# Protótipo de Software para Logística de Distribuição

Acadêmica: Viviane Bittencourt Rosa.

Orientador: Jomi Fred Hübner.

# Roteiro da apresentação

- Introdução
- Conceitos
  - Logística de distribuição
  - *Constraint Satisfaction Problem*
  - *Constraint Satisfaction Optimazation Problems*
- Trabalhos correlatos
- Implementação
  - Requisitos principais
  - Diagrama de atividades
  - Modelagem do banco de dados
- Técnicas e ferramentas utilizadas
  - Geração do programa para o interpretador *GNU Prolog*
- Resultados e discussões
- Conclusão
- Relevância pessoal

# Introdução

- Logística de distribuição
  - Definição da melhor rota
- *Constraint Satisfaction Problem*
- Objetivos
  - Objetivo principal
    - Construção do protótipo de software para logística de distribuição por vias rodoviárias
  - Objetivos específicos
    - Sugerir quais produtos serão transportados por quais veículos
    - Definição da melhor rota

# Conceitos

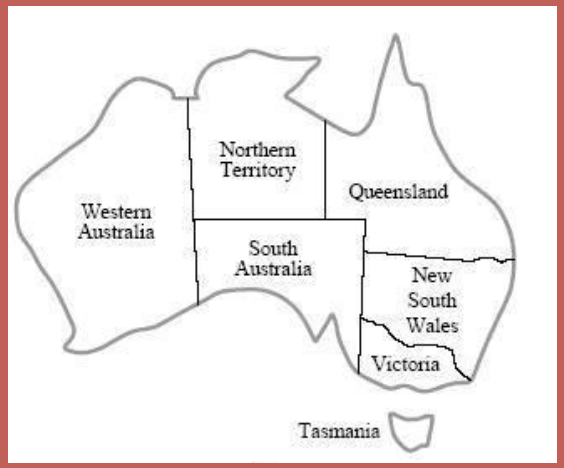
- Logística de distribuição
  - Importância
  - Melhor utilização dos veículos
    - Rota ou Plano de viagem
    - Roteirização e programação dos veículos
    - Despacho dos veículos
    - Seqüenciação de roteiros
    - Balanceamento das viagens

# Conceitos

- Constraint Satisfaction Problem (CSP)
  - Definição
    - Variáveis
    - Domínios
    - Restrições

# Conceitos

- Problema da pintura do mapa da Austrália
  - Definição do problema
  - Variáveis: cidades
  - Domínio: cores (vermelho, verde e azul)
  - Restrições: cidades vizinhas devem possuir cores diferentes.




# Conceitos

- Constraint Satisfaction Optimization Problems (CSOP)
  - Definição
    - Variáveis
    - Domínios
    - Restrições
    - **Otimização**

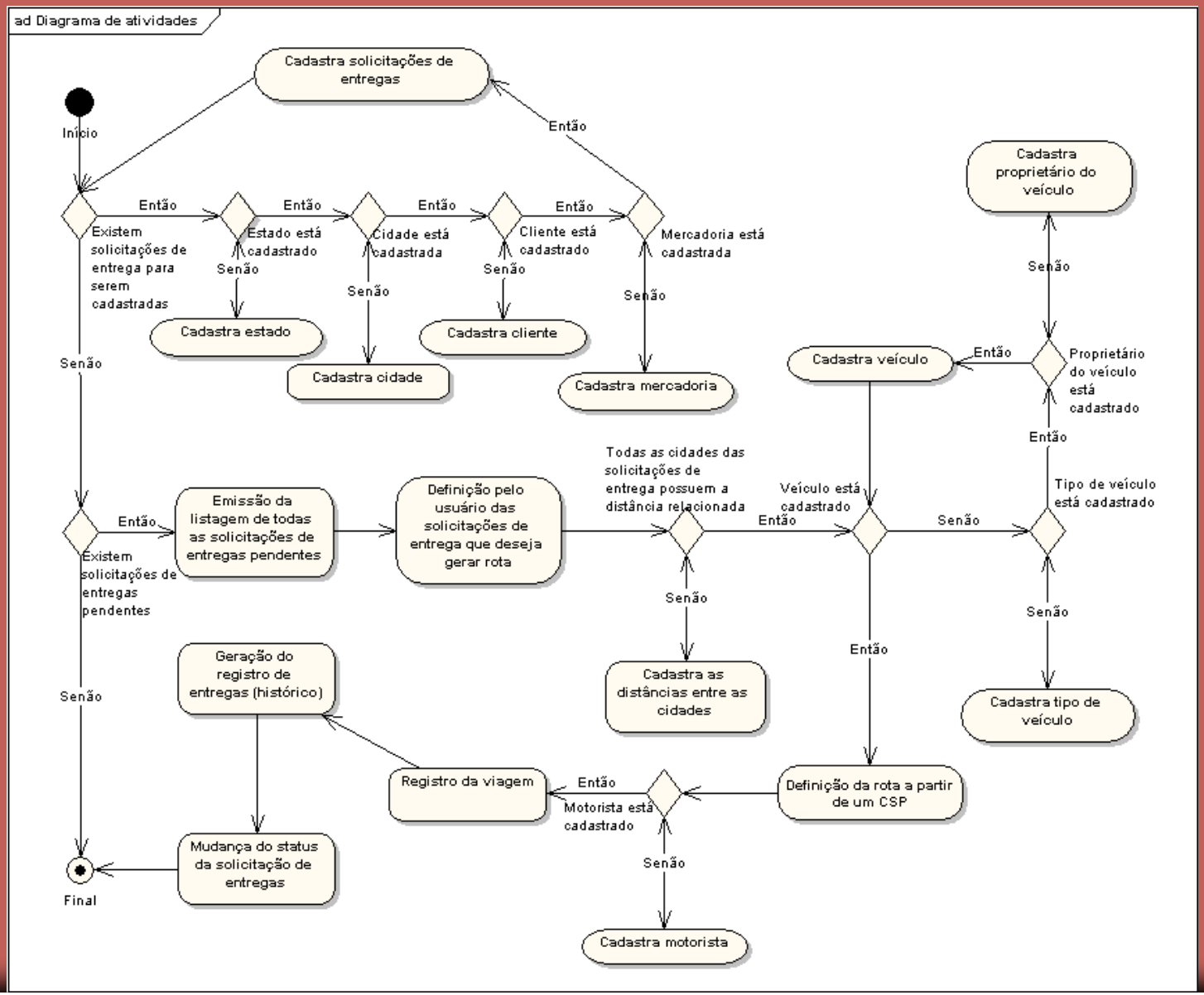
# Trabalhos correlatos

- Programação por propagação de restrições (Sucupira, 2004)
- Sistemas multi-agentes no desenvolvimento de um protótipo de um sistema de logística (Gonçalves, 1997)
- A logística na distribuição (Knaesel Neto, 1999)

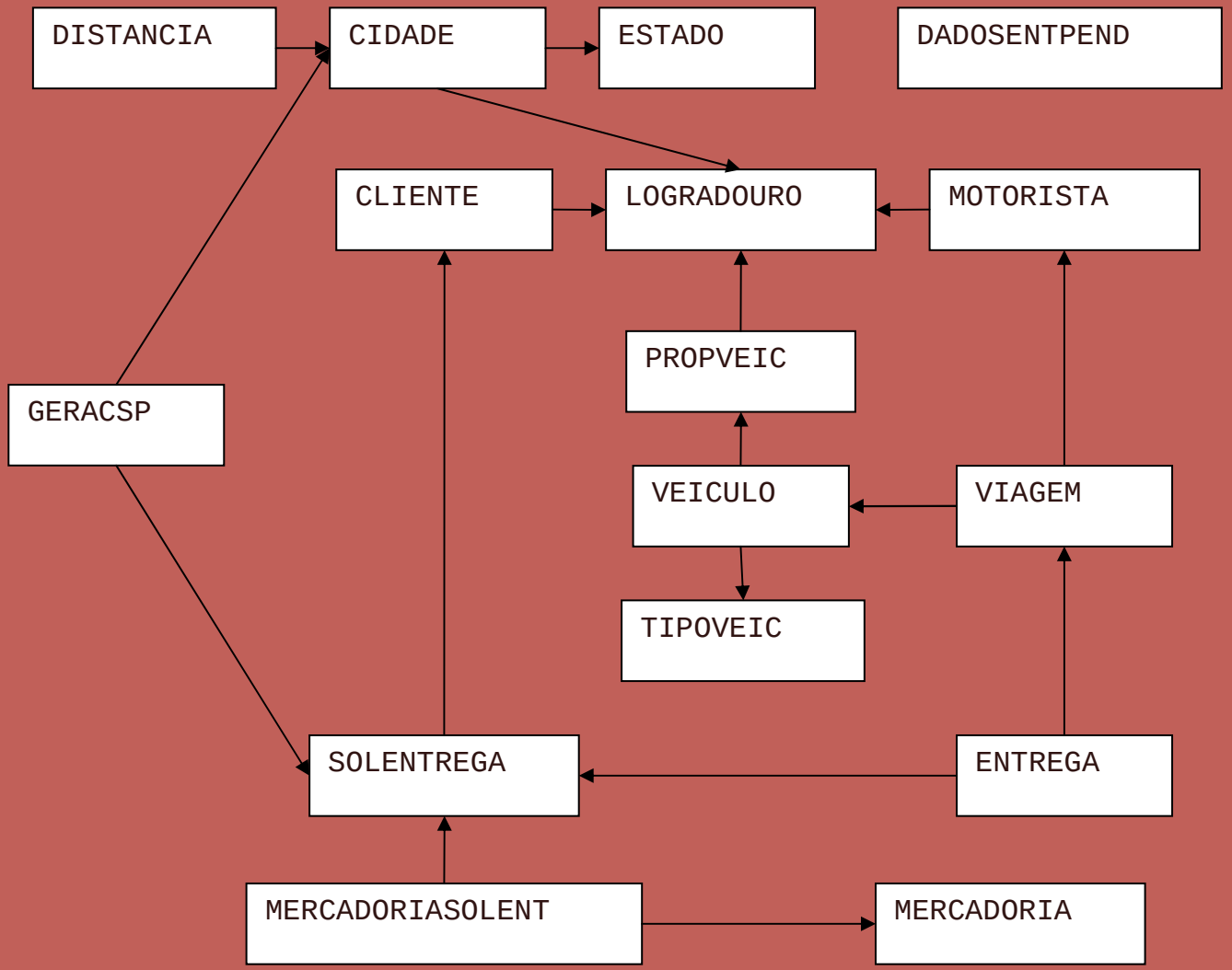
# Requisitos principais

- Funcionais:
  - Cadastros:
    - **Distâncias**
    - **Clientes**
    - **Veículos**
    - **Motoristas**
    - **Mercadorias**
  - Lançamentos:
    - **Solicitações de entregas**
    - **Entregas**
    - **Viagens**
  - Decisões
    - **Geração de rotas**
- Não funcionais:
  - Utilização da ferramenta de desenvolvimento *Delphi*, com a linguagem *Object Pascal*
  - Utilização do banco de dados *Firebird*
  - Utilização de uma ferramenta que resolve um CSP

# Diagrama de atividades



# Modelagem do banco de dados



# Técnicas e ferramentas utilizadas

- Escolha da ferramenta que resolve CSP
- Geração do programa para o interpretador *GNU Prolog*

# Geração do programa para o interpretador *GNU Prolog*

| SEQUENCIA | IDSOLENTREGA | IDCIDADE | PESO | VOLUME | TIPO |
|---|---|---|---|---|---|
| 1 | 0 | 626 | 0 | 0 | 0 |
| 2 | 483 | 465 | 150.000 | 576.000 | 0 |
| 3 | 483 | 626 | 150.000 | 576.000 | 1 |
| 4 | 482 | 465 | 100.000 | 384.000 | 0 |
| 5 | 482 | 1792 | 100.000 | 384.000 | 1 |
| 6 | 480 | 626 | 50.000 | 192.000 | 0 |
| 7 | 480 | 1792 | 50.000 | 192.000 | 1 |
| 8 | 0 | 626 | 0 | 0 | 1 |

# Geração do programa para o interpretador *GNU Prolog*

## Base de conhecimento:

```
% DISTÂNCIAS

% A distância de qualquer cidade para ela mesma é 0
dist(X,X,0).

% Todas as distâncias cadastradas no sistema necessárias
    para a solução
% Notação dist(cidadeorigem, cidadedestino, distância).
dist(465,626,56).
dist(465,1792,94).
Dist(626, 1792, 135).

% Regras para as definições das combinações de distâncias
distancia(X,Y,D) :- dist(X,Y,D). % relaciona dist com
    distância
distancia(X,Y,D) :- dist(Y,X,D). % toda as distâncias também
    servem para o inverso

% Cidades referentes às entregas
% A entrega X deverá ser realizada na cidade Y
entregaCidade(1,626).
entregaCidade(2,465).
entregaCidade(3,626).
entregaCidade(4,465).
entregaCidade(5,1792).
entregaCidade(6,626).
entregaCidade(7,1792).
entregaCidade(8,626).
```

```
% Pesos das entregas
peso(1,0).
peso(2,150000).
peso(3,-150000).
peso(4,100000).
peso(5,-100000).
peso(6,50000).
peso(7,-50000).
peso(8,-0).

% Volumes das entregas
volume(1,0).
volume(2,576000).
volume(3,-576000).
volume(4,384000).
volume(5,-384000).
volume(6,192000).
volume(7,-192000).
volume(8,-0).
```

# Geração do programa para o interpretador *GNU Prolog*

## CSP:

```
resolve :-
% VARIÁVEIS
Todas = [E1, E2, E3, E4, E5, E6, E7, E8],

% DOMÍNIO
fd_domain(Todas,1,8),

% RESTRIÇÕES
% todas as entregas devem ser diferentes
fd_all_different(Todas),
% Só entregar após coletar
E2 #< E3,
E4 #< E5,
E6 #< E7,

%Começar na entrega gerada para Blumenau
E1 #= 1,

%Terminar na entrega gerada para Blumenau
E8 #=8,

% RESOLVE e OTIMIZA o custo do CSP
fd_minimize(
  (fd_labeling(Todas), controlaPeso(Todas),
 controlaVolume(Todas), calculaCusto(Todas,CT) ),
  CT),
```

```
% IMPRIME RESPOSTA

% Seta a saída para um arquivo
tell('resultado.txt'),
% Inicia a inscrição do arquivo
append('resultado.txt'),
 write('Entrega1 = '), write(E1), nl,
 write('Entrega2 = '), write(E2), nl,
 write('Entrega3 = '), write(E3), nl,
 write('Entrega4 = '), write(E4), nl,
 write('Entrega5 = '), write(E5), nl,
 write('Entrega6 = '), write(E6), nl,
 write('Entrega7 = '), write(E7), nl,
 write('Entrega8 = '), write(E8), nl,
 write('Custo = '), write(CT),
%finaliza a saída do arquivo
told.
```

# Geração do programa para o interpretador *GNU Prolog*

## Controla Volume:

```
% controlaVolume
controlaVolume([E1, E2, E3, E4, E5, E6, E7, E8]) :-
%Monta a lista das variáveis
 EntVol = [E1, E2, E3, E4, E5, E6, E7, E8],

% Verifica qual posição da lista de entregas está a variável
    que recebeu o valor 1
 nth(Posicao1, EntVol, 1),
% Associa o peso da primeira entrega à variável V1
 volume(Posicao1, V1),
% Efetua os mesmo procedimentos para todas as outras
    variáveis
 nth(Posicao1, EntVol, 1),
 volume(Posicao1, V1),
 nth(Posicao2, EntVol, 2),
 volume(Posicao2, V2),
 nth(Posicao3, EntVol, 3),
 volume(Posicao3, V3),
 nth(Posicao4, EntVol, 4),
 volume(Posicao4, V4),
 nth(Posicao5, EntVol, 5),
 volume(Posicao5, V5),
 nth(Posicao6, EntVol, 6),
 volume(Posicao6, V6),
 nth(Posicao7, EntVol, 7),
 volume(Posicao7, V7),
 nth(Posicao8, EntVol, 8),
 volume(Posicao8, V8),
```

```
% Faz a soma do volume em cada entrega
 VolAtual1 #= V1,
 VolAtual2 #= VolAtual1 + V2,
 VolAtual3 #= VolAtual2 + V3,
 VolAtual4 #= VolAtual3 + V4,
 VolAtual5 #= VolAtual4 + V5,
 VolAtual6 #= VolAtual5 + V6,
 VolAtual7 #= VolAtual6 + V7,
 VolAtual8 #= VolAtual7 + V8,

% Restrição
% Não permite que o volume em cada entrega
ultrapasse a capacidade do veículo
 VolAtual1 #=<# 15750000,
 VolAtual2 #=<# 15750000,
 VolAtual3 #=<# 15750000,
 VolAtual4 #=<# 15750000,
 VolAtual5 #=<# 15750000,
 VolAtual6 #=<# 15750000,
 VolAtual7 #=<# 15750000,
 VolAtual8 #=<# 15750000.
```

# Geração do programa para o interpretador *GNU Prolog*

## Controla Peso:

```
% controlaPeso
controlaPeso([E1, E2, E3, E4, E5, E6, E7, E8]) :-
%Monta a lista das variáveis
 EntPeso = [E1, E2, E3, E4, E5, E6, E7, E8],
% Verifica qual posição da lista de entregas está a variável
    que recebeu o valor 1
 nth(Posicao1, EntPeso, 1),
% Associa o peso da primeira entrega à variável P1
 peso(Posicao1, P1),
% Efetua os mesmo procedimentos para todas as outras
    variáveis
 nth(Posicao1, EntPeso, 1),
 peso(Posicao1, P1),
 nth(Posicao2, EntPeso, 2),
 peso(Posicao2, P2),
 nth(Posicao3, EntPeso, 3),
 peso(Posicao3, P3),
 nth(Posicao4, EntPeso, 4),
 peso(Posicao4, P4),
 nth(Posicao5, EntPeso, 5),
 peso(Posicao5, P5),
 nth(Posicao6, EntPeso, 6),
 peso(Posicao6, P6),
 nth(Posicao7, EntPeso, 7),
 peso(Posicao7, P7),
 nth(Posicao8, EntPeso, 8),
 peso(Posicao8, P8),
```

```
% Faz a soma do peso em cada entrega
 PesoAtual1 #= P1,
 PesoAtual2 #= PesoAtual1 + P2,
 PesoAtual3 #= PesoAtual2 + P3,
 PesoAtual4 #= PesoAtual3 + P4,
 PesoAtual5 #= PesoAtual4 + P5,
 PesoAtual6 #= PesoAtual5 + P6,
 PesoAtual7 #= PesoAtual6 + P7,
 PesoAtual8 #= PesoAtual7 + P8,

% Restrição
% Não permite que o peso em cada entrega
ultrapasse a capacidade do veículo
 PesoAtual1 #=<# 20000000,
 PesoAtual2 #=<# 20000000,
 PesoAtual3 #=<# 20000000,
 PesoAtual4 #=<# 20000000,
 PesoAtual5 #=<# 20000000,
 PesoAtual6 #=<# 20000000,
 PesoAtual7 #=<# 20000000,
 PesoAtual8 #=<# 20000000.told.
```

# Geração do programa para o interpretador *GNU Prolog*

## Calcula Custo:

```
% calculaCusto
calculaCusto([E1, E2, E3, E4, E5, E6, E7, E8], CT) :-
 %Monta a lista das variáveis
 EntCusto = [E1, E2, E3, E4, E5, E6, E7, E8],

% Verifica qual posição da lista de entregas está a variável
    que recebeu o valor 1
 nth(Posicao1, EntCusto , 1),
% Associa a cidade da primeira entrega à variável C1
 entregaCidade(Posicao1, C1),
% Efetua os mesmo procedimentos para todas as outras
    variáveis
 nth(Posicao1, EntCusto , 1),
 entregaCidade(Posicao1, C1),
 nth(Posicao2, EntCusto , 2),
 entregaCidade(Posicao2, C2),
 nth(Posicao3, EntCusto , 3),
 entregaCidade(Posicao3, C3),
 nth(Posicao4, EntCusto , 4),
 entregaCidade(Posicao4, C4),
 nth(Posicao5, EntCusto , 5),
 entregaCidade(Posicao5, C5),
 nth(Posicao6, EntCusto , 6),
 entregaCidade(Posicao6, C6),
 nth(Posicao7, EntCusto , 7),
 entregaCidade(Posicao7, C7),
 nth(Posicao8, EntCusto , 8),
 entregaCidade(Posicao8, C8),
```

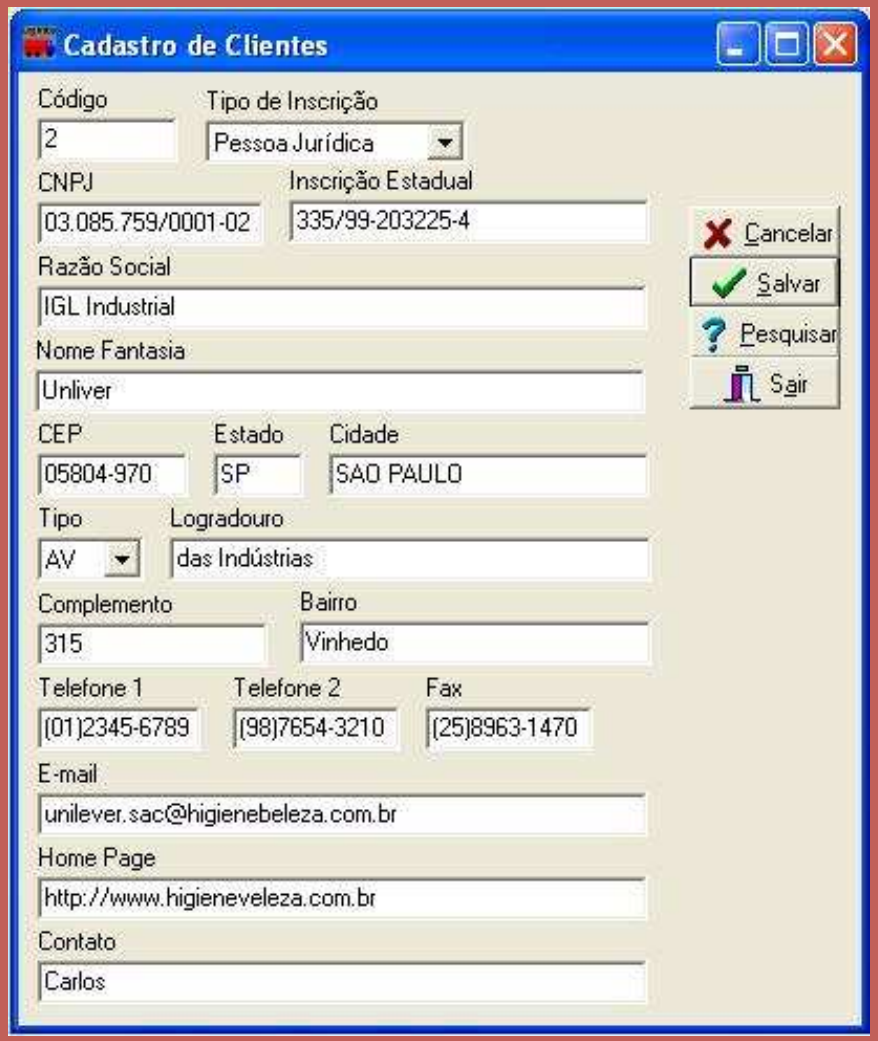
```
% Define as distâncias para calcular o custo
 distancia(C1, C2, D1),
 distancia(C2, C3, D2),
 distancia(C3, C4, D3),
 distancia(C4, C5, D4),
 distancia(C5, C6, D5),
 distancia(C6, C7, D6),
 distancia(C7, C8, D7),

 % Calcula o custo total das distâncias entre as
cidades
 CT #= D1 + D2 + D3 + D4 + D5 + D6 + D7.
```

# Geração do programa para o interpretador *GNU Prolog*

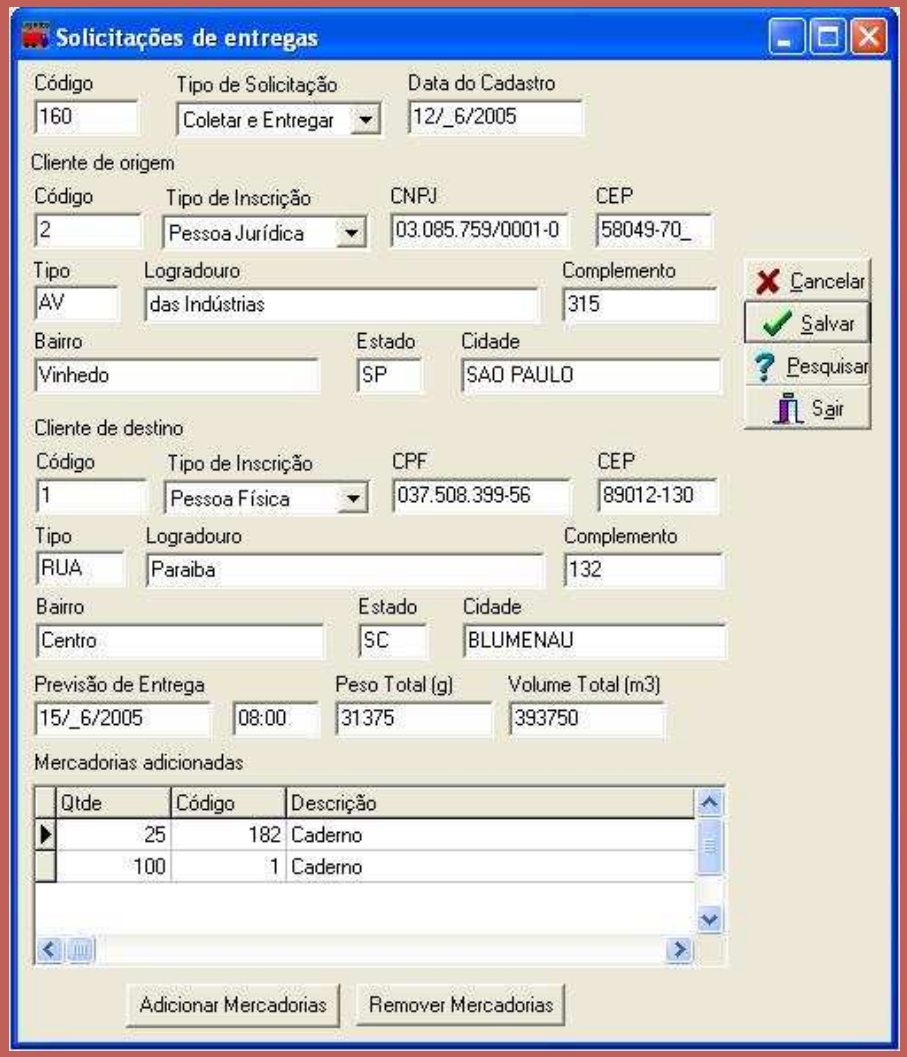
Resultado:

Entrega1 = 1
Entrega2 = 6
Entrega3 = 7
Entrega4 = 3
Entrega5 = 4
Entrega6 = 2
Entrega7 = 5
Entrega8 = 8
Custo = 300

# Operacionalidade da implementação

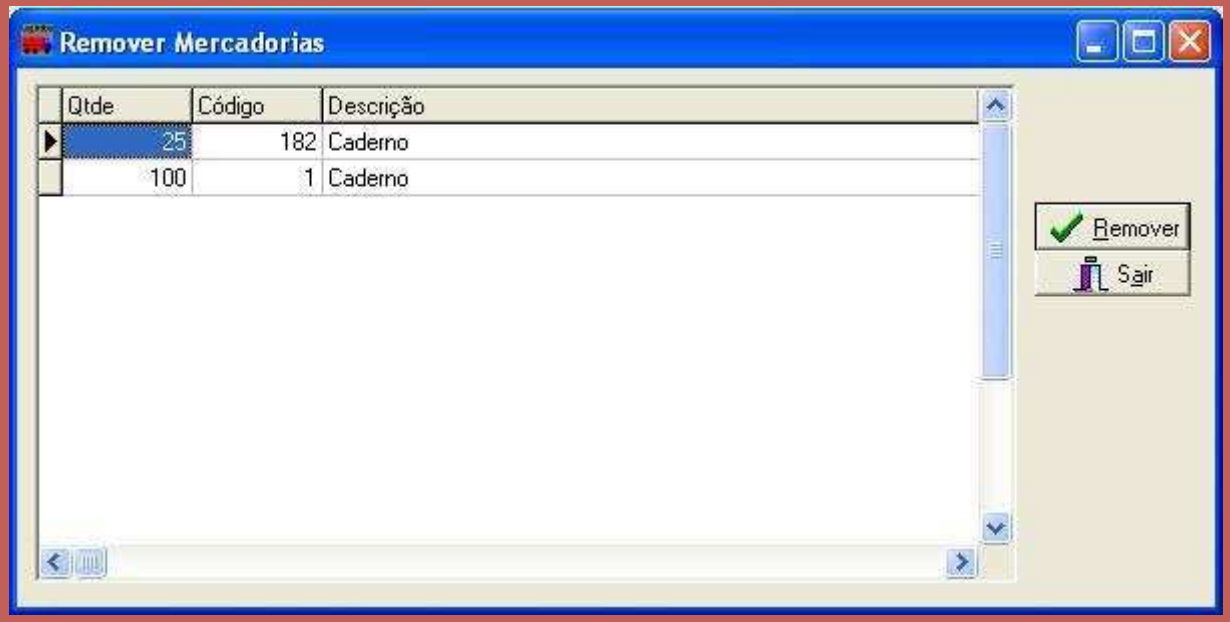


# Operacionalidade da implementação

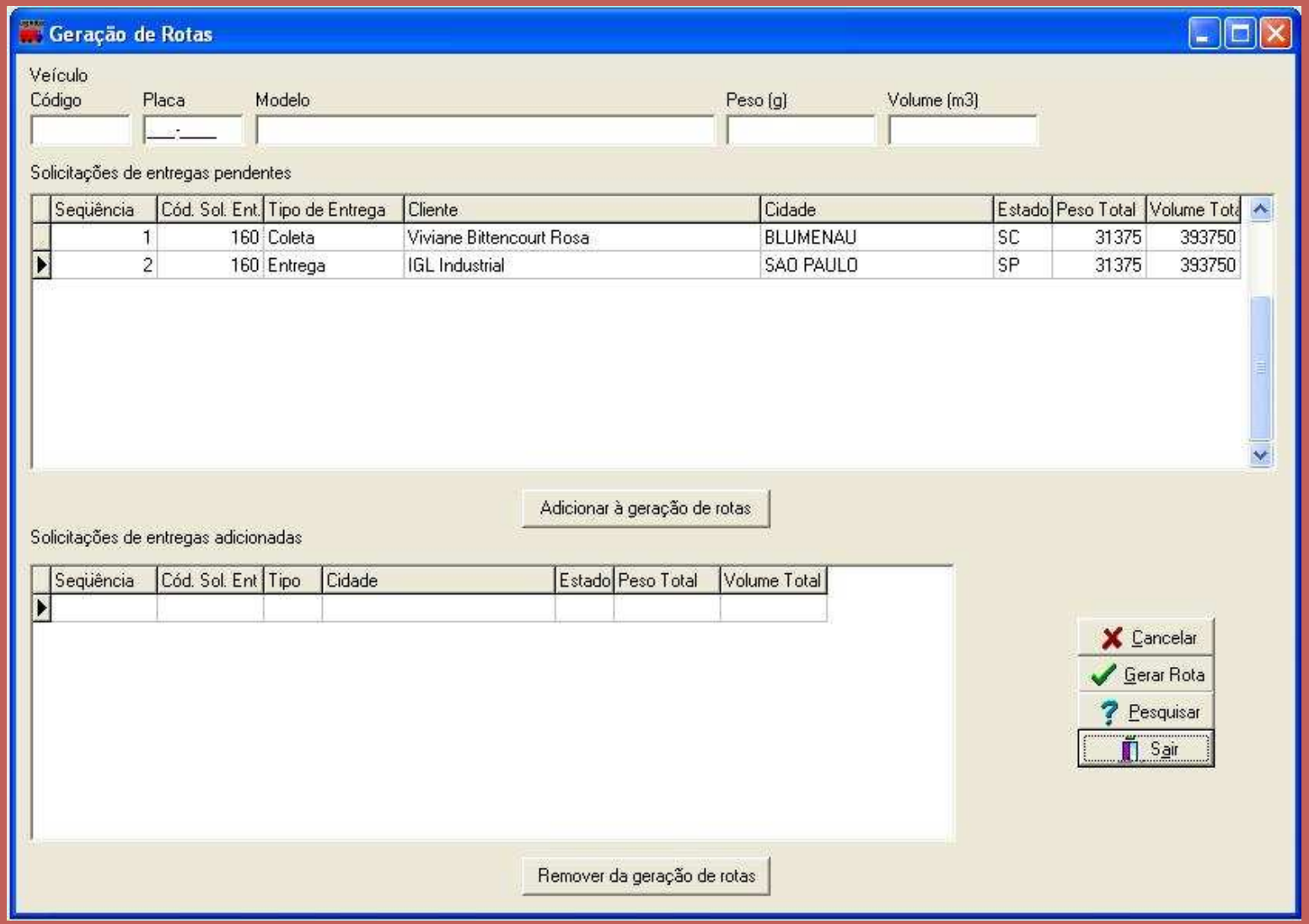


# Operacionalidade da implementação

# Operacionalidade da implementação

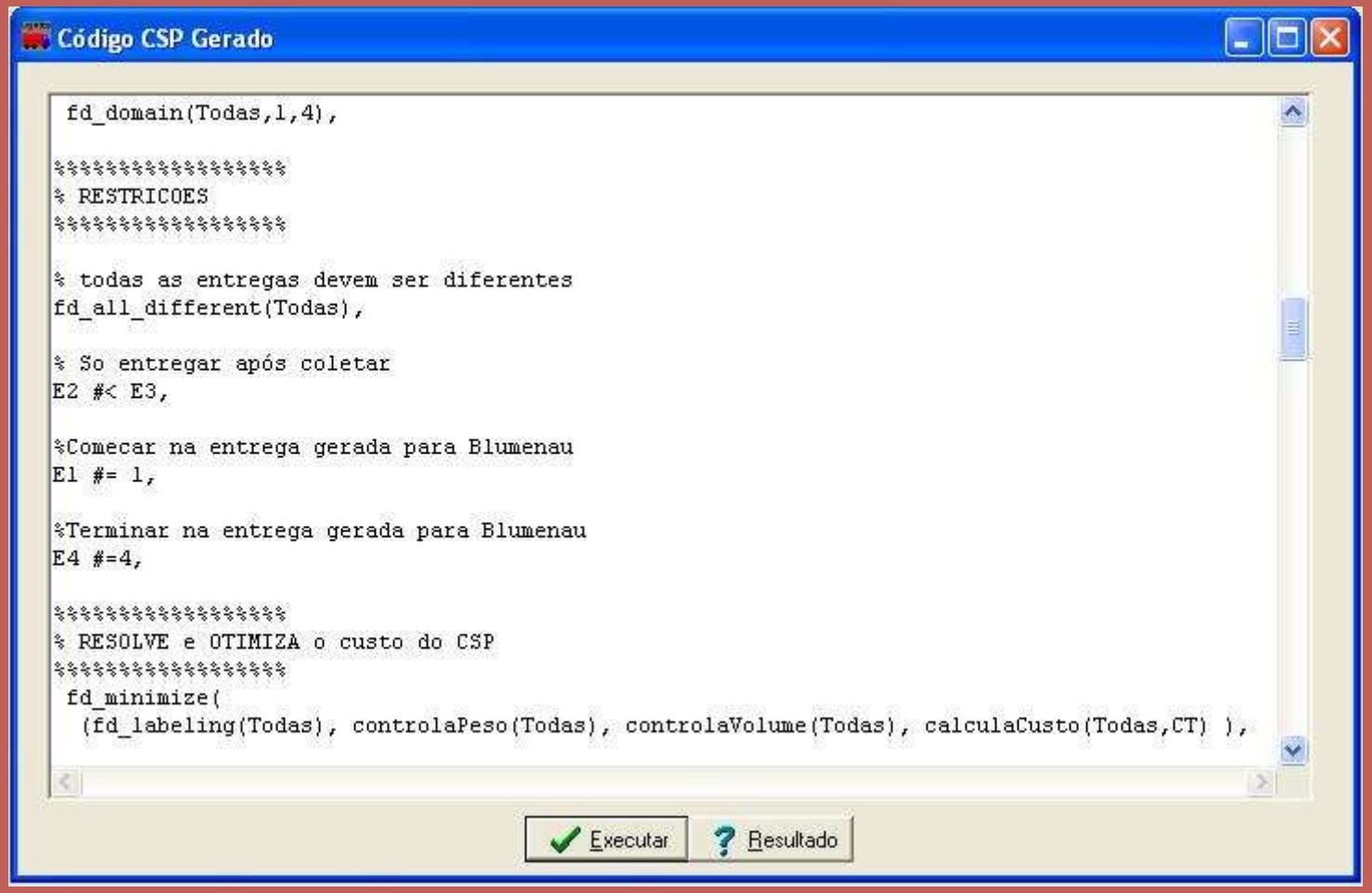


# Operacionalidade da implementação

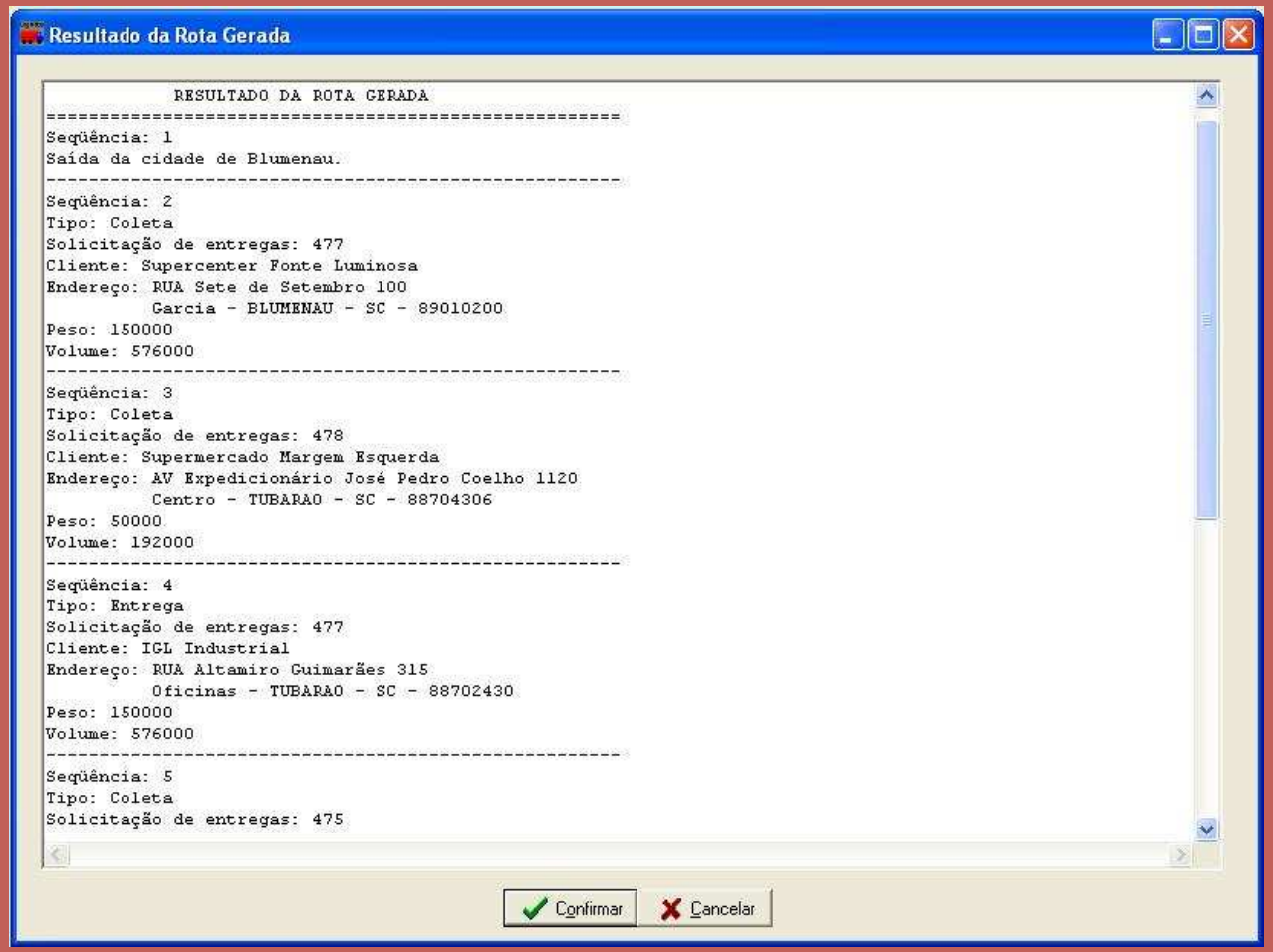


# Operacionalidade da implementação

# Operacionalidade da implementação



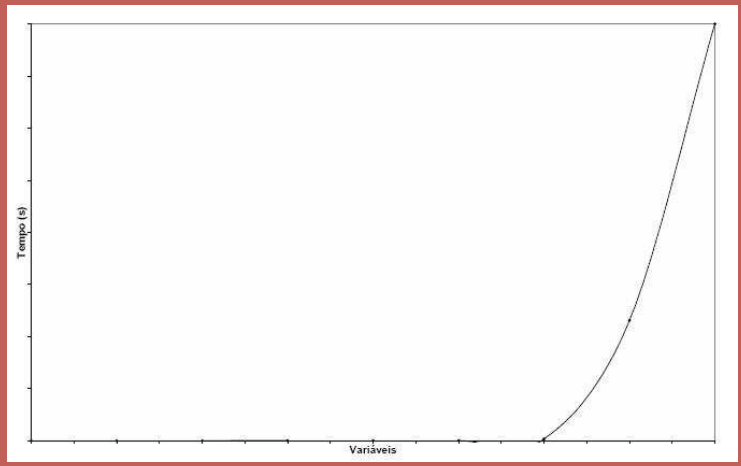
# Operacionalidade da implementação



# Operacionalidade da implementação

# Resultados e discussões

- Dificuldades ao encontrar o CSP para o problema
- Desempenho
- Trabalhos correlatos
  - Programação por propagação de restrições (Sucupira, 2004)
  - Sistemas multi-agentes no desenvolvimento de um protótipo de um sistema de logística (Gonçalves, 1997)
  - A logística na distribuição (Knaesel Neto, 1999)




| Variáveis | Tempo (s) |
|---|---|
| 2 | 0,4 |
| 4 | 0,6 |
| 6 | 1,5 |
| 8 | 2 |
| 10 | 3 |
| 12 | 18 |
| 14 | 1155 |
| 16 | 4000 |

# Conclusão

- Modelagem do CSP
- Objetivos
- Ferramentas utilizadas e estudadas
- Relevância
  - Comercial
  - Acadêmica
- Extensões
  - Definição das estradas envolvidas no trajeto
  - Definir quais os veículos que realizarão a entrega
  - Após a definição das estradas, fazer a pintura de uma mapa para o percurso definido

# Relevância pessoal

- Conhecimento da ferramenta de desenvolvimento Delphi
- Conhecimento da técnica
- Dificuldades encontradas

“Depois de algum tempo você aprende que as verdadeiras amizades continuam a crescer mesmo em longas distâncias, e o que importa não é o que você tem na vida, mas quem você tem na vida.”

William Shakspeare