UM PROBLEMA DE SELEÇÃO DE ATIVIDADES (Cormen, §16.1)

Considere "n" atividades, sendo c[i] o horário de começo da atividade "i" e f[i] o horário de fim da mesma atividade, com c[i] < f[i]. Digamos, ainda, que duas atividades "i" e "j" são compatíveis sse f[i] <= c[j] ou f[j] <= c[i], ou seja, sse não há choque de horário entre elas. O nosso problema é então o de, dados c[1..n] e f[1..n], encontrar um conjunto, tão grande quanto possível, que seja composto exclusivamente de atividades mutuamente compatíveis. Tal conjunto deve ser informado por meio de um vetor s[1..n], onde s[i] = 1 sse a atividade "i" foi selecionada, isto é, se ela pertence ao conjunto em questão, e s[i] = 0 em caso contrário.

UMA SOLUÇÃO POR PROGRAMAÇÃO DINÂMICA

Observe que o problema de seleção de atividades acima pode ser resolvido por programação dinâmica. De fato, todo conjunto de atividades compatíveis possui uma primeira atividade. Nesse caso, uma estratégia recursiva para resolver o problema é a seguinte:

1. Para cada atividade i ∈ [1..n], obtenha a melhor solução possível na qual a atividade "i" é a primeira atividade. Para tanto, resolva recursivamente a entrada composta pelas atividades que começam a partir do tempo f[i], e à solução encontrada adicione a atividade "i".

2. Retorne a maior das "n" soluções encontradas no passo anterior.

Antes de continuar o raciocínio:

1. EXERCÍCIO: o argumento de que o algoritmo acima de fato resolve o problema
   --------- depende do fato de que o problema possui subestrutura ótima, isto é, que, dada uma solução ótima "S" cuja primeira atividade é a atividade "i", então S \ {i} é uma solução ótima para a entrada composta por todas as atividades que começam a partir do tempo f[i]. Prove, então, que o problema de fato possui a propriedade em questão.

Retomando o raciocínio, observe agora que o primeiro passo da estratégia recursiva acima pode ser realizado de forma eficiente caso as atividades estejam ordenadas pelo tempo de começo; nesse caso, o conjunto das atividades que começam a partir do tempo f[i] é o conjunto {k, k+1, ..., n}, onde "k" é o menor "j" tal que f[i] <= c[j]. Assim, é conveniente realizar uma ordenação preliminar das atividades, o que implica ordenar o vetor "c" em ordem crescente, e realizar a permutação correspondente no vetor "f". Além disso, como o vetor de saída "s" deve ser preenchido de acordo com a numeração original das atividades, é útil manter também um vetor "I" tal que I[j] é o índice original da atividade que, após
a ordenação, passou a ter índice "j" (para obtê-lo, basta preenchê-lo inicialmente com "I[j] := j", e então permutá-lo da mesma maneira que o vetor "f").

Finalmente, observe que, combinando-se as ideias acima, obtém-se um algoritmo recursivo que resolve repetidas vezes uma instância composta pelas atividades {k, ..., n} (para algum k ∈ [1..n]), o que pode ser resolvido facilmente via programação dinâmica.

2. EXERCÍCIO: materialize as ideias acima em dois algoritmos: um que resolva as
   --------- instâncias recursivamente e que faça cálculos repetidos, conforme explicado acima, e outro que utilize programação dinâmica para evitar tal repetição. Qual é a complexidade assintótica do tempo deste último algoritmo?

UMA SOLUÇÃO MAIS ESPERTA

Observando o problema de seleção de atividades acima com mais cuidado, é possível perceber que o algoritmo de programação dinâmica acima faz mais cálculos do que o necessário. De fato:

3. LEMA: Para qualquer entrada c[1..n] e f[1..n], se uma atividade t ∈ [1..n] é
   ---- tal que f[t] <= f[i] ∀i ∈ [1..n], então existe uma solução ótima na
   qual a atividade "t" é selecionada.

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

Sejam "c", "f" e "t" como no enunciado, e seja S uma solução ótima para a entrada em questão. Se t ∈ S, então S é uma testemunha para a afirmação do enunciado e o resultado está provado. Se t ∉ S, seja "p" a primeira atividade de S (observe que S não é vazia, porque nenhuma solução ótima é vazia), e seja T = (S \ {p}) U {t}. Nós argumentamos que T possui apenas atividades compatíveis. De fato, como todas as atividades de S \ {p} estão em S, então elas são todas compatíveis entre si. Logo, resta apenas mostrar que "t" é compatível com todas as atividades de S \ {p}. De fato, seja q ∈ S \ {p}. Pela definição de "p", temos que f[p] <= c[q], e, pela hipótese do enunciado sobre "t", temos que f[t] <= f[p]; logo, f[t] <= c[q], ou seja, "t" e "q" são compatíveis. Logo, "t" é compatível com todas as atividades de S \ {p}, e portanto T possui apenas atividades compatíveis. Por fim, |T| = |S| -1 +1 = |S|, e portanto T também é solução ótima para a entrada em questão. Logo, T é testemunha para o enunciado e o resultado está provado.

==========

O resultado acima, juntamente com o fato de que o problema possui subestrutura ótima, implica que o problema pode ser resolvido simplesmente assim:

1. Selecione a atividade que termina primeiro.
2. Descarte as atividades incompatíveis com a atividade selecionada.
3. Se restarem atividades não descartadas, volte ao passo 1.

Analogamente ao caso do algoritmo de programação dinâmica da seção anterior, é conveniente ordenar as atividades, MAS DESTA VEZ PELO TEMPO DE FINALIZAÇÃO, e não pelo tempo de começo. Nesse caso, a estratégia passa a ser a seguinte:

1. Ordene as atividades em ordem crescente de tempo de finalização.
2. Selecione a primeira atividade.
3. Percorra as demais atividades, por ordem de tempo de finalização; para cada atividade considerada, selecione-a sse ela for compatível com a última atividade anteriormente selecionada.

4. EXERCÍCIO: Escreva um algoritmo O(n*lg n) que materialize a estratégia acima.
   --------- Você não precisa detalhar o passo de ordenação, isto é, o seu algoritmo pode começar com as linhas

   1. PARA i DE 1 A n
   2. | I[i] := i
   3. Ordene o vetor "f" em ordem crescente, e permute os vetores "c" e "I" de forma correspondente. /* De forma que c[i] e f[i] sejam o começo e o fim da atividade cujo índice original era I[i]. */

   As demais linhas do seu algoritmo devem ser normalmente detalhadas (cada execução de uma linha deve usar apenas tempo constante), e o restante do algoritmo deve executar em tempo O(n).

ALGORITMOS GULOSOS

O algoritmo O(n*lg n) acima é um "algoritmo guloso". Uma estratégia gulosa é uma que realize uma escolha que não seja baseada numa completa análise do espaço de soluções. No caso, a escolha gulosa do algoritmo é a de selecionar sempre a atividade que termina primeiro; em contraste, a estratégia de programação dinâmica apresentada anteriormente não realiza uma escolha gulosa, pois nela uma atividade "i" só é selecionada depois de se considerar todas as demais atividades, e de se verificar que ela leva a uma solução com o maior tamanho possível.

Nem todo algoritmo guloso sempre leva a soluções ótimas para um problema. Em alguns casos, porém (como no problema de seleção de atividades em questão), é possível provar que uma certa estratégia gulosa sempre leva a soluções ótimas. Nesses casos, o algoritmo guloso correspondente geralmente executa mais rapidamente que um algoritmo de programação dinâmica, devido a este último percorrer, mesmo que de forma esperta, todo o espaço de soluções.

5. EXERCÍCIOS:

a) O problema de seleção de atividades também pode ser resolvido por meio de grafos. De fato, considere o grafo direcionado em que as atividades são representadas por vértices, e onde um par (a,b) é uma aresta sse f[a] <= c[b]. Esse grafo é acíclico, e além disso a relação de vizinhança é transitiva: se (a,b) e (b,c) são arestas, então (a,c) também o é. Logo, é possível particionar os vértices em camadas: os vértices da camada 1 são aqueles que não são destino de nenhuma aresta; para k > 1, os vértices da camada "k" são aqueles que são destinos de arestas cujas origens estão todas nas camadas 1,...,k-1. Nesse caso:

   1) Mostre que o número de camadas é o tamanho do maior conjunto de atividades compatíveis.

   2) Escreva um algoritmo $O(n^2)$ que resolva o problema de seleção de atividades por meio da estratégia acima. Observe que a construção do grafo pode ser feita em tempo $O(n^2)$, assim como o particionamento dos vértices em camadas.

b) Na estratégia de programação dinâmica apresentada na 2ª seção deste texto, toda vez que o algoritmo considera selecionar a atividade "i", ele precisa da solução da entrada {k,...,n}, onde "k" é a menor atividade "j" tal que f[i] <= c[j]. Descobrir o valor de "k" leva tempo O(n-i+1) caso a busca seja feita de forma linear; entretanto, como os tempos de início estão ordenados, é possível realizar uma busca binária. Isso diminui a complexidade assintótica do tempo do algoritmo? E se não for necessário computar o conjunto de atividades em si, mas apenas o tamanho dele, o tempo do algoritmo diminui, assintoticamente falando?

c) Há outra estratégia de programação dinâmica para resolver o problema. Nela, computa-se, para cada atividade "i", uma solução tão boa quanto possível que inclua a atividade "i", SEM A RESTRIÇÃO DE QUE A ATIVIDADE "i" seja a primeira do conjunto. Nesse caso, é necessário ter computado anteriormente soluções para duas instâncias menores: o conjunto das atividades que terminam até o tempo c[i], e o conjunto das atividades que começam a partir do tempo f[i].

   Escreva então um algoritmo de programação dinâmica que materialize essa estratégia, e que execute em $O(n^3)$. Assim como nos casos anteriores, você pode considerar útil ordenar as atividades.