

Algoritmos em Grafos

Edson Norberto Cáceres
Dept. de Computação e Estatística
UFMS
edson@dct.ufms.br

22/04/2005

1

Grafos: Definições (1)

- Um *grafo* $G = (V, E)$
 - V = conjunto de vértices
 - E = conjunto de arestas = subconjunto de $V \times V$
 - Desta forma $|E| = O(|V|^2)$
- Em um *grafo não direcionado*:
 - Aresta $(u, v) =$ Aresta (v, u)
 - Sem *self-loops*
- Em um *grafo direcionado*:
 - Aresta (u, v) vai do vértice u para o vértice v , representado por $u \rightarrow v$

22/04/2005

3

Aula de Hoje

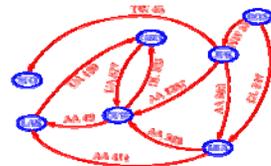
- Algoritmos Elementares em Grafos
 - Busca em Grafos
 - Aplicação
- Árvores Geradoras Mínimas
 - Algoritmo de Kruskal
 - Algoritmo de Prim

22/04/2005

2

Estrutura de Dados para Grafos

- Como podemos representar um grafo?
 - Temos que armazenar os vértices e as arestas.



22/04/2005

4

Representações para Grafos

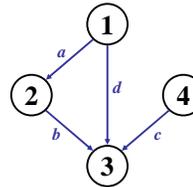
- $V = \{1, 2, \dots, n\}$
- Uma *matriz de adjacências* representa o grafo como uma matriz A $n \times n$:
 - $A[i, j] = 1$ se a aresta $(i, j) \in E$ (ou peso da aresta)
 - $= 0$ se a aresta $(i, j) \notin E$

22/04/2005

5

Grafos: Matriz de Adjacências

- Exemplo:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

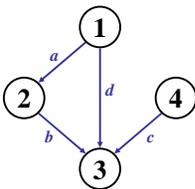
- $A: O(V^2)$

22/04/2005

7

Grafos: Matriz de Adjacências

- Exemplo:



A	1	2	3	4
1				
2				
3				
4				

??

22/04/2005

6

Grafos: Matriz de Adjacências

- A matriz de adjacências é uma representação densa
 - Utiliza muito espaço para grafos grandes
 - Pode ser muito eficiente para grafos pequenos
- Muitos grafos interessantes são esparsos
 - Ex., grafos planares, temos $|E| = O(|V|)$ (fórmula de Euler)

22/04/2005

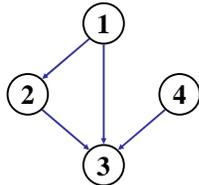
8

Representações para Grafos

- Uma lista de adjacência armazena para cada vértice $v \in V$ uma lista dos vértices adjacentes a

- Exemplo:

- $Adj[1] = \{2,3\}$
- $Adj[2] = \{3\}$
- $Adj[3] = \{\}$
- $Adj[4] = \{3\}$



- Lista de adjacências utiliza espaço $O(V+E)$

22/04/2005

9

Grafos: Definições (2)

- Um *grafo ponderado* associa pesos às suas arestas
 - Ex., um mapa rodoviário: as arestas representam a distância
- Expressamos os tempo de execução em termos de $|E|$ e $|V|$
 - Se $|E| \approx |V|^2$ o grafo é *denso*
 - Se $|E|$ *muito menor que* $|V|^2$ o grafo é *esparso*
- Dependendo do grafo (denso ou esparso) uma estrutura de dados específica pode ser mais adequada.
- Matriz de Adjacências** e **Lista de Adjacências**.

22/04/2005

11

Aplicações

- Circuitos Eletrônicos, redes de distribuição
- Transporte e redes de comunicação
- Modelagem de todo tipo de relacionamento (entre componentes, pessoas, processos, conceitos)



22/04/2005

10

Algoritmos para Busca em Grafos

- Entrada: um grafo $G = (V, E)$, direcionado ou não
- Objetivo: explorar de forma metódica toda aresta e todo vértice
- Saída: construir uma *árvore* no grafo
 - Escolha um vértice como raiz
 - Escolha determinadas arestas para produzir uma árvore (pode ser uma *floresta* se o grafo não for *conexo*)
- Aplicações
 - Compiladores
 - Computação Gráfica
 - Maze-solving
 - Mapping
 - Redes: roteamento, busca, clustering, etc.

22/04/2005

12

Busca em Largura (BFS)

- PRIM (MST) e Dijkstra (SSSP) usam idéias parecidas.
- Uma Busca em largura “explora” os componentes *conexos de um grafo*, obtendo uma *árvore geradora* com diversas propriedades.
- BFS em um grafo não direcionado é como descobrir a saída em um labirinto

22/04/2005

13

Busca em Largura (BFS)

- Associamos cores aos vértices para guiar o algoritmo
 - Vértices **brancos** ainda não foram descobertos
 - Todos vértices começam com branco
 - Vértices **cinza** estão descobertos mas não totalmente explorados
 - Podem ser adjacentes a vértices brancos
 - Vértices **pretos** estão descobertos e totalmente explorados
 - Somente são adjacentes a vértices pretos ou cinza
- Explore os vértices varrendo a lista de adjacências dos vértices cinza

22/04/2005

15

Busca em Largura (BFS)

- Atribua a distância 0 ao vértice inicial s .
- Na primeira rodada, a corda é desenrolada com 1 unidade de comprimento (aresta), e todas os vértices que estão a apenas uma aresta de distância da âncora são visitados (**descobertos**), e atribuídos distâncias 1
- Na segunda rodada, todos os novos vértices que podem ser atingidos com uma corda de 2 unidades de comprimento (2 arestas) são visitados e recebem distância 2
- O algoritmos continua até que todo vértice tenha um rótulo
- O rótulo de qualquer vértice v corresponde ao comprimento do menor caminho (em termos de arestas) de s a v

22/04/2005

14

Algoritmo BFS

```
BFS(G,s)
01 for each vertex u ∈ V[G]-{s}
02   color[u] ← white
03   d[u] ← ∞
04   π[u] ← NIL
05 color[s] ← gray
06 d[s] ← 0
07 π[s] ← NIL
08 Q ← {s}
09 while Q ≠ ∅ do
10   u ← head(Q)
11   for each v ∈ Adj[u] do
12     if color[v] = white then
13       color[v] ← gray
14       d[v] ← d[u] + 1
15       π[v] ← u
16       Enqueue(Q,v)
17   Dequeue(Q)
18   color[u] ← black
```

Inicialize todos vértices

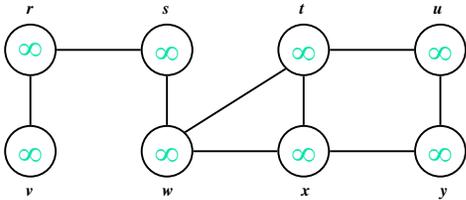
Inicialize BFS com s

Visite todos filhos de u antes de visitar qualquer filho de um filho

22/04/2005

16

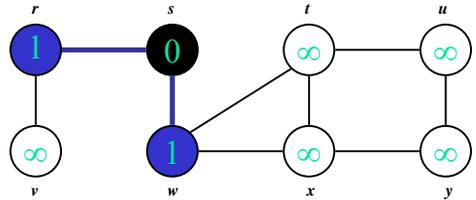
Busca em Largura: Exemplo



22/04/2005

17

Busca em Largura: Exemplo

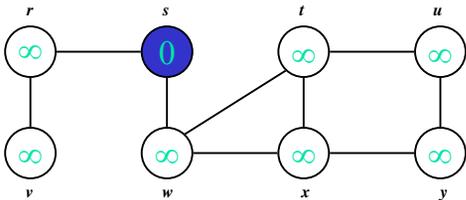


$Q: w \quad r$

22/04/2005

19

Busca em Largura: Exemplo

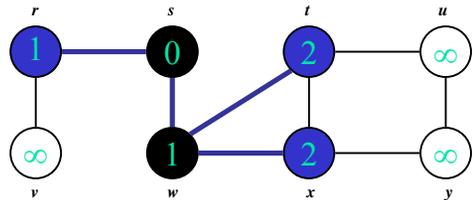


$Q: s$

22/04/2005

18

Busca em Largura: Exemplo

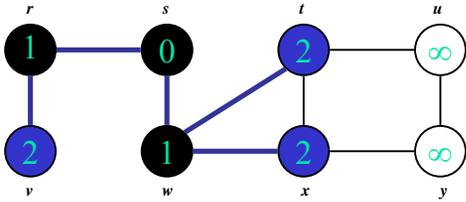


$Q: r \quad t \quad x$

22/04/2005

20

Busca em Largura: Exemplo

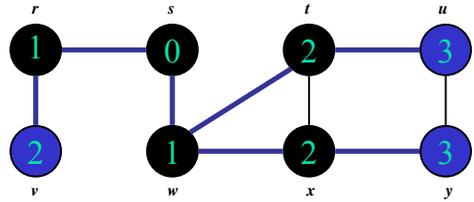


$Q: [t, x, v]$

22/04/2005

21

Busca em Largura: Exemplo

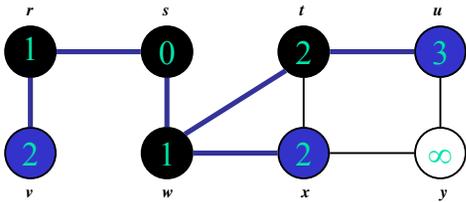


$Q: [v, u, y]$

22/04/2005

23

Busca em Largura: Exemplo

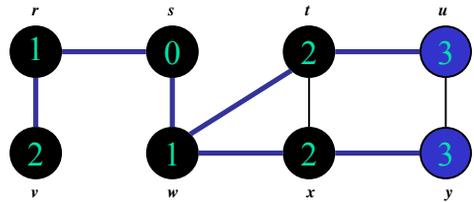


$Q: [x, v, u]$

22/04/2005

22

Busca em Largura: Exemplo

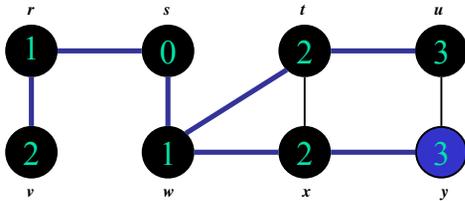


$Q: [u, y]$

22/04/2005

24

Busca em Largura: Exemplo



22/04/2005

25

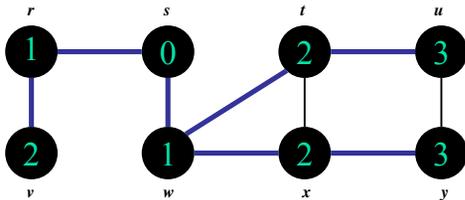
Tempo de Execução da BFS

- Dado um grafo $G = (V, E)$
 - Inicialização do algoritmo utiliza tempo $O(V)$
 - Vértices são enfileirados se possuem cor branca
 - Assumindo que enfileirar e desenfileirar toma tempo $O(1)$ o custo total desta operação é $O(V)$
 - A lista de adjacências de um vértice só é varrida quando um vértice é desenfileirado
 - A soma dos comprimentos de todas as listas é $\Theta(E)$. Conseqüentemente, é necessário um tempo de $O(E)$ para varrer todas as listas.
- **Tempo total de execução é $O(V+E)$** (linear no tamanho da representação de G)

22/04/2005

27

Busca em Largura: Exemplo



22/04/2005

26

Correção da Busca em Largura

- Veja no Livro.

22/04/2005

28

Propriedades da BFS

- Dado um grafo $G = (V, E)$, BFS **descobre todos os vértices que são atingíveis a partir de um vértice fonte s**
- O algoritmo computa:
 - a **menor distância** a todos os vértices atingíveis
 - a **árvore de largura** que contém esses vértices
- Para qualquer vértice v que pode ser atingido a partir de s , o caminho na árvore em largura de s a v , corresponde ao **menor caminho** de s a v em G

22/04/2005

29

Busca em Profundidade (1)

- Uma **busca em profundidade (DFS)** em um grafo não direcionado G é como buscar uma saída num labirinto com uma **corda** e com uma **lata de tinta**
 - Começamos no vértice s , amarrando a ponta de nossa corda (em s) e pintando s "visitado (descoberto)". Depois rotulamos s como nosso vértice corrente chamado u
 - Depois, caminhamos através de uma aresta arbitrária (u, v) .
 - Se a aresta (u, v) nos leva a um vértice já visitado v retornamos a u
 - Se o vértice v não foi visitado, desenrolamos nossa corda, movemos para v , pintamos v "visitado", e atribuímos v como sendo nosso vértice corrente, e repetimos os passos anteriores

22/04/2005

31

Árvore de Busca em Largura

- Subgrafo Predecessor de G

$$G_\pi = (V_\pi, E_\pi)$$

$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

- G_π é uma árvore de busca em largura
 - V_π consiste dos vértices atingíveis a partir de s , e
 - Para todo $v \in V_\pi$, existe um único **caminho simples** de s a v em G_π que também o menor caminho de s a v em G
- As arestas em G_π são denominadas **arestas da árvore**

22/04/2005

30

Busca em Profundidade (2)

- Eventualmente, atingimos um ponto onde **todas arestas incidentes em u levam a vértices visitados**
- Então a busca anda para trás (**backtrack**) desenrolando nossa corda a um vértice previamente visitado v . Então v torna-se nosso vértice corrente e repetimos os passos anteriores
- Então, se todas as arestas incidentes em v levam a vértices visitados, fazemos um backtrack como foi feito anteriormente. **Continuamos o backtrack através do caminho que já foi percorrido**, procurando e explorando arestas ainda não exploradas, e repetindo o procedimento

22/04/2005

32

Algoritmo DFS (1)

- Inicialize – pinte todos vértices com branco
- Visite cada (todos) vértice branco usando DFS-Visit
- Cada chamada para DFS-Visit(u) enraiza uma nova árvore de uma floresta de profundidade em um vértice u
- Um vértice é **branco** se ele não foi descoberto
- Um vértice é **azul** se ele foi descoberto mas algum de seus vértices adjacente ainda não foi descoberto
- Um vértice é **preto** após todos os seus vértices adjacentes serem descobertos.

22/04/2005

33

Algoritmo DFS (3)

- Quando DFS termina, todo vértice u recebe
 - Um tempo de descoberta $d[u]$, e um tempo de término $f[u]$
- Tempo de Execução
 - Os laços 1-3 e 5-7 em DFS utilizam tempo $\Theta(V)$ cada, excluindo o tempo para executar DFS-Visit
 - DFS-Visit é chamado exatamente uma vez para cada vértice
 - somente é chamada em vértices brancos, e
 - pinta o vértice de cinza imediatamente
 - Durante a execução do DFS-Visit o laço 4-7 é executado $|Adj[v]|$ e pelo fato que $\sum_{v \in V} |Adj[v]| = \Theta(E)$
 - O tempo de execução do DFS é $\Theta(V+E)$

22/04/2005

35

Algoritmo DFS (2)

```
DFS(G)
01 for each vertex u in V[G]
02   color[u] ← white
03   π[u] ← NIL
04 time ← 0
05 for each vertex u in V[G]
06   if color[u] = white then
07     DFS-VISIT(u)
DFS-VISIT(u)
01 color[u] ← gray
02 time ← time+1
03 d[u] ← time
04 for each v in Adj[u] do
05   if color[v] = white then
06     π[v] ← u
07     DFS-VISIT(v)
08 color[u] ← black
09 f[u] ← time
10 time ← time + 1
```

Inicialize todos vértices

Visite todos filhos recursivamente

22/04/2005

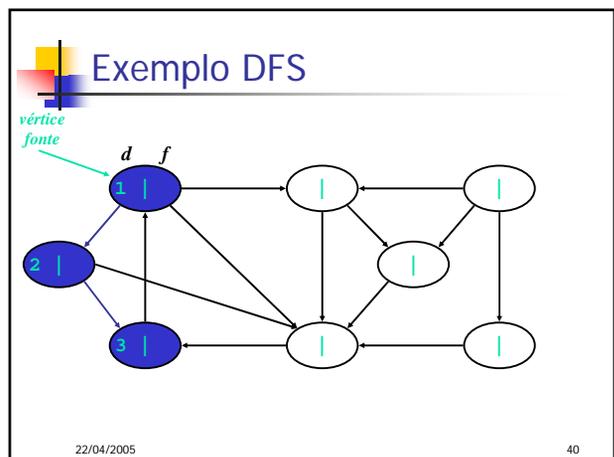
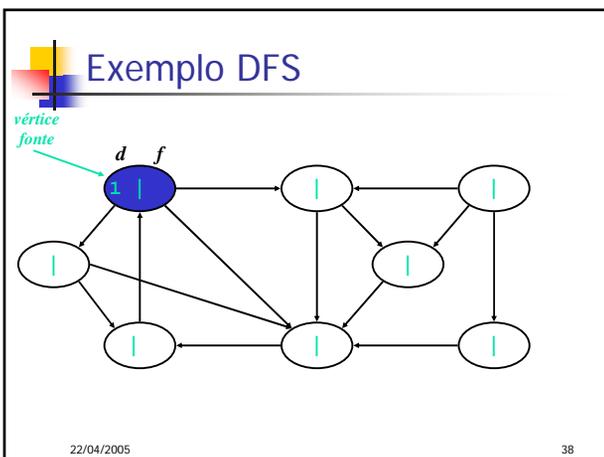
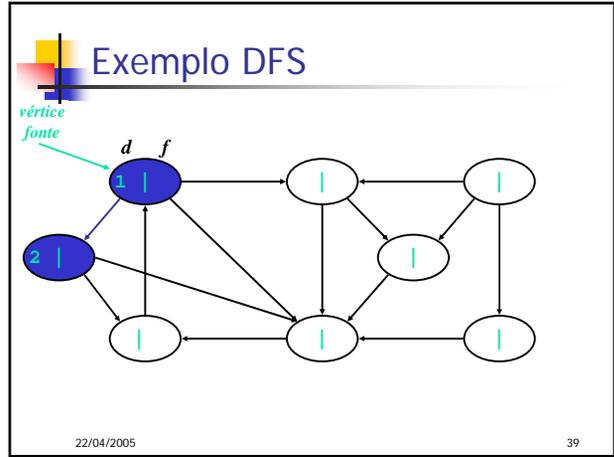
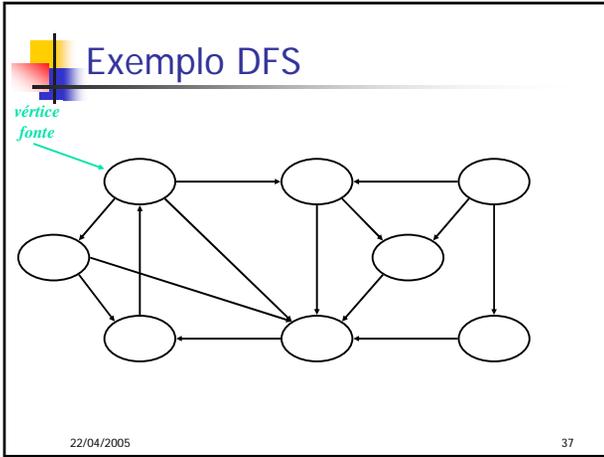
34

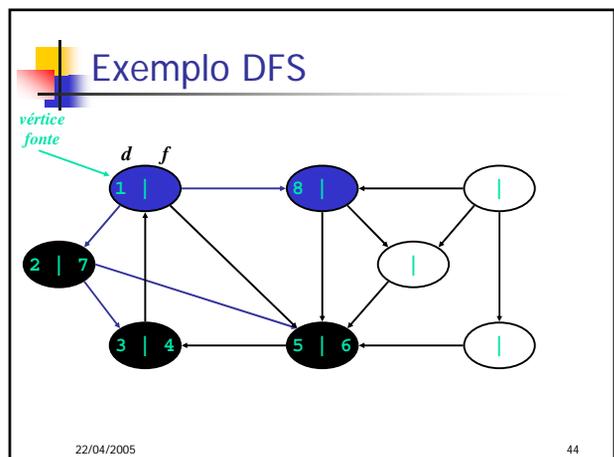
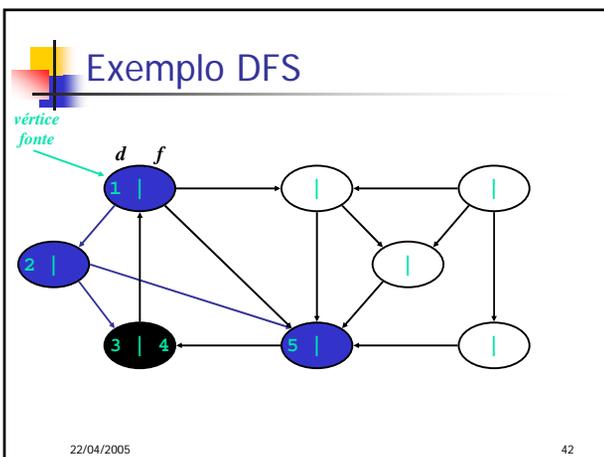
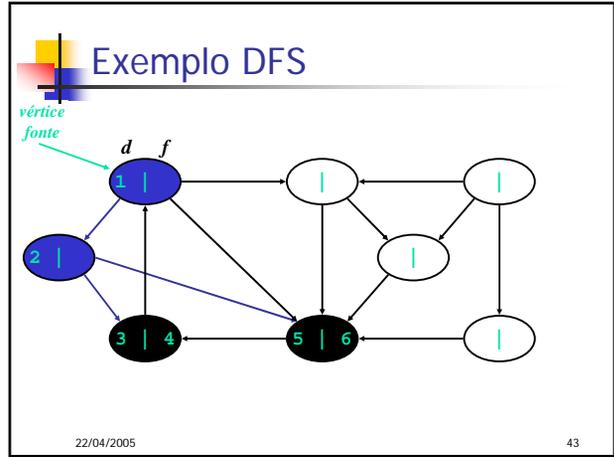
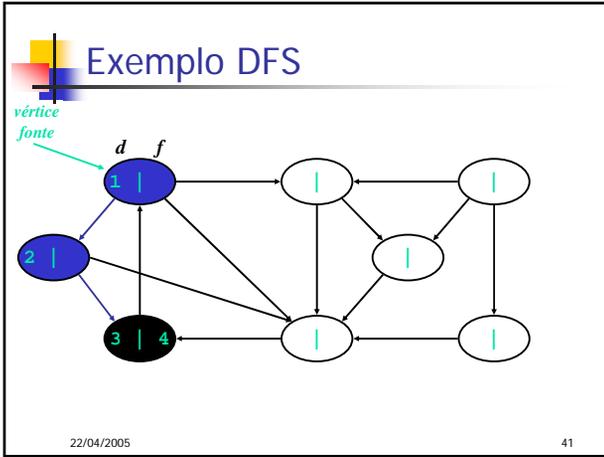
Subgrafo Predecessor

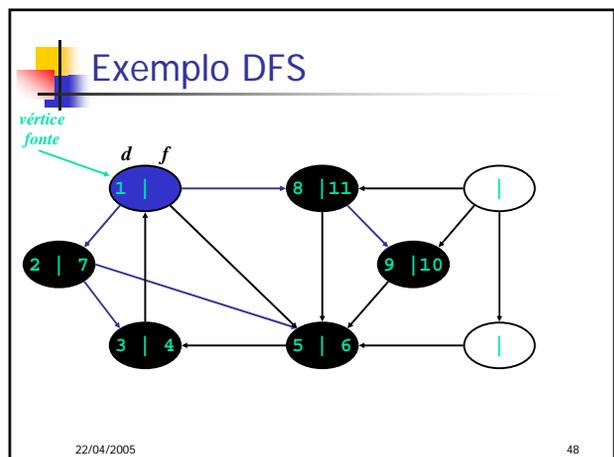
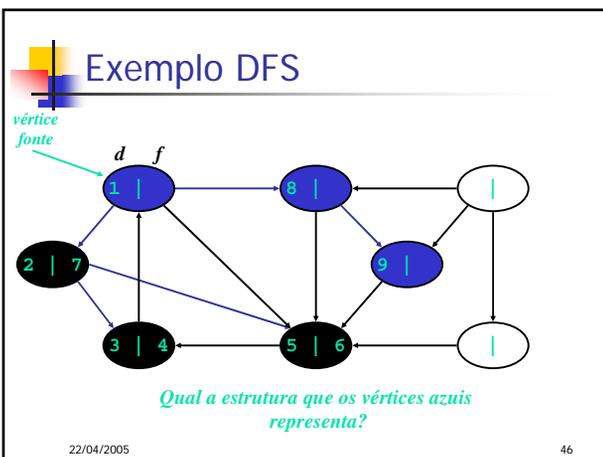
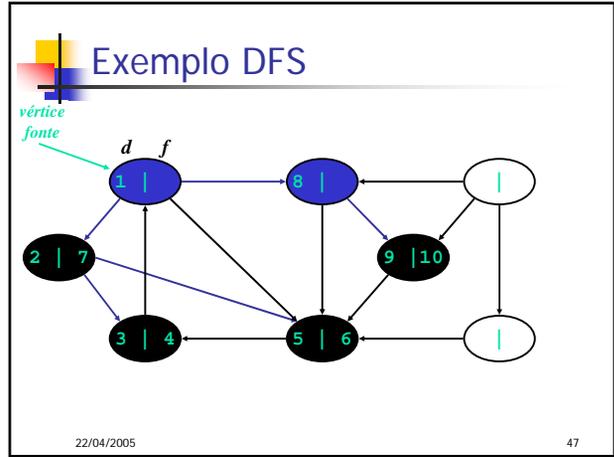
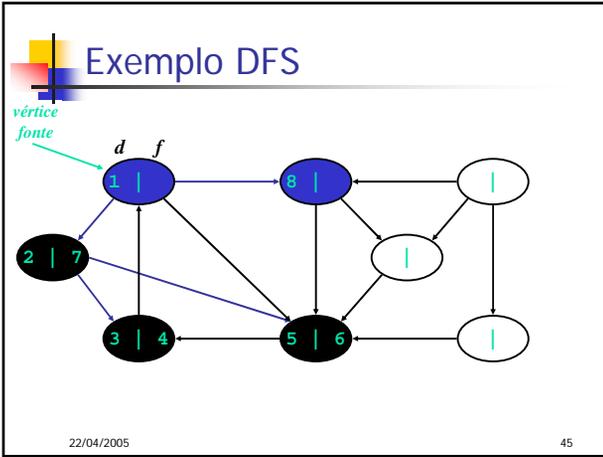
- Um pouco diferente da definição de BFS
$$G_\pi = (V, E_\pi)$$
$$E_\pi = \{(\pi[v], v) \in E : v \in V \text{ and } \pi[v] \neq \text{NIL}\}$$
- O subgrafo PD subgraph de uma busca em profundidade forma uma floresta de profundidade composta de diversas árvores de profundidade
- As arestas em G_π são chamadas arestas de árvore

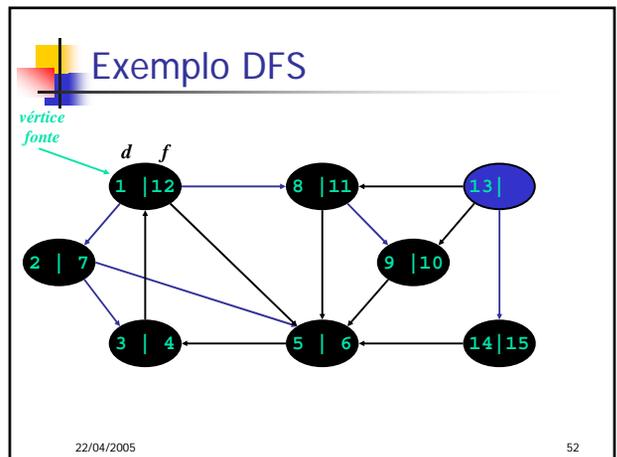
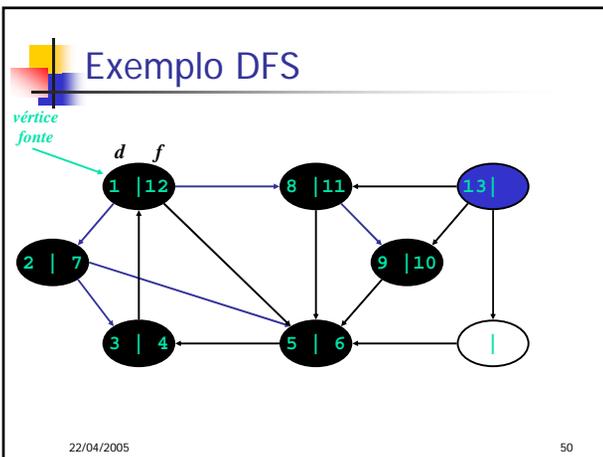
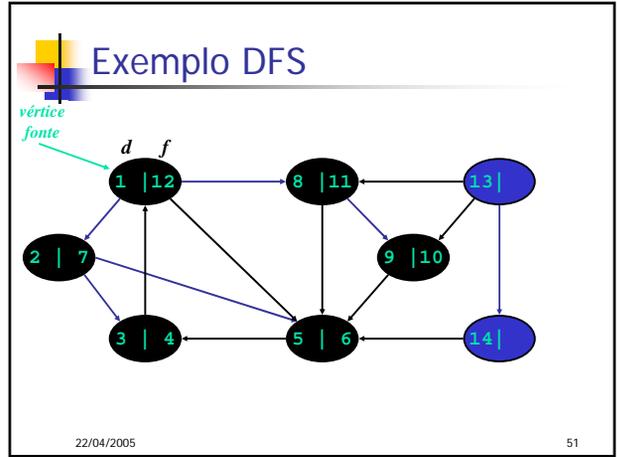
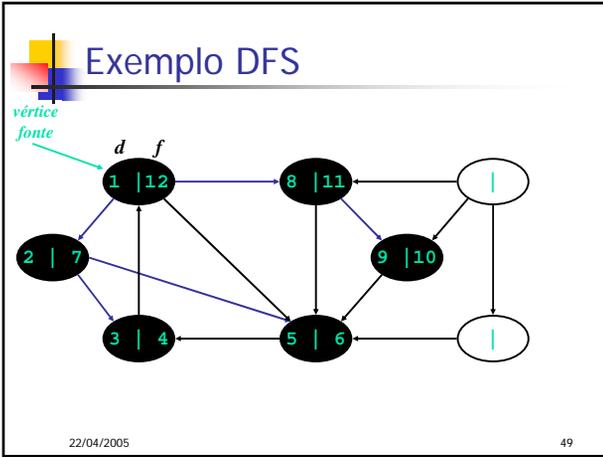
22/04/2005

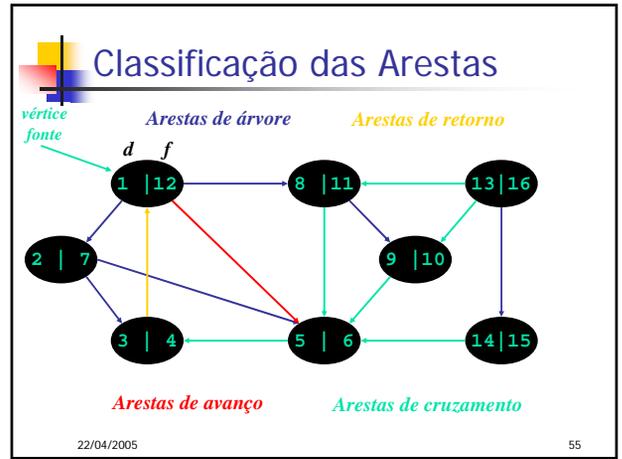
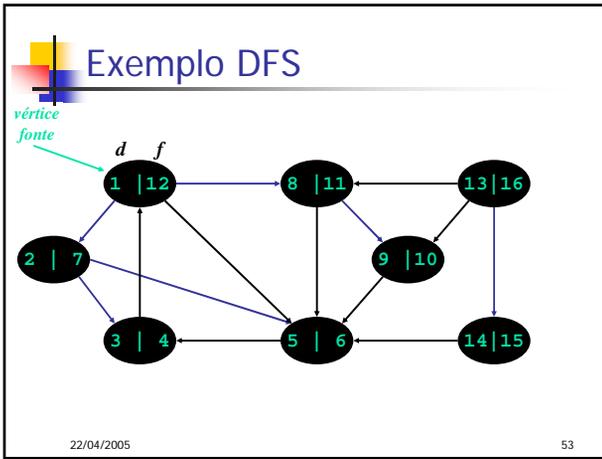
36







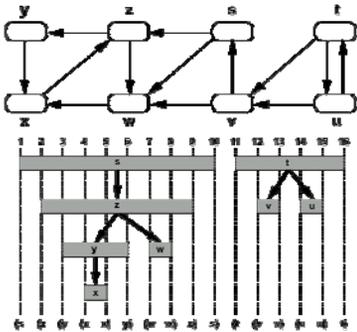




- ## Classificação das Arestas
- DFS introduz uma importante distinção entre as arestas do grafo original:
 - Aresta de árvore:** encontra novo vértice (branco)
 - Aresta de retorno:** de descendente para ancestral
 - Aresta de avanço:** de ancestral para descendente
 - Aresta de cruzamento:** entre uma árvore ou subárvores
 - De um vértice cinza para um vértice preto
- 22/04/2005 54

- ## Aplicação
- Tempos de descoberta e término tem um estrutura de parênteses.
 - represente descoberta de u com parênteses esquerdo "(u"
 - represente término de u com parênteses direito "u)"
 - Histórico das descobertas e terminos exibem uma expressão bem construída (parênteses estão corretamente aninhados)
 - Intuição para demonstração: quaisquer dois intervalos ou são disjuntos ou um contém o outro
 - Intervalos que se sobreponham significaria finalizando ancestral antes do descendente ou começando descendente sem começar o ancestral
- 22/04/2005 56

Aplicação

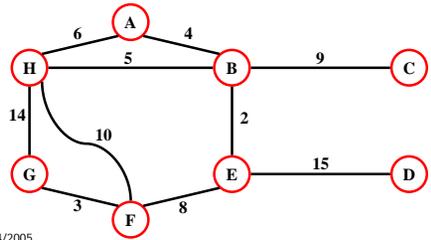


22/04/2005

57

Árvore Geradora Mínima

- Quais arestas formam a árvore geradora mínima (MST) do grafo abaixo?



22/04/2005

59

Árvore Geradora Mínima

- Uma **árvore geradora** de $G = (V, E)$ e uma função $W: E \rightarrow R$ é um subgrafo que
 - é uma árvore
 - contém todos vértices de G
- Uma **árvore geradora mínima** (MST) é uma árvore geradora que minimiza

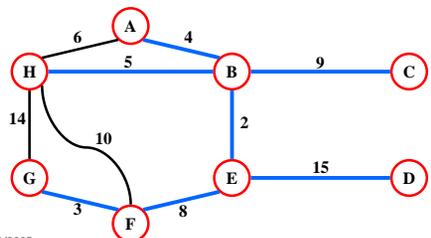
$$w(T) = \sum_{(u,v) \in T} w(u,v)$$
- Ou seja, encontrar uma árvore geradora usando arestas que minimizem o peso total

22/04/2005

58

Árvore Geradora Mínima

- Resposta:



22/04/2005

60

Árvore Geradora Mínima

- Algumas propriedades de uma MST:
 - Tem $|V| - 1$ arestas.
 - Não tem ciclos.
 - Pode não ser única.
- Construindo uma solução
 - Vamos construir um conjunto A de arestas.
 - Inicialmente, A não tem arestas.
 - A medida que adicionamos arestas em A , mantemos um invariante do laço:
 - Invariante do Laço: A é um subconjunto de alguma MST

22/04/2005

61

Encontrando uma Aresta Segura

- Como encontramos *arestas seguras*?
- Vamos analisar um exemplo. Aresta (c,f) tem o menor peso entre todas as arestas do grafo. Ela é segura para $A = \emptyset$?
- **Intuitivamente:** Seja $S \subset V$ qualquer conjunto de vértices que inclui c mas não inclui f (tal que f está em $V - S$). Em qualquer MST, deve haver uma aresta (pelo menos uma) que conecta S com $V - S$. Porque não escolher a aresta com peso mínimo?
- Neste caso, podemos escolher (c,f) .

22/04/2005

63

Algoritmo MST Genérico

- Adicione somente arestas que mantenham o invariante. Se A é um subconjunto de alguma MST, uma aresta (u,v) é uma para A se e somente se $A \cup \{(u,v)\}$ é também um subconjunto de alguma MST. **Aresta Segura** é uma aresta que não destrói a propriedade de A . Desta forma só adicionamos arestas seguras.

```
Generic-MST(G, w)
1 A ← ∅ // Contém arestas que pertencem a uma MST
2 while A não forma uma árvore geradora do
3   Encontre uma aresta (u,v) segura para A
4   A ← A ∪ {(u,v)}
5 return A
```

22/04/2005

62

Árvore Geradora Mínima

- Um *corte* $(S, V - S)$ de um grafo não direcionado $G=(V, E)$ é uma partição de V .
- Uma aresta *atravessa* o corte $(S, V - S)$ se um de seus pontos finais está em S e o outro em $V - S$.
- Um corte *respeita* um conjunto de arestas A se nenhuma aresta de A atravessa o corte.
- Uma *aresta leve* é uma aresta que atravessa um corte se seu peso é mínimo

22/04/2005

64

Árvore Geradora Mínima

- **Teorema:** Seja A um subconjunto de alguma MST, $(S, V-S)$ um corte que respeita A , e (u, v) uma aresta leve cruzando $(S, V-S)$. Então (u, v) é segura para A .
- **Demonstração:**
 - Seja T uma MST que inclui A .
 - Se T contém (u, v) , OK.
 - Vamos agora assumir que T não contém (u, v) . Vamos construir uma MST T' diferente que inclui $A \cup \{(u, v)\}$.
 - Uma árvore tem um caminho único entre cada par de vértice. Visto que T é uma MST, ela contém um caminho único p entre u e v . O caminho p deve cruzar o corte $(S, V-S)$ pelo menos uma vez. Seja (x, y) uma aresta de p que cruza o corte. Como a aresta (u, v) é uma aresta leve, temos que $w(u, v) \leq w(x, y)$.

22/04/2005

65

Árvore Geradora Mínima

Demonstração: (cont.)

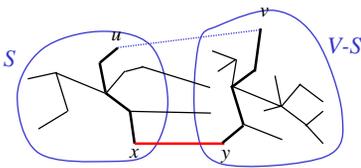
- Para formar T' a partir de T :
 - Remova (x, y) . Divide T em duas componentes.
 - Adicione (u, v) . Reconecta T .
- Desta forma $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
- T' é uma árvore geradora.
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
- visto que $w(u, v) \leq w(x, y)$. Visto que T' é uma árvore geradora, $w(T') \leq w(T)$, e T é uma MST, então T' deve ser uma MST.
- Temos que mostrar que (u, v) é segura para A :
 - $A \subseteq T$ e $(x, y) \notin A \Rightarrow A \subseteq T'$.
 - $A \cup \{(u, v)\} \subseteq T'$.
 - Visto que T' é MST, (u, v) é segura para A .

22/04/2005

67

Árvore Geradora Mínima

- **Demonstração:** (cont.)
 - Exceto pela aresta pontilhada (u, v) todas as arestas mostradas estão em T . A é algum subconjunto de arestas de T , mas não pode conter nenhuma aresta que atravesse o corte $(S, V-S)$, visto que este corte respeita A . As arestas sombreadas formam o caminho p .
 - Visto que o corte respeita A , a aresta (x, y) não está em A .



22/04/2005

66

Árvore Geradora Mínima

- No algoritmo genérico MST:
 - A é uma floresta contendo componentes conexos. Inicialmente cada componente é um único vértice.
 - Cada aresta segura une dois desses componentes em um. Cada componente é uma árvore.
 - Visto que uma MST tem exatamente $|V| - 1$ arestas, o laço for é repetido $|V| - 1$ vezes. Equivalentemente, após adicionar $|V| - 1$ arestas, temos uma única componente.
- **Corolário:** Se $C = (V_C, E_C)$ é um componente conexo numa floresta $G_A = (V, A)$ e (u, v) é uma aresta leve conectando C a algum outro componente em G_A , então (u, v) é segura para A .
- Isto naturalmente leva ao algoritmo de Kruskal para resolver o problema da árvore geradora mínima.

22/04/2005

68

Algoritmo de Kruskal

- $G = (V, E)$ é um grafo não direcionado, conexo, com peso nas arestas. $w: E \rightarrow R$.
- Inicie com cada vértice sendo seu próprio componente.
- Repetidamente junte dois componentes num único adicionando a aresta mais leve que conecta-os (i.e., a aresta mais leve atravessando o corte entre os componentes).
- Escolha as arestas na ordem crescente de peso.
- O algoritmo mantém A – uma **floresta de árvores**. Uma aresta é aceita se ela conecta vértices de árvores distintas
- Utilize uma estrutura de dados de conjuntos disjuntos para determinar se uma aresta conecta vértices em componentes diferentes.

22/04/2005

69

Algoritmo de Kruskal

Kruskal(V, E, w)

```
1 A = ∅;  
2 for cada v ∈ V do  
3   Make-Set(v);  
4 ordene E em ordem crescente de peso w  
5 for cada (u,v) ∈ E (em ordem) do  
6   if FindSet(u) ≠ FindSet(v)  
7     then A ← A ∪ {{u,v}};  
8     Union(u,v);  
9 devolva A
```

22/04/2005

71

Conjuntos Disjuntos

- Make-Set(x): faça um novo conjunto $S_i = \{x\}$, e adicione S_i a S .
- Union(x, y): se $x \in S_x, y \in S_y$, então $S \leftarrow S - S_x - S_y \cup \{S_x \cup S_y\}$.
 - O representante do novo conjunto é qualquer membro de $S_x \cup S_y$.
 - Destrua S_x e S_y (os conjuntos devem ser disjuntos).
- Find-Set(x): retorne o representante do conjunto contendo x .

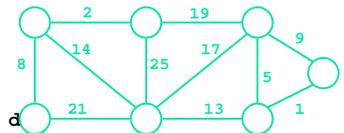
22/04/2005

70

Algoritmo de Kruskal

Kruskal(V, E, w)

```
1 A = ∅;  
2 for cada v ∈ V do  
3   Make-Set(v);  
4 ordene E em ordem crescente de peso w  
5 for cada (u,v) ∈ E (em ordem) do  
6   if FindSet(u) ≠ FindSet(v)  
7     then A ← A ∪ {{u,v}};  
8     Union(u,v);  
9 devolva A
```



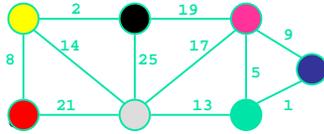
22/04/2005

72

Algoritmo de Kruskal

```

Kruskal(V,E,w)
1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E por ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



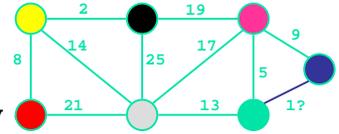
22/04/2005

73

Algoritmo de Kruskal

```

Kruskal(V,E,w)
1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



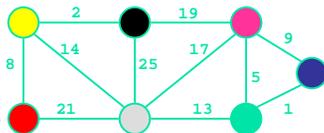
22/04/2005

75

Algoritmo de Kruskal

```

Kruskal(V,E,w)
1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



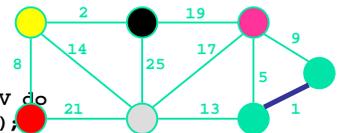
22/04/2005

74

Algoritmo de Kruskal

```

Kruskal(V,E,w)
1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

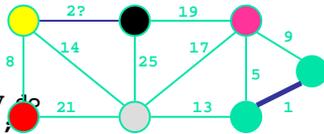
76

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

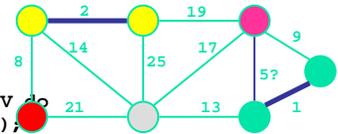
77

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

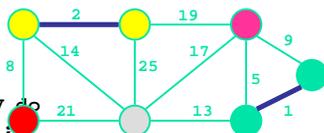
79

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for each v ∈ V do
3   Make-Set(v);
4 sort E by nondecreasing order by weight w
5 for each (u,v) ∈ E (in sorted order) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 return A
    
```



22/04/2005

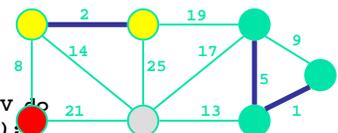
78

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

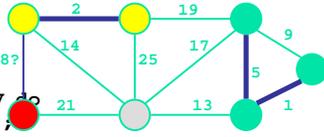
80

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

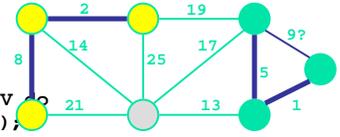
81

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

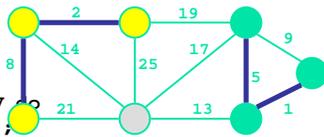
83

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

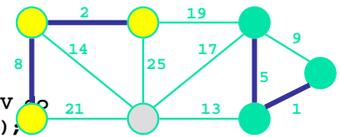
82

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

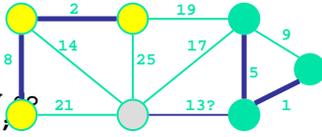
84

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

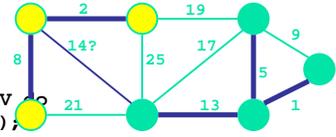
85

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

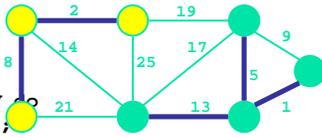
87

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

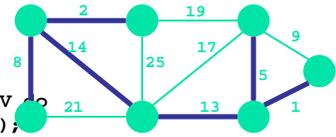
86

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V do
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

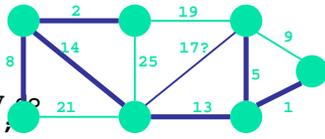
88

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

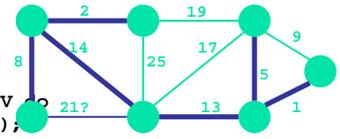
89

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

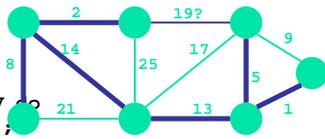
91

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

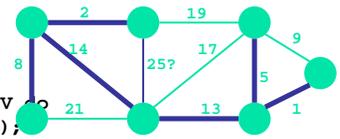
90

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

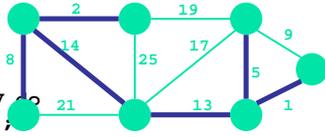
92

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```



22/04/2005

93

Correção do Algoritmo de Kruskal

- O algoritmo de Kruskal produz uma MST para T :
 - Assuma que o algoritmo não está correto: o resultado não é uma MST
 - Então o algoritmo adiciona uma aresta errada em algum ponto
 - Se ele adiciona uma aresta errada, deve existir uma aresta com peso menor
 - Mas o algoritmo escolhe a aresta de menor peso em cada passo. Contradição

22/04/2005

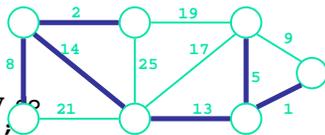
95

Algoritmo de Kruskal

Kruskal(V, E, w)

```

1 A = ∅;
2 for cada v ∈ V
3   Make-Set(v);
4 Ordene E em ordem crescente de peso w
5 for cada (u,v) ∈ E (em ordem) do
6   if FindSet(u) ≠ FindSet(v)
7     then A ← A ∪ {{u,v}};
8     Union(u,v);
9 devolva A
    
```

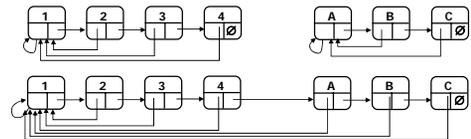


22/04/2005

94

Conjuntos Disjuntos Como Listas

- Cada conjunto – uma lista de elementos identificada pelo primeiro elemento, todos elementos na lista apontam para o primeiro elemento
- Make-Set(v): $O(1)$
- Find-Set(v): $O(1)$
- Union(u, v) – adiciona a lista menor na maior
- Union(u, v): $O(\min\{|C(u)|, |C(v)|\})$. Pode ser menor ($\alpha(n)$).



22/04/2005

96

Tempo de Execução do Alg. Kruskal

- Inicialização: tempo $O(V)$
- $O(V)$ chamadas para Make-Set
- Ordenação das arestas $\Theta(E \lg E) = \Theta(E \lg V)$
- $O(E)$ chamadas para Find-Set e Union
- Custo de cada Union
 - Seja $t(v)$ – o número de vezes que v é movido para um novo cluster
 - Cada vez que um vértice é movido para um novo cluster o tamanho do cluster contendo o vértice pelo menos dobra: $t(v) \leq \lg V$
- Tempo total gasto na Union: $\sum_{v \in V} t(v) \leq V \lg V$
- Tempo total: $O(E \lg V)$

Próxima Aula

- Árvore Geradora Mínima
 - Algoritmo de Prim
- Caminhos mais Curtos
 - Algoritmo de Bellman-Ford
 - Algoritmo de Dijkstra