PROJETO DE ALGORITMOS E ESTRUTURA DE DADOS III

MONITOR:

- GABRIEL MAGALHÃES

INTEGRANTES:

- LUCAS TELES DE SOUZA – 94 – GES
- IGOR NOGUEIRA OLÍVIO– 1860 – GEC
- TIAGO AUGUSTO COSTA CARVAHO – 1855 – GEC

SUMÁRIO

INTRODUÇÃO

Para o projeto “Entregas”, foi desenvolvida uma solução para o problema de um supermercado, visando maior eficiência desde o momento de distribuição dos produtos até as entregas, considerando que cada entregador deve seguir uma rota específica e pré-determinada para cada destino afim de realizar as entregas no menor tempo possível, respeitando também a capacidade máxima de peso suportado para cada entregador.

FUNCIONAMENTO DO ALGORITMO

Seguindo o objetivo principal de alocar as todas as compras informadas para os entregadores disponíveis, é intuitivo pensarmos em alocar estas compras da seguinte maneira:

As compras serão distribuídas de acordo com sua distância entre o mercado e o local para entrega, sendo àquela mais distante alocada primeiro. Consequentemente, os entregadores mais próximos do mercado receberão as compras que estão direcionadas para o local mais distante do mercado.

Dessa forma, se o primeiro entregador passar por mais de um local antes de realizar a entrega que está mais distante, é possível alocar para ele realize alguma outra entrega, se existir, que já está em seu caminho e que possui um peso aceitável dado o estado do entregador.

Se esta lógica for propagada para os demais entregadores e compras, poderemos alocar as compras aos entregadores de maneira a termos um melhor aproveitamento do tempo dos entregadores.

Em um caso especial, há a possibilidade de que ao executar o processo descrito acima, alguma(s) das entregas não tenha(m) sido alocada(s) em nenhum entregador e para isso, uma repescagem nas entregas deverá ocorrer verificando qual o melhor entregador (aquele que terá o menor tempo total no final) para realizar aquele entrega.

ANÁLISE DE COMPLEXIDADE

Durante o processo de *'Code Review'* (Revisão de código) foi realizada a análise de complexidade para todos os loops utilizados na estrutura do projeto e funções desenvolvidas.

Todas as descrições e notações O() podem ser encontradas nos comentários de cada bloco de código para devidos fins.

A complexidade final do código é O(n^3), sendo quantidade de Compras * quantidade de Locais ^ 2, dado o uso do algoritmo de Dijkstra para verificação das distâncias dos pontos no grafo.

DEFINIÇÃO DE CASOS PARA TESTES

Teste 1:

6// Quantidade de locais

5// números referente ao local de origem
// Cria as arestas usando o djkstra

1 2 1// origens - destino - tempo

1 3 5

1 4 7

1 5 4

1 6 9

2 3 14

2 4 6

2 5 8

2 6 11

3 4 8

3 5 10

3 6 10

4 5 12

4 6 4

5 6 8

-1 -1 -1// flag para saída do loop

5// Quantidade de entregas
// Local da entrega(destino)-peso

1 7

2 5

3 10

4 9

6 4

4// quantidades de entregadores

18// capacidade máxima da mochila

8//distâncias entregador 1 para o mercado

6//distâncias entregador 2 para o mercado

9//distâncias entregador 3 para o mercado

4//distâncias entregador 4 para o mercado

Teste 2:

5// Quantidade de locais

4// números referente ao local de origem

// Cria as arestas usando o djkstra

1 2 3// origens - destino - tempo

1 3 4

1 4 8

1 5 7

2 3 6

2 4 9

2 5 10

3 4 6

3 5 2

4 5 3

-1 -1 -1// flag para saída do loop

6 // Quantidade de entregas
// Local da entrega(destino)-peso

1 4

2 3

5 6

2 1

3 2

5 7

4// quantidades de entregadores

18// capacidade máxima da mochila

6//distâncias entregador 1 para o mercado

5//distâncias entregador 2 para o mercado

8//distâncias entregador 3 para o mercado

9//distâncias entregador 4 para o mercado

Teste3:

7// Quantidade de locais

3// números referente ao local de origem

// Cria as arestas usando o djkstra

1 2 3// origens - destino - tempo

1 3 9

1 4 8

1 5 2

1 6 7

1 7 6

2 3 4

2 4 6

2 5 3

2 6 1

2 7 10

3 4 5

3 5 11

3 6 5

3 7 4

4 5 9

4 6 3

4 7 4

5 6 9

5 7 6

6 7 7

-1 -1 -1 // flag para saída do loop

6// Quantidade de entregas
// Local da entrega(destino)-peso

1 8

2 5

4 6

7 4

5 9

6 7

5// quantidades de entregadores

15// capacidade máxima da mochila

5//distâncias entregador 1 para o mercado

6//distâncias entregador 2 para o mercado

3//distâncias entregador 3 para o mercado

7//distâncias entregador 4 para o mercado

9//distâncias entregador 5 para o mercado

TESTES E RESULTADOS

Resultado para teste 1:

```
Entregadores:

Entregador 1:
Tempo total: 16
Peso total: 4
Compras entregues: 1
        Compra 6 - 4Kg

Caminho percorrido: 5 6
################################

Entregador 2:
Tempo total: 15
Peso total: 10
Compras entregues: 1
        Compra 3 - 10Kg

Caminho percorrido: 5 1 3
################################

Entregador 3:
Tempo total: 14
Peso total: 5
Compras entregues: 1
        Compra 2 - 5Kg

Caminho percorrido: 5 1 2
################################

Entregador 4:
Tempo total: 19
Peso total: 16
Compras entregues: 2
        Compra 4 - 9Kg
        Compra 1 - 7Kg

Caminho percorrido: 5 1 4
################################
```

Resultado para teste 2:

```
################################
################################
########## Resultados ##########
################################

Entregadores:

Entregador 1:
Tempo total: 14
Peso total: 4
Compras entregues: 1
        Compra 1 - 4Kg

Caminho percorrido: 4 1
################################

Entregador 2:
Tempo total: 23
Peso total: 4
Compras entregues: 2
        Compra 2 - 3Kg
        Compra 2 - 1Kg

Caminho percorrido: 4 2
################################

Entregador 3:
Tempo total: 19
Peso total: 15
Compras entregues: 3
        Compra 3 - 2Kg
        Compra 5 - 6Kg
        Compra 5 - 7Kg

Caminho percorrido: 4 5 3
################################

Entregador 4:
Tempo total: 9
Peso total: 0
Compras entregues: 0

Caminho percorrido: 4
################################
```

Resultado para teste 3:

```
Entregador 1:
Tempo total: 12
Peso total: 9
Compras entregues: 1
        Compra 5 - 9Kg

Caminho percorrido: 3 2 5
################################

Entregador 2:
Tempo total: 11
Peso total: 6
Compras entregues: 1
        Compra 4 - 6Kg

Caminho percorrido: 3 4
################################

Entregador 3:
Tempo total: 14
Peso total: 13
Compras entregues: 2
        Compra 1 - 8Kg
        Compra 2 - 5Kg

Caminho percorrido: 3 2 1
################################

Entregador 4:
Tempo total: 12
Peso total: 7
Compras entregues: 1
        Compra 6 - 7Kg

Caminho percorrido: 3 6
################################

Entregador 5:
Tempo total: 13
Peso total: 4
Compras entregues: 1
        Compra 7 - 4Kg

Caminho percorrido: 3 7
################################
```

Teste utilizando o GPROF:

De acordo com o GPROF, cada amostra do arquivo tem o peso de 0,01 segundos de execução.

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00     1463     0.00     0.00  __gnu_cxx::new_allocator<int>::new_allocator()
  0.00      0.00     0.00     1463     0.00     0.00  __gnu_cxx::new_allocator<int>::~new_allocator()
  0.00      0.00     0.00     1463     0.00     0.00  std::allocator<int>::allocator<std::_List_node<int> >(std::allocator<std::_List_node<int> > const&)
  0.00      0.00     0.00     1463     0.00     0.00  std::allocator<int>::~allocator()
  0.00      0.00     0.00     1424     0.00     0.00  std::_List_base<int, std::allocator<int> >::_M_get_Node_allocator() const
  0.00      0.00     0.00     1186     0.00     0.00  std::_List_base<int, std::allocator<int> >::_M_get_Tp_allocator() const
  0.00      0.00     0.00      949     0.00     0.00  std::_List_const_iterator<int>::operator≠(std::_List_const_iterator<int> const&) const
  0.00      0.00     0.00      926     0.00     0.00  std::_List_iterator<int>::_List_iterator(std::_List_node_base*)
  0.00      0.00     0.00      859     0.00     0.00  operator new(unsigned int, void*)
  0.00      0.00     0.00      756     0.00     0.00  std::list<int, std::allocator<int> >::end()
  0.00      0.00     0.00      705     0.00     0.00  std::vector<Compra, std::allocator<Compra> >::size() const
  0.00      0.00     0.00      695     0.00     0.00  std::vector<Compra, std::allocator<Compra> >::operator[](unsigned int)
  0.00      0.00     0.00      666     0.00     0.00  std::_List_const_iterator<int>::_List_const_iterator(std::_List_node_base const*)
  0.00      0.00     0.00      650     0.00     0.00  std::_List_const_iterator<int>::operator++()
  0.00      0.00     0.00      620     0.00     0.00  std::_List_const_iterator<int>::operator*() const
  0.00      0.00     0.00      613     0.00     0.00  __gnu_cxx::new_allocator<std::_List_node<int> >::~new_allocator()
  0.00      0.00     0.00      593     0.00     0.00  __gnu_cxx::new_allocator<std::_List_node<int> >::deallocate(std::_List_node<int>*, unsigned int)
  0.00      0.00     0.00      593     0.00     0.00  __gnu_cxx::new_allocator<std::_List_node<int> >::allocate(unsigned int, void const*)
  0.00      0.00     0.00      593     0.00     0.00  __gnu_cxx::new_allocator<int>::destroy(int*)
  0.00      0.00     0.00      593     0.00     0.00  __gnu_cxx::new_allocator<int>::construct(int*, int const&)
  0.00      0.00     0.00      593     0.00     0.00  __gnu_cxx::new_allocator<std::_List_node<int> >::max_size() const
  0.00      0.00     0.00      593     0.00     0.00  std::_List_base<int, std::allocator<int> >::_M_get_node()
  0.00      0.00     0.00      593     0.00     0.00  std::_List_base<int, std::allocator<int> >::_M_put_node(std::_List_node<int>*)
  0.00      0.00     0.00      593     0.00     0.00  std::list<int, std::allocator<int> >::_M_create_node(int const&)
  0.00      0.00     0.00      593     0.00     0.00  std::list<int, std::allocator<int> >::_M_insert(std::_List_iterator<int>, int const&)
  0.00      0.00     0.00      574     0.00     0.00  std::list<int, std::allocator<int> >::push_back(int const&)
```

Link para mais detalhes:https://github.com/LucasGitDev/C204-B-2022-1/blob/release/Delivery_Project_1.1.1/Laborat%C3%B3rio/Projeto%201/src/gprof/log.txt

CONCLUSÃO

A solução foi desenvolvida partindo dos conceitos absorvidos em algoritmos 2 e 3, tais como problemas de otimização, programação dinâmica e busca binária. É evidente que conceitos básicos vistos em algoritmos 1 também foram indispensáveis.

Logicamente, alguns conceitos externos foram utilizados, dado o uso de bibliotecas de terceiros e importações de bibliotecas próprias, uma vez que a modularização básica não abrange.

REFERÊNCIAS

https://github.com/LucasGitDev/C204-B-2022-1/tree/release/Delivery_Project_1.1.1

https://github.com/LucasGitDev/C204-B-2022-1/tree/release/Delivery_Project_1.1.1/Laborat%C3%B3rio/Projeto%201/src

https://github.com/LucasGitDev/C204-B-2022-1/blob/release/Delivery_Project_1.1.1/Laborat%C3%B3rio/Projeto%201/src/Dijkstra.cpp

https://github.com/LucasGitDev/C204-B-2022-1/blob/release/Delivery_Project_1.1.1/Laborat%C3%B3rio/Projeto%201/src/ACME-Delivery.cpp

https://www.geeksforgeeks.org/vector-in-cpp-stl/

https://www.cplusplus.com/reference/vector/vector/

https://www.cplusplus.com/reference/algorithm/

https://www.cplusplus.com/reference/string/string/

https://www.cplusplus.com/reference/sstream/