SCC-218 Alg. Avançados e Aplicações

Lista 1 - Algoritmos Gulosos (*Greedy Algorithms*)

1. O que vc entende por algoritmo guloso? Diz-se que para que um problema aceite uma solução gulosa, este deve possuir estruturas que sejam sub-ótimas e que também tenha propriedade gulosa. Dê exemplos práticos do que isso significa, citando problemas solúveis com estratégia gulosa.

2. Seja um conjunto de pontos em $\mathbb{R}$ tal como: $\{0.7, 1.0, 1.5, 2.0, 2.3, 2.6, 3.1, 3.6, 3.9, 4.2, 4.7, 5.2, 5.5\}$. Determine o menor conjunto em um intervalo fechado de comprimento 1. Para o exemplo acima, os conjuntos de comprimento 1 seriam: $\{[0.7, 1.0, 1.5], [2.0, 2.3, 2.6], [3.1, 3.6, 3.9], [4.2, 4.7, 5.2], [5.5]\}$. Portanto, o menor conjunto seria último desta lista.

   Escreva o melhor algoritmo possível para este problema. Prove que ele está correto e analise sua complexidade computacional.

3. Vc vai dirigir de São Carlos até Santa Cruz de La Sierra. O seu tanque de combustível, quando cheio, tem autonomia para m Km e vc tem um mapa que dá as distâncias entre os postos de combustível em sua rota. Seja $d_1 < d_2 < \ldots < d_n$ o local de todas os postos da rota, onde d_i é a distância a partir de São Carlos. Assuma que a distância entre postos vizinhos é de no máximo m Km.

   Seu objetivo é faze o menor nro de paradas possível no trajeto. Escreva o algoritmo mais eficiente possível para encontrar quais são tais postos. Consegue provar que sua estratégia é ótima? Analise a complexidade dele em função de n.

4. Considere um tabuleiro de xadrez de 100×100. Encontre o menor número possível de movimentos necessários para que um cavalo vá de um canto $[1, 1)]$ para o canto diagonal oposto $[100, 100]$.

   Determine qual a solução gulosa. Consegue mostrar que ela é ótima (retorna o menor nro de movimentos possível)? Dica: pense em distância de Manhattan para enxergar a prova.

5. Seja um grafo conectado G, cujas arestas tenham custos distintos entre si. G tem n vértices e m arestas. Uma aresta particular e em G é dada. Escreva um algoritmo de ordem $\mathcal{O}(m + n)$ que indique se e está presente em uma Árvore Geradora Mínima de G.

6. Considere uma estrada no campo, com casa bastante espalhadas uma das outras. Você pode considerar que esta estrada é uma linha reta com duas extremidades: uma à direção oeste e outra à leste. Estes moradores adoram usar celular e não podem ficar sem sinal. Seu objetivo então é alocar torres de estações ao longo da estrada de forma que toda casa esteja a 4 Kms de uma destas torres. Escreva um algoritmo eficiente, usando o menor número possível de torres.

7. Responda com verdadeiro ou falso. Se verdadeiro, explique; se falso forneça um contraexemplo:

   - Em toda instância do problema de casamento estável, existe um casamento estável contendo um par (m, w) tal que m é ranqueado primeiro na lista de preferências de w e w é ranqueada primeiro na lista de preferência de m;
   - Considere uma instância do problema de casamento estável no qual existe um homem m e uma mulher w tal que m é ranqueado primeiro na lista de preferência de w e w é ranqueada primeiro na lista de preferência de m. Então, em todo casamento estável S para essa instância, o par (m, w) pertence a S.

8. Considere uma cidade com n homens e n mulheres que querem se casar entre si. O conjunto de 2n pessoas é dividido em duas categorias: pessoas boas e pessoas ruins. Suponha que para um determinado número k, $1 \leq k \leq n$, existem k homens bons e k mulheres boas; e portanto $n–k$ homens ruins e $n–k$ mulheres ruins.

   Qualquer um pode casar tanto uma pessoa boa quanto uma ruim. Formalmente, cada lista de preferências possui a propriedade que as pessoas boas do sexo oposto são ranqueadas antes das pessoas ruins do sexo oposto. Portanto, as k primeiras entradas são pessoas boas e as $n–k$ entradas restantes são de pessoas ruins.

   Mostre que em todo casamento estável, todos os homens bons são casados com mulheres boas.

9. Temos umam coleção de *jobs* que precisam ser executados. Para tanto, temos m máquinas idênticas $M_1, M_2, \ldots, M_m$ disponíveis. Executar um job j em qualquer uma delas toma um tempo t_j,onde $t_J > 0$. Nosso objetivo é atribuir jobs às máquinas de forma que o conhecido *makespan*, o tempo necessário para execução de todos os jobs, seja o menor possível.

   Para isso devemos espalhar os jobs da forma mais uniforme possível entre as máquinas. Este é o problema de balancemento de carga. Seja $A(i)$ o conjunto de jobs atribuídos à máquina M_i. O tempo total de ocupação de M_i é:

   $$T_i = \sum_{j \in A(i)} t_j;$$

   e o *makespan* é a atribuição igual a $\max_{1 \leq i \leq m} T_i$. O balanceamento de carga procura encontrar uma atribuição de jobs para as máquinas que minimiza o *makespan*, onde cada jobs é atribuído a uma única máquina.

   Este problema é NP-hard e uma primeira solução para o algoritmo é gulosa: tome os jobs, um a um, e atribua-os à máquina com a menor carga naquele momento. Escreva este algoritmo. Provavelmente sua primeira versão será de complexidade polinomial. Mostre como você pode derrubar a complexidade desta solução gulosa usando minheap no processo. Qual seria esta nova complexidade? A prova para este problema é chata, mas este algoritmo guloso funciona! (não precisa provar este, mas se o fizer contará ponto para ir pro céu)