Logística Urbana Para Entrega de Mercadorias

Desenho de Algoritmos

Hugo Gomes – up202004343

João Moreira – up202005035

Lia Vieira – up202005042

Turma 2 Grupo 23

Descrição do problema

No âmbito da unidade curricular de Desenho de Algoritmos do L.EIC, foi-nos desafiado a criação e implementação de um sistema de gestão da empresa de logística urbana, tornando a sua operação o mais eficiente possível.

A plataforma a desenvolver deve:

- Minimizar o número de estafetas para a entrega de todos os pedidos ou do maior número de pedidos, num dia.
- Maximizar o lucro da empresa para a entrega de todos os pedidos ou do maior número de pedidos num dia.
- Minimizar o tempo médio previsto das entregas expresso a serem realizadas num dia.
- Incluir documentação das classes implementadas, usando Doxygen.

Cenário 1

— **Variáveis de decisão:** x_{ij} – indica se o estafeta foi selecionado

c_{ij} – indica se o estafeta j fica com a encomenda i

— **Restrições:** não exceder o peso máximo nem o volume máximo de cada carrinha.

— **Objetivo:** minimizar $\sum x_{ij}$, ou seja, o número de estafetas para a entrega de todos os pedidos ou do maior número de pedidos num dia.

Descrição da solução

— Foi utilizado o algoritmo First Fit Decreasing ordenando os estafetas e as encomendas pelo produto entre o peso e o volume, por ordem decrescente.

— **Complexidade temporal:** O(O x C) O = nº de encomendas C = nº de estafetas

Cenário 2

— **Variáveis de decisão:** x_{en} – número de encomendas a entregar

x_{es} – número de estafetas a utilizar

— **Restrições:** não exceder o peso máximo nem o volume máximo de cada carrinha.

— **Objetivo:** maximizar o valor do lucro $\sum x_{en} - \sum x_{es}$ da empresa num dia.

Descrição da solução

— Foi utilizado o algoritmo First Fit Decreasing onde as encomendas estão ordenadas de forma a que a razão entre a recompensa e o produto do peso pelo volume seja maior. A ordenação dos estafetas segue a mesma linha de raciocínio, porem de forma a que esta razão seja menor.

— **Complexidade temporal:** O(O x C) O = nº de encomendas C = nº de estafetas

Cenário 3

— **Variáveis de decisão:** x_{jk} – indica se uma encomenda foi entregue

x_{te} – tempo de entrega de uma encomenda

— **Restrições:** só pode ser entregue uma encomenda por viagem.

— **Objetivo:** minimizar $\frac{\sum x_{te}}{\sum x_{jk}}$, ou seja, tempo médio previsto das entregas expresso a serem realizadas num dia.

Descrição da solução

— Foi utilizado um algoritmo ganancioso de escalonamento de atividades de modo a minimizar o tempo médio de conclusão de cada encomenda expresso. Para tal, as encomendas foram ordenadas de forma crescente pelo tempo.

— **Complexidade temporal:** O(n x log(n))

Resultados da avaliação empírica

— Para todos os cenários foram utilizados diversos algoritmos e vários datasets de diferentes dimensões. Os referidos anteriormente foram os que apresentaram os melhores resultados e mais consistentes.

Destaque de funcionalidade

O método ao lado apresentado tem como função calcular o melhor tempo médio previsto das entregas expresso a serem realizadas num dia.

```
void ManageOrders::averageTime() {
    int timeTotal = 0;
    int time = 0;
    int num = 0;

    if(orders.empty()){
        cout << "No express deliveries" << endl;
        cout << endl;
        return;
    }
    sort( first: orders.begin(), last: orders.end(), comp: compareOrdersTime());

    while(((timeTotal + orders.front().getTime())  < (3600 * 8)) && !orders.empty()){
        timeTotal += orders.front().getTime();
        time += timeTotal + orders.front().getTime();
        num++;
        if(((time / num) / (60*60))>= 1){
            cout << "OrderId:" <<  orders.front().getId() << " hour: " << (time/(num)) / (60*60) << " min: " << (((time/(num)) / (60)) %60)
                                                                                                    << " " << "sec: " << (time/num) % 60 << endl;
        }else {
            cout << "OrderId:" << orders.front().getId() << " min: " << (time / num) / (60) << " " << "sec: "<< (time / num) % 60 << endl;
        }
        orders.erase( position: orders.begin());
    }
}
```

Funcionalidades extra

O código em baixo apresentado tem como função calcular a eficiência da operação da empresa, ou seja, o quociente entre o número de pedidos entregues e o número de pedidos recebidos num dia.

```
cout << "Percentage of delivered orders: " << (double) (delivered * 100) / total << "%" << endl;
cout << endl;
```

```
Percentage of delivered orders: 78.6667%
```

Principais Dificuldades

Ao longo do projeto, as principais dificuldades surgiram na validação dos resultados obtidos e na escolha dos algoritmos mais eficientes.

O trabalho foi dividido igualmente entre os membros do grupo para a criação das classes e, respetivos, métodos, documentação e implementação.