5. (2.0pt) Um servidor recebe requisições de n clientes no instante 0, uma de cada cliente. Sabemos que o tempo que o servidor leva desde o instante que ele começa a atender a requisição do cliente i até o instante em que ele termina é t_i. Além disso, sabemos que o servidor atende apenas uma requisição por vez e que quando uma requisição começa a ser atendida ela não pode mais ser interrompida.

a) O tempo de espera de um cliente é definido como o tempo que ele espera até que sua requisição comece a ser atendida. Proponha um algoritmo eficiente para determinar a ordem em que o servidor deve atender os clientes de modo a minimizar o tempo médio de espera dos clientes.

6) Consider the problem of minimizing the maximum lateness of tasks with **release dates**. There are n tasks to be executed. Each task has a time t_i it takes be completed, and a due date d_i. However, task i is only released at time r_i (so you can only start doing this task after time r_i). Assume $d_i \geq r_i + t_i$, so can finish each task **by itself** before the deadline.

As before, the **lateness** of a task is the difference of when it was completed minus its due date. We want to decide when to start working on each task in order to minimize the maximum lateness. Recall at most 1 task can be done at a time.

Consider the greedy algorithm that orders the tasks by **increasing due date** and processes them in this order, starting the tasks in the first moment possible (respecting that you can do at most 1 job at a time and the release dates).

Does this greedy algorithm return an optimal schedule (that is, with smallest possible max lateness)? If so, give a brief justification; if not, give a counterexample.

14)

(a) Give an efficient algorithm that, given the start and finish times of all the sensitive processes, finds as small a set of times as possible at which to invoke `status_check`, subject to the requirement that `status_check` is invoked at least once during each sensitive process P.

Dica: basta considerar os tempos de termino dos processos como possiveis pontos pra rodar status_check

3. (3.0pt) Seja um grafo não direcionado $G = (V, E)$. Uma cobertura para G é um conjunto de vértices $C \subseteq V$ tal que toda aresta de E tem pelo menos uma extremidade em C. Considere o seguinte algoritmo guloso.

$C \leftarrow \emptyset$; $G' \leftarrow G$
Enquanto G' tem arestas
    $v \leftarrow$ vértice de G' com maior grau
    $C \leftarrow C \cup v$
    $G' \leftarrow G' - v$ [1]
Fim Enquanto
Devolva C

a) Este algoritmo sempre devolve uma cobertura? Por que?

b) Este algoritmo sempre devolve a cobertura com o número mínimo de vértices? Por que?

c) Explique como implementar o algoritmo acima de forma eficiente. Soluções mais eficientes terão maior pontuação.

- Capitulo 4 do livro Kleinberg-Tardos: exercicios 4, 5, 13