Pseudocódigo e explicação

Este documento apresenta uma solução para o problema UVA 10160 Servicing Stations.

O problema consiste em determinar o menor conjunto de vértices (ou “estações”) em um grafo tal que, ao instalar uma estação em um vértice, ela “cobre” aquele vértice e todos os seus vizinhos. Ou seja, precisamos selecionar alguns vértices de forma que, juntos, garantam que todos os vértices do grafo estejam cobertos.

Explicação da solução

1. **Representação da Cobertura com Bitmasks:**

Cada vértice possui uma “máscara” que representa a si mesmo e todos os vértices aos quais está conectado (seus vizinhos). Essa máscara é um número inteiro em que cada bit ligado indica que aquele vértice está coberto.

2. **Backtracking (DFS) com Poda:**

O algoritmo utiliza uma busca recursiva (backtracking) para testar diferentes combinações de vértices (estações).

Em cada passo, ele combina a cobertura dos vértices escolhidos até o momento (através da operação OR entre as máscaras) e verifica se essa união cobre todos os vértices do grafo.

A busca é feita de maneira progressiva, começando com uma quantidade pequena de estações e aumentando até encontrar a combinação mínima que cobre todo o grafo.

São aplicadas técnicas de poda para evitar explorar caminhos que, mesmo na melhor das hipóteses, não conseguirão cobrir todos os vértices.

3. **Resultado:**

Assim que o algoritmo encontra uma combinação de vértices que cobre todo o grafo, ele retorna o número mínimo de estações necessárias.

Como executar

Rode o script solution.py com:

```
python3 solution.py
```

O input está no arquivo input.txt, caso queira testar um diferente basta alterar esse arquivo e rodar novamente

Pseudocódigo

```
INÍCIO

  # Abrir e ler o arquivo de entrada
  Abrir o arquivo "input.txt" para leitura
  Ler todas as linhas não vazias e armazenar em uma lista chamada LINES
  Definir índice IDX ← 0

  # Processar cada caso de teste
  Enquanto IDX < Tamanho(LINES) faça:

    # Ler a linha de cabeçalho do caso (contém n e m)
    Dividir LINES[IDX] em PARTS
    Incrementar IDX
    Converter PARTS[0] para inteiro → n
    Converter PARTS[1] para inteiro → m

    # Verificar condição de parada (caso n + m == 0)
    Se (n + m == 0) então
      Encerrar o processamento

    # Inicializar vetor de cobertura ST para cada vértice
    Para i de 0 até n-1 faça:
      ST[i] ← (1 << i)   // cada vértice cobre a si mesmo
```

```
# Ler as m arestas e atualizar as máscaras de cobertura
Para cada aresta de 1 até m faça:
  Ler a linha LINES[IDX] e dividir em dois números: A e B
  Incrementar IDX
  A ← A - 1   // Ajustar para índice 0-based
  B ← B - 1   // Ajustar para índice 0-based
  ST[A] ← ST[A] OR (1 << B)
  ST[B] ← ST[B] OR (1 << A)

# Construir o vetor L_ARR para poda durante a busca (backtracking)
L_ARR[n-1] ← ST[n-1]
Para i de n-2 até 0, decrescendo, faça:
  L_ARR[i] ← ST[i] OR L_ARR[i+1]

# Definir a máscara completa que representa todos os vértices cobertos
FULL_MASK ← (1 << n) - 1

# Função DFS (backtracking) para testar combinações de estações
Função DFS(STATE, STEP, S, MAXLEN):
  Se STATE == FULL_MASK então:
    Retornar VERDADEIRO   # Todos os vértices estão cobertos

  Se STEP == MAXLEN então:
    Retornar FALSO   # Número máximo de estações atingido sem cobrir todos

  Se S >= n então:
    Retornar FALSO   # Não há mais vértices para selecionar

  Para i de S até n-1 faça:
    # Poda: se mesmo combinando com todos os vértices de i em diante
    # não se alcança a cobertura total, interromper o loop
    Se (STATE OR L_ARR[i]) ≠ FULL_MASK então:
      Sair do loop

    # Se adicionar o vértice i não aumenta a cobertura, pular para o próximo
    Se (STATE OR ST[i]) == STATE então:
```

```
            Continuar para a próxima iteração

        # Tentar incluir a estação no vértice i e fazer chamada recursiva
        Se DFS(STATE OR ST[i], STEP + 1, i + 1, MAXLEN) retorna VERDADEIRO ent
            Retornar VERDADEIRO

      Retornar FALSO   # Nenhuma combinação válida encontrada a partir deste es

    # Buscar o número mínimo de estações necessário
    Para MAXLEN de 1 até n faça:
      Se DFS(0, 0, 0, MAXLEN) retorna VERDADEIRO então:
        ANSWER ← MAXLEN
        Sair do loop

    Imprimir ANSWER

FIM
```