Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Conjunto Independente de Peso Máximo em Grafo Caminho

- Dado um grafo, um conjunto independente é um subconjunto S dos vértices tal que dois vértices adjacentes não podem pertencer a S.

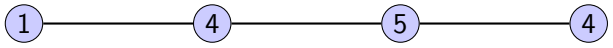
Conjunto Independente de Peso Máximo em Grafo Caminho

Dado um grafo caminho $G = (V, E)$, em que cada vértice $v \in V$ tem um peso não negativo w_v. Desejamos encontrar um subconjunto S de vértices não adjacentes (um conjunto independente - CI) de peso máximo.

Abordagens

- Uma tentativa força bruta: podemos testar todos os conjuntos independentes, lembrando daquele que tem o peso máximo.
- Porém existe $O(2^n)$ conjuntos independentes e portanto seria inviável para qualquer instância de tamanha razoável.
- Uma tentativa gulosa: Escolher o vértice de maior peso, que não seja adjacente a nenhum vértice já escolhido.
- Qual é a solução ótima e qual o resultado desse algoritmo nesse grafo?

a) 14 e 10
b) 8 e 6
c) 8 e 8
d) 9 e 8



Ideias - Subestrutura Ótima

- Vamos pensar sobre a estrutura de uma solução ótima.
- Em particular vamos tentar enxergar na solução ótima, soluções ótimas de subproblemas menores
- A ideia é que talvez a solução ótima possa ser obtida analisando um conjunto pequeno de subproblemas, e dessa forma uma busca direta nessas soluções é suficiente para encontrar a solução ótima.

- **Notação:** Seja $S \subseteq V$ ser um conjunto independente de peso máximo. Seja v_n o último vértice do caminho.
- Note que temos 2 opções aqui. Ou v_n está em S ou não está em S.
  - Caso 1: Suponha que $v_n \notin S$. Seja $G' = G$ com v_n removido.
  - Note que, S também é um conjunto independente em G' e também é o de peso máximo (suponha por contradição que não seja e você terá um CI mais pesado também para G, o que seria um absurdo)
  - Caso 2: Suponha que $v_n \in S$. Então v_{n-1} não pode mais estar na solução ótima. Seja $G'' = G$ com v_n e v_{n-1} removidos.
  - Note então que, $S - \{v_n\}$ é um conjunto independente em G'' e também é o de peso máximo (suponha por contradição que não seja e você terá um CI mais pesado também para G, o que seria um absurdo)

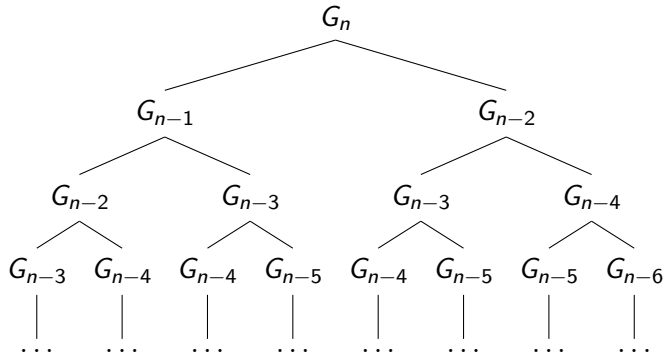
- Resumindo: Um CI de peso máximo em G deve ser
  1. um CI de peso máximo em G' ou
  2. v_n com um CI de peso máximo em G''
- Vamos tentar ambas em um algoritmo recursivo

Algoritmo 1: CIPM(G)

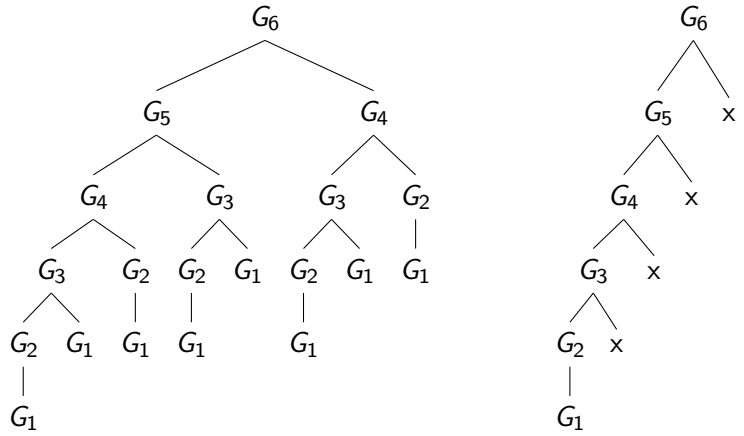
Entrada: Um grafo caminho G
Saída: Um conjunto independente de peso máximo de G

```
1 G' = G removendo o último vértice;
2 G'' = G removendo os dois últimos vértices;
3 X = CIPM(G');
4 Y = CIPM(G'') + ultimo vértice;
5 devolva o melhor entre X e Y;
```


- E qual o problema disso? Seja G_i o grafo só com os i primeiros vértices. Vejamos a árvore de recursão:



- Pode-se notar um grande número de repetições. Certo?
- E se memorizarmos numa tabela a primeira vez que encontrássemos um problema, e na próxima vez apenas consultamos ela? "Memoização"(Memoization)



Só temos n subproblemas diferentes!

- A memoização é conhecido como um método top-down, começa revolvendo do problema original e vai diminuindo o problema.
- Uma abordagem ainda mais interessante é a bottom-up, começando pelos problemas menores e usando a solução deles para construir os próximos, até chegar no problema original.
- Iremos manter um vetor $A[0 \ldots n]$ e preenche-lo da esquerda para a direita, sendo $A[i]$ o valor do CI de peso máximo de G_i.

Algoritmo 2: CIPM(G)

Entrada: Um grafo caminho G

Saída: O peso do conjunto independente de peso máximo de G

1 $A[0] = 0$; $A[1] = w_1$;
2 **para** $i = 2, 3, \ldots n$ **faça**
3 $\quad A[i] = \max\{ \quad A[i-1] \quad ; \quad A[i-2] + w_i \quad \}$;
4 devolva $A[n]$;

- Ainda precisamos reconstruir a solução (se precisarmos)
- Podemos fazer isso armazenando um vetor extra que vai contar qual decisão foi tomada em cada passo do algoritmo.
- Mas usualmente (e para economizar memória) o que fazemos é reconstruir a solução a partir do vetor que foi preenchido.
- O tempo de execução é $O(n)$ (também para reconstruir)
- A prova de corretude pode ser feita por indução.

Programação dinâmica

Princípios

Ingredientes para a Programação Dinâmica

- Identificar um conjunto pequeno de subproblemas
- Poder resolver subproblemas maiores usando a soluções dos subproblemas menores (já calculados). Usualmente escrevemos uma recorrência para demonstrar esse fato.
- Depois de resolver todos os subproblemas, poder computar (rapidamente) a solução final.

Problema da Mochila

Dado uma coleção I de n itens. Cada item $i \in I$ tem:

- Um valor v_i (não negativo)
- Um peso w_i (não negativo e inteiro)

Além disso também é dado uma capacidade W não negativa e inteira. Queremos encontrar um subconjunto $S \subseteq I$ cujo peso não ultrapasse W, ou seja,

$$\sum_{i \in S} w_i \leq W$$

e que maximiza

$$\sum_{i \in S} v_i$$

Esse problema aparece em vários contextos, basicamente qualquer problema em que temos uma quantidade limitada de recursos e queremos maximizar a eficiência.

Desenvolvendo um Algoritmo de Programação Dinâmica

- Vamos tentar pensar em como uma solução ótima pode ser formada usando a solução de subproblemas menores.
- O objetivo é encontrar recorrência que descreva esse fato.
- A ideia é partir de uma solução ótima e tentar disseca-la.
- Seja S uma solução para o problema da mochila com itens I e capacidade W.
- Assuma alguma ordenação dos itens
- Caso 1: o item n (último) não está em S. Logo S também é uma solução para o problema que considera só os $n-1$ primeiros itens.
- Caso 2: Suponha que $n \in S$. Então o que eu posso dizer sobre $S - \{n\}$?
  1. É solução para os $n-1$ primeiros itens e mochila com capacidade W
  2. É solução para os $n-1$ primeiros itens e mochila com capacidade $W - v_n$
  3. É solução para os $n-1$ primeiros itens e mochila com capacidade $W - w_n$
  4. Pode não ser viável

- **Notação:** Seja $V_{i,x}$ o valor da melhor solução que:
  1. só usa os i primeiros itens
  2. tem tamanho total $\leq x$
  3. Passo 1: encontra uma recorrência

Para $i \in I$ e para qualquer x,

$$V_{i,x} = \max \begin{cases} V_{(i-1),x} \\ v_i + V_{(i-1),(x-w_i)} \end{cases} \qquad \text{(somente se } w_i \leq x\text{)}$$

- Passo 2: Agora precisamos identificar os subproblemas.
- Variamos por todos os prefixos de itens $\{1, 2, \ldots, i\}$
- Variamos todas as capacidades residuais possíveis $x \in \{0, 1, 2, \ldots, W\}$
- Passo 3: Usar a recorrência para revolver todos os subproblemas (preencher tabela)
- Seja A um vetor bidimensional tal que $A[i, x]$ vai guardar o valor de $V_{i,x}$

Algoritmo 3: Knap(I, W)

Entrada: Um conjunto de itens I e uma capacidade W
Saída: O valor da solução ótima

```
1 A[0, x] = 0 para todo x;
2 para i = 1, 2, ..., n faça
3     para x = 0, 1, 2, ..., W faça
4         A[i, x] = max{A[i − 1, x]; A[i − 1, x − w_i] + v_i};
5 devolva A[n, W];
```


- O tempo de execução é $O(nW)$

Exemplo

- Um exemplo com 4 itens e Capacidade 6
- $v_1 = 3,\ w_1 = 4$
- $v_2 = 2,\ w_2 = 3$
- $v_3 = 4,\ w_3 = 2$
- $v_4 = 4,\ w_4 = 3$

Algoritmo 4: Knap(I, W)

```
1 A[0, x] = 0 para todo x;
2 para i = 1, 2, ..., n faça
3     para x = 0, 1, 2, ..., W faça
4         A[i, x] =
5             max { A[i − 1, x],
                  { A[i − 1, x − w_i] + v_i
6 devolva A[n, W];
```

6					
5					
4					
3					
2					
1					
x = 0					
i =	0	1	2	3	4

6	0	3	3	7	8
5	0	3	3	6	8
4	0	3	3	4	4
3	0	0	2	4	4
2	0	0	0	4	4
1	0	0	0	0	0
x = 0	0	0	0	0	0
i =	0	1	2	3	4