Relatório do Problema de Programação 2

Equipa:

N.º Estudante: 2019216949 Nome: Duarte Emanuel Ramos Meneses
N.º Estudante: 2019213995 Nome: Patrícia Beatriz Silva Costa

1. Descrição do Algoritmo

O nosso algoritmo assenta numa análise, em cada nó, do valor necessário para utilizar ou não esse membro. Atravessando da raiz até às folhas para partir de lá (caso base), vamos subindo na árvore tendo já calculados os valores para os nós abaixo (solução *top-down*). Deste modo, chegando à raiz, temos o valor ótimo para resolver este problema. Decidimos que o caso base devia ser um nó ser folha, uma vez que neste caso não existem filhos, logo o nó não tem de se preocupar em analisar os valores mais abaixo na árvore. Sendo folha, apenas tem de se analisar a si próprio. Se for usado, a variável *put* fica a (1, valor do nó), uma vez que se usa 1 nó e o valor que o conjunto de pessoas pagou é igual ao pago por esse nó. Caso não seja utilizado, a variável *no_put* fica a (0, 0) pois não foi usado nenhum membro, logo também não existe valor monetário a considerar.

Deste modo, tendo isto, o algoritmo vai subindo na árvore, e analisando para cada nó, o caso de esse membro ser utilizado ou não:

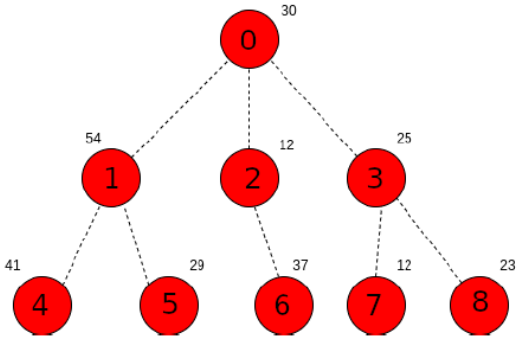
- Não usar: se não usamos o nó, temos que ter em conta os filhos pois precisamos deles para conhecer os nós abaixo. Com isto, é necessário usar os casos em que os filhos são usados e somar esses pares.
- Usar: se usamos o nó em si, precisamos também de, para além do par (1, valor do nó), escolher os melhores casos de cada filho e somar esses pares. Isto, pois, queremos "levar" os melhores casos até à raiz.

Sendo que o nó "0" representa o topo da pirâmide, ou seja, é o primeiro recrutado da empresa e tendo este conhecimento dos seus filhos (recrutados por ele), decidimos percorrer a árvore da raiz até às folhas de forma a chegar ao caso base. Estando aí, vamos voltar para cima na recursão, tendo os valores de baixo já calculados.

Depois de percorrer toda a árvore de baixo para cima, teremos então os dois pares da raiz com os seus valores respetivos. É através deste nó que teremos o resultado, verificando qual o melhor caso: *put* ou *no_put.*

Este algoritmo permite-nos analisar cada nó apenas uma vez. Para resolver este problema, teoricamente seria necessário calcular todas as possibilidades e, em cada uma, calcular o caso de utilizar ou não o membro. Da forma que fazemos, analisamos todas as possibilidades, escolhendo logo a melhor, poupando tempo de execução. Para cada nó é sempre necessário analisar os valores mais abaixo na árvore (repetitivo). Como vamos somando os valores à medida que vamos subindo, não necessitamos de os calcular novamente.

Exemplo do enunciado:



Casos Base:
Usar: (1, valor do nó); Não Usar: (0, 0).

Folha 4: Usar: (1, 41); Não Usar: (0, 0);

Folha 5: Usar: (1, 29); Não Usar: (0, 0);

Folha 6: Usar: (1, 37); Não Usar: (0, 0);

Folha 7: Usar: (1, 12); Não Usar: (0, 0);

Folha 8: Usar: (1, 23); Não Usar: (0,0).

Outros nós:
Usar: (1, valor do nó) + melhor caso de cada filho;
Não usar: Caso "Usar" de cada filho.

Nó 1: Usar: (1, 54) + (0, 0) + (0, 0) = (1, 54);
Não Usar: (1, 41) + (1, 29) = (2, 70)
Nó 2: Usar: (1, 12) + (0, 0) = (1, 12);
Não Usar: (1, 37)
Nó 3: Usar: (1, 25) + (0, 0) + (0, 0) = (1, 25);
Não Usar: (1, 12) + (1, 37) = (2, 49)
Raiz 0: Usar: (1, 30) + (1, 54) + (1, 37) + (1, 25) = (4, 146);
Não Usar: (1, 54) + (1, 12) + (1, 25) = (3, 91)

O resultado é obtido através das informações obtidas na raiz, onde é verificado o melhor caso: menor número de nós (membros para investigar que confirmem todas os recrutamentos) com o maior valor possível (em caso de "empate" no critério anterior).

Resultado para este caso: (3, 91)

2. Estruturas de dados

Para implementar com sucesso o nosso algoritmo, decidimos utilizar uma estrutura representativa de cada membro que tem como atributos o valor que esse membro pagou para entrar na organização, um vetor de inteiros que contém os ids dos membros recrutados por este membro e dois pares ordenados que correspondem ao número de pessoas necessárias para decifrar se é um esquema em pirâmide e o valor pago por essas pessoas caso se utilize aquele nó e caso não.

De modo a guardar estes membros utilizamos um *map* que tem como chave o id do membro e como valor a estrutura membro, já referida acima, correspondente.

3. Correção

Para a nossa solução não ser ótima, tem de existir pelo menos uma solução mais ótima para calcular o caso de utilizar o nó e/ou o caso de não o fazer.

Subestrutura ótima:

Assunção: A nossa solução é ótima.

Porém, existe outra solução mais ótima.

Negação: No caso de usar um membro, seria mais ótimo se escolhêssemos ou um valor de nó menor (impossível) ou se escolhêssemos valores melhores dos filhos. Sendo que neste caso já escolhemos os melhores, é impossível otimizar este caso.

Já no caso de não usar o nó, melhorávamos a solução se conseguíssemos valores de usar os filhos melhores. Sendo que para usar os filhos, utilizamos a técnica do parágrafo acima, é impossível otimizar.

Consequência: A nossa solução é ótima.

Contradição: Não existe nenhuma solução mais ótima que a nossa.

4. Análise do Algoritmo

Complexidade espacial: Sendo que no nosso algoritmo temos um *map* de tamanho igual ao número de membros (N), a complexidade espacial é:

$$N => O(N)$$

Complexidade temporal: No nosso algoritmo percorremos 1 vez os membros todos na leitura. Após termos os membros todos lidos e na estrutura devida, percorremos a "árvore" da raiz até as folhas passando por todos os nós recursivamente. Posto isto, chegando às folhas, o nosso algoritmo volta até à raiz passando novamente por todos os nós (recursividade). Deste modo, sendo N o número de membros, a complexidade temporal é dada por:

$$N + 2 * N => 3N => O(N)$$

5. Referências

- https://stackoverflow.com/questions/17724925/parse-string-containing-numbers-into-integer-array [acedido em 04/04/2022]