Logística Urbana para Entrega de Mercadorias

GRUPO 39

Bruno Leal – *up202008047*
Tomás Maciel - *up202006845*
Vitor Bizarro – *up202007888*

TRABALHO 1

DA 2021-2022

Descrição do Problema

- Neste trabalho, pretende-se especificar e implementar uma plataforma de gestão de uma empresa de logística urbana, tornando a sua operação o mais eficiente possível, particularmente nos seguintes cenários:
  1. Otimização do número de estafetas, em que se pretende minimizar o número de estafetas para a entrega do maior número de pedidos, num dia.
  2. Otimização do lucro da empresa, em que se pretende maximizar o lucro da empresa para a entrega do maior número de pedidos, num dia, pelos estafetas selecionados.
  3. Otimização das entregas expresso, cujo objetivo principal é minimizar o tempo médio previsto das entregas expresso a serem realizadas num dia.

Descrição das soluções

Formalização – Cenário 1

- Para abordar o cenário 1, começamos por ordenar o vetor de carrinhas por ordem decrescente de tamanho e o vetor de encomendas por ordem crescente de volume e peso, respetivamente. Isto permite-nos organizar os objetos com que vamos trabalhar neste cenário, tendo em conta o objetivo final.
- Depois, iteramos as carrinhas e vimos se a carrinha podia aceitar mais peso e volume, para alocar mais encomendas, e caso isso fosse possível, colocávamos-lhe a encomenda de menor peso, ou, se, não fosse possível, a de menor volume.
- Com isso, conseguimos alocar as encomendas todas nas carrinhas, e, graças à organização inicial dos vetores, minimizamos o número de estafetas na operação.

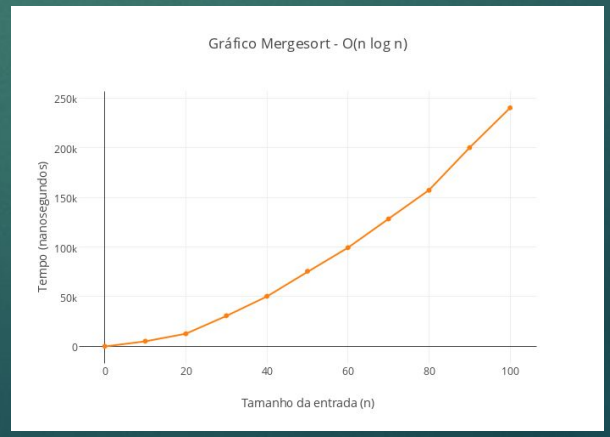
Descrição de algoritmos relevantes – Cenário 1

- Neste cenário, só usamos um algoritmo : algoritmo *greedy*.
- Este tipo de algoritmo é aplicável a problemas de otimização, como é o deste cenário, que se trata de uma minimização, daí a nossa opção por este algoritmo específico.
- No código abaixo, podemos ver a utilização deste mesmo algoritmo.

```
//Avaliar se a carrinha tem mais volume ou peso
if ((*itrcar).volMax >= (*itrcar).pesoMax) {
    //Avaliar se a carrinha consegue aceitar a encomendar de menor peso
    if (!aceitaEncomendas(itrcar, pesos, itrpes, estado, pedidos)) {
        //Avaliar se a carrinhas consegue aceitar a encomenda de menor volume caso, não consiga a de menor peso
        if (!aceitaEncomendas(itrcar, volumes, itrvol, estado, pedidos)) {
            itrcar++;
            //Caso ainda haja carrinhas, passar o pedido para o próximo
            if (itrcar != carrinhas.end()) estafetas++;
            itrcar--;
        }
    }
}
```

Análise das complexidades – Cenário 1

- Neste cenário, a nossa solução tem uma complexidade de temporal de O(n log n) e uma complexidade espacial de O(1).
- A complexidade espacial é O(1) uma vez que não depende do tamanho do input, sendo constante qualquer que seja o input do utilizador.
- A complexidade temporal é O(n log n), pois temos um ciclo *while* dentro de um ciclo de n iterações (neste caso, temos um ciclo *while* dentro de um ciclo *for*).



Resultados empíricos – Cenário 1

- Os resultados que o nosso algoritmo produziu foram 22 estafetas, que pensamos que é o mínimo possível para este cenário.
- Inicialmente, o nosso resultado estava a ser de 24 estafetas mas, após melhorarmos a eficiência do nosso algoritmo, conseguimos alcançar o resultado de 22 estafetas para 450 encomendas (as originais que se encontram no dataset *encomendas.txt* fornecido).

Formalização – Cenário 2

- De modo semelhante ao cenário 1, começamos por organizar os vetores tendo em conta os mesmos critérios.
- Criamos a função abaixo, que nos permite avaliar se uma carrinha é rentável ou não, e se deve ser usada.
- Iteramos as carrinhas e, caso a carrinha em questão tivesse espaço para levar a encomenda, avaliávamos se ela é rentável com a função abaixo. Com isto, conseguimos maximizar o lucro da empresa.

```
//Avaliar se a carrinha atual é rentável ou se não deve ser usada
bool avaliarRentabilidade(vector<carrinha>::iterator &itrcar, int &receitadiaria, int &receita, int &custo){
    if(receitadiaria <= (*itrcar).custo){
        custo -= (*itrcar).custo;
        return true;
    }
    receita += receitadiaria;
    receitadiaria = 0;
    return false;
}
```

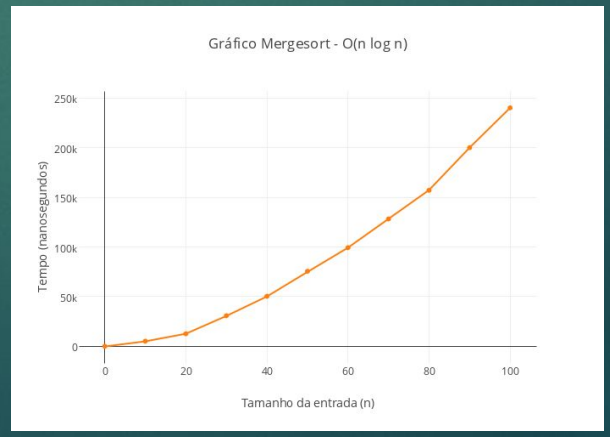
Descrição de algoritmos relevantes – Cenário 2

- Neste cenário, á semelhança do primeiro, abordamos o problema com o algoritmo greedy.
- Como se tratava de um problema de maximização, achamos que era o algoritmo mais adequado a usar.
- Em baixo, temos um exemplo do algoritmo *greedy* no código:

```
else {
    //Avaliar se a carrinha consegue AINDA aceitar a encomendar de menor volume
    if (!aceitaEncomendas2(itrcar, volumes, itrvol, estado, pedidos, receitadiaria)) {
        //Avaliar se a carrinhas consegue AINDA aceitar a encomenda de menor volume caso, não consiga a de menor peso
        if (!aceitaEncomendas2(itrcar, pesos, itrpes, estado, pedidos, receitadiaria)) {
            if (avaliarRentabilidade(itrcar, receitadiaria, receita, custo)) break;
            itrcar++;
            //Caso ainda haja carrinhas, passar o pedido para o próximo
            if (itrcar != carrinhas.end()) custo += (*itrcar).custo;
            itrcar--;
        }
    }
}
```

Análise das complexidades – Cenário 2

- Neste cenário, a nossa solução tem uma complexidade de temporal de O(n log n) e uma complexidade espacial de O(1).
- A complexidade espacial é O(1) uma vez que não depende do tamanho do input, sendo constante qualquer que seja o input do utilizador.
- A complexidade temporal é O(n log n), pois temos um ciclo *while* dentro de um ciclo de n iterações (neste caso, temos um ciclo *while* dentro de um ciclo *for*).



Resultados empíricos – Cenário 2

- Obtivemos o resultado final de 156822 euros de lucro da empresa.
- Foi o melhor valor que conseguimos obter, uma vez que na nossa abordagem inicial apenas obtínhamos um lucro de 102 euros, e, após algumas alterações, conseguimos atingir esse valor.

Formalização – Cenário 3

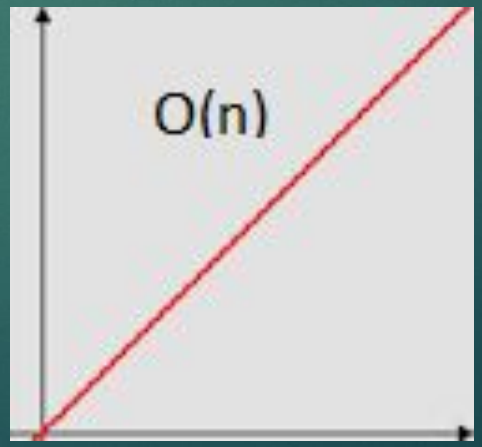
- Neste cenário, começamos por organizar as encomendas num vetor por ordem crescente de duração, uma vez que é a caraterística que nos interessa neste caso.
- Definimos um variável inteira timeleft = 28800 (8 x 60 x 60), que é o número de segundos em que se pode realizar entregas expresso (das 9:00 até às 17:00).
- Depois, iteramos o nosso vetor de encomendas, decrementando a variável timeleft no tempo que a encomenda em questão demora a entregar e incrementando o nosso contador de encomendas.
- Caso o tempo para entregas expresso fosse esgotado (timeleft<=0), incrementávamos o timeleft no tempo que que a encomenda demora a entregar, decrementando o nosso contador de encomendas.

Descrição de algoritmos relevantes – Cenário 3

- Neste cenário, não usamos nenhum algoritmo relevante, tendo sido usada uma abordagem mais rudimentar, em que procuramos obter o objetivo final sem ter em conta os meios.

Análise das complexidades – Cenário 3

- Neste cenário, a nossa solução tem uma complexidade de temporal de O(n) e uma complexidade espacial de O(1).
- A complexidade espacial é O(1) uma vez que não depende do tamanho do input, sendo constante qualquer que seja o input do utilizador.
- A complexidade temporal é O(n), pois temos um ciclo *de n* iterações (neste caso, um ciclo *for*).

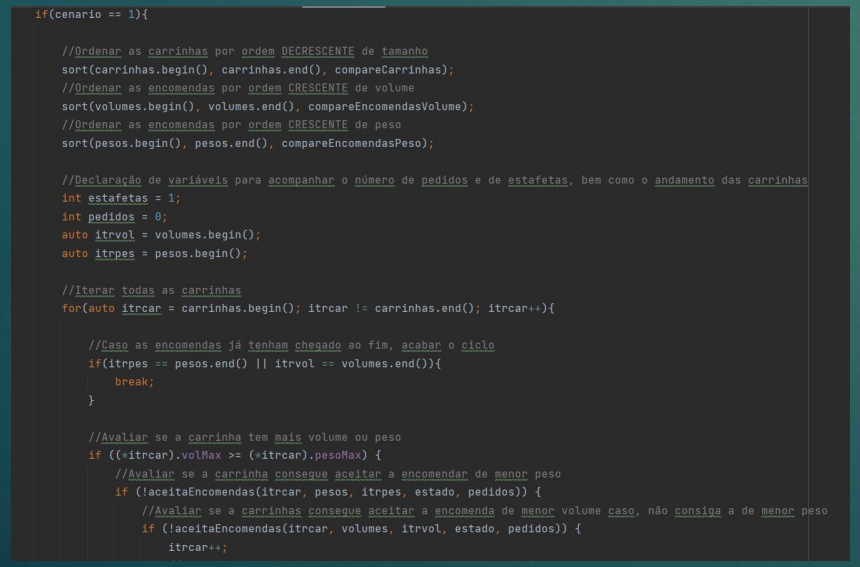


Resultados empíricos – Cenário 3

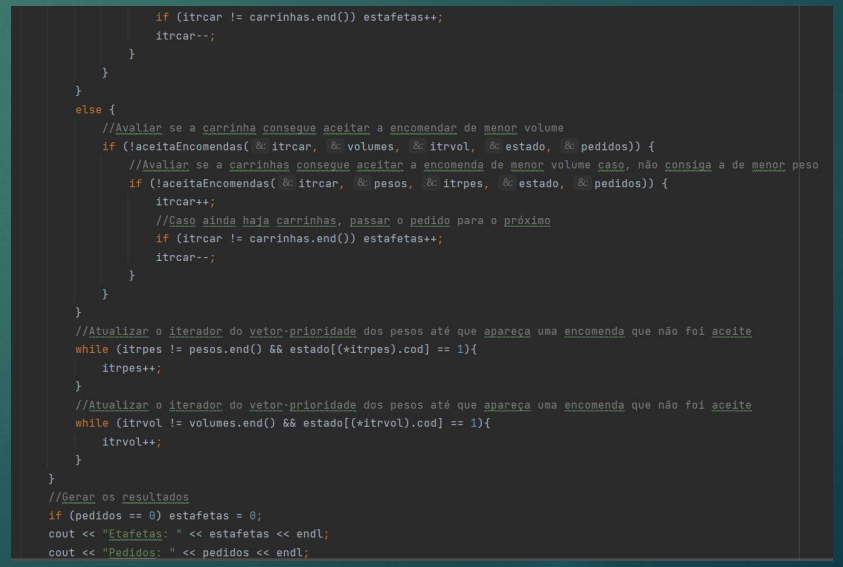
- Neste caso, obtivemos o valor de 231,065 segundos de tempo médio de entrega das entregas expresso.
- Este valor foi obtido logo na nossa primeira abordagem e, devido a alguma falta de tempo e ao facto de termos investido mais tempo nos restantes cenários, acabamos por não tentar melhorar.
- No entanto, achamos que obtivemos um valor próximo do tempo médio mínimo que se poderia obter

Solução algorítmica a destacar

- Resolvemos destacar o nosso algoritmo para o cenário 1, que achamos que foi o mais bem conseguido no nosso trabalho.
- Outro aspeto pelo qual o destacamos é o facto de mostrar perfeitamente a utilização do algoritmo greedy, com uma série de if's, que permitiram fazer a melhor escolha local em cada caso.



```
if(cenario == 1){

    //Ordenar as carrinhas por ordem DECRESCENTE de tamanho
    sort(carrinhas.begin(), carrinhas.end(), compareCarrinhas);
    //Ordenar as encomendas por ordem CRESCENTE de volume
    sort(volumes.begin(), volumes.end(), compareEncomendasVolume);
    //Ordenar as encomendas por ordem CRESCENTE de peso
    sort(pesos.begin(), pesos.end(), compareEncomendasPeso);

    //Declaração de variáveis para acompanhar o número de pedidos e de estafetas, bem como o andamento das carrinhas
    int estafetas = 1;
    int pedidos = 0;
    auto itrvol = volumes.begin();
    auto itrpes = pesos.begin();

    //Iterar todas as carrinhas
    for(auto itrcar = carrinhas.begin(); itrcar != carrinhas.end(); itrcar++){

        //Caso as encomendas já tenham chegado ao fim, acabar o ciclo
        if(itrpes == pesos.end() || itrvol == volumes.end()){
            break;
        }

        //Avaliar se a carrinha tem mais volume ou peso
        if ((*itrcar).volMax >= (*itrcar).pesoMax) {
            //Avaliar se a carrinha consegue aceitar a encomendar de menor peso
            if (!aceitaEncomendas(itrcar, pesos, itrpes, estado, pedidos)) {
                //Avaliar se a carrinhas consegue aceitar a encomenda de menor volume caso, não consiga a de menor peso
                if (!aceitaEncomendas(itrcar, volumes, itrvol, estado, pedidos)) {
                    itrcar++;
```



```
                    if (itrcar != carrinhas.end()) estafetas++;
                    itrcar--;
                }
            }
        }
        else {
            //Avaliar se a carrinha consegue aceitar a encomendar de menor volume
            if (!aceitaEncomendas( &itrcar,  &volumes,  &itrvol,  &estado,  &pedidos)) {
                //Avaliar se a carrinhas consegue aceitar a encomenda de menor volume caso, não consiga a de menor peso
                if (!aceitaEncomendas( &itrcar,  &pesos,  &itrpes,  &estado,  &pedidos)) {
                    itrcar++;
                    //Caso ainda haja carrinhas, passar o pedido para o próximo
                    if (itrcar != carrinhas.end()) estafetas++;
                    itrcar--;
                }
            }
        }
        //Atualizar o iterador do vetor-prioridade dos pesos até que apareça uma encomenda que não foi aceite
        while (itrpes != pesos.end() && estado[(*itrpes).cod] == 1){
            itrpes++;
        }
        //Atualizar o iterador do vetor-prioridade dos pesos até que apareça uma encomenda que não foi aceite
        while (itrvol != volumes.end() && estado[(*itrvol).cod] == 1){
            itrvol++;
        }
    }
    //Gerar os resultados
    if (pedidos == 0) estafetas = 0;
    cout << "Etafetas: " << estafetas << endl;
    cout << "Pedidos: " << pedidos << endl;
```

Principais dificuldades e esforço de cada elemento do grupo

- Na nossa ótica, a principal dificuldade encontrada foi, indiscutivelmente, a implementação do cenário 2, que nos pareceu um pouco mais complexo do que os restantes e nos levou um pouco mais de tempo a concretizar.
- Quanto ao esforço de cada elemento do grupo, pensamos que o trabalho foi bem distribuído entre nós e que todos tivemos um desempenho semelhante, sendo que cada um cumpriu na íntegra a tarefa que lhe foi designada.

FIM