**Nome: RA:**

**Luciano de Araújo Júnior 210017894**

**Matheus Caldas 210017233**

* **Sobre:** Documentação do Algoritmo Guloso e Backtracking
* **Linguagem adotada:** Python
* **Link para código fonte / Vídeos:**

https://github.com/jrllucianojunior/Algoritmo-Guloso-e-Backtracking

**Documentação do Algoritmo Guloso**

Este código implementa um algoritmo guloso para percorrer uma matriz tridimensional, com duas coordenadas, inicial e final, geradas aleatoriamente. O algoritmo coleta valores e demonstra o caminho percorrido em um gráfico tridimensional. Abaixo está a descrição detalhada de cada função e parte do código:

**Bibliotecas Utilizadas**

- `numpy`: Para manipulação eficiente de arrays e matrizes multidimensionais.

- `matplotlib.pyplot`: Para plotagem de gráficos.

- `mpl\_toolkits.mplot3d.Axes3D`: Para criar gráficos 3D.

**Funções Principais:**

**Função `inicializa\_plot(matrix, start, end)`**

Inicializa o plot do gráfico 3D com pontos iniciais e finais marcados em cores específicas.

Parâmetros:

- `matrix`: Matriz tridimensional que será percorrida.

- `start`: Coordenada inicial na matriz.

- `end`: Coordenada final na matriz.

Variáveis:

-`fig`: Referência à figura do gráfico 3D.

-`ax`: Referência aos eixos do gráfico 3D.

-`point`: Representa um ponto na matriz durante a iteração para plotagem.

-`x, y, z`: Coordenadas x, y e z de um ponto na matriz durante a iteração para plotagem.

- Retorno: Retorna a figura e os eixos do gráfico.

Comportamento: Cria uma nova figura para plotagem, adiciona o ponto inicial e final no gráfico 3D em verde e amarelo respectivamente, plota os demais pontos da matriz em azul e mostra o plot do gráfico 3D.

**Função `plot\_caminho(ax, caminhos\_coletados, block=False)`**

Plota o caminho coletado no gráfico 3D.

Parâmetros:

- `ax`: Eixos do gráfico.

- `caminhos\_coletados`: Lista de coordenadas dos pontos coletados.

- `block`: Booleano que indica se a execução do código deve ser bloqueada após o plot final.

Comportamento: Plota o caminho coletado no gráfico 3D, atualiza o plot com cada nova posição coletada e se block=True, mostra o plot final e bloqueia a execução do código.

**Função `Coletar(matrix, start, end)`**

Realiza a coleta de valores na matriz, movendo-se de uma posição inicial para uma posição final, seguindo um algoritmo guloso.

Parâmetros:

- `matrix`: Matriz tridimensional que será percorrida.

- `start`: Coordenada inicial na matriz.

- `end`: Coordenada final na matriz.

Variáveis:

- `matrix`: Representa a matriz tridimensional preenchida com valores aleatórios.

- `cordenada\_inicial`: Coordenada inicial gerada aleatoriamente.

- `cordenada\_final`: Coordenada final gerada aleatoriamente.

- `valores\_coletados`: Lista de coordenadas dos pontos coletados durante o percurso na matriz.

- `total\_coletado`: Total dos valores coletados durante o percurso na matriz.

- `posicao\_atual`: Representa a posição atual durante o percurso na matriz.

- `fig`: Referência à figura do gráfico 3D.

- `ax`: Referência aos eixos do gráfico 3D.

- `direcoes`: Lista de direções possíveis de movimento na matriz.

- `posicoes\_possiveis`: Lista para armazenar posições válidas para o próximo movimento.

-`prox\_pos`: Armazena a próxima posição calculada durante a iteração pelas direções possíveis.

- Retorno: Retorna o total coletado, a lista de posições visitadas e a posição final.

Comportamento: Inicializa o processo de coleta na matriz. Move-se da posição inicial para a posição final seguindo um algoritmo guloso. Coleta valores na matriz durante o percurso. Visualiza o caminho percorrido no gráfico 3D e informa ao usuário as posições visitadas e o total coletado.

**Funções auxiliares da Função Coletar:**

**Função limites(pos)**

Esta função auxiliar verifica se uma posição está dentro dos limites da matriz.

Parâmetros:

-`pos`: Uma tupla representando a posição a ser verificada.

Variáveis:

-`x, y, z`: Representam as coordenadas x, y e z da posição a ser verificada.

-Retorno : Retorna True se a posição estiver dentro dos limites da matriz, caso contrário retorna False.

**Função coleta\_valor(pos)**

Esta função auxiliar coleta o valor de uma posição na matriz e atualiza o total coletado.

Parâmetros:

-`pos`: Uma tupla representando a posição da qual o valor será coletado.

Variáveis:

-`x, y, z:` Representam as coordenadas x, y e z da posição da qual o valor será coletado.

-`valor`: Armazena o valor na posição especificada na matriz.

**Função eh\_extremo(pos)**

Esta função verifica se uma posição é um extremo da matriz.

Parâmetros:

-`pos`: Uma tupla representando a posição a ser verificada.

-Retorno: Retorna True se a posição for um extremo da matriz, caso contrário retorna False.

**Função pode\_escapar(pos)**

Esta função verifica se é possível escapar de uma posição sem ficar preso.

Parâmetros:

-`pos`: Uma tupla representando a posição a ser verificada.

-Retorno: Retorna True se houver pelo menos uma direção válida para escapar da posição sem ficar preso, caso contrário retorna False.

**Execução do Código**

- Configura a matriz e os pontos de início e fim.

- Executa a coleta de valores e visualiza o caminho no gráfico 3D, informando o usuário no terminal as posições visitadas e o total coletado.

- Imprime informações finais sobre a coleta, incluindo a posição final, as coordenadas iniciais e finais, e o total coletado.

Para fins de teste, o algoritmo foi modificado, para em vez de gerar valores aleatórios nas posições, foram gerados valores seguindo um caminho específico, onde na posição [0,0,0] ele começa com 100, e vai decrementando no decorrer da linha, e ao final da linha ele inicia na mesma posição X, incrementa 1 em y, e continua decrementando x. Somente foi preenchido a profundidade Z= 0, para fins de teste.

matrix = np.zeros((10, 10, 10), *dtype*=int)  *# Inicializa a matriz com zeros*

ini = 100

for y in range(10):  *# Itera sobre as linhas em z=0*

    if y % 2 == 0:

        for x in range(10):

            matrix[x, y, 0] = ini

            ini -= 1

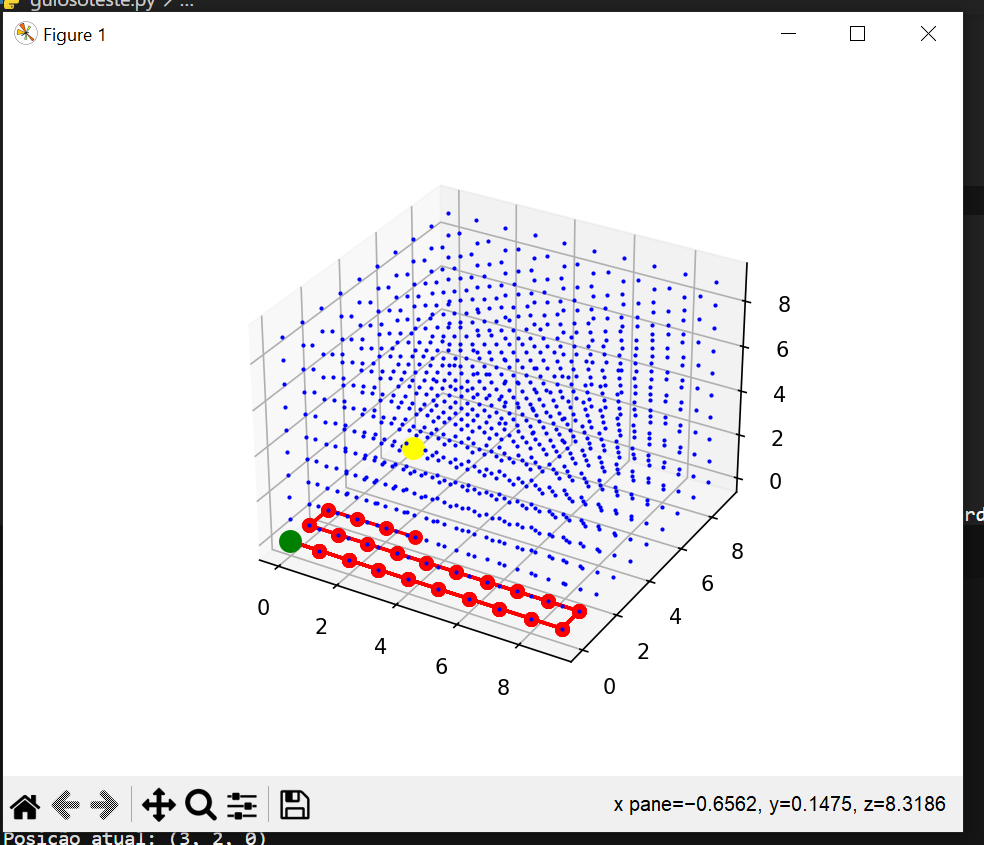
    else:  *# Linhas ímpares*

        for x in range(9, -1, -1):  *# Começa do final para o início*

            matrix[x, y, 0] = ini

            ini -= 1

Nos testes o programa funcionou conforme esperado, seguindo o caminho dos maiores valores:



**Documentação do Algoritmo Backtracking**

**(**Jogo do Cubo**)**

Este código implementa um jogo onde um cubo azul se move aleatoriamente em uma matriz tridimensional, enquanto o jogador tenta alcançar um cubo amarelo aleatório para marcar pontos. O jogo continua até que o jogador atinja uma pontuação máxima ou encontre o cubo vermelho. Abaixo está a descrição detalhada de cada função e parte do código:

**Bibliotecas Utilizadas**

* matplotlib.pyplot: Para plotagem de gráficos.
* mpl\_toolkits.mplot3d.Axes3D: Para criar gráficos 3D.
* numpy: Para manipulação eficiente de arrays e matrizes multidimensionais.
* threading: Para executar múltiplas tarefas simultaneamente.
* time: Para gerenciar temporizações.
* random: Para geração de números aleatórios.
* tkinter: Para criar a interface gráfica.

**Variáveis Globais:**

* TAMANHO\_JOGO: Tamanho da matriz tridimensional do jogo.
* PONTUACAO\_MAXIMA: Pontuação máxima que o jogador pode atingir.
* TAMANHO\_ALEATORIO: Intervalo de valores aleatórios para preencher a matriz.
* pontuacao: Pontuação atual do jogador.
* matriz: Matriz tridimensional que representa o jogo.
* posicao\_cubo: Lista que armazena a posição atual do cubo azul.
* posicao\_cubo\_aleatorio: Lista que armazena a posição do cubo amarelo aleatório.
* posicoes\_visitadas: Lista que armazena as posições visitadas pelo cubo azul.

**Funções Principais:**

Função desenhar\_cubos()

Atualiza a plotagem do jogo com a posição dos cubos. Comportamento: Limpa o gráfico atual, plota os cubos azul e amarelo, as posições visitadas pelo cubo azul e uma linha vermelha conectando essas posições.

Função movimento\_valido(nova\_posicao)

Verifica se uma nova posição é válida dentro dos limites do jogo. Parâmetros:

* nova\_posicao: Lista contendo as coordenadas da nova posição. Retorno: Retorna True se a nova posição for válida, caso contrário retorna False.

Função gerar\_movimento\_aleatorio()

Gera um movimento aleatório para o cubo azul. Comportamento: Gera uma nova posição aleatória para o cubo azul e verifica se é válida. Atualiza a pontuação e a plotagem do jogo.

Função verificar\_se\_cubo\_encontrado()

Verifica se o jogo terminou por atingir a pontuação máxima, perder pontuação ou encontrar o cubo vermelho. Comportamento: Exibe uma mensagem informando o resultado do jogo e fecha a janela do jogo.

Função gerar\_movimentos()

Gera movimentos aleatórios para o cubo azul até que o jogo termine. Comportamento: Executa continuamente a função gerar\_movimento\_aleatorio() e verificar\_se\_cubo\_encontrado() com um intervalo de tempo.

Função placar()

Exibe a pontuação atual do jogador em uma janela separada. Comportamento: Atualiza continuamente a pontuação exibida na janela com um intervalo de tempo.

Função inicializa\_plot(matrix, start, end)

Inicializa o plot do jogo com os cubos azul, amarelo e as posições na matriz. Parâmetros:

* matrix: Matriz tridimensional do jogo.
* start: Coordenada inicial do cubo azul.
* end: Coordenada do cubo amarelo.

Comportamento: Plota os cubos azul e amarelo, além das posições na matriz, em um gráfico 3D.

**Execução do Código**

* Inicializa a matriz tridimensional com valores aleatórios.
* Inicializa a posição dos cubos azul e amarelo.
* Inicia as threads para gerar movimentos aleatórios e exibir o placar.
* Inicializa a plotagem do jogo.
* Exibe a plotagem do jogo.

Este jogo proporciona uma experiência interativa onde o jogador controla o cubo azul em busca do cubo amarelo, enquanto tenta alcançar a pontuação máxima.

Em baixo temos algumas telas:

Pontuações:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Cubo e os pontos pelo caminho percorrido:

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

Janela de notificação final:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente