Uma imagem com Tipo de letra, Gráficos, design gráfico, captura de ecrã

Descrição gerada automaticamente

**Relatório de Algoritmia Avançada**

**Turma 3DL - Grupo 067**

1210818- João Gomes

1200585- Manuel Silva

1210804- André Gonçalves

1201458- Jorge Moreira

Índice

[Representação de conhecimento 3](#_Toc151844936)

[User Story 510 7](#_Toc151844937)

[Conclusões 19](#_Toc151844938)

[Anexo 19](#_Toc151844939)

# Representação de conhecimento

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura - Mapa de edifícios

Uma imagem com texto, captura de ecrã, design

Descrição gerada automaticamente

Figura – Mapa de piso

Uma imagem com texto, captura de ecrã, menu, design

Descrição gerada automaticamente

Figura - Mapa de piso

Uma imagem com texto, captura de ecrã, menu, design

Descrição gerada automaticamente

Figura - Mapa de piso

A representação do conhecimento do domínio em Prolog está estruturada através de factos e regras que descrevem os edifícios, pisos, elevadores e suas respetivas conexões.

**Análise de Regras e Factos:**

Os fatos estão organizados de forma clara e intuitiva, representando os edifícios, os pisos em cada edifício e os elevadores disponíveis e também as passagens entre pisos de diferentes edifícios.

Cada facto é definido de maneira direta, facilitando assim a identificação e compreensão dos elementos do sistema.

Por exemplo:

 

Figura - "elevador (edifício, [Piso1, Piso2, ...]) " Figura - facto "pisos (Edifício, [Piso1, Piso2, ...])"

Esses fatos estabelecem relações diretas entre os edifícios e os pisos, além de indicar os elevadores disponíveis em cada edifício, fornecendo uma base fundamental para consultas e inferências.

O mapa de pisos foi representado no sistema por meio de factos que descrevem as células do mapa para cada piso em cada edifício. Cada célula é definida com coordenadas (coluna, linha), um valor que indica a transitabilidade (0 para células transitáveis) e a associação ao respetivo edifício.

Por exemplo o piso “a1”:

% Mapa do piso a1

m (1,1,0, a1). m (1,2,0, a1).

m (2,1,0, a1). m (2,2,0, a1).

m (3,1,0, a1). m (3,2,0, a1).

m (4,1,0, a1). m (4,2,0, a1).

m (5,1,0, a1). m (5,2,0, a1).

Isso define um layout de 5 colunas por 2 linhas para o piso “a1”, onde todas as células têm o valor 0, indicando que são transitáveis. Esse padrão é repetido para cada piso em cada edifício, permitindo uma representação detalhada do mapa de pisos no sistema.

As regras definidas são cruciais para inferências sobre conexões entre pisos e edifícios. Por exemplo, a regra que estabelece os corredores de passagem entre os pisos de diferentes edifícios:



Figura - "elevador (edifício, [Piso1, Piso2, ...])"

# User Story 510

**Como gestor de tarefas pretendo encontrar caminhos entre edifícios que tentem otimizar um dado critério, por exemplo, minimizar o número de utilizações do elevador, ou passar pelo menor número de edifícios. Os caminhos dentro de cada corredor dos edifícios deverão tentar minimizar a distância percorrida e ser gerados a partir de um grafo entre “células” retangulares posicionadas no corredor, devendo sempre haver uma célula junto a cada acesso a gabinete ou sala.**

Os predicados fornecidos desempenham um papel crucial ao oferecer um mecanismo para encontrar caminhos entre pisos de maneira inteligente e eficaz. Através desses predicados, é possível explorar as conexões entre os pisos de diferentes edifícios, levando em consideração a presença de corredores que ligam diretamente dois pisos ou a utilização de elevadores para alcançar pisos superiores ou inferiores.

Este sistema não apenas busca uma rota entre os pisos desejados, mas também é capaz de determinar o melhor caminho com base em critérios específicos, como a minimização do número de elevadores ou corredores ao longo do percurso. Essa funcionalidade proporciona uma abordagem flexível para encontrar a rota mais adequada, levando em consideração diferentes preferências ou restrições durante a navegação pelos edifícios.

Foi também desenvolvido um sistema de mapeamento e busca de caminhos entre células de diferentes pisos em edifícios interconectados. Para cada mapa de piso, foi criado um grafo que representa as conexões entre as células, permitindo a exploração inteligente dos espaços e a determinação de rotas otimizadas.

O processo de criação dos grafos para cada mapa de piso foi realizado através do predicado cria\_grafo/3. Este predicado constrói uma representação em grafo das células de um piso, considerando as conexões possíveis entre elas. Ao percorrer as células do mapa do piso, estabelece conexões entre elas, permitindo a navegação por corredores e o uso de elevadores para alcançar diferentes áreas do mesmo piso.

Cada grafo gerado torna-se a base para a aplicação de algoritmos de busca, visando encontrar o melhor caminho entre células específicas de um mesmo piso. Para realizar essa busca, foram implementados algoritmos como Depth First Search (DFS), Breadth First Search (BFS) e A\* (A-Star), cada um com suas estratégias únicas para encontrar rotas otimizadas.

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

Figura - Predicado de busca de caminhos entre edifícios e pisos

* **caminho\_edificios (EdOr, EdDest, LEdCam): -**

Este é o ponto de partida para encontrar um caminho entre dois edifícios. Este predicado chama caminho\_edificios2 /4 com o edifício de origem, o edifício de destino e uma lista vazia ([EdOr]) para armazenar os edifícios visitados ao longo do caminho.

* **caminho\_edificios2 (EdX, EdX, LEdInv, LEdCam): -**

Este Predicado é uma condição de paragem. Quando o edifício de origem é igual ao edifício de destino, o caminho é concluído e a lista de edifícios visitados é revertida (reverse (LEdInv, LEdCam)).

* **caminho\_edificios2 (EdAct, EdDest, LEdPassou, LEdCam): -**

Esta cláusula busca um caminho entre edifícios. Ela verifica se há uma conexão direta (liga/2) entre os edifícios EdAct e EdInt (ou vice-versa). Se houver uma conexão e EdInt não estiver na lista de edifícios já visitados (\+member(EdInt, LEdPassou)), então o predicado é chamado recursivamente para continuar a busca do caminho entre EdInt e EdDest.

* **caminho\_pisos (PisoOr, PisoDest, LEdCam, LLig): -**

Este predicado procura um caminho entre pisos de edifícios. Ele verifica se os pisos de origem e destino estão nos edifícios correspondentes (pisos/2). Em seguida, chama caminho\_edificios/3 para encontrar um caminho entre os edifícios que contêm esses pisos e segue\_pisos/4 para determinar se há conexões entre os pisos dos edifícios encontrados.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura - Predicado de busca de caminhos entre pisos

* **segue\_pisos (PisoDest, PisoDest, \_, []): -**

Este é o caso base. Quando o piso de destino é o mesmo que o piso atual, a lista de ligações entre pisos é vazia.

* **segue\_pisos (PisoDest1, PisoDest, [EdDest], [elev(PisoDest1,PisoDest)]):-**

Este predicado lida com a situação em que o próximo piso é o piso de destino. Ele verifica se existe um elevador que conecta os dois pisos nos edifícios correspondentes e define a lista de ligações como um elevador entre esses pisos.

* **segue\_pisos(PisoAct,PisoDest,[EdAct,EdSeg|LOutrosEd],[cor(PisoAct,PisoSeg)|LOutrasLig]):-**

Aqui, o sistema verifica se há um corredor direto entre o piso atual e o próximo piso. Se houver, cria uma ligação do tipo corredor entre esses pisos e continua a busca pelo próximo piso.

* **segue\_pisos(PisoAct,PisoDest,[EdAct,EdSeg|LOutrosEd],[elev(PisoAct,PisoAct1),cor(PisoAct1,PisoSeg)|LOutrasLig]):-**

Este predicado lida com a situação em que não há um corredor direto entre os pisos. Ele procura por um piso intermediário conectado por um elevador nos edifícios correspondentes. Se encontrar, define a lista de ligações como um elevador do piso atual para o intermediário, seguido por um corredor do intermediário para o próximo piso.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura - Algoritmo que encontra o melhor caminho

* **melhor\_caminho\_pisos (PisoOr, PisoDest, LLigMelhor): -**

Este predicado é o ponto de entrada para encontrar o melhor caminho entre os pisos de origem e destino. Ele utiliza o predicado caminho\_pisos/4 para encontrar todos os caminhos possíveis entre os pisos e, em seguida, chama menos\_elevadores/4 para determinar o caminho com menos elevadores e corredores.

* **menos\_elevadores([LLig], LLig, NElev, NCor): -**

Este é o caso base. Quando há apenas um caminho na lista, ele conta o número de elevadores e corredores nesse caminho.

* **menos\_elevadores([LLig|OutrosLLig], LLigR, NElevR, NCorR): -**

Neste predicado, é realizada uma comparação entre diferentes caminhos para encontrar o caminho com menos elevadores. Ele calcula o número de elevadores e corredores em cada caminho e escolhe o caminho que tiver menos elevadores. Se houver um empate no número de elevadores, escolhe o caminho com menos corredores.

* **conta ([],0,0): -**

Este é o caso base para a contagem de elevadores e corredores em uma lista vazia.

* **conta ([elev (\_, \_) |L], NElev, NCor): -**

Este predicado conta a quantidade de elevadores em um caminho, incrementando o contador de elevadores.

* **conta ([cor (\_, \_) |L], NElev, NCor): -**

Este predicado conta a quantidade de corredores em um caminho, incrementando o contador de corredores.

Uma imagem com texto, Tipo de letra, caligrafia, branco

Descrição gerada automaticamente

Figura - Resultado do algoritmo melhor\_caminho\_pisos/3

Na figura 11, é possível evidenciar o resultado obtido quando invocado o predicado melhor\_caminho\_pisos/3 para os pisos a1 (edifício a) e o piso c3 (edifício c).

Assim, a sequência começa com **melhor\_caminho\_pisos/3** para encontrar todos os caminhos possíveis, passa por **caminho\_pisos/4** que utiliza **segue\_pisos/4** para determinar os caminhos entre pisos, e finalmente utiliza **menos\_elevadores/4** que, por sua vez, usa **conta/3** para determinar o caminho com o menor número de elevadores e corredores. Esses predicados estão interligados para encontrar e selecionar o melhor caminho entre os pisos desejados.

**Mapas de piso e algoritmos de busca:**

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura - Predicado responsável por criar grafos

**cria\_grafo/3:**

* cria\_grafo (\_, 0, \_): -!.

É uma cláusula de paragem que interrompe a recursão quando a variável Lin atinge zero.

* cria\_grafo (Col, Lin, P): -

cria\_grafo\_lin (Col, Lin, P), Lin1 is Lin - 1, cria\_grafo (Col, Lin1, P)., chama o predicado cria\_grafo\_lin/3 para criar o grafo para uma linha específica do piso. Em seguida, decrementa o contador de linhas Lin e chama recursivamente o cria\_grafo/3 até atingir a condição de paragem.

**cria\_grafo\_lin/3:**

* cria\_grafo\_lin (0, \_, \_): - !.

É uma cláusula de paragem que interrompe a recursão quando a variável Col atinge zero.

* A segunda cláusula realiza as operações para criar as conexões entre as células do mapa do piso.
* Verifica se a célula é transitável (m (Col, Lin, 0, P)).
* Estabelece conexões para as células vizinhas, verificando se são transitáveis e adiciona arestas no grafo (assertz(ligacel(...))).
* Leva em consideração conexões diagonais, adicionando arestas com um peso específico quando as células adjacentes também são transitáveis.
* Recursivamente, move-se para a próxima coluna (Col1 is Col - 1) para criar as conexões entre as células dessa linha.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 13 - Algoritmo Depth First Search

Este algoritmo é utilizado para encontrar um caminho entre células de um grafo, indo o mais fundo possível ao longo de cada ramificação antes de retroceder.

* **dfs/4:** Este é o predicado principal que inicia a busca DFS. Ele mede o tempo de execução, inicia a busca com uma lista de visitados que começa apenas com a célula de origem e retorna o caminho encontrado.
* **dfs/5:** Aqui temos a definição da busca em profundidade propriamente dita, recebendo como argumentos a célula de origem, a célula de destino, o piso, uma lista de células já visitadas e retorna o caminho encontrado.
* Na primeira cláusula, se a célula de origem está diretamente conectada à célula de destino no grafo (ligacel (CelulaOrigem, CelulaDestino, Piso, \_)), é gerado o caminho reverso da célula destino até a célula origem (o caminho é invertido para representar a ordem correta).
* Na segunda cláusula, verifica-se se a célula de origem possui uma conexão com uma célula intermediária (ligacel (CelulaOrigem, CelulaIntermediaria, Piso, \_)). Se essa célula intermediária ainda não foi visitada (\+ member (CelulaIntermediaria, Visitados)), então a busca DFS é chamada recursivamente, adicionando a célula intermediária na lista de visitados.

Uma imagem com Tipo de letra, texto, branco, caligrafia

Descrição gerada automaticamente

Figura - exemplo DFS

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura - Algoritmo DFS melhorado

Este é um aprimoramento do algoritmo de busca em profundidade (Depth First Search - DFS) chamado better\_dfs. Ele busca um caminho entre uma célula de origem (CelulaOrigem) e uma célula de destino (CelulaDestino) em um determinado piso (Piso), mas com uma melhoria: retorna o melhor caminho possível entre todos os caminhos encontrados.

* **better\_dfs/4:** Este predicado principal chama all\_dfs para gerar todos os caminhos possíveis entre a célula de origem e a célula de destino. Em seguida, utiliza o predicado melhor\_caminho para selecionar o caminho mais curto dentre todos.
* **all\_dfs/4:** Este predicado é responsável por encontrar todos os caminhos possíveis entre a célula de origem e a célula de destino no piso específico. Ele usa o predicado dfs/4 (que já explicamos anteriormente) para encontrar esses caminhos, utilizando findall para coletar todos os resultados.
* **melhor\_caminho/2:** Esse predicado é crucial para selecionar o caminho mais curto dentre todos os caminhos encontrados.

Funciona da seguinte forma:

* A primeira cláusula garante que se houver apenas um caminho, esse será o melhor.
* As cláusulas subsequentes comparam o tamanho (length) dos caminhos (Caminho1 e Caminho2). O predicado seleciona o caminho mais curto entre os dois e o mantém como MelhorCaminho. Uma imagem com texto, Tipo de letra, branco

  Descrição gerada automaticamente

Figura - exemplo better\_dfs

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 17 - Algoritmo Breadth First Search

No algoritmo BFS, ele explora todos os vizinhos da célula de origem antes de seguir para os vizinhos dos vizinhos, ou seja, ele expande uniformemente em todas as direções antes de seguir adiante.

* **bfs/4:** Esse é o ponto de entrada do algoritmo. Ele inicia a contagem do tempo (get\_time(Ti)), chama o predicado bfs2/4 e, ao final, calcula o tempo gasto na busca (get\_time(Tf) e cálculo de Tempo).
* **bfs2/4:** É a parte principal do BFS.

A primeira cláusula é a condição de paragem. Quando a célula de destino é alcançada, ela reverte o caminho e o retorna.

* A segunda cláusula é a expansão do BFS. Ela gera novos caminhos a partir da célula de origem (CelulaOrigem) e verifica se esses caminhos passam por células ainda não visitadas. Usa findall para criar uma lista Lista com novos caminhos possíveis a partir da célula de origem. Então, atualiza a Fila com esses novos caminhos (append(Fila,Lista,Fila2)) e continua a busca (bfs2 recursivamente).

Uma imagem com texto, Tipo de letra, branco, tipografia

Descrição gerada automaticamente

Figura - exemplo BFS

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura - Algoritmo A\*

Este é um algoritmo A\* (A-star), uma busca informada que utiliza uma heurística para encontrar o caminho mais curto entre um ponto de origem (Orig) e um ponto de destino (Dest) em um grafo representado pelos predicados ligacel/4.

* **aStar/5:** Este é o ponto de entrada do algoritmo. Ele inicia a contagem do tempo (get\_time(Ti)), chama o predicado aStar2/5 e, no final, calcula o tempo gasto na busca (get\_time(Tf) e cálculo de Tempo).
* **aStar2/5:** É a parte principal do A\*.
  + A primeira cláusula é a condição de paragem. Quando a célula de destino é alcançada, ela reverte o caminho e retorna-o.
  + A segunda cláusula é a expansão do A\*. Ele avalia os vizinhos da célula atual (Act) para encontrar um novo caminho. Para cada vizinho (X), verifica se já foi visitado, calcula o custo até aquele vizinho (CaX), calcula uma estimativa do custo do vizinho até o destino (EstX) usando uma heurística (no caso, a distância euclidiana entre as células), e calcula o custo total (CEX). Esses caminhos são adicionados à lista Novos.
  + Depois, todos os caminhos novos são adicionados à lista de outros caminhos (append(Outros,Novos,Todos)) e são ordenados (sort(Todos,TodosOrd)).
  + Por fim, ele continua a busca recursivamente com as novas possibilidades ordenadas.
* **estimativa/3:** É a heurística utilizada pelo algoritmo A\*. Neste caso, é a distância euclidiana entre duas células no espaço 2D. Essa função estima o custo entre uma célula (cel(X1,Y1)) e outra (cel(X2,Y2)) com base nas coordenadas dessas células.

Uma imagem com texto, Tipo de letra, branco

Descrição gerada automaticamente

Figura - exemplo A\*

**Integração da representação do conhecimento do Domínio e algoritmos:**

Para realizar a integração foi melhorado o predicado melhor\_caminho\_pisos/3, tendo agora a funcionalidade de buscar o melhor caminho dentro de um piso através do A\*.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura - melhor\_caminho\_pisos\_melhorado

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Figura - melhor\_caminho\_pisos\_melhorado

Foi criado o predicado **extrair\_pisos/2** que adiciona na lista Cam, todos os pisos por onde vai passando e consequentemente faz aStar entre os pisos encontrando o melhor caminho.

**Cel(3,1) corresponde a um corredor.**

**Cel(5,2) corresponde a um elevador.**

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura - exemplo do melhor\_caminho\_pisos\_melhorado

# Conclusões

O uso do algoritmo A\* para fazer o cálculo do custo quando procuramos o melhor caminho entre pisos é devido a duas razões. A primeira é que este algoritmo está mais propicio a utilizar diagonais, o que por sua vez causa que o custo seja menor, já que o robot iria ter de percorrer menos células. A segunda é que também o tempo necessário para executar o algoritmo é menor que nos outros. Isto pode ser verificado no ficheiro em anexo, onde é estudada a analise de complexidade.

Por sua vez o predicado melhor\_caminho\_pisos\_melhorado/3 permite ao robot calcular qual o melhor caminho a ser utlizado. A utilização do A\* permite que o robot seja eficiente enquanto travessa pelo campus, mas que seja o mais rápido possível, permitindo que o robot realize mais tarefas ao longo do dia.

A existência de outros algoritmos permite alterar que tipos de rotas queremos do robot, que apesar de mais lentas, podem ser necessárias. O uso seja do predicado bfs/4 ou de better\_dfs/4 permite obter o mesmo caminho, com o mesmo custo, mas devido ao tempo de execução de bfs ser substancialmente menor do que o tempo de better\_dfs, este seria o preferido em caso de necessidade de alterar o algoritmo. Esta necessidade pode ser causada pela preferência que o robot não faça diagonais, ou que caso as faça seja se apenas este movimento seja possível.

# Anexo

