Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009
2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff) `http://www.ime.usp.br/~pf/analise_de_algoritmos`

Algoritmos gulosos

Algoritmos gulosos

Normalmente aplicado a problemas de otimização, em que queremos computar a melhor solução.

Em cada passo, o algoritmo sempre escolhe a melhor opção local viável, sem se preocupar com as conseqüências futuras (dizemos que ele é "míope")

Algoritmos gulosos

▷ Nem sempre produz a solução ótima

▷ Não existe backtracking envolvido

▷ Na maioria das vezes, projetar ou descrever um algoritmo guloso é "fácil", mas provar sua corretude é difícil.

Estratégia gulosa vs Programação dinâmica

Algoritmo guloso ("ganancioso"):

▷ "abocanha"a alternativa mais promissora, sem explorar outras
▷ a execução costuma a ser muito rápida
▷ nunca se arrepende da decisão tomada
▷ prova de corretude difícil

Estratégia gulosa vs Programação dinâmica

Algoritmo de programação dinâmica:

▷ explora todas as alternativas, e faz isso de maneira eficiente
▷ a execução é um tanto "lenta"
▷ a cada iteração pode se arrepender de decisões tomadas anteriormente (pode rever o "ótimo corrente")
▷ prova de corretude fácil (explora todas as possibilidades)

Exemplo: Máximo e Maximal

$\triangleright$ S é uma coleção de subconjuntos de $\{1 \cdots n\}$

$\triangleright$ $X \subset S$ é **máximo** se não existe $Y \subset S$ tal que $|Y| > |X|$

$\triangleright$ $X \subset S$ é **maximal** se não existe $Y \subset S$ tal que $X \subset Y$, ou seja, se nenhum elemento de S é superconjunto próprio de X.

Exemplo: Máximo e Maximal

$S = \{\{1,2\},\{2,3\},\{4,5\},\{1,2,3\},\{1,2,4\},\{2,3,4,5\},\{1,3,4,5\}\}$

$\triangleright$ Elementos máximos: $\{2,3,4,5\}$ e $\{1,3,4,5\}$
$\triangleright$ Elementos maximais: $\{1,2,3\}$, $\{1,2,4\}$, $\{2,3,4,5\}$ e $\{1,3,4,5\}$

Note que todo máximo é maximal, mas a recíproca não é verdade.

Máximo e Maximal

▷ Procurar um elemento máximo em S é computacionalmente pesado (examinar todos os elementos)

▷ Mas encontrar um elemento maximal de S é muito fácil, aplicando a estratégia gulosa.

Máximo e Maximal

Algoritmo guloso para o maximal:

escolha algum X em S

enquanto $X \subset Y$ para algum Y em S **faça**

$X \leftarrow Y$

devolva X

Verifique que o algoritmo funciona para o exemplo descrito anteriormente

Máximo e Maximal

É ainda mais fácil encontrar um elemento maximal se a coleção S tiver caráter hereditário, ou seja, se tiver a seguinte propriedade: para cada X em S, todos os subconjuntos de X também estão em S. Nesse caso, basta executar o algoritmo:

$X \leftarrow \{\}$
para cada k em $\{1 \cdots n\}$ faça
    $Y \leftarrow X \cup \{k\}$
    se Y está em S
        então $X \leftarrow Y$
devolva X

Problema da seleção de atividades

Problema da seleção de atividades

Considere um conjunto $\{1, 2, \cdots, n\}$ de n atividades que competem por um recurso, por exemplo uma sala de aula.

Cada atividade tem um início s_i e um término t_i, com $s_i \leq t_i$.

O intervalo requerido pela atividade é $[s_i, t_i)$.

Problema da seleção de atividades

Duas atividades i e j são compatíveis se os intervalos $[s_i, t_i)$ $[s_j, t_j)$ não se interceptam ($s_i \geq t_j$ ou $s_j \geq t_i$).

Problema: encontrar o conjunto de atividades mutuamente compatíveis de tamanho máximo.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

Podemos verificar três estratégias:

$\triangleright$ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.

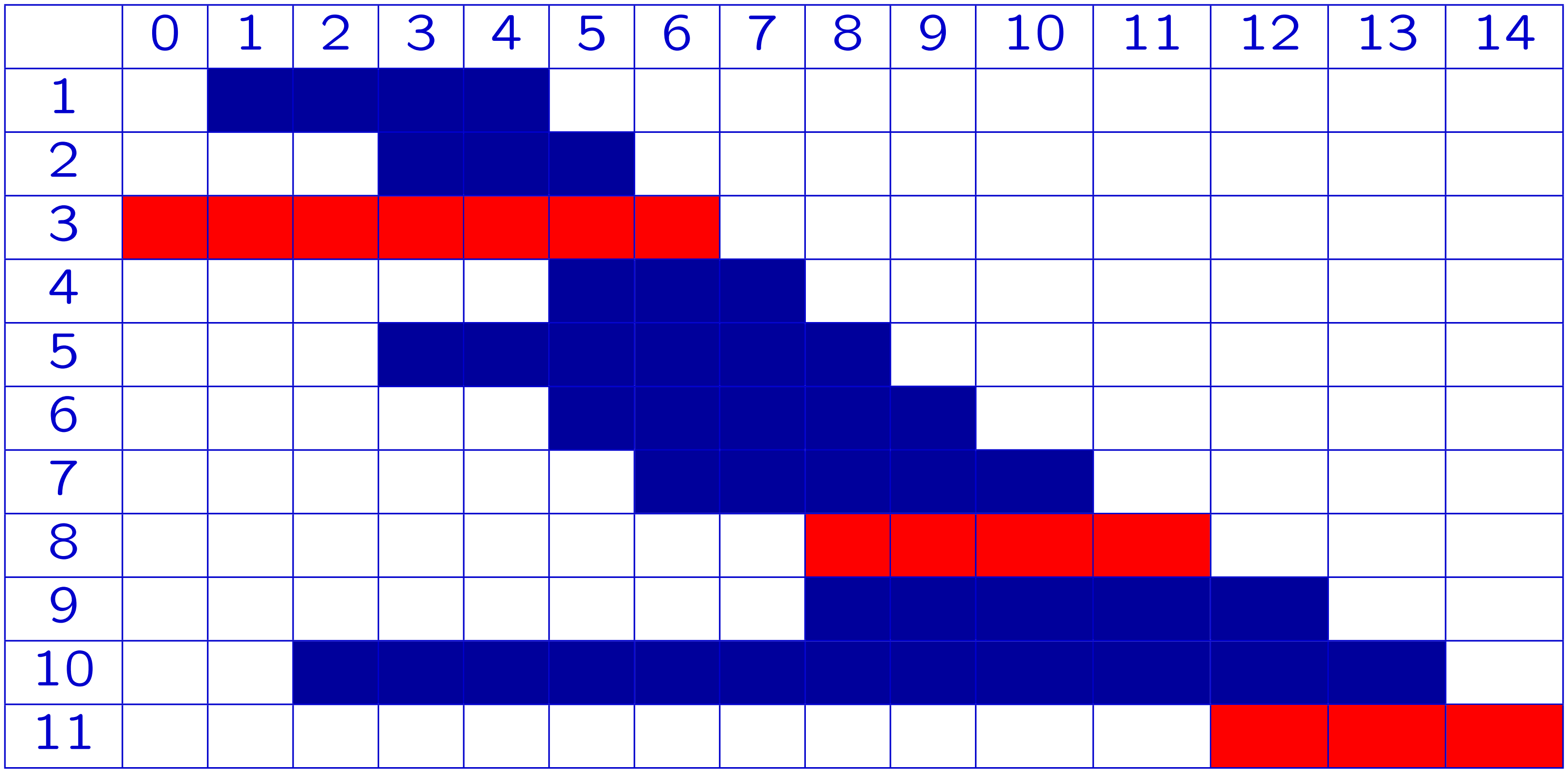
$\triangleright$ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.

$\triangleright$ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Problema da seleção de atividades

1^a tentativa: Escolher primeiro as atividades que começam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

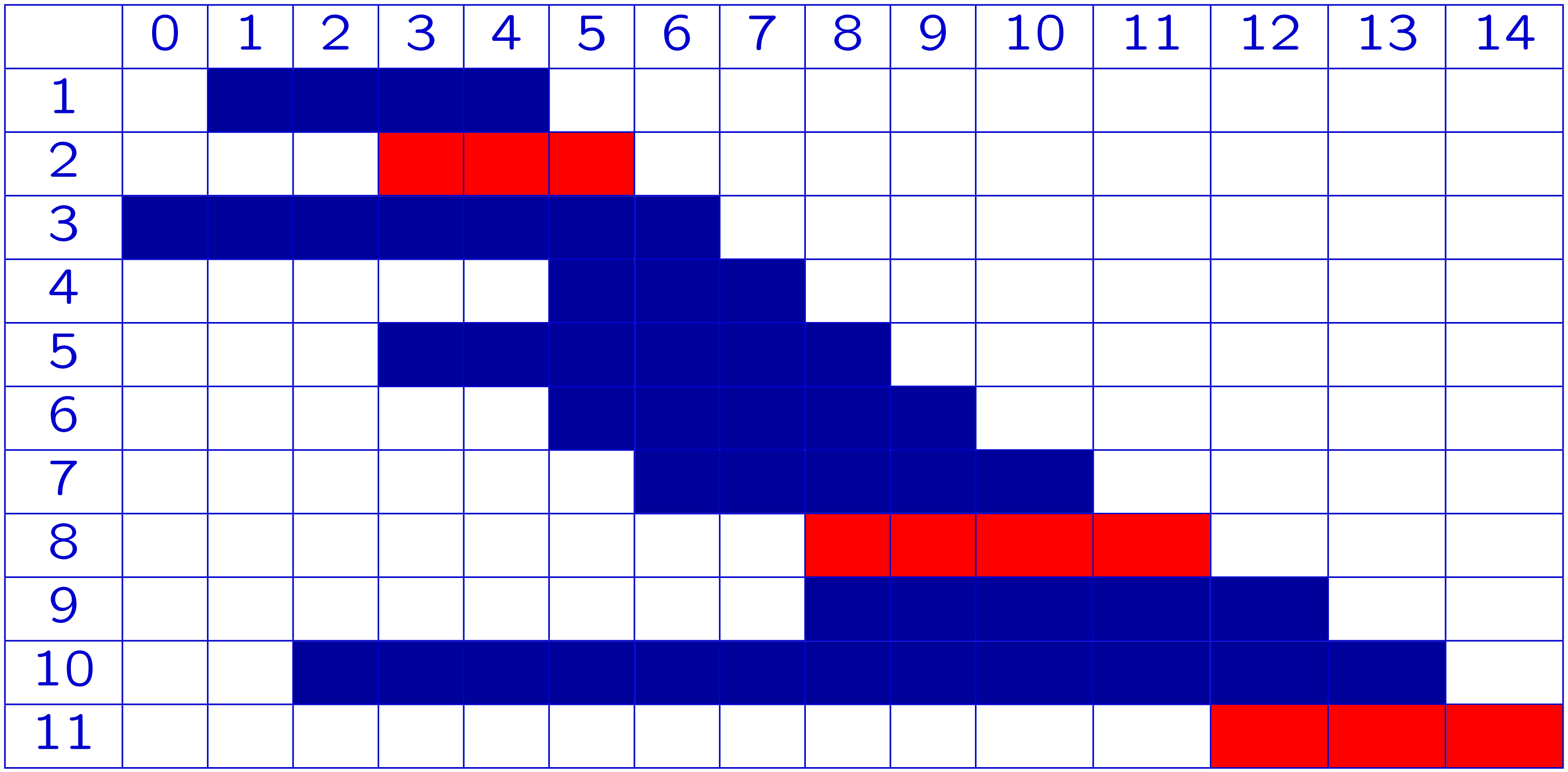


1^a tentativa: Escolher primeiro as atividades que começam primeiro. Escolhemos as atividades 3, 8 e 11.

Problema da seleção de atividades

2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

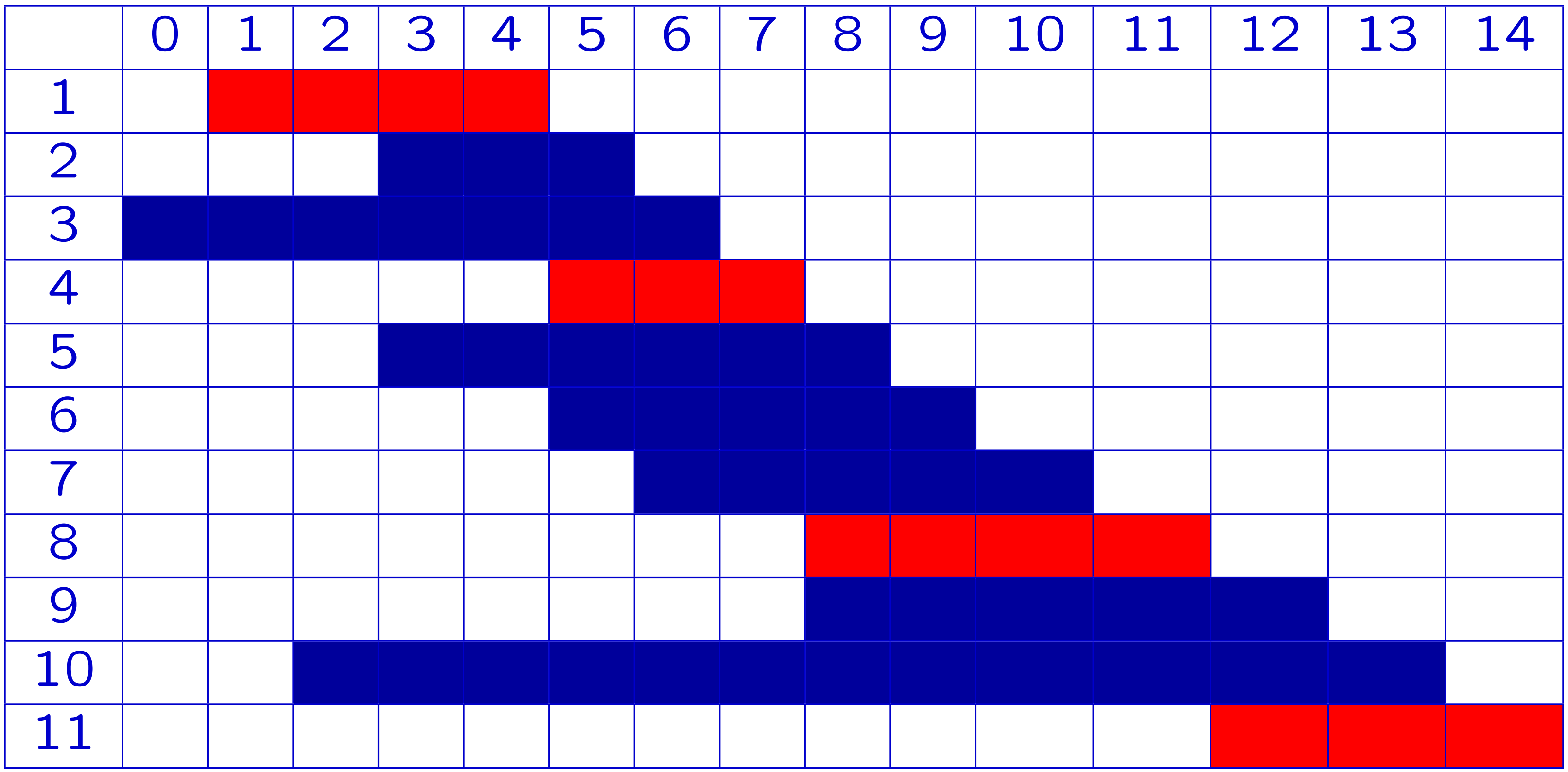


2^a tentativa: Escolher primeiro as atividades que demoram menos tempo. Escolhemos as atividades 2, 8 e 11.

Problema da seleção de atividades

3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo



3^a tentativa: Escolher primeiro as atividades que terminam primeiro. Escolhemos as atividades 1, 4, 8 e 11.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

▷ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.
▷ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.
▷ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

A 3^a tentativa apresentou a melhor estratégia de seleção de atividade. Dizemos que a solução é **ótima** para esta instância!

Problema da seleção de atividades

seleciona_atividades_guloso(s, t, n)

ordene s e t de tal forma que
$t[1] \leq t[2] \leq \cdots \leq t[n]$
$A \leftarrow \{1\}$ $\quad \triangleright\ t_1 \leq t_2 \leq \cdots \leq t_n$
$j \leftarrow 1$
para $i = 2$ **até** n
  faça se $s_i \geq t_j$ $\quad \triangleright\ i$ e j são compatíveis
    então $A \leftarrow A \cup \{i\}$
      $j \leftarrow i$
retorne A

Complexidade do algoritmo: $O(n \lg n)$

Estratégia gulosa

Ingredientes chaves de um algoritmo guloso:

- ▷ subestrutura ótima
- ▷ característica gulosa: Solução ótima global pode ser produzida a partir de uma escolha ótima local.

O problema do troco

O problema do troco

Dados os valores de moedas (cédulas e moedas) de um país, determinar o mínimo de moedas para dar um valor de troco.

O problema do troco

Escolha gananciosa: devolver o número mínimo de moedas de mais alto valor cuja soma total resulta no valor de determinado troco. O algoritmo guloso encontra a solução ótima, isto é, o troco com o menor número de moedas.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco $= 37$, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

$\triangleright$ **entrada:** conjunto $C = \{100, 50, 25, 10, 1\}$ de moedas em ordem decrescente e o valor do troco.
$\triangleright$ **saída:** conjunto S com as moedas utilizadas.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco $= 37$, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

Idéia: Em cada estágio adicionamos a moeda de maior valor possível, de forma a não ultrapassar a quantidade necessária para o valor do troco.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco = 37, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

Podemos fazer:

- $\triangleright$ $37 - 25 = 12$
- $\triangleright$ $12 - 10 = 2$
- $\triangleright$ $2 - 1 = 1$
- $\triangleright$ $1 - 1 = 0$

O total de 4 moedas: $\{25, 10, 1, 1\}$ **(solução ótima)**

O problema do troco

troco($C, valor$)

$S \leftarrow \{\}$ $soma \leftarrow 0$
enquanto ($soma < valor$) **e** (C não vazio) **faça**
  $m \leftarrow$ moeda de maior valor em C
  se $soma + m \leq valor$ **então**
    $soma \leftarrow soma + m$
    $S \leftarrow S \cup \{m\}$
  senão
    $C \leftarrow C - \{m\}$
se $soma = valor$ **então**
  retorne S
senão
  retorne "não encontrei a solução"

Problema da mochila

Problema da mochila

Dados:

$\triangleright$ Uma mochila que possui uma certa capacidade W (admite um peso).
$\triangleright$ Um conjunto de n objetos distintos, enumerados de 1 a n, cada um com um certo valor $v_1, v_2, \cdots, v_n$ e peso $w_1, w_2, \cdots, w_n$.

Objetivo:

$\triangleright$ Maximizar o valor do conjunto de objetos dentro da mochila respeitando a capacidade.

Problema da mochila

Há duas variações para o problema da mochila

- ▷ Mochila fracionária
- ▷ Mochila binária ou $0 - 1$

Problema da mochila fracionária

Os objetos podem ser particionados, ou seja, você pode colocar uma fração do objeto na mochila.

ex.: ouro em pó

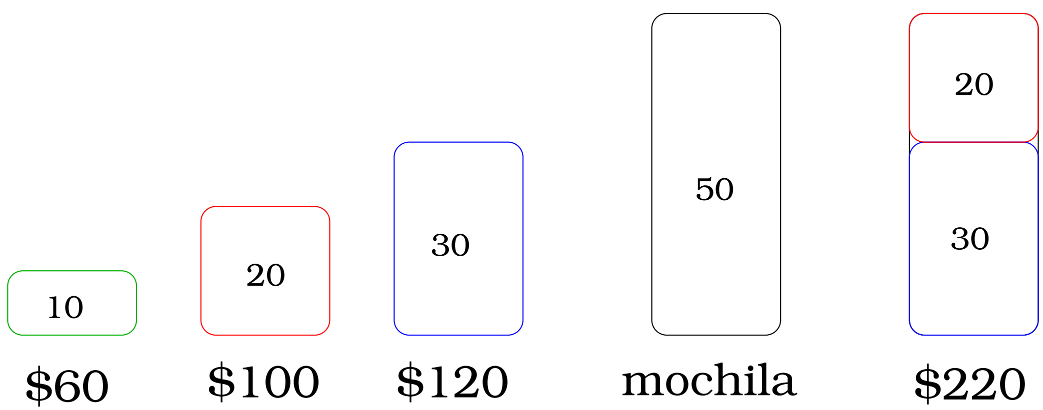
Problema da mochila binária

Os objetos não podem ser particionados, você só pode colocar itens inteiros e apenas uma vez.

ex.: ouro em barra

Problema da mochila

- Ambos tem sub-estrutura ótima.
- O problema da mochila fracionária tem solução gulosa, o problema da mochila binária não.



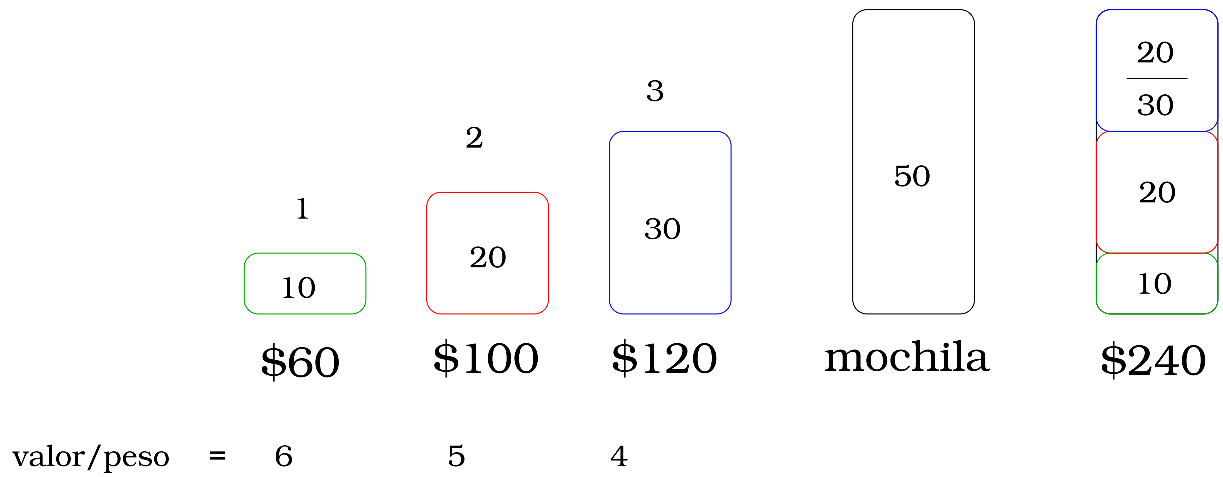
Problema da mochila fracionária

Ideia do algoritmo:

- ▷ Ordene os itens por $valor/peso$ em ordem decrescente.
- ▷ Começando em $i = 1$ coloque na mochila o máximo do item i que estiver disponível e for possível.
- ▷ Se a capacidade da mochila permitir passe para o próximo item.

Problema da mochila fracionária

Item	Valor	Peso	Valor/Peso
1	$60	10	6
2	$100	20	5
3	$120	30	4



Problema da mochila fracionária

```
mochila_fracionaria(w, v, n, W)
1   ordene w e v de tal forma que
    v[1]/w[1] ≥ v[2]/w[2] ≥ ··· v[n]/w[n]
2   para i = 1 até n faça
3       se w[i] ≤ W
4           então  x[i] ← 1
5                  W ← W − w[i]
6           senão  x[i] ← W/w[i]
7                  W ← 0
8   retorne x
```

Consumo de tempo na linha 1: $\Theta(n \lg n)$.

Consumo de tempo nas linhas 2-8: $\Theta(n)$.

Invariante

No início de cada execução da linha 2 vale que

$x' = x[1 \cdots i-1]$ é mochila ótima para $(w', v', i-1, W)$

onde

$$w' = w[1 \cdots i-1]$$
$$v' = v[1 \cdots i-1]$$

Na última iteração $i = n$ e portanto $x[1 \cdots n]$ é mochila ótima para (w, v, n, W)

Conclusão

$\boxed{\text{O consumo de tempo do algoritmo mochila\_fracionaria é } \Theta(n \lg n).}$

Salto do Sapo

Salto do Sapo

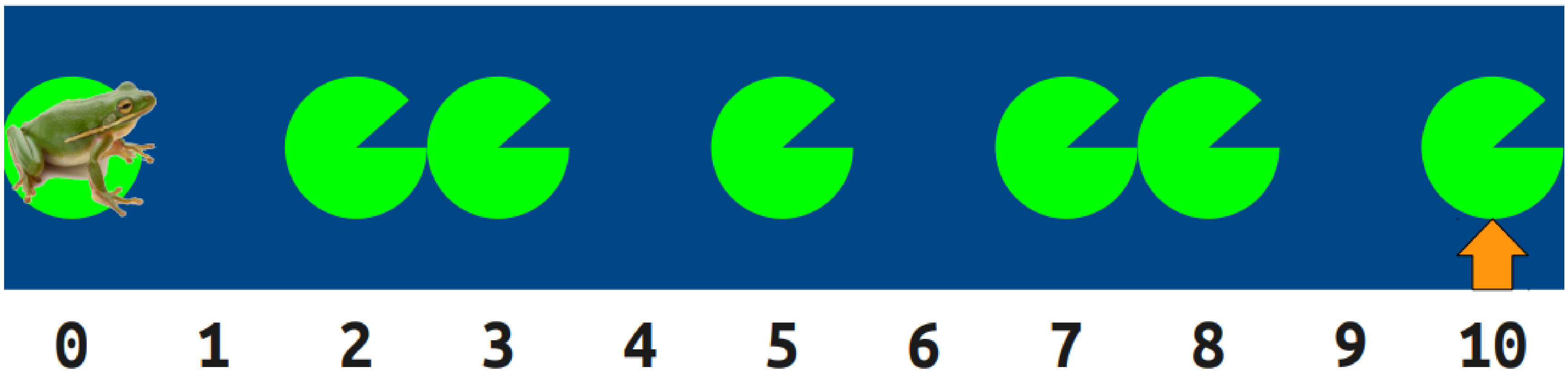
Existem n pedras numa reta numérica, em posições distintas $p_1, p_2, \cdots, p_n$. Dizemos que um sapo pode saltar de uma pedra p_i para outra pedra p_j desde que a distância entre elas seja menor ou igual a $delta$.

Salto do Sapo

Considere um sapo localizado inicialmente na pedra p_1. Qual é o menor número de saltos que ele precisa dar para chegar na pedra p_n?

Ou seja, é dado um vetor de n números distintos ordenados $p = \{p_1, p_2, \cdots, p_n\}$ e um número $delta$.

Salto do Sapo



Salto do Sapo

Uma sequência $u = u_1, u_2, \cdots, u_k$ é solução se:

- $\triangleright$ $u_1 = p_1$
- $\triangleright$ $u_k = p_n$
- $\triangleright$ $u_i = p_j$ para todo $i \in [1, k]$ e algum $j \in [1, n]$
- $\triangleright$ $|u_i - u_{i+1}| \leq delta$ para $i \in [1, k-1]$

Queremos uma sequência u que satisfaça as propriedades acima e que o tamanho k de u seja o mínimo possível. Vamos assumir que sempre existe pelo menos uma solução.

Salto do Sapo

Exemplo 1: entrada $n = 4$, $p = \{1, 2, 3, 4\}$, $delta = 1$.

$\triangleright$ existem diversas soluções possíveis entre elas
$\{1, 2, 3, 4\}$, $\{1, 2, 1, 2, 3, 4\}, \{1, 2, 1, 2, 3, 2, 3, 4\}$

A sequência de menor $k = 4$ é $u = \{1, 2, 3, 4\}$

Salto do Sapo

Exemplo 2: entrada $n = 6$, $p = \{1, 2, 3, 5, 6, 7\}$, $delta = 2$.

tem como solução ótima $u = \{1, 3, 5, 7\}$

Salto do Sapo

Exemplo 3: entrada $n = 3$, $p = \{1, 3, 4\}$, $delta = 1$.

não admite solução, já que a partir de 1 não é possível alcançar 3 ou 4.

Vamos desconsiderar este caso.

Salto do Sapo

Observe que:

▷ nunca vale "voltar", pois isso aumentaria a sequência de números

▷ O sapo deve pular o mais longe possível.

Salto do Sapo

salto_sapo($p, n, delta$)

$u \leftarrow \{p[1]\}$

$ultima\_pos \leftarrow p[1]$

para $i = 2$ **até** n **faça**

  se $p[i] - ultima\_pos > delta$ **então**

    $ultima\_pos \leftarrow p[i-1]$

    $u \leftarrow u \cup p[i-1]$

$u \leftarrow u \cup p[n]$

retorne u

complexidade: $O(n)$

Seleção de paradas

Seleção de paradas

Problema:

- ▷ Viagem da cidade A para a cidade B ao longo de uma rodovia.
- ▷ Tanque de combustível tem capacidade suficiente para cobrir n quilômetros.
- ▷ Mapa indica a localização dos postos de combustível ao longo do caminho.

Objetivo: minimizar a quantidade de paradas ao longo da viagem.

Seleção de paradas

solução gulosa: Avançar a viagem o máximo que puder antes de reabastecer

Seleção de paradas

algoritmo do caminhoneiro(p, n, C)

Ordene as paradas de modo que
$p_0 \leq p_1 \leq \cdots \leq p_n$
$S \leftarrow \{\}$ $\triangleright$ paradas selecionadas
$ultima\_parada = 0$
para $i = 0$ **até** n **faça**
    se $p_i - ultima\_parada > C$ **então**
        $ultima\_parada = p_{i-1}$
        $S \leftarrow S \cup (i-1)$
retorne S

complexidade: $O(n \lg n)$

Escalonamento (Schedule)

Escalonamento (Schedule)

Problema: Ordenar tarefas de tal forma que o tempo médio que cada tarefa fica no sistema é minimizado.

Escalonamento (Schedule)

Exemplo: Sistema é o caixa de banco, e a tarefa é o cliente na fila de atendimento.

Queremos minimizar o tempo que cada cliente espera desde quando entra no banco até o momento em que termina de ser atendido.

Escalonamento (Schedule)

Formalizando um pouco, temos n clientes, sendo que cada cliente i irá exigir um tempo de serviço t_i, $1 \leq i \leq n$, e queremos minimizar o tempo médio que cada cliente gasta no sistema.

Escalonamento (Schedule)

Como n (número de clientes) é fixo, temos que minimizar o tempo total gasto no sistema para todos os clientes, isto é:

$$T = \sum_{i=1}^{n} (\text{tempo no sistema gasto pelo cliente } i)$$

Escalonamento (Schedule)

Exemplo:

Suponha três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

Para a ordem de atendimento:

tarefa	tempo de chegada	tempo de atendimento
1	0	5
2	5	15
3	15	18

O cliente 1 foi atendido imediatamente.

Escalonamento (Schedule)

Exemplo:

Suponha três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

Para a ordem de atendimento:

tarefa	tempo de chegada	tempo de atendimento
1	0	5
2	5	15
3	15	18

O cliente 2 precisou esperar o cliente 1 ser atendido.

Escalonamento (Schedule)

Exemplo:

Suponha três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

Para a ordem de atendimento:

tarefa	tempo de chegada	tempo de atendimento
1	0	5
2	5	15
3	15	18

O cliente 3 precisou esperar os dois anteriores serem atendidos.

Escalonamento (Schedule)

Exemplo:

Suponha três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

Para a ordem de atendimento:

tarefa	tempo de chegada	tempo de atendimento
1	0	5
2	5	15
3	15	18

Tempo total de atendimento: $5 + 15 + 18 = 38$

Escalonamento (Schedule)

▷ Para a ordem de atendimento 123 o tempo total de atendimento é 38.

▷ Mas será que está é a melhor ordem de atendimento?
Vamos analisar todas as combinações.

Escalonamento (Schedule)

- Três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

As possíveis ordens de atendimento:

ordem	tempo total de atendimento
123	$5 + (5 + 10) + (5 + 10 + 3) = 38$
132	$5 + (5 + 3) + (5 + 3 + 10) = 31$
213	$10 + (10 + 5) + (10 + 5 + 3) = 43$
231	$10 + (10 + 3) + (10 + 3 + 5) = 41$
312	$3 + (3 + 5) + (3 + 5 + 10) = 29$
321	$3 + (3 + 10) + (3 + 10 + 5) = 34$

Escalonamento (Schedule)

- Três clientes, com $t1 = 5$, $t2 = 10$ e $t3 = 3$

As possíveis ordens de atendimento:

ordem	tempo total de atendimento
123	$5 + (5 + 10) + (5 + 10 + 3) = 38$
132	$5 + (5 + 3) + (5 + 3 + 10) = 31$
213	$10 + (10 + 5) + (10 + 5 + 3) = 43$
231	$10 + (10 + 3) + (10 + 3 + 5) = 41$
312	$3 + (3 + 5) + (3 + 5 + 10) = 29$ (solução ótima)
321	$3 + (3 + 10) + (3 + 10 + 5) = 34$

solução ótima: atender primeiro os clientes que gastam menos tempo.

Escalonamento (Schedule)

Idéia do algoritmo:

1. Ordenar os n clientes pelos tempos
2. Realizar o atendimento dos clientes

Complexidade: $O(n \log n)$

Código de Huffman

Código de Huffman

▷ Técnica de compressão de dados (economia de 20% a 90%, dependendo do arquivo a ser comprimido).

▷ O algoritmo guloso de Huffman usa uma **tabela de frequência de caracteres** para obter um código binário único (**código prefixo**) de cada caracter encontrado no arquivo de entrada.

Código de Huffman

Suponha um arquivo texto contendo 100000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As frequências de cada caracter no arquivo são indicadas na tabela abaixo.

	a	b	c	d	e	f
(1) Frequência (em milhares)	45	13	12	16	9	5
(2) Código de tamanho fixo	000	001	010	011	100	101
(3) Código de tamanho variável	0	101	100	111	1101	1100

Código de Huffman

Suponha um arquivo texto contendo 100000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As frequências de cada caracter no arquivo são indicadas na tabela abaixo.

	a	b	c	d	e	f
(1) Frequência (em milhares)	45	13	12	16	9	5
(2) Código de tamanho fixo	000	001	010	011	100	101
(3) Código de tamanho variável	0	101	100	111	1101	1100

Tamanho em bits do arquivo:

(1) codifica 800000 bits

(2) codifica 300000 bits

(3) codifica 224000 bits (melhor)

Código de Huffman

Problema da codificação: Dadas as frequências de ocorrências dos caracteres de um arquivo, encontrar a sequência de bits (códigos) para representá-los de modo que o arquivo comprimido tenha tamanho mínimo.

Código de Huffman

Códigos livres de prefixos: o código de um símbolo não é prefixo do código de nenhum outro símbolo.

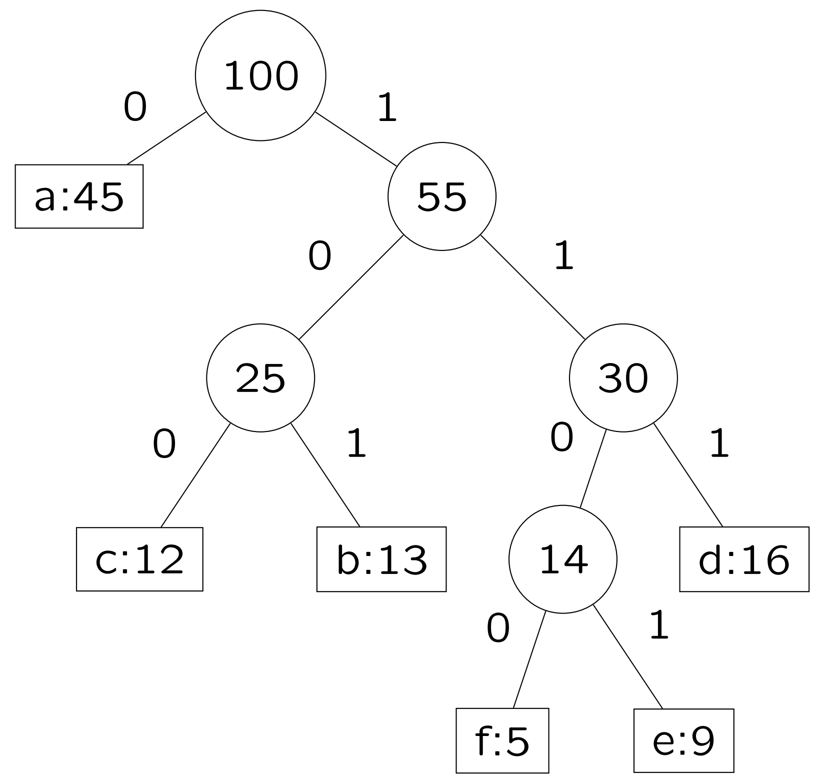
Código de Huffman

Códigos livres de prefixos: o código de um símbolo não é prefixo do código de nenhum outro símbolo.

Códigos livres de prefixos são fáceis de decodificar.

Código de Huffman

letra	freq	código
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100



Código de Huffman

Uma codificação ótima sempre pode ser representada por uma árvore binária **cheia**, na qual cada vértice interno tem exatamente dois filhos.

Assim, podemos dizer que, seja C o alfabeto em questão, então a árvore para um código livre de prefixos ótimo terá $|C|$ folhas e $|C| - 1$ nós internos.

Código de Huffman

O número de bits requeridos para codificar um arquivo é

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c) \Rightarrow \text{ custo da árvore } T$$

onde T é uma árvore binária, $f(c)$ é frequência do caracter c no arquivo e $d_T(c)$ é a profundidade da folha na árvore T.

Código de Huffman

Observando a árvore anterior, vamos calcular o número de bits requeridos para codificar em arquivo contituído pelos caracteres da árvore.

$$\begin{aligned}
a &= 45 \cdot 1 && = 45\\
b &= 13 \cdot 3 && = 39\\
c &= 12 \cdot 3 && = 36\\
d &= 16 \cdot 3 && = 48\\
e &= 9 \cdot 4 && = 36\\
f &= 5 \cdot 4 && = 20
\end{aligned}$$

Custo total: $B(T) = 254$ bits

Construindo o código de Huffman

algoritmo de Huffman

▷ **Entrada:** Conjunto de caracteres C com as frequências f dos caracteres em C.

▷ **Saída:** raíz da árvore binária representando uma codificação ótima livre de prefixos.

Construindo o código de Huffman

Huffman(C)

$Q \leftarrow C$
para $i = 1$ **até** $|C| - 1$ **faça**
    $x \leftarrow$ Extrai-Min(Q)
    $y \leftarrow$ Extrai-Min(Q)
    $z \leftarrow$ Alocar-No()
    $esq[z] \leftarrow x$
    $dir[z] \leftarrow y$
    $f[z] \leftarrow f[x] + f[y]$
    Insere(Q, z)
retorne Extrai-min(Q)

Desempenho de Huffman

- $\triangleright$ Q é fila de prioridades
- $\triangleright$ Extrai-min retira de Q um nó q com $f[q]$ mínima.
- $\triangleright$ Tamanho da instância: $n = |C|$
- $\triangleright$ $n - 1$ vezes: Extrai-Min, Extrai-Min, Insere
- $\triangleright$ Cada Extrai-Min e cada Insere consome $O(\lg n)$

total: $O(n \lg n)$

Construindo o código de Huffman

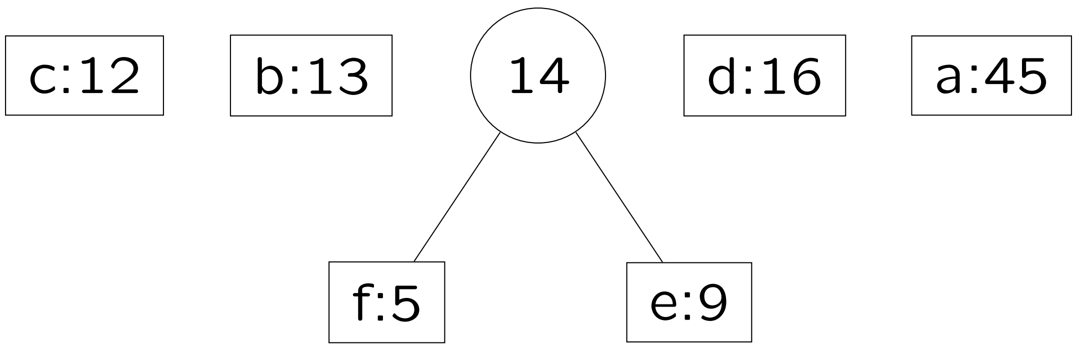
Podemos ilustrar a execução deste algoritmo com o seguinte conjunto de caracteres e suas frequências

letra	freq
a	45
b	13
c	12
d	16
e	9
f	5

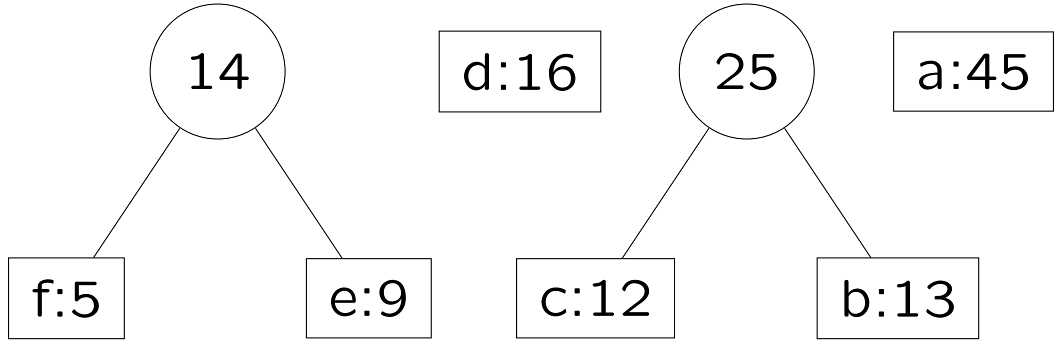
Construindo o código de Huffman (1)

f:5	e:9	c:12	b:13	d:16	a:45

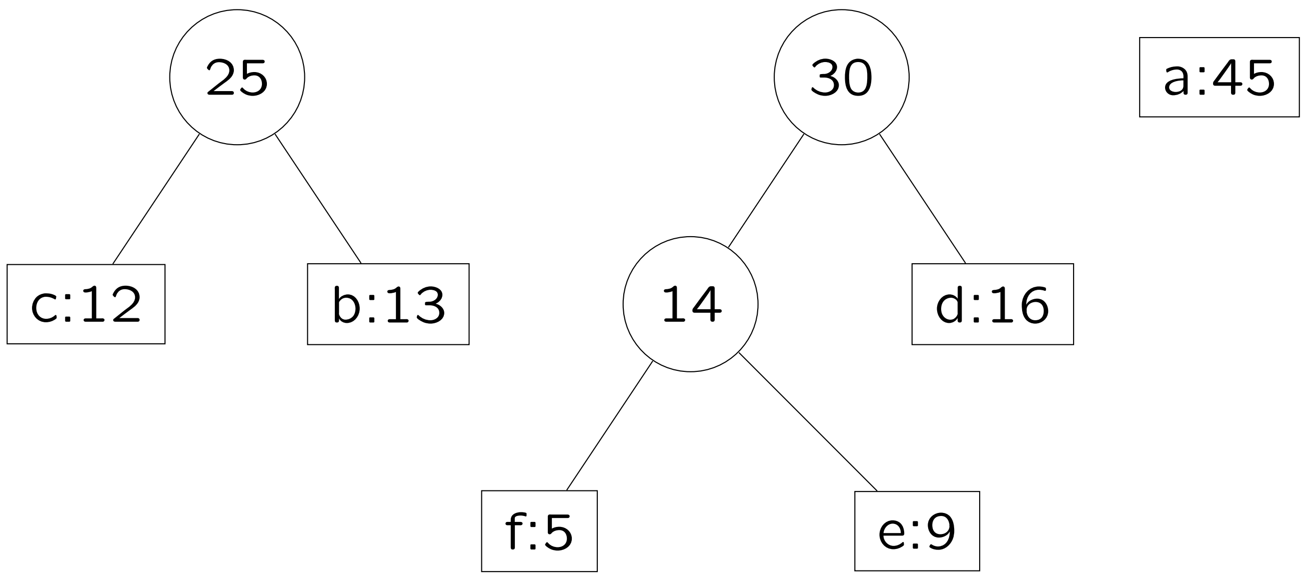
Construindo o código de Huffman (2)



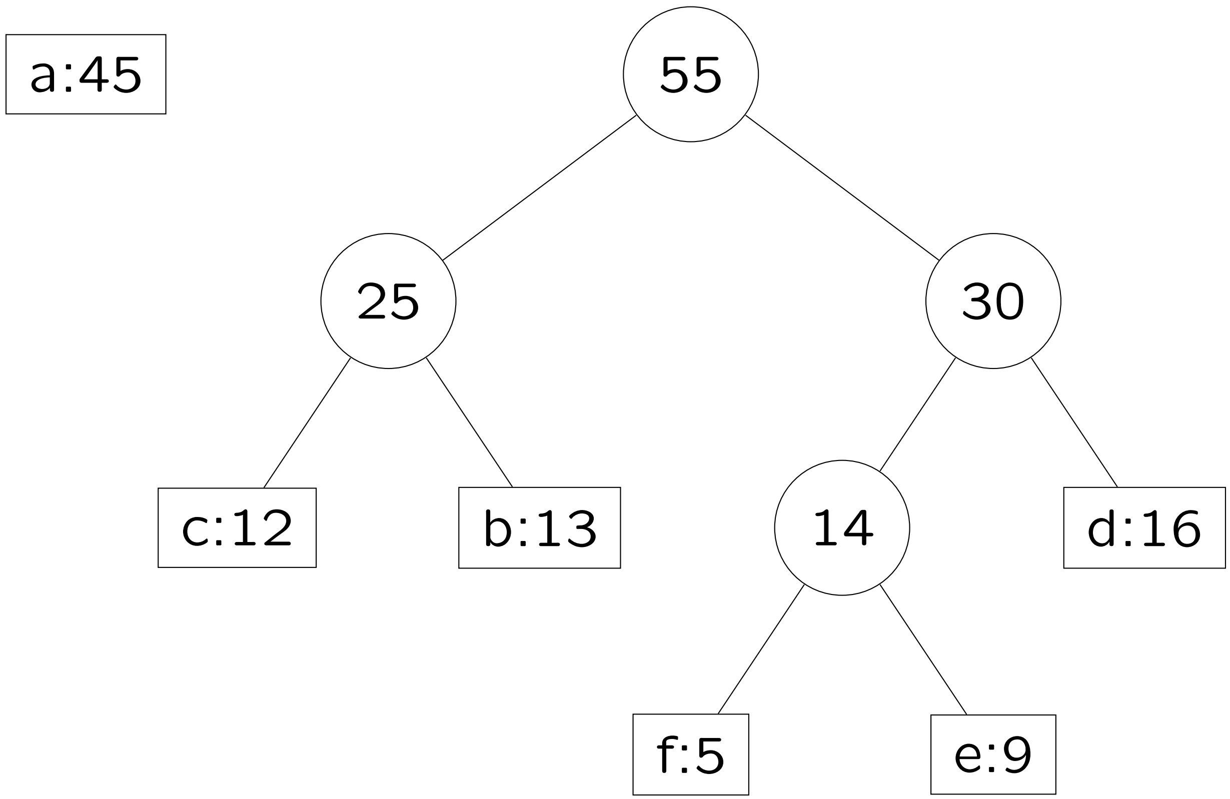
Construindo o código de Huffman (3)



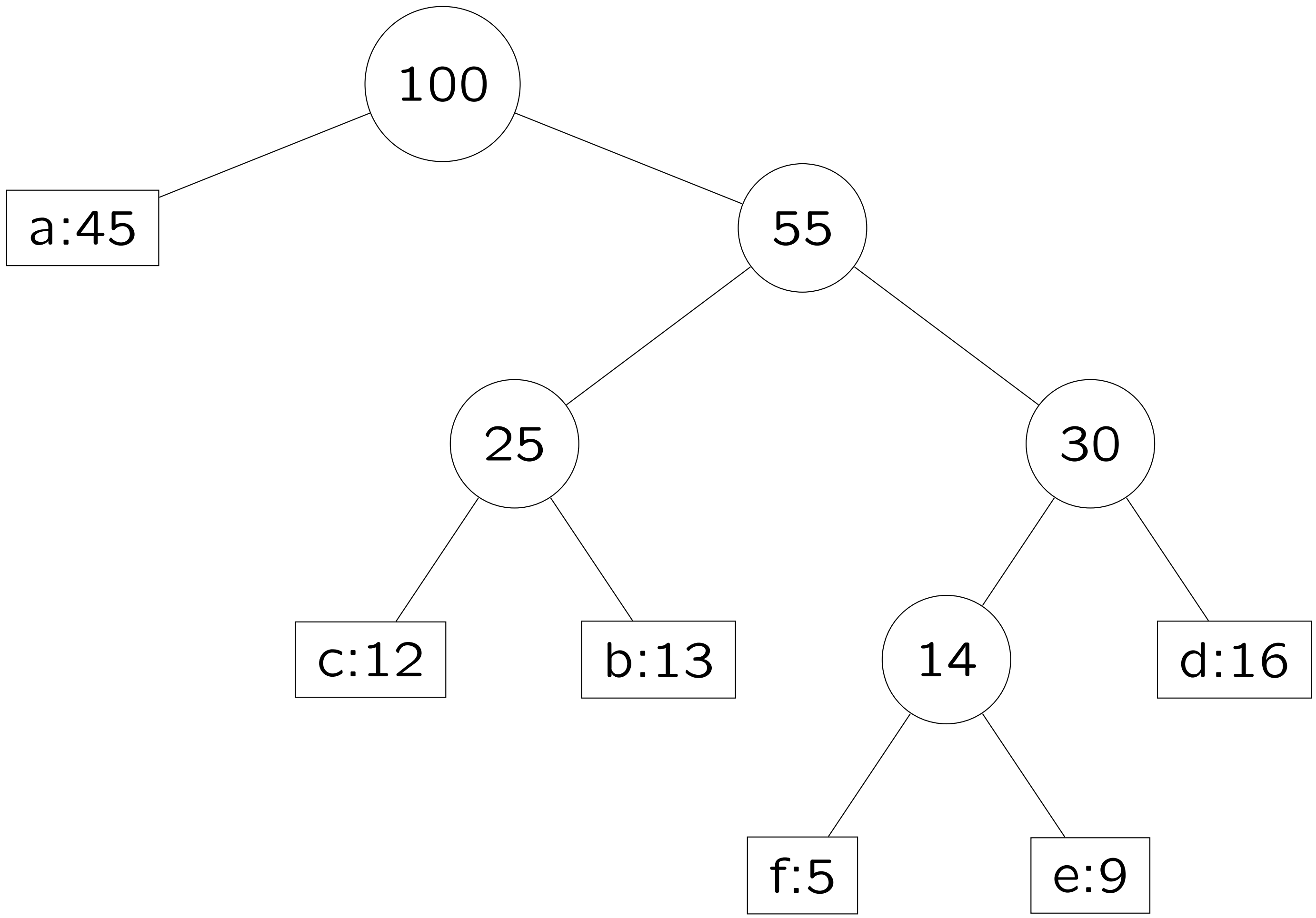
Construindo o código de Huffman (4)



Construindo o código de Huffman (5)



Construindo o código de Huffman (6)



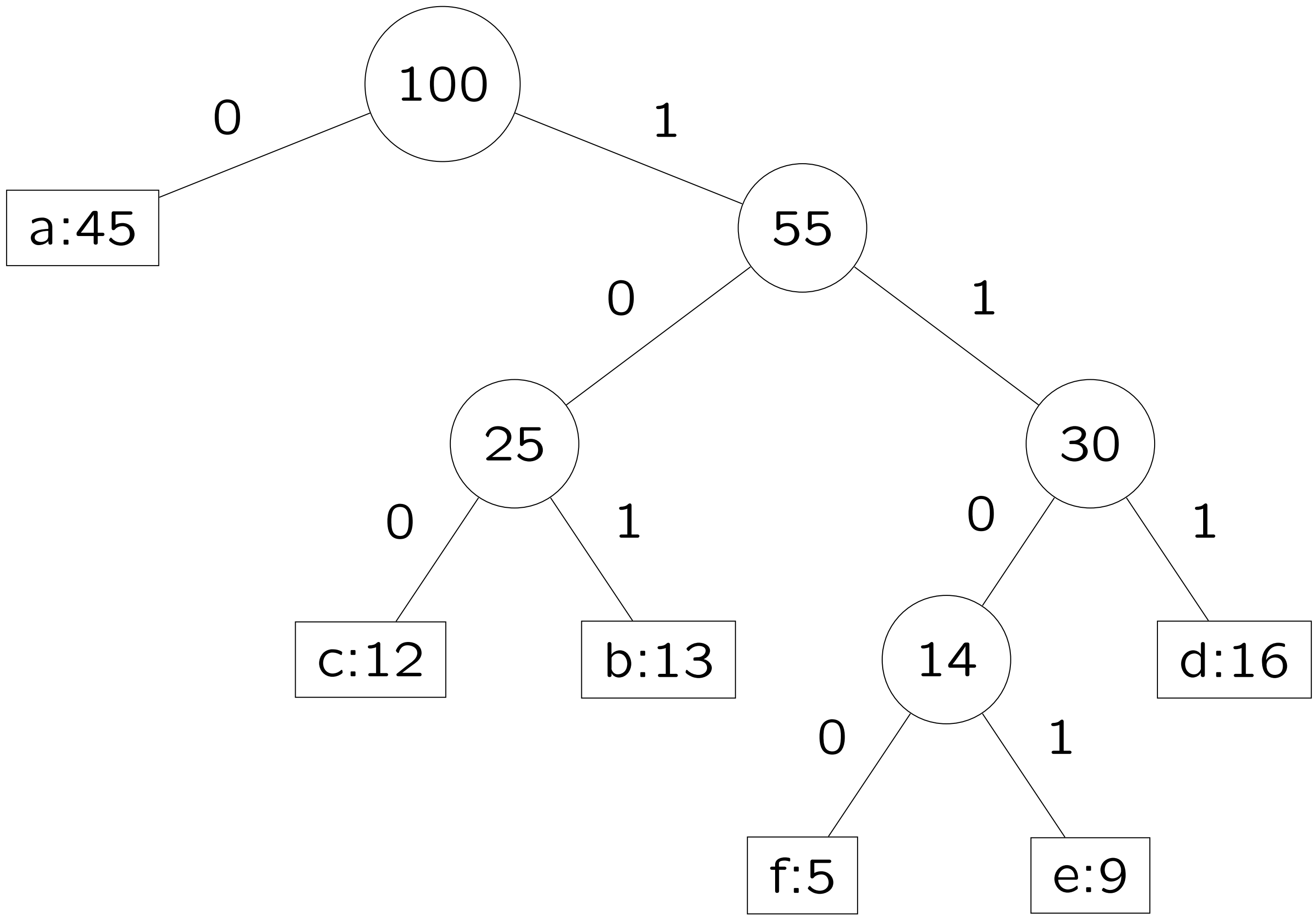
Código de Huffman

Como obter o código a partir da da árvore?

Associe a cada símbolo um código binário assim:

▷ Rotule com 0 as arestas da árvore que ligam um nó com seu filho esquerdo e com 1 as arestas que ligam um nó com seu filho direito.

Código de Huffman



Código de Huffman

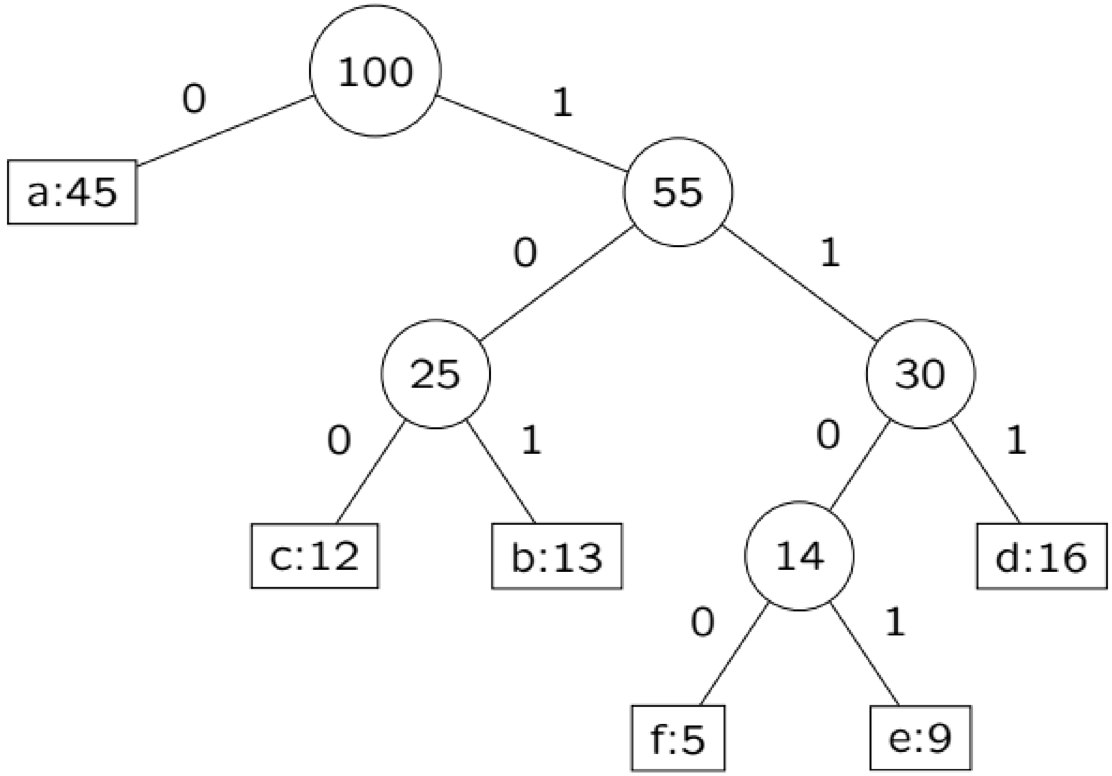
Como obter os códigos a partir da árvore?

Código de Huffman

Como obter os códigos a partir da árvore?

O código correspondente a cada símbolo é a concatenação dos bits associados às arestas do caminho da raiz até a folha correspondente ao símbolo.

Código de Huffman



Exemplo: O código b é 101

O problema do caminho mínimo

O problema do caminho mínimo

Seja G um grafo simples tal que a cada aresta (u, v) associamos um custo $w(u, v) \geq 0$.

Dados dois vértices s e t, o problema do caminho mínimo consiste em encontrar um caminho de menor custo entre s e t

O problema do caminho mínimo

Para resolver este problema, vamos estudar o algoritmo de Djikstra (1959). Como veremos, este algoritmo não só encontra o caminho mínimo de s a t, mas de s a qualquer outro vértice do grafo.

O algoritmo de Djikstra pode ser visto como uma generalização da busca em largura.

Inicialização

inicializa(G, s)

para cada vértice $v \in V[G]$ **faça**
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow NIL$
$d[s] \leftarrow 0$

Os algoritmos de caminhos mínimos associam a cada $v \in V[G]$ um valor $d[v]$ que é uma estimativa de distância dist(s, v).

O caminho pode ser recuperado por meio dos predecessores $\pi[]$

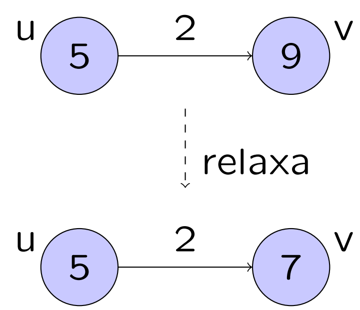
Relaxação

Tenta melhorar a estimativa $d[v]$ examinando (u, v)

```
relaxa(u, v, w)
  se d[v] > d[u] + w(u, v)
     então d[v] ← d[u] + w(u, v)
           π[v] ← u
```

Vamos assumir que $w(x, y) = \infty$ se $(x, y) \notin E(G)$

Relaxação dos vizinhos



Em cada passo o algoritmo seleciona um vértice u para cada vizinho v e aplica relaxa(u, v, w)

Algoritmo de Djikstra

Djikstra(G, w, s)

inicializa(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
enquanto $Q \neq \emptyset$ **faça**
    $u \leftarrow$ Extrai-min(Q)
    $S \leftarrow S \cup \{u\}$
    para cada vértice $v \in Adj[u]$ **faça**
        relaxa(u, v, w)

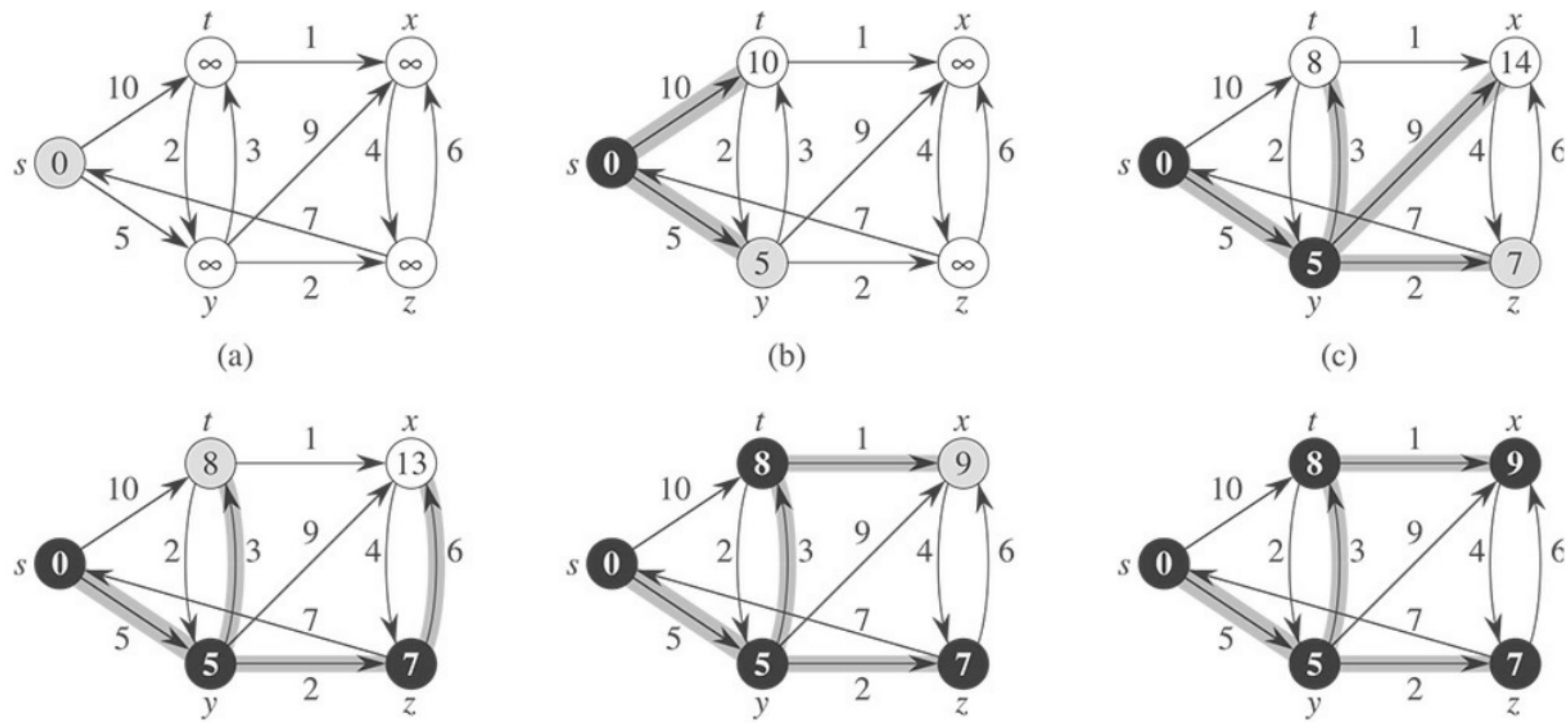
Análise do algoritmo Djikstra

▷ A complexidade de tempo de inicializa() é $O(V)$. Mas a complexidade do Djikstra depende da implementação de Q.

Usando vetor: $O(V^2)$

Usando heap:

- passo 3: construir o heap $O(V)$
- passo 5: Extrai-min() é $O(\lg |V|)$ executada $|V|$ vezes, i.e., $O(|V| \lg |V|)$
- passo 8: É preciso descer o valor no heap (passo 2 da rotina relaxa()) com tempo $O(\lg V)$. Pode haver $|E|$ chamadas de relaxa().
- **tempo total:** $O((|V| + |E|) \lg V)$



(a) (b) (c) (d) (e) (f)

Obrigado