Instituto Superior de Engenharia do Porto

Sprint 2 – ALGAV

Grupo 28

Tiago Marques – 1201276

Eduardo Silva – 1201371

João Vieira – 1201376

Pedro Alves – 1201381

Pedro Rocha – 1201382

dezembro, 2022

Índice de conteúdos

Representação do conhecimento do domínio

A base de conhecimento utilizada para gerar os resultados apresentados neste relatório é gerada conforme a informação presente na base de dados da aplicação. Esta encontra-se em anexo a este relatório.

Obtenção da Solução Ótima por geração de todas as soluções e escolha da melhor

O algoritmo para obtenção da solução ótima foi desenvolvido tendo por base os documentos de apoio ao Trabalho Prático de ALGAV 2022-2023 – 2 e 3.

Aproveitando o facto de que, na US anterior, é desenvolvido um predicado onde são geradas todas as soluções para um dado camião e um dado conjunto de entregas, o predicado é invocado da seguinte forma:

```
seq_tempo_min(LA, Tempo):-trajetos(Lista),
    run(Lista),
```

Como podemos ver na linha de código anterior, o predicado principal é *seq_tempo_min*, onde *LA* é a lista que contém a melhor solução a ser retornada e o *Tempo* o tempo correspondente a essa solução.

O predicado *run* é recursivo e, então, vai, para cada solução, analisar o tempo calculado através do predicado *calcula_tempo* e, caso seja a solução que, até ao momento, tenha o menor tempo, é atualizado o *tempo_min* com essa mesma solução e o tempo correspondente. Os blocos de código seguintes representam, respetivamente, o predicado *run* e a atualização da melhor solução através do *tempo_min*.

```
run([Traj|Lista]):-
    run(Lista),
    trajetoPorIdSemMat(Traj,TrajId),
    calcula_tempo(TrajId,Tempo),
    atualiza(TrajId,Tempo).
```

```
atualiza(LEPerm,Tempo):-
    tempo_min(_,TempoMin),
    ((Tempo<TempoMin,!,retract(tempo_min(_,_)),
    assertz(tempo_min(LEPerm,Tempo));true)).
```

Considerando que é o predicado *calcula_tempo* o responsável pelo cálculo do tempo de cada solução, iremos estudar mais aprofundadamente o funcionamento deste predicado.

O predicado *calcula_tempo* irá invocar, antes do cálculo do tempo, os predicados *soma_pesos* e *acrescenta_tara* já implementados e representados nos documentos de apoio especificados no início deste ponto, pelo que a sua implementação não será explicada neste

relatório. Após a invocação destes predicados, onde retornam duas listas, a primeira com a lista da soma (incremental) dos pesos das encomendas e a segunda é, a cada elemento da primeira lista, acrescentar o peso da tara do camião em questão. De seguida é chamado o predicado *tempo* que é este então que irá calcular o tempo da solução em questão.

O predicado *tempo*, após obter as informações (presentes na base de conhecimento) necessárias para o cálculo do tempo e energia de cada percurso entre dois armazéns, utiliza as fórmulas especificadas, mais uma vez, nos documentos de apoio ao trabalho prático para calcular o tempo e a energia com a carga momentânea do camião, representadas no seguinte bloco de código:

```
TempoComCarga is TempoViagem*(PesoComTara/(Tara+Capacidade)),
EnergiaDaViagem is Energia*(PesoComTara/(Tara+Capacidade)),
```

Atualizando a energia atual do camião, verifica-se se, com essa mesma energia, é possível chegar ao próximo armazém com 20% da bateria máxima do camião. Caso não consiga, terá de recarregar a sua bateria até ao valor base (80%). O tempo de recarregamento é calculado da seguinte forma:

```
(TempoCarregamento is (((0.8*MaxBateria)-Bateria)*TempoRecarga20a80)/(0.6*MaxBateria),
BateriaSeguinte is (0.8*MaxBateria)));
```

Caso o camião não carregue, o tempo adicional a considerar é o tempo de descarregar a entrega do camião.

Existe ainda uma exceção que, caso estejamos a analisar a viagem até ao armazém, o camião terá de carregar o mínimo para que chegue ao armazém base (Matosinhos) com 20% de bateria. Podemos verificar no seguinte bloco de código:

```
TempoCarregamento is ((EnergiaDaViagem2 + (0.2*MaxBateria)-Bateria)*TempoRecarga20a80)/(0.6*MaxBateria),
BateriaSeguinte is (EnergiaDaViagem2 + (0.2*MaxBateria)-Bateria));
```

Após as chamadas recursivas do predicado *tempo*, o tempo de cada percurso é acumulado ao tempo já calculado das iterações anteriores.

```
tempo([Armazem2,Armazem3|ListaArmazens], [PesoComTara2|ListaPesos], BateriaSeguinte, TempoAoMomento),
TempoTotal is TempoAoMomento+TempoComCarga+TempoAConsiderar+TempoAAdicionar.
```

Finalmente, o tempo de cada solução é retornado e avaliado com os restantes.

Estudo da complexidade do problema e da viabilidade de encontrar a solução ótima através da geração de todas as soluções

Estudo da complexidade

De maneira a analisar a complexidade do algoritmo, construímos a seguinte tabela:

Predicado	Complexidade	Justificação
seq_tempo_min	O(1) -> O(N!)	*trajetos* - O(N!) *run* - O(N!)
trajetos	O(1) -> O(N!)	*armazem_rota* - O(N) *findall* com *gerarTrajetorias* - (O(N^2)) -> O(N!)
run	O(N!) -> O(N!)	A complexidade real era O(n.n!), mas como o tempo adicional gerado pela multiplicação de n não é notória continua apenas O(N!). *trajetoPorIdSemMat* - O(N) *calcula_tempo* - O(N) *atualiza* - O(1)
calcula_tempo	O(1) -> O(N)	*soma_pesos* - O(N) *acrescenta_tara* - O(N) *tempo* - O(N)
trajetoPorIdSemMat	O(1) -> O(N)	*trajetoPorId* - O(N)
armazem_rota	O(N^2)	*findall* para cada entrega vai procurar na lista de armazéns.
gerarTrajetorias	O(1) -> O(N^2)	*permutação* - O(N^2)
permutação	O(N) -> O(N^2)	logo *apaga1* vai ser realizado N*N no pior caso -> O(N^2) *apaga1* - O(N)
apaga1	O(N)	Faz até encontrar, pior caso, ou seja, verifica todos os casos

Podemos concluir assim que a complexidade do algoritmo é O(n!).

Estudo da viabilidade

O predicado usado para calcular o tempo de execução da solução está representado no seguinte bloco de código:

```
calcular_tempo_execucao(L,T, TSol):-
    get_time(Ti),
    seq_tempo_min(L,T),
    get_time(Tf),
    TSol is Tf-Ti,
    !.
```

Com isto, foi possível estruturar a tabela abaixo apresentada.

Nº de Armazéns de entrega	Nº de soluções	Lista com a sequência de armazéns para as entregas	Tempo para fazer as entregas	Tempo de geração da solução (TSol)/s
5	120	5-8-3-1-11-9-5	412,57	0.023
6	840	5-8-3-1-17-11-9-5	453,59	0.110
7	5040	5-17-8-3-1-14-11-9-5	500,45	0.507
8	40320	5-17-8-3-12-1-14-11-9-5	532,44	3.939
9	362880	5-8-3-12-6-14-1-17-11-9-5	562,66	146.666
10	----------	----------	----------	----------

Após a breve análise à tabela anterior, verificamos que a partir de 9 armazéns por visitar, o tempo de execução é grande e impeditivo de utilizar este mesmo algoritmo.

Heurísticas para geração rápida de soluções

Foram desenvolvidas 3 heurísticas de modo a calcular soluções válidas para cada conjunto de entregas a fazer. As 3 heurísticas utilizam como fator de comparação dos percursos de um armazém para o outro:

- Menor tempo;
- Maior massa;
- Massa/Tempo.

Heurística do menor tempo

Nesta heurística, o fator que permite calcular qual o próximo armazém a visitar é o tempo, ou seja, o armazém demora menos para ser visitado do ponto atual de partida é o escolhido.

Sendo assim, o predicado *calcular_armazem_mais_proximo* analisa recursivamente o armazém que está a menor tempo do armazém atual. O predicado está representado no seguinte bloco de código:

```
calcular_armazem_mais_proximo(_,[],5000,_):-!.
calcular_armazem_mais_proximo(Origem,[H|T],R,Node):-calcular_armazem_mais_proximo(Origem, T, R1, Node1),
idArmazem(Origem,OrigemID),idArmazem(H,DestinoID),dadosCam_t_e_ta(_,OrigemID,DestinoID,_,Tempo,_,_),
    ((Tempo<R1,!, R is Tempo, Node = H);R is R1, Node = Node1).
```

Heurística da maior massa

Verificando a massa de cada entrega, o próximo armazém a ser visitado é aquele que está associado à entrega com maior massa. Assim, o camião irá entregar as entregas com maior peso para que seja mais rápido a entregar a restantes entregas.

Utilizando a mesma lógica da heurística anterior, o predicado *descobrirMaisPesado* é o responsável por avaliar as massas de cada entrega e retornar o armazém associado à entrega mais pesada.

```
descobrirMaisPesado([],0,0).
descobrirMaisPesado([X|L], MP, MPID):-descobrirMaisPesado(L,MP1, MPID1),
idArmazem(X, ID), entrega(_,_,Massa,ID,_,_),((Massa>MP1,!,MP is Massa, MPID = ID);MP is MP1, MPID = MPID1).
```

Heurística do rácio entre massa e tempo

Por último, a heurística desenvolvida para calcular o percurso com o maior rácio entre massa e tempo. Para esta heurística, fazemos o rácio entre a massa a ser descarregada e o tempo a que o armazém a ser considerado se encontra do armazém atual. Assim, o camião irá entregar primeiro as entregas ao armazém onde é descarregada maior massa em menos tempo.

```
calcular_relacao_armazem(_,[],1,0,_):-!.
calcular_relacao_armazem(Origem,[H|T],R,M,Node):-calcular_relacao_armazem(Origem,T,R1,M1,Node1),
idArmazem(Origem,OrigemID),idArmazem(H,DestinoID),dadosCam_t_e_ta(_,OrigemID,DestinoID,_,Tempo,_,_)
,entrega(_,_,Massa,DestinoID,_,_),((Massa/Tempo>M1/R1,!, R is Tempo,M is Massa, Node = H);R is R1, M is M1,
Node = Node1).
```

O predicado *calcular_relacao_armazem* analisa recursivamente o armazém que está a menor tempo do armazém atual e cuja massa a ser descarregada é a maior.

Análise da Qualidade das Heurísticas

A seguinte tabela permite a análise comparativa relativamente às 3 heurísticas enunciadas anteriormente. Assim, verificamos que, para grandes números de entregas e para o exemplo em consideração, a heurística que retorna a solução mais viável é a heurística do menor tempo.

Nº Armazéns de entrega	Solução ótima	Tempo para entregas Sol. Ótima	Heurística menor tempo	Heurística maior massa	Heurística tempo e massa combinada	Melhor solução das 3 heurísticas
5	8-3-1-11-9	412,57	513	549	484	5-11-9-8-3-1-5
6	5-8-3-1-17-11-9-5	453,59	540	592	527	5-17-11-9-8-3-1-5
7	5-17-8-3-1-14-11-9-5	500,45	561	613	580	5-17-8-3-11-9-14-1-5
8	5-17-8-3-12-1-14-11-9-5	532,44	590	632	639	5-17-8-3-11-9-12-14-1-5
9	5-8-3-12-6-14-1-17-11-9-5	562,66	594	669	647	5-17-8-3-11-9-12-6-14-1-5

Obtenção de Minorantes e Majorantes do tempo para a solução

Enquanto, para o cálculo do majorante, o próximo armazém a ser visitado é o que demora mais tempo até chegar ao próprio, considerando o peso total de todas as entregas. São considerados ainda os tempos de descarregar as entregas do camião e o tempo adicional de recarregamento das baterias do camião.

Para o minorante, apenas o peso das entregas é considerado e o próximo armazém é aquele que corresponde ao menor tempo de viagem do camião. Nenhum tempo adicional é considerado.

Sendo assim, o principal predicado para o cálculo do majorante é o *maiorTempoArmazem*, em que analisa todos os armazéns em que é possível o camião fazer a entrega e retorna aquele que demora mais tempo a lá chegar.

O tempo da viagem é calculado pela mesma forma que é calculado na US especificada no primeiro tópico deste relatório.

```
maiorTempoArmazem(_,[],_,0,_,_,_,0).
maiorTempoArmazem(ArmazemPartida,[A|ListaArmazens],ProximoArmazem,Tempo,PesoTotal,Tara,Capacidade,Energia):-
    dadosCam_t_e_ta(_,ArmazemPartida,A,_,TempoViagem,EnergiaViagem,TempoAdicional),
    TempoComCarga is TempoViagem*((Tara+PesoTotal)/(Tara+Capacidade)),
    EnergiaComCarga is EnergiaViagem*((Tara+PesoTotal)/(Tara+Capacidade)),
    TempoTotal is TempoComCarga + TempoAdicional,
    maiorTempoArmazem(ArmazemPartida,ListaArmazens,ProxA,T,PesoTotal,Tara,Capacidade,E),
    (((TempoTotal > T),Tempo is TempoTotal,Energia is EnergiaComCarga,ProximoArmazem = A);
    (Tempo is T,Energia is E,ProximoArmazem = ProxA)).
```

Por outro lado, o cálculo do minorante depende do predicado *menorTempoArmazem* que retorna o inverso do predicado apresentado anteriormente, isto é, retorna o armazém cujo tempo de chegada é o menor.

```
menorTempoArmazem(_,[],_,10000,_,_,_).
menorTempoArmazem(ArmazemPartida,[A|ListaArmazens],ProximoArmazem,Tempo,PesoTotal,Tara,Capacidade):-
    dadosCam_t_e_ta(_,ArmazemPartida,A,_,TempoViagem,_,_),
    TempoTotal is TempoViagem*((Tara+PesoTotal)/(Tara+Capacidade)),
    menorTempoArmazem(ArmazemPartida,ListaArmazens,ProxA,T,PesoTotal,Tara,Capacidade),
    (((TempoTotal < T),Tempo is TempoTotal,ProximoArmazem = A);(Tempo is T,ProximoArmazem = ProxA)).
```

Conclusões

Além das conclusões retiradas durante o relatório, podemos concluir que o algoritmo Obtenção da Solução Ótima por geração de todas as soluções e escolha da melhor não é viável a partir dos 9 armazéns, visto que como a complexidade é O(n!), a quantidade de soluções avaliadas é extremamente elevada e, previsivelmente, o tempo de execução é altíssimo.

A partir dos 9 armazéns, consideramos que a melhor heurística é a que relaciona massa e tempo, apesar de enunciarmos que, para o exemplo em questão, a heurística do menor tempo seria a mais viável, pois como esta tem atenção aos dois fatores, é mais factível em comparação às outras 2 heurísticas.

Anexos

%idArmazem(<local>,<codigo>)

idArmazem('Arouca','A01').

idArmazem('Espinho','E01').

idArmazem('Gondomar','G01').

idArmazem('Maia','M01').

idArmazem('Matosinhos','M02').

idArmazem('Oliveira de Azemeis','O01').

idArmazem('Paredes','P01').

idArmazem('Porto','P02').

idArmazem('Povoa de Varzim','P03').

idArmazem('Santa Maria da Feira','S01').

idArmazem('Santo Tirso','S02').

idArmazem('Sao Joao da Madeira','S03').

idArmazem('Trofa','T01').

idArmazem('Vale de Cambra','V01').

idArmazem('Valongo','V02').

idArmazem('Vila do Conde','V03').

idArmazem('Vila Nova de Gaia','V04').

carateristicasCam('11-10-AD',7500,4300,80,100,60).

dadosCam_t_e_ta('11-10-AD','V04','V03',33,67,25,0).

dadosCam_t_e_ta('11-10-AD','V04','V02',26,53,18,0).

dadosCam_t_e_ta('11-10-AD','V04','V01',45,90,38,0).

dadosCam_t_e_ta('11-10-AD','V04','T01',41,82,23,0).

dadosCam_t_e_ta('11-10-AD','V04','S03',40,80,30,0).

dadosCam_t_e_ta('11-10-AD','V04','S02',34,69,29,0).

dadosCam_t_e_ta('11-10-AD','V04','S01',27,55,24,0).

dadosCam_t_e_ta('11-10-AD','V04','P03',34,69,29,0).

dadosCam_t_e_ta('11-10-AD','V04','P02',14,29,6,0).

dadosCam_t_e_ta('11-10-AD','V04','P01',37,74,30,0).

dadosCam_t_e_ta('11-10-AD','V04','O01',41,82,38,0).

dadosCam_t_e_ta('11-10-AD','V04','M02',17,34,10,0).

dadosCam_t_e_ta('11-10-AD','V04','M01',21,42,13,0).

dadosCam_t_e_ta('11-10-AD','V04','G01',20,40,11,0).

dadosCam_t_e_ta('11-10-AD','V04','E01',21,42,14,0).

dadosCam_t_e_ta('11-10-AD','V04','A01',64,128,46,0).

dadosCam_t_e_ta('11-10-AD','V03','V04',32,65,24,0).

dadosCam_t_e_ta('11-10-AD','V03','V02',34,69,29,0).

dadosCam_t_e_ta('11-10-AD','V03','V01',70,141,61,37).

dadosCam_t_e_ta('11-10-AD','V03','T01',30,61,15,0).

dadosCam_t_e_ta('11-10-AD','V03','S03',65,130,52,25).

dadosCam_t_e_ta('11-10-AD','V03','S02',28,57,28,0).

dadosCam_t_e_ta('11-10-AD','V03','S01',52,105,46,0).

dadosCam_t_e_ta('11-10-AD','V03','P03',7,15,3,0).

dadosCam_t_e_ta('11-10-AD','V03','P02',33,67,23,0).

dadosCam_t_e_ta('11-10-AD','V03','P01',46,92,42,0).

dadosCam_t_e_ta('11-10-AD','V03','O01',66,132,60,35).

dadosCam_t_e_ta('11-10-AD','V03','M02',23,46,18,0).

dadosCam_t_e_ta('11-10-AD','V03','M01',28,57,16,0).

dadosCam_t_e_ta('11-10-AD','V03','G01',42,84,31,0).

dadosCam_t_e_ta('11-10-AD','V03','E01',46,92,37,0).

dadosCam_t_e_ta('11-10-AD','V03','A01',89,179,68,45).

dadosCam_t_e_ta('11-10-AD','V02','V04',26,53,17,0).

dadosCam_t_e_ta('11-10-AD','V02','V03',34,69,29,0).

dadosCam_t_e_ta('11-10-AD','V02','V01',47,95,45,0).

dadosCam_t_e_ta('11-10-AD','V02','T01',34,69,18,0).

```
dadosCam_t_e_ta('11-10-AD','V02','S03',40,80,38,0).
dadosCam_t_e_ta('11-10-AD','V02','S02',26,53,25,0).
dadosCam_t_e_ta('11-10-AD','V02','S01',41,82,36,0).
dadosCam_t_e_ta('11-10-AD','V02','P03',35,71,35,0).
dadosCam_t_e_ta('11-10-AD','V02','P02',21,42,13,0).
dadosCam_t_e_ta('11-10-AD','V02','P01',17,34,16,0).
dadosCam_t_e_ta('11-10-AD','V02','O01',43,86,44,0).
dadosCam_t_e_ta('11-10-AD','V02','M02',18,36,14,0).
dadosCam_t_e_ta('11-10-AD','V02','M01',18,36,12,0).
dadosCam_t_e_ta('11-10-AD','V02','G01',17,34,8,0).
dadosCam_t_e_ta('11-10-AD','V02','E01',37,74,30,0).
dadosCam_t_e_ta('11-10-AD','V02','A01',66,132,51,24).
dadosCam_t_e_ta('11-10-AD','V01','V04',43,86,38,0).
dadosCam_t_e_ta('11-10-AD','V01','V03',69,139,61,37).
dadosCam_t_e_ta('11-10-AD','V01','V02',45,90,44,0).
dadosCam_t_e_ta('11-10-AD','V01','T01',66,132,58,35).
dadosCam_t_e_ta('11-10-AD','V01','S03',21,42,10,0).
dadosCam_t_e_ta('11-10-AD','V01','S02',61,122,57,31).
dadosCam_t_e_ta('11-10-AD','V01','S01',30,61,17,0).
dadosCam_t_e_ta('11-10-AD','V01','P03',71,143,66,45).
dadosCam_t_e_ta('11-10-AD','V01','P02',44,99,38,0).
dadosCam_t_e_ta('11-10-AD','V01','P01',48,97,48,0).
dadosCam_t_e_ta('11-10-AD','V01','O01',13,27,9,0).
dadosCam_t_e_ta('11-10-AD','V01','M02',52,105,45,0).
dadosCam_t_e_ta('11-10-AD','V01','M01',54,109,46,0).
dadosCam_t_e_ta('11-10-AD','V01','G01',40,80,38,0).
dadosCam_t_e_ta('11-10-AD','V01','E01',40,80,35,0).
dadosCam_t_e_ta('11-10-AD','V01','A01',29,59,18,0).
dadosCam_t_e_ta('11-10-AD','T01','V04',39,78,22,0).
dadosCam_t_e_ta('11-10-AD','T01','V03',29,59,15,0).
dadosCam_t_e_ta('11-10-AD','T01','V02',33,67,17,0).
```

dadosCam_t_e_ta('11-10-AD','T01','V01',68,77,58,33).

dadosCam_t_e_ta('11-10-AD','T01','S03',61,62,52,25).

dadosCam_t_e_ta('11-10-AD','T01','S02',13,27,7,0).

dadosCam_t_e_ta('11-10-AD','T01','S01',52,105,47,0).

dadosCam_t_e_ta('11-10-AD','T01','P03',33,67,20,0).

dadosCam_t_e_ta('11-10-AD','T01','P02',38,76,20,0).

dadosCam_t_e_ta('11-10-AD','T01','P01',40,80,30,0).

dadosCam_t_e_ta('11-10-AD','T01','O01',64,128,57,31).

dadosCam_t_e_ta('11-10-AD','T01','M02',33,67,20,0).

dadosCam_t_e_ta('11-10-AD','T01','M01',23,46,11,0).

dadosCam_t_e_ta('11-10-AD','T01','G01',37,74,29,0).

dadosCam_t_e_ta('11-10-AD','T01','E01',53,107,35,0).

dadosCam_t_e_ta('11-10-AD','T01','A01',77,174,65,42).

dadosCam_t_e_ta('11-10-AD','S03','V04',37,74,32,0).

dadosCam_t_e_ta('11-10-AD','S03','V03',63,126,54,28).

dadosCam_t_e_ta('11-10-AD','S03','V02',39,78,38,0).

dadosCam_t_e_ta('11-10-AD','S03','V01',20,40,10,0).

dadosCam_t_e_ta('11-10-AD','S03','T01',60,120,53,26).

dadosCam_t_e_ta('11-10-AD','S03','S02',54,109,50,23).

dadosCam_t_e_ta('11-10-AD','S03','S01',16,32,6,0).

dadosCam_t_e_ta('11-10-AD','S03','P03',64,128,61,37).

dadosCam_t_e_ta('11-10-AD','S03','P02',43,86,33,0).

dadosCam_t_e_ta('11-10-AD','S03','P01',41,82,42,0).

dadosCam_t_e_ta('11-10-AD','S03','O01',9,19,8,0).

dadosCam_t_e_ta('11-10-AD','S03','M02',46,92,38,0).

dadosCam_t_e_ta('11-10-AD','S03','M01',47,97,41,0).

dadosCam_t_e_ta('11-10-AD','S03','G01',33,67,32,0).

dadosCam_t_e_ta('11-10-AD','S03','E01',30,61,19,0).

dadosCam_t_e_ta('11-10-AD','S03','A01',38,76,23,0).

dadosCam_t_e_ta('11-10-AD','S02','V04',33,67,27,0).

dadosCam_t_e_ta('11-10-AD','S02','V03',29,59,27,0).

dadosCam_t_e_ta('11-10-AD','S02','V02',26,53,25,0).

dadosCam_t_e_ta('11-10-AD','S02','V01',63,126,58,33).

dadosCam_t_e_ta('11-10-AD','S02','T01',12,25,7,0).

dadosCam_t_e_ta('11-10-AD','S02','S03',55,111,52,25).

dadosCam_t_e_ta('11-10-AD','S02','S01',48,97,46,0).

dadosCam_t_e_ta('11-10-AD','S02','P03',31,63,28,0).

dadosCam_t_e_ta('11-10-AD','S02','P02',29,59,23,0).

dadosCam_t_e_ta('11-10-AD','S02','P01',37,74,23,0).

dadosCam_t_e_ta('11-10-AD','S02','O01',59,118,57,31).

dadosCam_t_e_ta('11-10-AD','S02','M02',27,55,25,0).

dadosCam_t_e_ta('11-10-AD','S02','M01',21,42,18,0).

dadosCam_t_e_ta('11-10-AD','S02','G01',32,65,28,0).

dadosCam_t_e_ta('11-10-AD','S02','E01',44,88,41,0).

dadosCam_t_e_ta('11-10-AD','S02','A01',82,164,65,42).

dadosCam_t_e_ta('11-10-AD','S01','V04',26,53,23,0).

dadosCam_t_e_ta('11-10-AD','S01','V03',52,105,46,0).

dadosCam_t_e_ta('11-10-AD','S01','V02',37,74,34,0).

dadosCam_t_e_ta('11-10-AD','S01','V01',31,63,17,0).

dadosCam_t_e_ta('11-10-AD','S01','T01',49,99,46,0).

dadosCam_t_e_ta('11-10-AD','S01','S03',16,32,6,0).

dadosCam_t_e_ta('11-10-AD','S01','S02',46,92,52,0).

dadosCam_t_e_ta('11-10-AD','S01','P03',54,109,52,25).

dadosCam_t_e_ta('11-10-AD','S01','P02',32,65,24,0).

dadosCam_t_e_ta('11-10-AD','S01','P01',41,82,42,0).

dadosCam_t_e_ta('11-10-AD','S01','O01',20,40,14,0).

dadosCam_t_e_ta('11-10-AD','S01','M02',35,71,30,0).

dadosCam_t_e_ta('11-10-AD','S01','M01',39,78,33,0).

dadosCam_t_e_ta('11-10-AD','S01','G01',29,59,27,0).

dadosCam_t_e_ta('11-10-AD','S01','E01',17,34,14,0).

dadosCam_t_e_ta('11-10-AD','S01','A01',48,97,30,0).

dadosCam_t_e_ta('11-10-AD','P03','V04',33,67,28,0).

dadosCam_t_e_ta('11-10-AD','P03','V03',7,15,3,0).

dadosCam_t_e_ta('11-10-AD','P03','V02',35,71,34,0).

dadosCam_t_e_ta('11-10-AD','P03','V01',71,143,66,45).

dadosCam_t_e_ta('11-10-AD','P03','T01',33,67,19,0).

dadosCam_t_e_ta('11-10-AD','P03','S03',66,132,57,31).

dadosCam_t_e_ta('11-10-AD','P03','S02',30,61,29,0).

dadosCam_t_e_ta('11-10-AD','P03','S01',54,109,51,24).

dadosCam_t_e_ta('11-10-AD','P03','P02',34,69,28,0).

dadosCam_t_e_ta('11-10-AD','P03','P01',47,95,65,0).

dadosCam_t_e_ta('11-10-AD','P03','O01',67,134,65,42).

dadosCam_t_e_ta('11-10-AD','P03','M02',24,48,23,0).

dadosCam_t_e_ta('11-10-AD','P03','M01',27,55,24,0).

dadosCam_t_e_ta('11-10-AD','P03','G01',43,86,35,0).

dadosCam_t_e_ta('11-10-AD','P03','E01',47,95,41,0).

dadosCam_t_e_ta('11-10-AD','P03','A01',90,181,72,50).

dadosCam_t_e_ta('11-10-AD','P02','V04',16,32,6,0).

dadosCam_t_e_ta('11-10-AD','P02','V03',32,65,23,0).

dadosCam_t_e_ta('11-10-AD','P02','V02',18,36,12,0).

dadosCam_t_e_ta('11-10-AD','P02','V01',30,97,40,0).

dadosCam_t_e_ta('11-10-AD','P02','T01',30,61,24,0).

dadosCam_t_e_ta('11-10-AD','P02','S03',41,82,34,0).

dadosCam_t_e_ta('11-10-AD','P02','S02',26,53,22,0).

dadosCam_t_e_ta('11-10-AD','P02','S01',32,65,26,0).

dadosCam_t_e_ta('11-10-AD','P02','P03',34,69,30,0).

dadosCam_t_e_ta('11-10-AD','P02','P01',28,57,26,0).

dadosCam_t_e_ta('11-10-AD','P02','O01',44,88,38,0).

dadosCam_t_e_ta('11-10-AD','P02','M02',16,32,7,0).

dadosCam_t_e_ta('11-10-AD','P02','M01',17,34,10,0).

dadosCam_t_e_ta('11-10-AD','P02','G01',16,32,6,0).

dadosCam_t_e_ta('11-10-AD','P02','E01',29,59,18,0).

dadosCam_t_e_ta('11-10-AD','P02','A01',67,134,46,0).

dadosCam_t_e_ta('11-10-AD','P01','V04',34,69,30,0).

dadosCam_t_e_ta('11-10-AD','P01','V03',42,84,42,0).

dadosCam_t_e_ta('11-10-AD','P01','V02',14,29,16,0).

dadosCam_t_e_ta('11-10-AD','P01','V01',38,97,49,21).

dadosCam_t_e_ta('11-10-AD','P01','T01',38,76,31,0).

dadosCam_t_e_ta('11-10-AD','P01','S03',41,82,42,0).

dadosCam_t_e_ta('11-10-AD','P01','S02',37,74,22,0).

dadosCam_t_e_ta('11-10-AD','P01','S01',42,84,44,0).

dadosCam_t_e_ta('11-10-AD','P01','P03',44,88,48,0).

dadosCam_t_e_ta('11-10-AD','P01','P02',29,59,26,0).

dadosCam_t_e_ta('11-10-AD','P01','O01',44,88,48,0).

dadosCam_t_e_ta('11-10-AD','P01','M02',26,53,28,0).

dadosCam_t_e_ta('11-10-AD','P01','M01',26,53,26,0).

dadosCam_t_e_ta('11-10-AD','P01','G01',30,61,22,0).

dadosCam_t_e_ta('11-10-AD','P01','E01',35,71,38,0).

dadosCam_t_e_ta('11-10-AD','P01','A01',58,116,36,0).

dadosCam_t_e_ta('11-10-AD','O01','V04',40,80,38,0).

dadosCam_t_e_ta('11-10-AD','O01','V03',66,132,60,35).

dadosCam_t_e_ta('11-10-AD','O01','V02',42,84,44,0).

dadosCam_t_e_ta('11-10-AD','O01','V01',12,25,9,0).

dadosCam_t_e_ta('11-10-AD','O01','T01',63,126,58,33).

dadosCam_t_e_ta('11-10-AD','O01','S03',11,23,9,0).

dadosCam_t_e_ta('11-10-AD','O01','S02',21,116,56,30).

dadosCam_t_e_ta('11-10-AD','O01','S01',21,42,14,0).

dadosCam_t_e_ta('11-10-AD','O01','P03',67,134,66,45).

dadosCam_t_e_ta('11-10-AD','O01','P02',46,92,38,0).

dadosCam_t_e_ta('11-10-AD','O01','P01',44,88,48,0).

dadosCam_t_e_ta('11-10-AD','O01','M02',49,99,44,0).

dadosCam_t_e_ta('11-10-AD','O01','M01',51,103,46,0).

dadosCam_t_e_ta('11-10-AD','O01','G01',37,74,38,0).

dadosCam_t_e_ta('11-10-AD','O01','E01',35,71,27,0).

dadosCam_t_e_ta('11-10-AD','O01','A01',34,69,23,0).

dadosCam_t_e_ta('11-10-AD','M02','V04',13,27,7,0).

dadosCam_t_e_ta('11-10-AD','M02','V03',23,46,18,0).

dadosCam_t_e_ta('11-10-AD','M02','V02',17,34,14,0).

dadosCam_t_e_ta('11-10-AD','M02','V01',52,105,45,0).

dadosCam_t_e_ta('11-10-AD','M02','T01',26,63,20,0).

dadosCam_t_e_ta('11-10-AD','M02','S03',47,95,36,0).

dadosCam_t_e_ta('11-10-AD','M02','S02',26,53,26,0).

dadosCam_t_e_ta('11-10-AD','M02','S01',34,69,30,0).

dadosCam_t_e_ta('11-10-AD','M02','P03',24,48,24,0).

dadosCam_t_e_ta('11-10-AD','M02','P02',14,29,7,0).

dadosCam_t_e_ta('11-10-AD','M02','P01',22,55,28,0).

dadosCam_t_e_ta('11-10-AD','M02','O01',48,97,44,0).

dadosCam_t_e_ta('11-10-AD','M02','M01',12,25,9,0).

dadosCam_t_e_ta('11-10-AD','M02','G01',24,48,14,0).

dadosCam_t_e_ta('11-10-AD','M02','E01',27,55,20,0).

dadosCam_t_e_ta('11-10-AD','M02','A01',70,141,51,24).

dadosCam_t_e_ta('11-10-AD','M01','V04',19,38,11,0).

dadosCam_t_e_ta('11-10-AD','M01','V03',26,53,14,0).

dadosCam_t_e_ta('11-10-AD','M01','V02',16,32,13,0).

dadosCam_t_e_ta('11-10-AD','M01','V01',55,111,48,0).

dadosCam_t_e_ta('11-10-AD','M01','T01',22,44,11,0).

dadosCam_t_e_ta('11-10-AD','M01','S03',48,97,42,0).

dadosCam_t_e_ta('11-10-AD','M01','S02',21,42,19,0).

dadosCam_t_e_ta('11-10-AD','M01','S01',39,78,34,0).

dadosCam_t_e_ta('11-10-AD','M01','P03',25,50,26,0).

dadosCam_t_e_ta('11-10-AD','M01','P02',18,36,10,0).

dadosCam_t_e_ta('11-10-AD','M01','P01',22,55,27,0).

dadosCam_t_e_ta('11-10-AD','M01','O01',51,103,47,0).

dadosCam_t_e_ta('11-10-AD','M01','M02',13,27,10,0).

dadosCam_t_e_ta('11-10-AD','M01','G01',23,46,16,0).

dadosCam_t_e_ta('11-10-AD','M01','E01',32,65,24,0).

dadosCam_t_e_ta('11-10-AD','M01','A01',74,149,54,25).

dadosCam_t_e_ta('11-10-AD','G01','V04',18,36,10,0).

dadosCam_t_e_ta('11-10-AD','G01','V03',40,80,30,0).

dadosCam_t_e_ta('11-10-AD','G01','V02',17,34,8,0).

dadosCam_t_e_ta('11-10-AD','G01','V01',33,82,38,0).

dadosCam_t_e_ta('11-10-AD','G01','T01',33,67,29,0).

dadosCam_t_e_ta('11-10-AD','G01','S03',33,67,32,0).

dadosCam_t_e_ta('11-10-AD','G01','S02',30,61,27,0).

dadosCam_t_e_ta('11-10-AD','G01','S01',29,59,28,0).

dadosCam_t_e_ta('11-10-AD','G01','P03',44,84,36,0).

dadosCam_t_e_ta('11-10-AD','G01','P02',19,38,8,0).

dadosCam_t_e_ta('11-10-AD','G01','P01',31,63,23,0).

dadosCam_t_e_ta('11-10-AD','G01','O01',37,74,37,0).

dadosCam_t_e_ta('11-10-AD','G01','M02',23,46,14,0).

dadosCam_t_e_ta('11-10-AD','G01','M01',23,46,15,0).

dadosCam_t_e_ta('11-10-AD','G01','E01',25,50,22,0).

dadosCam_t_e_ta('11-10-AD','G01','A01',60,120,45,0).

dadosCam_t_e_ta('11-10-AD','E01','V04',23,46,14,0).

dadosCam_t_e_ta('11-10-AD','E01','V03',34,99,38,0).

dadosCam_t_e_ta('11-10-AD','E01','V02',34,69,30,0).

dadosCam_t_e_ta('11-10-AD','E01','V01',39,78,34,0).

dadosCam_t_e_ta('11-10-AD','E01','T01',42,95,42,0).

dadosCam_t_e_ta('11-10-AD','E01','S03',30,121,19,0).

dadosCam_t_e_ta('11-10-AD','E01','S02',44,88,41,0).

dadosCam_t_e_ta('11-10-AD','E01','S01',18,36,14,0).

dadosCam_t_e_ta('11-10-AD','E01','P03',51,103,44,0).

dadosCam_t_e_ta('11-10-AD','E01','P02',30,61,18,0).

dadosCam_t_e_ta('11-10-AD','E01','P01',37,74,38,0).

dadosCam_t_e_ta('11-10-AD','E01','O01',34,69,27,0).

dadosCam_t_e_ta('11-10-AD','E01','M02',32,65,22,0).

dadosCam_t_e_ta('11-10-AD','E01','M01',37,74,25,0).

dadosCam_t_e_ta('11-10-AD','E01','G01',27,55,22,0).

dadosCam_t_e_ta('11-10-AD','E01','A01',58,116,42,0).

dadosCam_t_e_ta('11-10-AD','A01','V04',64,128,45,0).

dadosCam_t_e_ta('11-10-AD','A01','V03',90,181,68,45).

dadosCam_t_e_ta('11-10-AD','A01','V02',66,132,51,24).

dadosCam_t_e_ta('11-10-AD','A01','V01',29,59,18,0).

dadosCam_t_e_ta('11-10-AD','A01','T01',87,174,66,45).

dadosCam_t_e_ta('11-10-AD','A01','S03',38,76,23,0).

dadosCam_t_e_ta('11-10-AD','A01','S02',32,164,64,40).

dadosCam_t_e_ta('11-10-AD','A01','S01',48,97,30,0).

dadosCam_t_e_ta('11-10-AD','A01','P03',92,185,74,53).

dadosCam_t_e_ta('11-10-AD','A01','P02',70,141,46,0).

dadosCam_t_e_ta('11-10-AD','A01','P01',58,116,35,0).

dadosCam_t_e_ta('11-10-AD','A01','O01',37,74,24,0).

dadosCam_t_e_ta('11-10-AD','A01','M02',73,147,52,25).

dadosCam_t_e_ta('11-10-AD','A01','M01',75,151,54,25).

dadosCam_t_e_ta('11-10-AD','A01','G01',61,122,46,0).

dadosCam_t_e_ta('11-10-AD','A01','E01',61,122,42,0).

%entrega(<idEntrega>,<data>,<massaEntrefa>,<armazemEntrega>,<tempoColoc>,<tempoRet>
)

entrega(4439, 20221205, 200, 'A01', 8, 10).

entrega(4438, 20221205, 150, 'P03', 7, 9).

entrega(4445, 20221205, 100, 'G01', 5, 7).

entrega(4443, 20221205, 120, 'P02', 6, 8).

entrega(4449, 20221205, 300, 'S02', 15, 20).

entrega(4398, 20221205, 310, 'V04', 16, 20).

entrega(4432, 20221205, 270, 'V01', 14, 18).

entrega(4437, 20221205, 180, 'S03', 9, 11).

entrega(4451, 20221205, 220, 'O01', 9, 12).