Prova de Complexidade de Algoritmos

Daniella Martins Vasconcellos

Abril 2021

1 Questão 1

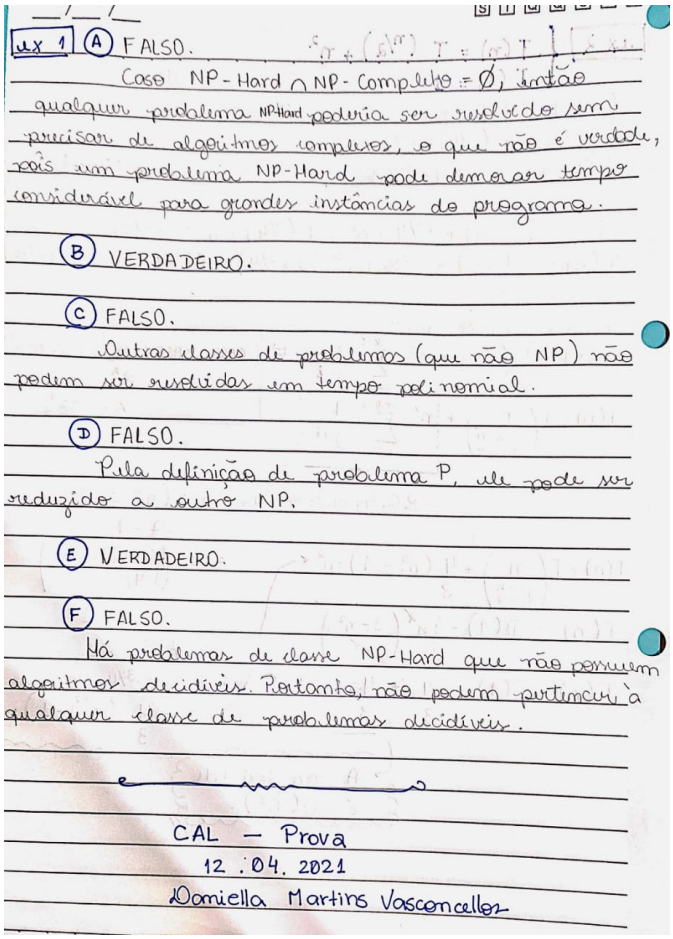

Figure 1: Resposta da Questão 1.

2 Questão 2

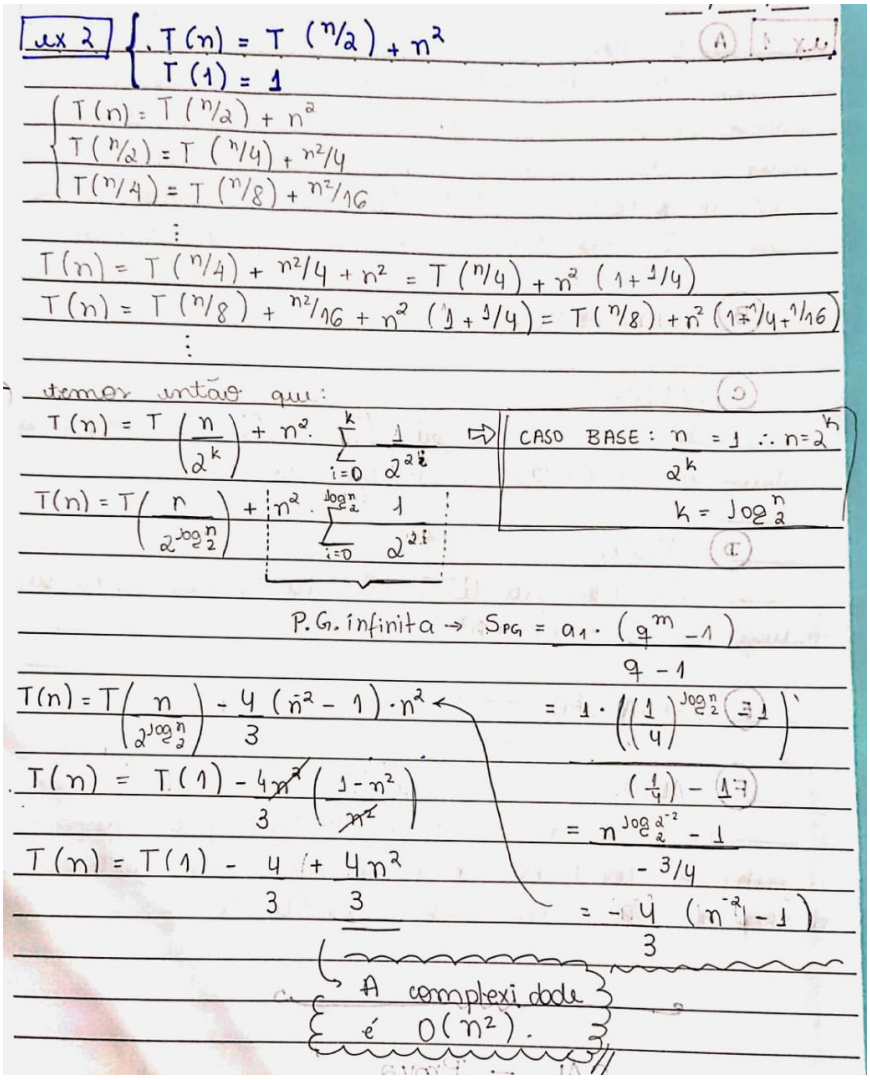

Figure 2: Resposta da Questão 2.

3 Questão 3

```
1   void heapfyLeaf(int *a, int k)
2   {
3       int anterior = parental(n);
4       if(a[k] > vetor[anterior])
5       {
6           swap(&a[anterior], &a[k]);
7           heapfyLeaf(a, anterior);
8       }
9   }
```

Figure 3: Implementação da função heapfyLeaf().

Olhando a figura 3, vemos que a complexidade de tempo da função O(n-1) por conta da recursivdade na linha 7 (todas as outras linhas são O(1) e não interferem no cálculo).

```
void insertHeap(int *a, int n, int l, int e)
{
    if (n < l)    O (1)
    {
        a[n] = e;    O(1)
        heapfyLeaf(a, n);    O(n-1)

    }
    else
    {
        printf("insertHeap: Heap overflow!");    O(1)
        exit(1);    O(1)
    }
}
```

Figure 4: Cálculo da complexidade de tempo do algoritmo apresentado pelo professor.

Sendo assim, calcula-se que tanto a complexidade de tempo do inserHeap() quanto a complexidade de espaço (por n-1 chamadas ser o pior caso) é O(n-1).

4 Questão 4

Abaixo, na figura 5 está representada a execução do algoritmo de Kruskal. A complexidade é O(E log V).

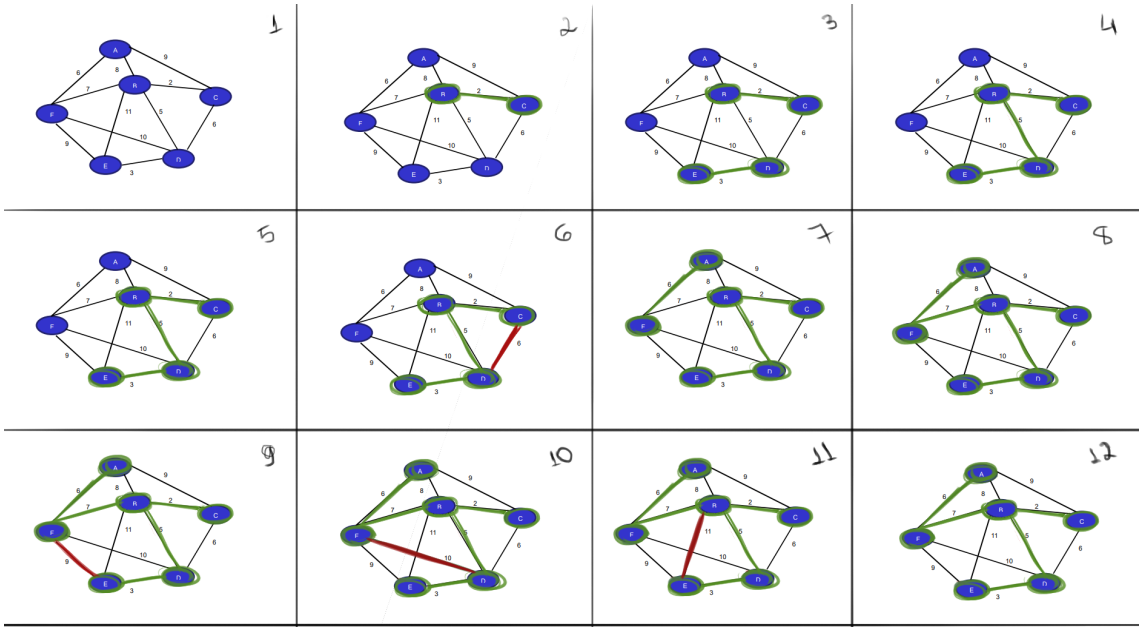

Figure 5: Algoritmo de Kruskal.

5 Questão 6

Para que essa teoria seja provada, primeiro deve existir um algoritmo que, dado uma instância do problema e um subconjunto de vértices, verifica em tempo polinomial se tal conjunto é um conjunto que não depende da instância. Segundo, transforma-se uma fórmula 3CNF-SAT em um grafo, criando um vértice para cada termo da fórmula e ligando os termos separados por operadores OU por uma aresta, assim como termos que possuem negações.