Report for Programming Problem 2

Team: 2019217030_2019227240

Student ID: 2019227240 Name: Sofia Botelho Vieira Alves

Student ID: 2019217030 Name: João Dionísio

1. Algorithm description

O algoritmo implementado começa por criar estruturas para os elementos da rede ("**no_aux**"). Cada elemento contém um vector com o id de todos os elementos com quem contactou ("**connects**"). É guardado também o valor que este pagou para poder pertencer ao esquema ("**weight**") e o seu id ("**id**"). Por fim, vamos guardando todos os elementos num dicionário ("**all_connects**"), contendo como chaves o **id** do elemento e o respetivo **nó**.

Depois de guardarmos todos os dados nas estruturas, passamos à resolução do problema em si. Para isso, criamos a função "**detective**". Esta função propõe resolver o problema com uma abordagem recursiva e top-down, já que será consecutivamente chamada para cada um dos nós à medida que vamos descendo no grafo criado com os elementos do esquema, tendo como caso base as folhas deste, que correspondem aos últimos elementos recrutados. Cada nó do grafo possui dois *pairs*, o *pair* **used** e o *pair* **unused**. Cada um guarda o valor movido e o número de nós visitados até ao nó atual (<valor, nós>), contando com este (**used**) ou não (**unused**).

Ao encontrarmos os nós folha, calculamos estes dois pares para todos os nós ao voltarmos na recursão. Para os nós folha, caso estes sejam usados, o par **used** irá ficar <valor do nó , 1>, já que nesse momento usamos apenas 1 nó, e o par **unused** irá ficar <0 , 0>. Para os restantes nós, estes irão considerar todos os pares dos nós anteriores a estes, used e unused, e escolher a melhor combinação, escolhendo aqueles que possuem o menor número de nós visitados. Caso existam combinações onde este número seja igual, o critério de desempate serão aqueles que moveram o maior valor. Vamos aplicando estes critérios até chegarmos à raiz, devolvendo assim o menor número de nós visitados e o valor movido pelos mesmos.

2. Data structures

Neste projeto foram usadas como estruturas de dados vetores, estruturas, "maps" e "pairs". A seguir, apresenta-se uma breve descrição de cada estrutura utilizada.

- **Map "all_connects"**: Dicionário que contém as estruturas "node" mapeadas

para os seus respectivos id 's.

- **Estrutura ”node”**: Representa um elemento do esquema em pirâmide.
    - **Vetor “connects”**: Onde colocamos os id ’s dos vários elementos recrutados pelo “node”.
    - **Pair <int, int> “used”**: Guarda o valor (weight) total da árvore até ao nó atual e o número de elementos caso o “node” seja usado.
    - **Pair <int, int> “unused”**: Guarda o valor (weight) total da árvore e o número de elementos caso o “node” não seja usado.

3. Correctness

Para provar que esta solução encontra-se correta, iremos recorrer à negação por absurdo.

Sub-problema: Dado o conjunto de pessoas pertencentes a um esquema em pirâmide, encontrar o subconjunto que permite conhecer todos os elementos (através de relações de contratação).

1. **Assunção:** Sendo o fator de otimização a redução do tamanho do conjunto solução, vamos assumir que a solução mais otimizada é S.
2. **Negação:** Sendo a solução J uma solução com mais elementos que S, vamos assumir que $|J| < |S+1|$.
3. **Consequência:** Retirando um elemento a J e $S+1$, obtemos que $|J-1| < |S|$.
4. **Contradição:** Contudo isto levaria à contradição do ponto 1.

Desta forma, conclui-se que $|S+1| = |J|$.

4. Algorithm Analysis

Para o cálculo da complexidade temporal, procedemos à análise da função “**detective**". Numa fase inicial, esta percorre os filhos de cada nó e, para cada filho, volta a chamar a mesma função, até chegar às folhas. Assim sendo, percorre **n-1** nós, neste ciclo, mais o nó inicial (índice 0), o que resulta em **n**. Ao voltar na recursão, percorremos outra vez os filhos de cada nó ao chamar a função “**compare_childs**” adicionando **n-1** (**n** nós menos um, que corresponde à raiz) à expressão anterior, obtendo assim, **2n - 1**. Assim, conclui-se que a complexidade temporal será $O(n)$.

Para o cálculo da complexidade espacial, analisando todas as estruturas de dados criadas chegamos à expressão

$$n * (1 + 1 + (10))$$

que corresponde ao tamanho de dicionário **all_connects**, sendo n o número de elementos, representados por estruturas, e $(1 + 1 + (10))$ o tamanho máximo de uma estrutura. $1 + 1$ representa os dois pares used and unused, e 10 representa o tamanho máximo do vector connects. Assim, conclui-se que a complexidade espacial será $O(n)$.

5. References

https://www.cplusplus.com/
Powerpoints da disciplina disponibilizados pelo docente