



# **trabalho DA: Gestão de distribuição em uma companhia de encomendas**

T06\_G62

Diogo Babo up202004950

João Oliveira up202004407

João Pinheiro up202008133

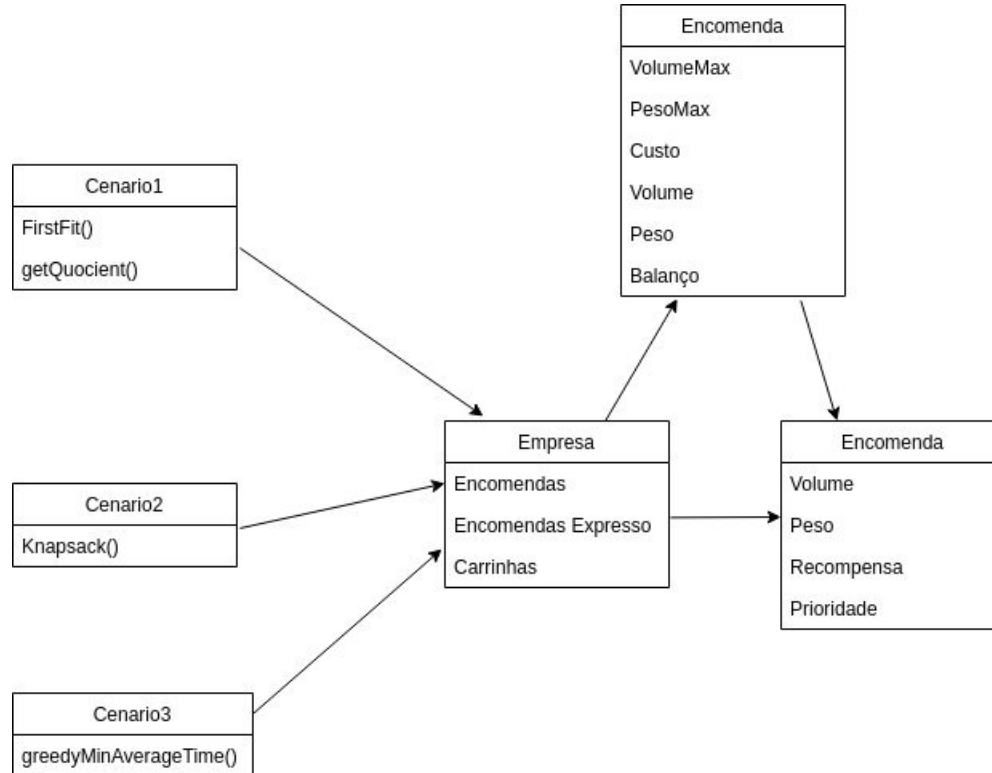


## O problema proposto:

“Uma empresa com base tecnológica pretende inovar, criando uma plataforma eletrónica de crowdsourcing para a entrega de mercadorias em zonas urbanas. A empresa tem o seu próprio armazém, onde recebe e mantém as mercadorias enviadas pelos fornecedores, que ficam a aguardar o transporte para o destino final.

A empresa realiza dois tipos de serviços, nomeadamente a entrega normal e a entrega expresso”

# Diagrama de classes






## Conceitos chave:

Variável decisiva : o balanço entre o peso e o volume total das encomendas relativamente a uma encomenda.

Variável decisiva com a recompensa : razão entre a variável decisiva e a recompensa da encomenda.

Variável decisiva com o custo: razão entre a variável decisiva e o custo de uma carrinha.



## Cenário 1 (objetivo)

- Otimizar / minimizar o número de estafetas necessários para entregar as encomendas, ou seja, maximizar o número de encomendas que cada estafeta entrega.
- Este problema assemelha-se ao problema do “bin-packing”, sendo que nós decidimos utilizar um algoritmo de First Fit Decreasing.

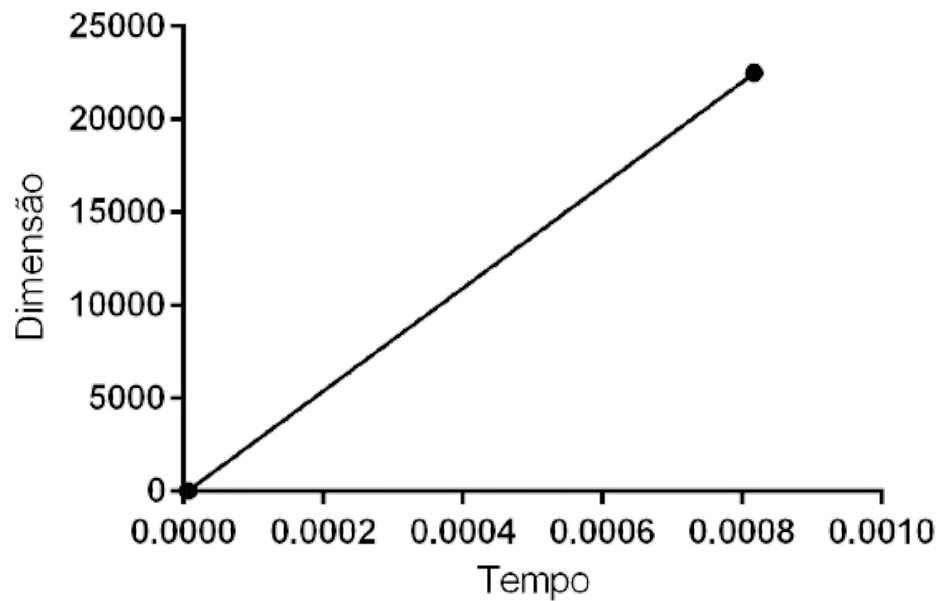
# Algoritmo cenário 1: (First Fit Decreasing)

- Ordenar de forma decrescente as encomendas e as carrinhas. (Pela “variável decisiva”.
- O algoritmo percorre para cada carrinha as encomendas, caso esta possa ser colocada na carrinha então é adicionada, caso não consiga ser colocada ou esta já tenha sido entrega então passa-se à próxima encomenda.
- Time complexity:  $O(n^2)$ .

```
int Cenario1::tentativa() {
    sort(first: encomendas.begin(), last: encomendas.end(), comp: sortByVarEncomenda);
    sort(first: carrinhas.begin(), last: carrinhas.end(), comp: sortByVarCarrinha);
    int numEntregues = 0;
    int numCarrinhas = 0;
    auto start :time_point<...> = std::chrono::high_resolution_clock::now();
    for(auto & carrinha :Carrinha * & : carrinhas) {
        for(auto & encomenda :Encomenda * & : encomendas) {
            if(encomenda->getEstado()) {
                continue;
            }
            if (!carrinha->verificaDisponibilidade(encomenda)) {
                carrinha->adicionarEncomenda(encomenda);
            }
        }
    }
    for(auto x :Carrinha * : carrinhas) {
        if(!x->getEncomendas()->empty()) {
            numCarrinhas++;
        }
    }
    auto end :time_point<...> = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> diff = end-start;
    cout << "Time passed: " << diff.count() << endl;
    return numCarrinhas;
}
```

# Algoritmo cenário 1: (Avaliação Empírica)

- Dataset Default: 0.00081717s  
(dimensão 450 x 50)
- Dataset Reduzido: 6.39e-06s  
(dimensão 10 x 3)





## Cenário 2 (objetivo)

- Pretende-se, seleccionar os estafetas que, para os pedidos de um determinado dia, irão maximizar o lucro da empresa naquele dia.
- Decidimos utilizar um algoritmo de knapsack 0-1 multidimensional com programação dinâmica.



# Algoritmo cenário 2: (Knapsack 0-1)

- Receber um vector com encomendas já ordenadas pela Variável decisiva com a recompensa e uma carrinha.
- O algoritmo percorre a tabela dp preenchendo da forma “bottom up” ou seja, começar pelo caso mais trivial para o mais complexo.
- Este processo é repetido até não haver mais carrinhas ou até as carrinhas não constituírem lucro
- Time complexity:  $O(V * P * E)$ ;
- Space complexity:  $O(V * P)$ ;

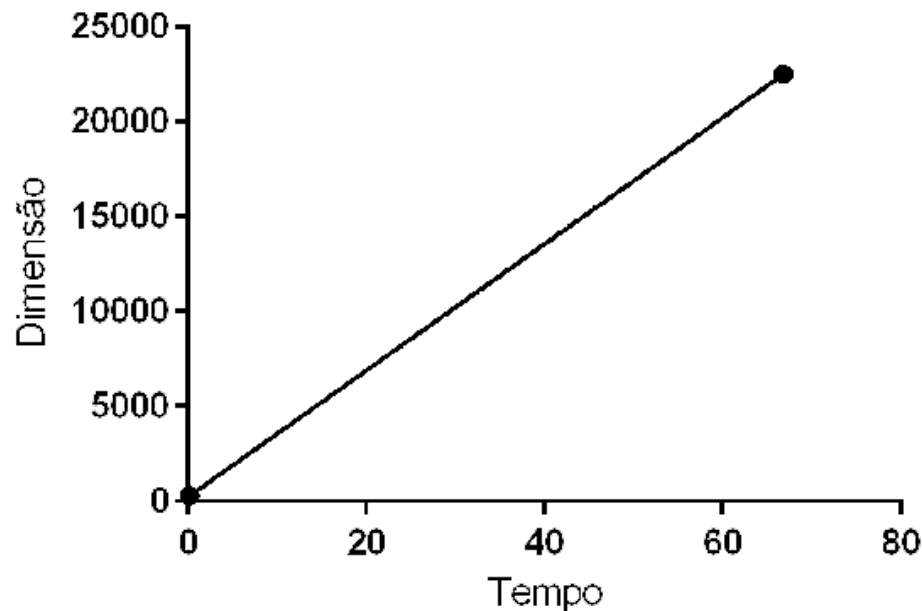
```
ENCOMENDA_VALOR Cenário2::solveKnapsack(Carrinha &c, vector<Encomenda >> encomendas) {
    if (!c.getEncomendas().empty()) return {};
    int n = (int) encomendas.size();
    // dp[index][volume][peso]
    vector<vector<vector<ENCOMENDA_VALOR>>> dp(n, 2, vector<vector<ENCOMENDA_VALOR>>(n, c.getVolMax() + 1, vector<ENCOMENDA_VALOR>(n, c.getPesoMax() + 1, value: ENCOMENDA_VALOR())));

    int cGetVol = (int) c.getVolMax(), cGetPeso = (int) c.getPesoMax();

    for (int i = 0; i <= n; i++) {
        int eVol, ePes, eRec; // vars da encomenda a ser iterada
        if (i != 0) {
            if (encomendas[i - 1] ->getEstado()) { // entregue
                continue;
            }
            eVol = (int) encomendas[i - 1] ->getVol(), ePes = (int) encomendas[i - 1] ->getPeso(), eRec = (int) encomendas[i - 1] ->getRecompensa();
        }
        // preencher tabela dynamic programming
        for (int v = 0; v <= cGetVol; v++) {
            for (int w = 0; w <= cGetPeso; w++) {
                if (i == 0 || v == 0 || w == 0) {
                    dp[i % 2][v][w].profit = 0;
                    continue;
                }
                if (eVol <= v && ePes <= w) { // encomenda cabe na célula da tabela
                    // adicionar encomenda atualmente a ser iterada
                    if ((encomendas[i - 1] ->getPrioridade() ? eRec + 4 : eRec) +
                        dp[(i - 1) % 2][v - eVol][w - ePes].profit > dp[(i - 1) % 2][v][w].profit) {
                        dp[(i - 1) % 2][v][w] = dp[(i - 1) % 2][v - eVol][w - ePes];
                        dp[(i - 1) % 2][v][w].profit += encomendas[i - 1] ->getPrioridade() ? eRec + 4 : eRec;
                        dp[(i - 1) % 2][v][w].CarrinhaEncomenda.push_back(encomendas[i - 1]);
                    } else { // nao compensa adicionar encomenda atualmente a ser iterada
                        dp[(i - 1) % 2][v][w] = dp[(i - 1) % 2][v][w];
                    }
                } else { // encomenda a ser iterada nao cabe, dp igual a linha anterior
                    dp[(i - 1) % 2][v][w] = dp[(i - 1) % 2][v][w];
                }
            }
        }
    }
    return dp[n % 2][cGetVol][cGetPeso];
}
```

## Algoritmo cenário 2: (Avaliação Empírica)

- Dataset Default: 66.8389  
(dimensão 450 x 50)
- Dataset Reduzido: 0.219354  
(dimensão 10 x 3)





## Cenário 3 (objetivo)

- Otimizar / minimizar o tempo médio previsto das entregas expresso a serem realizadas num dia.
- Este problema assemelha-se ao problema do “job scheduling”, onde os melhores resultados seriam obtidos ordenando as encomendas por menor duração.

# Algoritmo cenário 3: (Greedy)

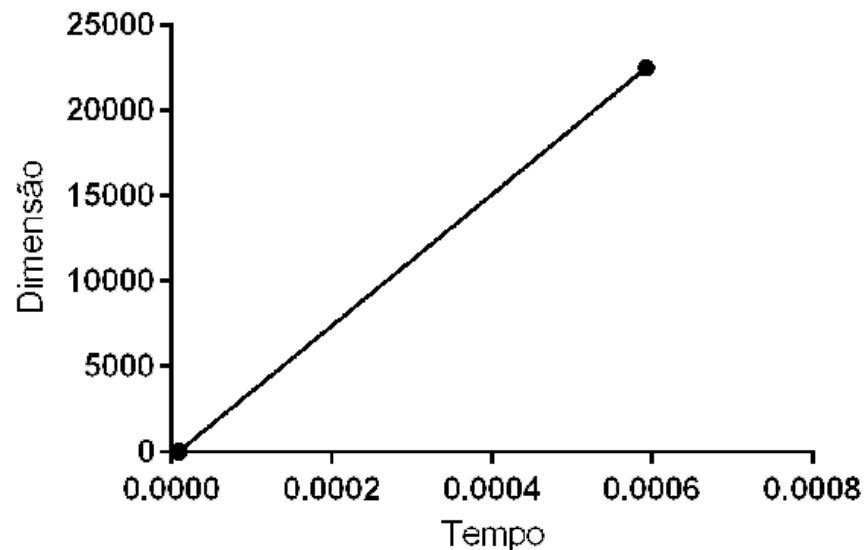
- Ordenar de forma crescente de duração as encomendas expresso. (greedy)
- O nosso algoritmo funciona para além de um dia, ou seja, enquanto as encomendas não forem todas entregues ele vai continuar no dia a seguir. (com prioridade)
- Time complexity:  $O(n)$ .

```
vector<int> Cenario3::greedyMinAvgTime() {
    sort(encomendasExp.begin(), encomendasExp.end(), [](const auto& a, const auto& b) { return a.getDuracao() < b.getDuracao(); });
    double sum = encomendasExp[0] -> getDuracao();
    int startTime = 32400 + (int) encomendasExp[0] -> getDuracao(); // corresponde a 09:00 + a 1st encomenda
    encomendasExp[0] -> setEstado( "entregue: true"); // entregue
    int endTime = 61200; // corresponde a 17:00
    int avgTime;
    int numEncomendas = 1;
    int temp = 0;
    vector<int> avgTimeAllDays;
    for(int i = 1; i < encomendasExp.size(); i++) {
        if(startTime + (int) encomendasExp[i] -> getDuracao() <= endTime) {
            temp += (int) encomendasExp[i-1] -> getDuracao();
            sum += temp + encomendasExp[i] -> getDuracao();
            numEncomendas++;
            startTime += (int) encomendasExp[i] -> getDuracao();
            encomendasExp[i] -> setEstado( "entregue: true");
        }
        else {
            avgTime = (int) sum / numEncomendas;
            avgTimeAllDays.push_back(avgTime);
            quociente.push_back((numEncomendas / (double) (numEncomendas + encomendasPorEntregar())));
            startTime = 32400 + (int) encomendasExp[i] -> getDuracao();
            if(startTime > endTime) {
                break;
            }
            if(i == encomendasExp.size() - 1) {
                avgTimeAllDays.push_back((int) encomendasExp[i] -> getDuracao());
                quociente.push_back((numEncomendas / (double) (numEncomendas + encomendasPorEntregar())));
                break;
            }
            numEncomendas = 1;
            temp = 0;
            sum = encomendasExp[i] -> getDuracao();
            encomendasExp[i] -> setEstado( "entregue: true");
        }
    }

    avgTime = (int) sum / numEncomendas;
    if(avgTimeAllDays.empty() || numEncomendas < encomendasExp.size()) {
        avgTimeAllDays.push_back(avgTime);
        quociente.push_back((numEncomendas / (double) (numEncomendas + encomendasPorEntregar())));
    }
    return avgTimeAllDays;
}
```

## Algoritmo cenário 3: (Avaliação Empírica)

- Dataset Default: 0.000592051  
(dimensão 450 x 50)
- Dataset Reduzido: 8.982e-06  
(dimensão 10 x 3)



# Lista de funcionalidades: CRUD

## Create:

```
int Carrinha::adicionarEncomenda(Encomenda *encomenda) {
    if(verificaDisponibilidade(encomenda)) {
        return 1; // encomenda não cabe na carrinha
    }

    encomenda->setEstado(true);
    encomendas.push_back(encomenda);
    vol+=encomenda->getVol();
    peso+=encomenda->getPeso();
    balanço+=(int) encomenda->getRecompensa();
    return 0;
}
```

## Read:

```
void Empresa::lerEncomendas(std::string *fileName) {
    fileEncomendas=*fileName;
    string s,tempo;
    string vol,peso,recompensa;
    ifstream file;
    file.open(*fileName);
    getline(file,s);
    while(getline(file,s)) {
        std::stringstream str(s);
        getline(str,vol,' ');
        getline(str,peso,' ');
        getline(str,recompensa,' ');
        getline(str,tempo,' ');
        auto *encomenda = new Encomenda(stoi(vol),stoi(peso),stoi(recompensa));
        auto *expEncomenda = new ExpressoEncomenda(stoi(vol),stoi(peso),stoi(recompensa), stoi(tempo));
        encomendas.push_back(encomenda);
        expEncomendas.push_back(expEncomenda);
    }
    file.clear();
    file.seekg(0, ios::beg);
    file.close();
}
```

## Update

&

## Delete:

```
void Empresa::atualizaCarrinhas() {
    for(auto & carrinha : carrinhas){
        auto v = carrinha->getEncomendas();
        for(auto & n : *v){
            carrinha->removeEncomenda(n);
        }
    }
}
```

```
void Empresa::removerEntregues() {
    for(auto itr= encomendas.begin(); itr!=encomendas.end(); ) {
        auto temp = itr;
        itr++;
        if((*temp)->getEstado()) encomendas.erase(temp);
    }
    for(auto itr= expEncomendas.begin(); itr!=expEncomendas.end(); ) {
        auto temp = itr;
        itr++;
        if((*temp)->getEstado()) expEncomendas.erase(temp);
    }
}
```

# Funcionalidade Extra:



```
Número de carrinhas usadas para entregar: 22  
Percentagem encomendas entregues: 100.00%
```

```
Dia: 1  
Tempo Médio: 11729  
Percentagem encomendas entregues: 27.56%
```

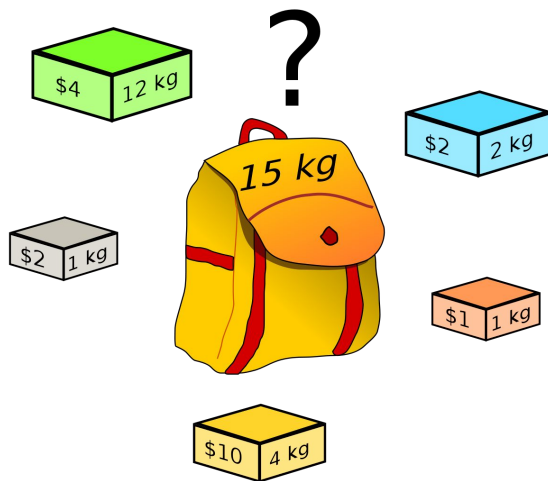
```
Total de carrinhas: 14, Balanco: 231621, Encomendas entregues: 335  
231621  
Percentagem encomendas entregues: 74.44%
```

Das funcionalidades extras que implementamos, decidimos dar destaque a esta que mostra ao utilizador a razão entre as encomendas entregues e as totais.

## Principais dificuldades:

1. Cenário 1;

2. Cenário 2;







# Obrigado pela atenção

Diogo Babo 33.3%  
João Oliveira 33.3%  
João Pinheiro 33.3%