Explicação do Código

1. **keys and rooms**

**Visão Geral**

O problema consiste em determinar se é possível visitar todas as salas de um conjunto, onde cada sala contém chaves para outras salas. Começamos na sala 0 e precisamos verificar se podemos acessar todas as outras salas usando as chaves disponíveis.

**mÉTODOS**

* 1. **CanVisitAllRooms**

public bool CanVisitAllRooms(IList<IList<int>> rooms)

 Método principal que inicia o processo

 Cria o array de visitados

 Chama os métodos auxiliares

 Retorna se todas as salas podem ser visitadas

* 1. **percorreSalas**

private void percorreSalas(IList<IList<int>> rooms, bool[] visitado)

 Implementa a lógica de percorrer as salas usando DFS (Busca em Profundidade)

 Usa uma pilha para controlar as salas a serem visitadas

 Começa pela sala 0

 Para cada sala:

* Pega as chaves disponíveis
* Marca novas salas como visitadas
* Adiciona novas salas à pilha
  1. **verificaTodasSalasVisitadas**

private bool verificaTodasSalasVisitadas(bool[] visitado)

 Verifica se todas as salas foram visitadas

 Retorna false se encontrar alguma sala não visitada

 Retorna true se todas as salas foram visitadas

Ou seja, ele começa na sala 0, marca ela como visitada e adiciona na pilha, fazendo o mesmo processo enquanto houver salas na pilha.

1. **Longest increasing path in a matrix**

**Visão geral**

O problema consiste em encontrar o caminho mais longo em uma matriz onde cada próximo elemento deve ser maior que o anterior. O caminho pode ser formado movendo-se para cima, baixo, esquerda ou direita.

**mÉTODOS**

* 1. **LongestIncreasingPath**

public int LongestIncreasingPath(int[][] matrix)

 Método principal que inicia a busca

 Verifica casos base (matriz nula ou vazia)

 Inicializa a matriz de memorização

 Percorre cada célula da matriz como ponto de partida

 Retorna o maior caminho encontrado

* 1. **DFS**

private int DFS(int[][] matrix, int linha, int coluna, int[,] memo)

 Implementa DFS com memorização

 Parâmetros:

* matrix: matriz de entrada
* linha: linha atual
* coluna: coluna atual
* memo: matriz de memorização

 Retorna o tamanho do maior caminho começando na posição atual

Ou seja, o algoritmo percorre cada célula da matriz e, partindo dela, usa DFS para explorar as 4 direções possíveis buscando valores maiores. Usa memorização para evitar recalcular caminhos já visitados, sempre mantendo registro do maior caminho encontrado.

1. **Number of flowers in full bloom**

**Visão geral**

O problema consiste em determinar quantas flores estão em plena floração em diferentes momentos quando pessoas visitam um jardim. Cada flor tem um tempo de início e fim de floração, e precisamos calcular quantas flores estão floridas em cada momento específico.

**mÉTODOS**

* 1. **FullBloomFlowers**

public int[] FullBloomFlowers(int[][] flowers, int[] people)

 Método principal que coordena o processo

 Extrai e ordena os tempos de início e fim

 Calcula o resultado para cada pessoa

* 1. **ContarMenorOuIgual**

private int ContarMenorOuIgual(int[] arr, int x)

 Implementa busca binária

 Encontra quantos elementos no array são menores ou iguais a x

 Retorna à posição onde x deveria ser inserido

* 1. **ExtrairEOrdenarTempos**

private int[] ExtrairEOrdenarTempos(int[][] flowers, bool isInicio)

 Extrai os tempos (início ou fim) das flores

 Ordena os tempos para permitir busca binária

 Retorna array ordenado

* 1. **CalcularFloresEmBloom**

private int CalcularFloresEmBloom(int tempoPessoa, int[] tempoInicio, int[] tempoFinal)

 Calcula quantas flores estão floridas em um momento específico

 Usa busca binária para contar flores que começaram e terminaram

 Retorna a diferença (flores ativas)

Ou seja,o algoritmo calcula quantas flores estão floridas em cada momento que uma pessoa visita. Para isso, mantém dois arrays ordenados: um com tempos de início e outro com tempos de fim da floração. Para cada pessoa, usa busca binária para encontrar quantas flores começaram a florescer até aquele momento e subtrai quantas já terminaram, resultando no número de flores em plena floração naquele instante.

1. **Shortest path visiting all nodes**

**Visão geral**

O problema consiste em encontrar o caminho mais curto que visita todos os nós de um grafo pelo menos uma vez. Cada nó é conectado a outros através de arestas, e o objetivo é calcular o menor número de passos para visitar todos os nós do grafo, independentemente do nó de partida.

**mÉTODOS**

* 1. **ShortestPathLength**

public int ShortestPathLength(int[][] graph)

 Implementa BFS modificado com estados de bits

 Inicia a busca a partir de todos os nós simultaneamente

 Usa máscara de bits para rastrear nós visitados

 Retorna a menor distância que visita todos os nós

Ou seja, o algoritmo usa bfs começando simultaneamente de todos os nós do grafo, utilizando uma máscara de bits para rastrear quais nós já foram visitados em cada caminho. Para cada nó atual, explora seus vizinhos calculando um novo estado de visitação, e continua até encontrar um estado onde todos os nós foram visitados ou até explorar todas as possibilidades.