Atílio Antônio Dadalto
Ícaro Madalena do Nascimento

Auxílio ao resgate de vítimas por robôs em ambientes acidentados

Vitória
2019

Atílio Antônio Dadalto
Ícaro Madalena do Nascimento

Auxílio ao resgate de vítimas por robôs em ambientes acidentados

Relatório apresentado como requisito parcial para aprovação na disciplina de Programação I, pela Universidade Federal do Espírito Santo.

Universidade Federal do Espírito Santo

Departamento de Informática

Vitória

2019

Sumário

INTRODUÇÃO

Este trabalho tem por objetivo documentar a estruturação de um projeto de auxílio à tomada de decisões dado um cenário hipotético de um ambiente acidentado. Neste, dados serão extraídos por robôs-célula que investigam o local, que serão processados de forma a aprimorar a atividade de resgate das vítimas que se encontram no ambiente. O plano de testes, presentes no Apêndice A, relata a confiabilidade das funções desenvolvidas.

No Capítulo 1, por sua vez, é abordada a forma como foram planejadas as resoluções de cada problema apresentado pelo contexto, assim como a implementação, de forma geral, das funções mais complexas. Por fim, a conclusão realiza uma breve avaliação da solução final, também pontuando comentários adicionais.

As referências para os métodos de testes, bem como paradigmas aplicativo e recursivo, que serão implementados podem ser encontrados nas notas de aula utilizadas pela disciplina, encontradas na página desta em junho de 2019. O projeto foi desenvolvido tendo como base a versão 3.6 ou 3.7 do Python.

1 Planejamento e implementação

O planejamento das soluções foi feito tendo em mente o objetivo final de um problema. As soluções foram subsequentemente divididas em passos cada vez menores e mais discretizados, possibilitando a abordagem de dividir para conquistar.

A implementação, por sua vez, conta com vários usos de expressões lambda e também da função `map`. O paradigma recursivo foi usado em situações que julgou-se mais simples ou intuitivo sua implementação, como por exemplo na ordenação ou na remoção de elementos duplicados em uma lista (mais sobre isso nas Seções 1.2 e 1.3). Para maior entendimento das soluções, é claro, é necessário visualizar os códigos, que aqui foram omitidos. Todo o projeto busca seguir as convenções da PEP-8.

1.1 Problema A

a) Calcular a distância percorrida por um determinado robô ao longo do processo de resgate das vítimas. Considere que a distância total percorrida deve ser calculada como a soma de todas as distâncias entre os pontos de passagem do robô.

COMPREENSÃO DO PROBLEMA E PLANEJAMENTO: deve-se extrair a tupla de um robô da lista de entrada, de acordo com o identificador dado. Em seguida, calcular a distância total que o robô percorreu e retornar esse valor.

Sendo assim, podemos criar as seguintes funções para solucionar o problema:

1. Função para cálculo de distância no plano cartesiano
2. Função para extrair todas as tuplas de um mesmo robô da lista
3. Função para extrair todos os pontos percorridos por um mesmo robô
4. Função para calcular a distância total de uma lista de pontos

A primeira, `dist_euclid`, é trivial e foi implementada diversas vezes durante o curso;

Na segunda, `lista_tuplas_robo_id`, foi utilizada compreensão de lista com condicional para formar uma lista de tuplas;

Na terceira, `pega_pontos_robo`, foi aplicada, a todos os elementos da lista de robôs, a função que retorna o local do robô dada uma tupla de robô, por meio da

função `map`. A partir daí, adiciona-se a origem à lista de pontos percorridos, pois todos os robôs partem da origem.

Por fim, a quarta, `distancia_total`, é utilizada a recursão para calcular a distância de um ponto até outro, partindo do primeiro ponto, dada uma lista de pontos como entrada.

Com essas quatro funções, temos o suficiente para solucionar o Problema A; a função que propriamente resolve essa questão é a `distancia_total_robo`, ao extrair todos os pontos de um robô dado seu id e a lista de robôs e calcular a distância do primeiro ponto até o último, passando por todos os pontos no meio.

1.2 Problema B

b) Determine qual dos robôs apresenta o seu último ponto de passagem no terreno de busca que possui a maior distância em relação à origem. Exiba o caminho percorrido pelo robô e o tempo total do percurso;

COMPREENSÃO DO PROBLEMA E PLANEJAMENTO: deve-se extrair, da lista de entrada, o último ponto de passagem de todos os robôs. Com isso, determinar qual robô possui ponto mais longe da origem e então imprimir caminho percorrido pelo robô, além de determinar tempo do percurso.

Sendo assim, podemos criar as seguintes funções para solucionar o problema:

1. Função para obter uma lista com os últimos pontos de todos os robôs da lista de entrada
2. Função para calcular a distância da origem a todos esses pontos, retornando uma lista
3. Função para buscar os robôs que obtiveram a maior distância
4. Função para determinar tempo de percurso de um robô

A primeira função, `ultimos_pontos_robos`, é muito simples, bastando aplicar a função `map` com uma função que retorne o último elemento de uma lista e a lista de tuplas de robôs.

A segunda, `dist_origem_ultimo_ponto`, também é simples e utiliza a função de cálculo de distância da origem até um ponto com todos os pontos obtidos pela primeira função, por meio da função `map`.

A terceira, `indices_ids_mais_distantes`, após calcular as distâncias até os últimos pontos de todos os robôs, calcula os índices dos robôs que tiveram maior distância (caso exista mais de um). Esses índices podem ser utilizados para acessar a lista de ids que a função `ids_robos` retorna, função essa que retorna uma lista com todos os ids dos robôs, sem repetição.

Finalmente, a quarta, `tempo_percurso`, simplesmente retorna o instante da última ocorrência de um robô, já que isso representa o tempo total de percurso atual de um robô.

Sendo assim, teremos os ids dos robôs mais distantes, os percursos destes — através da função `pega_pontos_robo` do Problema A com os ids supracitados — e o tempo de percurso.

1.3 Problema C

c) Exiba os caminhos percorridos por todos os robôs que entraram no terreno de busca, ordenados crescentemente pela distância total percorrida;

COMPREENSÃO DO PROBLEMA E PLANEJAMENTO: deve-se calcular as distâncias percorridas por todos os robôs, adicionando-as em uma tupla que será ordenada a partir das distâncias, impressa na tela e então retornada.

Sendo assim, podemos criar as seguintes funções para solucionar o problema:

1. Função para juntar id, distância total e caminho percorrido de todos os robôs em uma lista de tuplas
2. Função para ordenar essa lista pelas distâncias nas tuplas

A primeira função simplesmente percorre as listas de ids, distâncias totais e caminhos, juntando elemento a elemento em uma tupla e concatenando em uma lista, de forma recursiva. A lista de distâncias totais é calculada utilizando a função `distancia_total` do Problema A nas listas de pontos obtidas a partir da lista de robôs. Nas seções anteriores, discorre-se sobre a obtenção da lista de caminhos percorridos pelos robôs e da lista de IDs.

Para a segunda, que é a principal função do problema, foi utilizado o método de ordenação *merge sort*, em sua variação recursiva. Além disso, o *merge sort* implementado recebe uma lista de tuplas, das quais os segundos elementos são utilizados

como referência para a ordenação da lista. O segundo elemento da tupla, no caso deste problema, será a distância total do robô em questão.

Posto isto, basta chamar a função de ordenação com a saída da primeira função e o Problema C estará solucionado.

1.4 Problema D

d) Forneça a identidade do(s) robô(s) que conseguiu(ram) informar o maior número de vítimas (considerando que não há duplicação de identificação de vítima por um mesmo robô).

COMPREENSÃO DO PROBLEMA E PLANEJAMENTO: deve-se calcular o número de vítimas que cada robô avistou e identificar o robô que avistou o maior número. Retornar mais de um robô caso o maior número seja repetido.

Sendo assim, podemos criar as seguintes funções para solucionar o problema:

1. Função para extrair o número de vítimas vistas em cada ponto passado por um robô
2. Função para somar elementos de uma lista
3. Função que aplique essas duas últimas funções para todos os robôs da lista de entrada
4. Função para extrair o maior número da lista obtida pelo último item
5. Função para obter IDs dos robôs que obtiveram o máximo calculado anteriormente

Para a primeira função, `lista_vitimas_robo`, podemos lançar mão da compreensão de lista mais uma vez para, em cada ocorrência (elemento) da lista de robôs, extrairmos o número de vítimas avistadas nessa ocorrência se o robô da tupla em questão tiver o id de entrada.

A segunda, `soma_lista`, é trivial e foi implementada diversas vezes durante o curso;

A terceira, `total_vitimas_robo`, como já mencionado, apenas utiliza a primeira função para obter a lista com vítimas avistadas por um robô e, com a segunda função, soma a lista para calcular o número total de vítimas avistadas por um robô.

A quarta, `max_lista`, utiliza a função `reduce` com a função `maior` (que compara dois elementos e retorna o maior) e a lista de entrada para comparar elemento por elemento na lista, retornando o maior.

Agora, com a terceira, podemos utilizar a função `map` para extrair todas as vítimas de cada robô da lista de robôs, com auxílio da função `ids_robos` e uma função anônima. Teremos então uma lista com o total de vítimas de cada robô. Utilizando a função `indices_maximos` do Problema B, que retornará os índices de todos os elementos que coincidam com o maior elemento, podemos acessar a lista de ids com esses índices e retornar os ids dos robôs que avistaram o maior número de vítimas (se houver mais de um), solucionando, portanto, o Problema D.

CONCLUSÃO

Pelo estudo realizado neste trabalho, fica evidente como podemos construir diversas soluções tendo como ponto de partida o paradigma funcional. Utilizando a modularização e o reuso de funções através de importações, foi possível conceber uma central de processamento fictícia adaptada para o contexto proposto ao projeto. Ademais, a compreensão de problemas e planejamento de soluções permitiram que o desenvolvimento das resoluções fosse muito mais focado e centrado.

Os conceitos de paradigma recursivo e aplicativo, além da compreensão de lista foram fundamentais para a execução das soluções idealizadas. A utilização dessas abstrações permitiu uma solução mais limpa e sucinta do que se fosse empregado o paradigma procedural, por exemplo. O uso de recursão, no entanto, pode causar *stack overflow* e, embora a implementação do *merge sort* seja significativamente mais simples neste paradigma, um grande volume de dados poderia causar a suspensão da execução do programa; o próprio Python limita o nível de recursões a 1000 chamadas empilhadas por padrão, embora seja possível aumentar esse número.

APÊNDICE A – Planos de testes

Este apêndice serve como repositório para os planos testes das funções mais importantes implementadas no projeto. Algumas funções assumem certos valores de entrada, de forma a afunilar as possíveis entradas, dado que os valores a serem recebidos podem variar enormemente e que o projeto conta com muitas funções; no entanto, nos contemos a assumir entradas de maneira muito razoável, e, além disso, todos os casos básicos foram tratados devidamente dentro das funções.

Para melhor exemplificação e legibilidade dos testes das funções abaixo, usaremos a lista de dados de entrada ilustrada a seguir:

```
listaRobos = [
    ('robo3', 1, (7, 7), 3), ('robo4', 2, (7, 5), 2),
    ('robo3', 3, (5, 4), 3), ('robo3', 4, (8, 1), 4),
    ('robo4', 5, (4, 5), 3), ('robo5', 6, (7, 7), 4)
]
```

A.1 `distancia_total()`

A função calcula a distância total entre uma lista de pontos somando a distância entre um ponto e o próximo até o fim da lista.

lista	resultado esperado
[]	None
lista com um ponto	None
[p1, p2, p3]	distância de p1 até p2 + distância de p2 até p3

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	None	None
[(4,4)]	None	None
[(0,0),(3,4),(5,6)]	7.82842712474619	7.82842712474619

A.2 `distancia_total_robo()`

A função calcula a distância total percorrida por um robô, ou seja, a soma das distâncias de todos os pontos percorridos por ele.

lista	resultado esperado
[]	None
[(a,(p1)),(a,(p2))], 'a'	distância de (0,0) até p1 + distância de p1 até p2

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	None	None
[('robo1',2,(4,4),5)], 'robo2'	None	None
listaRobos, 'robo3'	17.74768689919494	17.74768689919494

A.3 `indices_maximos()`

A função recebe uma lista numérica e retorna uma lista com os índices de todas as ocorrências do valor máximo na lista.

lista	resultado esperado
[]	[]
[a, b, c, d]	índices das ocorrências do maior elemento da lista

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	[]	[]
[5]	[0]	[0]
[1,2,3,10,1,10]	[3,5]	[3,5]

A.4 `imprime_robos_mais_distantes()`

A função recebe uma lista de robôs e imprime os robôs que encontram-se mais distantes da origem, seus percursos e tempos de percurso

lista	resultado esperado
[]	None
[('b',(10,10),1)]	['b'],[(0,0),(10,10)],[1]
[('a',(5,4),1),('b',(10,10),1)]	['b'],[(0,0),(10,10)],[1]

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	None	None
[('robo',1,(10,10),3)]	[('robo',((0,0),(10,10)),1]	[('robo',((0,0),(10,10)),1]
[('robo',1,(10,10),3),('robo2',1,(2,1),3)]	[('robo',((0,0),(10,10)),3]	[('robo',((0,0),(10,10)),3]

A.5 `caminhos_robos_crescente()`

A função retorna uma lista de tuplas para cada robô, ordenadas em ordem crescente pela distância total percorrida pelos robôs.

Entradas	resultado esperado
[]	None
[(a),(b)]	soma do caminho percorrido a, soma do caminho b

Definindo entradas (saídas arredondadas para melhor visualização):

lista	resultado esperado
[]	None
[('robo1',2,(4,4),5)]	[('robo1', 5.6, [(0, 0), (4, 4)])]
[('robo1',2,(4,4),5),('robo3',7,(5,5),2)]	[('robo1', 5.6, [(0, 0), (4, 4)]), ('robo3', 7, [(0, 0), (5, 5)])]

resultado obtido
None
[('robo1', 5.6, [(0, 0), (4, 4)])]
[('robo1', 5.6, [(0, 0), (4, 4)]), ('robo3', 7, [(0, 0), (5, 5)])]

A.6 `merge_ordenada_tupla()`

Dadas duas listas ordenadas pelo segundo elemento da tupla, junta-as de forma ordenada.

lista	resultado esperado
[]	[]
[],[(a,b)]	[(a,b)]
[(c,d)],[]	[(c,d)]
[(a,b)],[(c,d)]	[(a,b),(c,d)]
[(a,b),(c,d)],[(b,c),(d,e)]	[(a,b),(b,c)(c,d),(d,e)]

Definindo entradas:

lista	resultado esperado	resultado obtido
[],[]	[]	[]
[(1,2)],[(3,4)]	[(1,2),(3,4)]	[(1,2),(3,4)]
[(1,2),(3,4)],[(5,6),(7,8)]	[(1,2),(3,4),(5,6),(7,8)]	[(1,2),(3,4)],[(5,6),(7,8)]

A.7 `merge_sort_tupla()`

Ordena uma lista de tuplas através do método *merge sort*, tendo como referência o segundo elemento das tuplas.

lista	resultado esperado
[]	[]
[(a)]	[(a)]
[(a,d),(c,b),(b,f)]	[(c,b),(a,d),(b,f)]

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	[]	[]
[(1,2)]	[(1,2)]	[(1,2)]
[(1,4),(2,3)]	[(2,3),(1,4)]	[(2,3),(1,4)]

A.8 `ids_mais_vitimas()`

A função retorna o(s) id(s) do(s) robô(s) que avistaram o maior número de vítimas.

lista	resultado esperado
[]	None
um robô	o id do único robô
mais de um robô, um robô com mais vítimas avistadas	id do robô com mais vítimas
mais de um robô, mais de um robô com mais vítimas	ids dos robôs com mais vítimas

Definindo entradas:

lista	resultado esperado	resultado obtido
[]	None	None
[('robo1',1,(1,1),2)]	['robo1']	['robo1']
[('robo1',1,(1,1),2),('robo2',5,(3,2),5)]	['robo2']	['robo2']
[('robo1',1,(1,1),2),('robo2',5,(3,2),2)]	['robo1', 'robo2']	['robo1', 'robo2']