Report for Programming Problem 2

Team: 2019219068_2019214567
Student ID: 2019214567 Name Miguel Antunes Carvalho Portugal Ferreira
Student ID: 2019219068 Name Tiago Miguel Gomes Marques Oliveira

1. Algorithm description

Durante a leitura do input, os membros são guardados numa estrutura ***members*** e no fim da leitura do caso é chamada a função recursiva ***minSizeCover*** com o primeiro elemento desta estrutura como argumento. Esta função retorna um *pair* em que o primeiro elemento é a soma do ***vertexCover*** de todos os nós pertencentes à solução e o segundo elemento é o ***totalCost*** desta solução. Ambas estas variáveis são explicadas na secção de **Data Structures**.

Dentro da função, verifica-se se o nó atual tem filhos, sendo esta uma das condições de paragem da recursão (a ausência de filhos indica-nos que chegamos a uma folha da árvore). De seguida verificamos se o nó atual já foi visitado durante a execução do programa e, se este tiver sido o caso, paramos a recursão de modo a poupar tempo.

O funcionamento do algoritmo toma, essencialmente, uma decisão: juntar o nó atual à solução atual ou não juntar.

Para a primeira opção a variável ***sizeWith*** é inicializada a 1 e a variável ***totalCostWith*** é inicializada com o custo do nó atual. De seguida o vector com os filhos do nó atual é percorrido e a função ***minSizeCover*** é chamada para cada um deles, somando o ***sizeWith*** e o ***totalCostWith*** com o primeiro e segundo valor retornados na recursão da função, respetivamente.

Para a segunda opção as variáveis ***sizeWithout*** e ***totalCostWithout*** são inicializadas a 0 e os filhos do nó atual são percorridos num ciclo, incrementando o ***sizeWithout*** em 1 por cada filho e somando o custo de cada filho ao ***totalCostWithout***. Ainda dentro deste ciclo, os *vectors* que contém os filhos dos filhos também vão ser percorridos, chamando a função ***minSizeCover*** para cada um deles e somando o ***sizeWithout*** e o ***totalCostWithout*** com o primeiro e segundo valor retornados na recursão da função, respetivamente.

No final destes dois trechos de código, comparamos o ***sizeWith*** (que indica o número mínimo de nós para a solução em que o nó atual é selecionado) com o ***sizeWithout*** (indica o número mínimo de nós para a solução em que o nó atual não é selecionado), selecionando o menor. Caso estes tenham o mesmo número de nós, será selecionado aquele que apresente o custo total maior, através da comparação do ***totalCostWith*** (soma dos custos quando o nó atual é selecionado) com o ***totalCostWithout*** (soma dos custos quando o nó atual não é selecionado).

2. Data structures

Foi utilizada a classe ***Member***, que representa um membro do nosso esquema em pirâmide. Esta classe contém as seguintes variáveis:

- o inteiro ***cost***, que indica o dinheiro que este membro pagou quando foi recrutado;
- o inteiro ***vertexCover***, que guarda o número total de membros que fazem parte da solução que está a ser desenvolvida;
- o inteiro ***totalCost***, que indica a soma dos montantes que os membros desta solução pagaram;
- o vector ***recruits***, que indica o ID dos membros recrutados pelo membro atual, fazendo com que a nossa estrutura usada para guardar os dados tome a forma de uma árvore;

Todos os membros são guardados no *unordered map* ***members***, em que a chave é o ID do membro e o valor é um ponteiro para o respetivo nó.

3. Correctness

O nosso algoritmo atingiu os 200 pontos porque explora todas as combinações possíveis e eficientes de nós, além de poupar tempo cortando execuções desnecessárias, parando a recursão em nós que já foram visitados e usando os valores previamente calculados para esse nó.

4. Algorithm Analysis

A complexidade espacial é O(V), em que V é o número de nós (membros), porque apenas precisamos de um vector para guardar todos os nós.

A complexidade temporal é O(V), visto que a função recursiva é chamada apenas uma vez para cada nó. Isto é assegurado através da paragem da recursão em nós que já foram previamente visitados.

5. References

GeeksforGeeks. (10 de 6 de 2021). *Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree)*. Obtido de GeeksforGeeks: https://www.geeksforgeeks.org/vertex-cover-problem-set-2-dynamic-programming-solution-tree/