Estratégias Algorítmicas

Problema de Programação 2

Pyramid Scheme

Equipa:

N° Estudante: 2019218981 Nome: Marco da Cruz Pais

N° Estudante: 2019215412 Nome: Simão Carvalho Monteiro

1. Descrição do Algoritmo

Algoritmo:

O nosso algoritmo consiste em avaliar, para cada nó, qual o caminho mais curto e, posteriormente, de maior valor.

Para fazer essa avaliação o algoritmo considera dois caminhos possíveis. O primeiro caminho nunca contém o nó pai, ou seja, tem que conter obrigatoriamente os nós filhos, enquanto que o segundo caminho contém sempre o nó pai, ou seja, pode ou não conter o nó filho.

Primeiro são calculados os caminhos para os nós filhos para depois poderem ser feitas as escolhas em relação ao nó pai.

Como, primeiramente, calculamos os caminhos dos nós filhos, no nó pai apenas precisamos de escolher o caminho certo de cada nó filho para cada caminho do nó pai.

Overlapping Subproblems:

Os subproblemas considerados foram o cálculo da cobertura, com o menor número possível de nós e maior valor, das subárvores esquerda e direita de cada nó.

Speed-up Tricks:

Não foram utilizados speed-up tricks, o uso da programação dinâmica juntamente com o depth first search permitiu-nos obter um algoritmo rápido e eficiente.

2. Estruturas de Dados

Foi usada uma classe, *Pyramid*, e 4 vetores muito importantes .

Classes:

- *Pyramid* – Criada para guardar as propriedades de cada caso de teste, como o número de índice máximo, os vetores *members* e *paid*. Aloja, também, funções como, *addEdge*, *addPaid*, *addSize* e as funções principais que são usadas para a resolução do problema, *minSizeVertexCover* e *dfs*.

Vetores:

- *members* – É um array de vetores usado para guardar o índice do nó pai de cada nó e os índices dos nós filhos.
- *paid* – É um array de vetores que guarda o valor pago por cada nó no seu índice respectivo.
- *dp* – Array de vetores que guarda número de nós mínimo para fazer a cobertura dos nós das diferentes subárvores.
- *value* – Array de vetores que guarda o peso máximo de cada combinação de nós pertencentes à cobertura.

3. Assertividade

A nossa solução está correta visto que são testadas todas as coberturas possíveis das subárvores escolhendo sempre a mais pequena e a que tem um maior valor. Para resolver este problema usamos programação dinâmica que, com o depth first search, permitiu-nos obter uma solução rápida e correta.

4. Análise do Algoritmo

- Complexidade Espacial é **O(N)** pois é referente ao tamanho dos diferentes vetores, ou seja, quanto maior seja o índice máximo, maior será o *vetor* dp e *value*. Existe então um relacionamento linear entre o número máximo de índice e os vetores referidos.
- Complexidade Temporal é **O(N)**.

5. Referências

As seguintes referências foram usadas com intuito de perceber alguns detalhes e funcionalidades sobre estruturas e funções do C++. Foram, também, utilizadas para compreender o *vertex cover problem* que é semelhante ao problema proposto e as características da programação dinâmica dadas na aula.

GeeksforGeeks

C++ Reference

Overlapping Subproblems Property in Dynamic Programming | DP-1 - GeeksforGeeks

Optimal Substructure Property in Dynamic Programming | DP-2 - GeeksforGeeks

Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm) - GeeksforGeeks

Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree) - GeeksforGeeks