DCC206 – Algoritmos 1

Aula 11 – Estratégia Gulosa – Parte 2

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Na aula anterior, discutimos as características fundamentais dos algoritmos que implementam a estratégia gulosa
- Vimos que, em alguns casos, a estratégia gulosa pode não encontrar uma solução ótima, ou até mesmo uma solução viável
- Discutimos também as características que certos problemas apresentam que permitem a construção de soluções ótimas gulosas
- Nessa aula, vamos investigar outros problemas práticos onde a estratégia pode ser empregada para obtermos soluções ótimas

Agendamento de tarefas em vários recursos

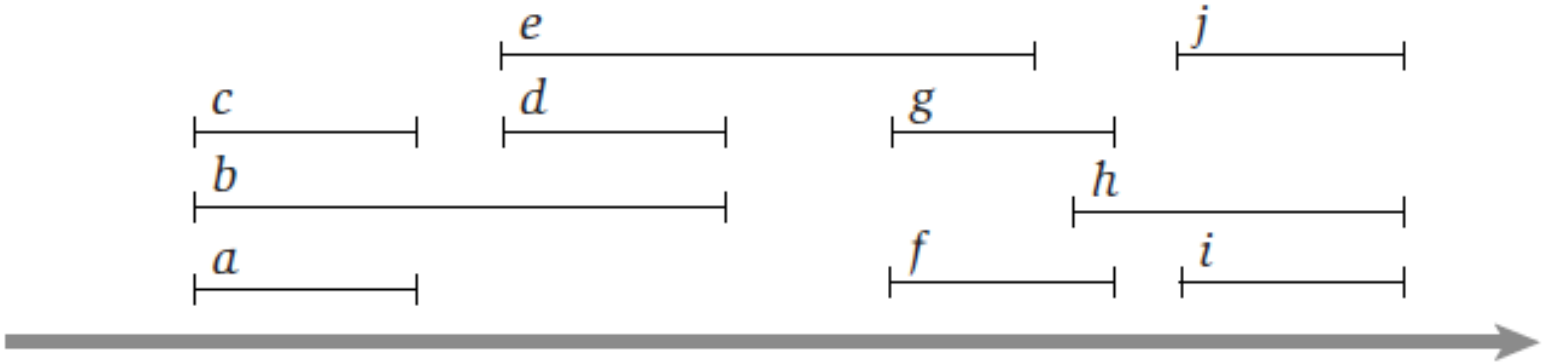
- O problema de agendamento de tarefas visto anteriormente consistia em alocar o máximo de tarefas compatíveis a um recurso (trabalhador) disponível
- Agora imagine um outro problema de agendamento em queremos desejamos agendar todas as tarefas usando o menor número de recursos possível
- Esse poderia ser o caso de termos a obrigatoriedade de executar as n tarefas, porém empregando o menor número de trabalhadores possível

Agendamento de tarefas em vários recursos

- Novamente, recebemos um conjunto de n tarefas
- Cada tarefa tem um horário de início, s(i), e um horário de término, f(i)
- Um trabalhador não pode executar duas tarefas ao mesmo tempo
- Mas dois trabalhadores distintos podem executar tarefas que se sobrepõem
- Podemos supor que temos à disposição quantos trabalhadores forem necessários
- Queremos distribuir as tarefas ao menor número de trabalhadores possível garantindo que todas sejam executadas

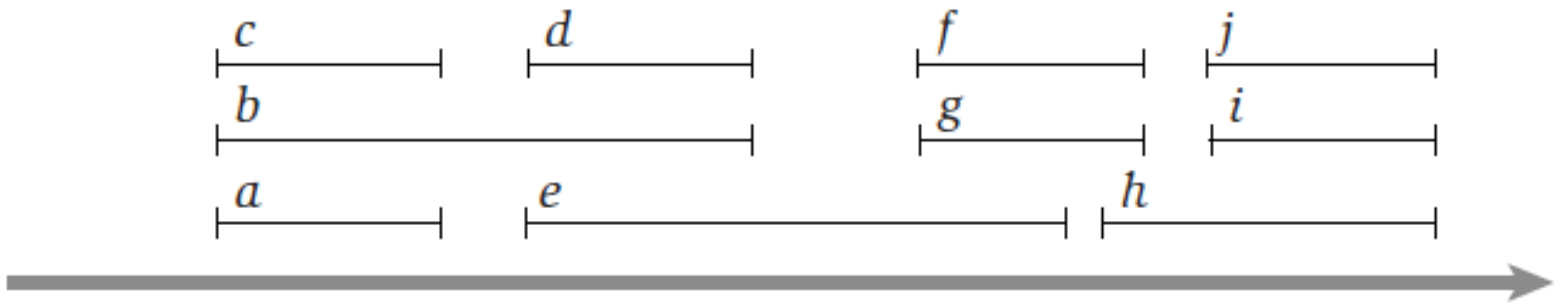
Agendamento de tarefas em vários recursos

- A figura abaixo mostra um exemplo de agendamento
- Nesse exemplo, foram usados quatro trabalhadores
- Esse agendamento é ótimo?



Agendamento de tarefas em vários recursos

- Podemos perceber pela figura abaixo que, se rearranjarmos as tarefas, é possível alocá-las a somente três trabalhadores
- Essa solução é ótima, mas qual característica o problema possui que nos permite chegar a essa conclusão?



Agendamento de tarefas em vários recursos

- Podemos notar que, em um dado instante de tempo, no máximo, três tarefas se sobrepõem
- Definimos o número máximo de tarefas que se sobrepõem em um instante de tempo como a **profundidade** do conjunto
- É fácil perceber que a profundidade é um limite inferior para o número mínimo de trabalhadores necessários para o agendamento
- Dessa forma, se mostrarmos que um agendamento retorna o mesmo número que a profundidade, então demonstramos que a solução é ótima
- A única dúvida que resta é se essa restrição puramente local é suficiente ou se existem outras restrições ao longo do tempo

Agendamento de tarefas em vários recursos

- Vamos desenhar um algoritmo guloso que retorna um agendamento usando um número de trabalhadores igual à profundidade do conjunto
- Assim, ao demonstrar essa propriedade do algoritmo, estaremos provando que sua solução é ótima
- Veja que essa é outra estratégia para demonstrar a otimalidade do algoritmo
- Encontramos um limiar para o problema e mostramos que o algoritmo atinge esse limiar

Agendamento de tarefas em vários recursos

- Suponha que a profundidade de um conjunto seja d
- Queremos distribuir as tarefas entre d trabalhadores
- Assim criamos rótulos {1, 2, ..., d} para atribuir às tarefas
  - Uma tarefa rotulada com t será executada pelo trabalhador t
- A escolha gulosa do nosso algoritmo para uma tarefa i será o primeiro rótulo disponível para essa tarefa
  - Observamos todas as tarefas anteriores incompatíveis com i e descartamos esses rótulos
  - O primeiro que sobrar é atribuído

Agendamento de tarefas em vários recursos

- O algoritmo tem o seguinte comportamento:

1. Ordene as tarefas em ordem de início
2. Para i de 1 até n
    1. D = {1..d}
    2. Para j de 1 até i-1
        1. Se f(j) ≥ s(i), D = D - r(j)
    3. Se |D| > 0, r(i) = D[1]
    4. Senão deixe i sem rotular

Agendamento de tarefas em vários recursos

- **Teorema 1**: O algoritmo guloso atribui um rótulo a toda tarefa e qualquer par de tarefas incompatíveis possui rótulos distintos.
- **Prova**: Primeiro vamos demonstrar que toda tarefa recebe um rótulo. Suponha que existam t tarefas anteriores que são incompatíveis com i. Assim, existem t+1 tarefas que se sobrepõem em um instante de tempo. Como a profundidade é d, sabemos que $t+1 \leq d$, ou seja, $t \leq d-1$. Logo, existe, pelo menos, um rótulo para i.

  Agora, considere duas tarefas i e j, $i \neq j$, que se sobrepõem. Suponha que $s(j) \leq s(i)$ (j vem antes de i na ordenação). Assim, quando o algoritmo for rotular i, o rótulo de j não será considerado. Portanto, elas terão rótulos distintos.

Agendamento de tarefas em vários recursos

- O teorema 1 demonstra que conseguimos agendar as tarefas usando d recursos
  - Ou seja, mostramos que a quantidade de recursos necessária para agendar as tarefas é igual à profundidade do conjunto
- Como a profundidade do conjunto é um limite inferior para o número mínimo de recursos, concluímos que o algoritmo retorna a solução ótima para o problema

Problema da mochila fracionário

- Agora vamos discutir uma solução gulosa para outro problema com grande apelo prático
  - De fato, esse problema é tão importante que voltaremos a discuti-lo em outras aulas durante o curso
- Suponha que queremos transportar n itens em uma carreta com capacidade para W unidades de peso (up).
- Cada item disponível possui um valor (v_i) associado, que é o preço que será pago para transportar w_i ups desse item
- Nossa intenção é maximizar o lucro do transporte sem exceder a capacidade da carreta

Problema da mochila fracionário

- Se restringirmos que os itens só podem ser levados em sua totalidade ou deixados para trás, então o problema não admite solução gulosa (veremos que, nesse caso, o problema é bastante complicado)
- Então, vamos abordar uma variante do problema em que podemos levar uma fração do item
  - Podemos levar uma quantidade $0 \leq x_i \leq w_i$
- Nosso objetivo é encontrar uma solução tal que seu valor seja $\max \sum_{i \in S} x_i v_i$
- Nossa restrição é que $\sum x_i \leq W$

Problema da mochila fracionário

- Podemos pensar em diferentes soluções gulosas para o problema
- Podemos escolher primeiro os itens de menor peso
  - A intuição é ocupar a carreta com mais itens o que potencialmente gera mais lucro
  - No entanto, percebemos que, se tivermos apenas dois itens com peso e valor iguais a (1,10) e (10,200) para levar em uma carreta com capacidade W=10, o algoritmo não retorna a solução ótima
- Alternativamente, poderíamos priorizar os itens com maior valor
  - Novamente, essa escolha não retornaria o ótimo
  - Se tivéssemos W=10 e itens (1,40), (10,50), o algoritmo retornaria 50, mas o ótimo seria 85.
- Uma escolha melhor seria priorizar os itens com melhor custo-benefício
  - Vamos priorizar os itens com maior valor unitário

Problema da mochila fracionário

- Assim chegamos no seguinte algoritmo
- Para i de 1 a n
  - $x_i = 0$
  - $u_i = v_i/w_i$
- c = 0, i = 1
- Ordene os itens em ordem decrescente de u_i
- Enquanto $c < W$ e $i \leq n$
  - $a = \min \{w_i, W-c\}$
  - $x_i = a$
  - $c = c + a$
  - $i = i+1$

Problema da mochila fracionário

- O custo do algoritmo é O(n log n)
- A execução da inicialização e do laço principal tem custo O(n)
- A ordenação domina custo (poderíamos não ordenar e usar um heap máximo, mas o custo total não altera)
- Precisamos demonstrar que a solução encontrada é ótima
- Nesse caso, vamos usar outra estratégia de demonstração chamada de **argumento da troca**
- Nesse tipo de demonstração, partimos de uma solução ótima qualquer e mostramos que, se trocarmos as diferenças entre as duas pela gulosa, obtemos uma solução que não é pior que a ótima

Problema da mochila fracionário

- **Teorema 2**: O algoritmo guloso encontra a solução ótima para o problema da mochila fracionário.
- **Prova**: Suponha que os itens disponíveis estejam ordenados em ordem decrescente de valor unitário. Suponha que não existam dois itens com o mesmo valor unitário.

Seja X = $<x_1, x_2, \ldots, x_n>$ a solução encontrada pelo algoritmo guloso.

Seja O = $<o_1, o_2, \ldots, o_n>$ uma solução ótima para o problema.

Se X = O, ou seja, para todo i, $x_i = o_i$, então não resta nada a demonstrar. Logo, suponha, para fins de contradição, que X não é ótimo e que, portanto, X ≠ O. Assim, $\sum x_i = \sum o_i = W$. A capacidade tem que ter sido exaurida, caso contrário a solução gulosa seria ótima (simplesmente levar todos os itens).

Seja i o primeiro item para o qual as quantidades escolhidas na solução gulosa e na ótima diferem ($x_i \neq o_i$). Pela escolha gulosa, sabemos que $x_i > o_i$. Mas, como os pesos transportados são iguais em ambas as soluções, deve existir j > i tal que $x_j < o_j$.

Seja d = min $\{x_i - o_i, o_j\}$. Podemos derivar uma nova solução Q a partir de O diminuindo a quantidade do item j e aumentando a quantidade do item i em d unidades. Ou seja, Q = $<o_1,\ldots,o_i+d,\ldots,o_j-d,\ldots,o_n>$.

É perceptível que não alteramos o peso transportado em Q. Contudo, como o valor unitário de i é maior que o de j, obtemos uma nova solução com valor estritamente maior que O. Isso é uma contradição, já que O é ótima.

Logo, a escolha gulosa não gera uma solução pior que uma solução ótima arbitrária. Portanto, ela é ótima.

Leitura

- Seção 4.1 Kleinberg e Tardos
- Seção 10.1 Goodrich e Tamassia