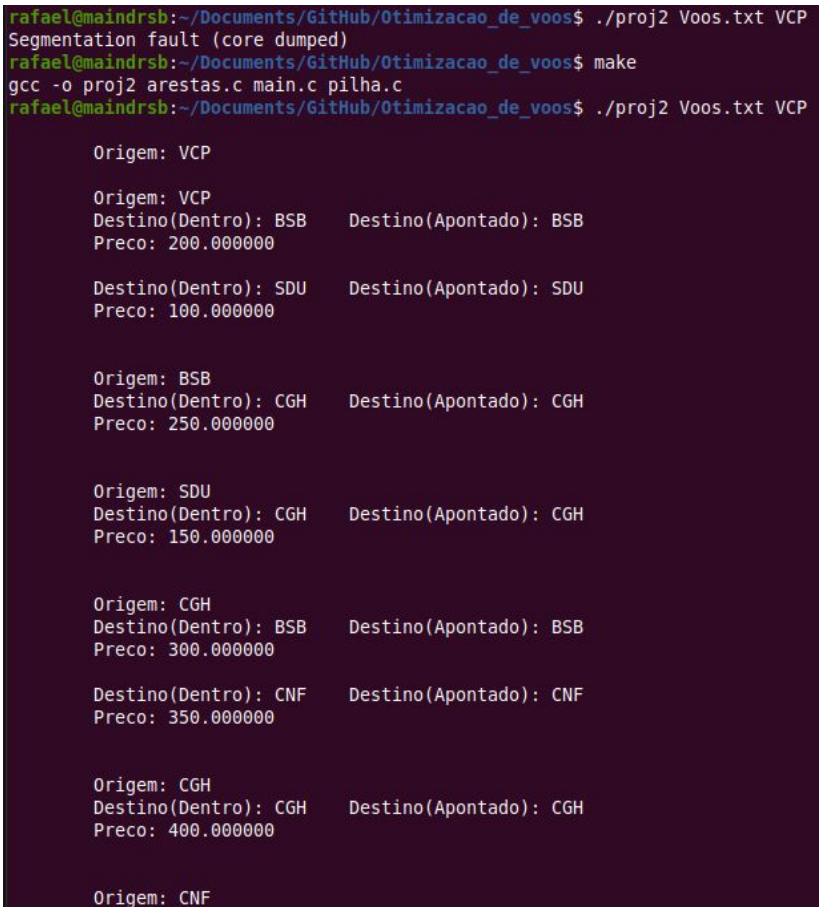

Imagem 3. Teste inicial da criação do grafo.

Após a constatação de tal erro, foi realizada uma verificação na leitura das strings do arquivo, pois houve a consideração de que haviam caracteres não imprimíveis (por exemplo ‘\n’) a partir da segunda leitura de um vôo do arquivo, fazendo com que a string anterior fosse diferente da atual. Tal verificação consistiu em exibir o tamanho da string na leitura do arquivo como mostrado a seguir:

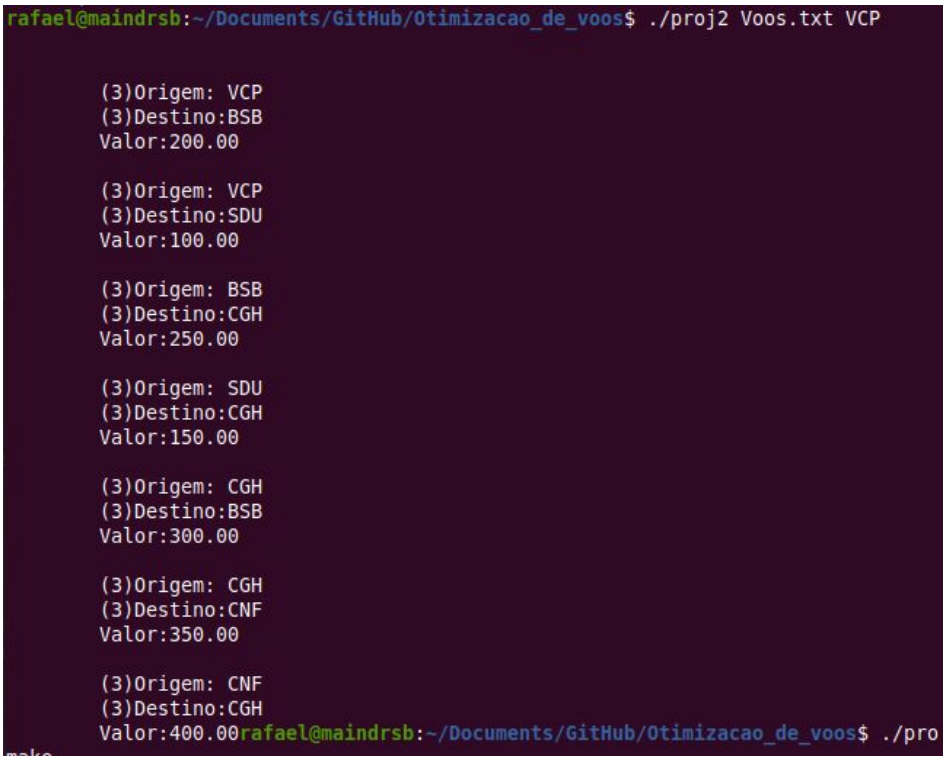

Imagem 4. Teste de tamanho das strings.

Ao perceber que o formato da string estava como esperado, houve a teoria de que as funções de adicionar uma aresta a um vértice do grafo estavam incorretas, porém após a análise das funções de criação constatou-se que todas correspondiam ao esperado. Então a análise foi voltada para o programa principal, em que o erro foi descoberto, e consistia em não realizar a atualização do histórico de strings lidas (verificação se o próximo vôo pertencia a mesma origem) fazendo com que o programa simplesmente criasse todos os aeroportos correspondentes a todas as origens de vôos. Logo depois da correção, houve o teste apenas da criação dos aeroportos:

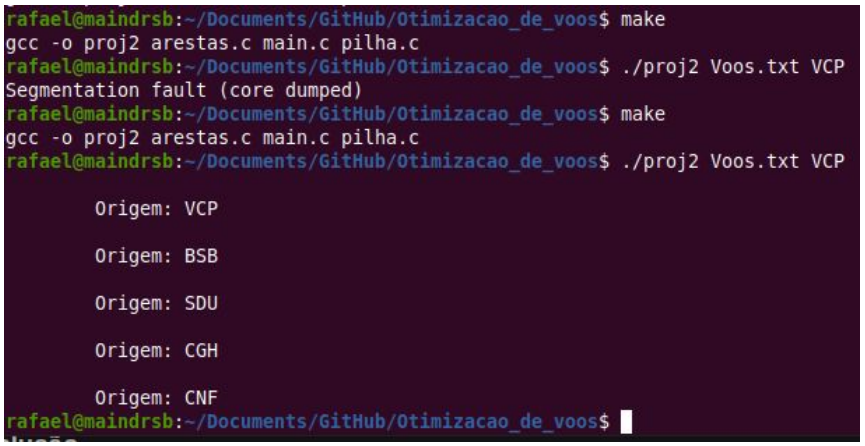

Imagem 5. Teste de criação dos aeroportos.

Notou-se que o teste foi como esperado, sendo apenas cinco aeroportos. Bem como foi realizado o teste para a conferência da criação dos vôos entre os aeroportos, demonstrado a seguir:

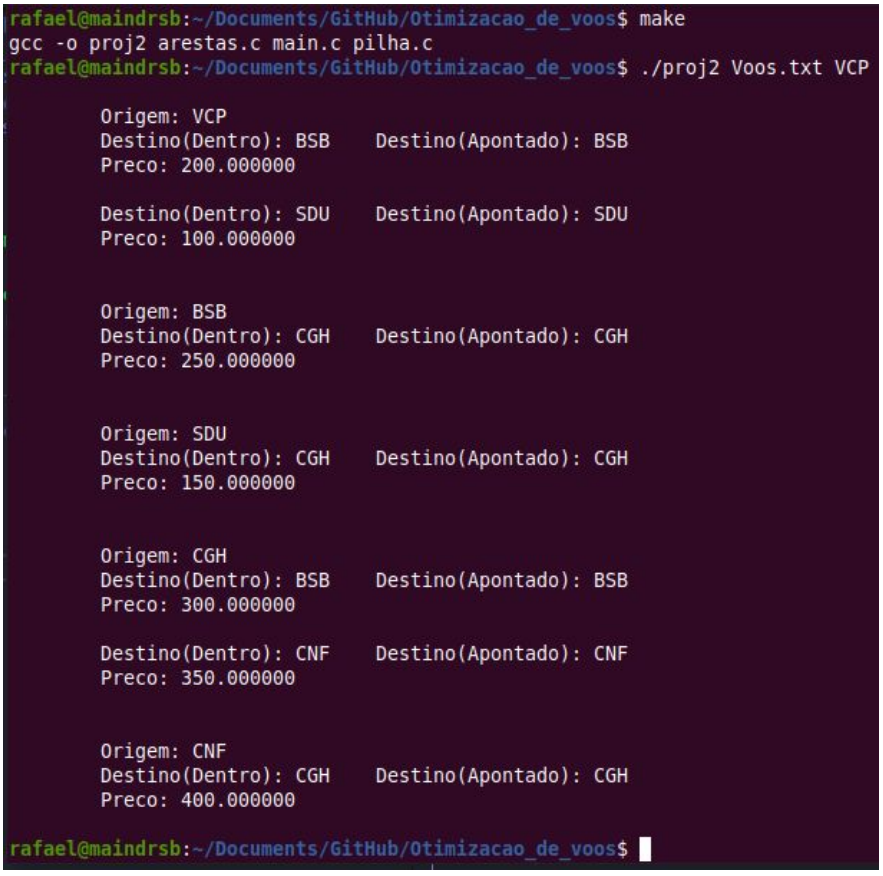

Imagem 6. Teste de criação do grafo.

Ao realizar os testes, foi retirada a string de dentro da estrutura de arestas, pois esta seria apenas uma conferência para os testes realizados. Também realizamos uma adição de uma variável booleana dentro da estrutura do aeroporto, para facilitar a conferência se o mesmo já foi visitado na execução de busca.

Para o primeiro algoritmo de manipulação de grafos, foi realizada a busca em profundidade. Em que inicialmente estava exibindo o aeroporto atual e houve a utilização de uma pilha que, posteriormente, foi considerada desnecessária pois a própria recursão realizava tal função. Para a solução da exibição do aeroporto atual, realizamos um loop com as adjacências, chamando recursivamente a função de busca, evitando, assim, a exibição do aeroporto atual.

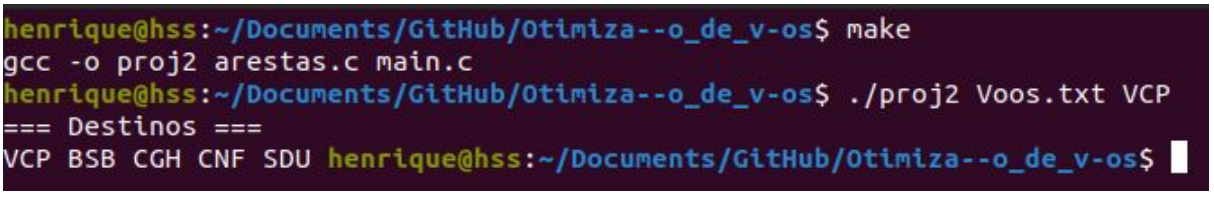

Imagem 7. Teste de busca em profundidade.

Após resolver o erro, também foi realizado o teste para a origem em todos os demais aeroportos, demonstrando os possíveis destinos de cada um a seguir:

Imagem 8. Resultado da busca em profundidade para cada aeroporto.

Com a busca em profundidade bem sucedida, foram feitas as funções para a busca em largura, bem como as funções de fila para a manipulação dos aeroportos, sendo utilizadas neste algoritmo. Entretanto, ao executar o programa para testes, foi detectada uma falha de segmentação. Houve, então a execução do programa Valgrind para detectar possíveis vazamentos de memória.

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ make
gcc -o proj2 arestas.c main.c
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP -> BSB -> CGH
CNF (3): VCP -> BSB -> CGH -> CNF
=== Menores custos ===
Segmentation fault (core dumped)
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ valgrind --leak-check=full
        --show-leak-kinds=all          --track-origins=yes          --verbose
       --log-file=valgrind-out.txt          ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP
CNF (3): VCP  -> CNF
=== Menores custos ===
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$
```

Imagem 9. Checagem de vazamento de memória para a execução da busca em largura.

Ao analisar o arquivo de saída do diagnóstico do Valgrind, foi comprovado o vazamento de memória:

```
==20786== HEAP SUMMARY:
==20786==     in use at exit: 65 bytes in 4 blocks
==20786==   total heap usage: 37 allocs, 37 frees, 10,740 bytes allocated
==20786==
==20786== Searching for pointers to 4 not-freed blocks
==20786== Checked 74,648 bytes
==20786==
==20786== 65 bytes in 4 blocks are definitely lost in loss record 1 of 1
==20786==    at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==20786==    by 0x109900: BFS (in /home/rafael/Documents/GitHub/Otimizacao_de_voos/proj2)
==20786==    by 0x109AF9: conexoes (in /home/rafael/Documents/GitHub/Otimizacao_de_voos/proj2)
==20786==    by 0x109FBC: main (in /home/rafael/Documents/GitHub/Otimizacao_de_voos/proj2)
==20786==
==20786== LEAK SUMMARY:
==20786==    definitely lost: 65 bytes in 4 blocks
==20786==    indirectly lost: 0 bytes in 0 blocks
==20786==      possibly lost: 0 bytes in 0 blocks
==20786==    still reachable: 0 bytes in 0 blocks
==20786==         suppressed: 0 bytes in 0 blocks
```

Imagem 10. Resultado do vazamento de memória.

Ao conferir a função, inicialmente houve a consideração de que estava tudo correspondendo ao esperado no algoritmo.

```
void BFS(fila *f){
    vertice *atual;
    int conexoes, tam;
    char *caminhos = reallocarray(caminhos,4,1);
    atual = removef(f,&conexoes,&caminhos,&tam);

    if(conexoes != 0){
        caminhos = reallocarray(caminhos,tam,1);
            //exit(0);
        caminhos[tam-8] = ' ';
        caminhos[tam-7] = '-';
        caminhos[tam-6] = '>';
        caminhos[tam-5] = ' ';
        caminhos[tam-4] = atual->aero[0];
        caminhos[tam-3] = atual->aero[1];
        caminhos[tam-2] = atual->aero[2];
        caminhos[tam-1] = '\0';
        printf("%s (%d): %s\n",atual->aero, conexoes, caminhos);
    }
```

Imagem 11. Manipulação inicial da função de busca em largura.

Para detectar onde estava o vazamento de memória foram feitas algumas funções para impressão das componentes das estruturas, sendo o grafo e suas variáveis.

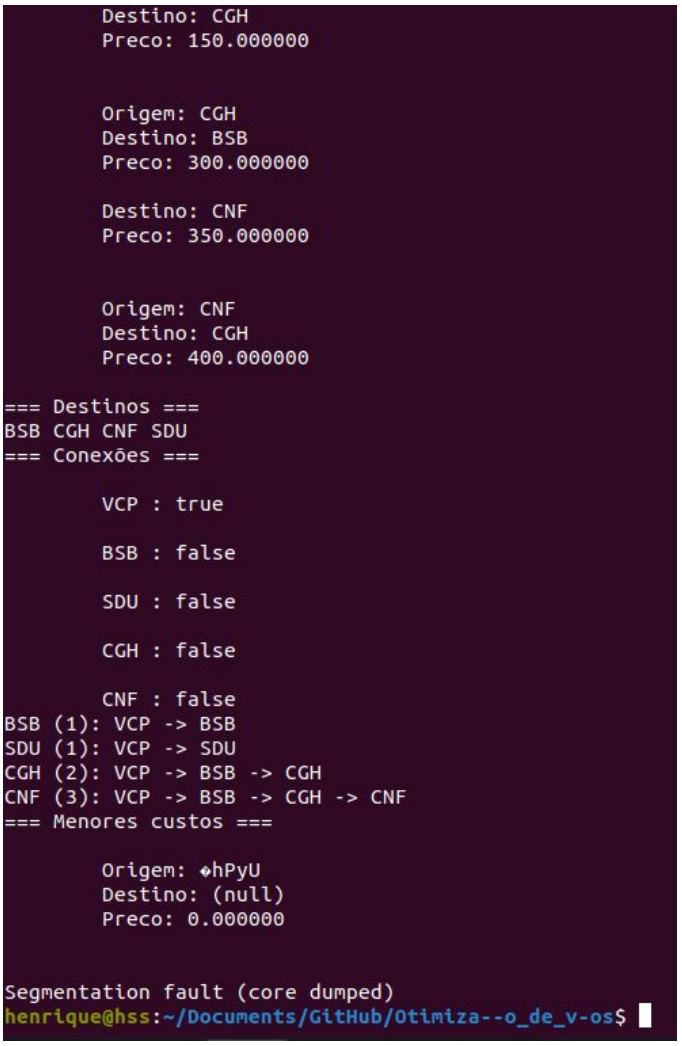

Imagem 12. Exibição das informações das estruturas para verificação (acima o grafo construído, ao meio as variáveis booleanas com o valor atribuído, e embaixo novamente o grafo).

Após a execução com as informações de cada componente exibida na tela, foi detectado que o grafo estava com o vazamento de memória, assim, foi realizada a exibição das informações da fila.

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ make
gcc -o proj2 arestas.c main.c
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===

        SDU VCP 1

        BSB VCP 1
BSB (1): VCP -> BSB

        CGH VCP -> BSB 2

        SDU VCP 1
SDU (1): VCP -> SDU

        CGH VCP -> BSB 2
CGH (2): VCP -> BSB -> CGH

        CNF VCP -> BSB -> CGH 3
CNF (3): VCP -> BSB -> CGH -> CNF
=== Menores custos ===

        Origem: ◆X◆vV
        Destino: (null)
        Preco: 0.000000


Segmentation fault (core dumped)
```

Imagem 13. Exibição dos itens da fila a cada passagem, juntamente com o grafo (este com perda de informações).

As informações da fila estavam com os valores esperados. O vazamento estaria ocorrendo possivelmente no momento de desalocar a memória de alguma componente da fila, por isso, houve a atribuição de NULL para o ponteiro de dentro da fila antes de desalocar a mesma, bem como foi substituída a declaração inicial da cadeia de caracteres que armazenava o caminho percorrido pelo algoritmo de busca em largura.

```c
void BFS(fila *f){
    vertice *atual;
    int conexoes, tam;
    char *caminhos /*= reallocarray(caminhos,4,1)*/ = (char*)malloc(f->ini->conexoes*7+4);
    atual = removef(f,&conexoes,&caminhos,&tam);

    if(conexoes != 0){
        caminhos = reallocarray(caminhos,tam,1);
        caminhos[tam-8] = ' ';
        caminhos[tam-7] = '-';
        caminhos[tam-6] = '>';
        caminhos[tam-5] = ' ';
        caminhos[tam-4] = atual->aero[0];
        caminhos[tam-3] = atual->aero[1];
        caminhos[tam-2] = atual->aero[2];
        caminhos[tam-1] = '\0';
        printf("%s (%d): %s\n",atual->aero, conexoes, caminhos);
    }
```

Imagem 14. Modificação na função de busca em largura.

Depois de realizar tais modificações, o programa foi executado novamente e foi constatado que não houve alterações ou desalocamentos inesperados.

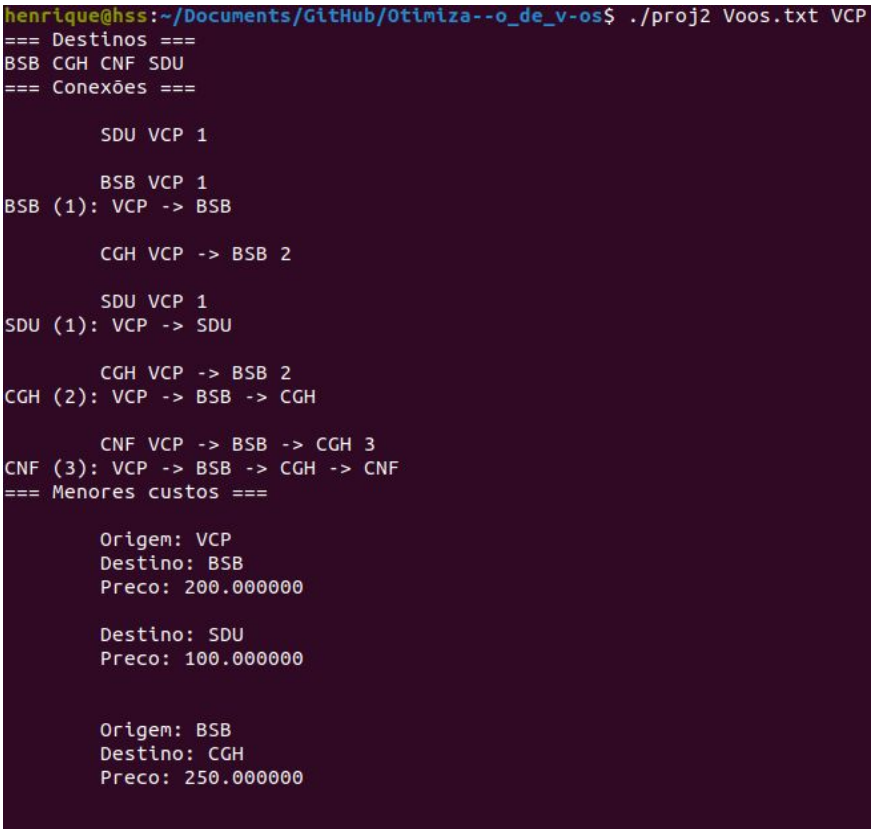

Imagem 15. Verificação das componentes da fila e do grafo após modificação.

Para ter a certeza de que não houvesse vazamentos de memórias, o programa Valgrind foi novamente usado para verificação.

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP -> BSB -> CGH
CNF (3): VCP -> BSB -> CGH -> CNF
=== Menores custos ===
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ valgrind --leak-check=full \
>          --show-leak-kinds=all \
>          --track-origins=yes \
>          --verbose \
>          --log-file=valgrind-out.txt \
>          ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP -> BSB -> CGH
CNF (3): VCP -> BSB -> CGH -> CNF
=== Menores custos ===
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$
```

Imagem 16. Teste para verificação de vazamento de memória.

Depois de verificar o arquivo de saída, foi constatado que o programa não estava com vazamento de memória.

```
==16508== HEAP SUMMARY:
==16508==     in use at exit: 0 bytes in 0 blocks
==16508==   total heap usage: 37 allocs, 37 frees, 10,794 bytes allocated
==16508==
==16508== All heap blocks were freed -- no leaks are possible
==16508==
```

Imagem 17. Programa não teve vazamento de memórias após modificação.

Com a busca em profundidade e a busca em largura funcionando, foram feitas algumas execuções para demonstrar os destinos e conexões de cada aeroporto registrado.

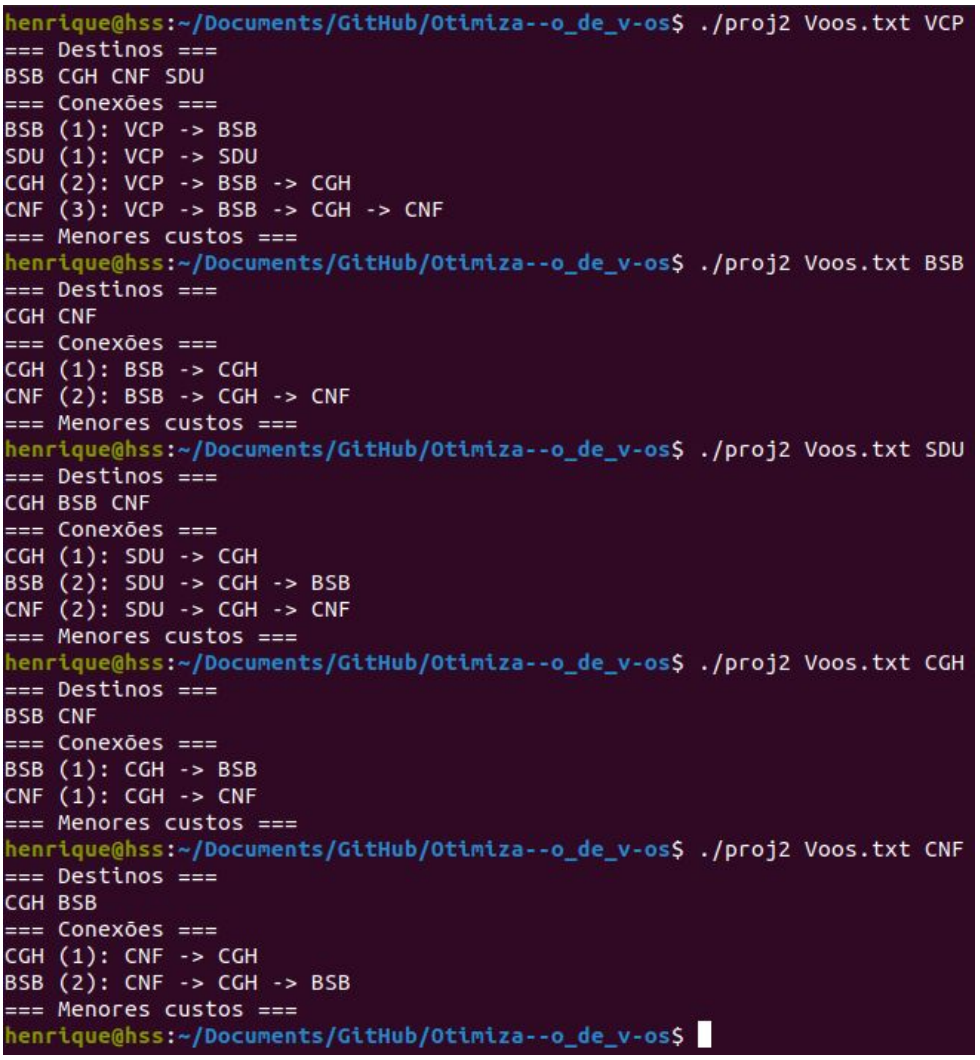

Imagem 18. Execução da busca em largura mostrando as conexões (não considerando o caminho mais curto).

Entretanto, para o planejamento de confecção da função de voos mais baratos, houve a percepção de que poderia haver uma junção dos algoritmos de conexões (busca em largura) e menores custos (algoritmo de caminho mais curto, ou o algoritmo de Dijkstra). Isso fez com que as variáveis das estruturas do grafo e da fila fossem modificadas, ou seja, na estrutura do grafo foi adicionada um ponteiro para guardar a informação do aeroporto anterior, uma variável de ponto flutuante para armazenar o total a ser pago e, também, uma variável inteira para armazenar o número de conexões. Na estrutura da fila, foi removida a cadeia de caracteres que armazenava os aeroportos que realizavam conexões.

```
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ make
gcc -o proj2 arestas.c main.c
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP -> SDU -> CGH
=== Menores custos ===
BSB (R$200.00): VCP -> BSB (R$200.00)
SDU (R$100.00): VCP -> SDU (R$100.00)
CGH (R$250.00): VCP -> SDU (R$100.00) SDU -> CGH (R$250.00) rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$
```

Imagem 19. Tentativa de impressão dos resultados.

Ao analisar a impressão, foi observado que as informações sobre o último aeroporto não estavam sendo exibidas. Tal problema foi resolvido apenas com a alteração na condição de parada do loop na chamada de função para impressão.

```
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ make
gcc -o proj2 arestas.c main.c
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt VCP
=== Destinos ===
BSB CGH CNF SDU
=== Conexões ===
BSB (1): VCP -> BSB
SDU (1): VCP -> SDU
CGH (2): VCP -> SDU -> CGH
CNF (3): VCP -> SDU -> CGH -> CNF
=== Menores custos ===
BSB (R$200.00): VCP -> BSB (R$200.00)
SDU (R$100.00): VCP -> SDU (R$100.00)
CGH (R$250.00): VCP -> SDU (R$100.00) SDU -> CGH (R$250.00)
CNF (R$600.00): VCP -> SDU (R$100.00) SDU -> CGH (R$250.00) CGH -> CNF (R$600.00)
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt BSB
=== Destinos ===
CGH CNF
=== Conexões ===
CGH (1): BSB -> CGH
CNF (2): BSB -> CGH -> CNF
=== Menores custos ===
CGH (R$250.00): BSB -> CGH (R$250.00)
CNF (R$600.00): BSB -> CGH (R$250.00) CGH -> CNF (R$600.00)
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt SDU
=== Destinos ===
CGH BSB CNF
=== Conexões ===
CGH (1): SDU -> CGH
BSB (2): SDU -> CGH -> BSB
CNF (2): SDU -> CGH -> CNF
=== Menores custos ===
CGH (R$150.00): SDU -> CGH (R$150.00)
BSB (R$450.00): SDU -> CGH (R$150.00) CGH -> BSB (R$450.00)
CNF (R$500.00): SDU -> CGH (R$150.00) CGH -> CNF (R$500.00)
```

Imagem 20. Impressão dos resultados.

```
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt CGH
=== Destinos ===
BSB CNF
=== Conexões ===
BSB (1): CGH -> BSB
CNF (1): CGH -> CNF
=== Menores custos ===
BSB (R$300.00): CGH -> BSB (R$300.00)
CNF (R$350.00): CGH -> CNF (R$350.00)
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$ ./proj2 Voos.txt CNF
=== Destinos ===
CGH BSB
=== Conexões ===
CGH (1): CNF -> CGH
BSB (2): CNF -> CGH -> BSB
=== Menores custos ===
CGH (R$400.00): CNF -> CGH (R$400.00)
BSB (R$700.00): CNF -> CGH (R$400.00) CGH -> BSB (R$700.00)
rafael@maindrsb:~/Documents/GitHub/Otimizacao_de_voos$
```

Imagem 21. Continuação da impressão dos resultados.

Acima é possível observar a execução do projeto de acordo com as especificações requeridas pelo enunciado do mesmo. Além disso, o teste foi realizado alterando a origem, testando todos os destinos possíveis do grafo utilizado. Bem como houve a percepção de que antes de realizar o cálculo do caminho mais barato, havia o apontamento errôneo com relação ao caminho realizado nas conexões, todavia, após o desenvolvimento da função que restava, tal exibição foi corrigida.

Para que houvesse a certeza de que o programa estaria funcionando para todas as situações, foi criado um novo grafo contendo quatro aeroportos e quatro vôos como descrito a seguir:

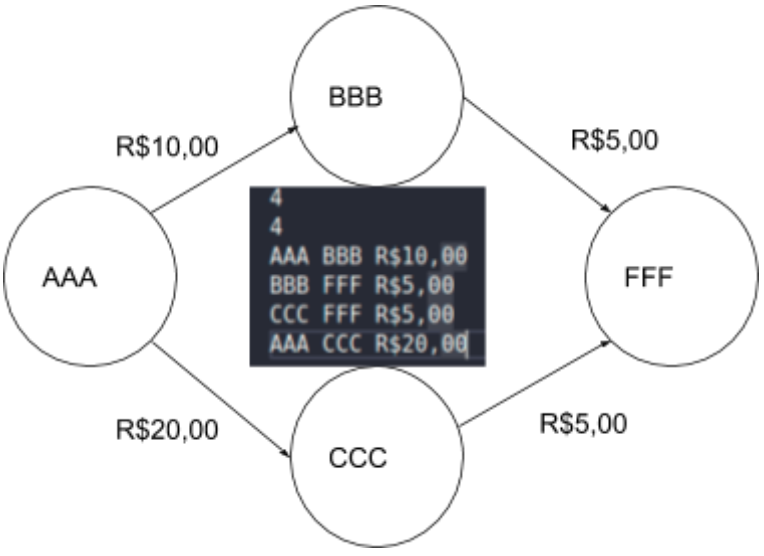

Imagem 22. Representação de um grafo utilizado para testes.

Tal grafo testaria o caminho mais curto e a manipulação da lista de adjacência para aeroportos origem que estariam fora de ordem. Porém a execução não foi como esperada.

```
rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d2866a6fbb04b2d205b60590fbe3312fb
d32a1$ make
gcc -o proj2 arestas.c main.c
rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d2866a6fbb04b2d205b60590fbe3312fb
d32a1$ ./proj2 teste.txt FFF
=== Destinos ===
Segmentation fault (core dumped)
rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d2866a6fbb04b2d205b60590fbe3312fb
d32a1$
```

Imagem 23. Falha de segmentação nos testes por conta da inexistência do aeroporto FFF.

Ao analisar o código, foi evidente que a manipulação para a criação dos aeroportos só abrangia aeroportos que partiam vôos, descartando todos os aeroportos que só recebiam vôos, bem como a criação de aeroportos em ordem sequencial, ou seja, caso houvesse um vôo com o mesmo aeroporto de origem fora de ordem no arquivo texto, o mesmo seria duplicado. A função antiga de criação de um aeroporto é demonstrada a seguir:

```
if(!arq){
    printf("\n\tErro ao abrir o arquivo.");
    exit(0);
}
fscanf(arq, "%d\n", &num_aerop);
fscanf(arq, "%d\n", &num_voos);
//printf("Num. Aeroportos: %d\nNum. Voos: %d", num_aerop, num_voos);
for(int i = num_voos; i > 0 ; i--){// Cria primeiro todos os aeroportos
    fscanf(arq, "%s %s R$%f,%f", origem, destino, &preco, &cent);
    check = strcmp(anterior, origem); // Verifica se há um novo aeroporto
    if(check != 0){
        cria_vertice(&grafo,origem);
        strcpy(anterior, origem);
    }
   //printf("\n\n\t(%d)Origem: %s\n\t(%d)Destino:%s\n\tValor:%.2f", strlen(
}
fclose(arq);
```

Imagem 24. Função inicial para criação dos aeroportos.

Após realizar a manutenção da função, foi novamente feito o teste e a execução resultou como esperada:

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 teste.txt AAA
=== Destinos ===
BBB FFF CCC
=== Conexões ===
BBB (1): AAA -> BBB
CCC (1): AAA -> CCC
FFF (2): AAA -> BBB -> FFF
=== Menores custos ===
BBB (R$10.00): AAA -> BBB (R$10.00)
CCC (R$20.00): AAA -> CCC (R$20.00)
FFF (R$15.00): AAA -> BBB (R$10.00) BBB -> FFF (R$15.00)
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$
```

Imagem 25. Execução bem sucedida do grafo teste.

Além disso, também foram feitas alguns testes com relação a dados de entrada incorretos, em que anteriormente somente o nome do arquivo era tratado numa exceção, ou seja, caso o usuário coloque o nome (juntamente com a extensão do mesmo) errado. Agora os testes se voltaram para o aeroporto de origem, caso o usuário coloque um aeroporto inexistente, anteriormente haveria uma falha de segmentação como demonstrado a seguir:

```
rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d2866a6fbb04b2d205b60590fbe3312fb
d32a1$ ./proj2 Voos.txt XXX
=== Destinos ===
Segmentation fault (core dumped)
rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d2866a6fbb04b2d205b60590fbe3312fb
d32a1$ ./proj2 inexistente.txt VCP

        Erro ao abrir o arquivo.rafael@maindrsb:~/Desktop/Otimizacao_de_voos-83d
2866a6fbb04b2d205b60590fbe3312fbd32a1$ ./proj2 VCP

        Padrão incorreto de parâmetros!rafael@maindrsb:~/Desktop/Otimizacao_de_v
oos-83d2866a6fbb04b2d205b60590fbe3312fbd32a1$
```

Imagem 26. Testes de entrada com nomes e parâmetros fora do padrão.

Para corrigir este possível erro foi criada uma função, logo após a inserção de todas as informações do grafo, na qual há a verificação de correspondência do aeroporto inserido pelo usuário com algum aeroporto presente no grafo, isto é, verifica se o nome do aeroporto de origem é igual ao nome que está no arquivo, conferindo também a existência do mesmo em ambas as entradas.

```
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ make
gcc -o proj2 arestas.c main.c
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voos.txt abc
        Aeroporto de origem inexistente no arquivo.
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voostxt VCP
        Erro ao abrir o arquivo.
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$ ./proj2 Voostxt
        Padrão incorreto de parâmetros!
henrique@hss:~/Documents/GitHub/Otimiza--o_de_v-os$
```

Imagem 27. Teste para o tratamento de entradas erradas.

Para testar se o programa lida com ciclos, foi desenvolvido um outro grafo denotado a seguir:

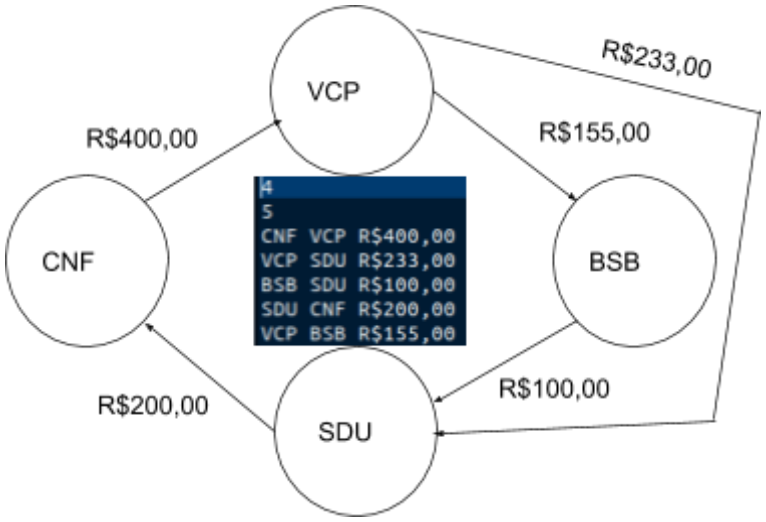

Imagem 28. Representação de um grafo cíclico utilizado para testes.

```
alcides@alcides-note:~/Documentos/projeto2$ ./proj2 Voos.txt VCP
=== Destinos ===
SDU CNF BSB
=== Conexões ===
SDU (1): VCP -> SDU
BSB (1): VCP -> BSB
CNF (2): VCP -> SDU -> CNF
=== Menores custos ===
SDU (R$233.00): VCP -> SDU (R$233.00)
BSB (R$155.00): VCP -> BSB (R$155.00)
CNF (R$433.00): VCP -> SDU (R$233.00) SDU -> CNF (R$433.00)
```

Imagem 29. Execução utilizando grafo cíclico.

```
alcides@alcides-note:~/Documentos/projeto2$ ./proj2 Voos.txt BSB
=== Destinos ===
SDU CNF VCP
=== Conexões ===
SDU (1): BSB -> SDU
CNF (2): BSB -> SDU -> CNF
VCP (3): BSB -> SDU -> CNF -> VCP
=== Menores custos ===
SDU (R$100.00): BSB -> SDU (R$100.00)
CNF (R$300.00): BSB -> SDU (R$100.00) SDU -> CNF (R$300.00)
VCP (R$700.00): BSB -> SDU (R$100.00) SDU -> CNF (R$300.00) CNF -> VCP (R$700.00)
```

Imagem 30. Representação de um grafo cíclico utilizado para testes.

```
alcides@alcides-note:~/Documentos/projeto2$ ./proj2 Voos.txt SDU
=== Destinos ===
CNF VCP BSB
=== Conexões ===
CNF (1): SDU -> CNF
VCP (2): SDU -> CNF -> VCP
BSB (3): SDU -> CNF -> VCP -> BSB
=== Menores custos ===
CNF (R$200.00): SDU -> CNF (R$200.00)
VCP (R$600.00): SDU -> CNF (R$200.00) CNF -> VCP (R$600.00)
BSB (R$755.00): SDU -> CNF (R$200.00) CNF -> VCP (R$600.00) VCP -> BSB (R$755.00)
```

Imagem 31. Representação de um grafo cíclico utilizado para testes.

```
alcides@alcides-note:~/Documentos/projeto2$ ./proj2 Voos.txt CNF
=== Destinos ===
VCP SDU BSB
=== Conexões ===
VCP (1): CNF -> VCP
SDU (2): CNF -> VCP -> SDU
BSB (2): CNF -> VCP -> BSB
=== Menores custos ===
VCP (R$400.00): CNF -> VCP (R$400.00)
SDU (R$633.00): CNF -> VCP (R$400.00) VCP -> SDU (R$633.00)
BSB (R$555.00): CNF -> VCP (R$400.00) VCP -> BSB (R$555.00)
```

Imagem 32. Representação de um grafo cíclico utilizado para testes.

A execução apresentada acima obteve os resultados esperados.

Conclusão

No início do desenvolvimento do projeto, a equipe tinha uma visão de que a implementação de grafos poderia ser complexa, porém, ao decorrer do desenvolvimento, mesmo obtendo alguns erros de manipulação de memória, foi possível implementar a estrutura de grafos, os algoritmos de busca em profundidade

e largura e o algoritmo de Dijkstra (em conjunto com a busca em largura) para cálculo do caminho com menor custo, com sucesso.

O projeto complementou o conhecimento teórico, adquirido ao decorrer do semestre, sobre grafos e seus algoritmos de maneira prática e simples.

Bibliografia

Disponível em:
<https://www.tutorialspoint.com/cprogramming/c_data_types.htm>.