Pseudocódigo e explicação

Explicação do Problema

No problema **UVA 116 – Unidirectional TSP**, o objetivo é encontrar um caminho com custo mínimo em uma matriz de dimensões $m \times n$. O caminho deve começar na primeira coluna e terminar na última, passando por células adjacentes. Em cada passo, é permitido mover para:

- A célula imediatamente à direita na linha acima (com wrap-around: da primeira para a última linha).
- A célula imediatamente à direita na mesma linha.
- A célula imediatamente à direita na linha abaixo (com wrap-around: da última para a primeira linha).

Além de minimizar o custo total (a soma dos valores das células visitadas), se houver caminhos com o mesmo custo, deve-se escolher aquele que seja **lexicograficamente menor,** ou seja, a sequência de índices (convertida para 1-index) do caminho deve ser a menor possível.

Explicação da Solução

A solução para o problema utiliza **programação dinâmica** e segue estes passos principais:

1. **Leitura e Processamento da Entrada:**

São lidas as dimensões da matriz e seus valores do arquivo input.txt, que representam os custos de cada célula.

2. **Inicialização:**

Na última coluna da matriz, o custo mínimo para cada célula é o próprio valor contido nela, pois não há mais movimentos possíveis a partir daí.

3. **Construção da Tabela de DP:**

A partir da coluna penúltima até a primeira, para cada célula, calcula-se o custo mínimo somando o valor atual com o menor custo entre as três células vizinhas na coluna seguinte.

Para garantir o critério lexicográfico, os vizinhos (linha acima, mesma linha e linha abaixo) são considerados em ordem crescente, utilizando o wrap-around.

4. **Reconstrução do Caminho:**

Após o preenchimento da tabela de DP, seleciona-se a linha inicial na primeira coluna que possui o menor custo total. A partir dela, utiliza-se um vetor auxiliar para reconstruir o caminho, célula a célula, convertendo os índices para o padrão 1-index.

5. **Saída:**

Ao final, o algoritmo imprime o caminho encontrado (lista de índices das linhas) e o custo total mínimo.

Como executar

Basta executar: python3 solution.py, caso queira testar outro input, basta mudar no arquivo em questão.

Pseudocódigo

```
Ler m, n  // dimensões da matriz
Ler a matriz de custos

// Inicialização: custo da última coluna
Para cada linha i:
    dp[i][n-1] = matriz[i][n-1]

// Preencher a tabela dp de trás para frente
Para j de n-2 até 0:
    Para cada linha i:
```

```
        vizinhos = [ (i-1) mod m, i, (i+1) mod m ]
        Ordenar vizinhos em ordem crescente
        melhor_custo = INF
        Para cada v em vizinhos:
            Se dp[v][j+1] < melhor_custo:
                melhor_custo = dp[v][j+1]
                next[i][j] = v
        dp[i][j] = matriz[i][j] + melhor_custo

// Escolher a linha inicial que minimiza dp[i][0] (critério lexicográfico)
Selecionar linha_inicial com o menor dp[i][0]

// Reconstrução do caminho
Inicializar caminho vazio
linha_atual = linha_inicial
Para j de 0 até n-1:
    Adicionar (linha_atual + 1) em caminho  // converter para 1-index
    Se j < n-1:
        linha_atual = next[linha_atual][j]

// Imprimir o caminho e o custo total (dp[linha_inicial][0])
```

Referências

- https://github.com/marioyc/Online-Judge-Solutions/blob/master/UVA/I/116 - Unidirectional TSP.cpp
- https://github.com/Diusrex/UVA-Solutions/blob/master/116 Unidirectional TSP.cpp
- https://vjudge.net/problem/uva-116
- https://codeforces.com/blog/entry/76867?f0a28=1
- https://algorithmist.com/wiki/UVa_116_-_Unidirectional_TSP