DCC206 – Algoritmos 1

Aula 10 – Estratégia Gulosa

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- A partir dessa aula, vamos discutir diferentes tipos de paradigmas para desenho de algoritmos
- Os algoritmos de um certo paradigma compartilham características na forma como a solução de um problema é encontrada
- Vamos iniciar por um paradigma do qual já vimos vários representantes: paradigma guloso
- Vimos que esses algoritmos compartilhavam a característica de sempre fazerem melhores escolhas imediatas sem considerarem o todo
- Posteriormente, demonstramos que o algoritmo efetivamente encontrava a solução para o problema

Introdução

- Apesar de já termos visto alguns representantes, vamos iniciar nossa discussão com um problema bastante simples: fornecer o troco em moedas
- Imagine que precisamos fornecer o troco no valor de R$0,85 para uma compra realizada em dinheiro
- Como dinheiro físico é algo raro hoje em dia, queremos minimizar o número de moedas devolvidas no troco

Introdução

- As moedas disponíveis hoje no Brasil são

- Assim, como podemos resolver o problema?
- Existe uma solução amplamente usada no mercado:
  - Iniciando pela moeda de maior valor, selecione o máximo possível sem exceder o troco
  - Subtraia o valor do troco
  - Repita o processo com o restante do troco

Introdução

- Esse algoritmo do senso comum possui daqueles empregados na obtenção da árvore geradora mínima e do caminho mais curto
- A cada iteração, fazemos a melhor escolha imediata sem nos importarmos com o todo
  - Selecionamos a moeda que naquele momento parecia fornecer a melhor solução, ou seja, minimizava a quantidade total de moedas a serem devolvidas no troco
- Anteriormente, vimos que as soluções gulosas estudadas encontravam a solução ótima
- Será que o mesmo se aplica nesse caso?

Introdução

- Para o troco em questão, R$0,85, selecionamos uma moeda de R$0,50, uma moeda de R$0,25 e uma moeda de R$0,10
- É perceptível que, nesse caso, o algoritmo realmente encontra a solução ótima
- Contudo, será que ele encontra o ótimo em qualquer conjunto de moedas (com diferentes valores)?
- A resposta é **não**!

Introdução

- Suponha que o presidente do Banco Central decidiu mudar os valores das moedas cunhadas no Brasil
- Agora elas terão os seguintes valores: R$1,00, R$0,70, R$0,34, R$0,21, R$0,10, R$0,01.
- Se tivéssemos que devolver um troco de R$1,40, o algoritmo guloso retornaria a solução ótima?
- Seguindo o algoritmo, ele retornaria uma moeda de R$1,00, uma moeda de R$0,34, e seis moedas de R$0,01
- No entanto, a solução ótima seria duas moedas de R$0,70

Introdução

- Em resumo, em alguns casos, mesmo que o algoritmo retorne o ótimo para uma instância do problema, ele pode falhar em outras instâncias
  - Exceto quando é demonstrado que ele encontra a solução ótima em todos os casos
- Na verdade, em casos mais extremos, o algoritmo pode até não encontrar uma solução válida
  - Para um troco de 6 tendo apenas moedas 4 e 3

Problemas de otimização

- Implicitamente, apresentamos uma estratégia que funciona para um certo tipo de problema: otimização
- Em problemas de otimização, possuímos uma função que afere a qualidade de uma solução
- Essa função é chamada de **função objetivo**
- Nesse tipo de problema, queremos encontrar uma solução que minimize (maximize) o valor dessa função
  - Isto é, queremos encontrar uma solução que possua o menor (maior) valor dentre todas as soluções válidas
  - Essa solução é chamada de **solução ótima**
- Naturalmente, existem muitas soluções candidatas. O conjunto de soluções candidatas é chamado de **espaço de busca** do problema

Estratégia Gulosa

- Na estratégia gulosa, percorremos o espaço de busca construindo a solução a partir de uma sequência de escolhas (gulosas)
- A construção da solução inicia de uma candidata mais simples conhecida
  - Iniciamos com um conjunto vazio de moedas
- A cada passo, o algoritmo escolhe o melhor refinamento (em termos da função objetivo) dentre todos os disponíveis naquela iteração
  - Escolhemos a moeda com o maior valor que não excede o valor atual do troco

Estratégia Gulosa

- Como visto, para alguns problemas, a estratégia gulosa não encontra a solução ótima
- Os problemas para os quais ela fornece a solução ótima possuem uma propriedade chamada de **escolha gulosa**
- A propriedade de escolha gulosa estabelece que:
  - A solução ótima é construída a partir de escolhas ótimas locais
  - A escolha local deve depender apenas da configuração atual do problema e não de futuras escolhas
  - As escolhas realizadas não são reconsideradas

Estratégia Gulosa

- Os problemas que admitem solução gulosa ótima também devem possuir outra característica chamada de **subestrutura ótima**
- Um problema apresenta subestrutura ótima se a solução ótima contém soluções ótimas para os seus subproblemas
  - Em caminhos mais curtos, o caminho mais curto de v para u inclui o menor caminho entre quaisquer pares de vértices nesse percurso ótimo

Estratégia Gulosa

- A demonstração de que um algoritmo guloso encontra a solução ótima para um problema explora essas duas propriedades
- A propriedade de escolha gulosa é usada para mostrar que qualquer outra escolha usada para encontrar uma solução ótima resultaria em uma solução possivelmente diferente, mas não melhor do que a escolha gulosa
- A subestrutura ótima é explorada para mostrar (normalmente por indução) que a combinação de soluções (ótimas) para subproblemas anteriores com a escolha gulosa resultam em uma solução ótima global
- Vamos avaliar esses passos na construção da solução gulosa para um problema prático

Agendamento de tarefas

- Suponha que desejamos montar a agenda de um trabalhador de forma a maximizar o total de tarefas executadas
- Cada tarefa possui um horário de início e fim, denotados, respectivamente, por s(i) e f(i)
  - Cada tarefa é executada no intervalo aberto de tempo [s(i),f(i))
- Foram requisitadas a execução de n tarefas. Contudo, algumas possuem sobreposição de horários e, dessa forma, não podem ser ambas executadas (as tarefas não podem mudar de horário, nem o trabalhador pode executar mais do que uma tarefa ao mesmo tempo)
- Duas tarefas i e j são compatíveis se $s(i) \geq f(j)$ ou $f(i) \leq s(j)$
- Nosso objetivo é maximizar o número de tarefas compatíveis a serem executadas pelo trabalhador

Agendamento de tarefas

- Vamos explorar algumas opções para construir nossa abordagem gulosa e vamos analisar a qualidade dessa solução
- A primeira solução é a mais intuitiva: iniciamos a escolha das atividades a partir da tarefa com o horário de início mais cedo entre todas
  - Como queremos maximizar a ocupação do nosso trabalhador, o mais natural é iniciar os trabalhos tão logo tenha uma tarefa a ser realizada
- Naturalmente, ao escolhermos a tarefa i com o menor s(i), devemos descartar todas as tarefas j tais que $s(j) < f(i)$
- Agora, repetimos o raciocínio com as tarefas restantes

Agendamento de tarefas

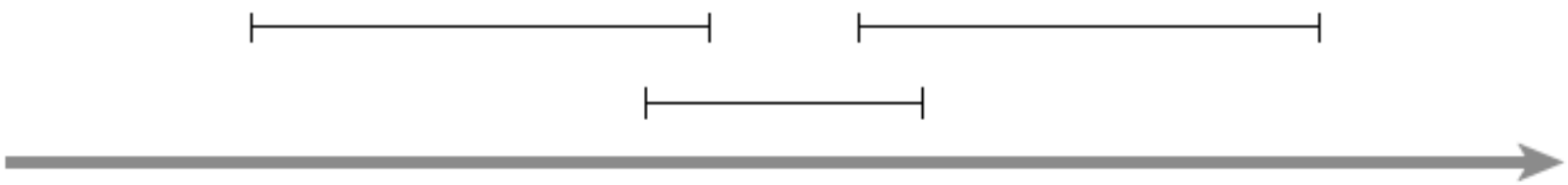
- Essa escolha não resulta em uma solução ótima
- Considerando a figura abaixo, o algoritmo escolheria a última tarefa, enquanto o ótimo seria selecionar as quatro de cima

Agendamento de tarefas

- O problema da escolha anterior é que a primeira tarefa pode ter a maior duração entre todas elas
- Assim, a escolha dessa tarefa inviabilizaria todas as outras
- Logo, poderíamos alterar nossa escolha para escolhermos primeiro as tarefas de menor duração
- O algoritmo então seria:
  - Escolha a tarefa i de menor duração
  - Descarte todas as tarefas j tais que $[s(i) \leq s(j) < f(i)]$ ou $[s(i) \leq f(j) \leq f(i)]$
  - Repita o processo com as tarefas restantes

Agendamento de tarefas

- Esse algoritmo também não encontra a solução ótima
- A escolha da tarefa de menor duração (a do meio) impede a escolha das outras duas
  - A solução ótima seria justamente escolhê-las

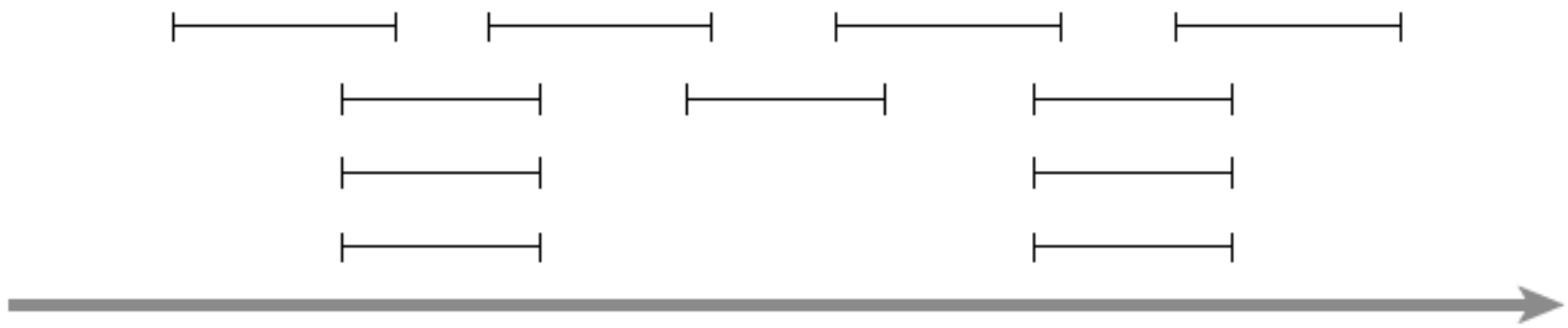


Agendamento de tarefas

- O problema da solução é que a tarefa escolhida competia com muitas outras tarefas
- Dessa forma, poderíamos modificar o algoritmo para priorizar aquelas que são compatíveis com o maior número de tarefas
- O algoritmo então seria:
  - Para cada tarefa, calcule o número de tarefas com as quais ela é incompatível
  - Escolha a tarefa com o menor número de tarefas incompatíveis
  - Descarte todas as tarefas com as quais ela é incompatível
  - Repita o processo para as demais

Agendamento de tarefas

- Essa solução também não encontra o ótimo
- O algoritmo escolhe primeiro a tarefa central da segunda linha
- Essa escolha inviabiliza a escolha das outras duas de cima
- Dessa forma, no máximo, três tarefas serão executadas
- A solução ótima seria escolher as quatro de cima



Agendamento de tarefas

- Vamos explorar outra ideia bastante natural: escolher a tarefa com o horário de término mais cedo entre todas
- A intuição nesse caso é deixar o trabalhador livre para executar outras tarefas o mais cedo possível
- Ou seja, maximizamos o tempo restante para ele executar outras tarefas
- Vamos formalizar esse algoritmo
  - Denotaremos com R o conjunto de tarefas restantes
  - A será o conjunto de tarefas escolhidas

Agendamento de tarefas

- $R = \{1 .. n\}$
- $A = \emptyset$
- Enquanto $R \neq \emptyset$
  - Escolha $i \in R$ com menor $f(i)$
  - $A = A \cup \{i\}$
  - Remova de R todas as tarefas incompatíveis com i
- Retorne A

Agendamento de tarefas

- Vamos analisar o algoritmo para verificar se ele encontra uma solução ótima
- É perceptível que o conjunto A contém apenas tarefas compatíveis
- Assim, devemos nos concentrar em demonstrar que |A| = |A*| para uma solução ótima A*
- Nesse caso, como mencionado anteriormente, devemos tentar verificar a propriedade de escolha gulosa e subestrutura ótima do problema

Agendamento de tarefas

- Suponha que |A| = k e que as tarefas foram adicionadas na seguinte ordem i_1, i_2, ..., i_k
- Similarmente, suponha que |A*| = m e que as tarefas são j_1, j_2, ..., j_m na ordem natural da esquerda para direita
  - Note que a ordem é a mesma se considerarmos s(j) ou f(j)
- Nosso objetivo é provar que k=m
- Pela construção do nosso algoritmo, sabemos que $f(i_1) \leq f(j_1)$
- Vamos demonstrar agora que, para r≥1, a r-ésima tarefa escolhida pela algoritmo guloso não termina depois da tarefa equivalente na solução ótima

Agendamento de tarefas

- **Lema 1**: Para todo $1 \leq r \leq k$, $f(i_r) \leq f(j_r)$.
- **Prova**: Vamos provar por indução em r. Claramente, para r=1, o lema é verdadeiro pela construção do algoritmo guloso.

Agora assuma como hipótese de indução que o lema é válido para r-1. Logo, pela HI, $f(i_{r-1}) \leq f(j_{r-1})$. Para que o lema fosse falso para r tarefas, a tarefa i_r deveria terminar depois da tarefa j_r , i.e. $f(i_r) > f(j_r)$. Pela propriedade do problema, sabemos que $f(i_{r-1}) \leq f(j_{r-1}) \leq s(j_r)$. Logo, j_r é compatível com as tarefas anteriores de A. Pela escolha gulosa, essa tarefa teria sido escolhida ao invés de i_r. Portanto, $f(i_r) \leq f(j_r)$.

Agendamento de tarefas

- **Teorema**: O algoritmo guloso retorna uma solução ótima A.
- **Prova**: A prova será por contradição. Assim, suponha que A não seja ótimo. Dessa forma, existe uma solução ótima A* com mais tarefas que A, $|A^*|=m > k=|A|$.
- Pelo lema 1 com r=k, temos que $f(i_r) \leq f(j_r)$. Como m>k, existe uma tarefa $j_{r+1} \in A^*$. Como as tarefas em A* são compatíveis, $f(i_r) \leq f(j_r) \leq s(j_{r+1})$. Logo, a tarefa j_{r+1} seria compatível com as tarefas de A. Contudo, assumimos que o algoritmo teria parado na k-ésima tarefa mesmo existindo tarefas para serem avaliadas em R. Contradição!

Agendamento de tarefas

- Podemos implementar esse algoritmo ordenando as tarefas pelo tempo de término com custo O(n log n)
- Com tempo O(n) construímos um vetor S contendo os tempos s(i) para corresponder à ordenação das tarefas
- Agora iteramos sobre as tarefas para escolher a de menor tempo de término, inicialmente i=1
- Inserimos a tarefa i no conjunto A
- Percorremos o vetor S iniciando pela posição j=i+1 e terminando com um j tal que $S[j] \geq f(i)$
- Fazemos i=j e repetimos o processo
- Dessa forma, com custo O(n), inserimos todas as tarefas no conjunto A. Assim, o custo total do algoritmo é O(n log n).

Leitura

- Seção 4.1 Kleinberg e Tardos