Projeto de Tópicos Avançados em Bases de Dados

Grupo:

João Temudo Vieira, up201905419

Pedro Alexandre Sampaio Costa Leite, up201906697

Pedro da Cruz Carvalho, up201906291

1 Introdução

Neste projeto foi implementada uma base de dados espacial, com uma ligação entre PostgreSQL e YAP que utilisa C como um meio intermediário que liga as duas, e uma ferramenta de visualização criada em Python de modo a tornar a informação na base de dados mais interpretável.

O objetivo deste projeto é criar uma base de dados que permite a um resolvedor jogar o jogo “Tangram”, sendo este resolvedor programado em Prolog no sistema YAP (*Yet Another Prolog*). Infelizmente, devido a uma falta de tempo e experiência com a linguagem Prolog, não foi possível criar o resolvedor.

Na secção 2, é explicada a estrutura da base de dados PostgreSQL, que foi gerada de modo a permitir não só o armazenamento das peças e puzzles, mas também de soluções, com o objetivo de ser interpretável por C e por Python. Na secção 3, é explicado o modo como C foi usado para a transformar a informação guardada na base de dados em termos Prolog, e os predicados Prolog criados em C. Na secção 4, é explicado o funcionamento do script Python que permite visualizar as peças criadas na secção 2. Na secção 5 são incluidas algumas imagens dos plots gerados pelo script de Python.

2 SQL

Na plataforma PostgreSQL (com um ficheiro importável “project.sql”), começamos por criar a tabela “pieces” que contém todas as peças do puzzle. Esta tabela é constituída pelas colunas: “id”, um inteiro que vai de 1 a 7 e que é incrementado automaticamente, “color”, uma string que se refere à cor da peça, e “shape” que é um campo “Polygon” em que cada ponto representa um vértice.

Criamos, também, uma tabela “puzzles” que contém os 3 puzzles por resolver. Esta tabela é constituída pelas colunas: “id” um inteiro que vai de 1 a 3 e que é incrementado automaticamente, “color”, uma string que se refere à cor do puzzle e “shape” que é um campo “Polygon” em que cada ponto representa um vértice.

No final, Criamos uma tabela “solver” que contém os 3 puzzles resolvidos. Esta tabela é constituída pelas colunas: “id” valor que vai de 1 a 3 e que é incrementado

automaticamente, referente à solução, “id_puzzle” referente ao puzzle que vai ser resolvido, “id_piece” referente à peça que vamos colocar no puzzle, “pT” é um ponto que diz a translação (no eixo do x e do y) que a peça deve tomar para se colocar na posição certa e “pR,” um float que nos diz a rotação em radianos, no sentido contrário aos ponteiros do relógio, que a peça deve fazer sobre o seu primeiro ponto para estar na posição certa (este valor é posteriormente multiplicado por pi no Python).

3 Ligação SQL-C-YAP

Na ligação entre a base de dados em PostgreSQL e YAP, que utiliza a linguagem C como intermediária, foram definidos 16 predicados de modo a facilitar a passagem de informação para o YAP de um modo fiável, e 3 funções auxiliares em C que existem de modo a traduzir a informação de modo universal.

O modo como poligonos e soluções são guardados no YAP é feito de formas diferentes, de modo a permitir a um resolvedor tentar obter uma solução e guardá-la, com um método para guardar esta informação de modo fácilmente interpretável e transformável. Poligonos são representados do modo [[n | p_1, …, p_n], ‘x_1 y_1’, … ‘x_k y_k’], onde n é o número de buracos no poligono mais 1, p_i representa o número de pontos no buraco (incluindo o segundo ponto na origem), à exceção de p_1, que representa o número de pontos no “hull” do poligono. Os pares x_i e y_i representam as coordenadas no ponto i, onde i pode ir de 1 até $\sum_{i=0}^{k} p_i$. As soluções ao puzzle são listas [x, y, R] onde x e y representam o movimento que a peça tem de fazer no eixo do x e do y, respetivamente, e R representa os π radianos no sentido contrário aos ponteiros do relógio que a peça tem de rodar sobre o seu primeiro ponto.

A explicação do código irá começar pelas funções auxiliares na secção 3.1, seguida pela explicação dos predicados na secção 3.2.

3.1 Funções auxiliares

A primeira função auxiliar é “*unravel_polygon*”, que recebe como primeiro parâmetro um apontador para um array de caracteres, no qual pode guardar a tradução do segundo parâmetro, sendo este um termo vindo do YAP que representa um poligono. Esta função é utilizada para mandar *querys* à base de dados a partir do YAP.

A segunda função auxiliar é “*print_puzzle”*, que recebe o resultado de uma query que contem um poligono, o numero de buracos no poligono e a sua representação em texto, e transforma-o no formato especificado no início da secção 3.

A terceira função auxiliar é “*print_piece”*, que é uma versão simplificada de *print_puzzle* que recebe apenas o número de pontos de um poligono e a sua representação em texto para gerar o termo de Prolog equivalente.

3.2 Predicados

O redicado “*db_connect*”, que tem como argumentos *host*, utilizador, *password* e base de dados, e guarda um *connection handle* no quarto argumento de output se

conseguir encontrar uma base de dados PostgreSQL que corresponde aos argumentos fornecidos. Retorna falso se não conseguir encontrar uma base de dados.

O predicado "*db_disconnect"* tem como argumento um *connection handle* tal como o fornecido por *db_connect* e fecha a ligação.

O predicado "*db_query*", que recebe como argumentos um *connection handle* e um string com uma *query*, guarda no seu terceiro argumento o *result set* dessa *query* caso esta tenha sido executada com sucesso e retornado elementos. Se não o tiver, retorna falso.

O predicado "*db_get_pieces"* recebe como argumento um *connection handle* e guarda o *result set* de uma *query* já formatada para a execução do predicado *get_piece*.

O predicado "*get_piece"*, que recebe como argumento um *result set* e um número inteiro, guarda no terceiro argumento um termo que representa o poligono encontrado no indice indicado no segundo argumento.

Os predicados "*db_get_puzzles"* e "*get_puzzle"* são idênticos a *db_get_pieces* e *get_piece*, mas para a tabela dos puzzles invés da tabela das peças para resolver um puzzle.

O predicado *"db_get_solutions"* difere de *"db_get_pieces"* ao receber 3 argumentos, sendo o primeiro um *connection handle* e o segundo um número que, se existir um puzzle com esse indice, permite ao predicado guardar um *result set* no terceiro argumento com as soluções de cada peça para esse puzzle.

O predicado "*get_solution"* é idêntico a *get_piece*, mas guarda o tuplo definido no início da secção 3 para soluções invés do poligono da peça do segundo argumento.

O predicado "*print_polygon"* recebe um poligono como argumento e imprime a sua representação em *WKT* (well-known text).

O predicado "*st_differentiate"* recebe como argumentos um *connection handle* e dois poligonos, e guarda no quarto argumento o primeiro poligono exceto a sua interseção com o segundo poligono, que pode ou não ser um conjunto vazio.

O predicado "*st_covers*", que recebe como argumentos um *connection handle* e dois poligonos, é verdade se todos os pontos do segundo poligono estiverem contidos no primeiro poligono.

Os predicados "*st_translate"* e "*st_rotate"* recebem os mesmos argumentos, sendo estes um connection handle, um poligono e guardam um tuplo de solução. A diferença reside no facto de *st_translate* guardar o resultado da translação do poligono recebido após ser movido pelo x e y do tuplo, enquanto *st_rotate* guarda o resultado da rotação por R (do tuplo) * π radianos, no sentido contrário aos ponteiros do relógio, do poligono.

4 Python

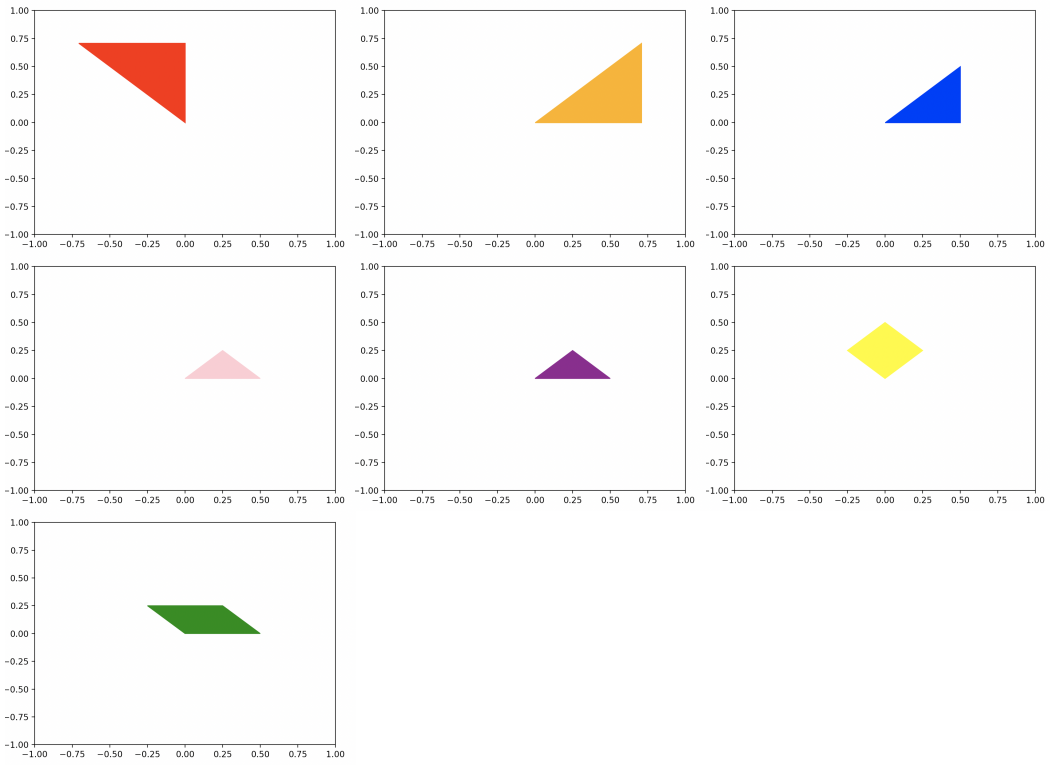
Em Python, no ficheiro "plot.py", damos a opção ao utilizador de escolher o que quer visualizar. Se o utilizador quer ver uma peça, seleciona o "1" e, de seguida, aparece-lhe uma mensagem semelhante, com uma escolha de 1 a 7, onde cada número representa uma das peças do puzzle. Após escolher a peça, aparece a sua representação gráfica. Se o utilizador quer ver os puzzles por resolver, deve selecionar "2" no primeiro menu e de seguida o puzzle que quer visualizar, sendo as escolhas de 1 a 3. Se o utilizador quiser ver os puzzles resolvidos, deve selecionar "3" no menu inicial e de seguida o puzzle resolvido que quer visualizar, as escolhas são, mais uma vez, de 1 a 3. Depois de visualizar, o utilizador recebe novamente a primeira mensagem e o processo repete-se, até ao utilizador pressionar "q" para terminar o programa.

Fazemos a conexão do python à nossa base de dados em PostgreSQL e criamos um cursor para interagir com a base de dados. E no caso do utilizador escolher visualizar as peças, fazemos uma consulta SQL que seleciona as colunas “shape“ , “color” e “id” (que vai ser igual ao “sub-number” selecionado), da tabela “pieces” e executamos utilizando o cursor. No caso do utilizador escolher visualizar os puzzles por resolver, fazemos uma consulta SQL que seleciona as colunas “shape“, “color” e id” (que vai ser igual ao “sub-number” selecionado), da tabela “puzzles” e executamos utilizando o cursor. No caso do utilizador escolher visualizar os puzzles resolvidos, fazemos uma consulta SQL que seleciona a “shape”, o “solver_pR”, “translation_x”, “translation_y” e “id” (que vai ser igual ao “sub-number” selecionado), das tabela “pieces”, “puzzle” e “solver”. Na consulta, aplicamos a rotação à “shape” utilizando o “solver_pR” multiplicado por pi e aplicamos translação ao produto dessa rotação utilizando o “translation_x” e o “translation_y”. No final executamos utilizando o cursor. No final, damos “fetch” ao cursor, para obter os resultados das nossas consultas.

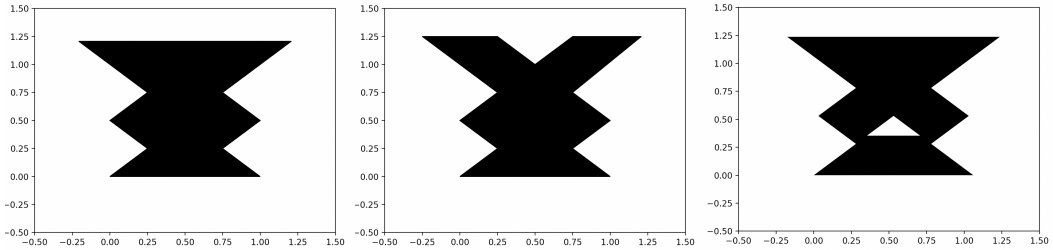
Desenhamos o nosso gráfico, que varia de dimensões consoante o que queremos visualizar, no caso das peças vai de -1 a 1, no x e no y, e no caso dos puzzles por resolver e resolvidos vai de -0.5 a 1.5, no x e no y. Na visualização de todas as figuras (com a exceção do terceiro puzzle), carregamos o “shape” da figura, retiramos as suas coordenadas e desenhamos o polígono, preenchido com a “color”. No caso da visualização do terceiro puzzle, fazemos o mesmo que fizemos anteriormente ao polígono exterior e ao polígono interior, sendo que o exterior foi preenchido a preto e o interior a branco. Visualizamos o gráfico e fechamos a conexão.

5 Resultados

Peças, de 1 a 7, respetivamente:



Puzzles por resolver, de 1 a 3, respetivamente:



Puzzles resolvidos, de 1 a 3, respetivamente: