Projeto Prático de Desenvolvimento de Software

Gestão de Viagens e Melhores Trajetos

Usando Estruturas Dinâmicas de Dados em C

Linguagens de Programação 1
Rui Silva Moreira
rmoreira@ufp.edu.pt

Bruno Gomes
bagomes@ufp.edu.pt

João Viana
jviana@ufp.edu.pt

Algoritmos e Estruturas de Dados 1
José Torres
jtorres@ufp.edu.pt

Tiago Soares da Costa
tscosta@ufp.edu.pt

Miguel Chaves
mchaves@ufp.edu.pt

(versão 1.0)
Outubro de 2021
Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia

Descrição do problema

Uma empresa de viagens do Porto, a Porto-Pontos, oferece serviços de planeamento de viagens aos seus clientes. Pretende-se desenvolver um sistema para registar e gerir dinamicamente toda a informação necessária ao funcionamento da empresa Porto-Pontos. Para além do registo de toda a informação, pretende-se fornecer aos clientes indicações sobre os melhores trajetos de viagem de acordo com os seus destinos, de modo a que possam selecionar o melhor caminho a realizar.

Como parte do serviço de planeamento de viagens, a Porto-Pontos mantém, para cada um dos utilizadores registados, um histórico de viagens realizadas. Identificação de utilizador, cidades visitadas (trajeto efetuado) e data de início/fim, são algumas das informações que estão presentes neste histórico. Para além do histórico de viagens, cada utilizador possui ainda data de nascimento, data de registo, morada, contactos e informação de faturação. Já para cada uma das cidades o sistema mantém informação geográfica e um conjunto de pontos de interesse.

Imaginem, por exemplo, que pretendiam viajar para a Austrália mas, dado o elevado número de locais que gostariam de visitar, existem inúmeros trajetos possíveis para percorrer esses pontos da forma mais optimizada.

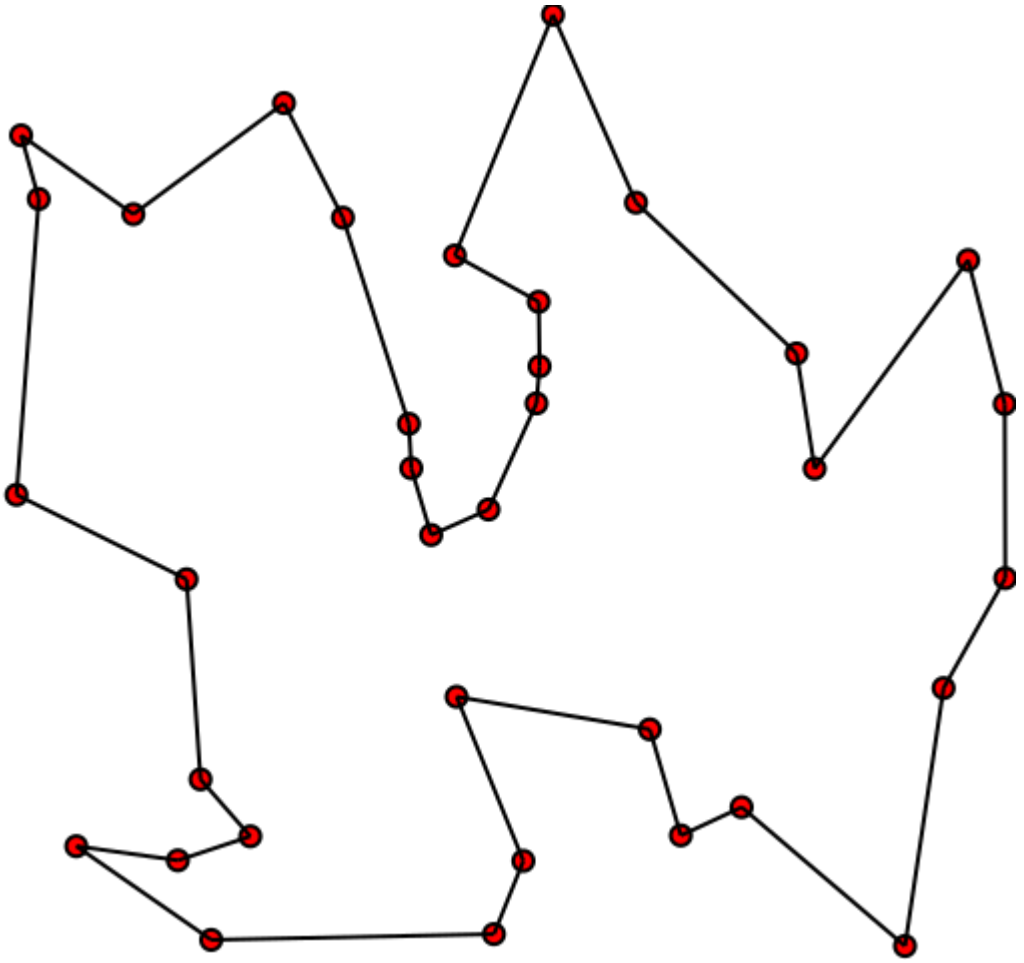

Figura 1: trajeto possível a efectuar entre várias cidades durante uma viagem.

Vamos supor que um cliente pretende visitar a Austrália, indicando à empresa quais os pontos que gostaria de visitar nesse país. A empresa começa por construir um mapa com a distribuição espacial (x,y) de cada local a visitar pelo cliente, como se pode ver na Figura 1. Com base nas coordenadas de cada local é possível calcular a distância Euclidiana (através do teorema de Pitágoras) entre cada dois pontos/locais. Com base nessas distâncias a empresa pretende calcular o trajeto mais curto para percorrer todos os locais, começando numa determinada cidade, passando em todos os locais uma única vez e retornando ao local inicial.

Este problema é conhecido como Problema do Caixeiro Viajante - *Travelling Salesman Problem* (TSP). É um problema de otimização complexo em termos computacionais porque exige combinar e comparar exaustivamente um elevado número de trajetos possíveis. Trata-se de um problema da classe NP-completo, significando que, no pior cenário, o tempo de execução para encontrar a melhor solução aumenta exponencialmente com o número de pontos a visitar.

Algoritmo Genético (AG)

Existem vários algoritmos que podem ser utilizados para encontrar soluções deste tipo de problemas. Neste projeto iremos utilizar um Algoritmo Genético (AG) que na prática utiliza uma heurística de pesquisa inspirada na teoria da evolução natural das espécies, de Charles Darwin. Este algoritmo reflete o processo de seleção natural onde os indivíduos mais aptos em cada geração (melhores trajetos encontrados nessa geração) são selecionados para reprodução, com maior probabilidade, para gerar novos trajetos para formar a próxima geração de descendentes, abrindo mais possibilidades de melhorar a solução (diminuir o trajeto entre todos os locais a visitar).

A abordagem deste algoritmo permite encontrar soluções viáveis de forma não exaustiva e portanto sem percorrer todo o espaço de possíveis soluções. Não garante que encontra a solução óptima mas, normalmente, permite ir evoluindo para a descoberta de uma melhor solução à medida que vai avançando nas gerações seguintes. O número total de gerações consideradas é, habitualmente, um parâmetro do AG.

Neste algoritmo, cada trajeto pode ser representado por um array contendo sequencialmente os IDs das cidades pela ordem que se vão visitar. Para cada trajeto será possível calcular a distância total percorrida com base na distância Euclidiana entre cada par de locais vizinhos. A população inicial de trajetos pode ser gerada de forma aleatória, ou seja, escolhendo sequências de visita possíveis entre os vários locais a visitar usando permutações aleatórias.

Neste algoritmo teríamos então que representar a seguinte informação:

- **Gene**: cidade representada por um ID e respectivas coordenadas (x, y);
- **Indivíduo** (aka Cromossoma): trajeto entre as cidades, que satisfaça as condições do problema:
    - i) visitar todas as cidades;
    - ii) passar em cada cidade uma única vez;
    - iii) retornar à cidade inicial;
- **População**: conjunto de trajetos possíveis, i.e., uma coleção de Indivíduos;

- **Pais**: dois trajetos possíveis (Indivíduos progenitores) cujos genes são combinados para criar dois novos trajetos (dois Indivíduos descendentes);
- **Seleção:** operação de escolha, entre os indivíduos da população duma geração, de pares de pais ou progenitores para se cruzarem e originarem pares de indivíduos filhos para a população da geração seguinte
- **Cruzamento**: operação de combinação dos genes de pares de pais para gerar a próxima População, i.e., próxima geração de trajetos a avaliar;
- **Mutação**: forma de introduzir variação na População de trajetos, trocando aleatoriamente duas cidades de um trajeto possível;
- **Aptidão** (Fitness): função que permite aferir a qualidade de um Indivíduo, i.e., quão melhor ou pior é um trajeto (no nosso caso poderá ser o inverso da distância do trajeto);
- **Elitismo**: critério para transportar os melhores indivíduos para a próxima geração. Permite a passagem direta de alguns indivíduos para a próxima geração, evitando assim que a população "piore".

O algoritmo genético (AG) segue repetidamente um conjunto de passos, que se podem resumir na seguinte sequência:

1. Criar a **População** inicial com P indivíduos;
2. Determinar a **Aptidão** de cada **Indivíduo** da População;
3. **Selecionar** pares de indivíduos, os **Pais**, a serem usados para cruzar a próxima geração;
4. Gerar, para cada par de pais, um par de filhos para a próxima **População/Geração** aplicando o operador de **Cruzamento** aos pais selecionados;
5. Aplicar **Mutações** nos indivíduos da nova geração para aumentar a variabilidade na População;
6. Repetir o processo a partir do passo (2).

Por exemplo, imaginemos que temos N=6 cidades para visitar. Cada cidade será identificada por um ID (0...5) e caracterizada por duas coordenadas (x,y):

C0 = {id=0, coordenadas={1.0, 0.0}}

C1 = {id=1, coordenadas={1.0, 1.0}}

C2 = {id=2, coordenadas={2.0, 1.0}}

C3 = {id=3, coordenadas={2.0, 0.0}}

C4 = {id=4, coordenadas={3.0, 0.0}}

C5 = {id=5, coordenadas={3.0, 1.0}}

Criar a População inicial: poderíamos gerar uma população inicial com, por exemplo, dois Indivíduos (trajetos) possíveis. O tamanho P da população inicial deve ser passado como parâmetro ao AG.

T1 = {0,1,2,3,4,5}

T2 = {0,2,3,1,5,4}

Determinar a Aptidão de cada Indivíduo da População: podemos calcular a Aptidão de cada Indivíduo (cf. trajetória), como sendo o inverso da soma das distâncias entre todos os pares de nós dessa trajetória, i.e.

aptidao(T1) = 1 / (dist(C0,C1) + dist(C1, C2) + … + dist(C4, C5) + dist(C5, C0))

aptidao(T2) = 1 / (dist(C0,C2) + dist(C2, C3) + … + dist(C5, C4) + dist(C4, C0))

Desta forma conseguimos obter uma lista de Indivíduos, que pode ser ordenada da maior para a menor Aptidão.

Selecionar o conjunto de Cruzamentos: selecionar o conjunto de Pais que serão usados para gerar a próxima geração, combinando as seguintes abordagens:

- Seleção por Aptidão proporcional (aka *Roulette Wheel*): definir uma probabilidade ponderada em função da Aptidão de cada Indivíduo, que meça a adequação para ser selecionado. Por exemplo, num conjunto com N indivíduos, sendo A_i a aptidão do Indivíduo i, a sua probabilidade poderia ser calculada por $P_i = A_i / \sum_{j=1}^{N} Aj$ (normalizando-se assim todas as Pi num intervalo/escala [0..1]). Tendo as P_i para todos os indivíduos, podem gerar-se então número aleatórios entre 0..1 que permitam ir escolhendo os Indivíduos em função da gama das suas probabilidades;
- Seleção por elitismo: pode-se também incluir na nova geração os Indivíduos com maior Aptidão, garantindo que persistem os mais aptos ao longo das gerações.

Gerar a próxima População: realizar os cruzamentos entre os genes dos pares de Pais por forma a criar dois novos Indivíduos para a nova geração (NB: mantendo as 3 condições do problema definidas acima). Precisamos de definir uma função de cruzamento entre Indivíduos como, por exemplo, escolher um sub-conjunto aleatório de genes do primeiro progenitor e depois preencher os restantes genes a partir do segundo progenitor pela ordem em que aparecem (e vice-versa), sem duplicar quaisquer genes do sub-conjunto selecionado do primeiro progenitor. Por exemplo, se fossemos gerar os filhos T3 e T4 através do cruzamento dos progenitores T1 e T2, poderíamos selecionar aleatoriamente de T1 e T2 as subsequências de 3 cidades {1,2,3} e {2,3,1}, que seriam então cruzadas/trocadas para obter T3 e T4 (mantendo-se os restantes elementos dos Indivíduos progenitores):

T1 = {0,**1,2,3**,4,5}

T2 = {0,**2,3,1**,5,4}

T3 = {0,**2,3,1**,4,5}

T4 = {0,**1,2,3**,5,4}

Neste caso, se tivéssemos mais Indivíduos (trajetórias) no conjunto de Cruzamentos, então poderíamos selecionar diretamente os E melhores Indivíduos (com valores maiores de Aptidão) para a nova geração (elitismo). O número E de indivíduos que passariam à geração seguinte por elitismo é um dos parâmetros passados ao AG.

Para introduzir mais diversidade nas soluções encontradas, devem também usar-se mutações nos genes dos indivíduos das novas gerações. Desta forma introduzem-se novos Indivíduos (trajetórias) que permitirão eventualmente explorar outras soluções. Cada gene, de cada indivíduo da população de cada geração, poderá, com probabilidade independente Q, ser mutado/alterado. No exemplo que temos vindo a acompanhar, se para um dado gene o valor sorteado entre 0 e 1 fosse menor que Q (probabilidade de mutação), então esse gene seria alterado do seu ID original, ID1, para um novo ID, ID2, que seria escolhido entre a gama de IDs possível. Poderíamos então proceder a uma troca entre pares de cidades (genes) do mesmo indivíduo, trocando ID1 e ID2.

Por fim, podemos repetir o ciclo de passos anterior, durante G gerações, sendo G outro dos parâmetros passados ao AG.

Será importante, no entanto, conseguirmos acompanhar a evolução dos Indivíduos (trajetórias) das diferentes gerações e respectivas distâncias, ao longo da execução do algoritmo. Para isso devemos ir recolhendo numa lista ligada, informação ao longo da execução do AG (e.g. Indivíduo mais apto (trajetória mais curta) e respectiva distância em cada geração). Desta forma poderemos no final imprimir ou gravar num ficheiro, um gráfico da evolução da solução (distância do melhor Indivíduo (trajetória/percurso) de cada geração versus número de gerações executadas).

Em resumo, os parâmetros envolvidos no algoritmo:

- P: tamanho da população inicial (número par) e para cada geração (exemplo P=20)
- Q: probabilidade de mutação dum gene (exemplo Q=0.01)
- E: número de indivíduos que passam por elitismo diretamente para a próxima geração (número par E, com E < P; exemplo E=2)
- G: número de gerações a serem executadas pelo AG (exemplo AG=50)
- N: número de cidades que representam o número de genes de cada indivíduo ou cromossoma (exemplo N=6)

Para o apoio ao processo de geração do AG, será necessário implementar estruturas que permitam o armazenamento dos dados inerentes a este processo. Como tal, é expectável que cada trabalho contenha estruturas capazes de

- População consiste num array de arrays de inteiros ou matriz (representando cada array de inteiros um Indivíduo que armazena um trajecto possível);
- Lista ligada em que cada elemento representa a informação recolhida para cada geração/iteração do AG;
- Matriz quadrada com as distâncias euclidianas, pré-calculadas, entre pares de cidades (pode conter apenas a metade inferior ou superior da matriz);

Abaixo podem encontrar o resumo dos requisitos a considerar para o projeto.

Requisitos

Pretende-se que os alunos proponham um conjunto de estruturas de dados dinâmicas (e.g. arrays dinâmicos, listas ligadas e combinações de ambos) e respectivas funções de manipulação de modo a obterem uma solução que cumpra os requisitos funcionais abaixo

indicados. As estruturas e funções associadas devem utilizar apontadores e estruturas dinâmicas para agregar e manipular os dados necessários. Deverão ser desenvolvidos, ainda, algoritmos de processamento, pesquisa e gestão da informação adequados às funcionalidades e requisitos elencados.

A avaliação será baseada na utilização de testes funcionais, i.e., cada requisito proposto será sujeito a um conjunto de testes que avaliarão o cumprimento do mesmo. Na organização de ficheiros do projeto submetido, os testes deverão estar localizados na pasta *test* que será reservada apenas para esse efeito de acordo com as indicações dadas nas aulas. Os dados de entrada para cada teste poderão ser gerados aleatoriamente (dados random) ou com recurso a dados predefinidos (dados determinísticos). NÃO deverá ser desenvolvida uma interface interactiva com o utilizador, ou seja, NÃO devem desenvolver menus ou interface gráfica.

Para cada caso de teste, será disponibilizado um ficheiro de dados de entrada e deverá ser produzido um ficheiro de dados de saída (test cases). Este ficheiro de saída será comparado com o output esperado.

A implementação deverá endereçar os seguintes requisitos funcionais:

1. Implementar as estruturas de dados dinâmicas necessárias e respetivo conjunto de funções (API) para manter e gerir a informação do **domínio do problema** (empresa Porto.Ponto). Implementar funções de gestão para cada uma das estruturas dinâmicas anteriores. Terão que existir funções para criação e redimensionamento de arrays, inserções diversas, pesquisas diversas, ordenações diversas, eliminações diversas, quer em arrays como em listas das entidades existentes:

    a. Deve existir uma **lista ligada** para todos os **clientes** contendo a sua identificação, informação de contacto e de facturação; Funções de manipulação da lista de clientes da empresa Porto.Ponto:

        i. Funções para: inserir à cabeça um cliente; inserir à cauda um cliente; inserir ordenadamente (pelo NIF ou nome) um cliente; remover um cliente; procurar um cliente (pelo NIF ou nome); ordenar a lista pelo NIF; etc.

    b. Cada cliente deverá possuir ainda um **array dinâmico** para registar todas as **viagens** já realizadas e/ou a realizar; Funções de manipulação do array de viagens de cada cliente:

        i. Funções para: criar um array dinâmico com uma dada dimensão; redimensionar o array dinâmico; inserir, editar e remover viagens e/ou dados de viagens;

    c. Para cada viagem (de cada cliente) deverá existir um **array dinâmico** de todas as **cidades** que esse cliente pretende visitar na viagem; Funções de manipulação do array de cidades de cada viagem:

        i. Funções para: criar um array dinâmico com uma dada dimensão; redimensionar o array dinâmico; inserir, editar e remover cidades e/ou

dados das cidades a visitar na viagem; pesquisar cidades em função do nome;

d. Cada **cidade** pode ter informação relevante associada (e.g. nome, descrição, lista de pontos de interesse (PoI)); Funções de manipulação da informação de cada cidade:

    i. Funções para pesquisar e editar informação presente na descrição da cidade;

    ii. Funções para: criar a lista de PoI de uma cidade; inserir, remover e pesquisar PoI de uma cidade;

e. Funções de consulta de histórico de viagens por cliente e permitir a pesquisa dentro de cada viagem por parâmetros específicos: e.g. viagem que passou no local/cidade X, viagem onde realizou atividade/visitou PoI Z;

f. Implementar funções de leitura e escrita de ficheiros de texto e binário das diversas entidades: clientes e respectivas viagens, cidades e respectivos PoI;

g. Implementar funções para gerar relatórios, em ficheiros de texto, sobre clientes e respectivas viagens;

2. Implementar as estruturas de dados dinâmicas necessárias e respetivo conjunto de funções (API) para manter e gerir a informação da execução do **algoritmo genético** (AG). Terão que existir funções para criação e redimensionamento de arrays, inserções, pesquisas, ordenações e eliminações diversas, quer em arrays, listas e matrizes das entidades existentes; Por exemplo:

    a. Deve existir uma estrutura com a informação da **parametrização** do AG, juntamente com uma **lista ligada** para registar informação sobre cada geração/iteração da execução do AG; Funções de gestão da lista de gerações do AG:

        i. Funções para inserir à cauda ou cabeça um nó de uma nova geração de execução do AG; pesquisar uma geração pelo ID da iteração; pesquisar gerações com aptidão superior a um dado valor; etc.;

        ii. Função para registar num nó correspondente a uma Geração, a informação dessa geração, i.e., os dois melhores Indivíduos (trajetos), as respectivas aptidões;

    b. Para cada geração devem existir dois array de arrays (matrizes) correspondentes à população de Indivíduos atual (progenitores) e à População de Indivíduos da próxima geração (herdeiros); estas matrizes podem ir sendo alternadas entre si, i.e., a população atual dará lugar à população seguinte, alternando as matrizes na próxima reprodução; Funções

de manipulação das matrizes de Indivíduos (trajetos) das populações progenitora e herdeira de cada geração:

i. Funções para efectuar a gestão das duas matrizes de Populações (e.g. criar, libertar, etc.), tendo em conta os parâmetros seleccionados para mutação, elitismo e cruzamento;

ii. Função para geração aleatória de uma população inicial;

iii. Função para quantificar a aptidão de cada indivíduo numa dada população;

iv. Funções para efetuar mutações, elitismo e cruzamentos, usando as duas matrizes das populações dos progenitores e herdeiros; estas funções poderão utilizar funções auxiliares para seleção de subconjuntos de genes de um indivíduo, mantendo as restrições do problema;

c. Generalizar a implementação do AG utilizando apontadores para funções (e.g. cruzamentos, mutações, adaptadores).

Cotação dos Requisitos em AED1 e LP1

Req	**R1.a**	**R1.b**	**R1.c**	**R1.d**	**R1.e**	**R1.f**	**R1.g**	**R2.a**	**R2.b**	**R2.c**
AED1	2	2	2	2	1	2.5	1	3	4	0.5
LP1	2	2	2	2	1	2	1	3	4	1

(NB: cotação 0-20)

Processo de Submissão

A aplicação final (código fonte) deve estar depurada de todos os erros de sintaxe e de acordo com os requisitos funcionais pedidos. Só serão considerados os projetos de software que não contenham erros de sintaxe e que implementam as funcionalidades pedidas total ou parcialmente.

A documentação, em html ou pdf, juntamente com todo o código-fonte desenvolvido, deve ser submetida num ficheiro zip/rar (project.zip) na plataforma de elearning canvas (ufp.instructure.com).

Código Fonte e Documentação a entregar

As estruturas de dados e os algoritmos especificados devem ser implementados em linguagem C, juntamente com os comentários apropriados, inseridos no código fonte desenvolvido, de modo a que facilitem a compreensão do mesmo. Todas as funções devem estar anotadas em formato doxygen (www.doxygen.nl) incluindo: uma breve explicação dos

algoritmos implementados; uma menção ao desempenho dos algoritmos (quando aplicável) assim como dos testes efetuados/implementados.

As principais estruturas de dados e variáveis devem também estar anotadas neste formato. Deverão usar o software doxygen para gerar a documentação com base nos comentários.

Os alunos deverão entregar também um ficheiro de texto no formato doxygen (.dox), descrevendo explicitamente quais foram: i) os requisitos implementados, ii) parcialmente implementados e iii) não implementados. Devem mencionar sempre o número do requisito de acordo com a numeração utilizada neste documento de especificação:

- Funcionalidades implementadas: devem identificar todas as funções desenvolvidas para assegurar os requisitos funcionais solicitados.
- Funcionalidades parcialmente implementadas: devem identificar as funcionalidades parcialmente implementadas e apontar as dificuldades na sua conclusão.
- Funcionalidades não implementadas: devem identificar as funcionalidades não implementadas e as dificuldades que impediram o seu desenvolvimento.

Bibliografia

[1] Introduction to Genetic Algorithms — Including Example Code | by Vijini Mallawaarachchi

[2] Evolution of a salesman: A complete genetic algorithm tutorial for Python

[3] An Introduction to Genetic Algorithms