Tópicos de Programação

Arthur Casals
(arthur.casals@usp.br)

IME - USP

Aula 10:

- Análise de algoritmos

Exercício: Revisão

"Usar o código anexo para implementar os exercícios pedidos em aula.

- Pilhas: lembrem-se de que o algoritmo de vocês deve achar um elemento na pilha, removê-lo caso exista e restaurar a pilha para seu formato original.

- Filas: idem observação acima."

Arquivos fornecidos:

- pilha.c
- fila_circular.c

Exercício: Revisão

pilha.c

```
59  //funcao para desempilhar o primeiro elemento da fila
60  int desempilha(int d, struct pilha *p) {
61      if(pilhaVazia(p) == 0) {
62          int retorno = p->item[p->topo];
63          p->item[p->topo] = -1; //usamos -1 para evitar trabalhar com valores nulos
64          p->topo = p->topo - 1;
65      }
66      return 0;
67
68  }
```

- desempilha: **NÃO** retorna o elemento desempilhado

Exercício: Revisão

pilha.c

```
72  int main()
73  {
74      struct pilha *variavel_pilha; //cria uma variavel do tipo "pilha"
75      iniciaPilha(variavel_pilha); //inicializa a pilha
76      empilha(9,variavel_pilha); //empilha o numero 9
77      printf("%d\n",variavel_pilha->item[topo(variavel_pilha)]);
78
79      return 0;
80  }
```

- main: *variavel_pilha não estava de fato inicializada!

Exercício: Revisão

fila_circular.c

```
61  int main()
62  {
63      struct fila *variavel_fila; //cria uma variavel do tipo "fila"
64      iniciaFila(variavel_fila); //inicializa a fila
65      enfileira(9,variavel_fila); //enfileira o numero 9
66      //desenfileira e imprime o numero 9
67      printf("%d\n",desenfileira(variavel_fila));
68
69      return 0;
70  }
71
```

- main: *variavel_fila não estava de fato inicializada!

Exercício: Revisão

Para os EPs:

- **NÃO** compartilhar código! Os exercícios passam por verificação de plágio.

Revisão: Estruturas de Dados

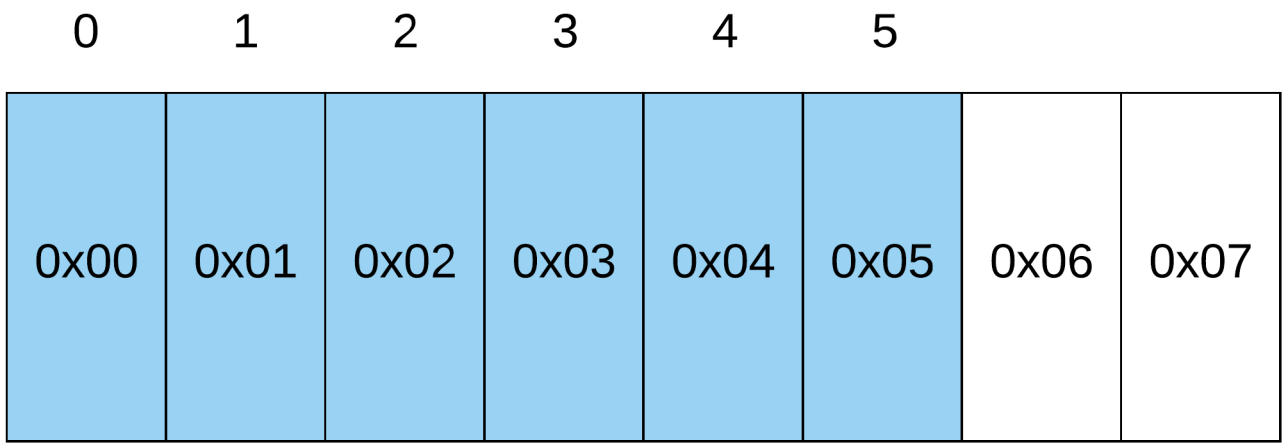
Estruturas de dados:

- Listas lineares (*arrays*)
- Listas ligadas
- Pilhas
- Filas
- Grafos
- Árvores

Revisão: Estruturas de Dados

Listas lineares *(arrays)*:

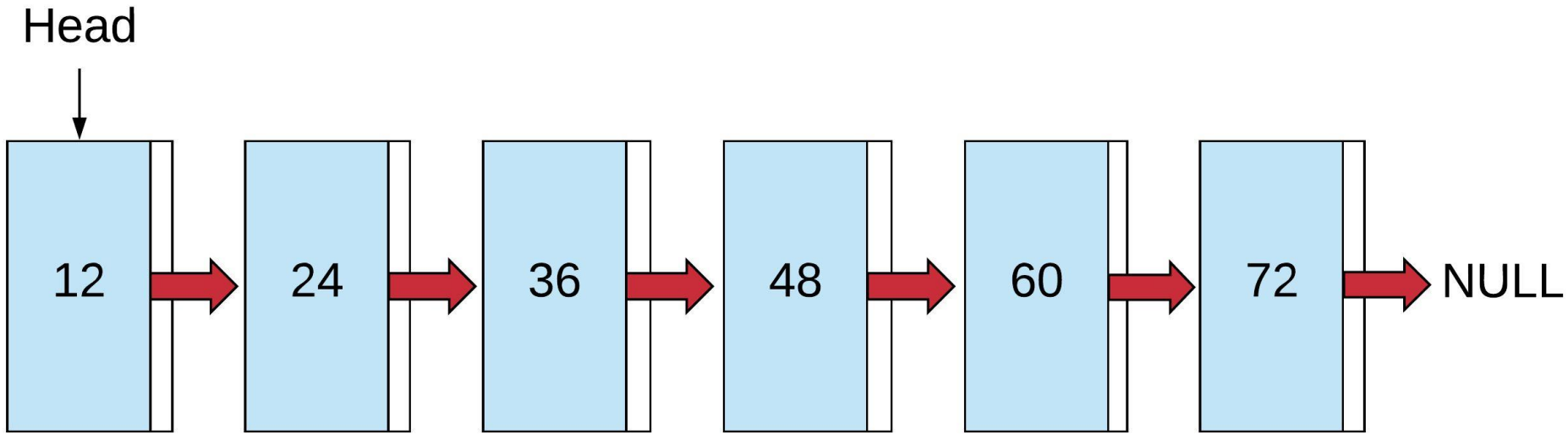
› Exemplo: imagine uma lista com seis posições. Então:



Revisão: Estruturas de Dados

Listas lineares ligadas:

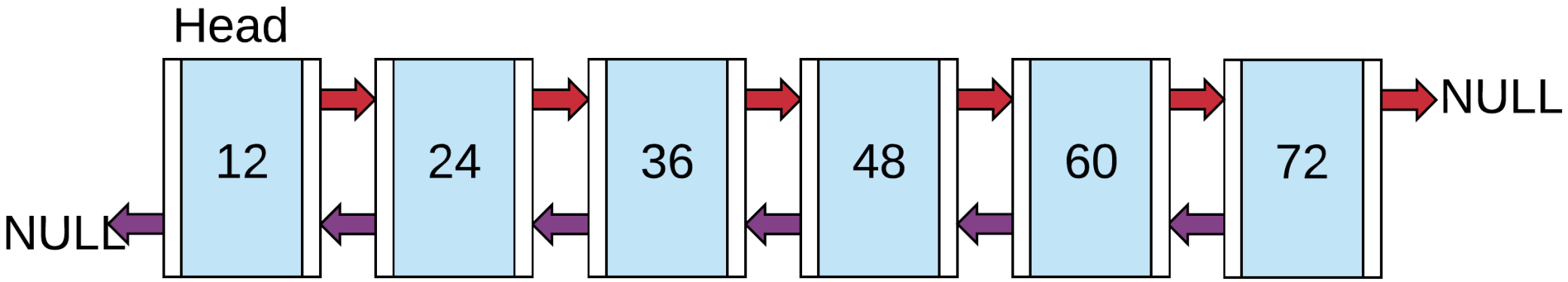
› Representação:



Revisão: Estruturas de Dados

Listas lineares duplamente ligadas:

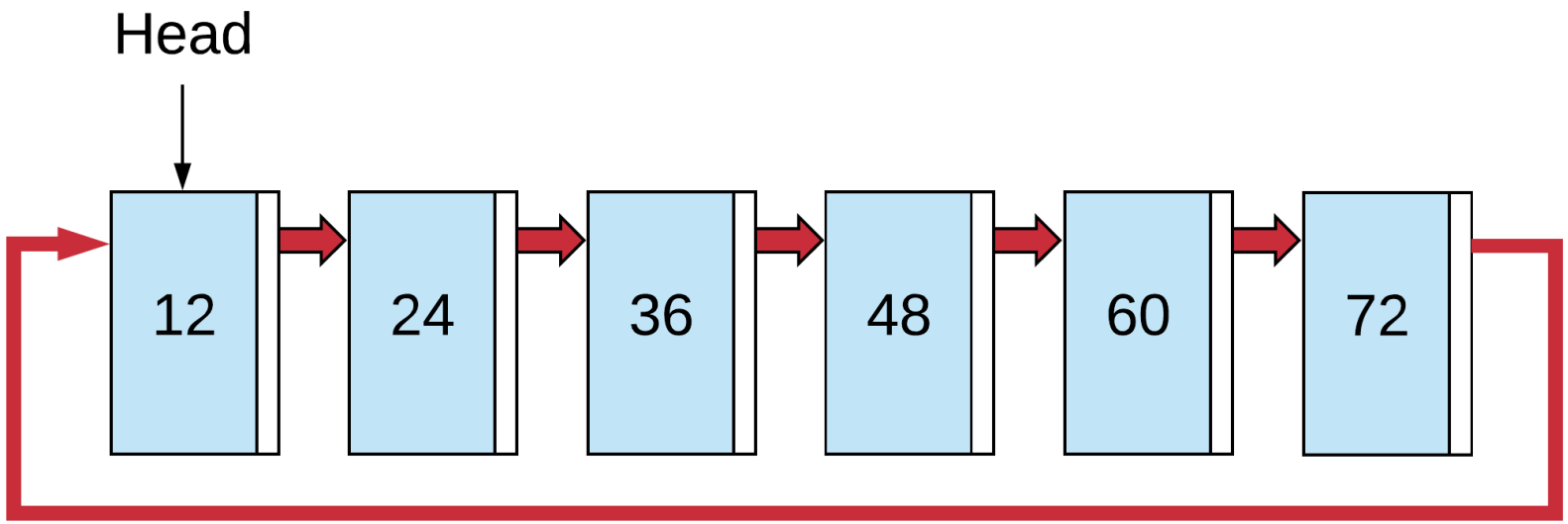
› Representação:



Revisão: Estruturas de Dados

Listas circulares ligadas:

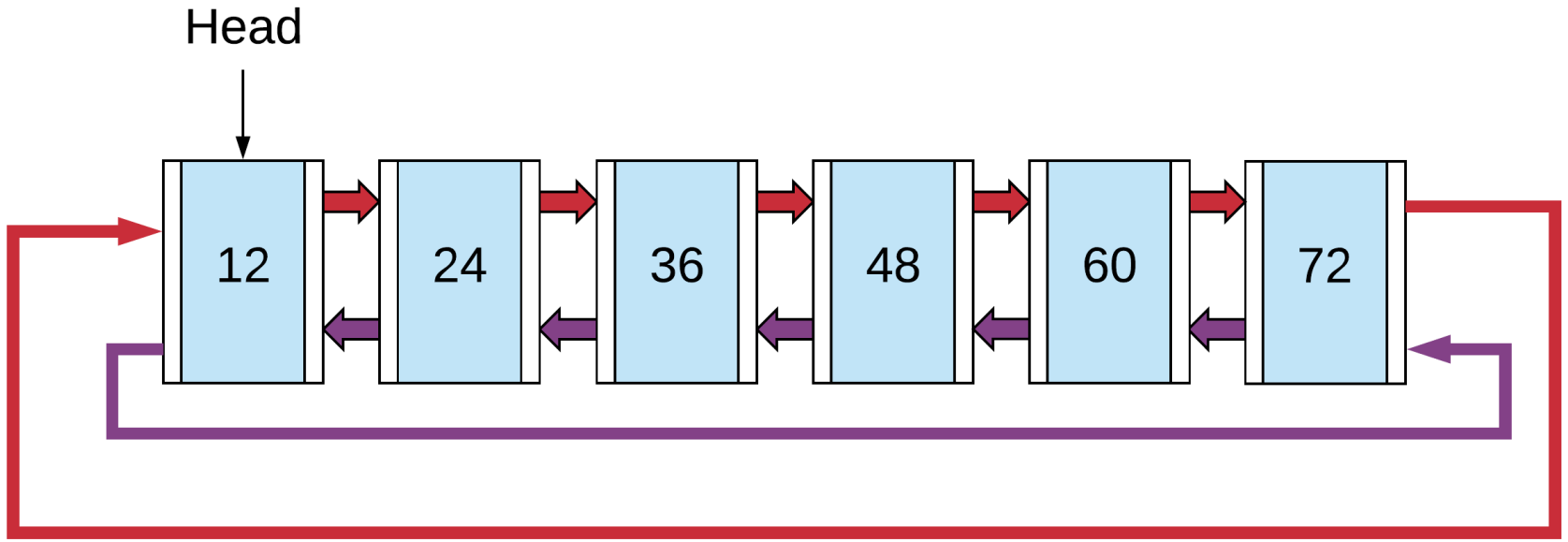
› Representação:



Revisão: Estruturas de Dados

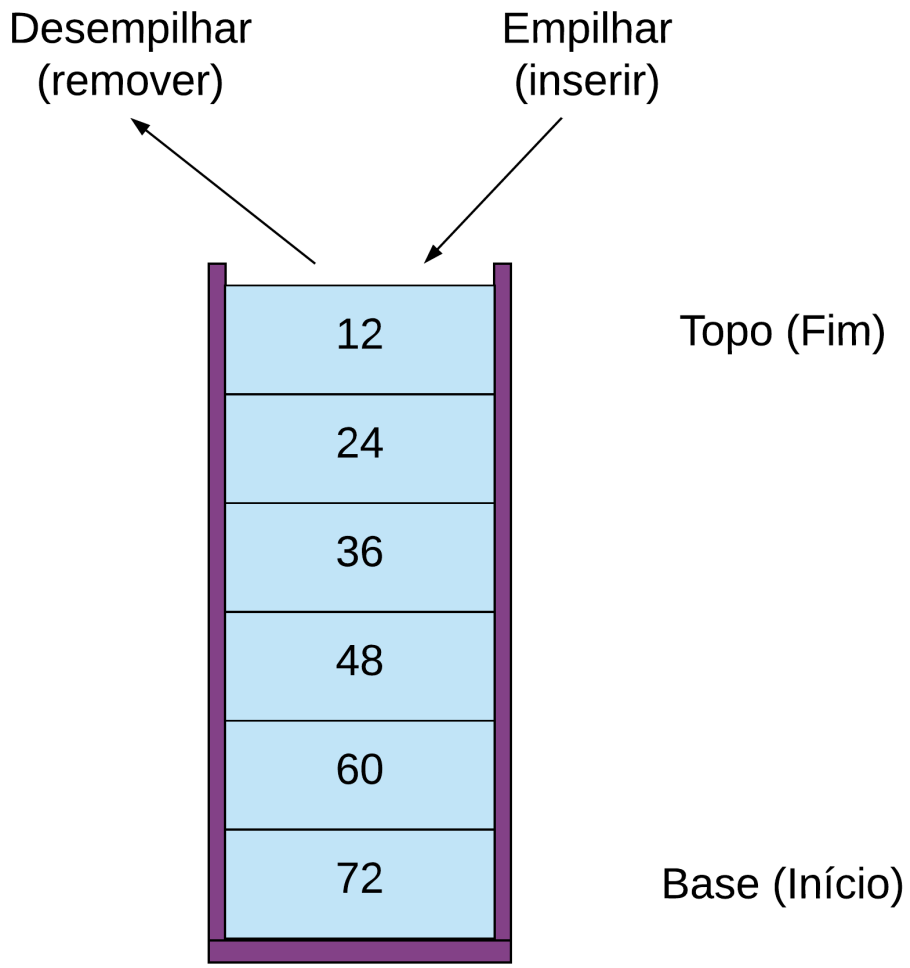
Listas circulares duplamente ligadas:

› Representação:



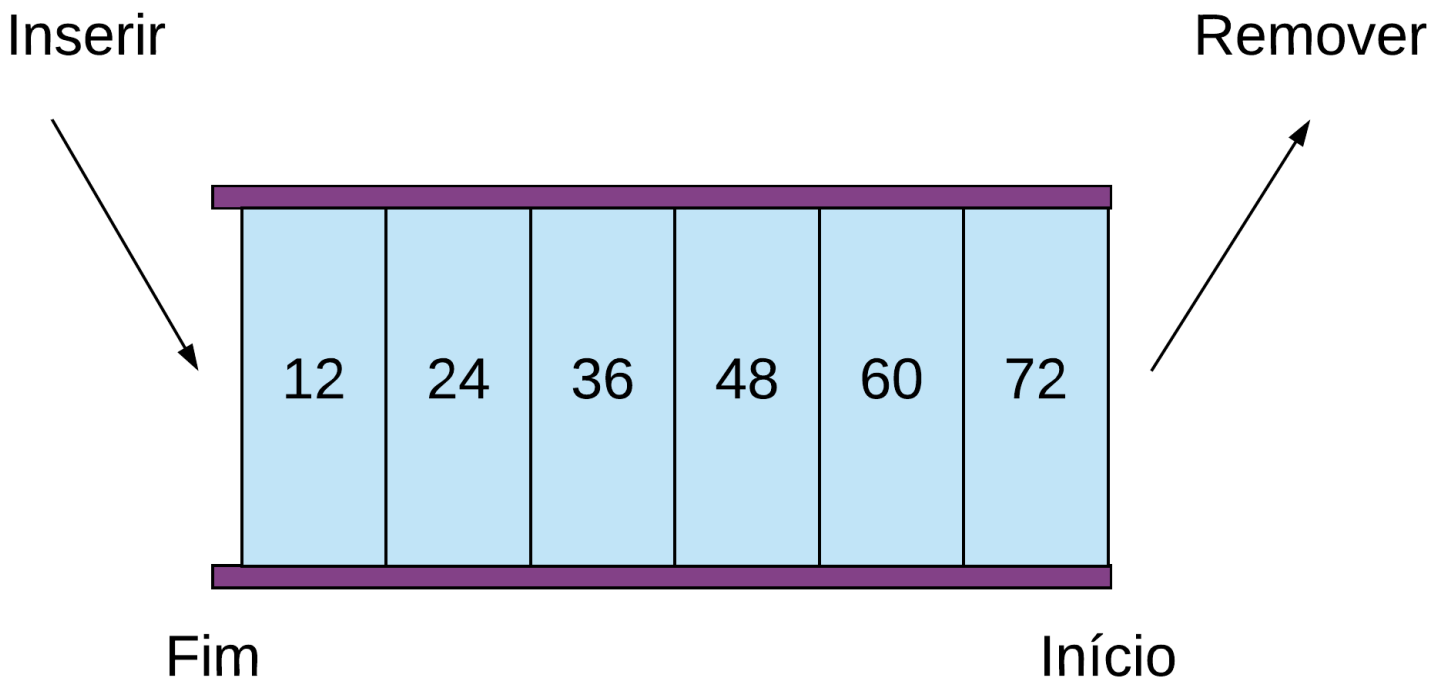
Revisão: Estruturas de Dados

Pilha:



Revisão: Estruturas de Dados

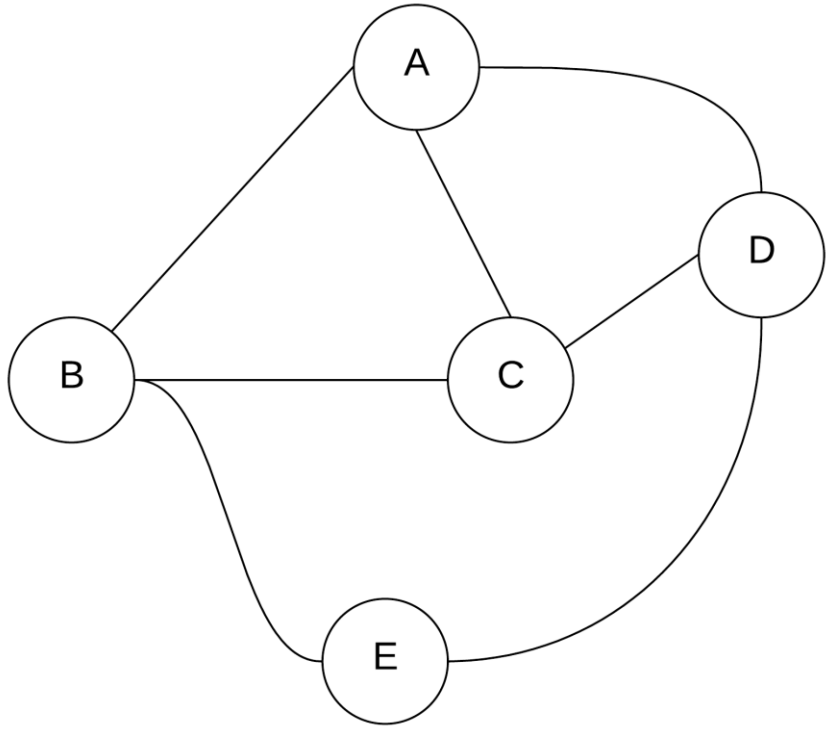
Fila:



Revisão: Estruturas de Dados

Grafos:

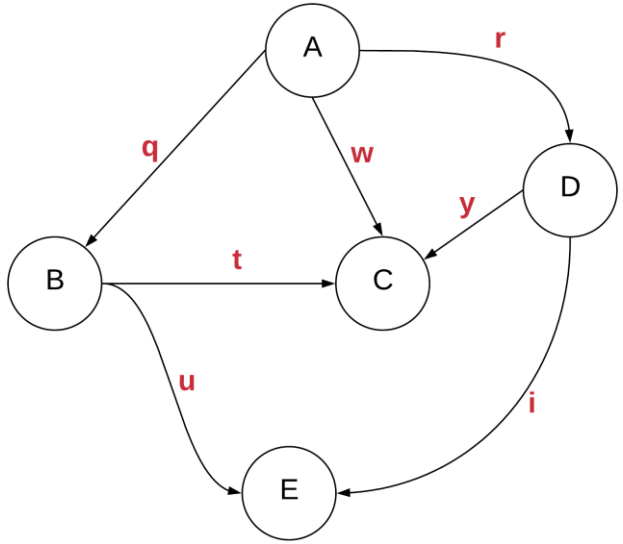
- Um conjunto de vértices conectados por arestas

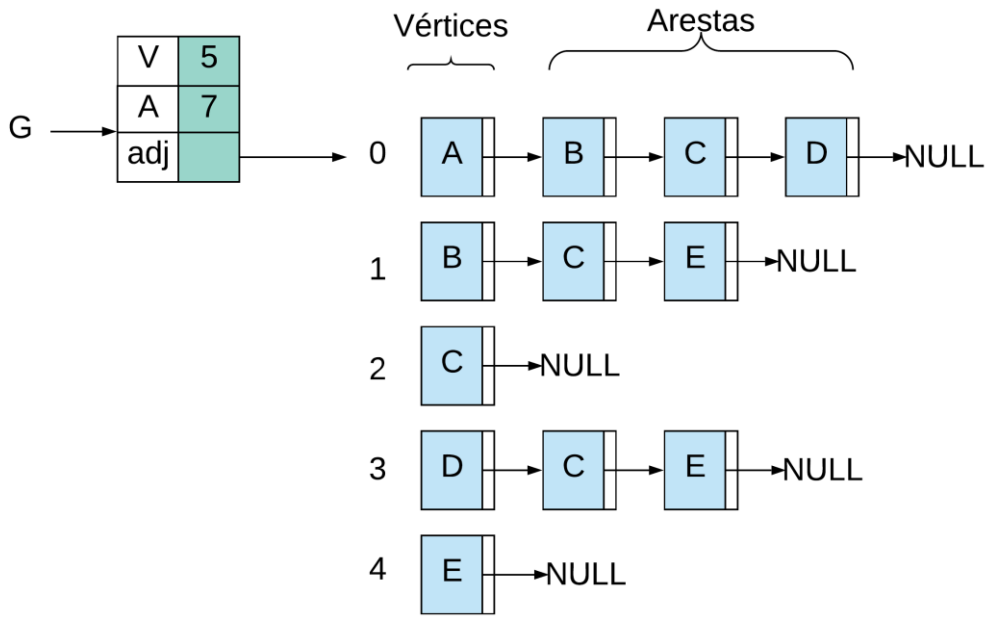


Revisão: Estruturas de Dados

Grafos:

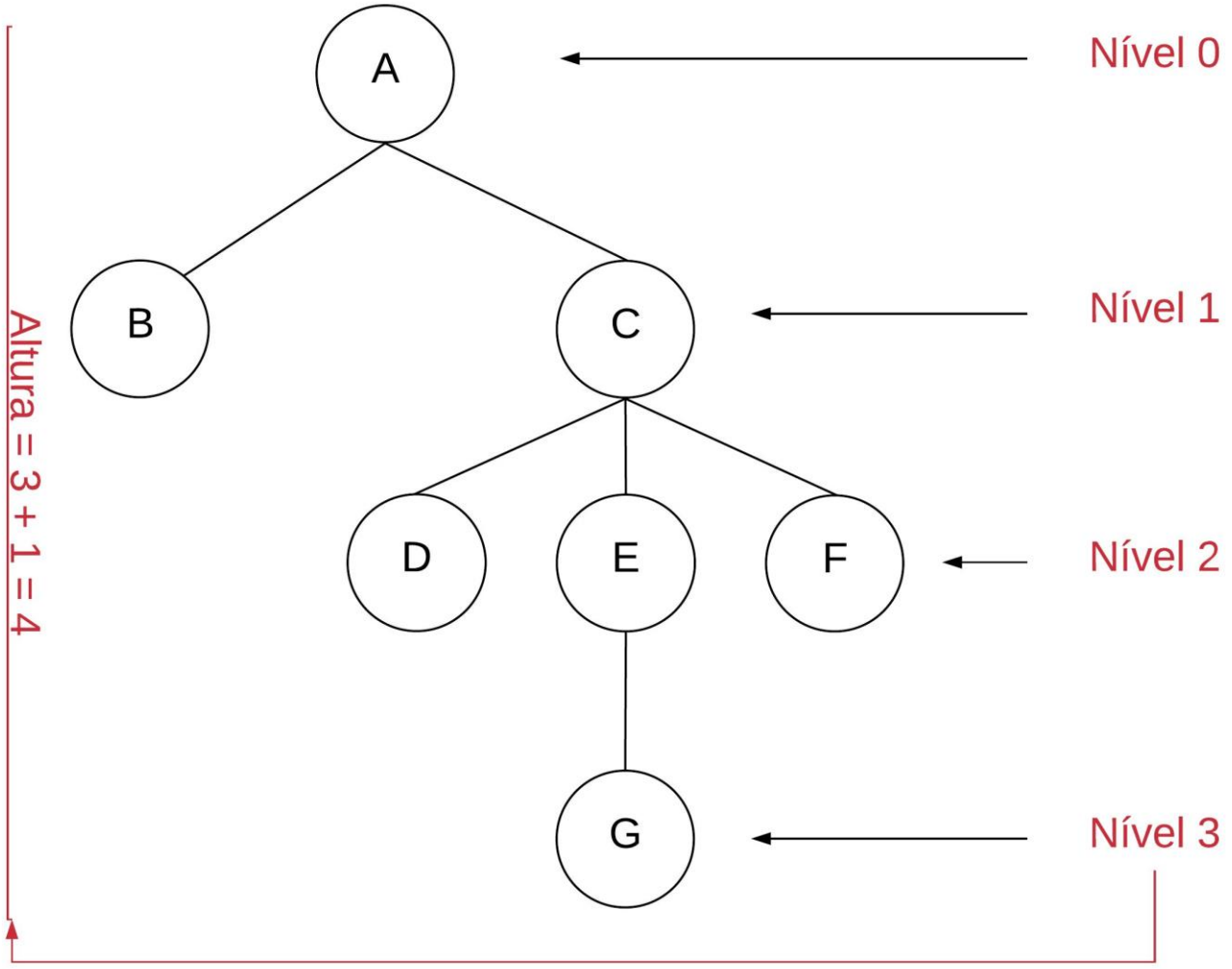
- Criando o grafo a partir da **lista de adjacência**





Na aula passada...

Árvores:



Na aula passada...

Árvores de busca binárias:

› Processamento de expressões matemáticas:

- É necessário que ocorra um *passeio* na árvore
- Passeio pode ser feito em *largura* ou em *profundidade*
  - ...e o passeio em profundidade depende da notação utilizada!

Na aula passada...

Árvores de busca binárias:

- Passeio em profundidade:

Notação	Sequência	Passeio
In-fixa	Exibir a folha esquerda (E)	**Em-ordem**
	Exibir a raiz (R)	
	Exibir a folha direita (D)	
Pre-fixa	Exibir a raiz (R)	**Pré-ordem**
	Exibir a folha esquerda (E)	
	Exibir a folha direita (D)	
Pos-fixa	Exibir a folha esquerda (E)	**Pós-ordem**
	Exibir a folha direita (D)	
	Exibir a raiz (R)	

Análise de algoritmos

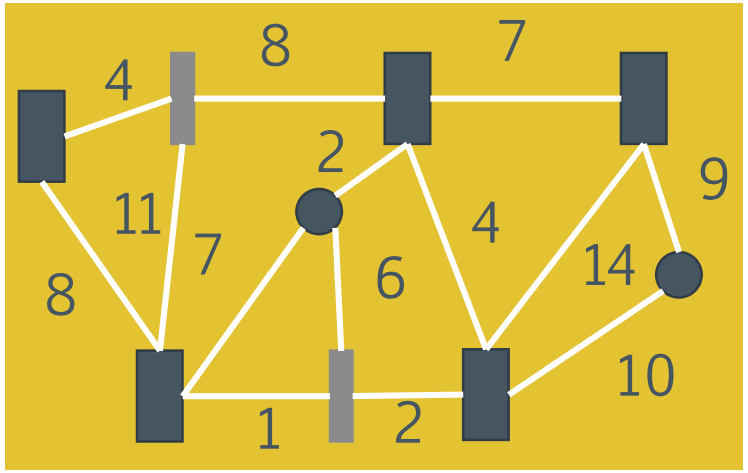
Problema: projeto de circuitos eletrônicos

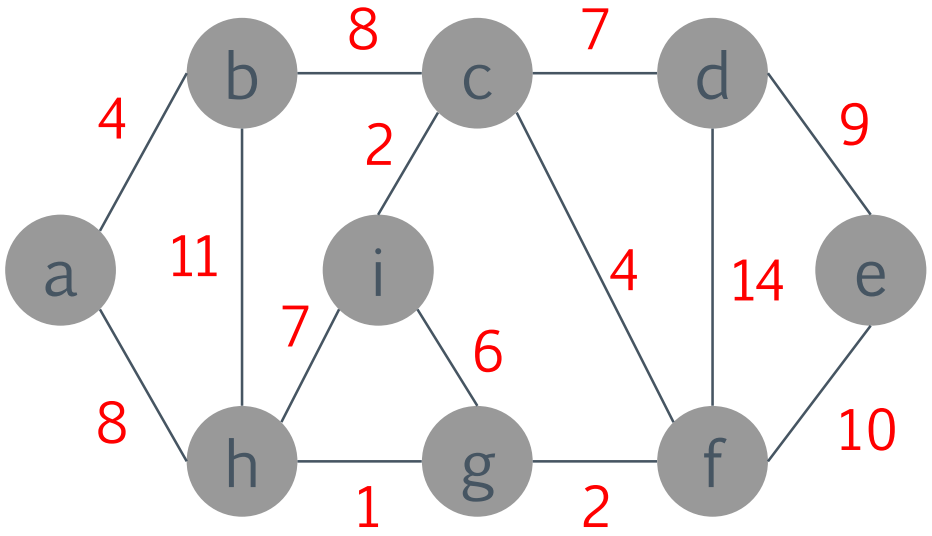
- Como conectar os pinos de vários componentes de uma placa de circuito entre si, gastando a menor quantidade de fios?

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Problema: projeto de circuitos eletrônicos



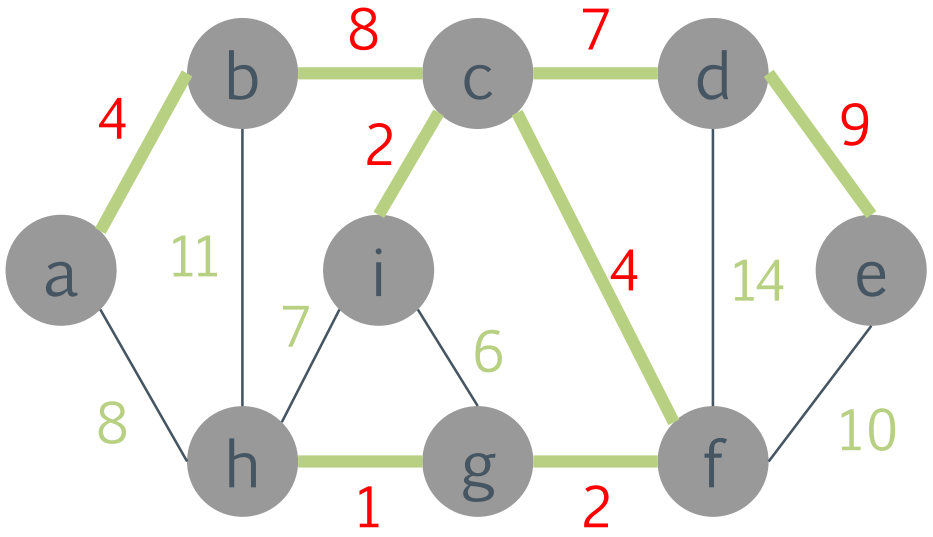


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Problema: projeto de circuitos eletrônicos

- Considerando o grafo G = (V, E) queremos encontrar um subconjunto T contido em E que conecte todos os vértices. Idealmente a soma dos pesos deve ser ***mínima***.



*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras:

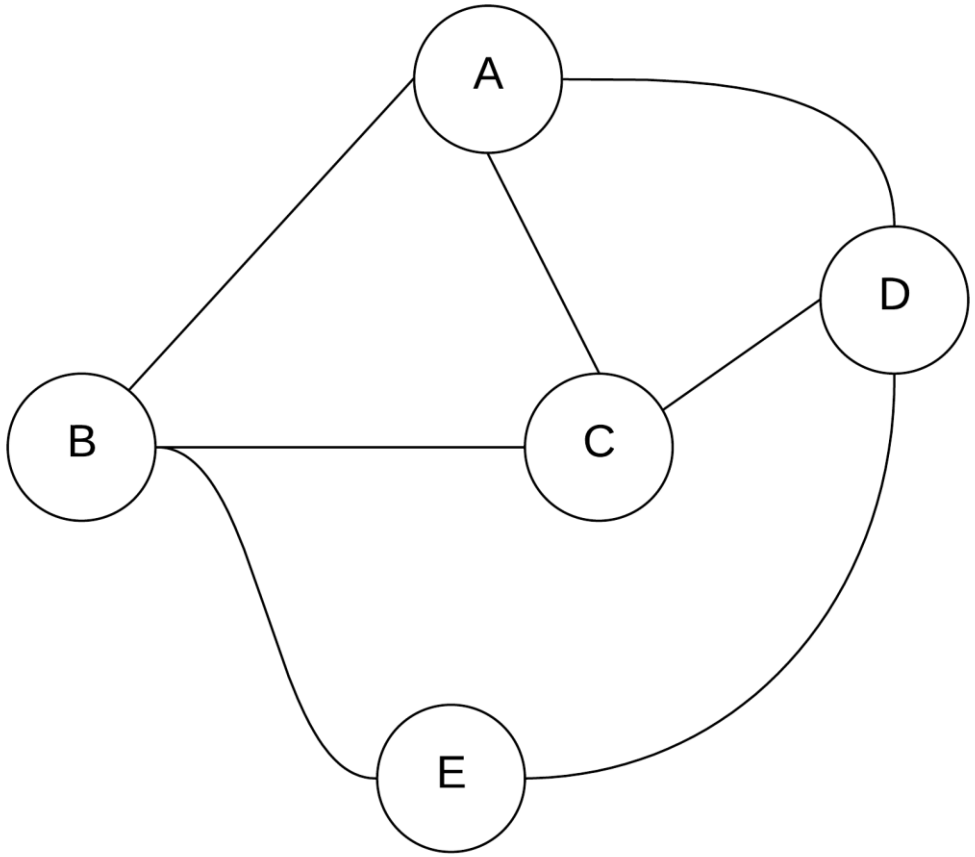
- É um subgrafo de G que conecta, sem ciclos, todos os vértices deste grafo.

- "Uma árvore de um grafo não-dirigido G é ***geradora*** se contém todos os vértices de G."*

***https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/spanningtrees.html*

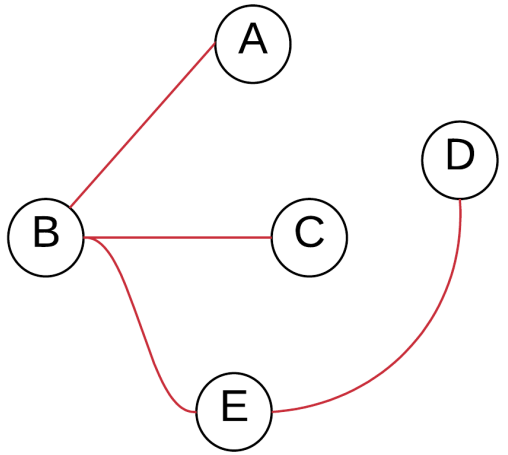
Análise de algoritmos

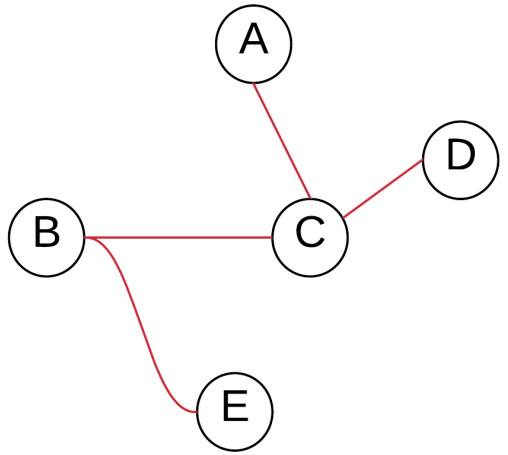
Árvores geradoras:

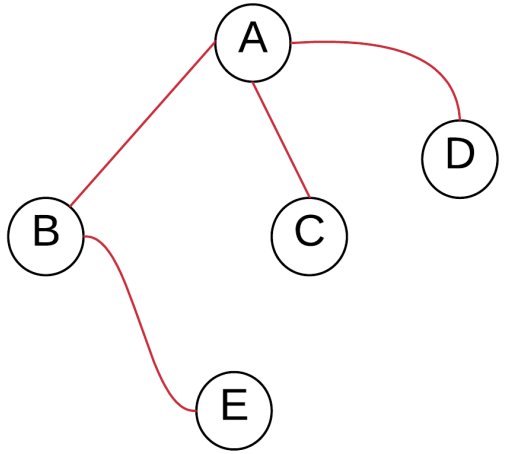


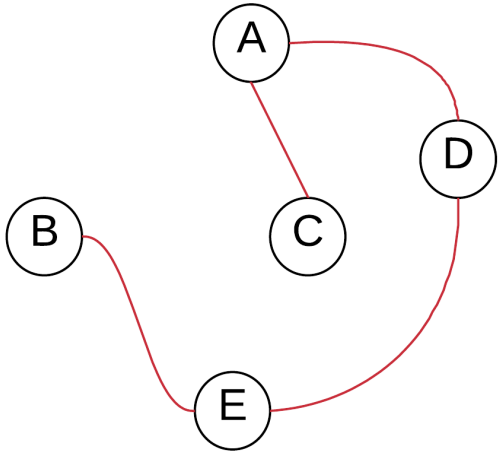
Análise de algoritmos

Árvores geradoras:









Análise de algoritmos

Árvores geradoras: Algoritmo

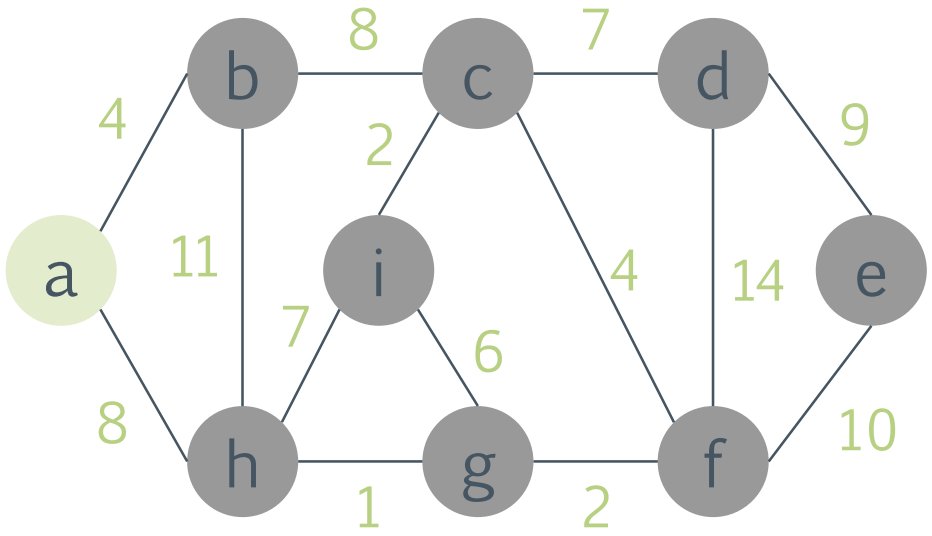
- Ideia: usando busca em largura, conectar todos os vértices adjacentes a um vértice (ainda não incluídos na árvore), impedindo ciclos
- Conceitos utilizados: Filas, Passeio em largura

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Árvores geradoras: Algoritmo

- Exemplo: começando pelo vértice **a**

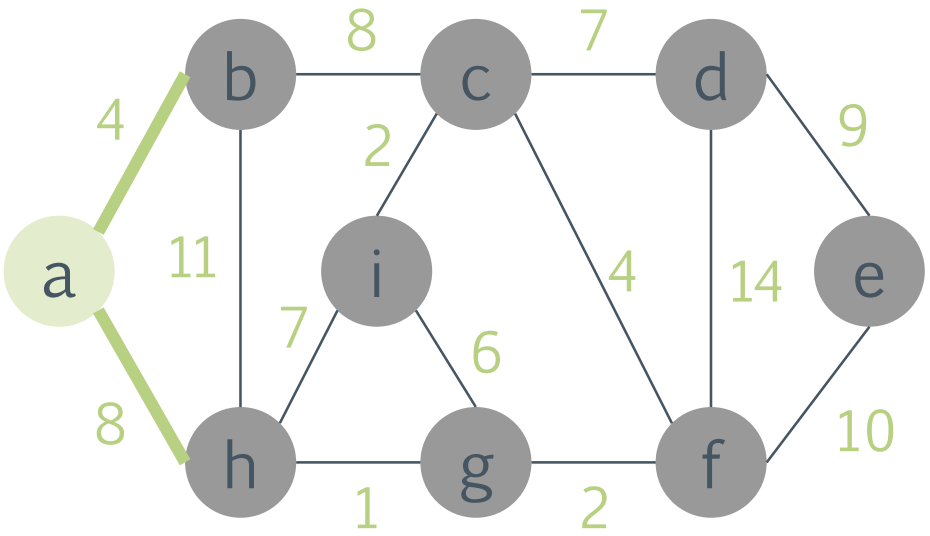


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: b, h

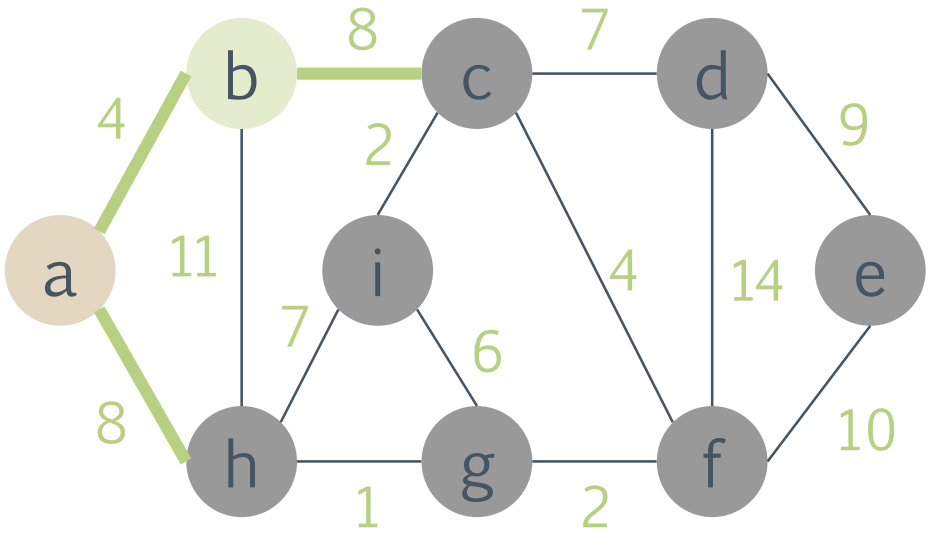


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: h, c

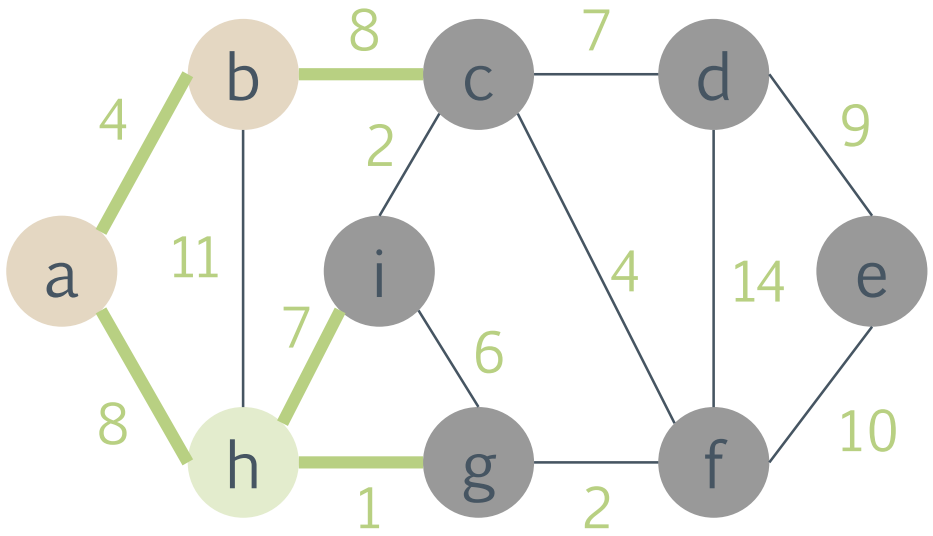


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: c, i, g

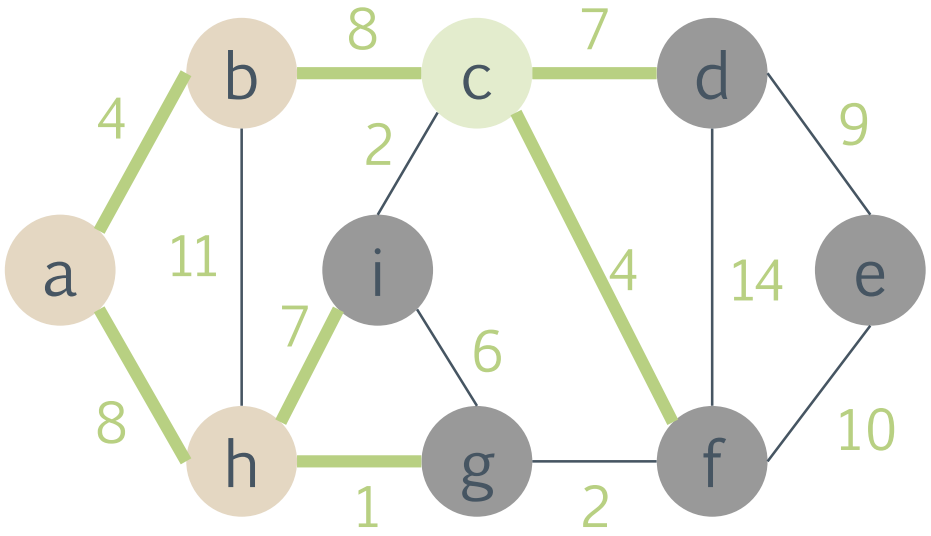


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: i, g, d, f

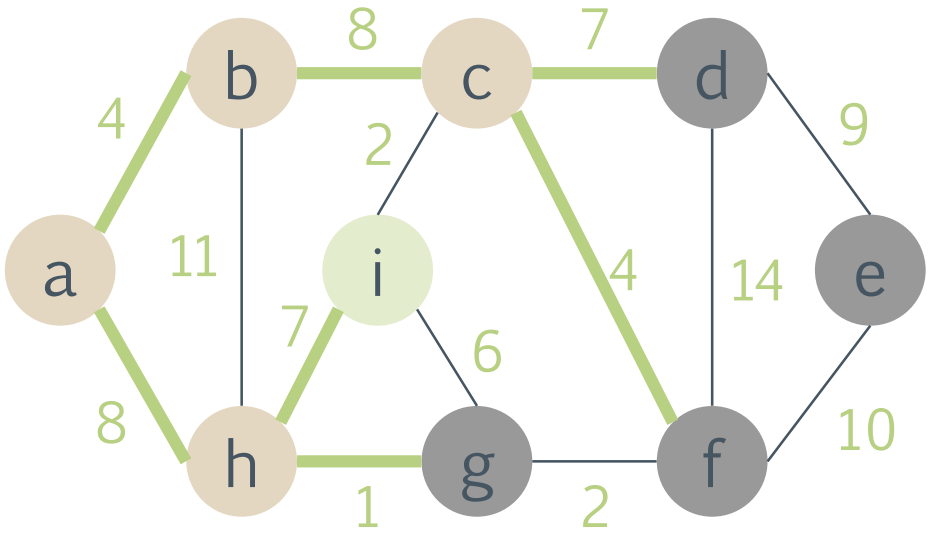


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: g, d, f

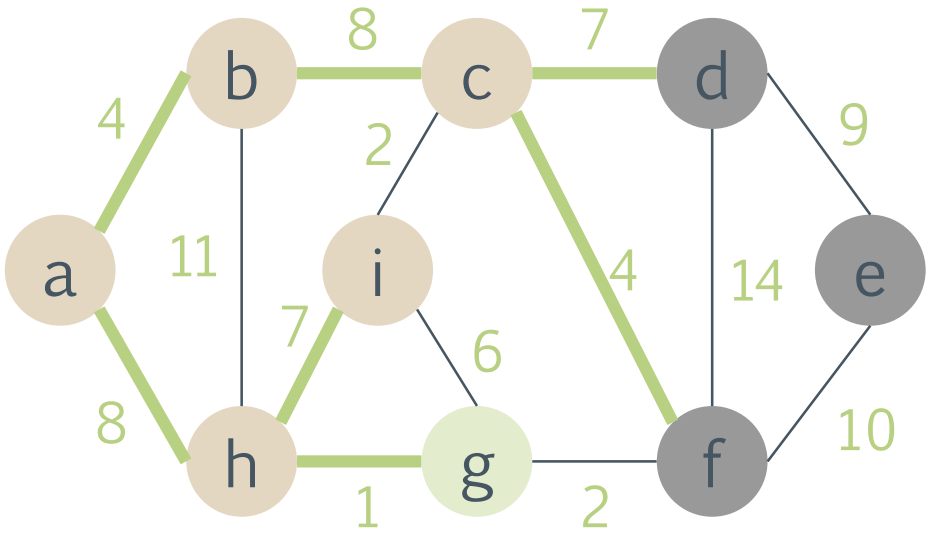


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: d, f

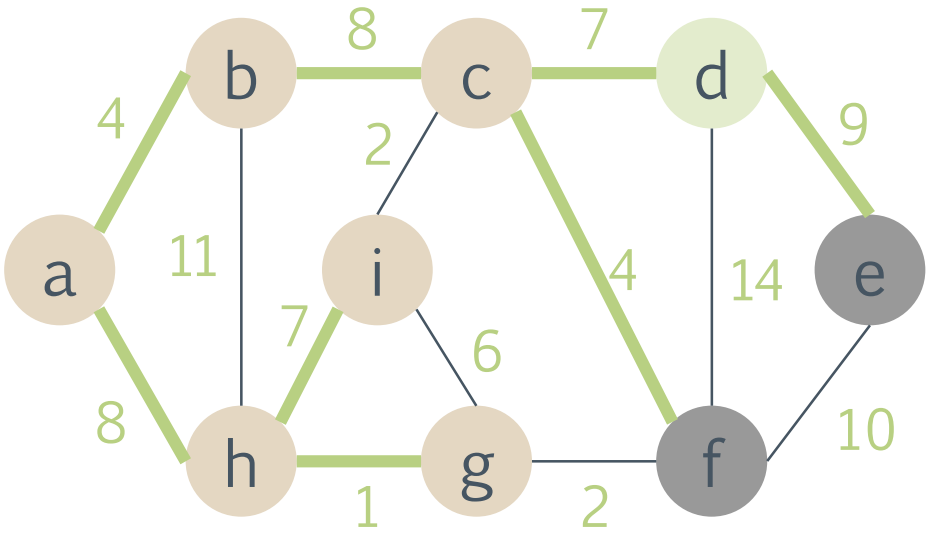


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: f, e

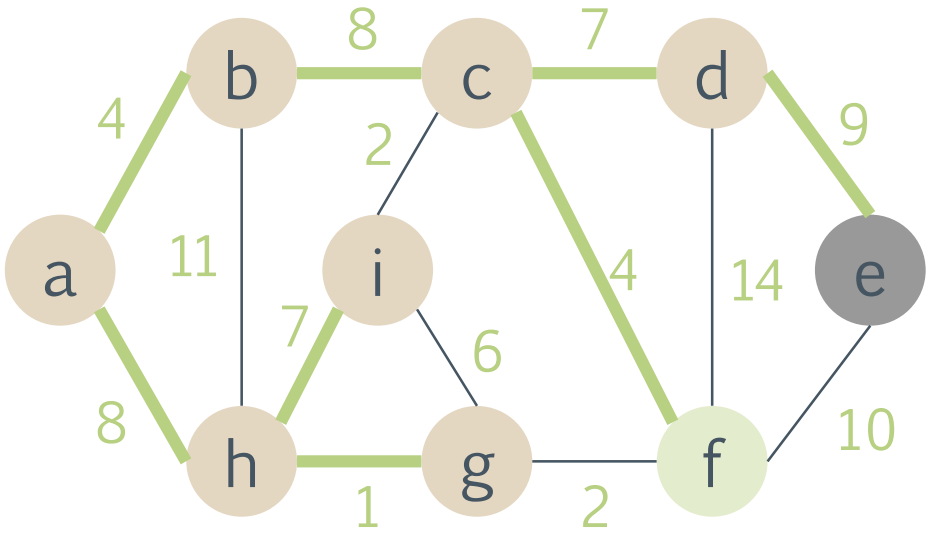


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: e

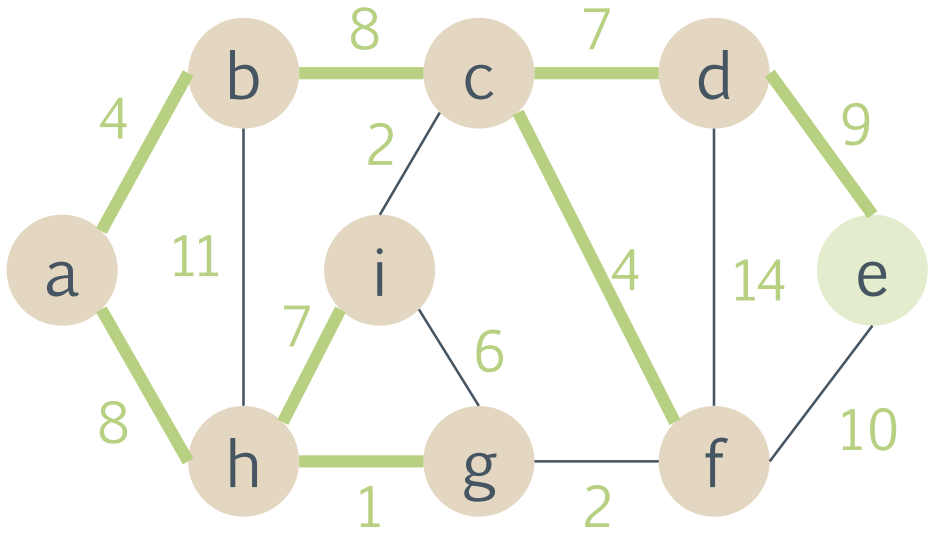


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

- Fila: (vazio) -> Soma dos pesos: 48



*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Árvores geradoras: Algoritmo

```
ST-BFS(G)
1   Seja Q uma nova Fila
2   Seja A um Conjunto de arestas
3   Seja Adicionados um Conjunto de vértices
4   Enqueue(Q, G.V[1]) // O 1º vértice
5   Adicionados = Adicionados ∪ {G.V[1]}
6   while not Queue-Empty(Q)
7     atual = Dequeue(Q)
8     for each u ∈ G.V – Adicionados
9       if G.Adjacencia[atual][u] > 0
10        Adicionados = Adicionados ∪ {atual}
11        A = A ∪ {(atual, u)}
12        Enqueue(Q, u)
13  return A
```

*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

...mas a árvore gerada é *mínima*?

- Existe uma árvore geradora cuja soma dos pesos seja menor que 48?

- Como gerar uma árvore cuja soma dos pesos seja mínima?

Análise de algoritmos

Árvores geradoras mínimas:

"Seja G um grafo não-dirigido com custos nas arestas. Os custos podem ser positivos ou negativos. O custo de um subgrafo não-dirigido T pertencente a G é a soma dos custos das arestas de T."*

*https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/mst.html

Análise de algoritmos

Árvores geradoras mínimas:

- **Definição:** "Uma árvore geradora mínima de G é qualquer árvore geradora de G que tenha custo mínimo.."*
- Algoritmos: **Kruskal** e **Prim**

*https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/mst.html

Análise de algoritmos

Árvores geradoras mínimas: Algoritmo de Kruskal

- Obtém um conjunto de árvores (floresta) geradoras mínimas

- Funciona com grafos conexos e desconexos

- Se o grafo for conexo, o conjunto é unitário

- **Floresta** é qualquer grafo acíclico, conexo ou desconexo (ou seja, é um conjunto de árvores)

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Árvores geradoras mínimas: Algoritmo de Kruskal

- Obtém um conjunto de árvores (floresta) geradoras mínimas

- Funciona com grafos conexos e desconexos

- Se o grafo for conexo, o conjunto é unitário

- **Floresta** é qualquer grafo acíclico, conexo ou desconexo (ou seja, é um conjunto de árvores)

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Árvores geradoras mínimas: Algoritmo de Kruskal

-Inicialmente o grafo é um conjunto de árvores (floresta), uma para cada vértice do grafo

- A cada passo conecta duas árvores da floresta que tenham entre elas a aresta de menor peso

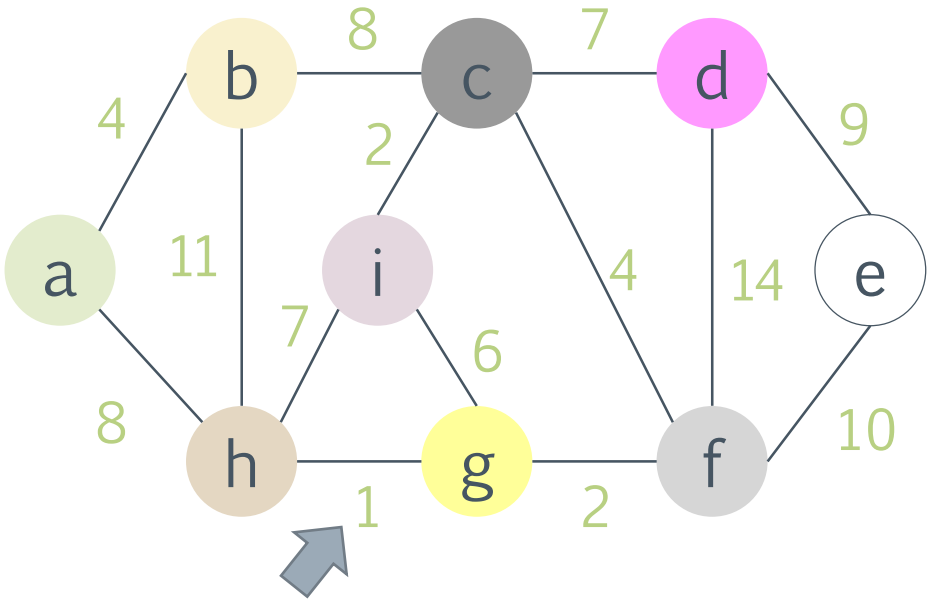
- Ao conectar 2 árvores, elas viram 1 árvore só

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- 9 árvores
- Aresta de peso mínimo: (h, g)

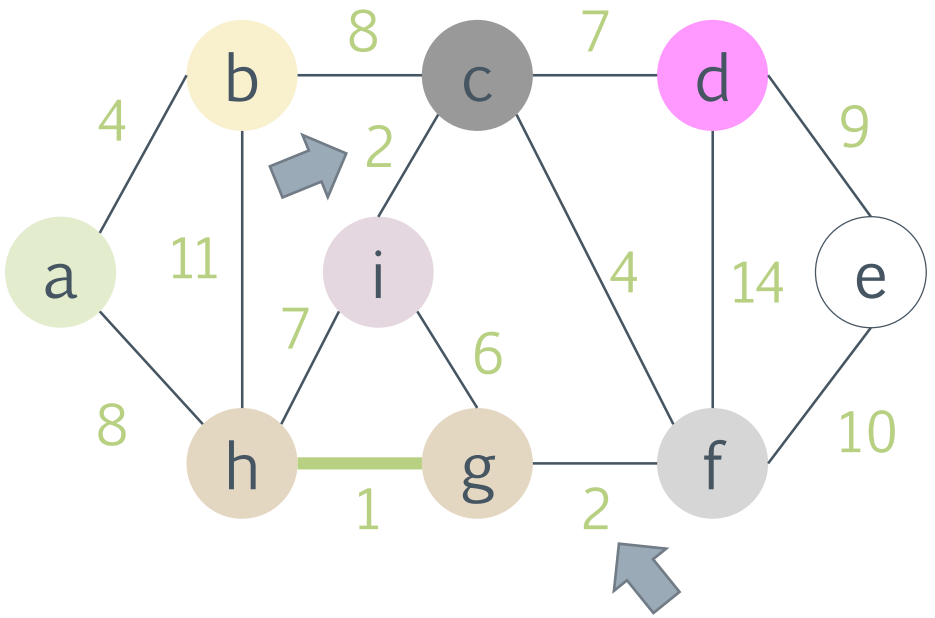


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (c, i), (f, g)

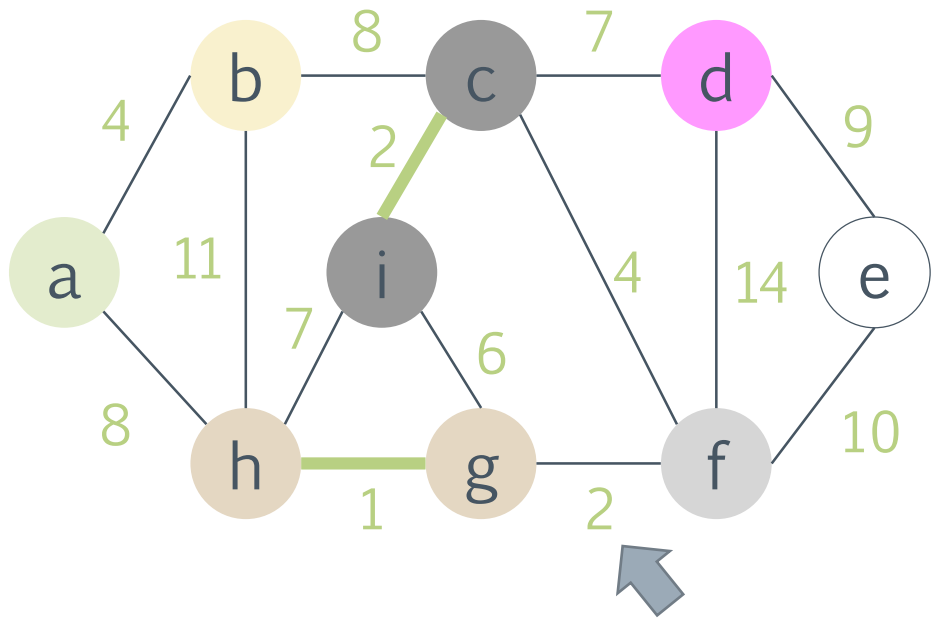


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (f, g)

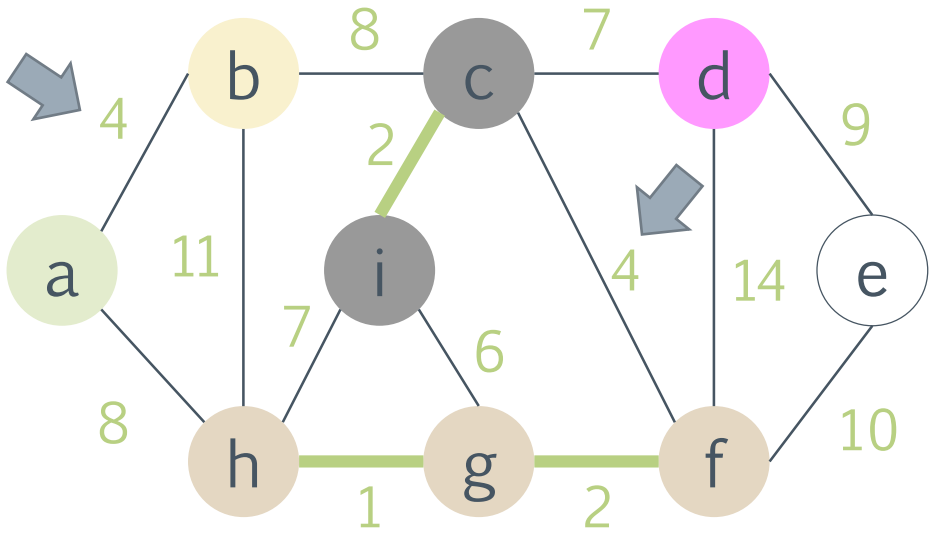


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (a, b), (c, f)

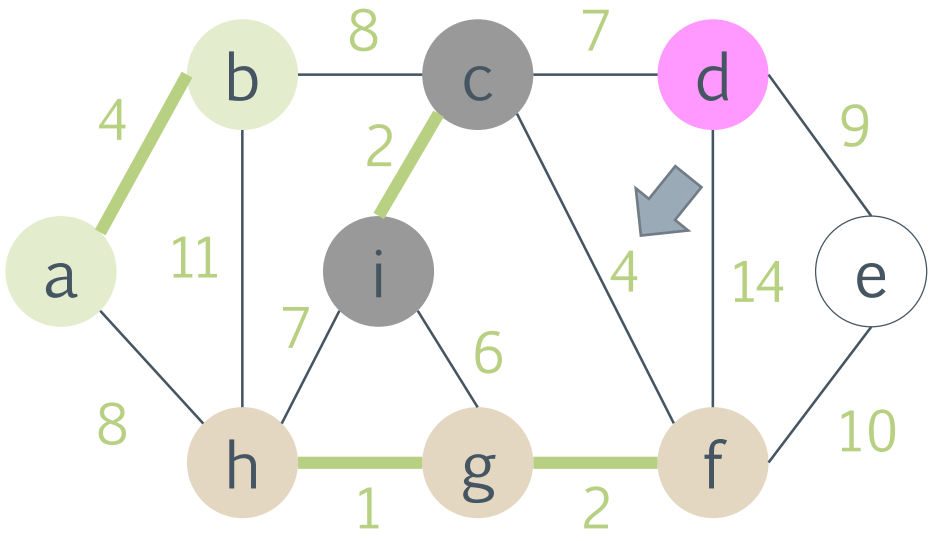


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (c, f)

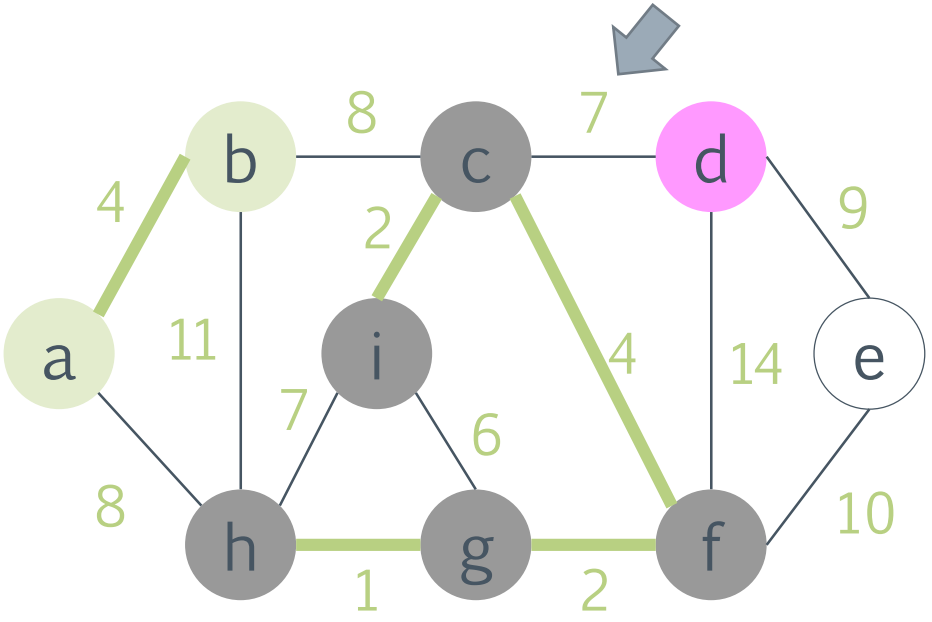


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (c, d)

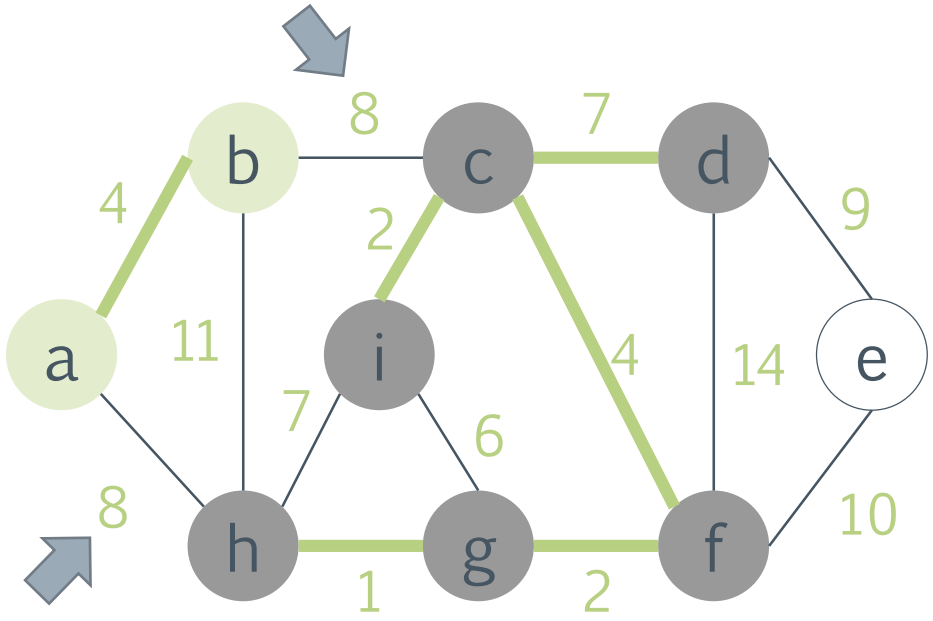


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (a, h), (b, c)

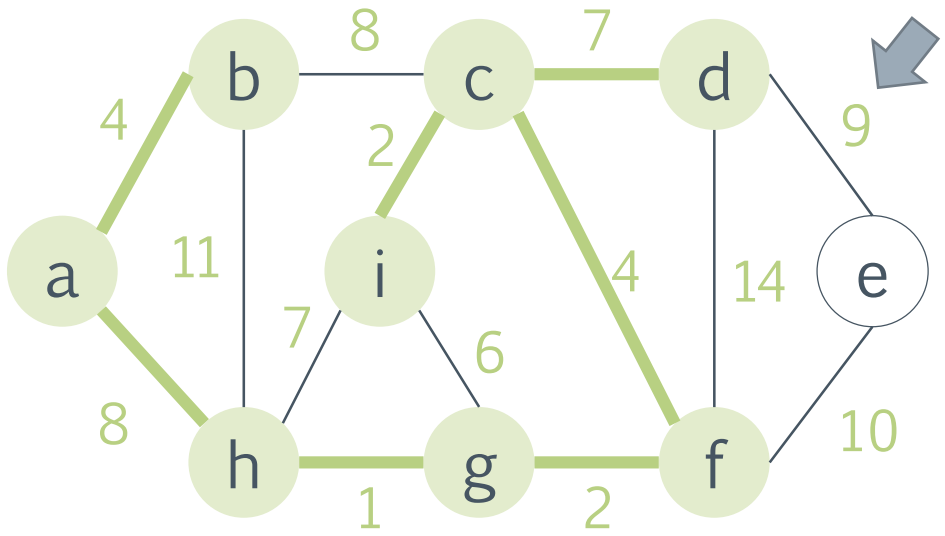


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Aresta de peso mínimo: (d, e)

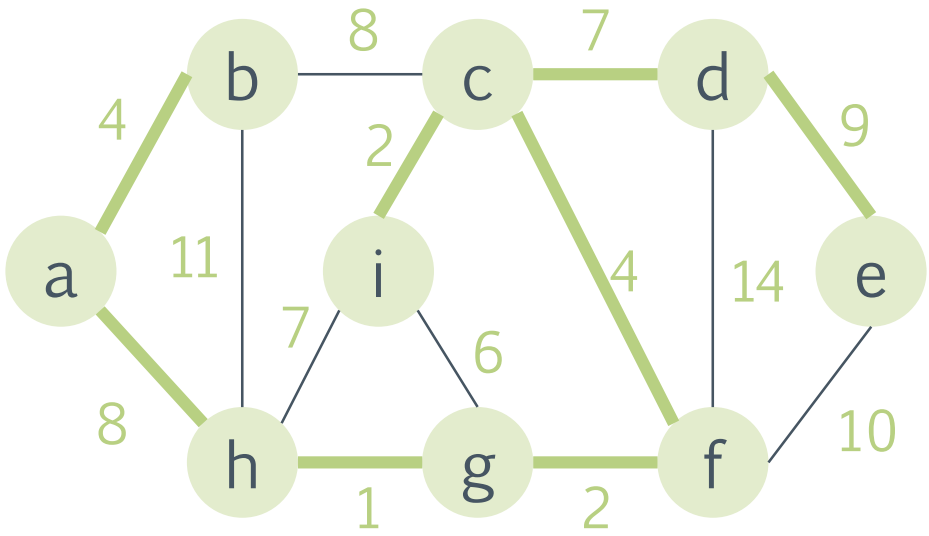


*adaptação: PCS3110, 2017, EPUSP

Análise de algoritmos

Exemplo: Algoritmo de Kruskal

- Soma dos pesos: **37**



*adaptação: PCS3110, 2017, EPUSP