Introdução à Programação Dinâmica

Conceitos e Trabalho Prático

Prof. Me. Lucas Bruzzone

Aula 13

Recapitulando: Limitações do Guloso

Vimos que algoritmos gulosos:

- São rápidos e simples
- Mas nem sempre produzem solução ótima
- Problema do troco com moedas $\{1, 3, 4\}$ e valor 6

A pergunta natural:

Como garantir solução ótima quando guloso falha?

Opções:

- Força bruta: $O(2^n)$ ou $O(n!)$ - inviável
- Programação Dinâmica: $O(n^2)$ ou $O(n^3)$ - viável

O que é Programação Dinâmica?

Definição:

- Técnica para resolver problemas de otimização
- Divide problema em subproblemas menores
- Resolve cada subproblema uma única vez
- Armazena resultados para reutilização

Ideia central:

"Não recalcular o que já foi calculado"

Diferenças das outras técnicas:

- Força bruta: recalcula tudo sempre
- Guloso: faz uma escolha e não reconsidera
- PD: calcula tudo uma vez e armazena

Quando Usar Programação Dinâmica?

Problema deve ter duas propriedades:

1. **Subestrutura Ótima**
   - Solução ótima contém soluções ótimas dos subproblemas
   - Podemos construir solução maior a partir de menores

2. **Sobreposição de Subproblemas**
   - Mesmos subproblemas são resolvidos várias vezes
   - Vale a pena armazenar resultados

Se ambas existem: Programação Dinâmica é aplicável!

Exemplo Clássico: Fibonacci

Definição recursiva:

$$F(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

Implementação recursiva ingênua:

- $F(5)$ chama $F(4)$ e $F(3)$
- $F(4)$ chama $F(3)$ e $F(2)$
- $F(3)$ é calculado múltiplas vezes!
- Complexidade: $O(2^n)$ - exponencial

Problema: Recalculamos os mesmos valores repetidamente

Fibonacci com Programação Dinâmica

Solução: Armazenar resultados já calculados

```
FibonacciPD(n):
    dp = array[0..n]
    dp[0] = 0
    dp[1] = 1

    para i de 2 ate n:
        dp[i] = dp[i-1] + dp[i-2]

    retornar dp[n]
```

Complexidade:

- Tempo: $O(n)$ - cada valor calculado uma vez
- Espaço: $O(n)$ - array para armazenar

Melhoria dramática: De $O(2^n)$ para $O(n)$!

Abordagens de Programação Dinâmica

Duas maneiras principais:

1. **Top-Down (Memoização)**
   - Começa do problema original
   - Recursão natural + cache de resultados
   - Calcula apenas o necessário

2. **Bottom-Up (Tabulação)**
   - Começa dos casos base
   - Iterativo, constrói tabela
   - Calcula todos os subproblemas

Neste curso: Focaremos em Bottom-Up (mais comum e direto)

Estrutura Geral de Solução PD

Passos para resolver com PD:

1. **Caracterizar a estrutura** da solução ótima
2. **Definir recursivamente** o valor da solução
3. **Calcular o valor** da solução (bottom-up)
4. **Construir a solução** a partir dos valores

Elementos essenciais:

- **Estado**: O que representa dp[i]?
- **Casos base**: Valores iniciais conhecidos
- **Transição**: Como calcular dp[i] a partir de valores anteriores?
- **Resposta final**: Onde está o resultado?

Relembrando: Troco com PD

Problema: Moedas $\{1, 3, 4\}$, dar troco de 6

Definição do estado:

- $dp[i]$ = mínimo de moedas para valor i

Recorrência:

$$dp[i] = \min_{m \in \text{moedas}, m \leq i} \{dp[i - m] + 1\}$$

Casos base:

- $dp[0] = 0$ (não precisa moedas para valor 0)

Resposta: $dp[6]$

Troco: Construção da Tabela

Moedas: $\{1, 3, 4\}$, Valor: 6

i	0	1	2	3	4	5	6
dp[i]	0	1	2	1	1	2	2

Cálculos importantes:

- $dp[1] = dp[0] + 1 = 1$ (usando moeda 1)
- $dp[3] = \min(dp[2] + 1, dp[0] + 1) = 1$ (usando moeda 3)
- $dp[4] = \min(dp[3] + 1, dp[1] + 1, dp[0] + 1) = 1$ (usando moeda 4)
- $dp[6] = \min(dp[5] + 1, dp[3] + 1, dp[2] + 1) = 2$ (3+3)

Resultado: 2 moedas (solução ótima: $\{3, 3\}$)

Comparativo Final: 3 Técnicas

Critério	**Força Bruta**	**Guloso**	**PD**
Complexidade	Exponencial	$O(n \log n)$	Polinomial
Garantia ótimo	Sim	Às vezes	Sim
Uso memória	Baixo	Baixo	Médio/Alto
Facilidade	Simples	Simples	Médio
Quando usar	n pequeno	Canônico	Geral

Conclusão:

- PD é o "meio termo" ideal
- Garante solução ótima
- Complexidade aceitável
- Requer mais cuidado na implementação

Trabalho: Comparação Guloso vs PD

Objetivo:

- Implementar solução gulosa para um problema
- Implementar solução com PD para o mesmo problema
- Comparar resultados e análise de complexidade
- Apresentar para a turma

Estrutura da apresentação :

1. Descrição do problema
2. Solução gulosa proposta
3. Solução com PD
4. Comparação de resultados em diferentes instâncias
5. Análise de complexidade temporal e espacial
6. Conclusões: quando cada abordagem é melhor

Problemas Disponíveis - Parte 1

1. **Problema do Troco**
   - Guloso: maior moeda primeiro
   - PD: mínimo para cada valor
   - Testar com moedas canônicas e não-canônicas

2. **Mochila 0/1 vs Fracionária**
   - Guloso: resolve fracionária (maior razão valor/peso)
   - PD: resolve 0/1 (itens completos)
   - Comparar resultados das duas versões

3. **Escalonamento de Tarefas com Pesos**
   - Guloso: escolher por maior peso
   - PD: combinação ótima considerando compatibilidade
   - Mostrar quando guloso falha

Problemas Disponíveis - Parte 2

4. Corte de Hastes

- Guloso: sempre cortar no tamanho de maior valor unitário
- PD: solução ótima para maximizar lucro
- Exemplo comercial interessante

5. Partição de Conjunto

- Guloso: adicionar ao grupo de menor soma
- PD: verificar se existe partição perfeita
- Comparar qualidade das soluções

6. Subsequência Crescente Mais Longa (LIS)

- Guloso: heurística (primeira crescente encontrada)
- PD: solução ótima
- Diferença clara nos resultados

Problemas Disponíveis - Parte 3

7. Escalonamento com Deadlines

- Guloso: ordenar por penalidade
- PD: considerar todas possibilidades
- Aplicação em gerenciamento de projetos

8. Compra de Ações

- Guloso: comprar no mínimo, vender no máximo seguinte
- PD: múltiplas transações de forma ótima
- Problema prático e motivador

9. Subsequência Comum Mais Longa (LCS)

- Guloso: heurística baseada em matches
- PD: solução ótima com matriz
- Aplicação: diff de arquivos, bioinformática

Problemas Disponíveis - Parte 4

10. Problema da Editação de Strings

- Guloso: substituir caracteres diferentes sequencialmente
- PD: mínimo de operações (inserção, remoção, substituição)
- Aplicação: corretor ortográfico

Observação:

- Todos os problemas permitem comparação entre as técnicas
- Alguns problemas o guloso funciona parcialmente
- Outros o guloso falha completamente
- Escolham baseado no interesse do grupo

Detalhamento dos Problemas

Material de apoio:

- Especificação completa no AVA
- Exemplos de entrada e saída
- Sugestões de casos de teste

Dica importante:

- Comecem testando com instâncias pequenas
- Implementem força bruta primeiro para validar
- Comparem os três: força bruta, guloso e PD
- Documentem bem os contra-exemplos

Organização dos Grupos

Formação:

- Grupos de 3-4 pessoas
- Cada grupo escolhe um problema diferente
- Não pode repetir problema entre grupos

Entregas:

- Código implementado (guloso + PD)
- Slides para apresentação
- Relatório breve (1-2 páginas) com análise

Cronograma:

- Hoje: Formação dos grupos e escolha do problema
- Próximas aulas: Desenvolvimento e dúvidas
- 01/12: Apresentações

Critérios de Avaliação

Implementação (40%):

- Corretude dos algoritmos
- Qualidade do código
- Casos de teste adequados

Análise (30%):

- Comparação clara entre as abordagens
- Análise de complexidade correta
- Discussão de quando usar cada técnica

Apresentação (30%):

- Clareza na explicação
- Exemplos bem escolhidos
- Gestão do tempo

Atividade de Hoje

Vamos formar os grupos agora!

Processo:

1. Formem grupos de 3-4 pessoas (10 min)
2. Cada grupo escolhe um problema da lista
3. Informem o problema escolhido para registro
4. Recebam especificação detalhada
5. Início do desenvolvimento (resto da aula)