# Métodos de Resolução de Problemas e de Procura.

Filipe Guimarães A85308

Sistemas de Representação de Conhecimento e Raciocínio
Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: a85308@alunos.uminho.pt

## 1 Introdução

Este relatório pretende apresentar o raciocínio usado para a resolução do problema proposto no instrumento de avaliação individual. Este instrumento pretende estimular o uso de técnicas de formulação de problemas, a aplicação de diversas estratégias para a resolução de problema com o uso de algoritmos de procura e o desenvolvimento de mecanismos de raciocínio adequados a esta problemática.

O caso de estudo proposto é a utilização dos circuitos de recolha de resíduos urbanos do concelho de Lisboa, a importação dos dados relativos aos diferentes circuitos e a respetiva representação numa base de conhecimento.

Posteriormente será necessário, através de procura não informada e informada, a seleção de circuitos que tenham em conta as seguintes considerações:

- **Inicial** – Percurso entre a garagem e o primeiro ponto de recolha;
- **Ponto de Recolha** - Remoção de resíduos num local de paragem;
- **Entre Pontos** – Percurso entre dois pontos de Recolha;
- **Transporte** – Percurso entre o último ponto de Recolha e o local de deposição de resíduos;
- **Final** – Percurso entre o local de deposição e a garagem.

## 2 Dataset e intrepretação dos dados

O *dataset* apresentado é uma adaptação do presente em http://dados.cm-lisboa.pt/no/dataset/circuitos-de-recolha-de-residuos-urbanos.

Deste *dataset* considerei importantes seis colunas das dez apresentadas, respetivamente, o ID de contentor, a latitude, a longitude, a freguesia, o tipo de lixo e a capacidade total desse mesmo contentor.

### 2.1 *Parsing* dos dados

Para usar em *prolog*, os dados tiveram de ser processados de forma a criar uma base de conhecimento. Para isso foi desenvolvido um programa em *java*. São processados os dados criando um predicado contentor com os campos descritos na seguinte figura.

```
% contentor(idContentor,latitude,longitude,freguesia,tipo,capacidade). %
contentor(0, -9.14330880914792, 38.7080787857025, "Misericórdia", "Descarga", 15000). %Local de descarga

contentor(355, -9.14330880914792, 38.7080787857025, "Misericórdia", "Lixos", 90).
contentor(356, -9.14330880914792, 38.7080787857025, "Misericórdia", "Lixos", 1680).
contentor(357, -9.14330880914792, 38.7080787857025, "Misericórdia", "Lixos", 90).
contentor(358, -9.14330880914792, 38.7080787857025, "Misericórdia", "Papel e Cartão", 1440).
contentor(359, -9.14330880914792, 38.7080787857025, "Misericórdia", "Papel e Cartão", 90).
contentor(364, -9.14337777820218, 38.7080781891571, "Misericórdia", "Lixos", 240).
contentor(365, -9.14337777820218, 38.7080781891571, "Misericórdia", "Lixos", 840).
contentor(366, -9.14337777820218, 38.7080781891571, "Misericórdia", "Lixos", 120).
contentor(367, -9.14337777820218, 38.7080781891571, "Misericórdia", "Lixos", 180).
contentor(368, -9.14337777820218, 38.7080781891571, "Misericórdia", "Lixos", 240).
```

**Figura 1.** Base de conhecimento de contentores (alguns exemplos)

Para a ligação entre os contentores foi criado um predicado ligação. Este predicado é constituído por dois contentores e a respetiva distancia entre eles.

Para a criação destas ligações criei um algoritmo no *parse* que liga todos os nós consecutivos e coloquei em *ligacoes.pl*. Para complicar a base de conhecimento,além das já referidas, adicionei também ligações de forma aleatória entre contentores e coloquei no ficheiro *ligacoes2.pl*. Exemplos de ligações são apresentadas na figura seguinte.

```
% ligacao(idContentor1,idContentor2,distancia). %
ligacao(0, 355, 0).

ligacao(355, 356, 0.0).
ligacao(356, 357, 0.0).
ligacao(357, 358, 0.0).
ligacao(358, 359, 0.0).
ligacao(359, 364, 0.007668852022762408).
ligacao(364, 365, 0.0).
ligacao(365, 366, 0.0).
ligacao(366, 367, 0.0).
ligacao(367, 368, 0.0).
```

```
% ligacao(idContentor1,idContentor2,distancia). %
ligacao(0, 355, 0).

ligacao(355, 520, 0.10622587559911699).
ligacao(355, 356, 0.0).
ligacao(356, 938, 0.5374458185022377).
ligacao(356, 357, 0.0).
ligacao(357, 611, 0.13751433115070186).
ligacao(357, 358, 0.0).
ligacao(358, 623, 0.2602214789969434).
ligacao(358, 359, 0.0).
ligacao(359, 364, 0.007668852022762408).
```

**Figura 2.** Base de conhecimento de ligações (alguns exemplos)

Os testes que serão realizados têm em conta estas duas bases de conhecimento de ligações.

## 3 Implementação da solução

### 3.1 Organização da solução

O projeto está dividido em 2 pastas, uma que contém a base de conhecimento, chamada *dados* que foi apresentada na secção anterior, e outra que contém os algoritmos desenvolvidos para o problema em questão. O ponto de partida é o *run.pl* que apresenta as definições necessárias para poder usar a base de conhecimento

```
:- include('dados/contentores.pl').
:- include('dados/ligacoes2.pl').

:- dynamic contentor/6.
:- dynamic ligacao/3.

:- include('algoritmos/breadthfirst.pl').
:- include('algoritmos/depthfirst.pl').
:- include('algoritmos/aestrela.pl').
```

e os respetivos testes para cada algoritmo desenvolvido.

```
testAEstrela(R):-
    resolve_aestrela(0,R,1500).

testBf(R):-
    breadthFirst(ligacao,0,15000,R).

testDf(R):-
    recolhaLixo(0,0,[0],"Lixos",15000,0,R).
```

### 3.2 Algoritmos implementados

Para o problema proposto desenvolvi dois algoritmos de pesquisa não informada, respetivamente *DepthFirst* e *BreadthFirst*, e um de pesquisa informada, nomeadamente, o *A**, Apresentados de seguida.

**DephtFirst**

Um algoritmo de busca deste género realiza uma busca não-informada que progride através da expansão do primeiro nó filho da árvore de busca, e se aprofunda cada vez mais, até que o alvo da busca seja encontrado ou até que ele se depare com um nó que não possui filhos . Quando isto acontece retrocede (faz *backtrack*) e começa no nó seguinte.

O algoritmo encontra-se na pasta algoritmos em *depthfirst.pl*. Para chegar à solução pretendida tive de criar 3 predicados com diferentes casos de paragem e o respetivo predicado para encontrar o nó seguinte.

O próximo nó é encontrado fazendo um *findall* a todas as ligações e escolhendo aquelas que ainda não foram visitadas.

Considerei nos testes que a garagem e a descarga eram no mesmo sitio mas pode ser indicado no predicado, respetivamente.

Tiveram de ser implementadas 2 restrições no predicado.

- **Tipo de Lixo**
  A cada nó adicionado à lista e verificado se é do tipo de lixo correto. Caso seja é descontado na capacidade. Caso contrário não é descontado da capacidade mas o camião pode na mesma passar por ele para ir para outros contentores.

- **Capacidade**
  Quando a capacidade chega a ao limite proposto (15 $m^3$), especificado no predicado, o camião sabe que tem de voltar para a garagem e depois voltar para continuar o trajeto. Para isto é considerando que ele retorna e volta ao nó que estava multiplicando por três a distância percorrida.

**BreadthFirst**

Esta pesquisa é considerada uma pesquisa não informada que expande e examina todos os vértices de um grafo. O algoritmo realiza uma busca exaustiva num grafo passando por todas as arestas e vértices do grafo. Sendo assim, o algoritmo deve garantir que nenhum vértice ou aresta será visitado mais de uma vez e, para isso, utiliza uma estrutura de dados fila para garantir a ordem de chegada dos vértices. Dessa maneira, as visitas aos vértices são realizadas através da ordem de chegada na estrutura fila e um vértice que já foi marcado não pode entrar novamente a esta estrutura.

O algoritmo encontra-se na pasta algoritmos em *breadthfirst.pl*. Para chegar à solução pretendida tive de criar 2 predicados com diferentes restrições para adicionar os nós à fila.

A minha implementação apenas diz os caminhos que pode fazer até esgotar a capacidade e ter de retornar à descarga não continuando a partir desse local.

A única diferença nas restrições do algoritmo anterior é que, como neste algoritmo os caminhos são colocados em filas, as restrições passaram para o *prepend* que as verifica e só adiciona caso sejam satisfeitas.

**A***

Este algoritmo procura o caminho em um grafo de um vértice inicial até um vértice final. Combina heurísticas presentes no algoritmo *BreadthFirst* e o algoritmo de *Dijkstra*.

O algoritmo encontra-se na pasta algoritmos em *aestrela.pl*. Para solucionar este problema usei uma das implementações usadas nas aulas práticas complementando com as respetivas alterações necessárias.

A modificação mais importante implementada foi no "goal"pretendido. Neste caso quer-se que o camião esteja sempre a passar por contentores até atingir a sua capacidade máxima. Para isso, introduzi ao longo do caminho o custo do mesmo que será a capacidade no momento e o "goal"será atingir o máximo do camião. Mais uma vez não implementei o resto do caminho tendo de ser necessário a introdução manual do ultimo visitado como o inicio da viagem para ele continuar.

## 4 Testes

Para testar os algoritmos, e como já tinha sido referido, foram utilizados dois casos, o caso de os contentores estarem encaminhados (*ligacoes.pl*) e o caso de terem também ligações aleatórias entre si (*ligacoes2.pl*).

Os testes apresentados de seguida têm a garagem e o deposito em 0 e uma capacidade de camião de 15 $m^3$.

### 4.1 Sem ligações aleatórias

| Estratégia | Tempo | Espaço | Encontrou melhor solução? |
|---|---|---|---|
| DepthFirst | util | <1G | sim |
| BreadthFirst | util | <1G | sim/não |
| A* | demorado | >2G | sim/não |

Para o algoritmo *DephtFirst* consegui-se obter uma solução completa em tempo útil recorrendo ao espaço predefinido pelo *swiprolog* (1G).

```
R = [913, 912, 898, 875, 874, 871, 870, 869, 790, 789, 881, 880, 907, 906, 963, 962, 961, 960, 959, 741, 740, 739, 738, 737, 736, 951, 897, 71
7, 716, 715, 966, 965, 924, 923, 922, 921, 846, 0, 845, 844, 843, 947, 946, 945, 944, 940, 939, 938, 700, 699, 698, 697, 696, 695, 694, 693, 6
92, 292, 291, 290, 85, 206, 205, 122, 121, 47, 48, 227, 164, 163, 267, 245, 225, 182, 181, 180, 213, 214, 67, 66, 232, 231, 265, 234, 233, 207
, 211, 208, 209, 210, 139, 138, 137, 136, 135, 134, 133, 39, 40, 41, 42, 43, 354, 353, 352, 351, 350, 349, 348, 347, 346, 0, 345, 341, 340, 34
4, 343, 342, 339, 338, 337, 336, 335, 334, 333, 664, 663, 662, 661, 660, 659, 658, 657, 656, 655, 654, 653, 652, 651, 650, 649, 648, 647, 646,
 645, 644, 643, 642, 641, 640, 639, 638, 637, 636, 635, 634, 633, 632, 631, 630, 629, 628, 627, 626, 625, 624, 623, 622, 621, 620, 619, 618, 6
17, 616, 615, 614, 0, 613, 612, 611, 603, 602, 594, 593, 592, 591, 590, 589, 588, 587, 586, 585, 584, 583, 582, 581, 580, 579, 578, 577, 576,
575, 574, 0, 573, 572, 571, 570, 569, 568, 567, 566, 565, 562, 561, 560, 559, 558, 557, 564, 563, 556, 555, 554, 553, 552, 551, 550, 549, 548,
 547, 546, 545, 544, 543, 542, 541, 540, 539, 538, 537, 536, 0, 535, 534, 533, 532, 531, 530, 529, 528, 527, 526, 525, 524, 523, 522, 521, 520
, 519, 518, 517, 516, 515, 514, 513, 512, 511, 510, 509, 508, 507, 506, 505, 504, 503, 502, 501, 500, 499, 498, 497, 493, 492, 496, 495, 494,
491, 490, 0, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480, 479, 0, 478, 477, 476, 475, 474, 473, 472, 471, 470, 469, 468, 467, 466, 465, 4
64, 463, 0, 462, 461, 460, 459, 458, 457, 456, 455, 454, 453, 452, 451, 450, 449, 448, 447, 0, 446, 445, 444, 443, 442, 441, 440, 439, 438, 43
7, 436, 435, 434, 433, 432, 431, 430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 0, 418, 417, 416, 415, 414, 413, 412, 411, 405, 4
04, 403, 410, 409, 408, 407, 406, 402, 401, 400, 399, 398, 397, 396, 395, 394, 393, 0, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382,
381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 363, 362, 361, 360, 370, 369, 368, 367, 366, 365, 364, 359, 358, 357, 356, 355, 0] ▮
```

**Figura 3.** Única solução encontrada para *DephtFirst*

Para o algoritmo *BreadthFirst* consegui-se obter uma solução imcompleta, ou seja, pelo menos um caminho até esgotar a capacidade, em tempo útil recorrendo ao espaço predefinido pelo *swiprolog* (1G).

```
R = [0, 355, 356, 357, 358, 359, 364, 365, 366, 367, 368, 369, 370, 360, 361, 362, 363, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,
 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 406, 407, 408, 409, 410, 403, 404, 4
05, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438
, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466,
467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 494, 495, 496, 49
2, 493, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523,
 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 5
52, 553, 554, 555, 556, 563, 564, 557, 558, 559, 560, 561, 562, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580
, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 602, 603, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622,
623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 65
1, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 333, 334, 335, 336, 337, 338, 339, 342, 343, 344, 340, 341, 345, 346, 347,
 348, 349, 350, 351, 352, 353, 354, 43, 42, 41, 40, 39, 133, 134, 135, 136, 137, 138, 139, 210, 209, 208, 211, 207, 233, 234, 265, 231, 232, 6
6, 67, 214, 213, 180, 181, 182, 225, 245, 267, 163, 164, 227, 48, 47, 121, 122, 205, 206, 85, 290, 291, 292, 692, 693, 694, 695, 696, 697, 698
, 699, 700, 938, 939, 940, 944, 945, 946, 947, 843, 844, 845, 846, 921, 922, 923, 924, 965, 966, 715, 716, 717, 897, 951, 736, 737, 738, 739,
740, 741, 959, 960, 961, 962, 963, 906, 907, 880, 881, 789, 790, 869, 870, 871, 874, 875, 898, 912, 913] .
```

**Figura 4.** Única solução encontrada para *BreadthFirst*

Para o algoritmo $A*$ consegui-se obter uma solução incompleta, ou seja, pelo menos um caminho até esgotar a capacidade, demorando alguns segundos aumentando o espaço disponivel para 4G.

```
R = [0, 355, 356, 357, 358, 359, 364, 365, 366, 367, 368, 369, 370, 360, 361, 362, 363, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,
 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 406, 407, 408, 409, 410, 403, 404, 4
05, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438
, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466,
467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 494, 495, 496, 49
2, 493, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523,
 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 5
52, 553, 554, 555, 556, 563, 564, 557, 558, 559, 560, 561, 562, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580
, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 602, 603, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622,
623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 65
1, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 333, 334, 335, 336, 337, 338, 339, 342, 343, 344, 340, 341, 345, 346, 347,
 348, 349, 350, 351, 352, 353, 354, 43, 42, 41, 40, 39, 133, 134, 135, 136, 137, 138, 139, 210, 209, 208, 211, 207, 233, 234, 265, 231, 232, 6
6, 67, 214, 213, 180, 181, 182, 225, 245, 267, 163, 164, 227, 48, 47, 121, 122, 205, 206, 85, 290, 291, 292, 692, 693, 694, 695, 696, 697, 698
, 699, 700, 938, 939, 940, 944, 945, 946, 947, 843, 844, 845, 846, 921, 922, 923, 924, 965, 966, 715, 716, 717, 897, 951, 736, 737, 738, 739,
740, 741, 959, 960, 961, 962, 963, 906, 907, 880, 881, 789, 790, 869, 870, 871, 874, 875, 898, 912, 913]
```

**Figura 5.** Única solução encontrada para *A**

### 4.2 Com ligações aleatórias

| Estratégia | Tempo | Espaço | Encontrou melhor solução? |
|---|---|---|---|
| DepthFirst | util | <1G | sim/não |
| BreadthFirst | util | <1G | sim/não |
| A* | demorado | >4G | não |

Para o algoritmo *DephtFirst* consegui-se obter algumas varias soluções em tempo útil recorrendo ao espaço predefinido pelo *swiprolog* (1G) mas não se conseguem obter todas para comparação de produtividade de distâncias.

```
1537.0801472623216
R = [515, 962, 961, 960, 959, 871, 870, 638, 637|...] [write]
R = [515, 962, 961, 960, 959, 871, 870, 638, 637, 636, 635, 634, 633, 632, 413, 412, 411, 405, 404, 403, 455, 563, 556, 555, 554, 483, 233, 60
2, 514, 513, 0, 512, 511, 409, 408, 407, 406, 452, 451, 450, 449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 421, 420, 4
56, 592, 591, 516, 499, 498, 497, 464, 463, 0, 462, 461, 460, 459, 402, 401, 400, 399, 490, 489, 488, 487, 486, 0, 485, 484, 40, 382, 381, 380
, 379, 378, 377, 376, 375, 532, 139, 138, 137, 577, 576, 560, 559, 558, 557, 474, 473, 472, 471, 546, 481, 0, 480, 479, 478, 477, 476, 475, 54
0, 603, 655, 654, 653, 652, 0, 651, 650, 649, 648, 646, 645, 644, 621, 620, 619, 618, 617, 227, 164, 163, 589, 588, 587, 586, 585, 584, 583, 5
82, 581, 580, 579, 643, 642, 467, 466, 465, 741, 612, 611, 357, 356, 355, 0] ;
1537.5641787050417
R = [374, 373, 372, 530, 529, 528, 527, 526, 525, 906, 963, 962, 961, 960, 959, 871, 870, 638, 637, 636, 635, 634, 633, 632, 413, 412, 411, 40
5, 404, 403, 455, 563, 556, 555, 554, 483, 233, 602, 514, 513, 0, 512, 511, 409, 408, 407, 406, 452, 451, 450, 449, 448, 447, 446, 445, 444, 4
43, 442, 441, 440, 439, 438, 437, 436, 421, 420, 456, 592, 591, 516, 499, 498, 497, 464, 463, 0, 462, 461, 460, 459, 402, 401, 400, 399, 490,
489, 488, 487, 486, 0, 485, 484, 40, 382, 381, 380, 379, 378, 377, 376, 375, 532, 139, 138, 137, 577, 576, 560, 559, 558, 557, 474, 473, 472,
471, 546, 481, 0, 480, 479, 478, 477, 476, 475, 540, 603, 655, 654, 653, 652, 0, 651, 650, 649, 648, 646, 645, 644, 621, 620, 619, 618, 617, 2
27, 164, 163, 589, 588, 587, 586, 585, 584, 583, 582, 581, 580, 579, 643, 642, 467, 466, 465, 741, 612, 611, 357, 356, 355, 0] ;
```

**Figura 6.** Algumas soluções encontradas para *DephtFirst*

Para o algoritmo *BreadthFirst* consegui-se obter uma solução imcompleta, ou seja, pelo menos um caminho até esgotar a capacidade, em tempo útil recorrendo ao espaço predefinido pelo *swiprolog* (1G).

```
R = [0, 355, 520, 414, 415, 416, 514, 515, 516, 517, 518, 559, 560, 561, 562, 565, 566, 567, 568, 569, 570, 571] ;
R = [0, 355, 520, 414, 415, 180, 181, 182, 626, 627, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392] ;
R = [0, 355, 520, 414, 415, 180, 450, 451, 869, 870, 871, 874, 664, 333, 334, 335, 336, 337, 338, 339, 342, 343] ;
```

**Figura 7.** Algumas soluções encontradas para *BreadthFirst*

Para o algoritmo *A** não se conseguiu observar uma solução nem completa nem incompleta mesmo aumentando o espaço disponível para 4G.

## 5 Conclusão e análise de resultados

De um modo geral este trabalho permitiu-me aprofundar o conceito de pesquisa não informada e informada em *prolog* abordada nas ultimas aulas práticas desta unidade curricular.

A principal dificuldade encontrada foi o tratamento do *dataset* fornecido para a criação da base de conhecimento. Primeiramente tentei fazer as ligações de forma aos contentores mais próximos ficarem conectados. A base de conhecimento ficava demasiado complexa não executando nem em tempo nem em espaço útil em nenhum algoritmo. Mesmo com a simplificação penso que os problemas que alguns algoritmos têm seriam mitigados com alteração nestas ligações.

De modo geral consegui verificar que o algoritmo que requer menos poder computacional é o *DepthFirst* executando sempre em tempo útil. O algoritmo mais pesado será o $A*$ que chega a não terminar por falta de espaço de memória.