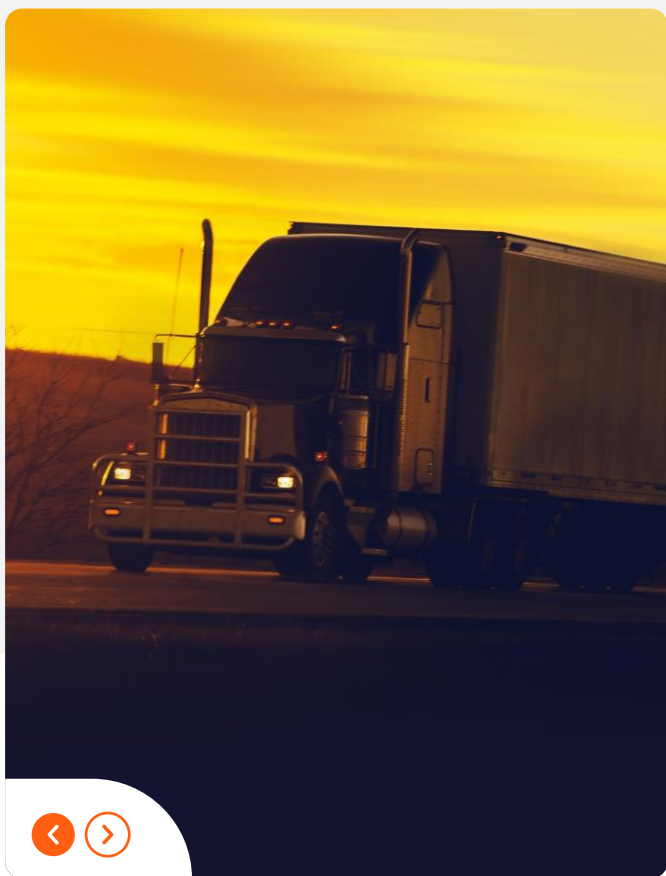


Logística

Logística Urbana para Entrega de Mercadorias

Miguel Tavares - 202002811
Sofia Sousa - 202005932
Pedro Correia - 202006199





Descrição dos Problemas

Pretendemos implementar os seguintes cenários para melhorar a qualidade de gestão da empresa de logística:

24h

1 Minimizar número de estafetas para entrega de pedidos

24h

2 Maximizar Lucro da Empresa

9h às 17h

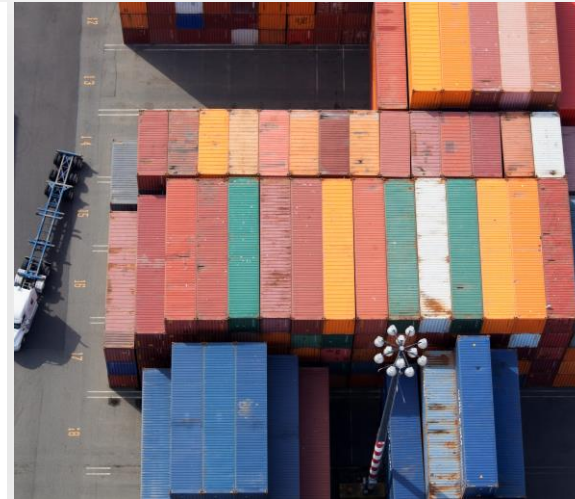
3 Minimizar tempo médio de entregas expresso a serem realizadas em um dia

Formalização **Cenário 1**



Dados de entrada

- Vetor<Encomenda> Enco-
Conjunto de encomendas
caracterizadas por:
 - Peso
 - Volume
 - Recompensa
 - Duração



Dados de entrada

- Vetor<Carrinha> C –Conjunto de
carrinhas caracterizadas por:
 - PesoMax
 - VolMax
 - Custo
 - PesoAtual
 - VolAtual



Dados de saída

- Vetor<Encomenda> P- Conjunto
de pedidos que vão ser entregues
caracterizadas por (Recompensa e
duração irrelevantes):
 - Peso
 - Volume



Dados de saída

- Vetor<Carrinha> Est–Conjunto de
carrinhas usadas, caracterizadas
por (Custo, PesoAtual e VolAtual
irrelevantes):
 - PesoMax
 - VolMax

Restrições e Domínios de valores:

- Minimizar: $\sum est, est \in Est$
- Dentro de um estafeta:
 $\sum(p.Peso, p.Volume) \leq$
 $(est.PesoMax, est.VolMax),$
 $p, est \in P, Est$ respetivamente

Descrição da Solução



Baseado no algoritmo de bin-packing: first fit (decreasing).

Após a ordenação das carrinhas e encomendas por ordem decrescente de capacidade (peso+volume), percorremos todos os camiões um a um e inserimos as encomendas que cabem no camião atual.

```
procedure FitsInTruck(Carrinha c, Encomenda e)
    if ((c.VolAtual-e.Vol)>=0 AND (c.PesoAtual-e.Peso)>=0)
        c.VolAtual <- c.VolAtual-e.Vol
        c.PesoAtual <- c.PesoAtual-e.Peso
        return 1

    return 0

for each c ∈ C do
    flag <- 0
    for each e ∈ E do
        if FitsInTruck(c,e) then
            P.push_back({e.peso,e.vol})
            E.erase(e);
            flag <- 1

    if flag == 1 then
        Est.pushback(c.peso,c.vol)
```

Pseudo código da função aplicada

Complexidade



Complexidade temporal do algoritmo: $O(n^3)$

Complexidade espacial do algoritmo: $O(n)$

```
procedure FitsInTruck(Carrinha c, Encomenda e) //O(1)
    if ((c.VolAtual-e.Vol)>=0 AND (c.PesoAtual-e.Peso)>=0)
        c.VolAtual <-c.VolAtual-e.Vol
        c.PesoAtual <- c.PesoAtual-e.Peso
        return 1

    return 0

for each c ∈ C do // O(n)
    flag <-0
    for each e ∈ E do // O(n)
        if FitsInTruck(c,e) then
            P.push_back({e.peso,e.vol}) // O(1)
            E.erase(e); // O(n)
            flag <- 1

    if flag == 1 then
        Est.pushback(c.peso,c.vol)
```

Pseudo código da função aplicada

Avaliação Empírica

Ficheiros: encomendas.txt e carrinhas.txt

Nº Estafetas: 28
Peso Total: 8574
Vol total: 8422
Nº Pedidos: 450
Peso total: 7339
Vol total: 7123
Eficiência=100%

Decrescente

Nº Estafetas: 22
Peso Total: 7864
Vol total: 7866
Nº Pedidos: 450
Peso total: 7339
Vol total: 7123
Eficiência=100%

Ficheiros: encomendas3.txt e carrinhas3.txt

Nº Estafetas: 1
Peso Total: 39
Vol total: 39
Nº Pedidos: 6
Peso total: 29
Vol total: 32
Eficiência=54.54%

Crescente

Nº Estafetas:1
Peso Total:39
Vol total: 39
Nº Pedidos: 2
Peso total: 17
Vol total: 39
Eficiência=18.18%

Decrescente

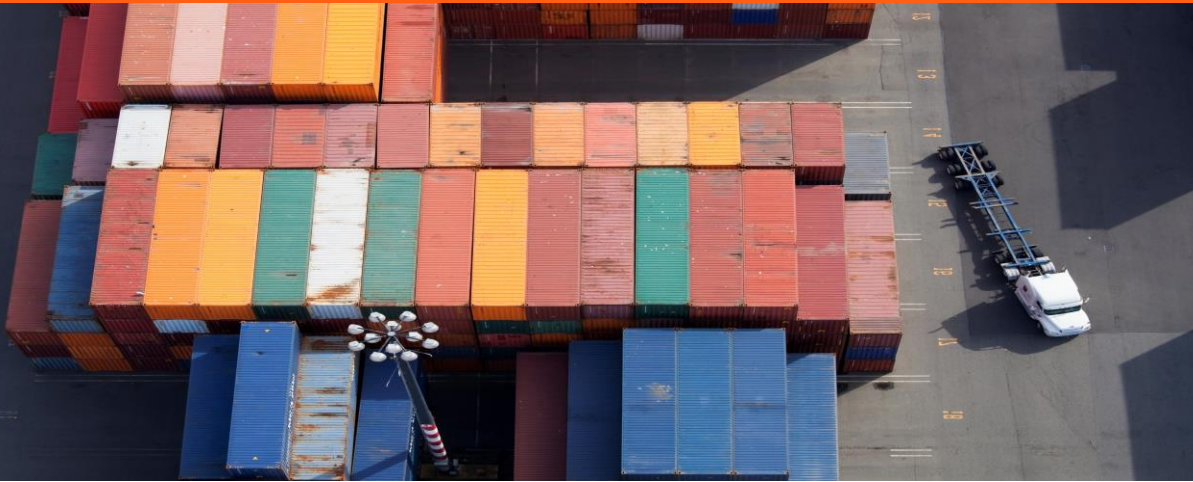
Comparar Crescente

```
bool compararCarrinhas(Carrinha a, Carrinha b){  
    return (a.getVolMax()+a.getPesoMax())<=(b.getVolMax()+b.getPesoMax());  
}  
  
bool compararEnc(Encomenda a, Encomenda b){  
    return((a.getPeso()+a.getVol())<=(b.getPeso()+b.getVol()));  
}
```



Comparar Decrescente

```
bool compararCarrinhas(Carrinha a, Carrinha b){  
    return (a.getVolMax()+a.getPesoMax())>=(b.getVolMax()+b.getPesoMax());  
}  
  
bool compararEnc(Encomenda a, Encomenda b){  
    return((a.getPeso()+a.getVol())>=(b.getPeso()+b.getVol()))  
}
```

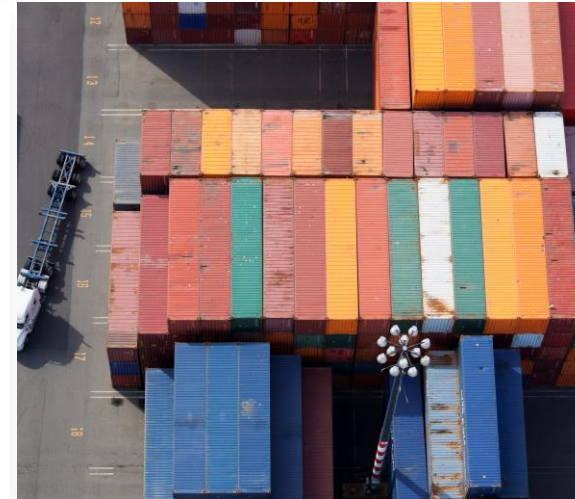


Formalização **Cenário 2**



Dados de entrada

- Vetor<Encomenda> Enco-
Conjunto de encomendas
caracterizadas por:
 - Peso Volume
 - Recompensa
 - Duração



Dados de entrada

- Vetor<Carrinha> C –Conjunto de
carrinhas caracterizadas por:
 - PesoMax
 - VolMax
 - Custo
 - PesoAtual
 - VolAtual



Dados de saída

- Vetor<Encomenda> P- Conjunto
de pedidos que vão ser entregues
caracterizadas por (Duração
irrelevante):
 - Peso
 - Volume
 - Recompensa



Dados de saída

- Vetor<Carrinha> Est–Conjunto de
carrinhas usadas, caracterizadas
por (PesoAtual e VolAtual
irrelevantes):
 - PesoMax
 - VolMax
 - Custo



Restrições e Domínios de valores:

- Todos os valores têm de pertencer a \mathbb{N}
- Maximizar: $\sum p. Recompensa - \sum est. Custo, p, est \in P, Est$ respetivamente
- Dentro de um estafeta:
 - $\sum (p. Peso, p. Volume) \leq (est. PesoMax, est. VolMax)$
 - $(\sum p. Recompensa - est. Custo) > 0, p, est \in P, Est$ respetivamente

Descrição da Solução

Baseado no 0/1 knapsack problem.

Preenchemos cada carrinha (ordenando-as por ordem decrescente de capacidade (peso+volume) e crescente de custo) com encomendas de forma a que soma das recompensas usadas fosse máxima.

```
procedure Preenchertruck2(Carrinha c, Vector E, Vector P){  
    int BV[E.size+1][c.Peso+1][c.Vol+1]<-{0}  
  
    for (item<-1; i<=E.size; i++) do  
        for peso<-0 to c.Peso do  
            for vol<-0 to c.Vol  
                BV[item][peso][vol]<- B[item-1][peso][vol]  
                if((peso >= E[item-1].Peso) AND (vol >= E[item-1].Vol) AND (BV[item][peso][vol] < BV[item-1][peso-E[item-1].Peso][vol-E[item-1].Vol] + [item-1].Recompensa)) then  
                    BV[item][peso][vol]<- BV[item-1][peso-E[item-1].Peso][vol-E[item-1].Vol] + E[item-1].Recompensa  
            }  
    For each c ∈ C do  
        temp <- Preenchertruck2(c,E,P)  
        if temp !=0 then  
            Est.push_back({c.Peso,c.Vol,c.Custo})
```



Complexidade

Complexidade temporal do algoritmo: $O(n^3)$

Complexidade espacial do algoritmo: $O(n^3)$

```
procedure Preenchertruck2(Carrinha c, Vector E, Vector P){  
    int BV[E.size+1][c.Peso+1][c.Vol+1]<-{0}  
  
    for (item<-1; i<=E.size; i++) do //O(n)  
        for peso<-0 to c.Peso do //O(n)  
            for vol<-0 to c.Vol // O(n)  
                BV[item][peso][vol]<- B[item-1][peso][vol]  
                if((peso >= E[item-1].Peso) AND (vol >= E[item-1].Vol) AND (BV[item][peso][vol] < BV[item-1][peso-E[item-1].Peso][vol-E[item-1].Vol] + [item-1].Recompensa)) then  
                    BV[item][peso][vol]<- BV[item-1][peso-E[item-1].Peso][vol-E[item-1].Vol] + E[item-1].Recompensa  
            }  
    }  
    For each c ∈ C do  
        temp <- Preenchertruck2(c,E,P)  
        if temp !=0 then  
            Est.push_back({c.Peso,c.Vol,c.Custo})
```



Avaliação Empírica

Forma 1

```
static bool compararCarrinhas(Carrinha a, Carrinha b){  
    return ((a.getVolMax()+a.getPesoMax())>=(b.getVolMax()+b.getPesoMax())) && (a.getCusto()<=b.getCusto());  
}
```

Forma 2

```
static bool compararCarrinhas(Carrinha a, Carrinha b){  
    return ((a.getVolMax()+a.getPesoMax())<=(b.getVolMax()+b.getPesoMax())) && (a.getCusto()<=b.getCusto());  
}
```

Forma 1

Nº Estafetas: 15
Nº Pedidos: 350
Lucro:226924
Eficiência=77.78%

Forma 2

Nº Estafetas: 18
Nº Pedidos: 370
Lucro:204967
Eficiência=82.22%

Forma 3

Nº Estafetas: 14
Nº Pedidos: 341
Lucro:222942
Eficiência=75.78%

Forma 4

Nº Estafetas: 20
Nº Pedidos: 383
Lucro:174536
Eficiência=85.11%

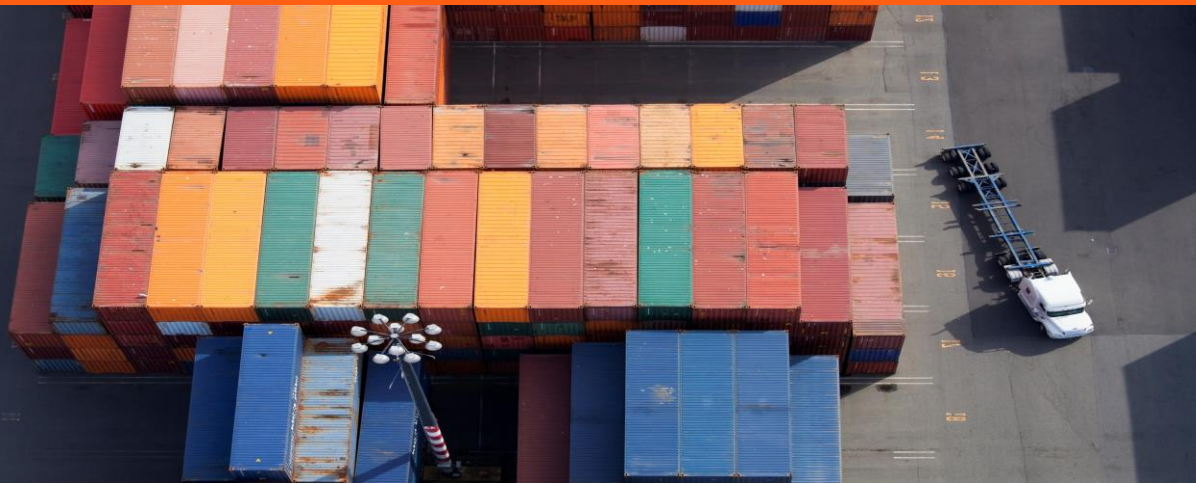


Forma 3

```
static bool compararCarrinhas(Carrinha a, Carrinha b){  
    return ((a.getVolMax()+a.getPesoMax())>=(b.getVolMax()+b.getPesoMax()));  
}
```

Forma 4

```
static bool compararCarrinhas(Carrinha a, Carrinha b){  
    return ((a.getVolMax()+a.getPesoMax())<=(b.getVolMax()+b.getPesoMax()));  
}
```





Formalização **Cenário 3**

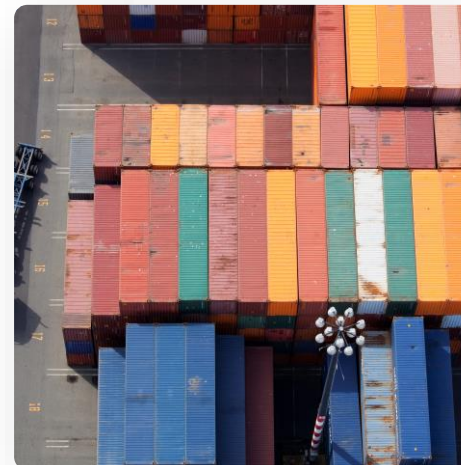
Restrições e Domínios de valores:

- Todos os valores têm de pertencer a \mathbb{N} .
- Minimizar: $\frac{\sum p.Duração}{P.size}, p \in P$



Dados de entrada

- Vetor<Encomenda> Encos-
Conjunto de encomendas
caracterizadas por:
 - Peso
 - Volume
 - Recompensa
 - Duração



Dados de saída

- Vetor<Encomenda> P- Conjunto
de pedidos que vão ser entregues
caracterizadas por (Recompensa
irrelevante):
 - Peso
 - Volume
 - Duração

Descrição da Solução

Baseando no conceito de Greedy Algorithm vamos percorrer o vetor Encos, organizado por ordem decrescente de duração, inserindo-as no vetor P até o período de tempo se esgotar.

```
int temp <- 28800 // tempo em segundos do horário 9-17
for i<-0 to Encos.size do
  if(temp-Encos[i].getDuracao())>0) then
    temp <- temp-Encos[i].getDuracao
    P.push_back(Encos[i])
  end if
```

Pseudo código do algoritmo Greedy



Complexidade

Complexidade temporal do algoritmo: $O(n)$

Complexidade espacial do algoritmo: $O(n)$

```
int temp <- 28800 // tempo em segundos do horário 9-17
for i <- 0 to Encos.size do //  $O(n)$ 
    if (temp - Encos[i].getDuracao() > 0) then
        temp <- temp - Encos[i].getDuracao
        P.push_back(Encos[i]) //  $O(1)$ 
    end if
```

Pseudo código do algoritmo usado



Avaliação Empírica

Crescente

Média: 11729.6
Nº Pedidos: 124
Eficiência=27.5556%

Decrescente

Média: 15071.9
Nº Pedidos: 27
Eficiência=6%

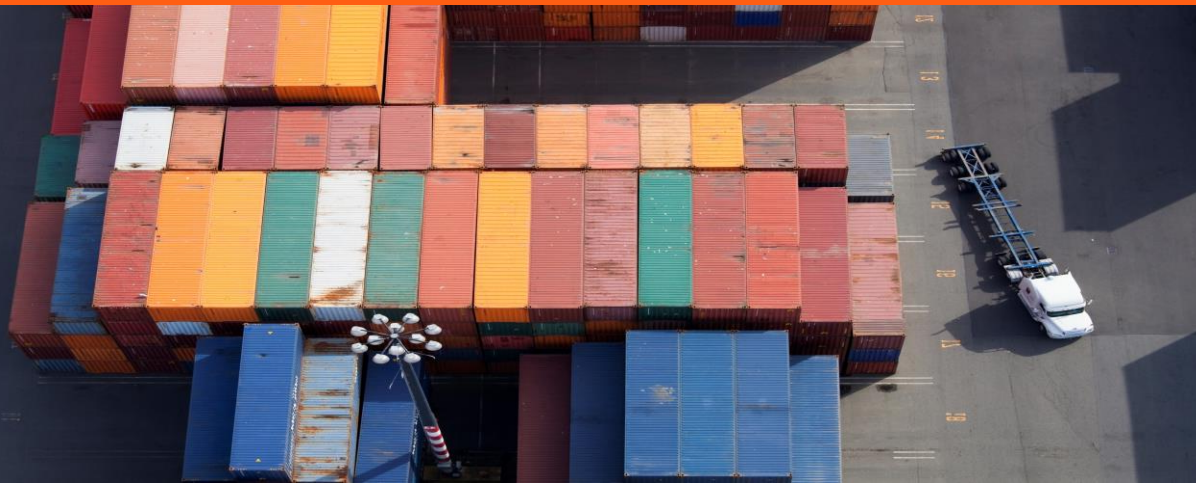
Comparar Crescente

```
bool compararEnc(Encomenda a, Encomenda b){  
    return(a.getDurar()<=b.getDurar());  
}
```



Comparar Decrescente

```
bool compararEnc(Encomenda a, Encomenda b){  
    return(a.getDurar()>=b.getDurar());  
}
```





Funcionalidade **Extra**



Em todos os cenários descobrimos a eficiência das operações da empresa.



Guardamos o número total de encomendas antes de aplicar o algoritmo numa variável e no final dividimos o tamanho do vetor onde guardamos as encomendas usadas pelo número total de encomendas.





Algoritmo a destacar



Algoritmo do cenário 2:
0/1 knapsack problem por
programação dinâmica





Dificuldades

1

Escolha do algoritmo para o cenário 2, respetivamente escolher fazer o knapsack problem de forma recursiva ou por programação dinâmica.

2

Modo de guardar as carrinhas usadas nos cenários 1 e 2 e no modo de guardar as encomendas entregues no cenário 2.

3

Modo de ordenação das encomendas e das carrinhas para os cenários 1 e 2.

4

Decisão do algoritmo usado no cenário 1.

Muito Obrigado

Logística Urbana para Entrega de Mercadorias

Auto-Avaliação

Miguel Tavares, 202002811- 33%
Sofia Sousa, 202005932 – 33%
Pedro Correia, 202006199 – 33%