* ESCALONAMENTO (ou "AGENDAMENTO") DE TAREFAS * (Cormen, §16.5)

1. DEFINIÇÃO: a entrada do problema são um vetor de naturais d[1..n] tal que
   1 <= d[i] <= n, para todo i, e um vetor de penalidades w[1..n] reais não
   negativas. Uma solução para o problema é uma permutação S[1..n] do conjunto
   {1 ... n}. O objetivo do problema é encontrar uma solução de custo mínimo,
   sendo o custo c(S) de uma solução S definido como segue:

   * t(i,S) ou t(i) = j, onde S[j] = i, para todo i de 1 a n.   // término de i

   * c(i,S) = | w[i], se t(i) > d[i] | , para todo i de 1 a n. // custo de i
              | 0,   se t(i) <= d[i] | // em S

   * c(S) = SOMATÓRIO_{i=1}^n c(i,S) .   // custo de S

2. DEFINIÇÃO: Dada uma entrada (d[1..n] , w[1..n]) para o problema, uma
              PRÉ-SOLUÇÃO ÓTIMA é um vetor S[1..n] que é igual a alguma solução
   ótima para a entrada em questão, exceto por possuir "q" elementos S[i] iguais
   a zero, para algum q de 0 a n.

   Dada uma pré s.o. S e uma tarefa i ∈ {1 ... n}, nós dizemos que "i" está
   ESCALONADA/AGENDADA em S se e somente se S[j] = i para algum j. Nós então
   definimos ESC(S) ⊆ {1 ... n} como o conjunto das tarefas escalonadas em S, e
   NESC(S) = {1 ... n} \ ESC(S) (isto é, o conj. das tarefas não escalonadas).

3. DEFINIÇÃO: se

   (a) S[1..n] é uma pré s.o. para uma entrada (d,w) e NESC(S) não é vazio;
   (b) Se, dentre as tarefas em NESC(S), "i" é uma de maior penalidade; e
   (c) Se todas as atividades em ESC(S) tem penalidade maior ou igual a "i",

   então próx(S,i) é o vetor P[1..n] definido a seguir:

   - P[j] = S[j] para todo j tal que S[j] > 0.
   - Se existe j <= d[i] tal que S[j] = 0, então P[j] = i para o maior tal j;
     se não existe tal j, então P[j] = i para o maior j tal que S[j] = 0;
   - P[j] = 0 para todos os demais valores de j.

4. LEMA: se R[1..n] é uma pré s.o. para uma entrada (d,w), i ∈ NESC(R) e
         S = próx(R,i), então S também é uma pré s.o. para a entrada em questão.

==========
PROVA:

Como R é uma pré s.o. para (d,w), então existe uma solução ótima T para (d,w)
tal que, para todo j, R[j] != 0 => R[j] = T[j]. Assim sendo, observe
primeiramente que, se o mesmo vale para S, isto é, se S[j] != 0 => S[j] = T[j]
para todo j, então S também é, por definição, pré s.o. para (d,w), CQD.

Daqui para a frente, portanto, suponhamos que existe k tal que T[k] != S[k] > 0.
Observe, então, que, nesse caso, pela definição de S, S[k] = i para todo tal k,
o que implica que existe exatamente um tal valor k. Sejam, então:

  - h = T[k];
  - p = t(i,T);
  - U a solução obtida a partir de T trocando-se as tarefas i e h de posição.

Nós mostraremos a seguir que c(U) <= c(T), do que segue que S é pré s.o., CQD.
De fato, considere os casos possíveis:

* Caso 1: k <= d[i] e p <= d[i]. Nesse caso, temos p < k, pois T[k] != S[k].
          Logo, para toda tarefa t de 1 a n, temos:

- c(t,U) = c(t,T) = 0, se t = i, pois t(i,U) , t(i,T) <= d[i];
- c(t,U) <= c(t,T),   se t = h, pois t(h,U) = p < k = t(h,T);
- c(t,U) = c(t,T),   em caso contrário, pois t(t,U) = t(t,T).

Logo, c(U) <= c(T), CQD.

* Caso 2: k <= d[i] e p > d[i]. Nesse caso, temos:

```
c(T)  = c(i,T) + c(h,T) + SOMATÓRIO_{t != i,h} c(t,T)
      = w[i]   + c(h,T) + SOMATÓRIO_{t != i,h} c(t,T)    --> p > d[i]
     >= w[i] + SOMATÓRIO_{t != i,h} c(t,T)
     >= w[h] + SOMATÓRIO_{t != i,h} c(t,T)  --> w[i] = máx {w(t): t ∈ NESC(R)}
      = w[h] + SOMATÓRIO_{t != i,h} c(t,U)
      = w[h] + SOMATÓRIO_{t != i,h} c(t,U) + c(i,U)      --> k <= d[i]
     >= c(h,U) + SOMATÓRIO_{t != i,h} c(t,U) + c(i,U)
      = c(U), CQD.
```

* Caso 3: k > d[i]. Nesse caso, pela definição de S, temos i != R[x] > 0 para todo x <= d[i]. Logo, também vale p > d[i]. Além disso, pela definição de S, k = máx {x : R[x] = 0}, e portanto p < k. Temos então:

```
c(T)  = c(i,T) + c(h,T) + SOMATÓRIO_{t != i,h} c(t,T)
      = c(i,U) + c(h,T) + SOMATÓRIO_{t != i,h} c(t,U)    --> c(i,T/U) = w[i]
     >= c(i,U) + c(h,U) + SOMATÓRIO_{t != i,h} c(t,U)    --> t(h,U) < t(h,T)
      = c(U), CQD.
```

==========

5. UM ALGORITMO GULOSO, baseado no lema acima:

==========

```
Algoritmo: esc_tar_alto_nível
Entrada: vetores d[1..n] e w[1..n], conforme descrito anteriormente.
Saída: um vetor de naturais.
--------------------------------------------------------------------------------
1. Disponha as tarefas em ordem decrescente de penalidade,
   colocando os índices das tarefas num vetor I[1..n]
2. PARA j de 1 a n
3. | S[j] := 0
4. PARA j de 1 a n
5. | i := I[j]
6. | k := o maior índice x <= d[i] tal que S[x] = 0, se houver tal x,
7. |      ou, se não houver tal x, o maior índice x tal que S[x] = 0
8. | S[k] := i
9. RETORNE S.
```

==========

6. EXERCÍCIOS:

a) Se certifique de que a prova do lema 4 faz sentido para você. Em particular, um aluno comentou em sala que o enunciado do lema era tão óbvio para ele que ele não saberia demonstrá-lo; você entende, neste momento, como "o óbvio" foi provado precisamente a partir das definições?

b) Escreva uma implementação mais detalhada da linha 6 do algoritmo guloso acima, de forma a evidenciar que o algoritmo executa em tempo $O(n^2)$. Como critério de detalhamento, se certifique de que cada linha do código novo executa em tempo constante cada vez que é executada.

c) É fácil ver que é possível implementar a linha 1 de forma que ela execute em tempo O(n*lg n). Você consegue pensar numa implementação da linha 6 que também execute nessa complexidade assintótica de tempo?

d) Existe uma maneira de implementar eficientemente a linha 6 do algoritmo acima, utilizando uma estrutura de dados para conjuntos disjuntos. Utilizando-se a representação de conjuntos disjuntos por listas encadeadas (Cormen, §21.1 e §21.2), é possível implementar o algoritmo acima de forma que ele execute em tempo O(n*lg n). Utilizando-se a representação por árvores (Cormen, §21.3), todo o trecho do algoritmo após a linha 1 executa em tempo O(n*α(n)), onde α(n) é uma função que cresce muito lentamente, e que vale no máximo 4 em qualquer aplicação concebível da estrutura de dados em questão; em outras palavras; o trecho em questão do algoritmo executa em tempo praticamente linear, e portanto o algoritmo todo executa nessa complexidade de tempo CASO as tarefas de entrada já sejam fornecidas ordenadas para o algoritmo.

   A sua tarefa neste item é escrever uma implementação do algoritmo acima baseada no tipo de dados "conjuntos disjuntos" (Cormen, §21.1), independentemente de que estrutura de dados seja utilizada para implementá-lo. A sua implementação deve respeitar as complexidades de tempo mencionadas acima, caso alguma das duas estruturas acima seja utilizada para implementar o tipo de dados em questão.