Report for Programming Problem 2

Team:

Student ID: 2018298933 | **Name:** Gustavo Toste Bizarro
Student ID: 2019219057 | **Name:** Thomas Pereira Fresco

1. Algorithm description

O algoritmo desenvolvido tem como base uma abordagem dinâmica, resolvendo-se primeiro o sub-problema mais pequeno (as folhas) até ao maior (a árvore inicial). A obtenção de soluções ótimas para esses sub-problemas permite, de seguida, uni-las para obter a solução do problema inicial.

A árvore é preenchida de baixo para cima, contendo em cada nó a solução ótima do seu sub-problema, ou seja, o número mínimo de membros necessários e também o valor máximo. Tendo essa solução e pretendendo-se calcular a do nível acima, calcula-se como ficaria o resultado incluindo e não incluindo o nó do nível superior, escolhendo-se aquele que apresenta, prioritariamente, o menor número de membros na solução e, em caso de empate, o maior valor total.

Para *speed-up*, há uma condição que impede que um nó seja recalculado, este passo permite diminuir imenso o tempo de execução.

2. Data structures

1. Class Member - Representa um nó da árvore
    - int value - valor do membro;
    - int countVertex, sumValue - aqui é guardado o resultado dos sub-problemas/problema;
    - vector<int> recruited - os membros recrutados por ele;

2. Class Pyramid - Representa o esquema da pirâmide, é responsável por guardar a informação dos membros.
    - vector<int> temp - array de tamanho 2 utilizado no cálculo das soluções, assim, não é necessário utilizar mais memória em cada passo da recursão;

- map<int,Member> tree - dicionário contendo a informação dos membros;

3. Correctness

A solução desenvolvida obteve a pontuação máxima (200 pontos) na plataforma Mooshak. A obtenção do máximo de pontos dever-se-á à utilização válida do conceito denominado por programação dinâmica.

Para garantir um resultado correto, é essencial que as soluções calculadas para os sub-problemas sejam ótimas. A figura abaixo demonstra como são obtidas essas soluções.

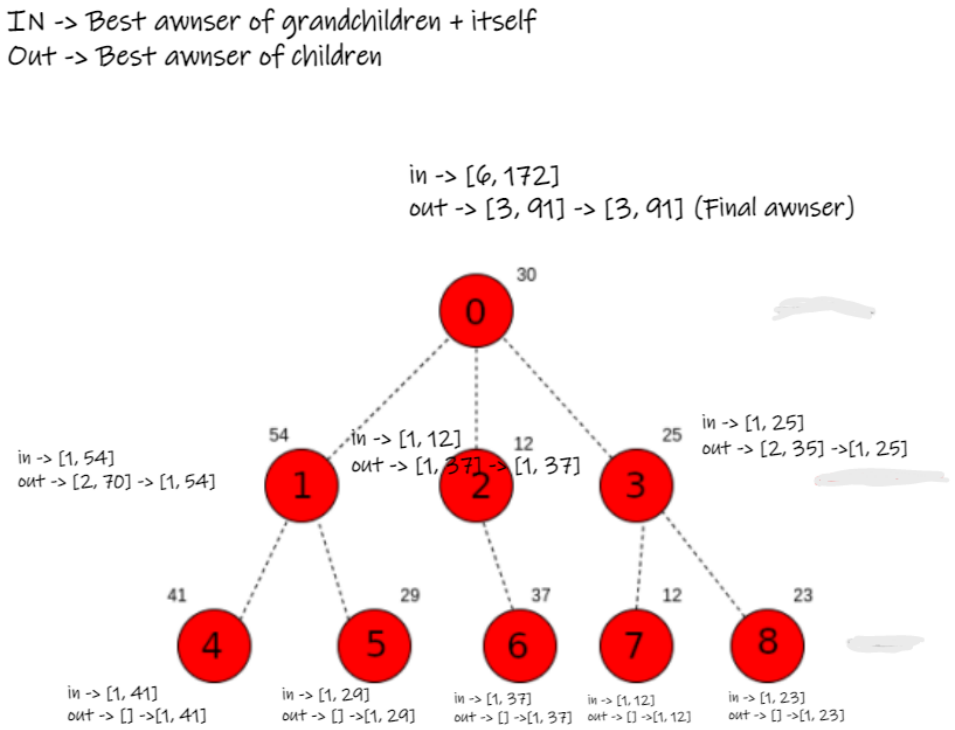

Figura 1 - Representação gráfica da execução da solução sobre um caso de teste.

4. Algorithm Analysis

A complexidade espacial do algoritmo é fixa, pois é linearmente proporcional à quantidade de membros. Tendo apenas os seus dados guardados (na forma de uma árvore) e trabalhando sempre sobre eles, obtemos uma complexidade espacial de O(n).

Quanto à complexidade temporal, na versão final do código e tendo em mente que não são recalculados os diferentes nós (se fossem, teria uma complexidade exponencial), há dois passos a considerar: a navegação única da árvore e os cálculos efetuados nos nós. Ambos têm uma complexidade O(n), pois são linearmente proporcionais ao número de membros. Há também a fase de leitura dos dados, também com uma complexidade O(n). Com estes dados referidos, poder-se-á notar que a complexidade temporal total do programa é O(n).

5. References

- GeeksforGeeks, 10 junho 2021, acesso a 10 de abril de 2022, disponível em: https://www.geeksforgeeks.org/vertex-cover-problem-set-2-dynamic-programming-solution-tree/