DCC206 – Algoritmos 1

Aula 15 – Programação Dinâmica

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Até o momento, estudamos as estratégias de desenho de algoritmos dividir-e-conquistar e gulosa
- Essas estratégias são úteis para solucionar diversos problemas práticos
- No entanto, elas não são a solução para todos os problemas
  - Alguns problemas, como vimos, não admitem solução ótima gulosa
  - A divisão de alguns problemas em subproblemas para depois combinar as soluções não necessariamente é mais rápido
- Estudaremos, nessa aula, uma nova estratégia para desenho de algoritmos chamada **programação dinâmica**

Introdução

- A técnica de programação dinâmica foi proposta pelo matemático estadunidense Richard Bellman nos anos 1950s para solucionar problemas de otimização
- Ele usou a palavra programação para se referir ao ato de planejar as ações a serem tomadas, e não ao ato de escrita de código de computador
- Apesar de conceitualmente diferente, programação dinâmica assemelha-se a dividir-e-conquistar por resolver problemas maiores através de subproblemas

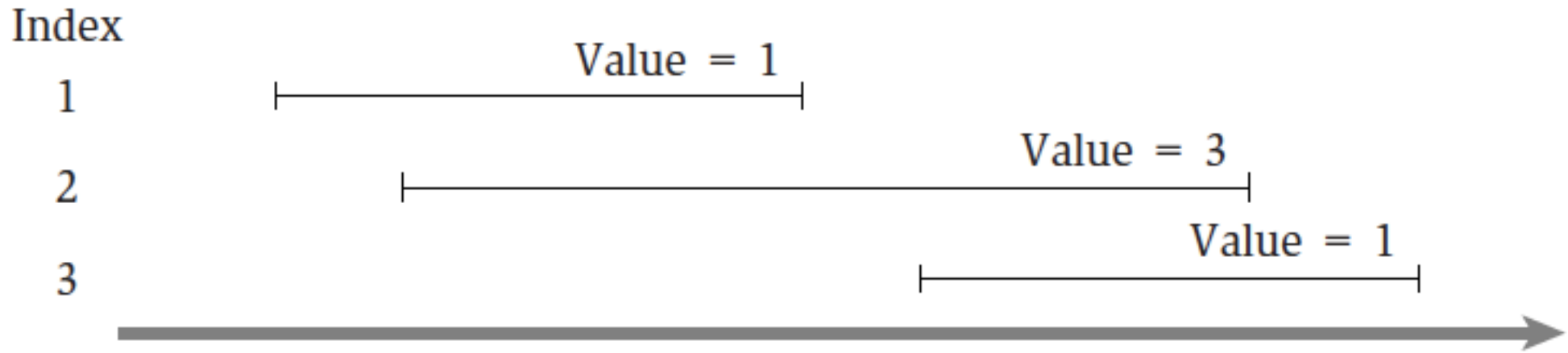
Introdução

- A ideia geral da estratégia dividir-e-conquistar é particionar o problema original em problemas menores independentes, resolvê-los e combinar os resultados para obter a solução global
- A programação dinâmica é útil nos casos em que os subproblemas não são independentes
- Exemplo:
  - Sequência de Fibonacci: F(n) = F(n-1) + F(n-2)
  - Os subproblemas não são independentes. Logo, muitos são resolvidos diversas vezes, repetindo esforço e gerando um custo desnecessário
- A programação dinâmica faz uso de tabelas para contornar esse problema

Problema do agendamento de tarefas

- O entendimento da programação dinâmica é mais fácil através de exemplos.
- Para isso, vamos desenvolver o raciocínio com base em um problema de agendamento de tarefas visto anteriormente no contexto de algoritmos gulosos
- Suponha que o problema seja necessário escolher, dentre diversas tarefas, aquelas que geram o maior valor agregado
- Toda tarefa tem um horário de início e fim
  - Duas tarefas são compatíveis se elas não podem se sobrepõem
- Ao contrário do que acontecia antes, toda tarefa está associada a um valor (lucro, prioridade, …)
- O objetivo é escolher um conjunto máximo de tarefas compatíveis que maximizem a soma desses valores

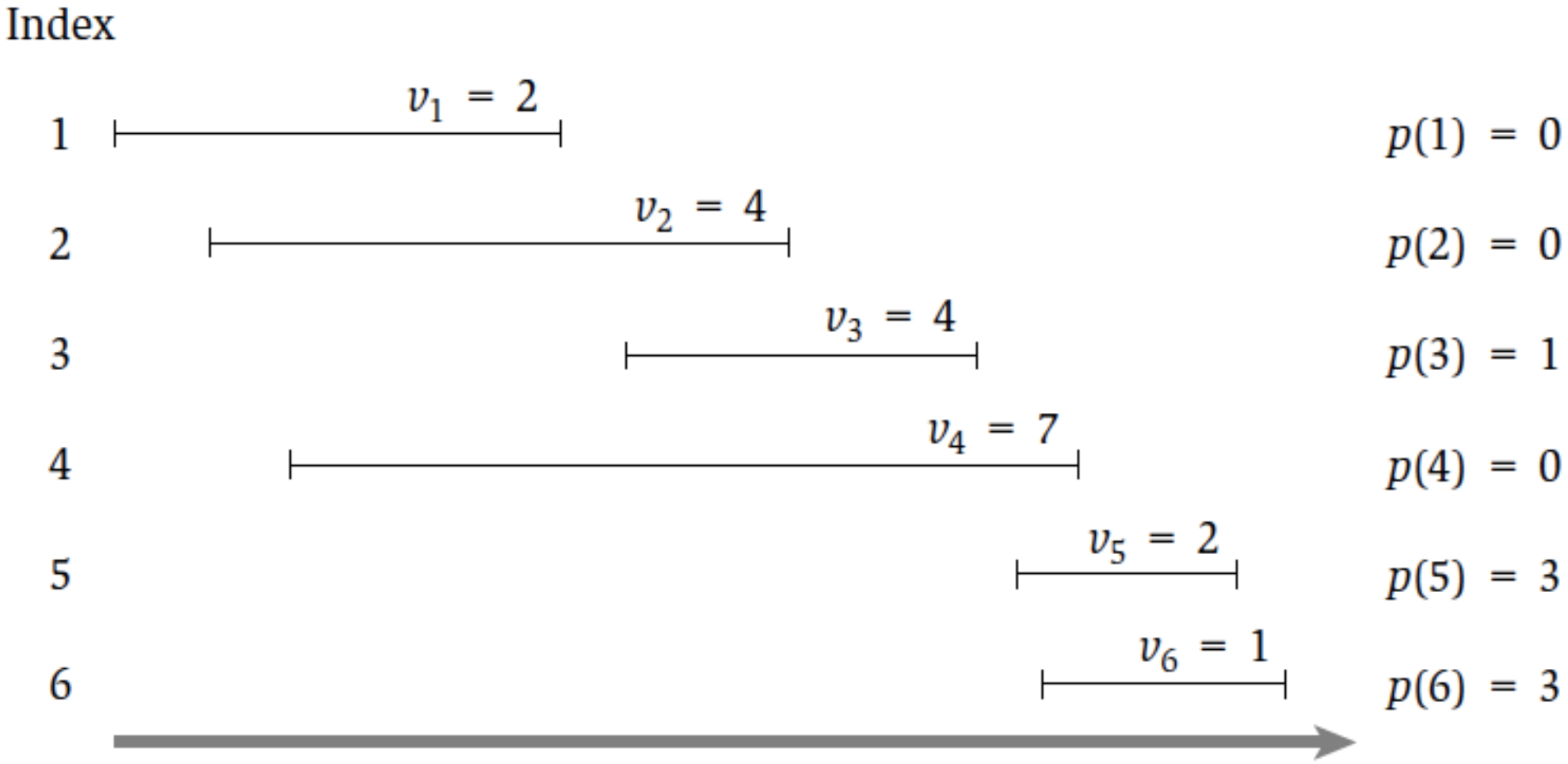
Problema do agendamento de tarefas



Problema do agendamento de tarefas

- Se rotularmos as tarefas de 1 a n ordenadas pelo horário de término, o problema é:
  - Encontrar $S \subseteq \{1, 2, \ldots, n\}$ tal que $\sum_{i \in S} v(i)$ é máximo
  - Onde v(i) é o valor do compromisso i
  - Para todo i, f(i) < s(i+1)
- Para solucionar o problema, definimos uma função p(j) como sendo o maior valor de i tal que i < j (f(i) < s(j))
  - p(j) = 0 se não existir um compromisso com fim anterior ao início de j

Problema do agendamento de tarefas



Problema do agendamento de tarefas

- Seja S* uma solução ótima para o problema
- O último compromisso pode ou não pertencer a S*
- Se n pertencer a S*, então nenhum compromisso entre p(n) e n pode pertencer a S*
- Além disso, se n pertencer a S*, então S* deve conter uma solução ótima para o subproblema {1, 2, …, p(n)}
- Similarmente, se $n \notin S^*$, então S* deve conter uma solução ótima de {1,2,…,n-1}

Problema do agendamento de tarefas

- As observações anteriores nos levam a definir o valor S(j) da solução ótima para {1,2,…,j} como
  - $S(j) = \max\left(v(j) + S\left(p(j)\right), S(j-1)\right)$
  - S(0) = 0
- Podemos derivar uma função recursiva diretamente dessa definição para computar o valor máximo
- Também podemos afirmar se j pertence ou não à solução se v(j) + S(p(j)) ≥ S(j-1)

Problema do agendamento de tarefas

- O problema de tal algoritmo recursivo é que, assim como a versão recursiva de Fibonacci, vários S(i) são computados repetidas vezes
  - A complexidade do algoritmo se torna exponencial
- Podemos modificar o algoritmo para ser um pouco mais ‘esperto’ *memoizando* resultados já computados
  - Criamos uma tabela para armazenar S(i)
  - A primeira vez que S(i) for necessário, computamos seu valor
  - Na próxima vez, retornamos o valor presente na tabela

Problema do agendamento de tarefas

- A tabela contém um valor para cada $1 \leq i \leq n$, além de zero
  - Logo, preenchê-la tem um custo O(n)
- Computados os valores de S(i), podemos computar a solução efetiva, mostrando as tarefas a serem cumpridas
- Tarefas(j)
  - Se j = 0, retornar
  - Senão
    - Se $v(j) + S(p(j)) \geq S(j-1)$, Tarefas(p(j)); mostrar “j”
    - Senão Tarefas(j-1)

- A função Tarefas também tem custo O(n)

Programação dinâmica

- O problema anterior destaca a aplicabilidade de programação dinâmica
- Através do simples armazenamento de soluções parciais, foi possível reduzir drasticamente a complexidade de tempo do algoritmo
- Ainda podemos refinar o algoritmo anterior
  - Podemos transformá-lo em um algoritmo iterativo computando os valores a partir de zero (bottom-up)
- A estratégia de programação dinâmica é exatamente essa ideia

Programação dinâmica

- O desenvolvimento de um algoritmo de programação dinâmica envolve em geral quatro etapas
  1. Caracterizar a estrutura de uma solução ótima
  2. Definir uma relação de recorrência para o valor de uma solução ótima
  3. Calcular o valor de uma solução ótima por um procedimento bottom-up
  4. Construir uma solução ótima a partir dos valores computados

Programação dinâmica

- A aplicabilidade de programação dinâmica em geral depende da identificação das seguintes características
  - Existe apenas um número polinomial de subproblemas
  - A solução do problema original deve ser facilmente computada através das soluções dos subproblemas
  - Existe uma ordem natural sobre os subproblemas, permitindo resolvê-los do ‘menor’ para o ‘maior’

Leitura

- Seções 6.1 – 6.3 (Kleinberg e Tardos)