Observações:

- A prova tem seis questões dos quais quatro tem que ser respondidas. Caso você responde mais que quatro questões, por favor indique quais devem ser avaliados.

Prova

Questão 1 (Fluxos 1, 2.5pt)
Temos um grafo direcionado $G = (V, A)$ com capacidades inteiras positivas $c_a \in \mathbb{Z}_+$, $a \in A$ nos arcos e um fluxo inteiro máximo $f_a \in \mathbb{Z}_+$, $a \in A$ correspondente. Supõe que aumentamos a capacidade de um determinado arco $a \in A$ por um, i.e., $c'_a = c_a + 1$. Mostra que é possível obter o novo fluxo máximo em $O(m + n)$.

Questão 2 (Fluxos 2, 2.5pt)
Descreve o algoritmo push-relabel. Quais as principais operações e como eles são aplicados? Fornece um pseudocódigo (em alto nível). Qual a complexidade do algoritmo?

Questão 3 (Melhor não atrasar, 2.5pt)
Temos n tarefas que queremos executar numa única máquina. Cada tarefa possui uma duração p_i, $i \in [n]$. Além disso, cada tarefa vem com um prazo d_i, $i \in [n]$. É desejável que toda tarefa termina antes do seu prazo. Caso contrário, a tarefa é atrasada. Queremos encontrar uma permutação das tarefas, tal que o número de tarefas atrasadas é o menor possível. Como não importa o quanto uma tarefa atrasa, sempre existe uma solução na qual todas tarefas no prazo precedem todas tarefas atrasadas. O seguinte algoritmo de Moore determina o conjunto de tarefas no prazo:

```
Ordene as tarefas tal que d_1 ≤ d_2 ≤ ··· ≤ d_n.
S := ∅, t := 0
for i = 1,2,...,n do
  S := S ∪ {i}
  t := t + p_i
  if t > d_i then
    encontra a tarefa j em S de maior duração p_j
    S := S \ {j}
    t := t − p_j
  end if
end for
return S
```

Propõe uma implementação concreta desse algoritmo e determina a complexidade do algoritmo proposto.

Questão 4 (O caixeiro viajante focado, 2.5pt)
(Como sempre em problemas com caixeiros supõe que temos um grafo completo não-direcionado $G = (V, A, d)$ com distâncias d_a, $a \in A$.)
O caixeiro viajante focado sabe que não terá boas vendas em todas cidades: ele quer visitar somente um subconjunto $R \subseteq V$ delas, com uma boa demanda. (Obviamente ele mesmo mora numa cidade em R, e a sua viagem inicia nessa cidade.) Em outras palavras, ele quer visitar cada cidade em R exatamente uma vez, e, além disso, pode visitar cada cidade em $V \setminus R$ no máximo uma vez.
Qual a complexidade do problema? Propõe um algoritmo que resolve esse problema. Qual a complexidade do algoritmo proposto? Esta problema pode ser aproximado? Caso sim, como e com qual qualidade? Caso não: por quê?

Questão 5 (Algoritmos e a sua complexidade, 2.5pt)
Considere os algoritmos

> Hopcroft-Karp (emparelhamento máximo), Algoritmo Húngaro (emparelhamento máximo), Edmonds-Karp (fluxo máximo), Push-relabel (fluxo máximo), Dijkstra, Busca em profundidade

e as complexidades

$$\log n, n, m, \log m, n \log n, n^2, \sqrt{n}m, n\sqrt{m}, n \log m, m \log n, nm, n^2 m, nm^2, n^2 m^2, n^3$$

Atribui a cada algoritmo a complexidade correta. Uma complexidade pode ser atribuida a mais que um algoritmo. Não toda complexidade tem que ser atribuida. Caso um algoritmo possui variantes com complexidades diferentes, explica qual variante foi escolhida e justifica a complexidade.

Questão 6 (O cuco encadeado, 2.5pt)
Um colega propõe a seguinte nova ideia para uma tabela hash: similar ao cuco hashing usa duas funções hash, mas em caso de colisões resolve via encadeamento (i.e. cada posição possui uma lista de elementos). Para inserir uma chave, ele sempre insere na lista de menor tamanho. Para buscar uma chave ele busca nas duas listas. O colega afirma que isso une as melhores características das duas abordagens.
Essa ideia é boa? Qual a complexidade das operações? O colega tem razão? Compara com o cuco hashing normal, e com hashing com encadeamento e endereçamento aberto.