

# PAA - Aula 17

## Programação Dinâmica, Conjunto Independente de Peso Máximo em Caminhos

Vamos começar a estudar a técnica de projeto de algoritmos

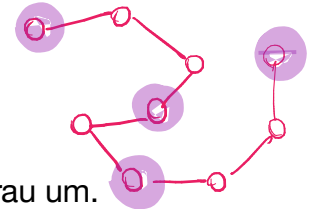
- chamada **Programação Dinâmica**.

Para tanto, vamos abordar um problema bastante particular e didático,

- o **Conjunto Independente** de Peso Máximo em Grafos Caminhos.

Entrada: um grafo caminho  $G=(V,E)$  com

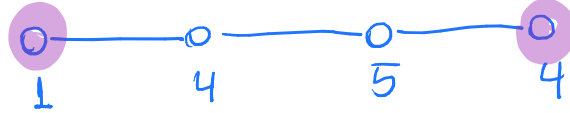
- pesos não negativos nos vértices,  $w: V \rightarrow \mathbb{R}^+$
- Um grafo caminho é um grafo conexo sem bifurcações, ou seja,
  - o grau de todos os vértices é dois,
    - exceto pelos dois vértices dos extremos que têm grau um.



Saída: um conjunto independente de peso máximo.

- Um conjunto é **independente** se nenhum par de vértices é adjacente,
  - i.e., se nenhum par do conjunto tem aresta em comum.

Exemplo: Caminho com quatro vértices e pesos 1, 4, 5, 4



$$1 + 4 = 5$$

Antes de projetar um algoritmo usando a nova técnica,

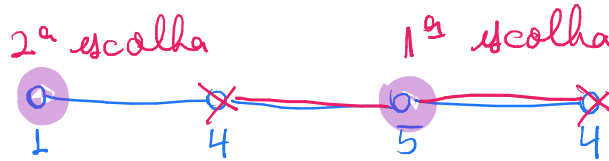
- vamos testar as técnicas que já conhecemos,
  - para entender as dificuldades do problema
  - e limitações das técnicas.

Abordagem por força bruta:

- Vamos **enumerar** todos os  $2^n$  diferentes subconjuntos,
  - **descartar** todos os que tem dois vértices adjacentes,
  - e **encontrar** o de peso máximo dentre os que sobraram
- Vantagem: Esta abordagem encontra o resultado correto.
- Desvantagem: Ela leva tempo  $\Omega(2^n)$

## Abordagem gulosa:

- Dentre várias possíveis, uma ideia é escolher
  - sempre o vértice de **maior peso** para fazer parte da solução.



Alg. Guloso  $5 + 1 = 6$

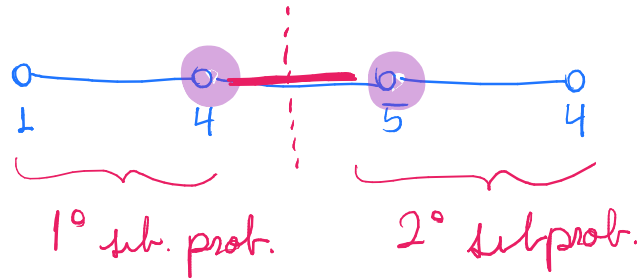
- Contra-exemplo: escolhendo o vértice de peso **5** do grafo anterior
  - ficamos impossibilitados de escolher seus vizinhos.
    - Com isso, nossa solução terá peso
- Por outro lado o **ótimo** teria peso



Ótimo  $4 + 4 = 8$

Abordagem por divisão e conquista:

- Dividir o caminho ao meio parece uma boa ideia
  - (semelhante a dividir um vetor ao meio).
- Então podemos resolver recursivamente cada subproblema.



- Contra-exemplo: no grafo anterior a solução ótima
  - do primeiro subcaminho  $(1,4)$  é 4
  - e a do segundo subcaminho  $(5,4)$  é 5
- Observe que não parece fácil combinar essas soluções,
  - já que elas têm vértices adjacentes.

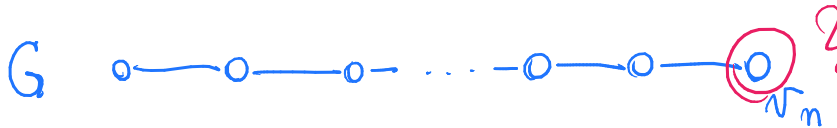
## Subestrutura ótima:

A parte central do projeto de algoritmos utilizando programação dinâmica

- é encontrar a subestrutura ótima das soluções do problema.
- Isso significa mostrar como uma solução ótima
  - é composta de soluções ótimas para subproblemas menores.
- A princípio isso serve para entender melhor o problema
  - e reduzir o espaço soluções de uma busca exaustiva,
    - mas o impacto no tempo de execução pode ser muito maior.

No problema do Conjunto Independente de Peso Máximo em Grafos Caminhos,

- imaginamos uma solução ótima  $S$ 
  - e consideramos o último vértice  $v_n$  do grafo caminho  $G$



Então analisamos por casos.

Caso 1)  $v_n \notin S$ . Sendo  $G' = G \setminus \{v_n\}$  temos que  $S$  é uma solução ótima em  $G'$



Provando por contradição: suponha que  $S^*$  é uma sol. ótima p/  $G'$  e/ custo maior que  $S$ . Como  $S^*$  é um subconj. dos vértices de  $G'$ , também é subconjunto de  $G$ . Portanto,  $S^*$  é uma solução de  $G$  e/ custo maior que  $S$  (contradição)

Caso 2)  $v_n \in S$ . Neste caso  $v_{n-1}$  não pode estar em  $S$



- Seja  $G'' = G \setminus \{v_n, v_{n-1}\}$  e  $S'' = S \setminus \{v_n\}$ 
  - Temos que  $S''$  é uma solução ótima em  $G''$

Provando por contradição: suponha que  $S^*$  é solução de  $G''$  e/ custo maior que  $S''$ . Tome  $S^* \cup \{v_n\}$  e note que este conj. é uma solução p/  $G$ , como  $w(S^*) > w(S'') = w(S) - w(v_n)$

Temos que  $w(S^* \cup \{v_n\}) = w(S^*) + w(v_n) > w(S) - w(v_n) + w(v_n) = w(S)$  (contradição)

De posse da subestrutura ótima, conseguimos descrever uma solução ótima para  $G_2$

- como a **melhor** entre uma solução ótima em  $G' = G_2 \setminus \{v_n\}$ 
  - e  $\{v_n\} \cup$  uma solução ótima em  $G'' = G_2 \setminus \{v_n, v_{n-1}\}$
- Não sabemos qual dos dois casos se aplica,
  - mas isso nos sugere o seguinte algoritmo recursivo.

Conj Ind Rec ( $G = (V, E), w$ ):

se  $|V| = 1$ : devolva  $w(v_1)$

se  $|V| = 2$ : devolva  $\max\{w(v_1), w(v_2)\}$

$S' = \text{conj Ind Rec}(G' = G \setminus \{v_n\})$

$S'' = \text{conj Ind Rec}(G'' = G \setminus \{v_n, v_{n-1}\})$

devolva  $\max\{S', S'' + w(v_n)\}$

Embora correto, esse algoritmo **leva tempo exponencial** no tamanho da entrada.

- **Quiz**: Tentem escrever a recorrência de tempo do mesmo
  - e resolvê-la usando o método da substituição.
- Dica: podem simplificar a recorrência, já que basta um limitante inferior
  - para mostrar que o algoritmo leva tempo exponencial.
- O tempo deste algoritmo é "parecido" com os números de Fibonacci.

## Algoritmo de programação dinâmica:

Da subestrutura ótima temos que uma solução ótima para  $G$  é a melhor entre:

- o uma solução ótima em  $G' = G \setminus \{v_n\}$
- o  $\{v_n\} \cup$  uma solução ótima em  $G'' = G \setminus \{v_n, v_{n-1}\}$

Dessa relação segue a recorrência:  $A[i] = \max \{ A[i-1], A[i-2] + w(v_i) \}$

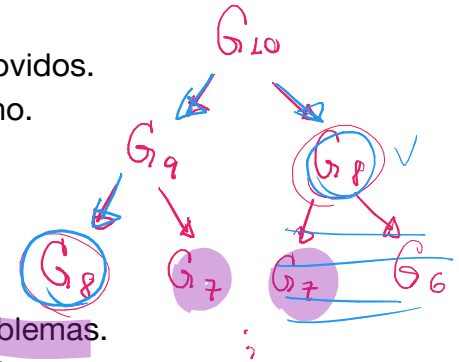
- sendo  $A[i]$  o valor do ótimo para o caminho com os  $i$  primeiros vértices.

Observem o padrão de formação dos subproblemas.

- Apenas vértices do extremo direito do caminho são removidos.
  - o Assim, cada subproblema é um prefixo do caminho.
- Como o caminho original tem  $n$  vértices,
  - o temos  $\Theta(n)$  diferentes subproblemas.

De fato, o algoritmo recursivo gasta tempo exponencial

- o apenas por ficar recalculando os mesmos subproblemas.
- Podemos torná-lo polinomial, se guardarmos numa tabela
  - o o valor de um subproblema na primeira vez que o calcularmos
  - o e verificarmos essa tabela antes de recalcularmos um subproblema.
- Essa técnica é conhecida como memorização (memoization)
  - e tem uma relação próxima com programação dinâmica.





Vamos finalmente construir nosso algoritmo iterativo de programação dinâmica.

- Para tanto vamos preencher o vetor  $A[i]$  de baixo para cima,
  - seguindo a regra da recorrência

$$A[i] = \max\{A[i-1], A[i-2] + w(v_i)\}$$

⇒ No pseudocódigo supomos que o vetor  $A[0]$  vai de 0 até  $n$ , e os vértices de 1 até  $n$

conj Ind  $(G = (V, E), w)$ :

$$A[0] = 0$$

$$A[1] = w(v_1)$$

⇒ para  $i = 2$  até  $n$ :

$$A[i] = \max\{A[i-1], A[i-2] + w(v_i)\}$$

devolva  $A[n]$

$O(n)$

Corretude: Segue da subestrutura ótima e pode ser provada por indução.

Eficiência:  $O(n)$ , pois o algoritmo só tem um laço principal

- e realiza um número de operações constante dentro deste.

## Reconstruindo a solução:

Embora nosso algoritmo encontre o valor da solução ótima,

- ele não obtém a solução em si.
- Uma opção é **modificar** o algoritmo para que
  - ele armazene a solução dos subproblemas,
  - ao menos daqueles que ainda podem fazer parte da solução ótima.
- Mas isso não costuma ser eficiente em questão de memória,
  - especialmente em algoritmos de maior dimensão
    - e com recorrências complexas.

Por isso, em geral, o mais eficiente é reconstruir a solução

- fazendo **engenharia reversa** no vetor de soluções.
- Para tanto vamos olhar novamente para nossa recorrência:
  - $A[i] = \max \{ A[i-1], A[i-2] + w(v_i) \}$
- e observar que um vértice  $v_i$  está na solução para  $G_i$  se, e somente se,
  - $w(v_i) + \text{custo da solução para } G_{i-2} \geq \text{custo da solução para } G_{i-1}$
- sendo  $G_i$  o prefixo de  $G$  com os primeiros  $i$  vértices.

Assim, vamos percorrer o vetor  $A[i]$  do fim para o início e, em cada posição,

- verificamos qual foi a escolha que o algoritmo fez (usando a recorrência).
  - Se for o caso, adicionamos o vértice corrente à solução
  - e avançamos no vetor para a posição adequada.
- Para ficar mais claro, considere os seguintes exemplos e pseudocódigo
  - supondo que o vetor  $A[i]$  foi preenchido pelo algoritmo *coj.ind.*

rec Sol ( $G, w, A$ ):

$S = \emptyset$

$i = n$

enquanto  $i \geq 2$ :

se  $A[i-1] \geq A[i-2] + w(v_i)$ :

$i = i - 1$

senão:

$S = S \cup \{v_i\}$

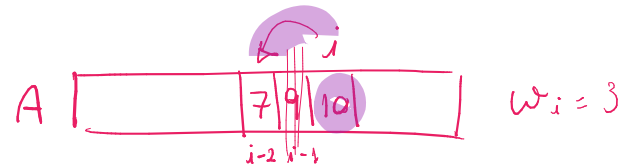
$i = i - 2$

se  $i = 1$ :  $S = S \cup \{v_i\}$

devolva  $S$

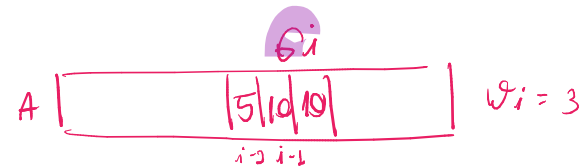
Eficiência:  $O(n)$

$O(n)$



$$7 + 3 = 10 > 9 \Rightarrow v_i \in S$$

$$i = i - 2$$



$$5 + 3 = 8 < 10 \Rightarrow v_i \notin S$$

$$i = i - 1$$