

Teoria de Grafos

Notas de Aula

Ricardo Dorneles

CCET-UCS

21 de junho de 2023

Índice (1)

- Representação de Grafos
- Ciclo Euleriano
- Algoritmos de Busca em Grafos
 - Busca em Amplitude (BFS)
 - Busca em Profundidade (DFS)
- Árvores
 - Algoritmo de Prim para Árvore geradora mínima
 - Algoritmo de Kruskal para Árvore geradora mínima
- Algoritmos para Caminho Mínimo
 - Algoritmo de Dijkstra para menor caminho em grafos valorados
 - Shortest Path Faster Algorithm (SPFA)
 - O Uniform-Cost Search (UCS)
 - Algoritmo de Floyd-Warshall para menor caminho entre todos os pares
 - Distância entre vértices em árvores por LCA (Lowest Common Ancestor)
 - Enraizamento de uma Árvore
 - Quantidade de caminhos entre pares de vértices

Índice (2)

- Planaridade
- Coloração
- Isomorfismo
 - Isomorfismo de Árvores
- Conectividade em Grafos - Componentes Conexas e Biconexas
- Problemas de circulação de Fluxo em Redes
 - Problemas de fluxo máximo - custo mínimo
 - Problemas Produtor x Consumidor
 - Associação Máxima em Grafos Bipartidos
 - Algoritmo de Hopcroft-Karp para Associação Máxima em Grafos Bipartidos
 - Associação Máxima em Grafos não-bipartidos
- Número cromático por Alteração Estrutural
- Ciclo Hamiltoniano e o Problema do Caixeiro Viajante
 - Caixeiro Viajante por Programação Dinâmica
- Problemas de Caminho Crítico - PERT/CPM
- Ordenação Topológica

Índice (3)

- Componentes fortemente conexas
 - Algoritmo de Kosaraju para identificação de componentes fortemente conexas
 - Algoritmo de Tarjan para identificação de componentes fortemente conexas
- Cobertura de Arestas
- Conjunto de vértices independentes
- Cobertura de Vértices
- Particionamento de Grafos
- O problema do corte mínimo (min-cut)
- Exercícios da Área 1
- Exercícios da Área 2

- Questões do ENADE
- Questões do POSCOMP
- Últimas Provas - Área 1
- Últimas Provas - Área 2
- Heaps, priority queues e Heapsort
- Disjoint-sets(Union-finds)
- A STL (Standard Templates Library) do C++
- Recursos Online
- Problemas NP-Completos em Grafos

- Grafos são modelos matemáticos utilizados para resolver problemas práticos do dia-a-dia.
- Podem ser utilizados para modelar:
 - Circuitos elétricos
 - Redes de distribuição: energia, bens, informação
 - Relações de parentesco entre pessoas
 - Redes sociais
 - Ligações entre cidades (estradas/vôos):
 - Waze, Google Maps
 - Redes de computadores
 - Páginas Web
 - Simulação de materiais e modelos físicos
 - e muitos outros...

Introdução

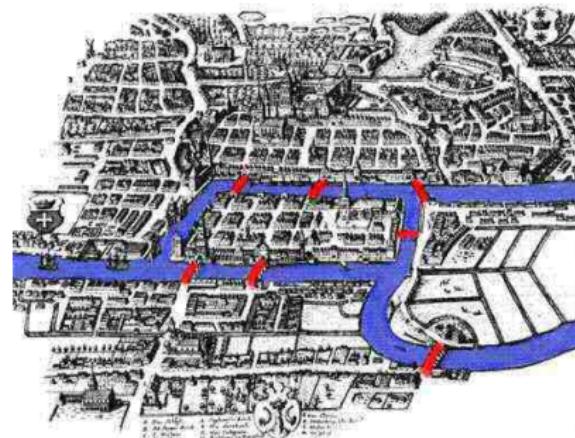
- Um dos problemas que deram origem à Teoria dos Grafos foi o chamado problema de Königsberg, resolvido por Euler em 1736.



Leonard Euler

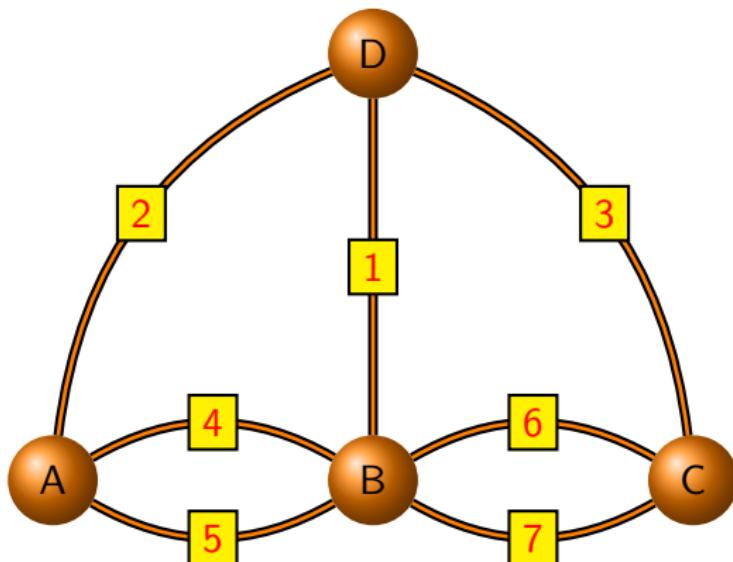
Introdução

- O problema consistia em sair de uma das margens do rio Pregel, passar exatamente uma vez em cada uma das 7 pontes e voltar ao ponto de partida.



Königsberg

- O problema pode ser melhor visualizado se forem extraídos do problema original somente os elementos essenciais, resultando no modelo abaixo:



Grafo representando o problema da ponte de Königsberg ▶

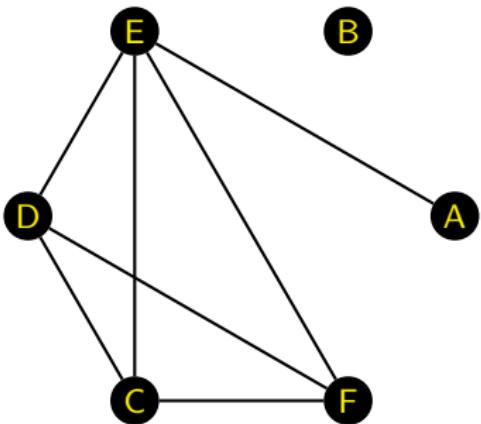
- Caminho Euleriano : Passar por todas as arestas sem repetir nenhuma. Aplicação: Um caminhão de lixo deve passar por todas as ruas de uma cidade. Encontrar um caminho que percorra todas as ruas minimizando o número de ruas repetidas.
- Problema das 4 cores (Francis Guthrie, 1852) - Número Cromático
 - Dado um mapa arbitrário, plano, o problema consiste em colorir cada país com uma cor, de maneira que não existem dois países fronteiros com a mesma cor (para este problema foi formulada uma solução em 1977 por Appel e Haken).

Exemplo de problema de coloração

- O diretório dos estudantes da Universidade Estadual da Utopia está sediando um simpósio em problemas urbanos com seis palestrantes no dia de abertura. Cada palestrante planeja utilizar uma hora. Se os palestrantes apresentarem em seqüência, as atividades do dia serão um pouco compridas. Por outro lado, desde que diversos palestrantes são especialmente populares, é indesejável que eles palestrem na mesma hora porque isto requereria que uma pessoa escolhesse qual dos dois ou mais palestrantes gostaria de ouvir.

- Após alguma reflexão, é decidido que não deve haver mais do que quatro seções de palestras. A tabela a seguir, construída a base de uma pesquisa dos estudantes, indica quais palestrantes podem ou não podem discursar simultaneamente. Como serão distribuídos os palestrantes com no máximo 4 sessões? Podemos representar os palestrantes que não podem palestrar num mesmo horário através do grafo a seguir:

	A	B	C	D	E	F
A					X	
B						
C				X	X	X
D			X		X	X
E	X		X	X		X
F			X	X	X	



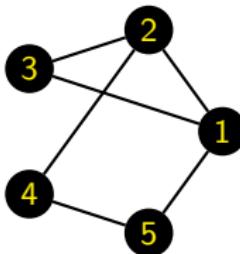
Incompatibilidade de palestrantes e grafo equivalente

Isomorfismo

- Dois grafos são isomorfos se um pode ser transformado no outro apenas renomeando os vértices. Pode-se representar moléculas como grafos e verificar, por exemplo, se uma dada molécula já consta de uma base de dados.

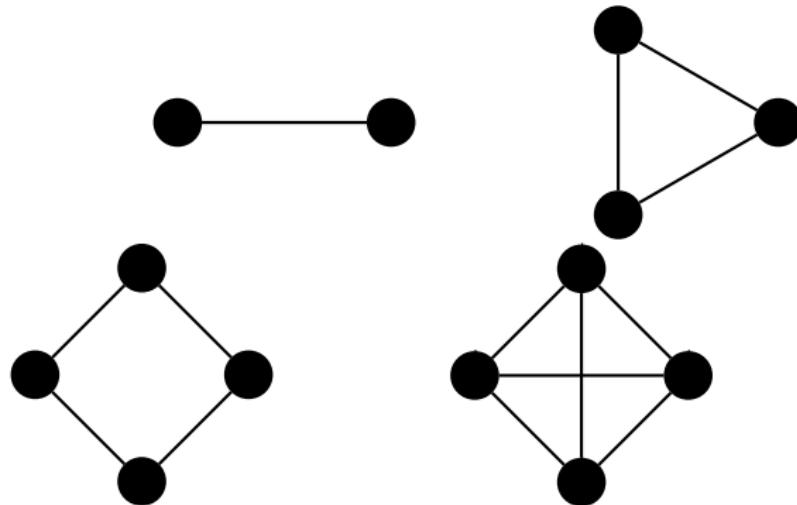
- Um grafo **grafo** é uma estrutura que permite a representação de relações arbitrárias entre objetos.
- **Def:** Um grafo é um par $G=(V,E)$, formado por um conjunto V de vértices e um conjunto E de arestas (ou elos, ou arcos).
 - 1 Os vértices representam os objetos;
 - 2 As arestas representam relações entre os vértices;
 - 3 Cada aresta define uma relação entre dois vér.

- Ex: $G(V, E)$
 - $V = \{1, 2, 3, 4, 5\}$
 - $E = \{(1, 2), (2, 3), (1, 3), (2, 4), (1, 5), (4, 5)\}$



- **Def:** Dois vértices v_1, v_2 são **adjacentes** se forem ligados por uma aresta (Ex. os vértices 1 e 3 são adjacentes).
- **Def:** Duas arestas são **adjacentes** se possuem um vértice em comum (Ex: (1,2) e (2,4)).
- **Def:** O número de vértices adjacentes a um vértice (ou número de arestas que pertencem a um vértice) é chamado de **grau** do vértice (Ex: $\text{grau}(1)=3$)
- A soma dos graus dos vértices de um grafo é sempre par (por que?)
- O número de vértices de grau ímpar em um grafo deve ser par (lema do 'aperto de mão' - handshaking lemma)

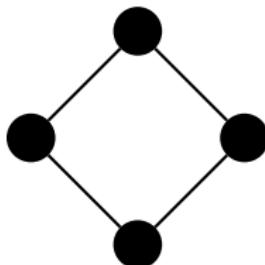
- **Def:** Um grafo é **regular** (de grau γ) quando todos os seus vértices possuem grau γ .



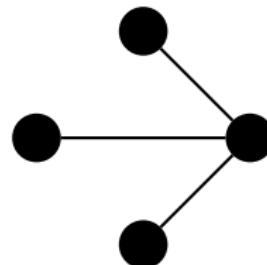
Grafos regulares

- **Def:** Um vértice de grau 0 é chamado de **isolado**.
- **Def:** Um **multigrafo** é um grafo que possui mais de uma aresta entre dois vértices.
- **Def:** Um **caminho** (ou trajeto) é uma seqüência de vértices v_1, \dots, v_k tal que v_i, v_{i+1} com $1 \leq i < k$ é uma aresta.
 - Ex: 1,3,2,4 é um caminho, 4,5,1,3,2,4 é um caminho
- **Def:** Um **caminho simples** é um caminho v_1, \dots, v_k onde $\forall i, j, i \neq j, 1 \leq i < j \leq k, v_i \neq v_j$, isto é, não existem vértices repetidos.
 - Ex: 4,5,1,2 e 1,3,2,4 são caminhos simples, 4,5,1,3,2,1 não é.

- **Def:** Um caminho cujo vértice inicial é o mesmo que o final é um **ciclo** (ou circuito).
 - Ex: 4,5,1,2,4
- **Def:** Um grafo que possui algum ciclo é chamado **grafo cíclico**. Se o grafo não possui ciclos então é chamado de **acíclico**.



Grafo Cíclico



Grafo Acíclico

- **Def:** Um **laço** é uma aresta que liga um vértice a ele mesmo.
- **Def:** Um **caminho hamiltoniano** é um caminho que contém todos os vértices do grafo, cada um exatamente uma vez.
 - Ex: 1,5,4,2,3
- **Def:** Um **caminho euleriano** é um caminho que contém todas as arestas do grafo, cada uma exatamente uma vez.
 - Ex: 1,5,4,2,1,3,2
- **Teorema :** Um grafo conexo G tem um caminho Euleriano entre dois vértices x_i e x_j **se e somente se** os únicos vértices de grau ímpar são x_i e x_j .
- **Def:** Um **ciclo hamiltoniano** é um caminho hamiltoniano que começa e termina no mesmo vértice.
 - Ex: 1,5,4,2,3,1
- **Def:** Um **ciclo euleriano** é um caminho euleriano que começa e termina no mesmo vértice.
 - Ex: 1,2,3,4,1

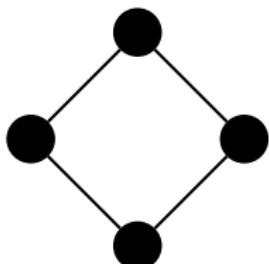
Exercícios fora de contexto

- Faça uma função que receba um grafo representado como uma lista de adjacências e retorne:
 - 1 Se o grafo é cíclico
 - 2 Se é acíclico
- Faça uma função que receba um grafo representado como uma lista de adjacências e retorne:
 - 1 Se o grafo contém um ciclo hamiltoniano
 - 2 Caso contrário

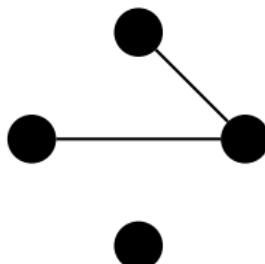
Def: Um grafo é **conexo** quando há pelo menos um vértice a partir do qual é possível atingir todos os demais.

Def: Um grafo **fortemente conexo** é um grafo no qual, a partir de qualquer vértice, se pode atingir todos os outros vértices do grafo.

Obs: Se um grafo não dirigido é conexo, então ele é fortemente conexo.



Grafo Conexo



Grafo Não-conexo

Ciclos Eulerianos

Teorema: Um grafo G conexo possui ciclo Euleriano se e somente se todo vértice v de G possui grau par. Temos que demonstrar que:

- 1 (necessidade) "Seja $G = (V, E)$ um grafo conexo. Se G possui ciclo Euleriano, então $\forall v \in V$, $\text{grau}(v)=2K$, $K = 1, 2, \dots$ ". (o grau de todos os vértices é par).
- 2 (suficiência) "Seja um grafo conexo. Se $\forall v \in V$, $\text{grau}(v) = 2K$, $K \in N$ (Naturais) - $\{0\}$, então possui ciclo Euleriano"

[Voltar para o índice](#)

- Prova da necessidade:

- Suponha que o ciclo Euleriano inicia e termina no vértice x_i e passa t vezes por x_i , entrando e saindo por arestas não utilizadas. Há um total de $2t + 2$ arestas adjacentes a x_i . Isto pode ser aplicado a todos os vértices do grafo, portanto todos os vértices têm grau par.

- Prova da suficiência:
 - Suponha que percorremos o grafo a partir de um vértice v_i qualquer, passando apenas por arestas que ainda não foram percorridas, até chegar em um vértice de onde não saem arestas ainda não percorridas. Este vértice é v_i (se não fosse, este vértice teria um número ímpar de arestas, o que contraria a hipótese). Duas situações surgem:
 - a) Todas as arestas de G foram percorridas e portanto temos um caminho Euleriano

- Prova da suficiência:
 - Suponha que percorremos o grafo a partir de um vértice v_i qualquer, passando apenas por arestas que ainda não foram percorridas, até chegar em um vértice de onde não saem arestas ainda não percorridas. Este vértice é v_j (se não fosse, este vértice teria um número ímpar de arestas, o que contraria a hipótese). Duas situações surgem:
 - a) Todas as arestas de G foram percorridas e portanto temos um caminho Euleriano
 - b) Algumas arestas de G não estão no caminho descrito acima.

- No caso b, chamamos de π_1 o ciclo formado e removemos suas arestas de G obtendo o novo grafo H (H pode não ser conexo).
- Como G era conexo, algum vértice de H tem um vértice v_i em comum com π_1 , e todos os vértices de H têm grau par.
- Partindo-se de v_i sobre H, chega-se novamente em v_i criando um ciclo sobre aquela componente de H que contém v_i .
- Este ciclo é acrescentado a π_1 , a partir de v_i , gerando o novo ciclo π_2 .
- Este procedimento prossegue até que não haja mais arestas em H. O nome desse algoritmo é algoritmo de Hierholzer, devido a Karl Hierholzer, matemático alemão que o demonstrou em 1871.

Ex: Encontre um ciclo Euleriano no grafo a seguir (inventar um grafo bem complicado).

Representação de Grafos

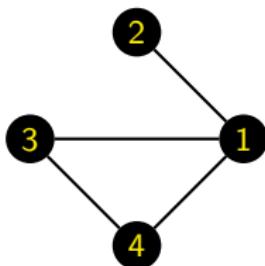
- Diversas estruturas de dados podem ser utilizadas para representar grafos. A estrutura mais apropriada vai depender principalmente de quais as operações que serão feitas com maior frequência, mas fatores como o tamanho dos grafos a serem tratados e a habilidade do programador também podem ser levadas em consideração.
- As principais estruturas utilizadas são:
 - Matrizes - adjacência ou incidência
 - Vetores - adjacência ou incidência
 - Listas de adjacências

- Matriz de adjacência: Cada posição $M[i][j]$ representa a adjacência entre o vértice i e o vértice j , isto é, se há uma aresta entre o vértice i e o vértice j .
- grafo com n vértices, $MA = n \times n$
- - Grafo não dirigido :

$$MA[i,j] = \begin{cases} 1 & \text{se } i \text{ e } j \text{ são adjacentes} \\ 0 & \text{caso contrário} \end{cases}$$

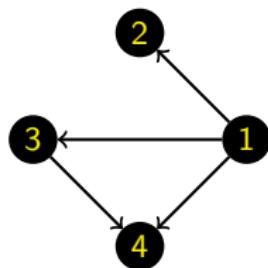
- - Grafo dirigido : $MA[i,j] = \begin{cases} 1 & \text{se } i \text{ é adjacente a } j \\ 0 & \text{caso contrário} \end{cases}$

Exemplo:



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

Figura: Grafo não dirigido



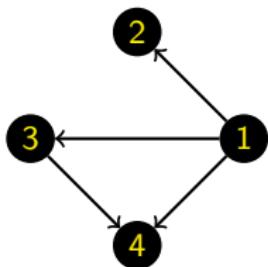
	1	2	3	4
1	0	1	1	1
2	0	0	0	0
3	0	0	0	1
4	0	0	0	0

Figura: Grafo dirigido

- Matriz de Incidência: Cada coluna da matriz guarda a informação de uma aresta do grafo.
- para uma matriz de **n** vértices e **m** arcos : $MI = n \times m$
 - vértices \Rightarrow linhas
 - arcos \Rightarrow colunas

$$MA[i,j] = \begin{cases} 1 & \text{se o arco } j \text{ é incidente do vértice } i \\ 0 & \text{se não há relação de incidência} \\ -1 & \text{se o arco } j \text{ é incidente ao vértice } i \end{cases}$$

Ex:



	(1,2)	(1,3)	(1,4)	(3,4)
1	1	1	1	0
2	-1	0	0	0
3	0	-1	0	1
4	0	0	-1	-1

Figura: Grafo dirigido e Matriz de Incidências

- Vantagens da representação por matriz:
 - Tempo de acesso a qualquer elemento é igual (se for utilizado array)
 - Fácil para manter trocas, inserções e remoções de arcos/arestas
- Desvantagens:
 - Quando há poucos arcos/arestas
 - Variação grande dos vértices

- Vetor de adjacências: utiliza dois vetores, guardando para cada vértice o número de cada vértice adjacente a ele
- para um grafo de n vértices e m arcos ou arestas:
 - Grafo dirigido : V_1 com n elementos, V_2 com m elementos
 - Grafo não dirigido : V_1 com n elementos, V_2 com $2m$ elementos
 - V_1 : cada posição $V_1[i]$ contém o número da posição em V_2 onde começa a relação de vértices adjacentes a i .
 - V_2 : contém os vértices adjacentes a i .

1	4	4	5
---	---	---	---

Tabela: V_1

2	3	4	4
---	---	---	---

Tabela: V_2

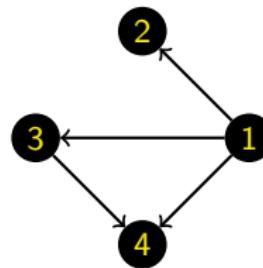
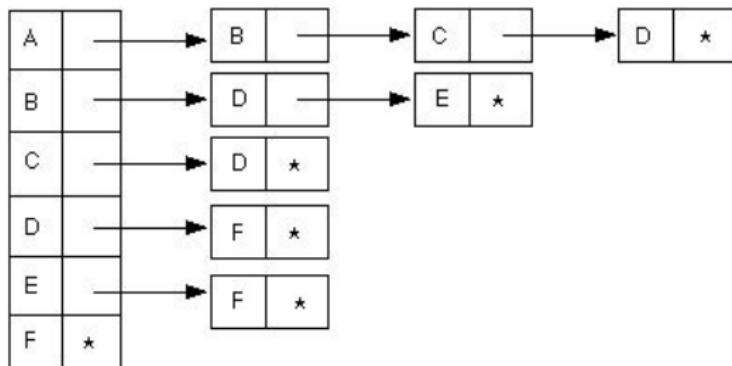
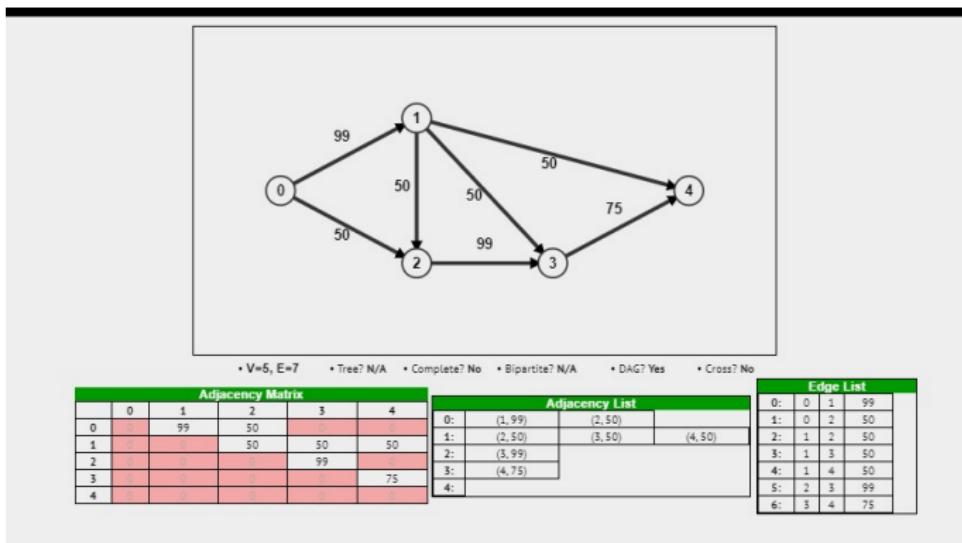


Tabela: Vetor de adjacências para grafo dirigido

- Listas de Adjacências: Mantem-se uma lista encadeada para cada vértice v_i , contendo os vértices adjacentes a v_i
- Qual o grafo representado pela figura abaixo?



- O site VisuAlgo, na seção Graph Structures () apresenta a representação em matriz de adjacências, lista de adjacências e lista de arestas para diversos grafos valorados e não valorados, dirigidos e não dirigidos.



- Exercício:

- Escreva uma função para receber como entrada um grafo e retornar 0 se o grafo não contiver um ciclo euleriano e 1 se ele contiver. Caso ele contenha um ciclo euleriano, a função deve retornar o ciclo euleriano (estrutura de dados?) iniciando pelo vértice 1.

Busca em Grafos

Algoritmo de Busca : Permite a exploração do grafo, é um processo sistemático para percorrer seus vértices e arestas (arcos). Quando o grafo é uma árvore podem aplicar-se os algoritmos já conhecidos de pré-ordem; e se for binária, in-ordem.

1 Pré ordem:

Visitar raiz Pré-ordem (esq-1) Pré-ordem (esq-2) ...
Pré-ordem (esq-n)

2 Pós-ordem:

Pós-ordem (esq-1) Pós-ordem (esq-2) ... Pós-ordem(esq-n)
Visitar raiz

3 In-ordem:

In-ordem (esq) Visitar raiz In-ordem (dir)

Estes encaminhamentos procuram descer na árvore tão profundamente quanto possível, da esquerda para a direita, sistematicamente.

Grafos: Problemas para percorrê-los:

- 1 Não há um referencial geral : esquerda, direita, nível, raiz, etc. Como percorrer sistematicamente?
- 2 De acordo com a seqüência de vértices que se percorre, pode-se voltar a um mesmo vértice e consequentemente passar por arestas já percorridas, pois pode haver ciclos. Como evitar repetições desnecessárias?

Soluções:

- 1 Escolhe-se um vértice inicial (arbitrário).
- 2 Utiliza-se como recurso a marcação dos vértices já percorridos. Se um vértice estiver marcado, ele não será percorrido novamente.

Algoritmo Básico de Busca em Grafos

- 1 Todos os vértices estão desmarcados
- 2 Escolhe-se um vértice v e marca-se como visitado.
- 3 A partir de v escolhe-se uma aresta (v, w) ainda não explorada e marca-se w .
- 4 O processo termina quando todas as arestas do grafo estiverem marcadas.

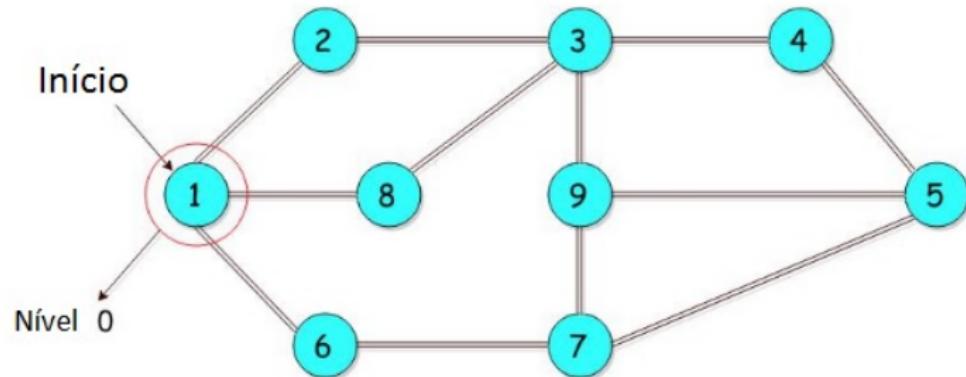
⇒ Um vértice pode ser alcançado muitas vezes, mas pode ser marcado somente uma vez.

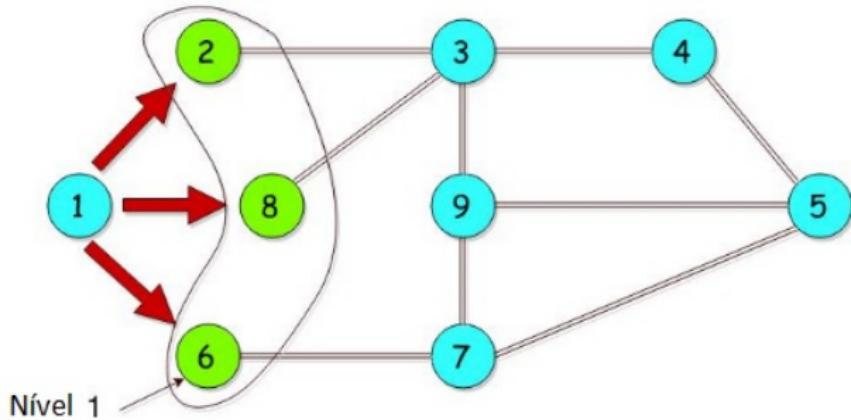
A escolha do vértice e aresta a serem visitados é arbitrária.

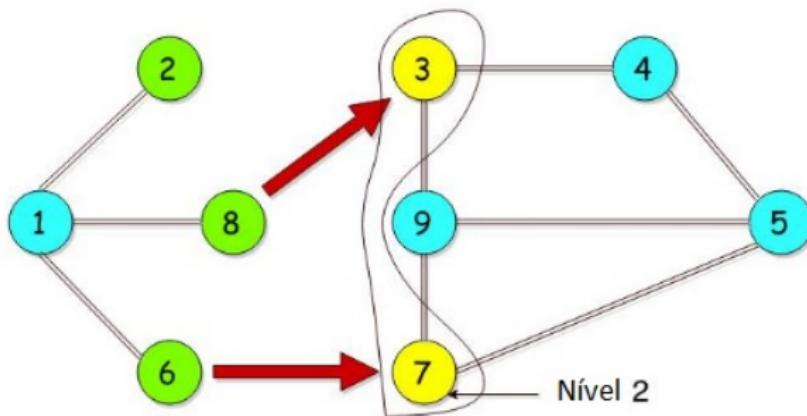
- Há várias ordens possíveis para percorrer os vértices de um grafo, cada ordem caracterizando um algoritmo de busca diferente. Os mais utilizados são:
 - Busca em Amplitude (BFS - Breadth First Search)
 - Busca em Profundidade (DFS - Depth First Search)
 - Best First Search

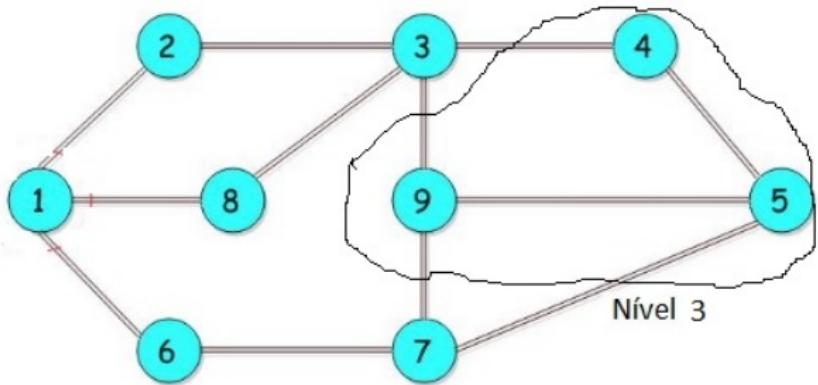
Busca em Amplitude

- BFS : Breadth First Search : Nos algoritmos de busca em amplitude, a partir de um vértice v , percorre-se todos os vértices adjacentes a v antes de passar para o próximo nível.









Algoritmo de BFS (algoritmo de Moore)

- Marque o nível de todos os vértices como -1
- Marque o vértice inicial como nível 0
- $\text{nível_corrente} = 0$
- Enquanto houver algum vértice com nível=-1
 - Para cada vértice com nível=nível_corrente, encontre todos os seus adjacentes ainda não marcados e marque-os com nível=nível_corrente+1
 - Se o vértice t foi alcançado, fim do algoritmo (este passo é usado se o objetivo é encontrar a distância de s a t)
 - $\text{nível_corrente} = \text{nível_corrente} + 1$

[Voltar para o índice](#)

- O cavalo no jogo de xadrez move-se em L, ou seja, duas casas em qualquer direção e uma para o lado. A figura a seguir mostra, a partir de uma posição (marcada com C), as casas que um cavalo pode atingir (marcadas com X).

	X		X	
X				X
		C		
X				X
	X		X	

Tabela: Movimentos possíveis para um cavalo no jogo de xadrez

- Considere o tabuleiro de xadrez da figura a seguir. Utilizando um algoritmo de caminho mínimo entre dois pontos (como o algoritmo de Moore) encontre o percurso mínimo que o cavalo deve fazer para ir da casa g1 (marcada com I) até a casa b1 (marcada com F) sem passar por nenhuma casa marcada com X.

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
								6
		X	X	X	X	X	X	5
		X	X	X	X		X	4
X		X	X		X			3
		X					X	2
X	F	X			X	I		

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
								6
		X	X	X	X	X	X	5
		X	X	X	X		X	4
X		X	X		X			3
		X					X	2
X	F	X			X	I		1

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
								6
		X	X	X	X	X	X	5
		X	X	X	X		X	4
X		X	X		X		1	3
		X		1			X	2
X	F	X			X	I		1

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
								6
		X	X	X	X	X	X	5
		X	X	X	X		X	4
X		X	X		X	2	1	3
		X		1	2		X	2
X	F	X			X	I		1

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
								6
		X	X	X	X	X	X	5
		X	X	X	X	3	X	4
X		X	X		X	2	1	3
		X		1	2		X	2
X	F	X	3		X	I	3	1

a	b	c	d	e	f	g	h	
		X	X			X	X	8
X	X	X	X					7
					4		4	6
		X	X	X	X	X	X	5
		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
	4	X		1	2		X	2
X	F	X	3		X	I	3	1

a	b	c	d	e	f	g	h	
		X	X	5		X	X	8
X	X	X	X		5		5	7
				4		4	6	
		X	X	X	X	X	X	5
5		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
4	X			1	2	5	X	2
X	F	X	3		X	I	3	1

a	b	c	d	e	f	g	h	
		X	X	5	6	X	X	8
X	X	X	X		5	6	5	7
6		6		4		4	6	
		X	X	X	X	X	X	5
5		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
4	X			1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7		X	X	5	6	X	X	8
X	X	X	X		5	6	5	7
	6		6	7	4	7	4	6
7	X	X	X	X	X	X	X	5
5		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
	4	X		1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7		X	X	5	6	X	X	8
X	X	X	X	8	5	6	5	7
	6		6	7	4	7	4	6
	7	X	X	X	X	X	X	5
5		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
	4	X		1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7		X	X	5	6	X	X	8
X	X	X	X	8	5	6	5	7
	6	9	6	7	4	7	4	6
	7	X	X	X	X	X	X	5
5		X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
	4	X		1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7	10	X	X	5	6	X	X	8
X	X	X	X	8	5	6	5	7
	6	9	6	7	4	7	4	6
10	7	X	X	X	X	X	X	5
5	10	X	X	X	X	3	X	4
X		X	X	4	X	2	1	3
	4	X		1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7	10	X	X	5	6	X	X	8
X	X	X	X	8	5	6	5	7
11	6	9	6	7	4	7	4	6
10	7	X	X	X	X	X	X	5
5	10	X	X	X	X	3	X	4
X	11	X	X	4	X	2	1	3
11	4	X		1	2	5	X	2
X	F	X	3	6	X	I	3	1

a	b	c	d	e	f	g	h	
7	10	X	X	5	6	X	X	8
X	X	X	X	8	5	6	5	7
11	6	9	6	7	4	7	4	6
10	7	X	X	X	X	X	X	5
5	10	X	X	X	X	3	X	4
X	11	X	X	4	X	2	1	3
11	4	X	12	1	2	5	X	2
X	F	X	3	6	X	I	3	1

- O código a seguir (`bfs_matriz_sem_fila.cpp`) implementa o algoritmo de Moore sobre uma matriz de adjacências.
- Tem como parâmetros o vértice inicial v_i , a matriz de adjacências e o vetor de níveis de cada vértice.
- A cada nível, a partir do nível 0, procura-se vértices do nível corrente e seta-se seus adjacentes ainda não marcados como $nível+1$
- Isso se repete até que não seja encontrado nenhum vértice não marcado

```

void bfs(int vi, int G[N][N], int nivel[N])
{
    for (int i=0; i<N; i++) nivel[i]=-1; // inicializa todos os níveis como -1
    nivel[vi]=0; // e o nível do vértice inicial como 0
    int NivelCorrente=0;
    while (1)
    {
        int trocou=0;
        for (int i=0; i<N; i++)
            if (nivel[i]==NivelCorrente)
                for (int j=0; j<N; j++)
                    if (G[i][j]==1 && nivel[j]==-1)
                    {
                        nivel[j]=NivelCorrente+1;
                        trocou=1;
                    }
        if (trocou==0) break;
        NivelCorrente++;
    }
}

```

- O algoritmo pode ser implementado de forma muito eficiente ($O(N+M)$) utilizando uma fila.

```
void bfs(int vi) // vi = vértice inicial
{
    int fila[1001],PA=1,TD=0,i; // PA = Põe aqui TD = Tira daqui
    fila[0]=vi;
    nivel[vi]=0; // vetor global com o nível de cada vértice
    while (PA!=TD)
    {
        int v=fila[TD++];
        for (i=1;i<=n;i++)
            {
                if (M[v][i]==1 && nivel[i]==-1)
                    {
                        fila[PA++]=i;
                        nivel[i]=nivel[v]+1;
                    }
            }
    }
}
```

Exercício de BFS no Hacker Rank

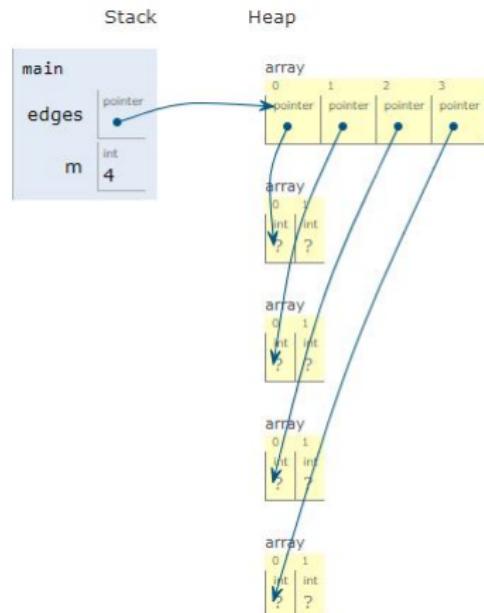
- Faça um cadastro (ou, se já tiver cadastro, logue-se) no HackerRank.
- Selecione o exercício Practice→Algorithms→Graph Theory→Breadth First Search: Shortest Reach
- Nos exercícios do HackerRank, o programa principal e a entrada de dados já é fornecida, deve-se apenas implementar a função pedida, para o qual é fornecido o protótipo. No caso desse exercício, é:

```
int* bfs(int n, int m, int edges_rows, int edges_columns, int**  
edges, int s, int* result_count)
```

```
int* bfs(int n, int m, int edges_rows, int edges_columns, int**  
edges, int s, int* result_count)
```

- n - Número de vértices no grafo
- m - Número de arestas no grafo
- A lista de arestas (edges) é fornecida em uma matriz dinâmica $m \times 2$, onde cada linha possui os números dos vértices ligados pela aresta. Os vértices são numerados a partir de 1.
- edges_columns = 2
- edges_rows = m
- s é o vértice inicial (source)
- result_count é o número de valores do vetor com o resultado

- A lista de arestas é passada como `int **edges` por ser uma matriz dinâmica, como mostra a figura abaixo:



- Uma matriz dinâmica como a mostrada acima pode ser criada pelo código abaixo:

```
int main() {  
    int **edges,m=4;  
    edges=(int **)malloc(m*sizeof(int *));  
    for (int i=0;i<m;i++)  
        edges[i]=(int *)malloc(sizeof(int)*2);  
    return 0;  
}
```

re

- Nesse exercício a função deve retornar (como valor de retorno da função) o endereço de um vetor com a distância de cada vértice ao vértice inicial multiplicada por 6. Para aqueles vértices que não são alcançáveis a partir do vértice inicial, o vetor deve conter -1.
- Esse vetor pode ser um vetor global, um vetor declarado como estático, pelo tamanho máximo:
 - static int result[1001];
- Uma variável estática é uma variável que, para efeito de escopo, é local, mas ela é alocada na mesma área das variáveis globais de modo que ao terminar a execução da função, ela continua existindo e mantém o valor entre chamadas da função.

- Ou o vetor pode ser alocado dinamicamente:

```
int *result=(int*)malloc((n-1)*sizeof(int));
```

- O problema em alocar dinamicamente é que a função chamadora deverá se encarregar de desalocar a memória alocada (C não tem garbage collector).
- O vetor retornado não pode ser global, porque variáveis locais são criadas na pilha e ao encerrar a execução da função a variável deixa de existir.
 - Um bom compilador avisa que a função está retornando um pointer para uma variável local. O do hackerrank avisa.

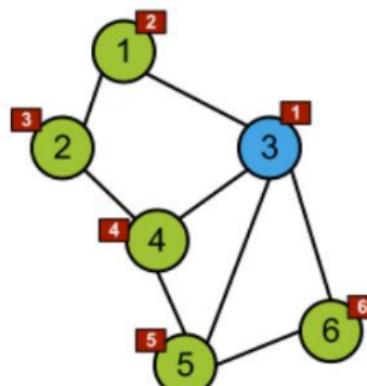
- HackerRank: Breadth First Search: Shortest Reach
- HackerRank: Snakes and Ladders: The Quickest Way Up
- URI Online Judge:
 - 1466 - Percurso em Árvore por Nível
 - 1621 - Labirinto
 - 2732 - Reino de Alice
 - 1550 - Inversão
 - 1910 - Ajude Clotilde
 - 1923 - Rerisson e o Churrasco
 - 2151 - Soco do Rulk

Uso de BFS em outros algoritmos de grafos

- Buscas em grafos podem ser utilizadas como parte de outros algoritmos para grafos.
- Pode-se verificar se um grafo é conexo (isso é, de cada vértice pode-se chegar a todos os outros por algum caminho) efetuando uma busca (BFS ou DFS) e verificando se algum vértice deixou de ser visitado.
- Pode-se verificar se um grafo é cíclico efetuando uma busca e verificando se algum vértice é visitado por mais de um caminho.

Algoritmos de DFS (Busca em Profundidade)

- Busca em profundidade (DFS : Depth First Search): Nos algoritmos de busca em profundidade, a partir de um vértice v escolhe-se uma aresta e percorre-se todos os vértices alcançáveis a partir daquela aresta, antes de passar para a próxima.
- A figura abaixo mostra a ordem de visitação dos vértices em uma DFS a partir do vértice 3



- A forma mais simples de efetuar uma busca em profundidade é através de uma função recursiva como a mostrada abaixo:

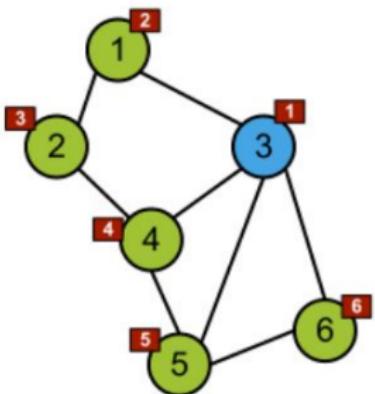
```
int visit[N]={0};  
int G[N][N];  
void dfs(int v){  
    visit[v]=1;//Visita-se  $v$  marcando-o como visitado  
    for (int i=0;i<N;i++)  
        if (G[v][i]==1 && visit[i]==0)  
            //Para cada vértice adjacente a  $v$  ainda não visitado:  
            dfs(i);  
}  
void main(){  
    for (int i=0;i<N;i++) visit[i]=0;  
    dfs(s); // s é o vértice inicial (source)  
}
```

- A implementação do DFS é muito sucinta porque ela é baseada em uma pilha, e utiliza a pilha do sistema para guardar todo o caminho percorrido do vértice inicial até a raiz, o que simplifica a implementação.
- O código a seguir efetua a DFS a partir do vértice 3 no grafo acima. A figura após mostra o estado da pilha durante a dfs, no momento em que o vértice 4 (embaixo) está sendo visitado.
- Pode-se ver na pilha no topo o vértice inicial (3) que chama o vértice 1, que chama o vértice (2) que chama o vértice 4.

```

int M[7][7]={ {0,0,0,0,0,0,0}, // 0 - não usado
{0,0,1,1,0,0,0}, // (1,2),(1,3)
{0,1,0,0,1,0,0}, // (2,1),(2,4)
{0,1,0,0,1,1,1}, // (3,1),(3,4),(3,5),(3,6)
{0,0,1,1,0,1,0}, // (4,2),(4,3),(4,5)
{0,0,0,1,1,0,1}, // (5,3),(5,4),(5,6)
{0,0,0,1,0,1,0}}; // (6,3),(6,5)
char visit[7]={0,0,0,0,0,0,0};
void dfs(int v)
{
    visit[v]=1;
    for (int i=1;i<=6;i++)
        if (M[v][i]&&visit[i]==0)
            dfs(i);
}
int main() {
    dfs(3);
    return 0;
}

```



dfs

v
i

dfs

v | int
 1
i | int
 2

dfs

int
2

dfs

v int
4
i int
?

- Há algoritmos não recursivos para efetuar busca em profundidade.
- Na falta da pilha da recursão para poder retornar ao vértice pai (vértice a partir de onde o vértice foi alcançado), é usada a marcação de arestas para que o algoritmo saiba para onde deve retornar ao terminar a exploração de um vértice.
- Algoritmos desse tipo são o algoritmo de Trémaux e o de Tarjan.

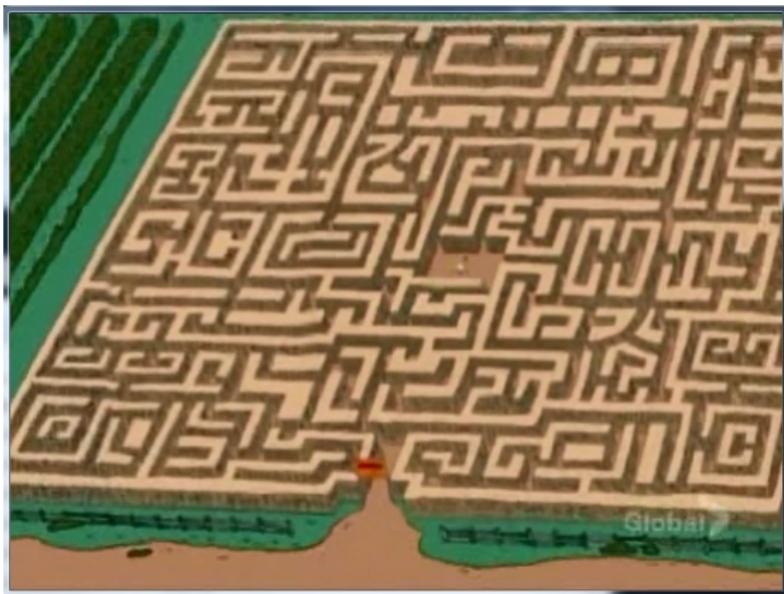
■ 1)Algoritmo de Trémaux (Even 79)

No algoritmo de Tremaux as passagens (uma passagem é a ligação de um vértice a uma aresta) são marcadas com E cada vez que são percorridas e ao entrar em um vértice pela primeira vez, a passagem usada é marcada com F, assinalando o caminho de volta. Este algoritmo percorre todas as arestas uma e somente uma vez em cada sentido.

[Voltar para o índice](#)

- 1** $v \leftarrow s$
- 2** Se não há passagens não marcadas em v , vá para 4
- 3** Escolha uma passagem não marcada (v, u) , marque com E e vá para u . Se u tem alguma passagem marcada (isto é, não é a primeira vez que é alcançado) marque a passagem, através da qual u foi alcançado, com E , volte pela passagem para v , e vá para o passo 2. Se u não tem passagens marcadas (isto é, é um vértice novo), marque a passagem através da qual u foi alcançado com F , $v \leftarrow u$ e vá para o passo 2.
- 4** Se não há passagens marcadas com F , o algoritmo termina (está-se de volta em s e a varredura do grafo está completa)
- 5** Use a passagem marcada com F , atravesse a aresta até u , $v \leftarrow u$ e vá para o passo 2.

Aplicação Prática - Simpsons S18E20



Aplicação Prática - Simpsons S18E20



Aplicação Prática - Simpsons S18E20



Algoritmo de Hopcroft e Tarjann (Even 79)

- Basicamente o mesmo de Tremaux mas numera os vértices à medida que são alcançados.
- O número do vértice é armazenado em $k(v)$.
- Também gera um vetor $f(v)$ com o número do vértice "pai" de cada um, ou seja, aquele vértice a partir do qual v foi alcançado.
- Este vetor substitui a marcação da passagem por F no algoritmo de Tremaux.

- 1** Marque todas as arestas como não usadas. Para todo $v \in V$, $k(v) \leftarrow 0$, $i \leftarrow 0$ e $v \leftarrow s$.
- 2** $i \leftarrow i + 1$, $k(v) \leftarrow i$
- 3** Se v não tem arestas incidentes não utilizadas, vá para o passo 5.
- 4** Escolha uma aresta incidente não utilizada $e = (v, u)$. Marque e como usada. Se $k(u) \neq 0$, vá para o passo 3. Senão (ou seja, se $k(u) = 0$) então $f(u) \leftarrow v$, $v \leftarrow u$ e vá para o passo 2.
- 5** Se $k(v) = 1$ então termina o algoritmo
- 6** $v \leftarrow f(v)$ e vá para o passo 3.

Def: O **comprimento** de um caminho é o número de arestas do caminho. Ex: 4,5,1,3,2,1 tem comprimento 5

Def: A **distância** entre dois vértices é o comprimento do menor caminho que une os dois vértices.

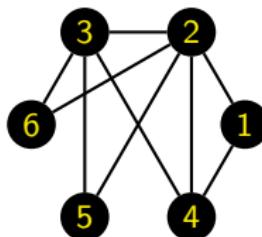


Figura: $d(1, 6) = 2$

Def: O **complemento** de um grafo $G = (V, E)$ é o grafo G' que possui o conjunto de arestas que não estão em E .

$$G' = (V', E')$$

$$1) V' = V$$

$$2) \forall v, w \in V, (v, w) \notin E \Rightarrow (v, w) \in E'$$

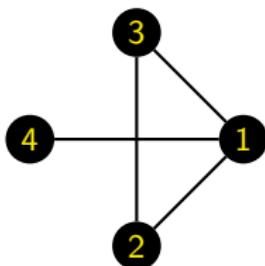


Figura: Grafo G

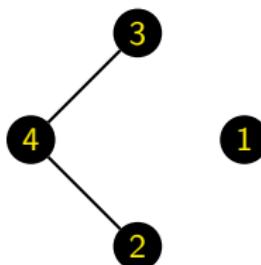


Figura: G' (complemento do Grafo G)

- **Def:** Um grafo é **completo** quando existe uma aresta entre cada par de vértices.
- Um grafo completo de N vértices é denotado por K_N **Def:** Uma **clique** de um grafo G é um subgrafo completo.



Figura: K_2

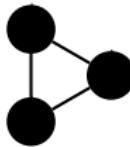


Figura: K_3

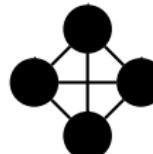


Figura: K_4

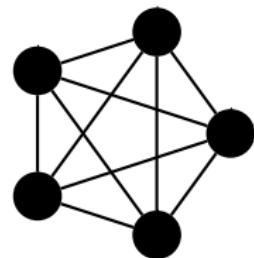


Figura: K_5

Figura: Grafos completos

Def: Um grafo **valorado** (ou rotulado) é um grafo com valores associados aos vértices e/ou arestas. Exercícios:

Para o grafo

$$G = (V_1, E_1)$$

$$V_1 = \{1, 2, 3, 4, 5\}$$

$$E_1 = \{(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (4, 1), (5, 5)\}$$

- 1 Desenhe uma representação geométrica
- 2 Há um caminho Hamiltoniano? Qual?
- 3 Há um ciclo Euleriano? Qual?
- 4 Quais os vértices adjacentes a 4?
- 5 Há algum laço em G_1 ?
- 6 G_1 é fortemente conexo? Por quê?
- 7 Quais os caminhos entre 1 e 4?
- 8 Qual a distância entre 1 e 4?
- 9 Defina um subgrafo induzido G_2 de G_1 tal que $V_2 = \{1, 3, 4\}$
- 10 Determine o complemento do grafo:

Árvores

Def: Uma **árvore** é um grafo não dirigido acíclico e conexo : $T(V, A)$, com n vértices e $n - 1$ arestas.

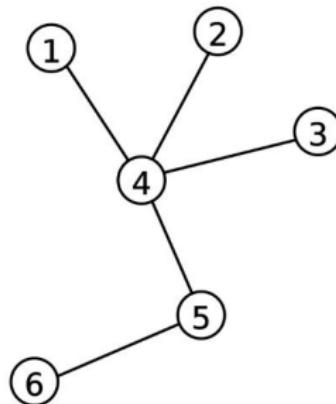
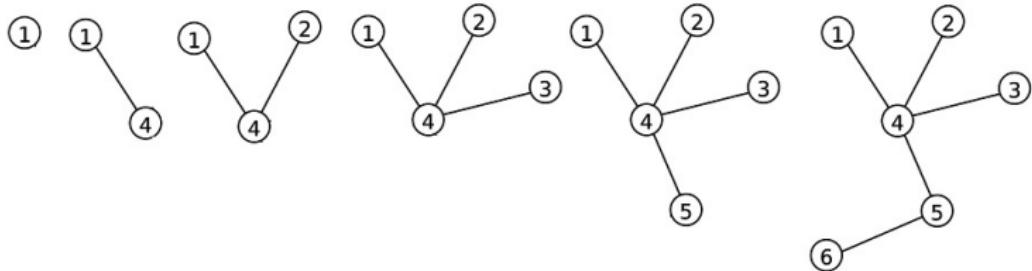
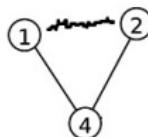


Figura: Árvore

- A relação entre o número de vértices e o número de arestas pode ser verificada na construção de uma árvore a partir de um vértice inicial.
- Inicialmente a árvore tem 1 vértice e 0 arestas.
- A cada passo são inseridos um novo vértice e uma aresta, mantendo o grafo conexo e acíclico, e mantendo a relação entre número de vértices e arestas



- Se durante esse processo de construção for inserida uma aresta ligando dois vértices já existentes, isso fechará um ciclo, e o grafo deixará de ser uma árvore.
- Da mesma forma, se for inserido um vértice sem a aresta correspondente, isso deixará o grafo desconexo, e ele deixará de ser uma árvore.



- Pode-se verificar as duas condições de uma árvore através de uma busca (dfs ou bfs) ligeiramente modificada.
- Se após executar a busca todos os vértices foram visitados, o grafo é conexo.
- Se durante a busca nenhum vértice for alcançado mais de uma vez, ele é acíclico.
- A dfs modificada a seguir retorna 0 se o grafo não é cíclico e 1 se o grafo é cíclico.

```

int dfs(int G[N][N], int v, int pai[N], int visitados[N])
{
    visitados[v]=1;
    for (int i=0;i<N; i++)
        if (G[v][i])
            if (!visitados[i])
            {
                pai[i]=v;
                if (dfs(G, i, pai, visitados))
                    return 1; // achou ciclo
            }
            else // i já foi visitado
                if (i!=pai[v])
                    return 1; // fechou ciclo
    return 0; // não tem ciclo
}

```

```
int ciclo_dfs(int G[N][N])
{
    int visitados[N], pai[N];
    for (int i=0;i<N;i++) visitados[i]=0;
    for (int i=0;i<N;i++)
        if (!visitados[i] && dfs(G,i,pai,visitados))
            return 1;
    return 0;
}

int main()
{
    int G[N][N]={{0,1,0,0},
    {1,0,1,0},
    {0,1,0,1},
    {0,0,1,0}};
    int visitados={0},pai[N];
    printf("%d\n",ciclo_dfs(G));
}
```

- A condição de acíclico e conexo, que resulta na relação entre o número de vértices e arestas, torna as condições a seguir equivalentes :
 - 1 G não tem ciclos, mas o acréscimo de qualquer aresta a G cria um ciclo;
 - 2 Para qualquer dois vértices em G existe somente um caminho entre eles;
 - 3 G é conexo, e a remoção de qualquer aresta o torna desconexo.

- Árvores no contexto de Teoria de Grafos normalmente não possuem hierarquia entre os nodos, os nodos não possuem relação pai-filho e elas não possuem raiz.
- Em algumas situações, árvores podem ter raiz e ser dirigidas.
- Uma árvore que possui uma raiz e hierarquia entre os nodos é dita **enraizada**
- Árvores e algoritmos envolvendo árvores aparecem frequentemente em situações em que se busca um subconjunto de arestas de custo mínimo, bem como parte de outros algoritmos de grafos.

- **Def:** Seja o grafo $G(V, E)$, e o subgrafo $SG(V_1, E_1)$. SG é um **subgrafo gerador** do grafo G sss $V = V_1$. Quando o subgrafo gerador é uma árvore recebe o nome de **árvore geradora**.
- **Def:** Uma **árvore geradora** de um grafo $G = (V, E)$ é obtida pela remoção dos ciclos de um grafo G . O grafo resultante é uma árvore e contém todos os vértices de G .

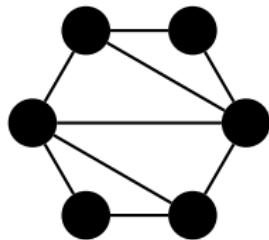


Figura: Grafo

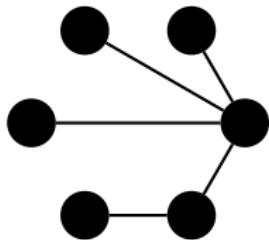


Figura: A.G.1

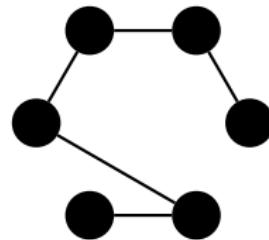


Figura: A.G. 2

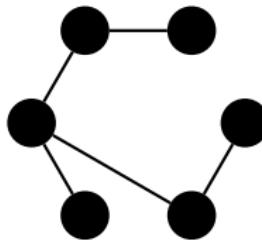


Figura: A.G. 3

Figura: Grafo e árvores geradoras

- Todo grafo conexo G possui árvore geradora. Caso um grafo não seja conexo, pode-se obter uma árvore geradora de cada componente conexo e ao conjunto delas denomina-se **floresta geradora**.
- Obs: Um subgrafo gerador pode conter ciclos. Uma árvore geradora não pode.

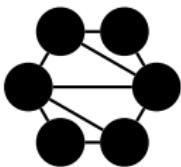


Figura: Grafo

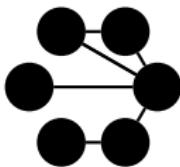


Figura: S.G.1

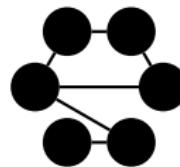
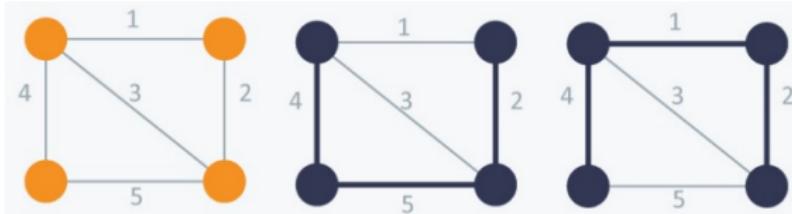


Figura: S.G. 2

Figura: Grafo e subgrafos geradores

- **Def:** Uma **árvore geradora mínima** (minimum spanning tree) é uma árvore geradora que apresenta a menor soma de valores associados às arestas (custo ou peso mínimo).
- **Aplicações :** Caminho de menor custo para ligar elementos como cidades, computadores ou centrais elétricas através de linhas telefônicas, fios, estradas, etc. Os valores associados podem significar pesos ou valores absolutos (p.ex., distâncias)
- A figura abaixo mostra duas árvores geradoras (em azul) para o grafo à esquerda. A primeira não é mínima (tem custo 11) e a segunda é mínima (custo 7)



Algoritmo de Prim

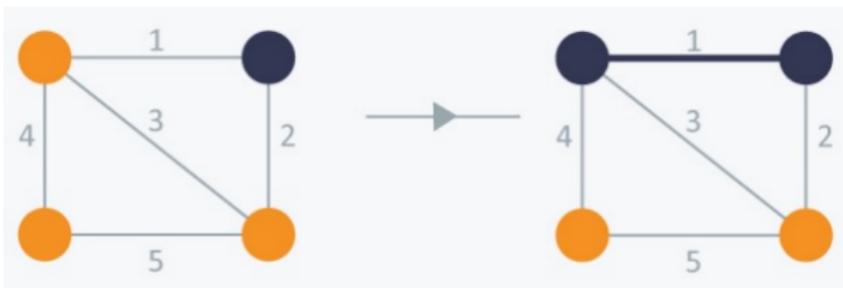
- Um algoritmo bastante usado para cálculo da árvore geradora mínima é o algoritmo de Prim. Ele mantém dois conjuntos de vértices, os que já foram incorporados à árvore e os ainda não selecionados.



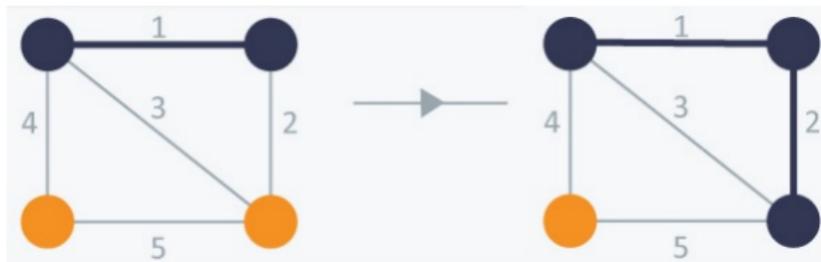
Figura: Robert Clay Prim
(1921-...)

- Ele inicia a montagem da árvore a partir de um vértice inicial, e a cada passo acrescenta à árvore geradora a aresta de menor valor que liga um vértice já selecionado a um vértice ainda não selecionado.
- A sequência de figuras a seguir ilustra o processo:

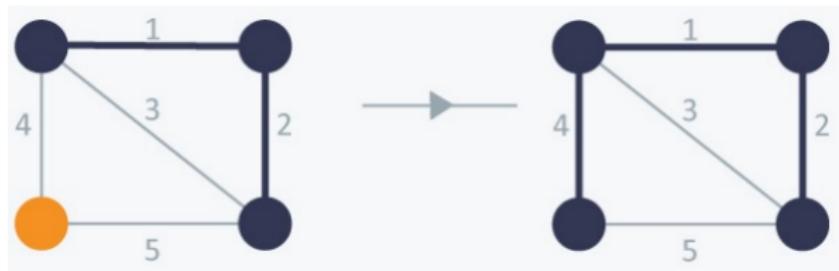
- Iniciando a construção da árvore pelo vértice superior direito da figura (poderia ser qualquer um). Escolhe-se a aresta de menor valor que liga um vértice ainda não selecionado a um vértice já selecionado (no caso, a aresta de valor 1).



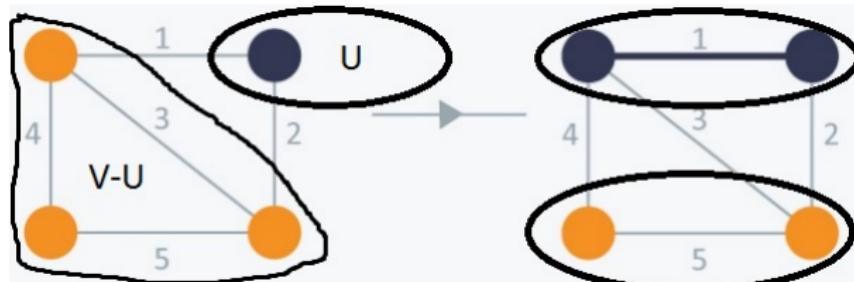
- No passo seguinte seleciona-se a aresta de menor valor que liga um vértice já selecionado (azuis) a um ainda não selecionado e acrescenta-se o vértice e a aresta à arvore.

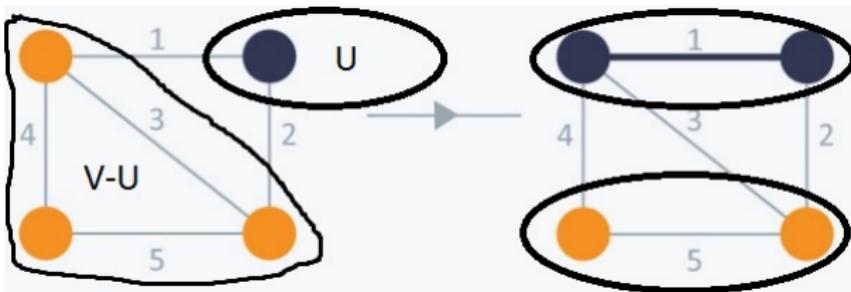


- Isso se repete até que todos os vértices tenham sido incluídos na árvore geradora

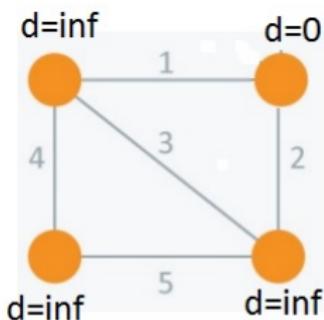


- O algoritmo de Prim pode ser resumido assim.
 - Considere V o conjunto de todos os vértices do grafo original e U o conjunto de vértices já selecionados (que já fazem parte da árvore):
 - Enquanto houver vértices em $V-U$:
 - Selecione a menor aresta (u,v) em que o vértice u está em U e o vértice v está em $V-U$
 - Adicione o vértice v ao conjunto U e a aresta (u,v) à árvore

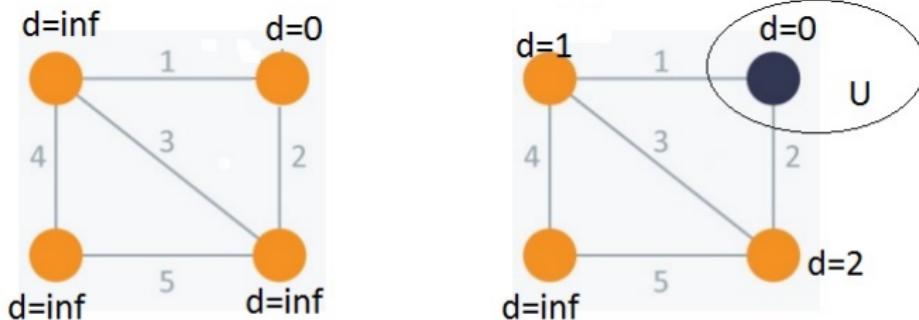


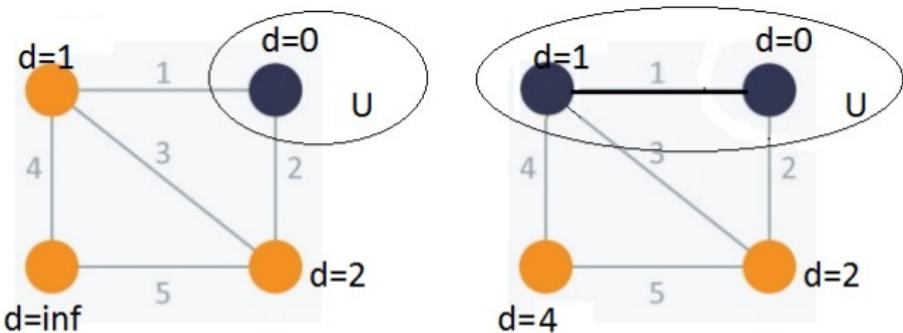


- A identificação de qual vértice deve ser selecionado a cada passo pode ser feita de forma eficiente mantendo, para cada vértice, a menor distância dele à subárvore já montada (chamemos esse vetor de \mathbf{d}).
- Inicialmente considera-se que o vértice inicial está a uma distância 0 e todos os outros vértices estão a uma distância infinita.

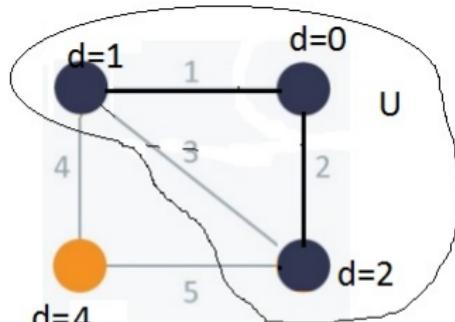
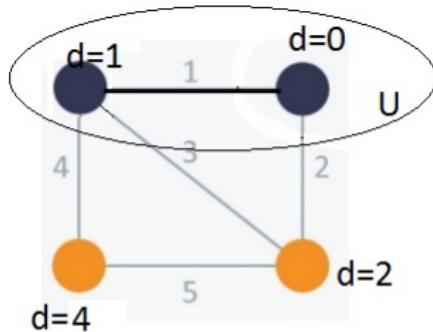


- Para identificar se um dado vértice v já foi selecionado, pode-se usar um vetor de flags (U).
- A cada passo seleciona-se, dentre os vértices ainda em $V-U$, o vértice u de menor valor d .
- Acrescenta-se o vértice u a U , e recalcula-se a distância mínima de cada vértice adjacente a u .





- Seleciona-se, dentre os vértices que ainda não estão em U , o de menor distância (no caso, o vértice com $d=1$)
 - Atualiza-se a distância dos vértices adjacentes ao selecionado



- Seleciona-se, dentre os vértices que ainda não estão em U , o de menor distância (no caso, o vértice com $d=2$)
 - Atualiza-se a distância dos vértices adjacentes ao selecionado

■ Código Prim1.c. Retorna o custo da árvore mínima.

```
int prim(int G[TAM][TAM]) {
    int i;
    int U[TAM], d[TAM]; // inicialmente nenhum vértice está em U
    for (i=0;i<TAM; i++) {U[i]=0;d[i]=INF;}
    d[0]=0; // iniciará montagem a partir do vértice 0
    for (i=0;i<TAM; i++)
    {
        int menor=INF, j, u; // seleciona vértice u de menor aresta
        for (j=0;j<TAM; j++)
            if (U[j]==0 && d[j]<menor) {menor=d[j]; u=j;}
        U[u]=1; // incorpora vértice u ao conjunto U
        for (j=0;j<TAM; j++) // atualiza distância de todos
            // os vértices adjacentes a u
            if (U[j]==0 && G[u][j]<d[j])
                d[j]=G[u][j];
    }
    int soma=0;
    for (i=0;i<TAM; i++) soma+=d[i];
    return soma;
}
```

- A versão anterior do código retorna a soma total da árvore geradora, mas não fornece o conjunto de arestas que a compõe.
- Uma forma de obter essa informação é manter, para cada vértice, a informação de qual seu vértice "pai", ou seja, a partir de qual vértice ele foi atingido, que resultou na menor distância.
- O código Prim2 mostra as alterações necessárias.

■ Código Prim2.c. Retorna o custo da árvore mínima.

```
int prim(int G[TAM][TAM]) {
    int i;
    int U[TAM], d[TAM]; // inicialmente nenhum vértice está em U
    for (i=0;i<TAM; i++) {U[i]=0;d[i]=INF;}
    d[0]=0; // iniciará montagem a partir do vértice 0
    for (i=0;i<TAM; i++)
    {
        int menor=INF, j, u; // seleciona vértice u de menor aresta
        for (j=0;j<TAM; j++)
            if (U[j]==0 && d[j]<menor) {menor=d[j]; u=j;}
        U[u]=1; // incorpora vértice u ao conjunto U
        for (j=0;j<TAM; j++) // atualiza distância de todos
            // os vértices adjacentes a u
            if (U[j]==0 && G[u][j]< d[j])
                {d[j]=G[u][j]; pai[j]=u;}
    }
    int soma=0;
    for (i=0;i<TAM; i++) soma+=d[i];
    return soma;
}
```

- O passo de maior custo a cada iteração é a seleção da aresta de menor valor, que tem um custo $O(N)$.
- O custo desse passo pode ser reduzido mantendo as distâncias em uma min-heap, um fila de prioridades em que o custo para obter o menor valor tem custo $O(1)$ e o custo para reorganizar a fila é de $O(\log N)$, tornando o custo total do algoritmo da ordem de $O(N \log N)$.

- Implemente o algoritmo de Prim sobre uma matriz de adjacências.
- HackerRank: Prim's (MST) : Special Subtree
- URI Online Judge (todos podem ser resolvidos com matriz de adjacências):
 - 1552 - Resgate em Queda Livre
 - 1774 - Roteadores
 - 2404 - Reduzindo detalhes em um mapa
 - 2522 - Rede do DINF
 - 2550 - Novo Campus
 - 2683 - Espaço de Projeto

Algoritmo de Kruskal

- O algoritmo de Kruskal é outro algoritmo para cálculo da árvore geradora mínima



Figura: Joseph Kruskal
(1928-2010)

- No algoritmo de Kruskal, ao contrário de Prim, durante o processo de construção da árvore geradora não há a preocupação de manter conexa a árvore sendo construída.
- O algoritmo é basicamente o seguinte:
 - 1 Iniciar com os n vértices, sem nenhuma aresta
 - 2 A cada etapa do cálculo introduzir, dentre as arestas restantes, a de menor valor que não completa um ciclo.
 - 3 Parar quando o número de arestas introduzidas for $n - 1$

Índice

- Como obter, de forma eficiente, a aresta de menor custo entre as restantes?
 - Uma fila de prioridades (min-heap).
 - Na verdade basta ordenar as arestas em ordem crescente no início, já que o valor das arestas não muda.
- Como verificar, de forma eficiente, se a introdução da aresta forma um ciclo?
 - Um disjoint-set.

Árvore geradora máxima

- Em algumas situações busca-se encontrar a árvore geradora de maior soma ao invés de menor soma. (URI 1476).
- Nesse caso também pode-se usar os algoritmos de Prim e Kruskal, selecionando a cada passo a maior aresta.
- Ou invertendo o sinal de todas as arestas invertendo a relação de ordem entre elas.

O Problema do caminho mínimo

- o problema do caminho mínimo consiste na minimização do custo de travessia de um grafo entre dois vértices.
- Esse problema ocorre nas mais diversas áreas de aplicação como:
 - Redes de computadores (roteamento de pacotes)
 - Telecomunicações
 - Transportes
 - Planejamento de rotas (Google Maps, Waze)
 - Etc., etc., etc...

[Voltar para o índice](#)

- A situação mais comum é procurar o menor caminho entre dois vértices, mas em algumas aplicações pode-se querer calcular:
 - O menor caminho entre todos os pares de vértices
 - O menor caminho de cada vértice a um único vértice destino (também chamado SSSP - Single Source Shortest Path)
 - O menor caminho de um vértice de origem a todos os outros
- havendo algoritmos específicos para cada situação.

[Voltar para o índice](#)

- Como já foi visto, se o grafo não é valorado e busca-se o caminho de menor número de arestas, a busca em amplitude encontra a menor distância entre dois vértices ou a menor distância entre um vértice e todos os outros.
- Se o grafo é uma árvore (acíclico e dirigido) a distância entre quaisquer dois vértices pode ser encontrada pelo algoritmo de busca de ancestral comum (**LCA** - Lowest Common Ancestor)

- Para grafos valorados existem diversos algoritmos disponíveis
 - Algoritmo de Dijkstra - Utilizado em situações com um único vértice de origem (e um ou mais vértices de destino) em grafos cujas arestas tenham peso maior ou igual a zero.
 - Algoritmo de Bellman-Ford - Utilizado em situações com um único vértice de origem. Arestas podem ter pesos negativos, mas o grafo não pode ter ciclos de soma negativa.
 - Algoritmo A* - Semelhante a Dijkstra, mas tem um desempenho melhor em situações em que é possível estimar a distância de cada vértice ao vértice destino.
 - Algoritmo de Floyd-Warshall - Determina a distância entre todos os pares de vértices de um grafo.
 - Algoritmo LCA (Lowest Common Ancestor - Ancestral Comum Mais Próximo) - Utilizado quando o grafo é uma árvore (acíclico e conexo)

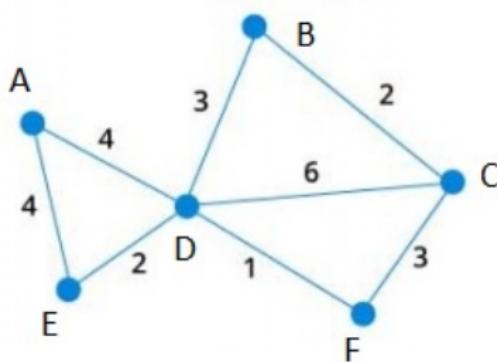
Algoritmo de Dijkstra para dígrafos valorados

- Edsger Dijkstra foi um dos pioneiros da computação.
- Suas principais contribuições foram o uso de semáforos em sistemas operacionais e o algoritmo para caminho mínimo em grafos valorados.
- Seu algoritmo para caminho mínimo foi criado em 1956 e publicado em 1959.

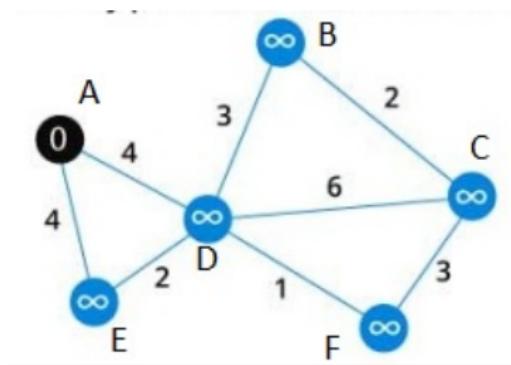


Figura: Edsger Dijkstra
(1930-2002)

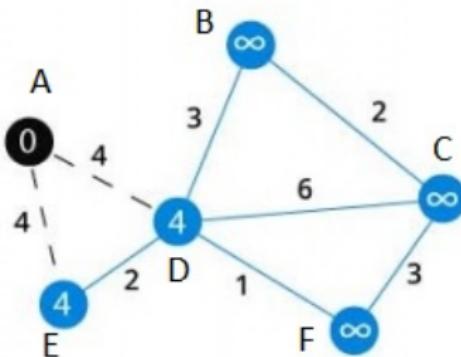
- No algoritmo de Dijkstra, a partir de um vértice inicial, a cada passo é selecionado, dentre os vértices remanescentes (que ainda não foram selecionados), o de menor valor.
- Digamos que se deseja calcular a distância do vértice A aos outros no grafo abaixo:



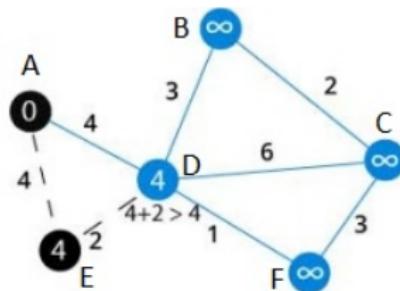
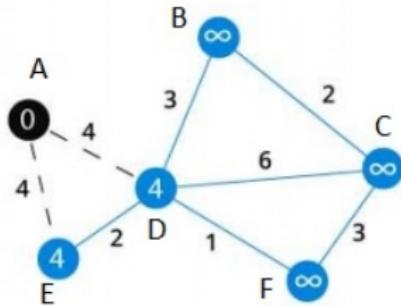
- Durante todo o processo mantém-se a informação da menor distância já encontrada de cada vértice x ($\text{dist}(x)$) ao vértice inicial.
- Inicialmente o vértice inicial tem distância igual a zero e os outros vértices tem distância infinita (nenhum caminho foi explorado até o momento).



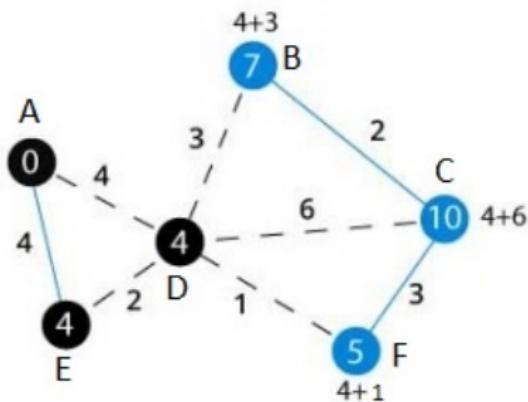
- A cada passo seleciona-se, entre os ainda não selecionados, o vértice u de menor distância $\text{dist}(u)$ e atualiza-se a distância de todos seus adjacentes.
 - A distância calculada para cada vértice v é a distância já calculada para o vértice u , mais o custo da aresta (u,v) .
 - Se a distância calculada é menor que $\text{dist}(v)$, $\text{dist}(v)$ é atualizada.



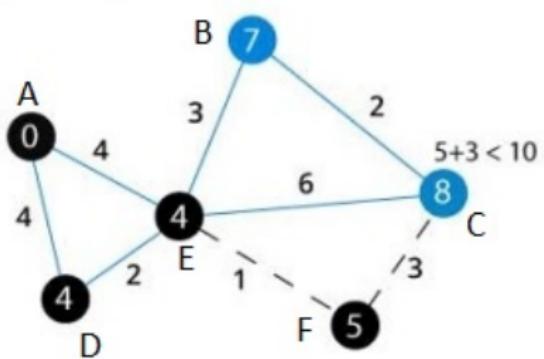
- Nesse passo os vértices **E** e **D** tem o mesmo valor já calculado (4). Qualquer um deles pode ser selecionado, e essa escolha não afetará o resultado final do algoritmo.
 - Selecionado o vértice E, $\text{dist}(D)$ pode ser atualizado, mas a nova distância ($\text{dist}(E)+2=6$) é maior que $\text{dist}(D)$ e esse permanece inalterado.



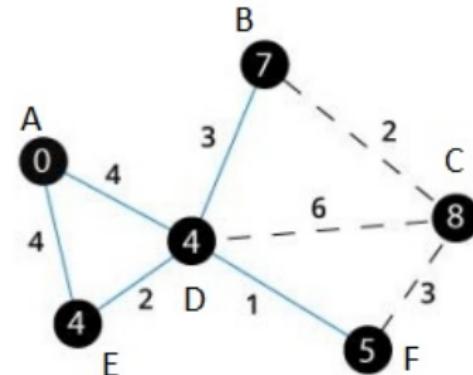
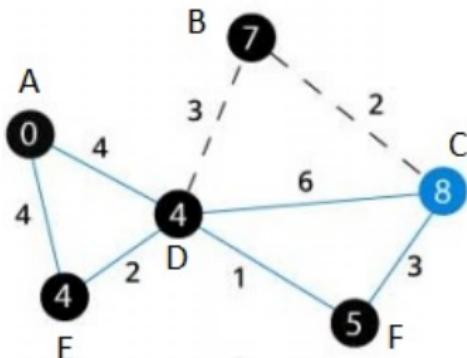
- O próximo vértice selecionado é o vértice D ($\text{dist}(D)=4$).
- Atualiza-se a distância dos vértices adjacentes a D que ainda não foram selecionados.



- O próximo vértice selecionado é o vértice F ($\text{dist}(F)=5$).
 - Atualiza-se a distância dos vértices adjacentes a F que ainda não foram selecionados.
 - O vértice C tem sua distância reduzida de 10 para 8, resultado da descoberta de um caminho mais curto de A até C, através de F (o melhor caminho anterior vinha direto de E)



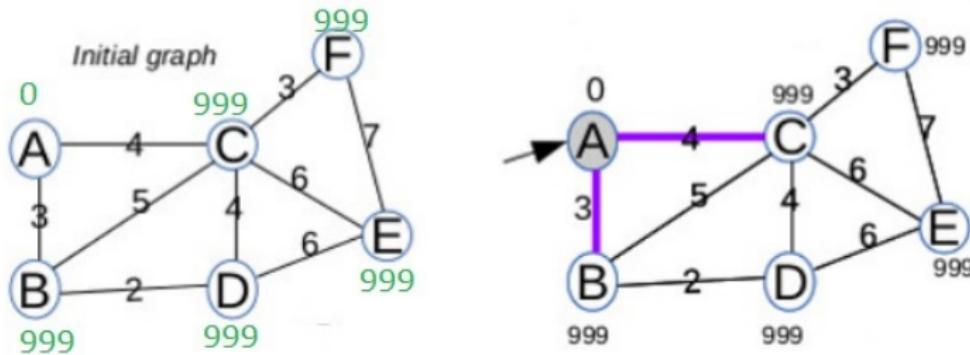
- Isso se repete até que todos os vértices tenham sido selecionados ou até que o vértice destino (caso se procure a distância até um vértice específico) seja selecionado.
- A distância de um vértice no momento em que ele é selecionado é a menor distância possível até ele a partir do vértice inicial.
- Enquanto ele não for selecionado, sua distância pode ser reduzida encontrando-se um caminho mais curto até ele.



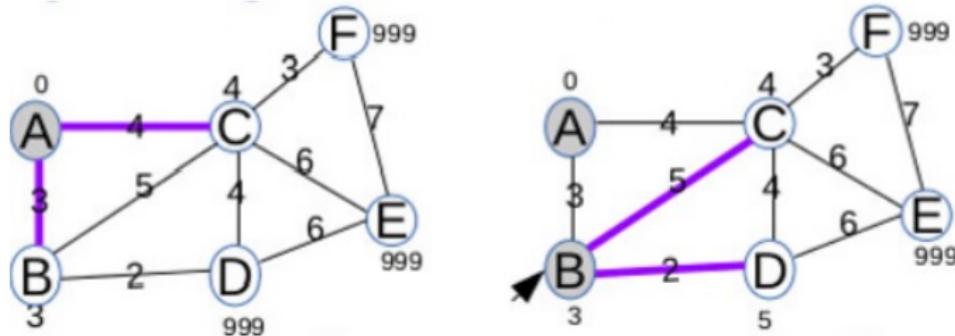
Algoritmo de Dijkstra

Objetivo : Busca encontrar o menor caminho entre s e t . $d[v]$ designa a menor distância de v a s . $l(e)$ designa o peso da aresta e . Q representa o conjunto de vértices para os quais ainda não se calculou a distância.

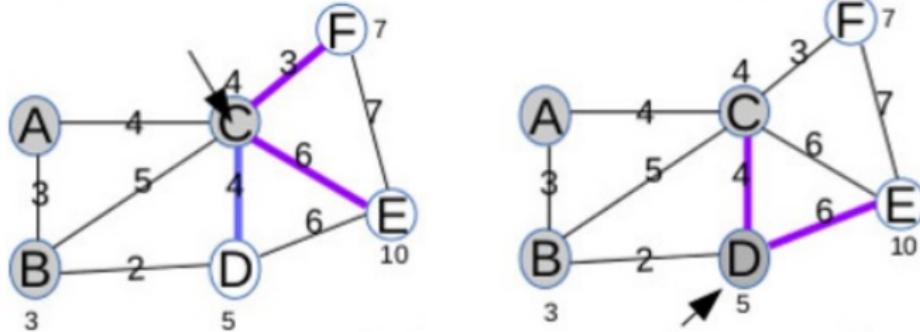
- $d[s] \leftarrow 0$ e para todo $v \neq s$, $d[v] \leftarrow \infty$
- $Q \leftarrow V$
- Enquanto houver algum vértice não selecionado ($Q \neq \emptyset$)
 - Seja u um vértice em Q para o qual $d[u]$ seja mínimo.
 - Se $u = t$ então termina o algoritmo
 - Para toda aresta $e = (u, v)$,
 - se $v \in Q$ e $d[v] > d[u] + l(e)$
 - então $d[v] \leftarrow d[u] + l(e)$
 - remova u do conjunto Q



- Inicialmente todos os vértices começam com distância ∞ e o vértice inicial começa com distância 0. O vértice de menor distância (vertice A, com distância 0) é selecionado.

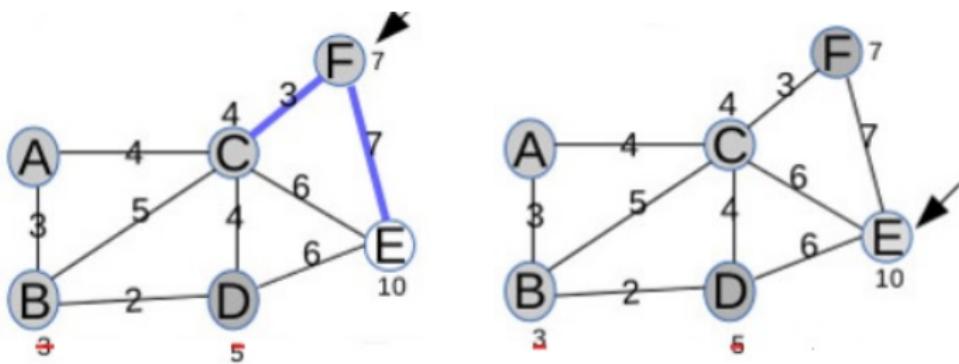


- As distâncias dos vértices adjacentes a A (B e C) são atualizadas. Em seguida B é selecionado e seus adjacentes (C e D) são atualizados. Como $d[C]$ é menor que $d[B]+5$, $d[C]$ não é alterado.



- O vértice de menor distância entre C, D, E e F (C) é selecionado e seus vértices adjacentes (D, E e F) são atualizados. Como $d[D]$ é menor que $d[C]+4$, $d[D]$ não é alterado.
- Em seguida o vértice de menor distância entre D, E e F (D) é selecionado. Como seu único vértice adjacente (E) tem $d[E]$ menor que $d[D]+6$, $d[E]$ não é alterado.

Exercícios

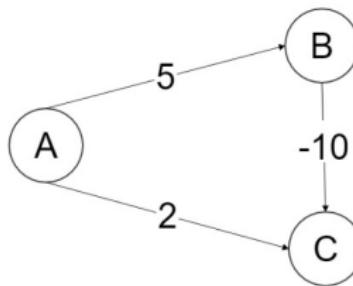


- O vértice de menor distância entre E e F (F) é selecionado. Como seu único vértice adjacente (E) tem $d[E]$ menor que $d[F]+7$, $d[E]$ não é alterado.
 - F é selecionado e termina o algoritmo.

Índice

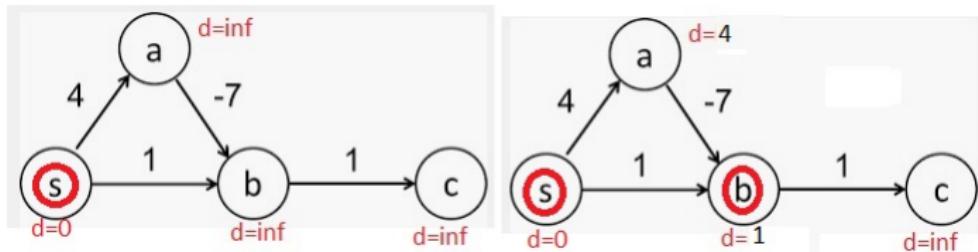
Exercício

- A complexidade do algoritmo depende da estrutura usada para armazenar as distâncias $d[v]$. Se for utilizada uma fila de prioridades, a complexidade do algoritmo é $O(V \log E)$. Se for utilizado um vetor, é $O(V^2)$
- Para grafos não dirigidos, basta substituir cada aresta por dois arcos, um em cada sentido, e aplicar o algoritmo.
- O algoritmo não funciona se as arestas podem ter pesos negativos (figura abaixo).



Algoritmo de Bellman-Ford para grafos com arestas de peso negativo

- O algoritmo de Dijkstra não funciona para arestas com peso negativo. No grafo abaixo, após selecionado o vértice s , as distâncias de **a** e **b** são atualizadas.



- No próximo passo o algoritmo de Dijkstra irá selecionar o vértice b ($d=1$) supondo que não há uma caminho menor que 1 até b , e não encontrará o caminho $s-a-b$, de custo -3.

- Nesse caso (arestas negativas mas sem ciclo de soma negativa) pode ser usado o algoritmo de Bellman-Ford a seguir. Se existirem ciclos de custo negativo, o algoritmo indica sua existência.
- A cada iteração o algoritmo percorre TODAS as arestas, verificando quais podem diminuir o valor da distância de algum vértice.
- Para N vértices, no máximo após N iterações as distâncias devem ter estabilizado para o valor mínimo.
- Se após alguma iteração nenhuma das distâncias for atualizada, as distâncias são finais e o algoritmo pode ser interrompido.

- Ao contrário do algoritmo de Dijkstra, que após selecionar um vértice não recalcula mais sua distância, o algoritmo de Bellman-Ford nunca supõe que atingiu a menor distância para um vértice.
- Se após N iterações as distâncias continuarem diminuindo, o grafo tem ciclos com custo negativo.
- No código a seguir, o vetor **p** guarda para cada vértice **v** a informação do "pai" de cada vértice, ou seja, o vértice a partir de onde **v** foi alcançado, permitindo reconstituir o caminho mínimo.

Algoritmo de Bellman-Ford

- $d[s] \leftarrow 0$ e para todo $v \neq s$, $d[v] \leftarrow \infty$
- para i de 1 até $V-1$
 - para toda aresta $e = (u, v)$,
se $d[v] > d[u] + l(e)$ então
 $d[v] \leftarrow d[u] + l(e)$
 $pai[v] \leftarrow u$
 - para toda aresta $e = (u, v)$,
se $d[v] > d[u] + l(e)$ então
"Há ciclos com custo negativo!!!"

Índice

Shortest Path Faster Algorithm (SPFA)

- O algoritmo de Bellman-Ford testa repetidamente todas as arestas (u,v) até que não haja nenhuma aresta que melhore os valores das distâncias. Nesses testes uma aresta (u,v) só irá melhorar a distância $d(v)$ se o vértice u teve sua distância atualizada na iteração anterior ou na corrente.
- O algoritmo SPFA verifica a cada passo somente as arestas (u,v) originadas em vértices u que foram atualizados na iteração corrente ou na anterior.
- Esse controle pode ser feito colocando em uma fila os vértices recém atualizados, e a cada passo retirando um vértice da fila e testando as arestas originadas nele.

Índice

O Uniform-Cost Search (UCS)

- O *Uniform-Cost Search* é uma variante do algoritmo de Dijkstra em que é mantido um mapa dos vértices que inicialmente contém apenas o vértice inicial. Ele é colocado em uma fila de prioridades, e a partir dele, a cada passo, é removido um vértice da fila e colocados os adjacentes no mapa e na fila de prioridades.
- Dessa forma, esse algoritmo pode ser utilizado para grafos implícitos, como uma busca em um espaço de estados, em que inicialmente não se sabe o conjunto total de vértices.

■ ▶ Botão

Índice

Exercícios do URI Online Judge de Dijkstra

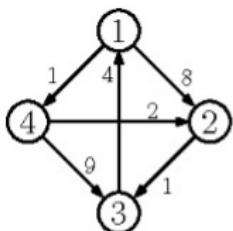
- 1454 - O País das Bicicletas (dijkstra modificado)
- 1148 - Países em Guerra

O Algoritmo de Floyd-Warshall

- Em situações em que se busca o caminho mínimo entre todos os pares de vértices em grafos não esparsos, o algoritmo de Floyd-Warshall é mais eficiente que o algoritmo de Dijkstra.
 - Em grafos esparsos, o cálculo de Dijkstra a partir de cada vértice a todos os outros pode apresentar um desempenho melhor.
- É baseado em programação dinâmica, de forma que a solução final vai sendo composta a partir das soluções parciais.
- A cada passo k o algoritmo calcula o caminho mínimo entre dois vértices que passa apenas por vértices de numeração menor ou igual a k .

Índice

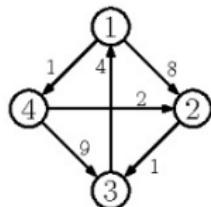
- Durante todo o processo é mantida uma matriz com o comprimento, para cada par de vértices i,j o comprimento do menor caminho que passa em vértice de numeração menor ou igual a k .
 - Inicialmente a matriz mantém o custo das arestas que ligam cada par de vértices ($k=0$).
 - Para os pares de vértices que não possuem ligação direta, a distância é considerada infinita.



$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

K=0 (distâncias diretas)

- No passo seguinte ($k=1$) calcula-se para cada par de vértices (i,j), o custo do menor caminho que passa em vértices de numeração no máximo igual a 1.
- No grafo abaixo, os caminhos que passam pelo vértice 1 seriam os caminhos 3-1-2 e 3-1-4.
- O custo direto do caminho 3-2 é infinito, já que não há aresta direta. O custo do caminho 3-1-2 é a soma dos caminhos 3-1 e 1-2, ou seja, $4+8 = 12$



$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

$K=0$ (distâncias diretas)

- O mesmo é feito para todos os pares de vértices (i,j) , e para cada par em que a distância for reduzida passando pelo vértice 1, a distância é atualizada na matriz
- Para cada valor de k calcula-se, para cada par de vértices (i,j) se o custo do caminho $i \rightarrow k + k \rightarrow j$ é menor do que o custo do caminho $i \rightarrow j$ já calculado. Se for, a matriz é atualizada.
- A linha e coluna hachurada em azul em cada uma das matrizes abaixo representam as posições (i,k) e (k,j) para valores de $k=1,2$ e 3 . Em cada uma delas, para cada posição (i,j) fora da área hachurada, se o valor na posição (i,j) for maior do que a soma de $(i,k) + (k,j)$, a posição (i,j) é atualizada com $(i,k) + (k,j)$.

$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

$$d^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}$$

$$d^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

```
for (i=0;i<N;\ i++)
    for (j=0;j<N;j++)
        se existe a aresta (i,j)
            dist[i][j] = comprimento da aresta de i a j
        senao
            dist[i][j] = INFINITY
    for (k=0;k<N;k++)
        for (i=0;i<N;i++)
            for (j=0;j<N;j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Índice

- Se for necessário identificar o caminho mínimo, isso pode ser feito com uma pequena alteração no algoritmo. Pode-se usar uma outra matriz para armazenar, em cada posição $[i,j]$, qual o último vértice visitado antes do vértice j .
- A lâmina a seguir mostra a modificação no algoritmo. A matriz $p[i,j]$ guarda o último vértice visitado antes do vértice j no caminho de i a j :

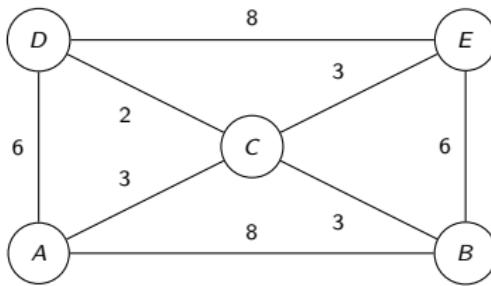
```

for ( i=0; i<N; i++)
    for ( j=0; j<N; j++)
        se existe a aresta (i, j)
            dist[i][j] = comprimento da aresta de i a j
            p[i][j]=i;
        senao
            dist[i][j] = INFINITY
for (k=0; k<N; k++)
    for ( i=0; i<N; i++)
        for (j=0; j<N; j++)
            if (dist[i][j]>dist[i][k] + dist[k][j])
                dist[i][j] = dist[i][k] + dist[k][j]
                p[i][j]=p[k][j]

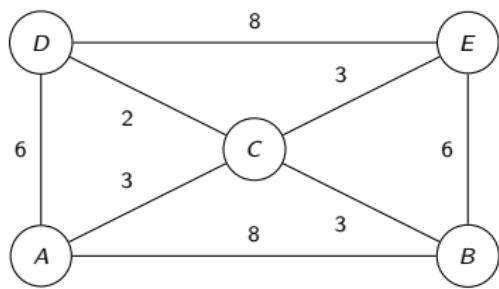
```

Índice

- Aplique o algoritmo de Floyd-Warshall no grafo a seguir, mostrando a matriz de distâncias a cada iteração.



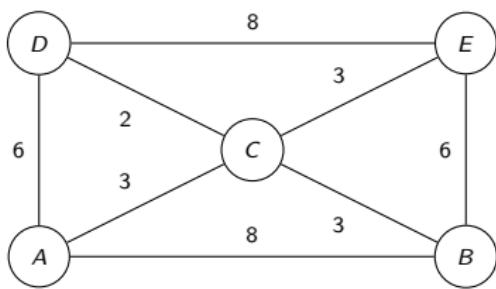
Índice



	A	B	C	D	E
A		8	3	6	∞
B			3	∞	6
C				2	3
D					8
E					

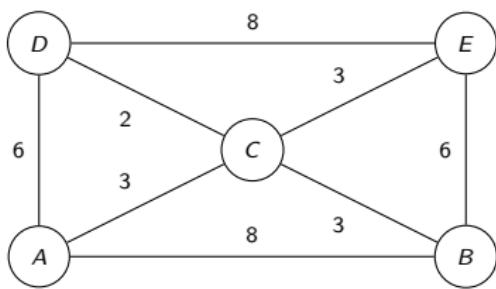
Figura: Inicialização

Índice



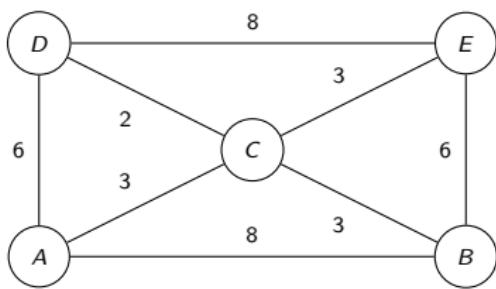
	A	B	C	D	E
A		8	3	6	∞
B			3	∞	6
C				2	3
D					8
E					

Figura: $k = A \cdot BD > BA + AD$



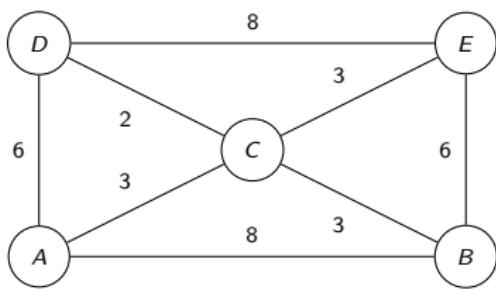
	A	B	C	D	E
A		8	3	6	∞
B			3	14	6
C				2	3
D					8
E					

Figura: $k = A \cdot BD > BA + AD$



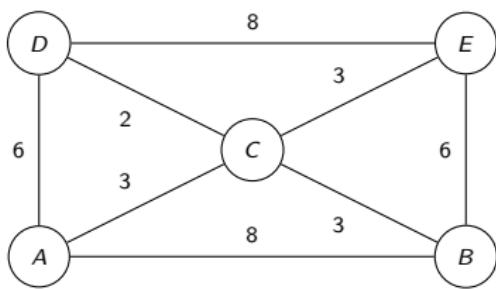
	A	B	C	D	E
A		8	3	6	∞
B			3	14	6
C				2	3
D					8
E					

Figura: $k = B$ $AE > AB + BE$



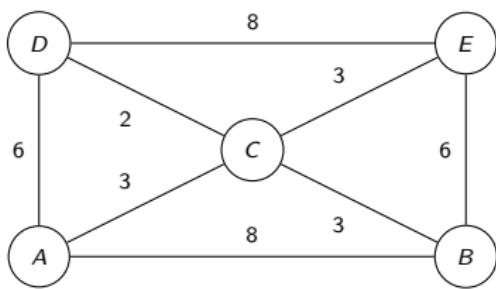
	A	B	C	D	E
A		8	3	6	14
B			3	14	6
C				2	3
D					8
E					

Figura: $k = B \cdot AE > AB + BE$



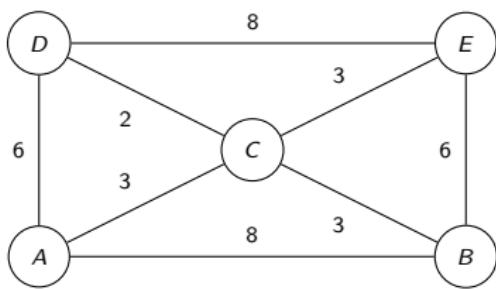
	A	B	C	D	E
A		8	3	6	14
B			3	14	6
C				2	3
D					8
E					

Figura: $k = C \ AB > AC + CB$



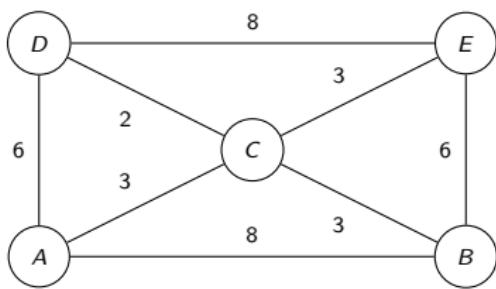
	A	B	C	D	E
A		6	3	6	14
B			3	14	6
C				2	3
D					8
E					

Figura: $k = C \ AB > AC + CB$



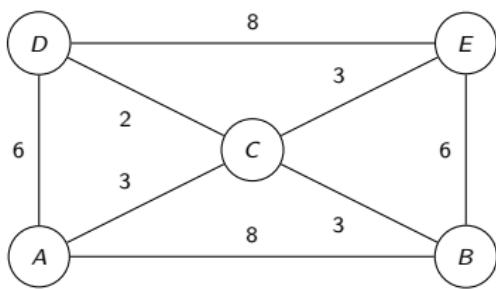
	A	B	C	D	E
A		6	3	6	14
B			3	14	6
C				2	3
D					8
E					

Figura: $k = C \cdot AD > AC + CD$



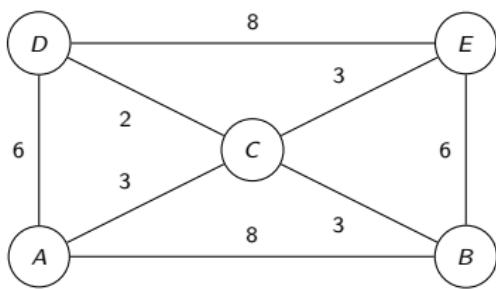
	A	B	C	D	E
A	6	3	5	6	
B		3	14	6	
C			2	3	
D				8	
E					

Figura: $k = C \cdot AD > AC + CD$



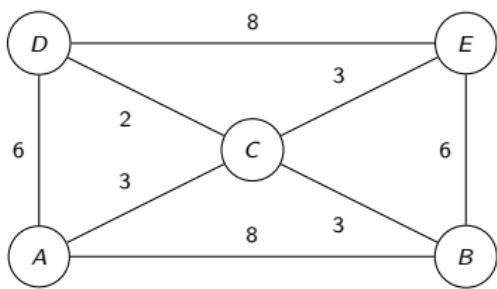
	A	B	C	D	E
A		6	3	5	14
B			3	14	6
C				2	3
D					8
E					

Figura: $k = C \ AE > AC + CE$



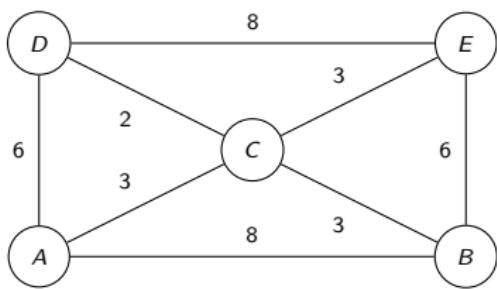
	A	B	C	D	E
A	6	3	5	6	
B		3	14	6	
C			2	3	
D				8	
E					

Figura: $k = C \ AE > AC + CE$



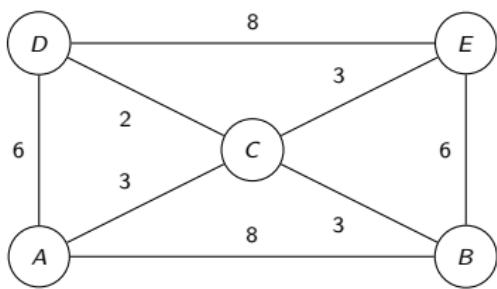
	A	B	C	D	E
A	6	3	5	6	
B		3	14	6	
C			2	3	
D				8	
E					

Figura: $k = C \cdot BD > BC + CD$



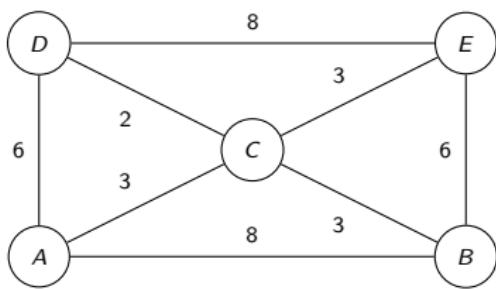
	A	B	C	D	E
A		6	3	5	6
B			3	5	6
C				2	3
D					8
E					

Figura: $k = C \cdot BD > BC + CD$



	A	B	C	D	E
A		6	3	5	6
B			3	5	6
C				2	3
D					8
E					

Figura: $k = C \ DE > DC + CE$



	A	B	C	D	E
A		6	3	5	6
B			3	5	6
C				2	3
D					5
E					

Figura: $k = C \ DE > DC + CE$

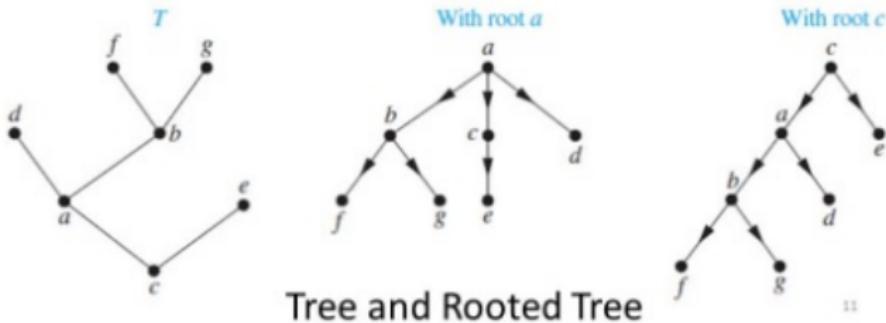
- Floyd-Warshall no URI Online Judge

- 2676 - Cidade no Centro
- 2768 - Grafo do Dabriel
- 1539 - Empresa de Telecom

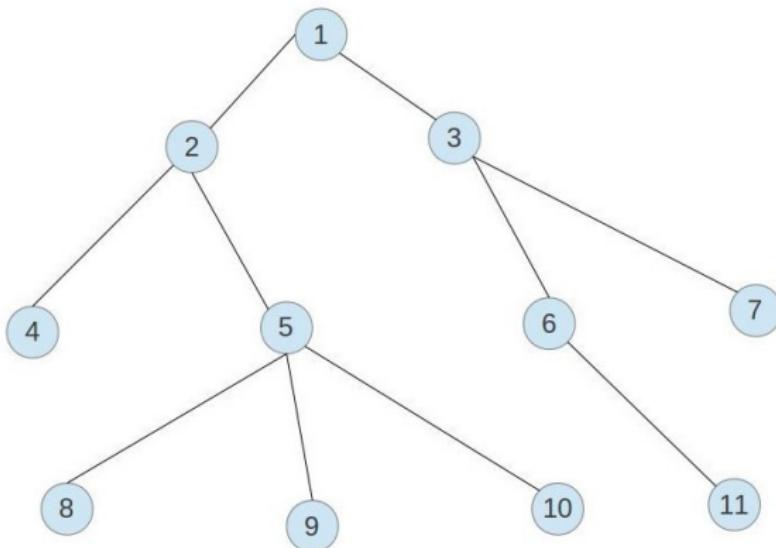
Distância entre vértices por LCA (Lowest Common Ancestor)

- Se o grafo considerado é uma árvore (grafo conexo acíclico), a distância entre dois vértices quaisquer pode ser encontrada encontrando o LCA (lowest common ancestor), ancestral comum mais baixo.
- Em primeiro lugar, é gerada uma árvore enraizada a partir do grafo, isso é, define-se um vértice como a raiz da árvore, e calcula-se para cada vértice o seu nível e quem é o "pai" do vértice.
- Isso pode ser feito através de uma busca (DFS ou BFS) a partir de qualquer vértice.

[Voltar para o Algoritmos de Menor Distância](#)



- Na árvore enraizada mais à direita, **c**, **b** e **a** são ancestrais comuns de **f** e **g**, mas o LCA (lowest common ancestor) é o vértice **b**.



Enraizamento de uma árvore

- Os códigos a seguir geram, a partir de uma lista de adjacências de um conjunto de vértices iniciados em 0, três vetores:
 - $\text{pai}[v]$: o vértice "pai" de cada vértice. Identifica-se a raiz como sendo o vértice que é seu próprio pai (ou algum valor como -1).
 - $\text{nível}[v]$: a raiz é considerada como nível 0.
 - $\text{dpai}[v]$: o valor da aresta de v ao seu pai, caso o grafo seja valorado.
 - O primeiro código implementa uma BFS e o segundo uma DFS.

```

int PF=1,TF=0;
fila [0]=0;
pai[0]=0;
nivel[0]=0;
while (PF!=TF)
{
    int v=fila [TF++];
    tnode *aux=lv [v];
    while (aux)
    {
        int vaux=aux->v;
        if (pai[vaux]==-1)
        {
            pai[vaux]=v;
            nivel[vaux]=nivel[v]+1;
            dpai[vaux]=aux->valor;
            fila [PF++]=vaux;
        }
        aux=aux->prox;
    }
}

```

```
// a raiz será o vértice passado na primeira chamada
// dfs ( 0, 0 ); //o vértice 0, raiz, será o de nível 0

void dfs(int v, int Niv)
{
    nível[v]=Niv;
    tadj *aux=ladj[v];
    while (aux)
    {
        int vaux=aux->v;
        if (nível[vaux]==-1)
        {
            pai[vaux]=v;
            dpai[vaux]=aux->valor;
            dfs(vaux, Niv+1);
        }
        aux=aux->prox;
    }
}
```

- A partir das estruturas geradas pode-se encontrar a distância entre qualquer par de vértices em um tempo igual ao número de arestas que os separam.
- Também há algoritmos baseados em programação dinâmica para encontrar as distâncias em tempo $O(\log n)$, $O(\sqrt{n})$ e $O(1)$
- A função abaixo encontra a distância entre os vértices S e V em tempo N.
- (URI 1135 - Colônia de Formigas)

```
long long int dist(int S,int V)
{
    long long int d=0;
    while (S!=V)
    {
        if (nivel[S]>nivel[V]){
            d+=dpai[S];
            S=pai[S];}
        else if (nivel[S]<nivel[V]){
            d+=dpai[V];
            V=pai[V];}
        else {
            d=d+dpai[V]+dpai[S];
            S=pai[S];
            V=pai[V];}
    }
    return d;
}
```

Quantidade de caminhos entre pares de vértices

- **Teorema:** Seja G um dígrafo (possivelmente com laços) com vértices v_1, \dots, v_n . Seja M a matriz de adjacências de G . Então M_{ij}^k é igual ao número de caminhos de comprimento k de v_i a v_j .
- Produto Matricial: $PROD[p, r] = M[p, q] \times N[q, r]$:
for (i=0;i<p;i++)
 for (j=0;j<r;j++)
 {
 soma=0;
 for (k=0;k<q;k++)
 soma=soma+M[i,k]*N[k,j];
 PROD[i,j]=soma;
 }

Prova: Usamos indução em k . A hipótese de indução é que M_{ij}^k é igual ao número de caminhos de comprimento k de v_i a v_j para todo i, j .

A base de indução é a definição da matriz de adjacências,

$$MA_{i,j}^1 = \begin{cases} 1 & \text{se } i \text{ é adjacente a } j \\ 0 & \text{caso contrário} \end{cases}$$

Supondo que a hipótese valha para algum $k \geq 0$. Provamos que vale para $k + 1$. Cada caminho de comprimento $k + 1$ de v_i para v_j consiste de um caminho de comprimento k de v_i para algum vértice intermediário v_m seguido de uma aresta (v_m, v_j) . Assim, o número de caminhos de comprimento $k + 1$ de v_i a v_j é igual a:

$$M_{iv_1}^k M_{v_1j} + M_{iv_2}^k M_{v_2j} + \dots + M_{iv_n}^k M_{v_nj}$$

Que é precisamente o valor de M_{ij}^{k+1} , e a hipótese vale também para $k + 1$.

Pergunta

Como essa estrutura pode ser utilizada para obter, além da contagem dos caminhos, os próprios caminhos?

Exercício

Faça um programa/algoritmo que leia uma matriz de adjacências representando um conjunto de 10 cidades e escreva ao final o número de caminhos diferentes entre cada par de cidades.

Problema 125 da ACM - Numbering Paths

■ O Problema

- Dadas as esquinas de uma cidade, você deve escrever um programa que determina o número de rotas diferentes entre cada par de esquinas. Uma rota é uma seqüência de ruas que liga duas esquinas.
- As esquinas são identificadas por inteiros não-negativos. Uma rua de mão única é especificada por um par de esquinas. Por exemplo, $j \ k$ indica que há uma rua da esquina j até a esquina k . Note-se que ruas de mão dupla podem ser modeladas pela especificação de duas ruas de mão única.

- Considere-se uma cidade de quatro esquinas (0 a 3) ligadas por ruas da seguinte forma:

0 1

0 2

1 2

2 3

- Há uma rota da esquina 0 à esquina 1, duas rotas da 0 à 2 (as rotas são 0-2 e 0-1-2) duas rotas de 0 a 3, uma rota de 1 a 2, uma rota de 1 a 3, uma rota de 2 a 3, e não há outras rotas.

É possível que um número infinito de rotas diferentes possa existir para um dado par de cidades. Por exemplo, se ao conjunto de ruas acima é acrescentada a rua $3 \rightarrow 2$, ainda há apenas uma rota $0-1$, mas há um número infinito de diferentes rotas $0-2$. Isso ocorre porque o caminho $2-3-2$ pode ser repetido gerando uma seqüência diferente de ruas e, portanto, uma rota diferente. Assim, a rota $0-2$ é uma rota diferente $0-2-3-2$.

A Entrada

A entrada é uma seqüência de especificações de cidades. Cada especificação começa com o número de ruas de mão única na cidade, seguido da especificação de cada rua, dada pela esquina inicial e esquina final. Cada par $j \ k$ representa uma rua de mão única de j a k . Em todas as cidades, as esquinas são numerados seqüencialmente de 0 a n . Todos os números inteiros na entrada são separados por um espaço em branco. A entrada é terminada por end-of-file.

Nunca haverá uma rua de mão única saindo e chegando na mesma esquina. Nenhuma cidade terá mais do que 30 esquinas.

A Saída

Para cada especificação de cidade, uma matriz quadrada com o número de rotas diferentes de j para k deve ser escrita. Se a matriz é chamada M , então $M[j][k]$ é o número de rotas diferentes de j para k . A matriz M deve ser escrita por linhas, cada linha da matriz escrita em uma linha diferente. Cada matriz deve ser precedida pela string "matrix for city" k (com k começando por 0). Se houver um número infinito de diferentes caminhos entre dois cruzamentos um -1 deverá ser escrito. Não se preocupe em justificar e alinhar a saída de cada matriz. Todas as entradas em uma linha devem ser separados por espaços em branco.

Exemplo de entrada

7 0 1 0 2 0 4 2 4 2 3 3 1 4 3

Saída correspondente

matrix for city 0

0 4 1 3 2

0 0 0 0 0

0 2 0 2 1

0 1 0 0 0

0 1 0 1 0

Exemplo de entrada

5

0 2

0 1 1 5 2 5 2 1

Saída correspondente

matrix for city 1

0 2 1 0 0 3

0 0 0 0 0 1

0 1 0 0 0 2

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

Exemplo de entrada

9

0 1 0 2 0 3

0 4 1 4 2 1

2 0

3 0

3 1 Saída correspondente

matrix for city 2

-1 -1 -1 -1 -1

0 0 0 0 1

-1 -1 -1 -1 -1

-1 -1 -1 -1 -1

0 0 0 0 0

Problema 104 da ACM - Arbitrage

■ O Problema

- *Arbitrage* é a troca de uma moeda por outra com o objetivo de tirar vantagem das pequenas diferenças nas taxas de conversão entre as diversas moedas de modo a obter lucro. Por exemplo, se 1 dólar americano compra 0.7 libras, uma libra compra 9.5 francos fanceses e 1 franco francês compra 0.16 dólares americanos, então um comerciante de moeda pode começar com 1 dólar e ganhar $1 \times 0.7 \times 9.5 \times 0.16 = 1.064$ dólares, com um lucro de 6.4%.
- Você escreverá um programa que determina se há uma sequência de trocas que leva a algum lucro, como descrito acima.
- Para resultar em uma *arbitrage* bem sucedida, a sequência de trocas deve começar e terminar com a mesma moeda, e pode iniciar por qualquer moeda.

■ A entrada

- O arquivo de entrada consiste de uma ou mais tabelas de conversão. Você deve resolver o problema de arbitrage para cada uma das tabelas do arquivo de entrada.
- Cada tabela é precedida de um inteiro n em uma linha significando a dimensão da tabela. A dimensão máxima é 20; a dimensão mínima é 2.
- A tabela então é apresentada em linhas, sem o elemento da diagonal principal (que assume-se igual a 1.0). Assim, a primeira linha da tabela representa as taxas de conversão entre o país 1 e os outros $N - 1$ países, i.e., a quantidade de moeda do país i ($2 \leq i \leq N$), que pode ser comprada com uma unidade da moeda do país 1.
- Assim, cada tabela consiste de $N + 1$ linhas no arquivo de entrada, 1 linha contendo a dimensão N da tabela e N linhas representando a tabela.

A saída

Para cada tabela do arquivo de entrada você deve determinar se existe uma sequência de trocas que resulte em um lucro de mais de 1% (0.01). Se a sequência existe você deve escrever a sequência de trocas que resultam em lucro. Se há mais de uma sequência que resulte em lucro de mais de 1 por cento, você deve escrever a sequência de comprimento mínimo, i.e., uma das sequências que usa a menor quantidade de trocas que gere lucro.

Como o IRS (United States Internal Revenue Service) percebe sequências compridas, todas as sequências devem consistir de N ou menos transações, onde N é a dimensão da tabela. A sequência 1 2 1 representa duas conversões.

Se uma sequência que dê lucro existe, você deve escrever a sequência de trocas que resulta em lucro. A sequência é escrita como uma sequência de inteiros em que o inteiro i representa o país i . O primeiro inteiro da sequência é o país de onde inicia a sequência lucrativa. Este inteiro também deve terminar a sequência.

Se não existe sequência lucrativa com N ou menos transações, então a linha

no arbitrage sequence exists
deve ser escrita.

Exemplo de entrada

3

1.2 .89

.88 5.1

1.1 0.15

4

3.1 0.0023 0.35

0.21 0.00353 8.13

200 180.559 10.339

2.11 0.089 0.06111

2

2.0

0.45

Exemplo de saída

1 2 1

1 2 4 1

no arbitrage sequence exists

O Portal de Problemas da ACM

- O portal de problemas da ACM (Association for Computing Machinery) é um repositório de problemas de programação armazenados na Universidade de Valladolid, na Espanha, e é utilizado em competições internacionais de programação.
- A URL é <https://uva.onlinejudge.org/>
- O portal contem, além do conjunto de problemas, um mecanismo de validação automática das soluções submetidas.
- Ele fornece diversas estatísticas como quantidade de problemas diferentes resolvidos por cada usuário, e os tempos das melhores soluções para cada problema

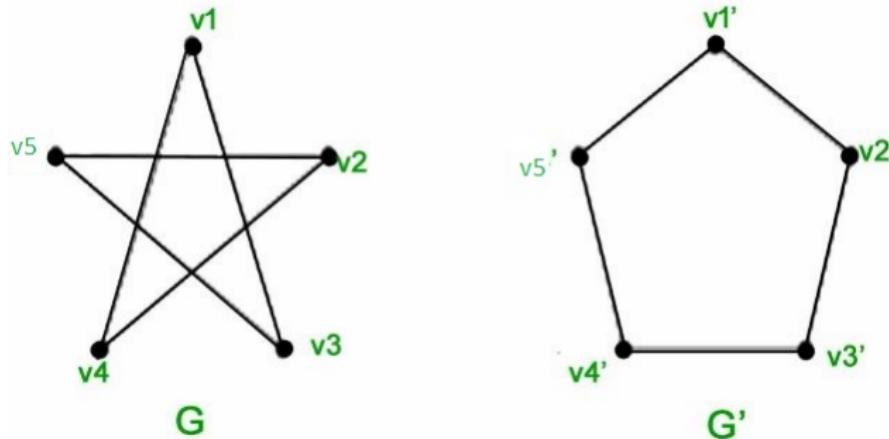
- Os programas podem ser submetidos em C, C++ ou Java.
- Após a execução o sistema envia por email o resultado da submissão que pode ser:
 -

- O UVA Online Judge é um juiz automatizado on-line para problemas de programação, alojadas pela Universidade de Valladolid, na Espanha.
- Seu acervo conta com mais de 4300 problemas e o sistema é aberto a todos.
- Um usuário pode apresentar uma solução em ANSI C (C89), C ++ (C ++ 98), Pascal , Java ou C ++ 11 .

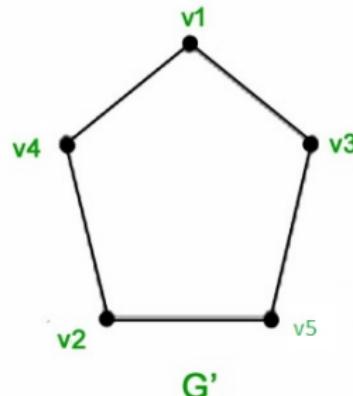
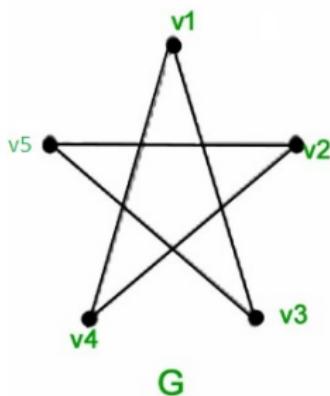
- Originalmente ele começou sem as duas últimas opções, mas a opção Java foi adicionada em 2001 e a opção C ++ 11 foi adicionada em 2014.
- O UVA é utilizado em competições internacionais de programação.
- No ambiente de competição o usuário tem um tempo limitado para resolver um pequeno conjunto de problemas.
- O UVA foi criado em 1995 por Miguel Ángel Revilla, matemático professor de algoritmos Universidade de Valladolid.

Isomorfismo

- **Def:** Dois grafos $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ são ditos **isomorfos** se possuem a mesma estrutura, diferindo apenas nos nomes dos vértices.
- Se é possível transformar um grafo no outro, apenas renomeando os vértices, então eles são isomorfos.
- Os 2 grafos abaixo são isomorfos?

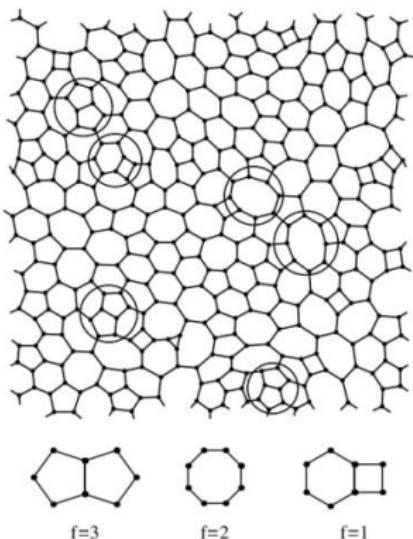


- Os grafos anteriores são isomorfos. Apesar de G estar representado como uma estrela e G' como um pentágono, não é a representação gráfica que define um grafo, mas sua estrutura. As duas figuras abaixo representam o mesmo grafo, apesar das duas representações gráficas diferentes.
- Ambos contem os vértices $\{v_1, v_2, v_3, v_4, v_5\}$ e as arestas $\{(v_1, v_3), (v_3, v_5), (v_5, v_2), (v_2, v_4), (v_4, v_1)\}$



- Aplicações de isomorfismo incluem a identificação de compostos químicos. Vértices representam átomos e as arestas as ligações entre eles. A identificação do composto é feita pela verificação de isomorfismo entre o composto analisado e os compostos em uma base de dados.
- Outra aplicação de isomorfismo é na eletrônica, em que os vértices representam componentes eletrônicos e as arestas representam as ligações entre eles, e usa-se isomorfismo para verificar se um circuito integrado contém como parte dele um determinado circuito.
- Verificação de isomorfismo é utilizado até na verificação de similaridade de programas. Modela-se a estrutura dos programas como um grafo e analisa-se a similaridade entre ambos.

- Em seu TCC, Henrique Scalon utilizou isomorfismo para analisar características de compostos de carbono, procurando a ocorrência de alguns padrões em redes de carbono.



- Não há algoritmos eficientes para verificar se dois grafos são isomorfos.
- Uma possibilidade é testar todos os mapeamentos possíveis entre vértices e verificar se algum mapeamento transforma um grafo no outro (custo?)
- O número de mapeamentos para um grafo de n vértices é da ordem de fatorial de n ($n!$), o que torna a abordagem impraticável mesmo para grafos pequenos.

[Voltar para o índice](#)

[Exercícios de isomorfismo](#)

- Para mostrar que 2 grafos não são isomorfos pode-se procurar alguma diferença estrutural que mostre que tal mapeamento não é possível.
- Isomorfismo encontra muitas aplicações na químioinformática (identificação de moléculas semelhantes) e projeto de circuitos.
- É um dos raros problemas em teoria de grafos para o qual ainda não se provou ser P ou NP-Completo.

[Voltar para o índice](#)

[Exercícios de isomorfismo](#)

Isomorfismo

Def: Dois grafos $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ são **isomorfos** se:

- 1 $|V_1| = |V_2|$ (possuem o mesmo número de vértices)
- 2 $\exists f : V_1 \rightarrow V_2$ bijetora, tal que se $(v, w) \in E_1$ então $(f(v), f(w)) \in E_2, \forall v, w \in V_1$

Ex:

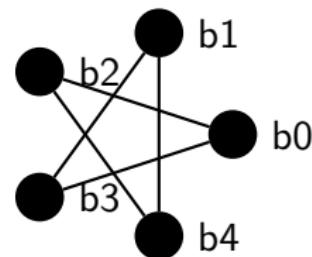
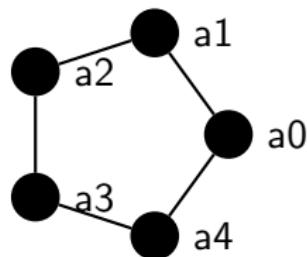


Figura: Grafos isomorfos

Onde pode-se definir $f : V_1 \rightarrow V_2$ da seguinte maneira:

$$F(v) = v', \forall v \in V_1$$

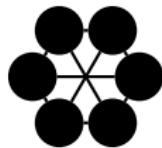


Figura: Grafo 1

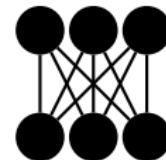


Figura: Grafo 2

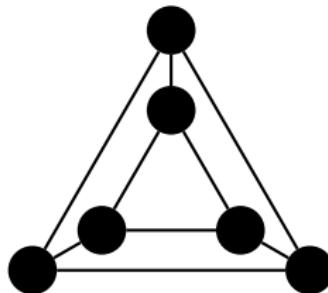
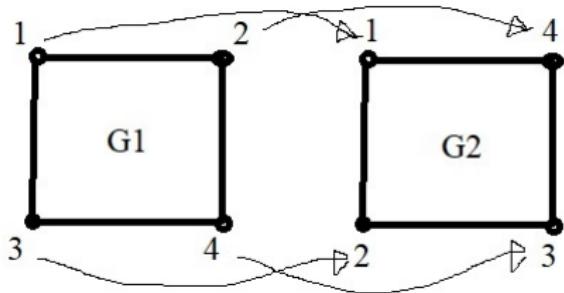


Figura: Grafo 3

Quais dos grafos acima são isomorfos?

[Voltar para o índice](#)

- A função de mapeamento de um grafo **G1** para um grafo **G2** pode ser implementada por um vetor r que associa a cada vértice de G1 o número do vértice o vértice correspondente em G2.
- Os dois grafos abaixo são isomorfos, e o mapeamento do primeiro para o segundo pode ser representado pelo vetor $r=[1,4,2,3]$



- Tendo a função de mapeamento, basta verificar se os dois grafos são iguais. Para cada par de vértices (i,j) de G_1 , o par $(r[i],r[j])$ em G_2 deve ter a mesma relação de adjacência.

```
iso=1;  
for (i=0;i<N;i++)  
    for (j=0;j<N;j++)  
        if (G1[i][j]!=G2[r[i]][r[j]]) iso=0;
```

O código a seguir escreve todas as permutações de um vetor.
Como esse código pode ser utilizado para verificar se dois grafos
são isomorfos?

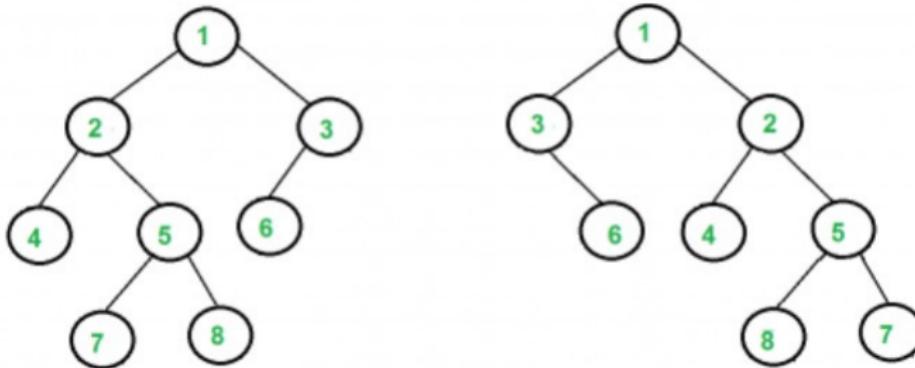
```
#include <stdio.h>
#include <conio.h>
#define TAM 4

int perm[TAM];

void esc_perm(int pos, int N)
{
    int i;
    if (pos==N)
        {for (i=0;i<=N; i++) printf("%d ",perm[i]); printf("\n"); return;}
    for (i=pos; i<=N; i++)
    {
        int aux=perm[i]; perm[i]=perm[pos]; perm[pos]=aux;
        esc_perm(pos+1,N);
        aux=perm[i]; perm[i]=perm[pos]; perm[pos]=aux;
    }
}
int main()
{
    int i;
    for (i=0;i<TAM; i++) perm[i]=i; // preenche o vetor com os números de 0 a N
    esc_perm(0,TAM-1);
    getch();
}
```

Isomorfismo de Árvores

- Para algumas categorias de grafos, o problema de identificar isomorfismo pode ser resolvido em tempo polinomial.
- A identificação de isomorfismo entre árvores pode ser feita em tempo $O(n)$
- As duas árvores enraizadas binárias abaixo são isomorfas?

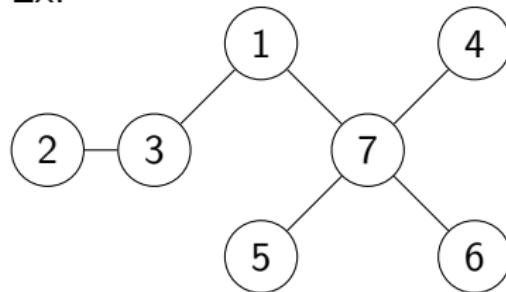


- A verificação de isomorfismo de duas árvores binárias **T1** e **T2** enraizadas pode ser feita verificando recursivamente o isomorfismo dos filhos de **T1** e **T2**.
- Se a árvore não é enraizada, transforma-se a árvore em uma árvore enraizada.
 - Como escolher o vértice raiz? Há algum vértice em uma árvore que o diferencie dos outros, para que nas duas árvores seja escolhido o mesmo vértice?
 - E se a árvore não é binária?
 - Nesse caso pode-se verificar todas as combinações de subárvores. Ou ordená-las de alguma forma.

- **Def:** Uma **árvore enraizada** (rooted tree), é uma árvore que possui um vértice especial escolhido como raiz. Estabelece-se uma hierarquia de ascendentes e descendentes, como pais, filhos e irmãos. O vértice que não possui pai é a raiz.
- **Obs:** Uma árvore livre torna-se uma árvore enraizada dirigida se:
 - 1 Escolhe-se um vértice como raiz
 - 2 Orienta-se cada vértice a partir da raiz
 - 3 Há apenas um caminho da raiz a qualquer outro vértice
- Aplicam-se os conceitos de árvore binária, n-ária, ordenada, balanceada, profundidade, etc.

Def: Se um vértice v de uma árvore T possuir $\text{grau}=1$ então v é uma **folha**, caso contrário é um **vértice interior**.

Ex:



$$\text{Folhas} = \{2, 4, 5, 6\}$$

$$\text{Vértices interiores} = \{1, 3, 7\}$$

Obs₁: Toda árvore T com n vértices possui exatamente $n - 1$ arestas.

Obs₂: O número de folhas varia entre um número mínimo de 2 e um número máximo de $n - 1$ ($n = \text{número de vértices}$) para $n > 2$.

- Denomina-se **excentricidade** $e(v)$ de $v \in V$ ao valor da distância máxima entre v e w , para todo $w \in V$. A figura abaixo mostra a excentricidade para cada vértice do grafo.

Ex:

vértice	Excent.
a	3
b	3
c	2
d	2
e	2
f	3
g	3

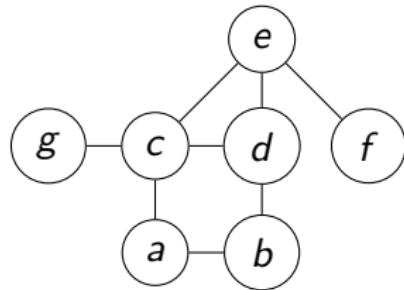
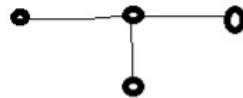


Tabela: Excentricidade

- O **raio** de G é a excentricidade mínima dos pontos. A máxima excentricidade é o **diâmetro**.
- O diâmetro pode ser obtido calculando-se as distâncias entre todos os pares de vértices (ex. com Floyd-Warshall). O diâmetro é a distância entre os dois pontos mais afastados.

- Se o grafo é uma árvore, o diâmetro pode ser obtido efetuando-se uma busca em amplitude (BFS) a partir de um vértice qualquer e guardando-se o último vértice atingido (**u**). Efetua-se uma nova BFS a partir de **u** guardando-se o último vértice atingido **v**. O diâmetro do grafo é a distância entre **u** e **v**.
- Pode-se provar que o último vértice atingido na primeira BFS faz parte do caminho mais longo na árvore.

- Um **ponto central** é um vértice v para o qual $e(v) = r(G)$, e o **centro** de G é o conjunto de pontos centrais.
- Se o grafo é uma árvore, o ponto (ou pontos) central, pode ser encontrado removendo, a cada passo, todas as folhas da árvore.
- Ou calculando o caminho mais longo. O centro é o ponto do meio do caminho. Ou os dois pontos, se o comprimento for par.
- Uma árvore pode conter um ou dois pontos centrais.

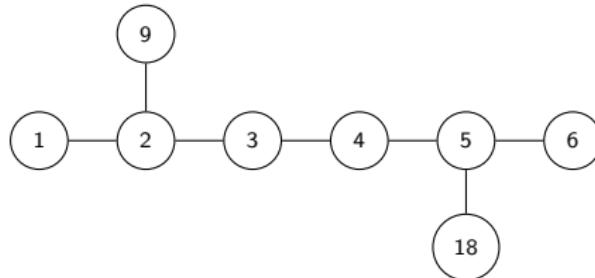


1 ponto central

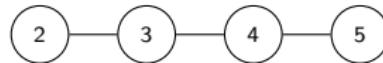


2 pontos centrais

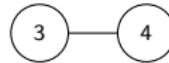
Teorema Toda árvore tem um centro consistindo de um ou dois pontos adjacentes, este centro pode ser encontrado por um processo iterativo onde a cada iteração são removidas todas as folhas gerando um novo grafo. O processo termina quando o número de vértices for igual a 1 ou 2. Estes vértices são o centro da árvore. No exemplo anterior, as iterações seriam as seguintes:
Passo 1:



Passo 2:

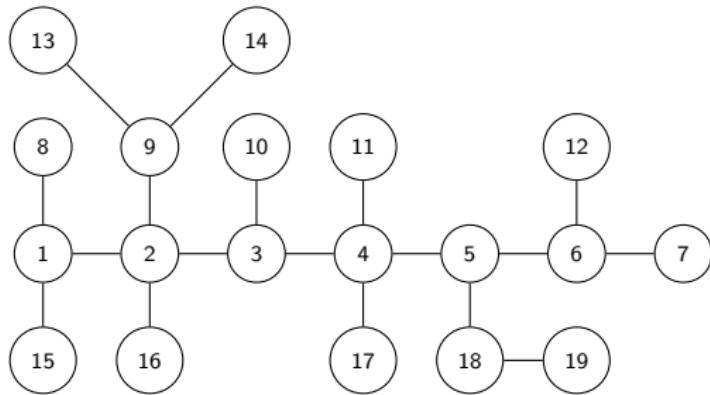


Passo 3:



Logo, o centro da árvore são os vértices 3 e 4.

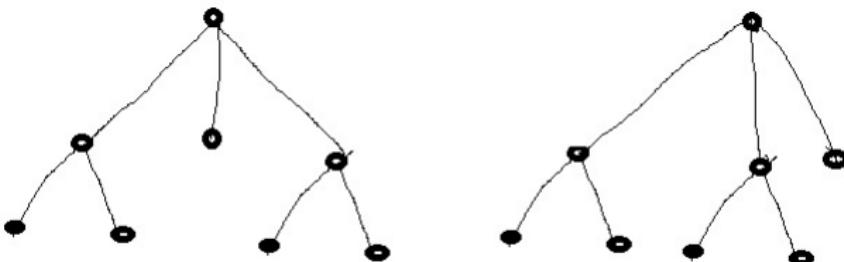
Ex: Para o subgrafo a seguir calcule:



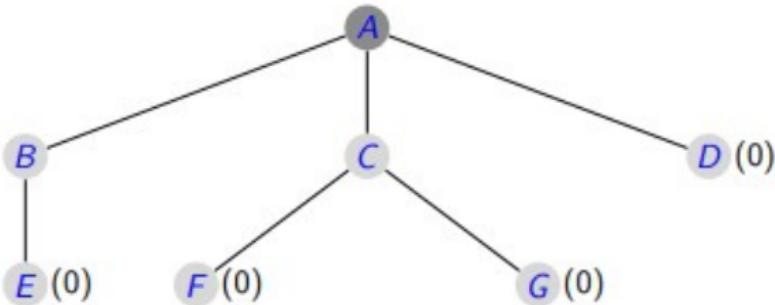
- 1 A excentricidade de cada ponto
- 2 O diâmetro da árvore : 7
- 3 O raio da árvore : 4
- 4 O centro (dois pontos de $e(v) = 4$)

- Voltando ao problema de isomorfismo de árvores, se as árvores são não enraizadas pode-se enraizá-las escolhendo o ponto central como raiz, transformando o problema no problema de isomorfismo de árvores enraizadas.
- Se as árvores tem dois pontos centrais, testa-se o isomorfismo selecionando como raiz cada um dos dois pontos centrais.

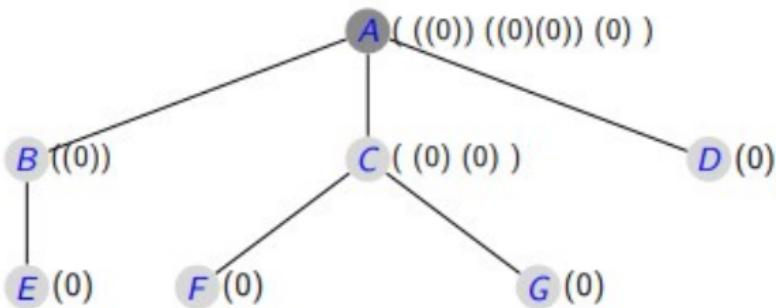
- Se a árvore não é binária, pode-se testar todas as combinações de subárvores de G1 com todas as combinações de subárvores de G2. É muita combinação.
- Ou pode-se ordenar as subárvores transformando a árvore em uma árvore canônica, ou seja, uma representação única, comum a todas as árvores de uma família de árvores isomorfas e somente a ela.
- As árvores abaixo são isomorfas?



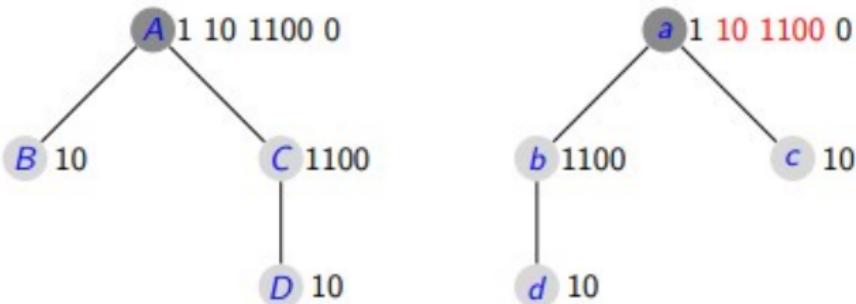
- O algoritmo de Aho, Hopcroft e Ullman associa a cada vértice da árvore uma tupla que registra a estrutura completa da sub-árvore enraizada por ela.
- A cada folha é associada a tupla "(0)"



- E a cada nodo interno é associada uma tupla contendo a concatenação das tuplas dos filhos.



- O próximo passo é substituir as tuplas de parênteses por nomes, eliminando os 0's das tuplas e representando os ('s por 1's e o ')s por 0's.
 - Se a cada vértice as tuplas dos filhos forem ordenadas antes de serem concatenadas, todas as árvores de uma mesma família de isomorfismo gerarão nomes idênticos, permitindo a identificação de isomorfismo entre duas árvores.



Algoritmo de geração de nome canônico

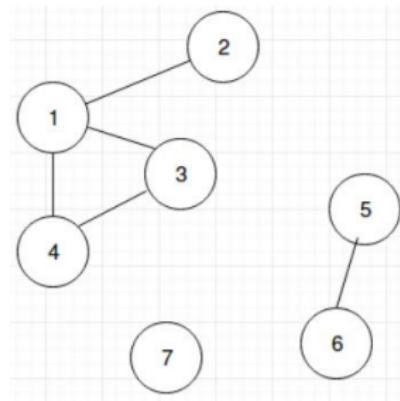
■ Assign-Canonical-Names(v)

- 1 if v é uma folha
- 2 Associe a v o nome "10"
- 3 else
- 4 para cada filho w de v faça
- 5 Assign-Canonical-Names(w)
- 6 fimpara
- 7 Ordene os nomes dos filhos de v
- 8 Concatene os nomes dos filhos de v gerando temp
- 9 Dê ao vértice v o nome "1" temp "0"
- 10 endif

- É a verificação de isomorfismo entre duas árvores T_1 e T_2 , de raízes r_1 e r_2 , pode ser verificada gerando os nomes canônicos para r_1 e r_2 , e comparando-os. Se possuem o mesmo nome canônico, são isomórficas, caso contrário não são.

Componentes Conexas

- **Def:** **Componentes conexos** de um grafo são subgrafos conexos **maximais**, ou seja, que não são subconjuntos de nenhum outro subgrafo conexo. Em outras palavras, correspondem às porções contíguas da sua representação geométrica.
- Quantas componentes conexas tem o grafo abaixo?



Componentes Conexas

- Pode-se identificar as componentes conexas de um grafo efetuando buscas (em profundidade ou amplitude) a partir de cada vértice **v**. Todos os vértices alcançáveis a partir de **v** pertencem à mesma componente conexa que ele.
 - O código a seguir ilustra esse processo:

```
for ( i=0; i<N; i++ ) visitado [ i ] = -1;
ncc = 0; // número de componentes conexas
for ( i=0; i<N; i++ )
if ( visitado [ i ] == -1 )
dfs ( i , ncc++ , G , visitado );
```

- Onde:
 - **N** é o número de vértices do grafo
 - **ncc** conterá, ao final, o número de componentes conexas do grafo
 - O vetor visitado mantém, para cada vértice, o número de sua componente conexa.
 - **dfs** efetua uma busca em profundidade (pode ser utilizada uma busca em amplitude) marcando, para cada vértice alcançado a partir de **i**, o número da componente conexa **ncc** no vetor **visitado**.
- Exercício do URI de Componente Conexa: 1835 - Promessa de Campanha

- **Def:** Um **subgrafo** $G' = (V', E')$ de um grafo $G = (V, E)$ é um grafo tal que
 - 1 $V' \subseteq V$
 - 2 $E' \subseteq E$ ($\text{se } (v, w) \in E' \Rightarrow v, w \in V'$)
- Se E' contém todas as arestas (v, w) de E , e $v, w \in V'$, então G' é denominado **subgrafo induzido** de G
 - 1 $V' \subseteq V$
 - 2 $E' \subseteq E, \forall v, w \in V', \text{ se } (v, w) \in E' \Rightarrow (v, w) \in E'$

- Ex:
 - $G = (V, E)$
 - $V = \{1, 2, 3, 4\}$
 - $E = \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$
- G' é um subgrafo de G , mas não é um subgrafo induzido
 - $G' = (V', E')$
 - $V' = \{1, 2, 4\}$
 - $E' = \{(1, 2), (2, 4)\}$
- G'' é um subgrafo induzido de G
 - $G'' = (V'', E')$
 - $V'' = \{1, 2, 4\}$
 - $E'' = \{(1, 2), (2, 4), (1, 4)\}$

- Faça uma função que receba dois grafos representados por matrizes de adjacências $G[7][7]$ e $G1[7][7]$ e retorne:
 - 1 - Se $G1$ é um subgrafo induzido de G ;
 - 0 - Em caso contrário.
- Considere que os vértices de $G1$ são somente os vértices que tem alguma aresta adjacente (i.e. Os vértices não isolados).

```
int induzido(int G[7][7], int G1[7][7])
{
    int i,j,tem[7];
    for (i=0;i<7;i++)
    {
        // o flag "tem" diz, para cada vértice v, se v está em G1
        tem[i]=0;
        for (j=0;j<7;j++) if (G1[i][j]==1) tem[i]=1;
    }
    for (i=0;i<7;i++)
        if (tem[i]==1)
            for (j=0;j<7;j++)
                if (tem[j]==1)
                    if (G[i][j]!=G1[i][j]) return 0;
    return 1;
}
```

Operações sobre grafos

- **Def:** Seja G e H dois grafos quaisquer. A **soma** de G e H , representada por $G + H$ denota o grafo que contém todas os vértices e arestas de G e H , mais arestas ligando todos os vértices de G a todos os vértices de H .
- **Def: (eliminação de um vértice)** Seja G um grafo qualquer. $G - v$ denota o grafo obtido de G removendo de G o vértice v e todas as arestas adjacentes, mantendo todos os outros vértices.
- **Def:** Seja G um grafo qualquer, $G - [x_1, x_j]$ (ou $G - E$ onde E é a aresta $[x_1, x_j]$) denota o grafo obtido de G removendo a aresta $[x_1, x_j]$. A remoção da aresta não implica na remoção dos vértices.

Subconjuntos maximais e minimais

- **Def:** Seja S um conjunto e S' um subconjunto de S . Diz-se que S' é **maximal** em relação a uma certa propriedade P quando satisfaz esta propriedade P e não é subconjunto próprio de nenhum outro subconjunto que também satisfaça P .
- Obs: S' não precisa necessariamente ser o maior subconjunto que atende à propriedade, basta que não seja subconjunto próprio de outro subconjunto que atenda à propriedade.
- Por exemplo, uma componente conexa pode ser definido como um subgrafo conexo maximal.
- Ou uma árvore geradora pode ser definida como um subgrafo acíclico maximal.

- Da mesma forma, um subconjunto S' é dito **minimal** em relação a uma certa propriedade P quando atende a essa propriedade e não contém propriamente nenhum subconjunto S'' que também atenda à propriedade.

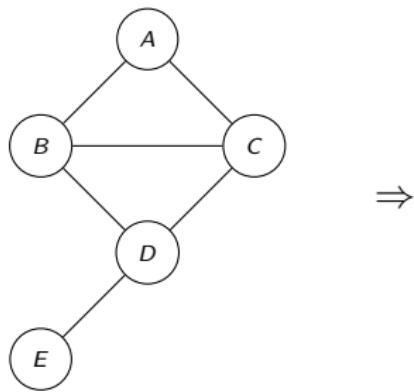
Índice

Conectividade em Grafos

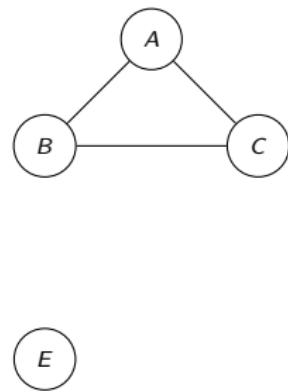
- Em um grafo que representa uma rede de computadores os vértices representam as entidades que se comunicam e as arestas representam os meios de comunicação.
- Para que todas as entidades da rede possam se comunicar entre si é necessário que o grafo seja conexo.
- O estudo da conectividade de grafos permite determinar, por exemplo, a capacidade de tolerância a falhas de uma rede.
 - Por exemplo: Quantos nodos intermediários precisam cair para que a conexão entre dois pontos seja perdida?

[Voltar para o índice](#)

- **Def:** Um vértice em um grafo simples é uma **articulação** (ou **vértice de corte**) se sua remoção (juntamente com suas arestas associadas) torna o grafo resultante desconexo (se ele era conexo) ou aumenta o número de componentes conexas.
- Ex: No grafo a seguir, a remoção do vértice D torna o grafo desconexo.
- Uma forma ineficiente de verificar se um vértice é uma articulação é removê-lo e verificar se o número de componentes conexas aumentou, mas há formas melhores.
- A identificação de articulações é importante para identificar fragilidades em uma rede.



⇒



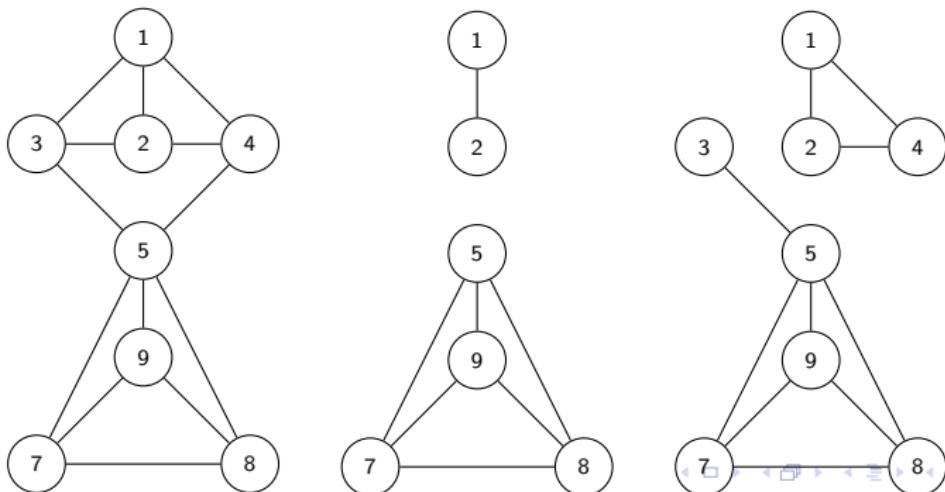
- **Def:** Seja $G(V, E)$ um grafo conexo. Um **corte de vértices** de G é um subconjunto minimal (ou seja, que não tem como subgrafo nenhum corte de G) de vértices $V' \subseteq V$ cuja remoção de G aumenta o número de componentes conexas.
- No grafo anterior, a remoção do vértice B não aumenta o número de componentes conexas, logo ele não é uma articulação.
- Mas a remoção dos vértices B e C separa o vértice A dos vértices D e E, caracterizando o conjunto $\{B,C\}$ como um corte de vértices.

- **Def:** Uma aresta E é chamada de **ponte** se $G - E$ tem mais componentes conexas que G .
- Uma aresta é uma ponte se e somente se ela não faz parte de nenhum ciclo.
 - Se ela faz parte de um ciclo, há pelo menos dois caminhos entre os seus vértices adjacentes, e a remoção da aresta não tornará o grafo desconexo.
 - Se ela não faz parte de um ciclo, é o único caminho entre seus vértices adjacentes e sua remoção tornará o grafo desconexo.

- **Def:** Um **corte de arestas** de G é um conjunto minimal E' de G que quando retirado desconecta G . Para qualquer subconjunto E'' de E' , $G - E''$ é conexo.
- Se G é completo, não existe um subconjunto V' de vértices que desconecta G , a não ser que G se torne trivial. Ex : K_5

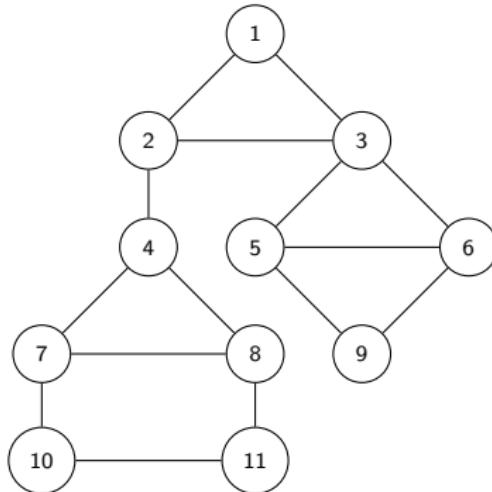
- **Conectividade em vértices** c_v de G é a cardinalidade do menor corte de vértices em G , ou seja é o número mínimo de vértices que devem ser removidos para que o grafo deixe de ser conexo.
- Da mesma forma, **conectividade de arestas** c_E é a cardinalidade do menor corte de arestas de G .
- Se G é desconexo, $c_v = c_E = 0$

No exemplo abaixo, o subconjunto $\{3,4\}$ é um corte de vértices. $\{3,4,7\}$ não é um corte de vértices porque não é minimal. $\{(1,3),(2,3),(4,5)\}$ é um corte de arestas. O corte de vértices de cardinalidade mínima é $\{5\}$ e o corte de arestas de cardinalidade mínima é $\{(3,5),(4,5)\}$, logo, as conectividades de vértices e arestas deste grafo são respectivamente 1 e 2.



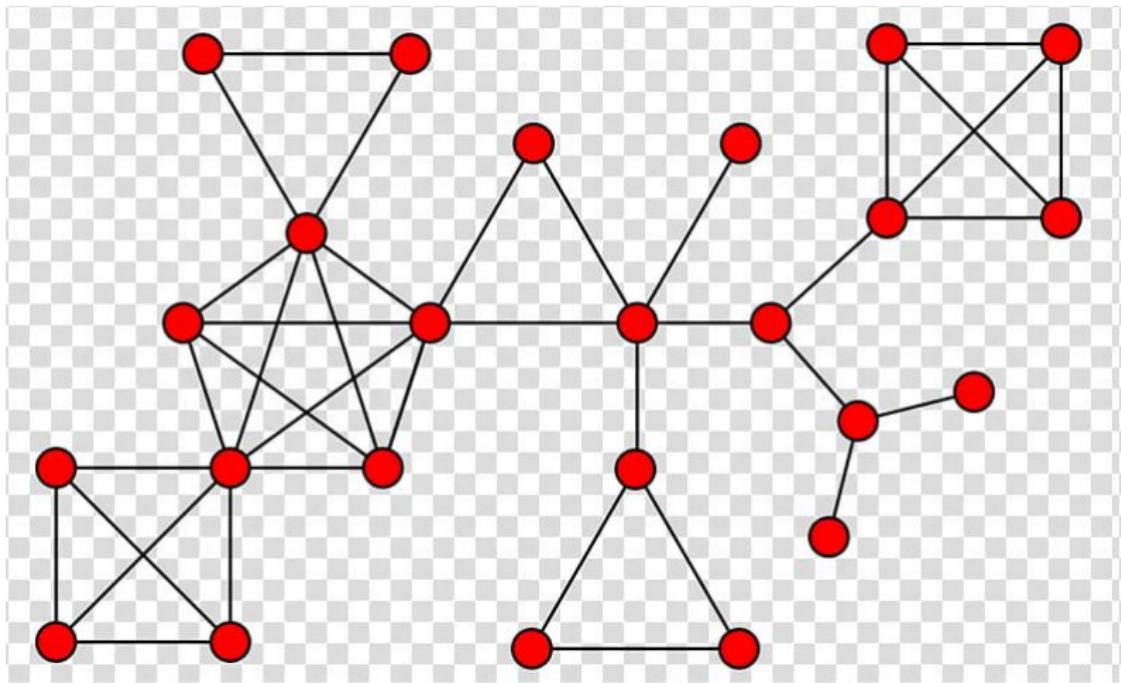
- **Def:** Um grafo é **k -conexo em vértices** quando sua conectividade de vértices é $\geq k$, ou seja, quando é necessário remover pelo menos k vértices para torná-lo desconexo.
- Um grafo é **k -conexo em arestas** quando sua conectividade de arestas é $\geq k$.
- Se G é k -conexo em vértices (arestas) então não existe corte de vértices(arestas) de cardinalidade $\geq k$.
- Um grafo 2-conexo é chamado **biconexo** (ou, de outra forma, um grafo biconexo é um grafo conexo que não tem articulações), ou que é necessário remover pelo menos 2 vértices para torná-lo desconexo.

Encontre as articulações e pontes do grafo a seguir:



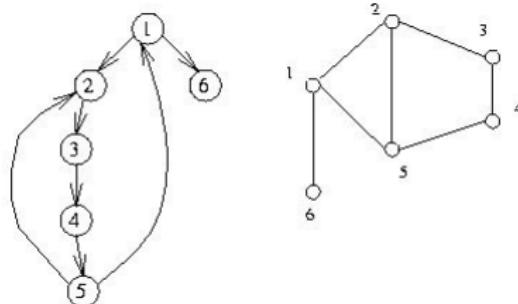
Ponte: $\{(2, 4)\}$ Articulações: $\{2, 3, 4\}$

- **Def:** Denominam-se **componentes biconexas** de G (ou blocos) aos subgrafos que são biconexos em vértices. A separação do grafo em suas componentes biconexas é útil para melhor compreender a estrutura de um grafo.
- A identificação das componentes biconexas de um grafo é importante na análise de tolerância a falhas em redes. Em cada componente biconexa a queda de um dos nós não a torna desconexa.
- Quais as componentes biconexas do grafo a seguir?



Identificação das articulações

- Seja $G(V, E)$ um grafo e $T(V, E_T)$ uma árvore gerada por uma busca em profundidade de G (código na lâmina seguinte).
- Na geração dessa árvore é efetuada uma busca em profundidade no grafo a partir de qualquer vértice. As arestas que forem utilizadas para alcançar algum vértice pela primeira vez farão parte da árvore.
- A figura abaixo à esquerda mostra a árvore gerada para o grafo da direita após uma DFS a partir do vértice 1.



Geração da árvore por DFS

- Globais: matriz de adjacências g.
- = 2 - aresta da árvore
- = 1 - aresta que não faz parte da árvore
- nível = vetor global com o nível de cada vértice. Inicializado com -1
- v - vértice corrente da dfs
- niv - nível do vértice corrente da dfs
- N - quantidade de vértices (numeração inicia em 0)

```
int dfs(int v, int niv, int N)
{
    nível[v]=niv;
    int i;
    for (i=0;i<N;i++)
        if (g[v][i]==1 && nível[i]==-1)
        {
            g[v][i]=2;
            g[i][v]=0;
            dfs(i,niv+1,N);
        }
}
```

- globais: grafo g, vetor nível e vetor low, com lowpt de cada vértice

```
int lowpt(int v, int N)
{
    if (low[v]==-1) return low[v]; // se low[v] já está definido, retorna
    low[v]=v; // valor inicial é o próprio v
    int i;
    for (i=0;i<N;i++)
        if (g[v][i]==2 && nível[lowpt(i,N)]<nível[low[v]])
            low[v]=low[i]; // se lowpt do filho é menor, atualiza lowpt do pai
        else
            if (g[v][i]==1 && nível[i]<nível[low[v]])
                low[v]=i; // se tem aresta de retorno para cima, atualiza lowpt
    return low[v];
}
```

- Chamemos de **arcos da árvore** aos arcos de E_T e **arcos de retorno** aos arcos de $E - E_T$.
- Define-se a função $lowpt : V \rightarrow V$ da seguinte forma: para cada vértice $v \in V$, $lowpt(v)$ é igual ao vértice mais próximo da raiz de T que pode ser alcançado a partir de v , caminhando-se em T para baixo através de zero ou mais arestas de árvore e, em seguida, para cima, utilizando no máximo uma aresta de retorno (código na lâmina após).

Ex: Faça uma busca em profundidade no grafo a seguir, a partir do vértice 8 selecionando a cada vez a aresta de menor número e calcule lowpt de cada vértice.

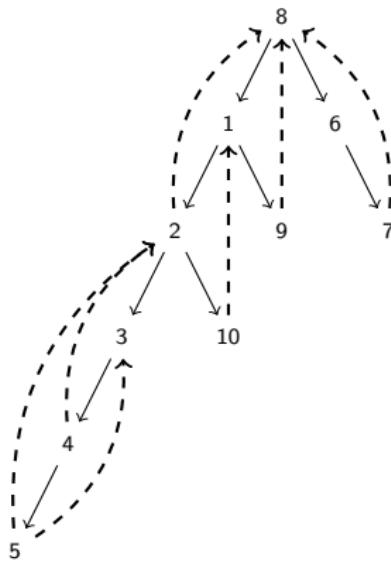
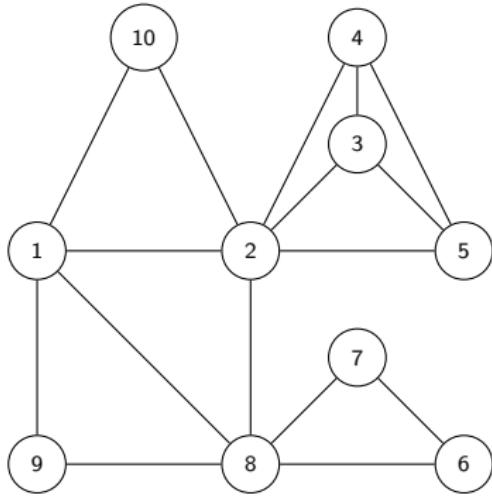


Figura: Grafo e Árvore em Profundidade

A função $\text{lowpt}(v)$ é dada a seguir:

v	1	2	3	4	5	6	7	8	9	10
lowpt	8	8	2	2	2	8	8	8	8	1

- A partir do cálculo dos lowpt pode-se identificar as pontes do grafo. As pontes são todas as arestas **da árvore** (isso é, as arestas de retorno não entram) que incidem em um vértice cujo lowpt é ele mesmo.
- Sugestão de exercício: URI 1790 - Detectando Pontes

- A aplicação da função `lowpt` na determinação das articulações é dada pelo Lema abaixo:
- Seja $G(V, E)$ um grafo conexo e T uma árvore de profundidade de G . Um vértice v é uma articulação se e somente se
 - 1 v é raiz de T e v possui mais de um filho, ou
 - 2 v não é raiz de T , e v possui um filho tal que $\text{lowpt}(w) = v$ ou w .

Identificação das componentes biconexas

- Sejam v, w vértices de G , v pai de w em T e $lowpt(w) = v$ ou w . Então w é chamado de demarcador de v . Uma articulação é pai de um ou mais demarcadores. Os filhos da raiz também são demarcadores.
- **Lema:** Seja $G(V, E)$ um grafo e T uma árvore de profundidade de G . Sejam $v, w \in V$, com w um demarcador de v tal que a subárvore T_w com raiz w não contém articulações de G . Então os vértices de T_w , juntamente com v , induzem um componente biconexo em G .

Exercícios

Algoritmo

- Inicialmente calculam-se as articulações e demarcadores de G .
- No passo geral escolhe-se um demarcador w tal que a subárvore T_w de T , com raiz w , não possua articulações de G .
- O vértice v , juntamente com os vértices de T_w induzem um componente biconexo em G .
- Retirar T_w de T .
- Além disso, se w é o único demarcador de v em T , não considerar mais v como articulação.
- O processo se repete até que não haja mais demarcadores.

Exercícios

- No grafo do exemplo:
 - articulações : 8 e 2
 - demarcadores : 1, 6, 3
- Escolhe-se o demarcador 3 ou 6 (a subarvore de 1 contém uma articulação). Digamos 3.
- Os vértices 2,3,4,5 compõem uma componente biconexa.
- Remove-se 3,4,5.
- O vértice 2 deixa de ser uma articulação.
- Escolhe-se o demarcador 1 (ou 6).
- Os vértices 8,9,1,2,10 compõem uma componente biconexa.
- Remove-se os vértices 1,2,9 e 10.
- Escolhe-se o demarcador 6.
- Os vértices 8,6 e 7 formam uma componente.
- Remove-se os vértices 6 e 7.
- Fim.

Faça uma função

```
void DFS(int raiz , /* número do vértice raiz */  
          int mat[N][N] , /* matriz de adjacências do  
          int pai[N] , /* vetor com o número do vértice  
                           de cada vértice na árvore geradora */  
          int nivel[N]) /* nível do vértice – 0 para a raiz */
```

que receba uma matriz de adjacências $\text{mat}[N][N]$ representando um grafo de N vértices e o vértice raiz, e efetue uma busca em profundidade no grafo, a partir do vértice raiz, retornando a árvore geradora gerada na própria matriz (1 - arestas de retorno, 2 - arestas diretas). A função deve retornar também vetores com o número do nível e do número do pai de cada vértice.

```
void DFS(int v)
{
int i;
for (i=1;i<=N; i++)
    if ((mat[v][i]!=0)&&(nivel[i]==1000))
    {
        mat[v][i]=2;
        mat[i][v]=0;
        pai[i]=v;
        nivel[i]=nivel[v]+1;
        DFS(i);
    }
}
```

- Faça uma função que receba uma matriz representando a árvore geradora da questão anterior e o vetor de nível de cada vértice, e retorne o vetor com o lowpt de cada vértice da matriz.

```
void f_lowpt(int v)
{
int i;
lowpt[v]=v;
for (i=1;i<=N; i++)
{
    if (mat[v][i]==2)/* aresta direta */
    {
        if (lowpt[i]==1000) f_lowpt(i);
        if (nivel[lowpt[i]]<nivel[lowpt[v]]) lowpt[v]=lowpt[i];
    }
    if (mat[v][i]==1) /* aresta de retorno */
        if (nivel[i]<nivel[lowpt[v]]) lowpt[v]=i;
}
}
```

Grafos Bipartidos

Def: Um grafo $G = (V, E)$ é **bipartido** (ou bipartite) se $\exists V_1, V_2 \subseteq V$ tal que $\forall v_1 \in V_1$, se $(v_1, v_2) \in E$, então $v_2 \in V_2$ (idem com $v_2 \in V_2$). Ex:

$$\begin{aligned}V_1 &= \{1, 2, 3\} \\V_2 &= \{4, 5, 6, 7, 8\}\end{aligned}$$

$$\begin{aligned}V_1 &= \{1, 2, 4, 6\} \\V_2 &= \{3, 5, 7\}\end{aligned}$$

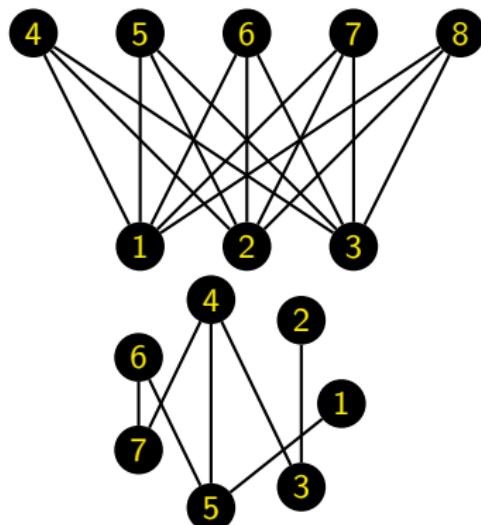


Figura: Grafos bipartidos

- Um grafo bipartido pode ser denotado pelos dois conjuntos de vértices ($G = (V_1 \cup V_2, E)$). Ex: $G = (\{1, 2\} \cup \{3, 4\}, E)$
- **Def:** Um grafo $G = (V, E)$ é **bipartite completo** se possui uma aresta para cada par de vértices $v_1, v_2, v_1 \in V_1$ e $v_2 \in V_2$. Sendo $n_1 = |V_1|$ e $n_2 = |V_2|$, um grafo bipartite completo é denotado K_{n_1, n_2} e possui $n_1 \times n_2$ arestas.
- **Teorema :** Um grafo $G(V, E)$ é bipartite se e somente se todos os ciclos têm comprimento par.

Exercícios de grafos bipartidos

Dígrafos (Grafos dirigidos)

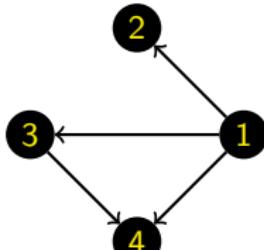
- **Def:** Um **dígrafo** G é um par (V, E) onde V é um conjunto finito e não vazio de vértices e E é um conjunto de arcos.
- **Def:** Um **arco** é uma relação entre vértices que possuem uma ordem implícita (são pares ordenados). Assim, $(1, 2) \neq (2, 1)$
- **Def:** Diz-se que um arco é **incidente de** um vértice v , se parte de v .
- **Def:** Um arco é **incidente a** um vértice v se chega em v .
 - Ex: $(1, 2)$ é incidente de 1 e incidente a 2.
- Obs: Uma aresta é equivalente a 2 arcos.

Def: Sejam 2 vértices v_1 e v_2 de um grafo $G = (V, E)$ Então:

- v_1 é adjacente a v_2 se $(v_2, v_1) \in E$ e
- v_2 é adjacente a v_1 se $(v_1, v_2) \in E$

Def: O **grau de saída** de um vértice v é o número de arcos que partem de V . O **grau de entrada** de um vértice V é o número de arcos que chegam a V .

Ex:



$$gr_e(1) = 0$$

$$gr_s(1) = 3$$

Def: **Grafo subjacente** é o grafo resultante da retirada das direções do grafo.

Planaridade

- **Def:** Um grafo G é dito **planar** se admite uma representação geométrica em um plano, na qual não haverá cruzamento de linhas (arestas).
- **Obs:** Todo grafo planar possui uma representação em que todas as arestas são retas.

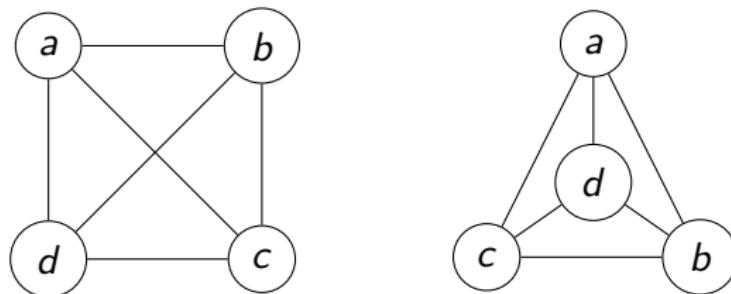
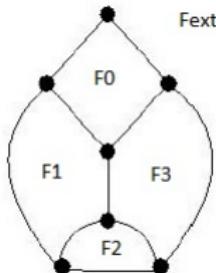
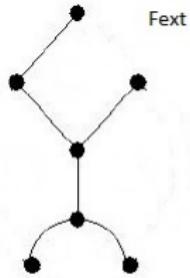


Figura: Duas representações do mesmo grafo

- **Def:** Uma **face** de um grafo planar é toda área minimal delimitada por um conjunto de arestas que formem um ciclo.
- O grafo abaixo possui 4 faces internas (F_0 a F_3).
- Considera-se a região externa ao grafo como uma face, pois também está delimitada pelo perímetro externo do grafo. É chamada de face externa.

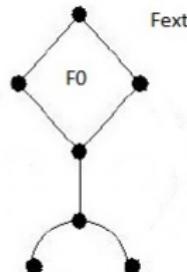


- **Teorema:** Seja G um grafo planar. Então $|V| + |F| = |E| + 2$ (fórmula de Euler para poliedros)
- **Prova:** Partindo da árvore geradora de G , $|V| = |E| + 1$. A cada passo o acréscimo de uma aresta acrescenta uma face interna ao grafo. Após completado o grafo, foram acrescentadas X arestas e consequentemente X faces e portanto $|F| + |V| = |E| + 1$. Se for considerada a face externa, $|F| + |V| = |E| + 2$

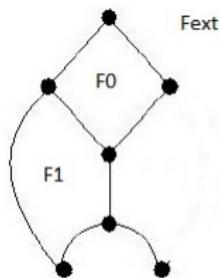


$$|V| + |F| = |E| + 2$$

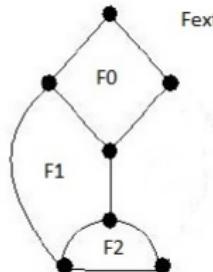
$$7 + 1 = 6 + 2$$



$$|V| + |F| = |E| + 2$$



$$|V| + |F| = |E| + 2$$



$$|V| + |F| = |E| + 2$$

- O teorema pode ser estendido para o caso de p componentes conexas, quando teremos $|F| + |V| = |E| + p + 1$
- Quanto maior o número de arestas com relação ao número de vértices, mais difícil se torna a obtenção de uma representação geométrica plana para G . Obs: O limite máximo para o número de arestas de um grafo planar é dado pelo lema a seguir:
 - **Lema:** Seja G um grafo planar. Então $|E| \leq 3|V| - 6$. É usual representar o número de arestas por m e o número de vértices por n , resultando na expressão $m \leq 3n - 6$.
- **Prova :** Cada face é delimitada por um mínimo de 3 arestas e cada aresta pertence a exatamente duas faces. Logo,
 $2m \geq 3f$. Substituindo na fórmula de Euler tem-se
 $f = m - n + 2 \Rightarrow (2/3)m \leq m - n + 2 \Rightarrow m \leq 3n - 6$.

Lema : Os grafos K_5 e $K_{3,3}$ são não planares.

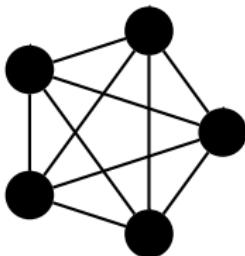


Figura: K_5

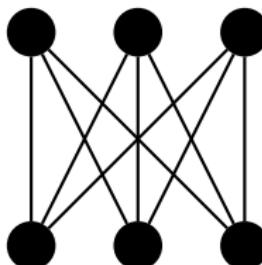


Figura: $K_{3,3}$

Prova : K_5 não atende a condição $m \leq 3n - 6$ ($m = 10$, $n = 5$) , logo K_5 não é planar. Para $K_{3,3}$ $n=6$ e $m=9$. Se $K_{3,3}$ é planar deve satisfazer a fórmula de Euler, ou seja, $f = m - n + 2 = 5$. Em qualquer representação plana de $K_{3,3}$ cada face deve ter pelo menos 4 arestas (ciclos em grafos bipartites devem ter comprimento par) e as arestas pertencem a 2 faces, logo, $2m \geq 4f$, o que contradiz $m=9$, $f=5$.

Def: Uma **subdivisão** de uma aresta (v, w) é uma função que transforma (v, w) em um caminho $v, v_1, v_2, v_3, \dots, w$

Ex:

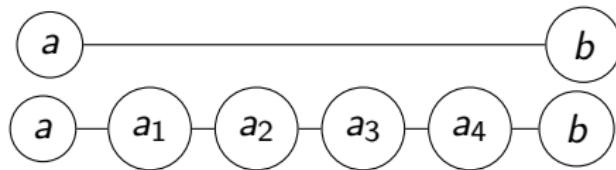
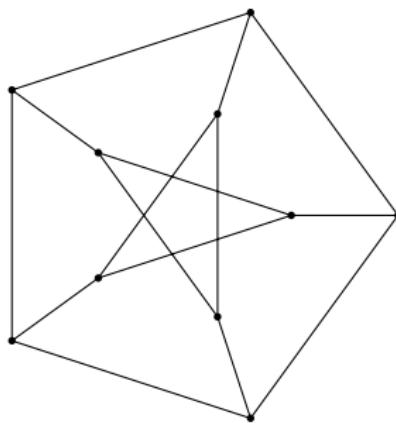


Figura: Subdivisão de uma aresta

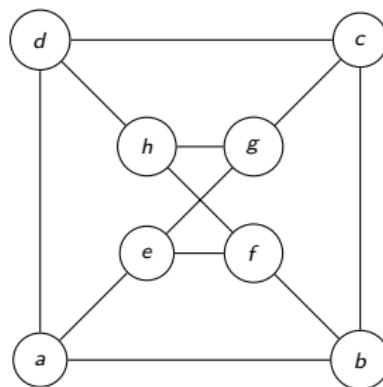
- **Def:** Um grafo G_2 é uma **subdivisão** (ou um homeomorfismo) de um grafo G_1 , quando G_2 puder ser obtido de G_1 através de uma seqüência de subdivisões de arestas de G_1 .
- **Teorema de Kuratowski:** Um grafo é planar se e somente se não contém como subgrafo uma subdivisão de $K_{3,3}$ ou K_5 .
- **Prova da Necessidade :** Como $K_{3,3}$ ou K_5 não são planares suas subdivisões também não são e qualquer grafo que as contenha também não é.
- **Prova da Suficiência :** Não será demonstrada

Ex: Utilizando o Teorema de Kuratowski mostre que o grafo de Petersen, mostrado a seguir, não é planar



(sugestão: remova um dos vértices do pentágono ou remova as duas arestas horizontais)

Ex: Idem, para o grafo a seguir:



Resposta: Remova a aresta horizontal inferior e resultará um homeomorfismo de $K_{3,3}$

Ex: Verifique se K_6 é planar.

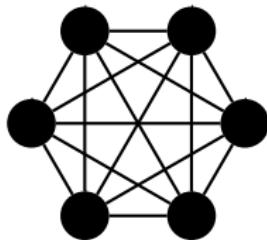


Figura: K_6

Resposta : K_6 tem 15 arestas e 6 vértices, logo, não atende a condição $|E| \leq 3|V| - 6$, não sendo, portanto, planar.

Ex: Verifique se o grafo a seguir é planar

$$18 < 3|12| - 6$$

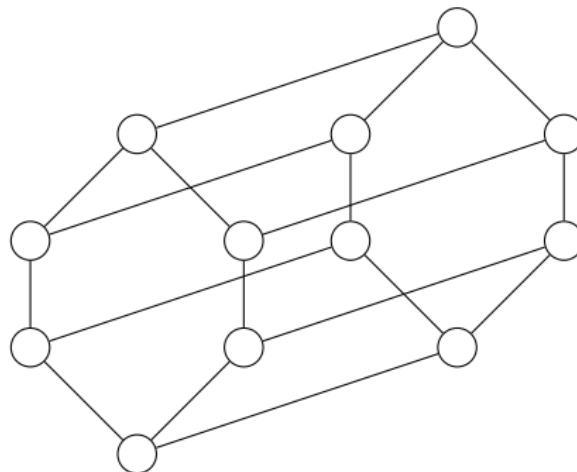


Figura: É planar?

Resposta : É planar. Basta colocar um hexágono dentro do outro.

Algoritmo de Planaridade de Demoucron (quadrático)

Pré-processamento e observações

- 1 Se o grafo não é conexo, o teste de planaridade é aplicado em cada componente conexa
- 2 Se o grafo é separável (isto é, tem um ou mais pontos de articulação), ele é planar se cada bloco é planar.
Desconecta-se então o grafo e aplica-se o teste a cada bloco.
- 3 Laços podem ser eliminados sem afetar a planaridade.
- 4 Cada vértice de grau 2 e suas arestas adjacentes podem ser substituídos por uma única aresta sem afetar a planaridade.
- 5 Arestras paralelas podem ser removidas sem afetar a planaridade.

[Voltar para o índice](#)

Os três últimos passos podem ser aplicados repetidamente até que não haja mais simplificações a fazer. Após isso, pode-se aplicar o teste:

- Se $|E| < 9$ ou $n < 5$ então o grafo é planar
- Se $|E| > 3n - 6$ então o grafo não é planar

E aqui começa o algoritmo de verificação de planaridade. Algumas definições:

Seja $G_1 = (V_1, E_1)$ um subgrafo de $G = (V, E)$. Uma **peça** (*piece*) de G relativo a G_1 é, então:

- Uma aresta $(u, v) \in E$ onde $(u, v) \notin E_1$ e $u, v \in V_1$.
- Uma componente conexa de $(G - G_1)$ mais todas as arestas incidentes a essa componente.

Os vértices que ligam a peça a G_1 são chamados pontos de contato. Se uma peça tem dois ou mais pontos de contato, é chamada de ponte.

- Obs:
 - \hat{G}_1 - representa uma representação planar do grafo G
 - *Embeddable* - significa que possui representação planar.
 - f - número de faces do grafo a cada passo
 - $F(B, G_i)$ - conjunto de faces onde a ponte B pode ser inserida

Algoritmo

1. Encontre um ciclo C em G
2. $i \leftarrow 1$
3. $G_1 \leftarrow C, \hat{G}_1 \leftarrow C$
4. $f \leftarrow 2$
5. $\text{Embeddable} \leftarrow \text{true}$
6. while $f < |E| - n + 2$ and Embeddable do
- 6.1. begin
- 6.2. encontre cada ponte B de G relativa a G_i ;
- 6.3. para cada B encontre $F(B, G_i)$
- 6.4. se para algum B , $F(B, G_i) = \emptyset$ então
- 6.4.1. begin
- 6.4.2. $\text{Embeddable} \leftarrow \text{false}$
- 6.4.3. escreva "Não é planar"
- 6.4.4. end
- 6.5. se Embeddable então
- 6.5.1. begin
- 6.5.2. se para algum B , $|F(B, G_i)| = 1$
- 6.5.3. então $F \leftarrow F(B, G_i)$
- 6.5.4. senão faça B ser qualquer ponte e F qualquer face tal que $F \in F(B, G_i)$
- 6.5.5. fim se
- 6.5.6. encontre um caminho $P_i \subseteq B$ conectando dois pontos de contato de B a G_i
- 6.5.7. $G_{i+1} \leftarrow G_i + P_i$
- 6.5.8. Obtenha uma representação planar \hat{G}_{i+1} de G_{i+1} desenhando P_i na face F de G_i
- 6.5.9. $i \leftarrow i + 1$
- 6.5.10. $f \leftarrow f + 1$
- 6.5.11. if $f = |E| - n + 2$ then escreva "G é planar"
11. end /Se
12. end /While
13. end.

Coloração

- Seja $G(V, E)$ um grafo e $C = \{C_1, C_2, \dots, C_n\}$ um conjunto de cores. Define-se:
 - **Coloração de G** : atribuição de alguma cor de C para cada vértice $v \in V$, de modo que dois vértices adjacentes tenham cores diferentes.
 - **K-coloração** : coloração que utiliza um total de k cores.
 - **Número cromático de G** : menor número de cores que define uma coloração de G .
 - **Def:** Uma coloração que utiliza um número mínimo de cores é chamada mínima (o número cromático define uma coloração mínima).

- Coloração pode ser utilizada para modelar alguns problemas de atribuição de valores com restrições (atribuição de horários, salas e semelhantes).
- Ex1: É necessário armazenar produtos químicos, e alguns deles não podem ser armazenados com outros (porque explodem). Qual o número mínimo de armários necessários?
- Ex2: Alocação de registradores em compiladores.

- Obs: O estudo da coloração iniciou com o problema das 4 cores, que visa colorir um mapa de forma que países vizinhos tenham cores diferentes.
- 4 cores são suficientes para colorir qualquer mapa?
 - Mapa : particionamento de uma região em subregiões, delimitadas por linhas.
 - Duas subregiões são adjacentes se possuem uma linha em comum.
- Representação por grafos:
 - Regiões \Rightarrow vértices
 - Linhas \Rightarrow arestas

Obs1: O grafo obtido a partir de um mapa é necessariamente planar.

Obs2: Problema de coloração dos mapas é equivalente à coloração de grafos planares.

Def: Um grafo $G(V, E)$ é **bicolorável** se e somente se é bipartite (ou, de forma equivalente, se não tem ciclos de comprimento ímpar).

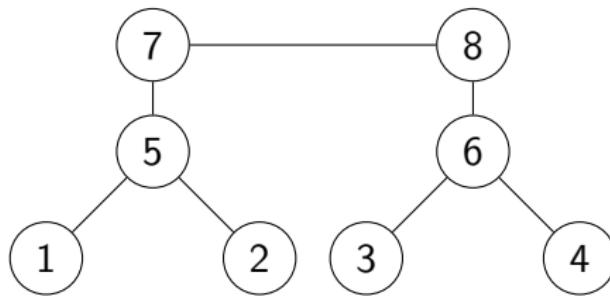
Coloração Aproximada

- Obter o número cromático de um grafo é bastante difícil.
Muitas vezes é suficiente obter um número aproximado ao número cromático.
- Seja:
 - $Adj(v)$: Conjunto de vértices adjacentes a v .
 - C_i : Conjunto de vértices que possuem a cor i .
 - n : número de vértices

Algoritmo de coloração aproximada.

- 1 Ordenar V em ordem decrescente, v_1, v_2, \dots, v_n de graus
- 2 $C_1 := C_2 := \dots := \emptyset$
- 3 Colorir v_1 com a cor 1 ($C_1 := \{V_1\}$)
- 4 Para $j = 2, \dots, m$ efetuar
 - 1 $r := \min \{i | Adj(v_j) \cap C_i = \emptyset\}$
 - 2 Colorir v_j com coloração r ($C_r := C_r \cup \{v_j\}$)

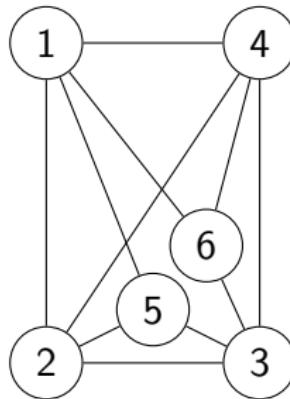
Ex: Coloração do grafo a seguir:



Para este exemplo o algoritmo dá a coloração ótima? Qual é a coloração ótima?

Exercício:

O grafo a seguir:



- 1) Produz coloração com quantas cores?

Resposta:3

Exercício: Escrever o algoritmo de coloração aproximada para um grafo representado por uma lista de adjacências.

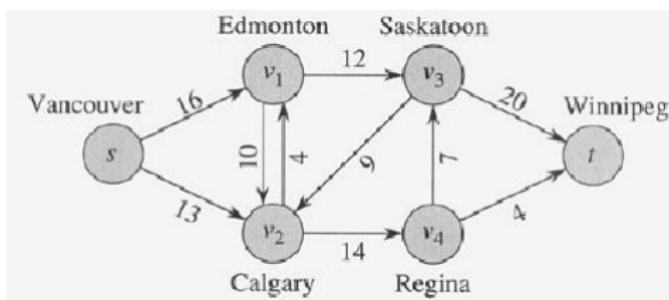
Problemas de circulação de fluxo em rede

- São problemas que envolvem circulação de alguma espécie (veículos, informação, matéria prima) em redes
- Aplicações incluem, entre outras:
 - Avaliação do fluxo máximo entre dois pontos
 - Líquidos em encanamentos
 - Elétrons em circuitos elétricos
 - Carros em vias
 - Identificação de "gargalos" no caminho (arestas que limitam o fluxo entre dois pontos)
 - Problemas produtor-consumidor (é possível escoar toda a produção de arroz/petróleo/... de uma região produtora para os centros consumidores?)
 - Associação em grafos bipartidos: buscar associação entre dois grupos de vértices maximizando o número de ligações entre eles

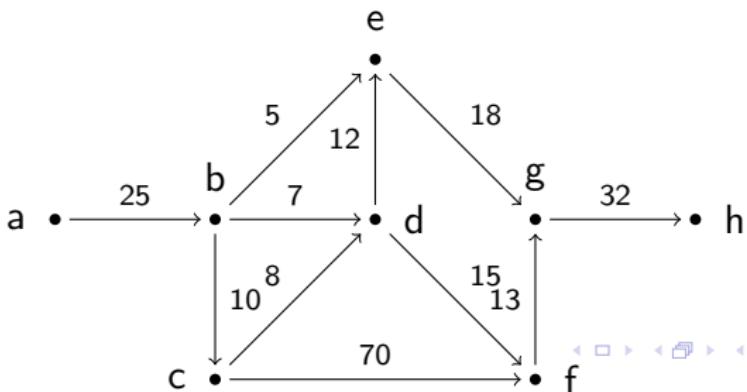
- Análise de tráfego de veículos:
 - Previsão de engarrafamentos
 - Como a redução/aumento no número de pistas de uma via impactará no fluxo entre dois pontos?



- Uma **rede** é um grafo dirigido que possui dois vértices especiais.
 - 1 O vértice **fonte** v_1 , do qual só partem arcos ($gre(v_1) = 0$) e todos os demais podem ser atingidos
 - 2 O vértice **sumidouro** v_2 , no qual só chegam arcos ($grs(v_2) = 0$)



- Ex: Na rede a seguir, os nós representam esquinas e os arcos ruas de mão única. As capacidades são expressas em dezenas de veículos por minuto. A rua representada pelo arco (c, f) está parcialmente fechada para obras, sendo sua capacidade normal de 70 veículos por minuto. Que influência terá, sobre o valor do fluxo máximo, a reabertura desta rua? Quais os "gargalos", ou seja, qual a ruas ou rua que limitam o bom fluxo de veículos?



- **Rede:** Dígrafo $D(V, E)$, com dois vértices especiais e distintos.
 - Origem (fonte) \Rightarrow Alcança todos os vértices : s
 - Destino (sumidouro) \Rightarrow Alcançado por todos os vértices : t
- A cada arco está associado um número real positivo denominado capacidade da aresta ($C(e), e \in E$) (Define a capacidade máxima de fluxo naquela aresta).

- Um fluxo f de s a t é uma função de f a t que a cada aresta de E associa um número real positivo, tal que:
 - 1 $0 \leq f(e) \leq c(e), \forall e \in E$ (o fluxo em cada aresta não ultrapassa sua capacidade).
 - 2 $\sum_{w1} f(w1, v) = \sum_{w2} f(v, w2), \forall v \in V, v \neq s \text{ e } v \neq t$ (o fluxo se conserva em cada aresta diferente de t e s : para cada vértice v , a soma dos fluxos dos arcos convergentes é igual à soma dos arcos divergentes de v . Este somatório é chamado valor do **fluxo** em v .

Exemplo 1: (o fluxo é o valor entre parênteses, a capacidade máxima é o valor fora dos parênteses)

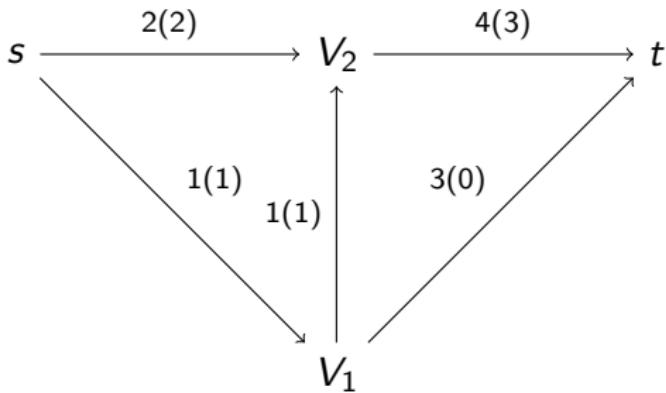


Figura: Exemplo 1

Assim, o valor do fluxo em $V_1 = 1$ e $V_2 = 3$

Em s , o valor do fluxo = soma das arestas divergentes = $2 + 1 = 3$

Em t , o valor do fluxo = soma das arestas convergentes = $3 + 0$

$\equiv 3$

É um fluxo válido?

Fluxo da rede :

Qual a capacidade máxima da rede?

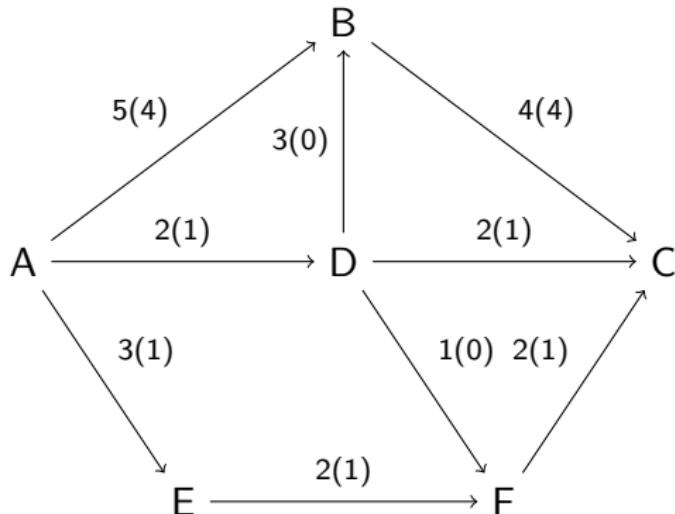


Figura: Exemplo 2

É um fluxo válido? Sim

Fluxo da rede : 6

Qual a capacidade máxima da rede? 8

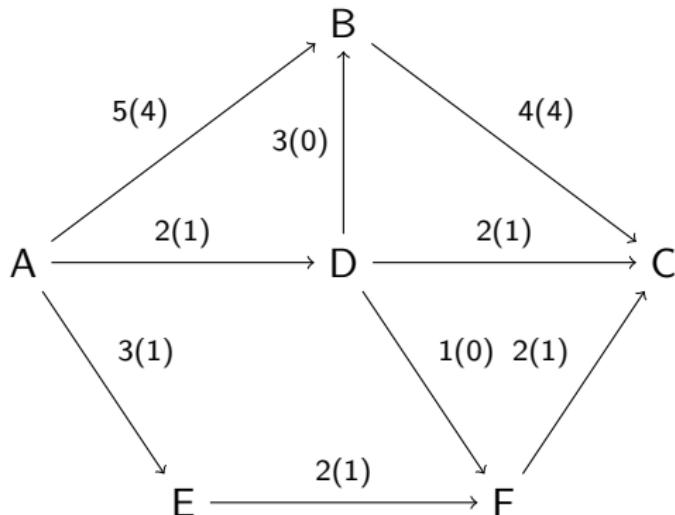


Figura: Exemplo 2

É um fluxo válido?

Fluxo da fonte :

Fluxo do sumidouro :

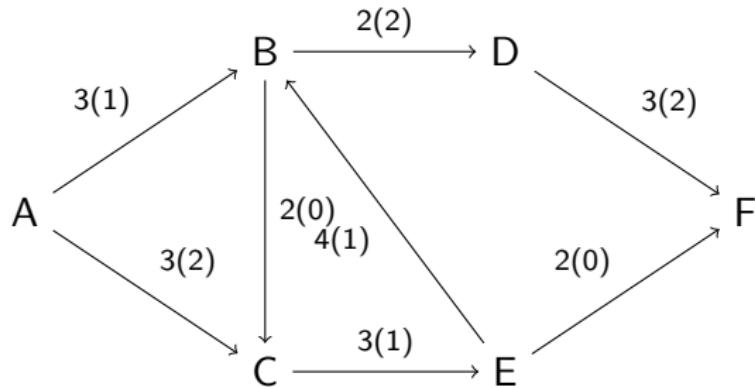


Figura: Exemplo 3

É um fluxo válido? Não

Fluxo da fonte : 3

Fluxo do sumidouro : 2

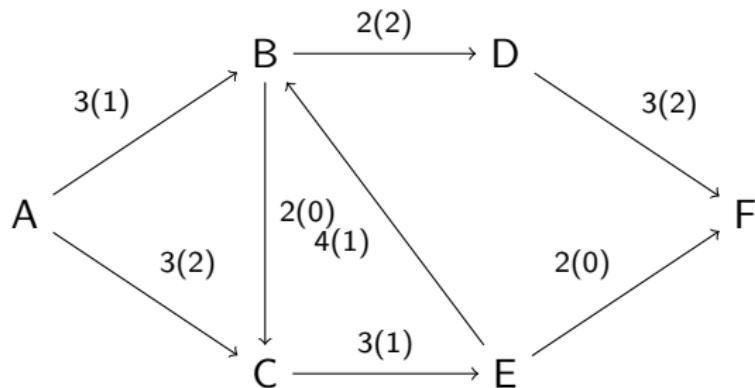


Figura: Exemplo 3

É um fluxo válido?

Capacidade máxima:

Fluxo da rede:

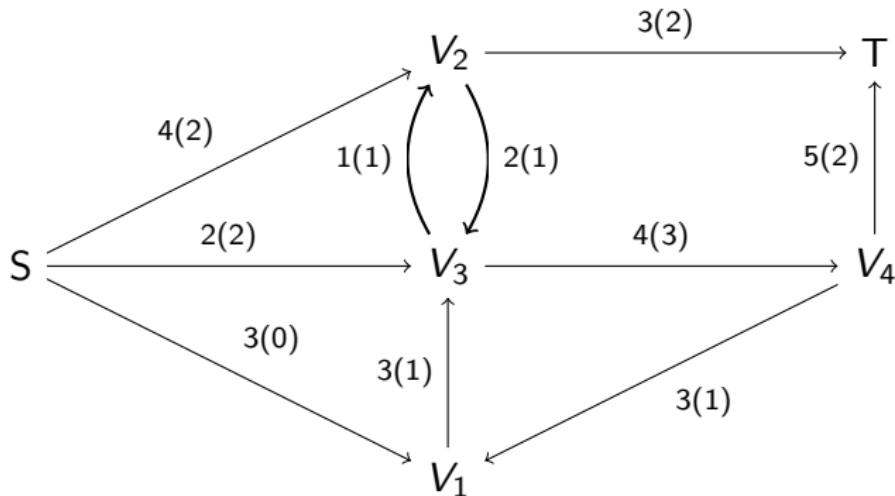


Figura: Exemplo 4

É um fluxo válido? Sim

Capacidade máxima: 8

Fluxo da rede: 4

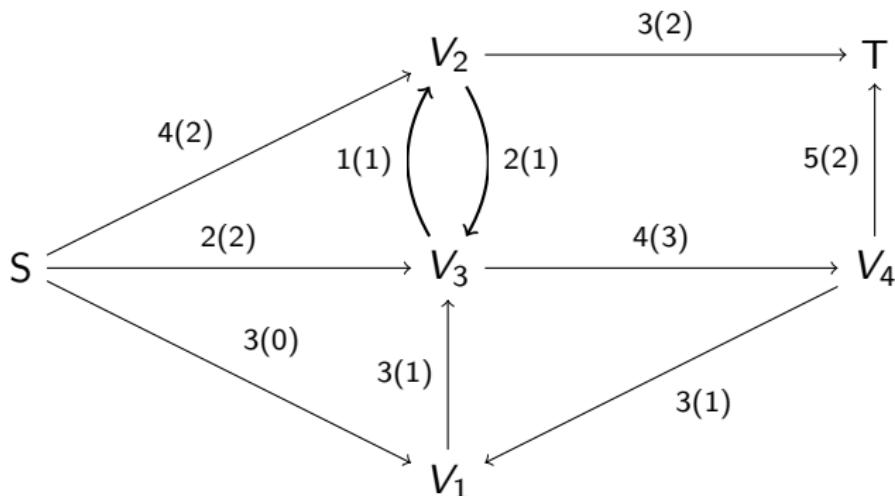


Figura: Exemplo 4

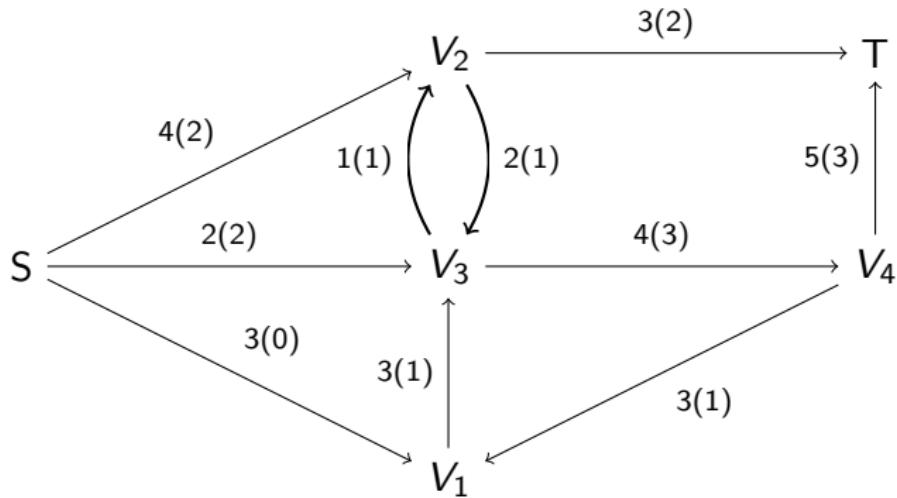


Figura: Exemplo 5

Não é um fluxo válido

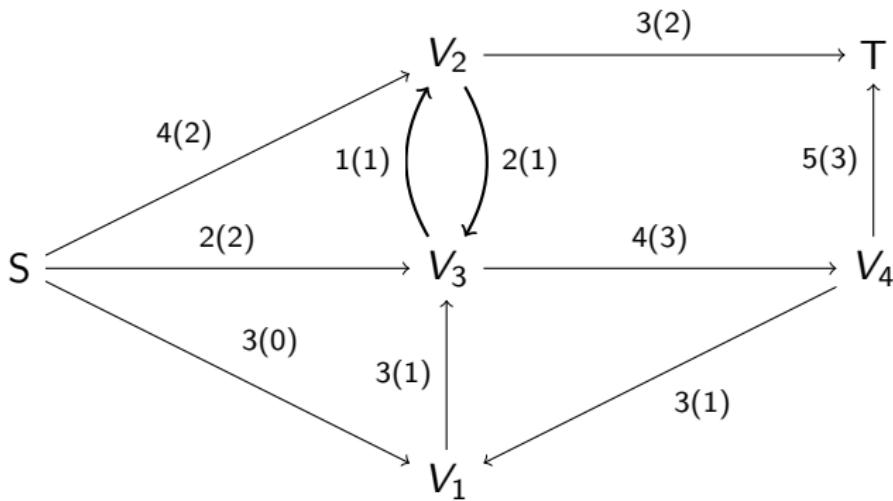


Figura: Exemplo 5

- **Fluxo da rede** : Soma dos fluxos do vértice fonte
- O valor de fluxo em uma rede pode variar de um mínimo igual a zero até um certo máximo. Por ex: o valor do fluxo da seguinte rede é 7, o qual é máximo para sua rede.

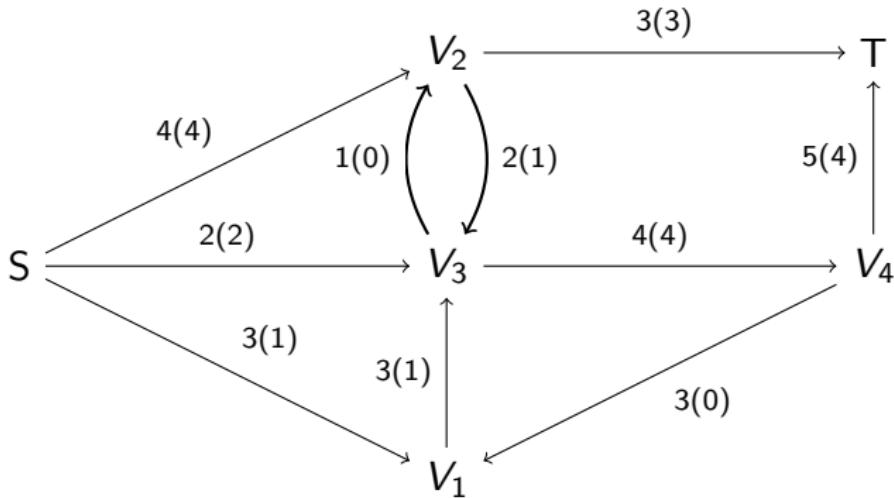


Figura: Exemplo de fluxo máximo

- O problema de fluxo máximo consiste em, dada uma rede, determinar tal fluxo, denominado fluxo máximo.
- **Def:** Diz-se que uma **aresta** está **saturada** se o seu fluxo é igual à sua capacidade ($f(e) = c(e)$)
- **Def:** Um **vértice saturado** é um vértice no qual todas suas arestas convergentes estão saturadas.

- Na rede abaixo, as arestas (S, v_2) , (S, v_3) , (v_2, T) , (v_4, T) e (v_3, v_4) estão saturadas
- E o vértice T está saturado.

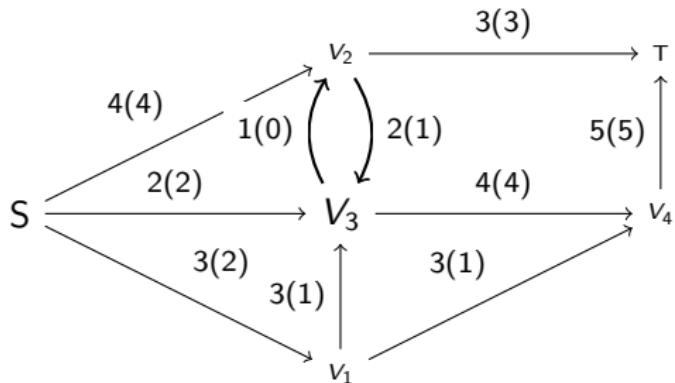


Figura: Exemplo de vértices e arestas saturadas

- O algoritmo para encontrar o fluxo máximo se baseia em encontrar, a cada passo, um caminho entre S e T que não tenha nenhuma aresta saturada, e acrescentar em todas as arestas do caminho o mesmo valor de fluxo.
- Isso poderia ser obtido a partir de uma busca (DFS ou BFS) a partir de S , tomando somente arestas não saturadas, até chegar a T .
- Não funciona quando, para chegar ao fluxo máximo, é necessário reduzir o fluxo em alguma aresta. Isso ocorre quando o fluxo na rede é maximal mas não é máximo.

- **Fluxo maximal** : Quando todo caminho de s a t possui pelo menos uma aresta saturada.

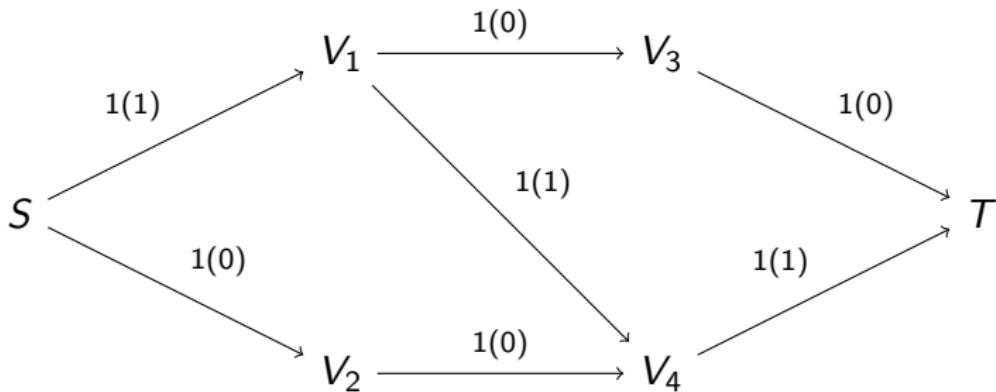
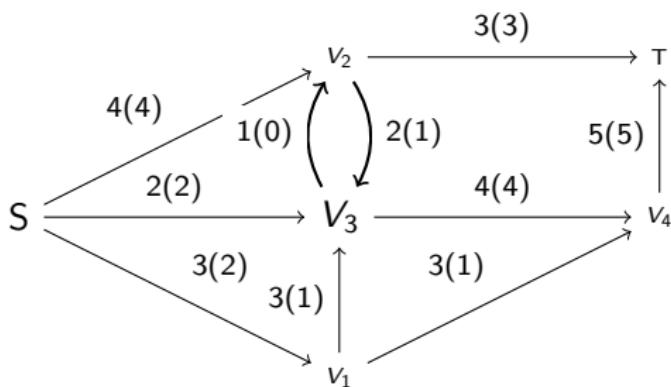


Figura: Este fluxo é maximal mas não é fluxo máximo

Cálculo do Fluxo Máximo

- Seja f um fluxo em uma rede D . O objetivo é aumentar o valor de f se possível. Toda aresta e não saturada pode receber um adicional de fluxo $\leq c(e) - f(e)$
- **Def:** Uma aresta $e \in E$, tal que $c(e) - f(e) > 0$ denomina-se **aresta direta**.
- Da mesma forma, toda aresta e em que $f(e) > 0$ pode ter seu fluxo reduzido em um valor $\leq f(e)$.
- **Def:** Uma aresta $e \in E$, tal que $f(e) > 0$ denomina-se **aresta contrária**.
- Algumas arestas podem ser ao mesmo tempo diretas e contrárias.

- Na rede abaixo, as arestas (S, v_1) , (v_1, v_3) e (v_1, v_4) são diretas, por terem folga entre a capacidade e o fluxo.
- E as arestas (S, v_1) , (v_1, v_3) e (v_1, v_4) também são contrárias, por terem fluxo maior que zero e consequentemente poder ter seu fluxo reduzido.



- Há situações em que a única forma de aumentar f consiste em, além de incrementar o fluxo de algumas arestas, decrementá-lo em outras.

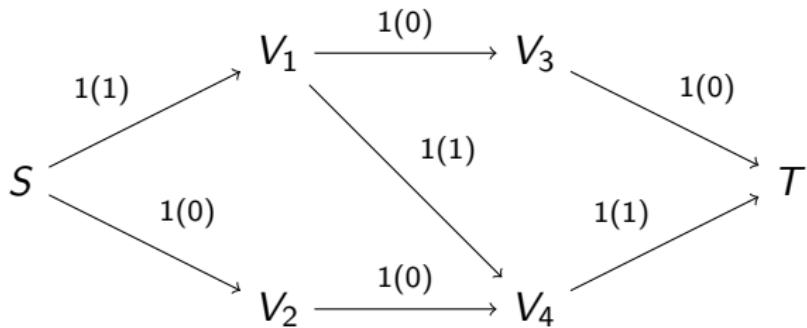


Figura: Para aumentar o fluxo é necessário decrementá-lo em (V_1, V_4)

- Para aumentar o valor de fluxo de D , é necessário decrementá-lo na aresta (v_1, v_4)

- **Def:** Dados f e $D(V, E)$, define-se **rede residual** $D'(f)$ como aquela em que o conjunto de vértices coincide com o de D e cujas arestas são obtidas pela seguinte construção:
 - Se (v, w) é aresta **direta** de D , então (v, w) é aresta de D' , também chamada **direta**, com capacidade $c'(v, w) = c(v, w) - f(v, w).$
 - Se (v, w) é aresta **contrária** de D , então (w, v) é aresta de D' , também chamada **contrária** e com capacidade $c'(w, v) = f(v, w).$
 - Isto é, as capacidades das arestas de D' representam as possíveis variações de fluxo que as arestas de D podem sofrer com o direcionamento de cada aresta indicando suas variações positivas e negativas.

Ex:

Arestas diretas : $\{(S, V_2), (V_2, V_4), (V_1, V_3), (V_3, T)\}$

Arestas contrárias : $\{(S, V_1), (V_1, V_4), (V_4, T)\}$

Rede residual D'

Def: Um caminho de s a t , na rede residual $D'(t)$ é denominado aumentante (ou caminho de acréscimo de fluxo) para f .

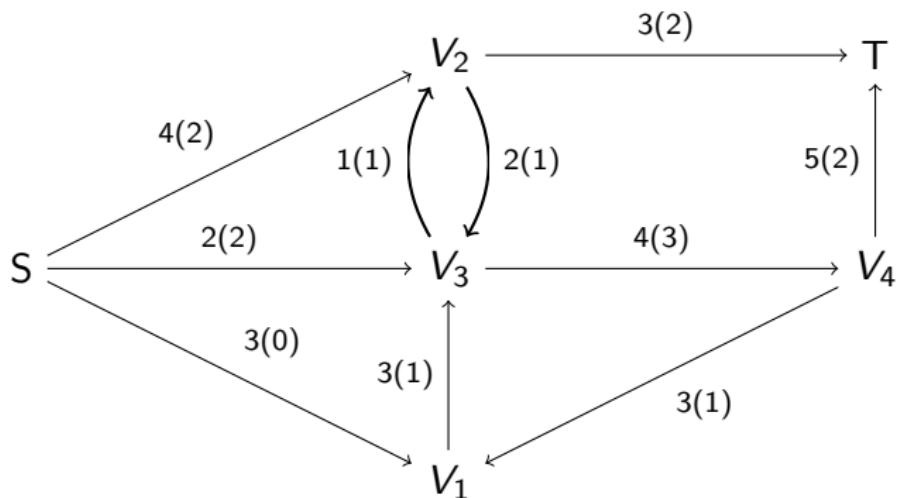
Por exemplo, o caminho s, v_2, v_4, v_1, v_3, t na rede anterior é aumentante para o fluxo da rede original. O seguinte lema descreve uma aplicação de rede residual ao problema do fluxo máximo.

Lema : Seja um fluxo em uma rede $D(V, E)$ e D' a rede residual correspondente. Suponha que exista em D' um caminho aumentante v_1, \dots, v_k de $v_1 \neq s$ até $v_k = t$. Então $f(D)$ pode ser aumentado de um valor

$$F' := \min c'(v_j, v_{j+1}) | 1 \leq j \leq k$$

Ex: Na rede residual do exemplo anterior, seja s, v_2, v_4, v_1, v_3, t um caminho aumentante, então $f(D)$ pode ser aumentado em 1. Incrementando (ou decrementando) em cada aresta do caminho em 1, teremos o fluxo máximo para 2.

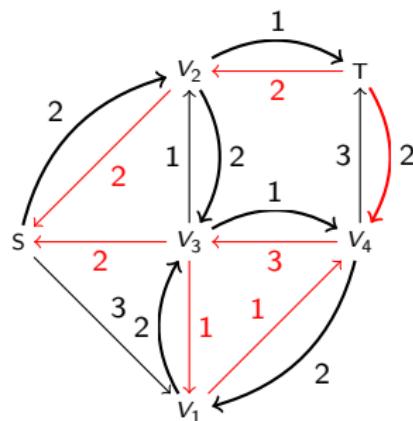
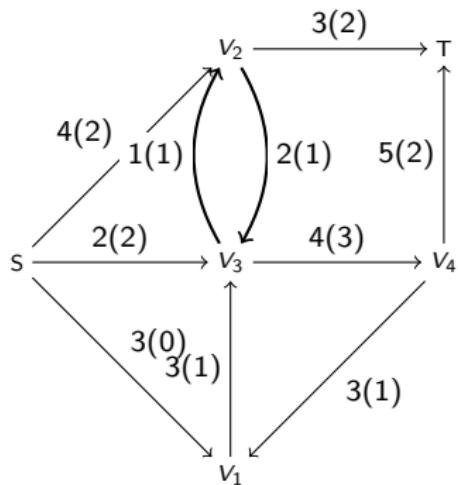
Ex : Ache a rede residual para a rede a seguir:



Arestas diretas: $\{(S, V_1), (S, V_2), (V_2, V_3), (V_1, V_3), (V_3, V_4), (V_4, V_1), (V_4, T), (V_2, T)\}$

Arestas contrárias: $\{(S, V_2), (S, V_3), (V_2, V_3), (V_3, V_2), (V_1, V_3), (V_3, V_4), (V_2, T), (V_4, V_1), (V_4, T)\}$

Rede residual D' (arestas contrárias em vermelho)



Nesta rede residual há um caminho aumentante s, v_1, v_4, t , de fluxo $F' = 1$. Isto significa que o fluxo em D pode ser incrementado de 1.

Seja $D(V, E)$ uma rede onde cada aresta $e \in E$ possui capacidade $c(e) > 0$.

Dados : rede (D, E) tal que $c(e) \in N$ e $c(e) > 0 \forall e \in E$.

Origem : s

Destino : t

$F := 0$

Para $e \in E$ efetuar $f(e) := 0$

Construir a rede residual $D'(f)$

Enquanto existir caminho v_1, \dots, v_k de $s = v_1$ até $t = v_k$ em D' efetuar

$F' := \min \{c'(v_j, v_{j+1}) - 1 \leq j \leq k\}$

Para $j = 1, \dots, k - 1$ efetuar

Se (v_j, v_{j+1}) é aresta direta então

$f(v_j, v_{j+1}) = f(v_j, v_{j+1}) + F'$

Senão $f(v_j, v_{j+1}) = f(v_j, v_{j+1}) - F'$

Fim se

$F := F + F'$

Fim Para

Construir a rede residual $D'(F)$

Fim Enquanto

Fim

Figura: Algoritmo de Ford-Fulkerson para encontrar o fluxo máximo

Ex:

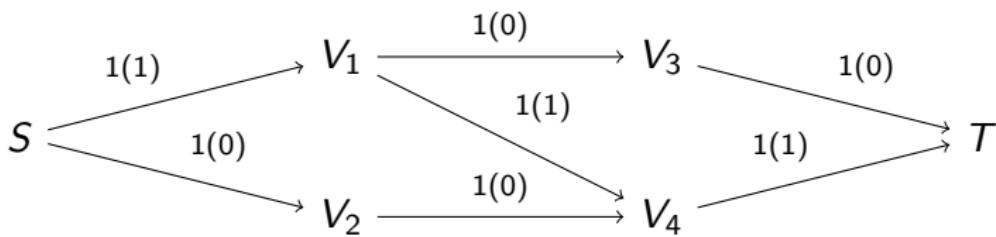


Figura: Rede inicial

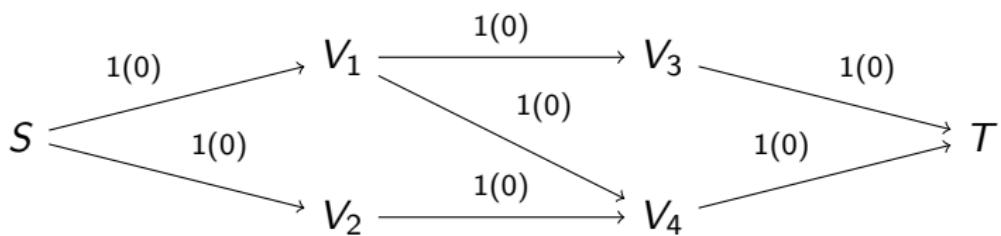


Figura: Zerar Fluxo

Calcular Rede residual e identificar caminhos de S a T.

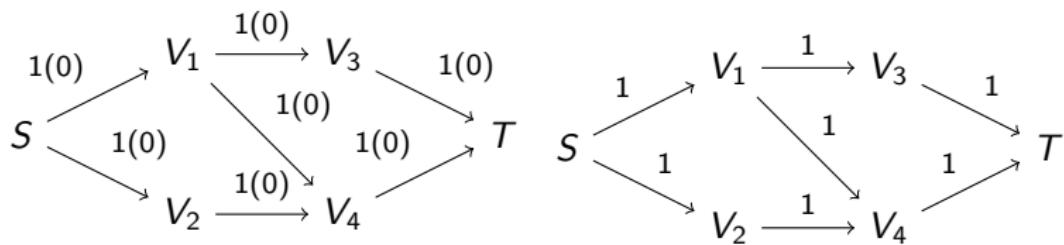


Figura: Rede Original com fluxo zerado

Figura: Rede Residual

Calcular Rede residual e identificar caminhos de S a T
 $(S, V_1, V_3, T).F' = 1$

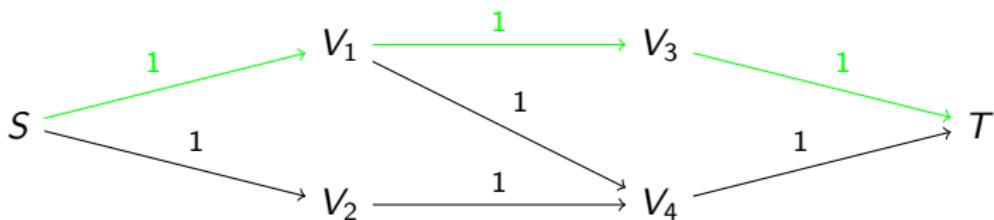


Figura: Rede Residual

Calcular novo fluxo em D

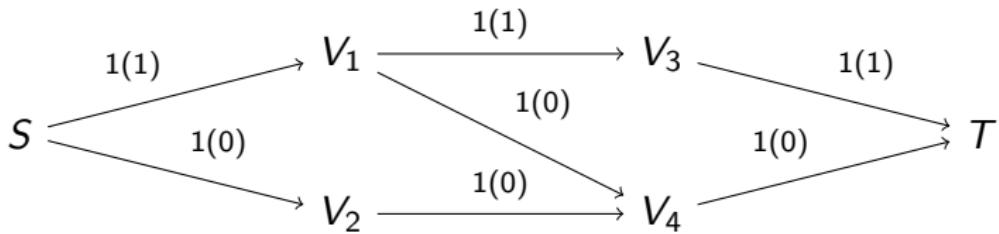


Figura: Novo Fluxo

Calcular nova rede residual e identificar caminhos de S a T.

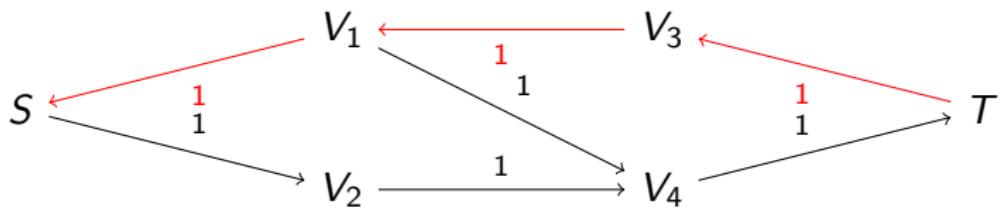


Figura: Nova Rede Residual

Calcular nova rede residual e identificar caminhos de S a T
 $(S, V_2, V_4, T).F' = 1$

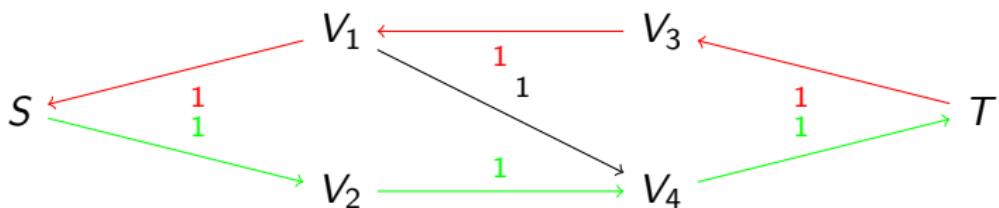


Figura: Nova Rede Residual

Calcular novo fluxo em D

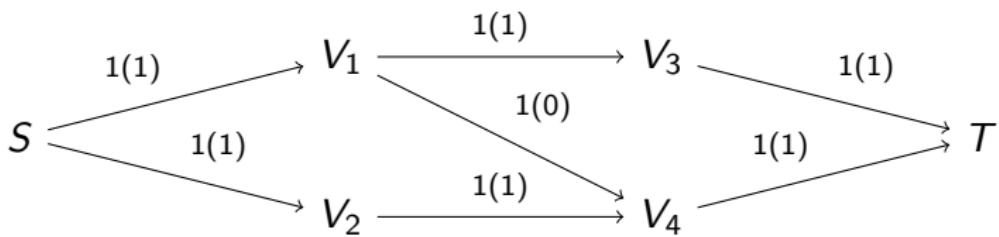


Figura: Novo Fluxo

Calcular a nova rede residual e identificar caminhos de S a T (não há mais).

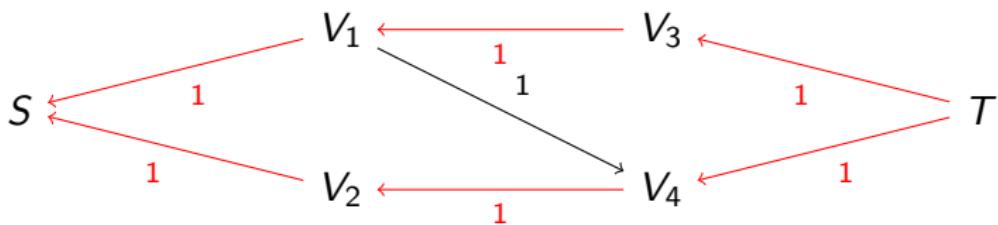


Figura: Última rede residual

Exemplo 2

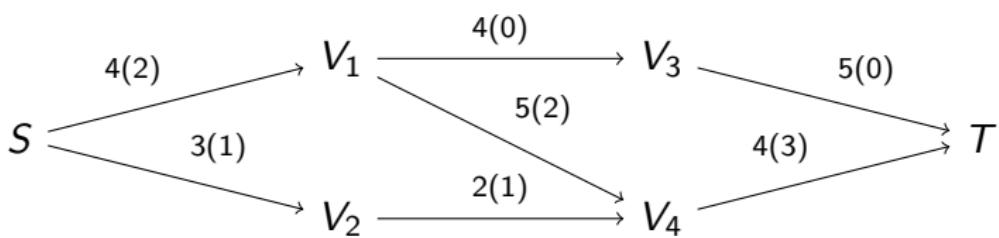


Figura: Rede inicial

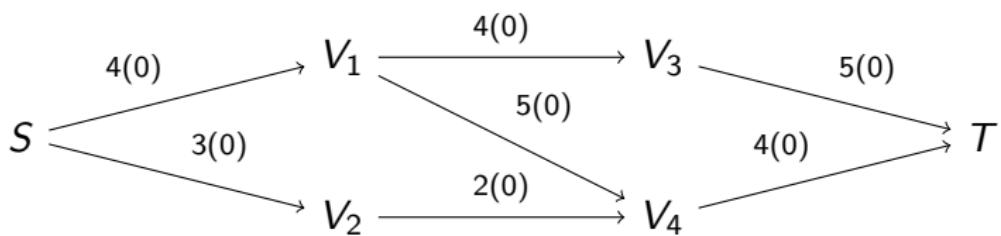


Figura: Zerar Fluxo

Calcular Rede residual e identificar caminhos de S a T.

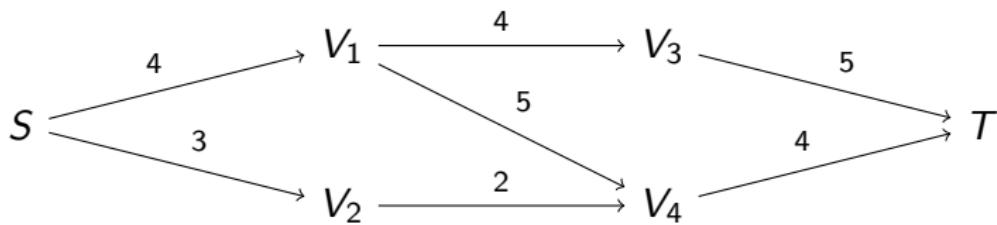


Figura: Primeira rede residual

Calcular Rede residual e identificar caminhos de S a T
 (S, V_1, V_3, T) . $F' = 4, F = 4$

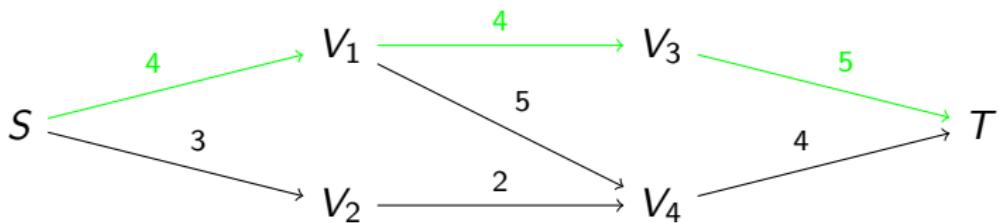


Figura: Primeira rede residual

Calcular novo fluxo em D

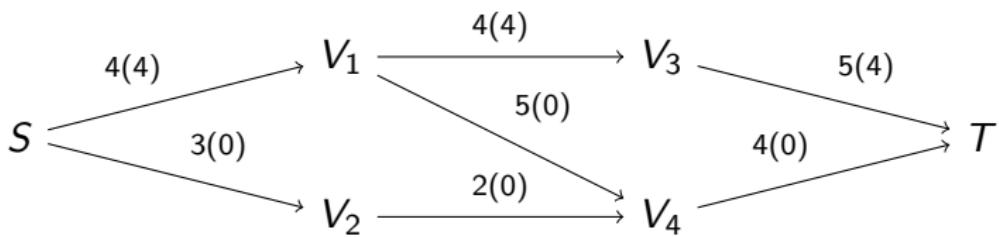


Figura: Novo fluxo

Calcular nova rede residual e identificar caminhos de S a T.

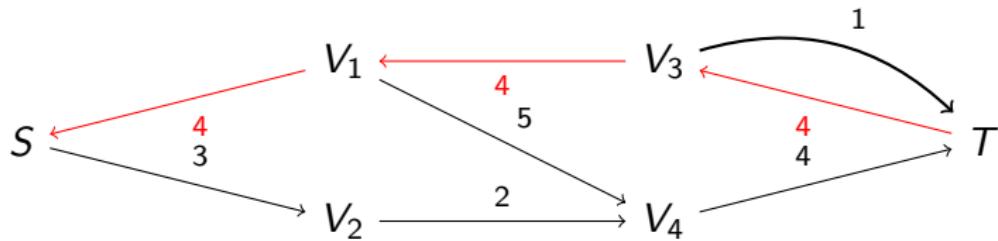


Figura: Segunda rede residual

Calcular nova rede residual e identificar caminhos de S a T
 (S, V_2, V_4, T) . $F' = 2, F = 6$

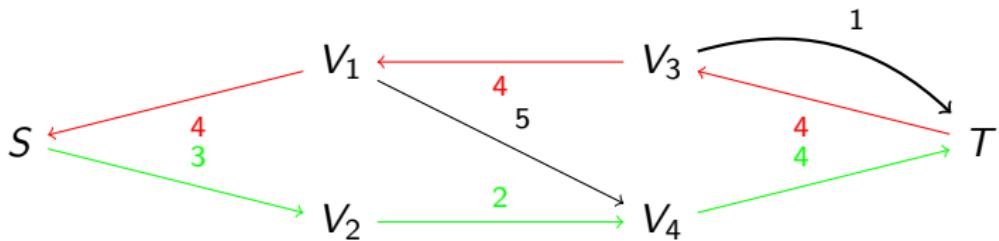


Figura: Segunda rede residual

Calcular novo fluxo em D

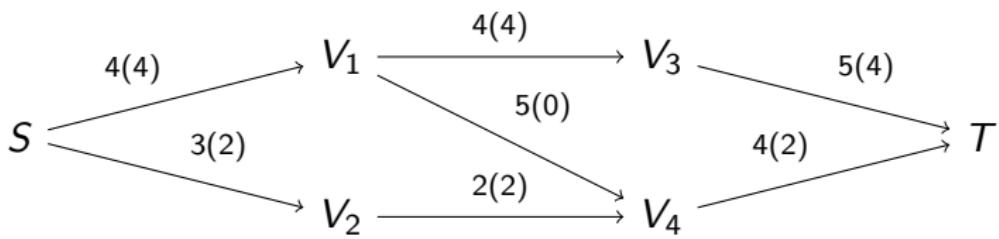


Figura: Novo fluxo

Calcular nova rede residual e identificar caminhos de S a T (não há mais).

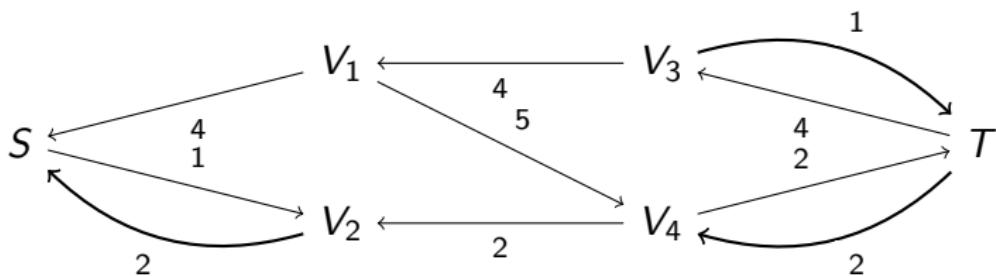


Figura: Última rede residual

Exercício: Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade [EVEN79, prob.5.1].

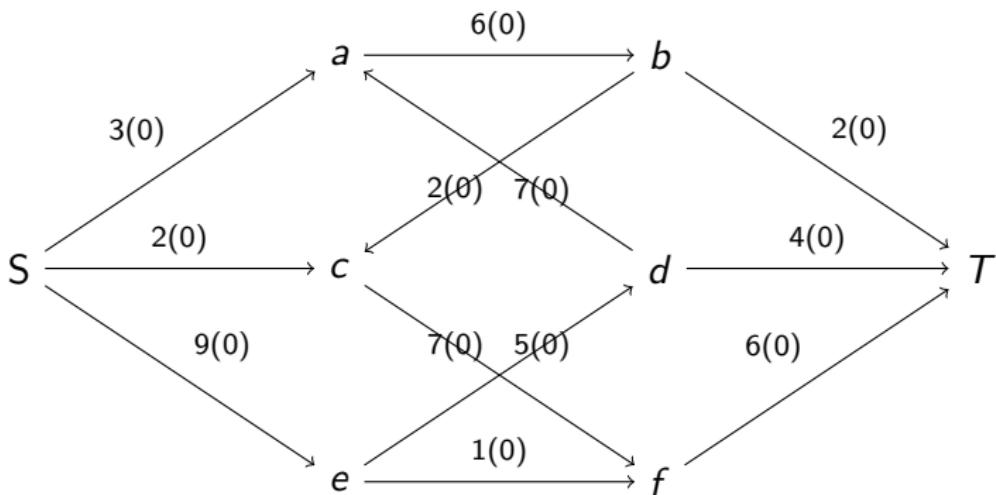


Figura: Encontre o fluxo máximo

Exercício: Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade [EVEN79, prob.5.1].

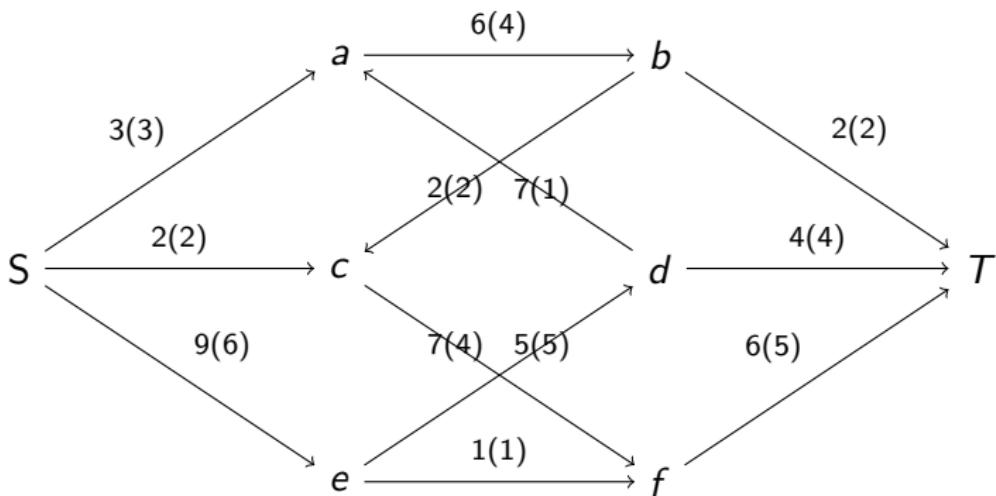
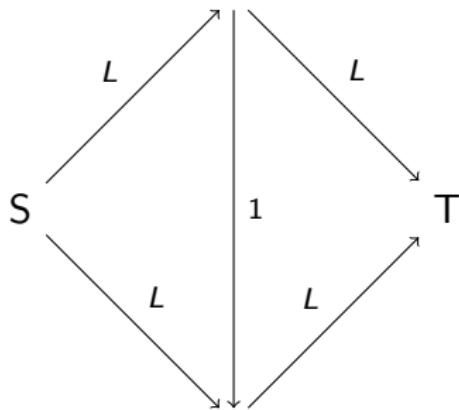


Figura: Resposta do problema anterior

- A complexidade do algoritmo de Ford-Fulkerson é da ordem de $O(M.F)$, onde M é o número de arestas e F é o fluxo máximo. Qual o número de iterações para a rede a seguir se a cada iteração for escolhido o caminho através da aresta 1?



Algoritmo de Edmonds-Karp

- Se o caminho escolhido for o de menor número de arestas (encontrado por uma busca em amplitude) a complexidade do algoritmo se reduz a $O(VE^2)$. Nesse caso, o algoritmo se chama algoritmo de Edmonds-Karp.

Problemas de fluxo máximo-custo mínimo

- Em algumas situações, problemas de fluxo em redes podem ter um custo associado a cada aresta, sendo o custo proporcional ao fluxo que circula na aresta.
- Um exemplo disso seria problemas relacionados a melhor caminho entre uma fábrica e um depósito em que além da capacidade máxima de transporte em cada rota há um custo associado (tempo, combustível)
- Nesses problemas pode-se buscar um fluxo **K**, de custo mínimo ou, dentre as diversas alternativas de fluxo máximo, aquela de custo mínimo.

[Voltar para o índice](#)

- Basicamente a solução desses problemas pode ser obtida por uma variante do algoritmo de Ford-Fulkerson, em que, ao buscar o caminho aumentante, busca-se a cada passo o de menor custo entre s e t .
- Se os custos são positivos, pode-se usar Dijkstra.
- Se os custos podem ser negativos, pode-se usar Bellman-Ford.

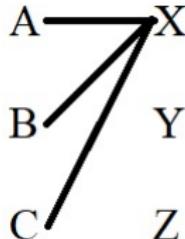
Problemas de associação de custo mínimo em grafos bipartidos

- São problemas em que se deve associar N tarefas a N pessoas minimizando o custo total. Imagine que se tem a seguinte matriz com o custo de associar a cada pessoa A, B e C, as tarefas X, Y e Z.
- Cada posição (i,j) da matriz representa o custo de atribuir a tarefa **j** à pessoa **i**
- Busca-se a atribuição de uma tarefa a cada pessoa que minimize o custo total.

	X	Y	Z
A	10	20	30
B	7	22	17
C	15	21	21

- Problemas de associação de custo mínimo podem ser resolvidos de forma tabular pelo chamado Algoritmo Húngaro, que consiste dos seguintes passos (para uma matriz de ordem N):
 - Para cada linha subtraia de todos os seus valores o valor do menor elemento da linha.
 - Para cada coluna subtraia de todos os seus valores o valor do menor elemento da linha.
 - Encontre o número mínimo L de linhas e colunas para cobrir todos os 0's da matriz
 - Enquanto $L <> N$:
 - $M = \text{menor número entre as posições não cobertas}$
 - Subtraia M de todas as posições não cobertas
 - Adicione M a todas as posições que estão em intersecções de linhas e colunas cobertas
 - Recalcule L (o número mínimo de linhas e colunas para cobrir todos os 0's)

- O número mínimo de linhas e colunas para cobrir todos os zeros da matriz pode ser obtido encontrando a **cobertura mínima de vértices** em um grafo bipartido onde os vértices do primeiro grupo correspondem às linhas da matriz, e os vértices do segundo grupo correspondem às colunas.
- Cada 0 na posição (i,j) da matriz acrescentará uma aresta no grafo ligando o vértice i ao vértice j .



	X	Y	Z
A	0	10	20
B	0	15	10
C	0	6	6

- Aplicando no exemplo acima:
- Subtraindo de cada linha o menor elemento da linha:

	X	Y	Z
A	10	20	30
B	7	22	17
C	15	21	21

	X	Y	Z
A	0	10	20
B	0	15	10
C	0	6	6

- Subtraindo de cada coluna o menor elemento da coluna:

	X	Y	Z
A	0	10	20
B	0	15	10
C	0	6	6

	X	Y	Z
A	0	4	14
B	0	9	4
C	0	0	0

- Como o número de linhas/colunas necessárias para cobrir todos os zeros ($L=2$) é menor do que N , subtrai-se de todas as posições não cobertas o menor valor não coberto (4) e soma-se aos elementos nas intersecções das linhas e colunas que cobrem os zeros (no caso, a posição (C,X)).

	X	Y	Z
A	0	4	14
B	0	9	4
C	0	0	0

	X	Y	Z
A	0	0	10
B	0	5	0
C	4	0	0

- Como o número de linhas/colunas necessárias para cobrir todos os zeros ($L=3$) é igual a N , encontrou-se uma atribuição ótima (na verdade, duas), identificadas pelos conjuntos de zeros abaixo:

	X	Y	Z
A	0	0	10
B	0	5	0
C	4	0	0

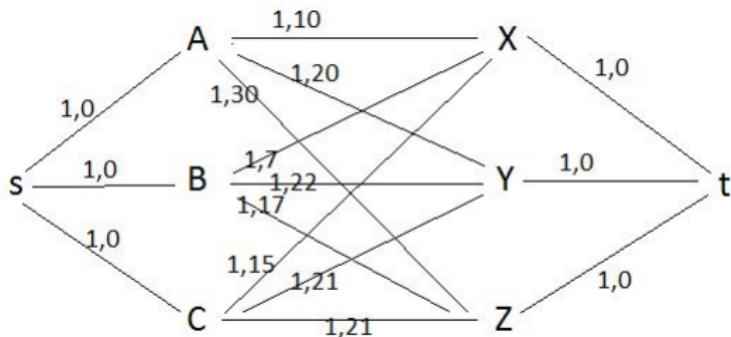
	X	Y	Z
A	0	0	10
B	0	5	0
C	4	0	0

- Que correspondem às atribuições a seguir, ambas com custo total igual a 48

	X	Y	Z
A	10	20	30
B	7	22	17
C	15	21	21

	X	Y	Z
A	10	20	30
B	7	22	17
C	15	21	21

- Problemas de associação com custo mínimo também podem ser resolvidas de forma semelhante à associação máxima em grafos bipartidos. O problema do exemplo poderia ser modelado como a rede abaixo, onde o primeiro número na aresta é a capacidade, e o segundo número é o custo:



Problemas produtor x consumidor

- Problemas produtor x consumidor, no contexto de Fluxo em Redes ou Programação Linear (em Sistemas Operacionais tem outro significado) são problemas em que se deseja analisar o fluxo entre um ou mais produtores (bens, informação, etc) e um ou mais consumidores.
- Considere o problema a seguir:
 - A demanda de transporte de arroz no Rio Grande do Sul, após a colheita (em milhares de toneladas/dia) é de 15 a partir da região de Pelotas (Pe), 18 a partir de Santa Maria (SM) e 12 a partir de São Borja (SB).

- Em Porto Alegre (PA) e em Caxias do Sul (Cx) há duas estações de beneficiamento que podem processar até 30.000 e 20.000 ton/dia respectivamente.
- O arroz é adquirido por : Florianópolis (Flo), 4.000 ton/dia; Curitiba (Cur), 7.000; São Paulo (SP), 19.000 e Rio de Janeiro (Rio), 15.000.
- Considera-se que o beneficiamento não acarreta perdas.
- O transporte pode ser feito, nos diversos itinerários, até os seguintes limites (em 1.000 ton/dia).

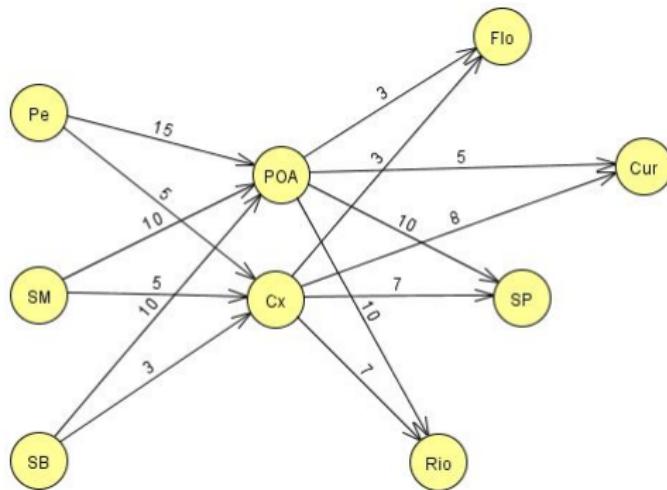
[Voltar para o índice](#)

De	Para	Até	De	Para	Até	De	Para	Até
Pe	PA	15	Pe	Cx	5	SM	PA	10
SM	Cx	5	SB	PA	10	SB	Cx	3
PA	Flo	3	PA	Cur	5	PA	SP	10
PA	Rio	10	Cx	Flo	3	Cx	Cur	8
Cx	SP	7	Cx	Rio	7			

Tabela: Limite de transporte em cada itinerário

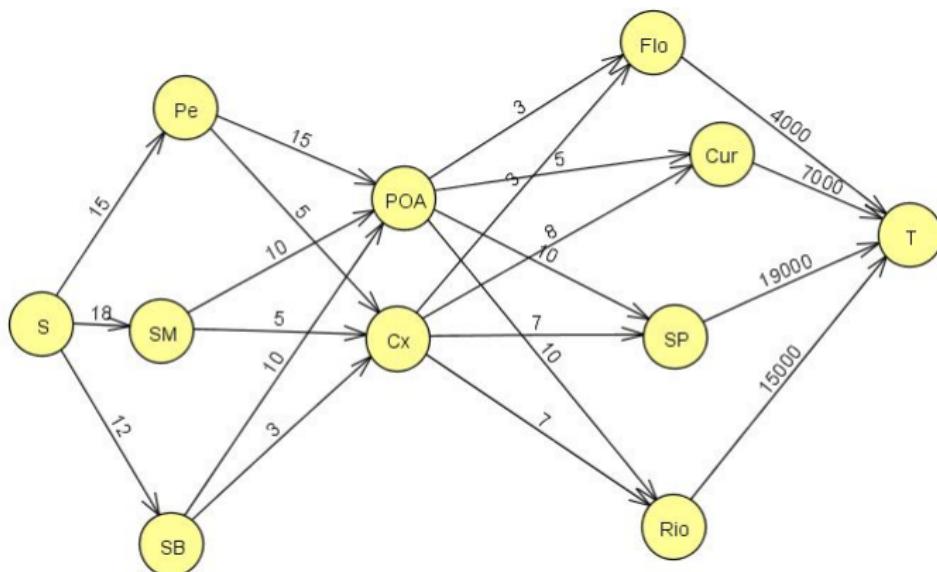
- Ache o fluxo máximo e procure verificar onde haveria oportunidades para ingresso de novos caminhoneiros no transporte de arroz.

- A figura abaixo mostra uma possível modelagem para a rede de transporte.



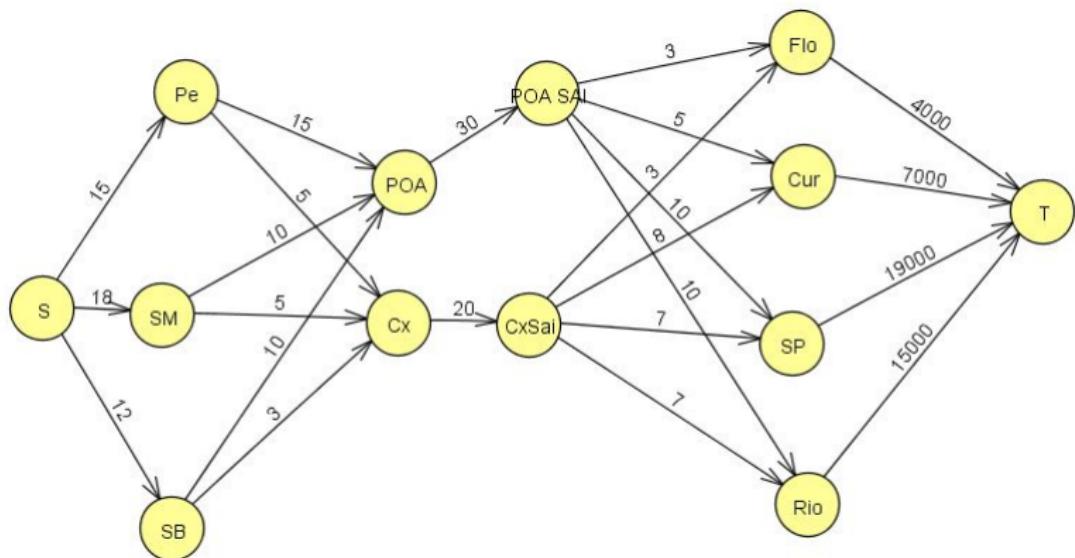
- Esse modelo não contem a informação de qual a produção em cada um dos centros produtores.
- Uma forma de modelar isso é acrescentar um vértice inicial e arestas do vértice inicial para cada vértice produtor.
 - A capacidade de cada aresta é a capacidade de produção do produtor.
- Da mesma forma, a necessidade de consumo de cada consumidor pode ser modelada acrescentando um vértice final e arestas de cada consumidor até ese vértice final.
 - A capacidade de cada aresta é a necessidade de consumo do consumidor.

- A figura abaixo mostra uma possível modelagem para a rede de transporte incluindo a capacidade de cada produtor e necessidade de cada consumidor.



- O modelo ainda não inclui a informação da capacidade de beneficiamente de arros em Porto Alegre e Caxias.
- No caso geral, isso representa uma restrição no fluxo máximo através de um vértice.
- Uma forma de modelar isso é separar o vértice em dois vértices, um deles representando o fluxo que chega ao vértice e um representando o fluxo que sai do vértice, e ligar esses dois vértices por uma aresta com capacidade igual ao fluxo máximo que pode passar pelo vértice.

- A figura abaixo mostra o modelo incluindo a limitação de fluxo através de Porto Alegre e de Caxias.



- Ex: Na rede a seguir, x_1 , x_2 e x_3 são fontes do mesmo produto. A quantidade disponível em x_1 é 5, em x_2 é 10 e em x_3 é 5. Os vértices y_1 , y_2 e y_3 são consumidores. A quantidade de produto necessária em y_1 é 5, em y_2 é 10 e em y_3 é 5.
- Descubra se é possível atender a estas condições (Dica: Uma forma de resolver este tipo de problema é introduzir uma fonte auxiliar s e um sumidouro t ; conectar s a x_i por uma aresta de capacidade igual à quantidade disponível em x_i ; conectar cada y_i a t através de uma aresta de capacidade igual à demanda de y_i ; encontrar um fluxo máximo na rede resultante e observar se todas as demandas são atendidas.

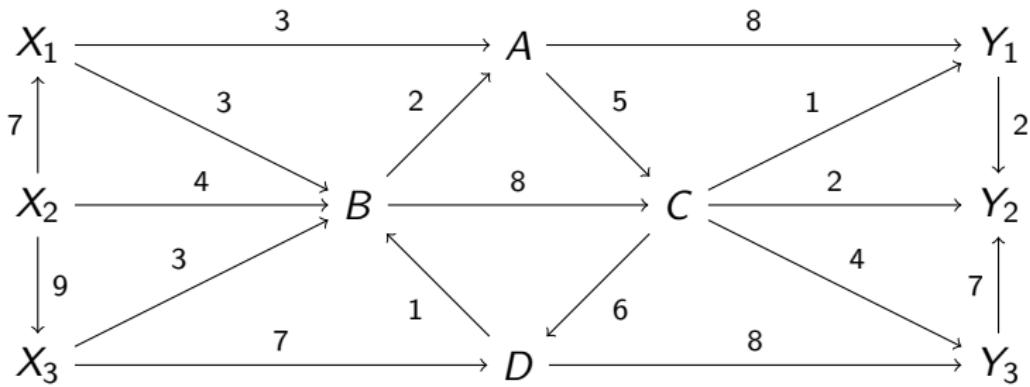


Figura: Problema produtor x consumidor

- Na rede da página a seguir, além das capacidades das arestas, cada vértice, exceto s e t , tem um limite superior no fluxo que pode circular através dele. As capacidades dos vértices estão escritas abaixo do label.
- Encontre um fluxo máximo para a rede. (Dica: Uma forma de resolver este tipo de problema é substituir cada vértice v por dois vértices v' e v'' ligados por uma aresta (v', v'') , com $c(e)$ igual ao limite de fluxo do vértice v . Todas as arestas que entravam em v passam a entrar em v' e todas as arestas que saíam de v passam a sair de v'' .

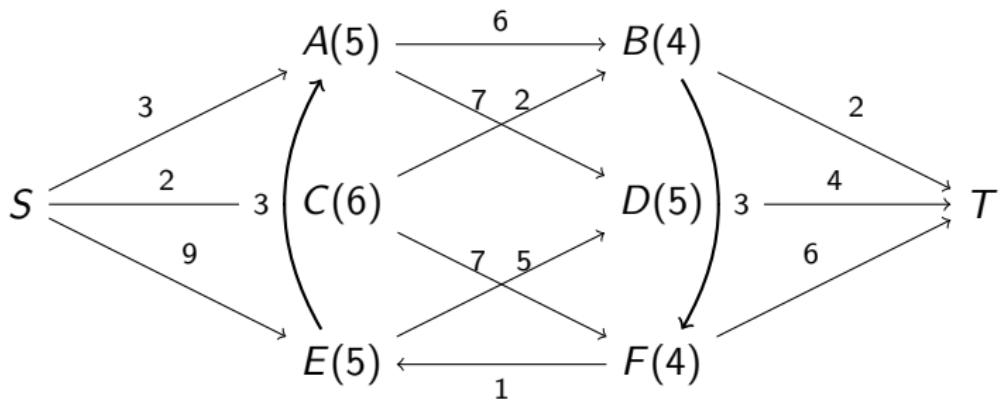
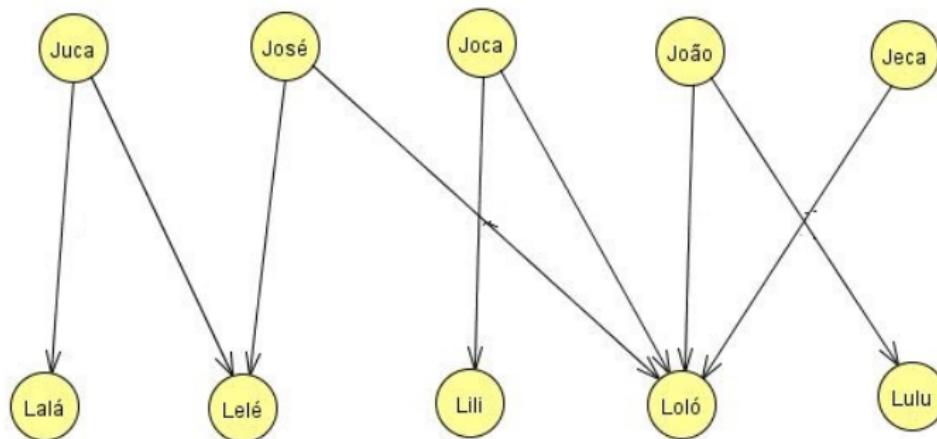


Figura: Problema produtor x consumidor com restrições nos vértices

Resolução de problemas de (associação máxima) Maximum Matching em grafos bipartites usando fluxo em redes

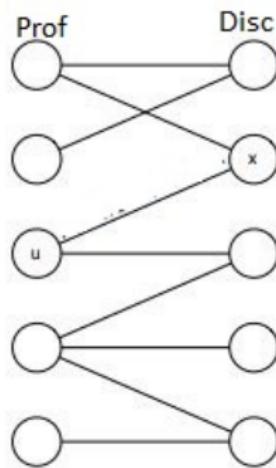
- Considere o seguinte problema prático: Haverá um baile e 5 rapazes, João, José, Juca, Joca e Jeca querem convidar as cinco irmãs Lalá, Lelé, Lili, Loló e Lulu, mas cada um tem suas preferências:
 - João gostaria de ir com Loló, mas não se importaria de ir com Lulu
 - José gostaria de ir com Lelé, mas poderia ir com Loló
 - Juca gostaria de ir com Lalá, mas poderia ir com Lelé
 - Joca gostaria de ir com Loló, mas poderia ir com Lili
 - Jeca não abre mão de ir com Loló
- É possível montar duplas rapaz/moça de modo a atender todos os interesses, se não com a primeira opção, com a segunda opção dos rapazes?

- A figura abaixo modela a situação:

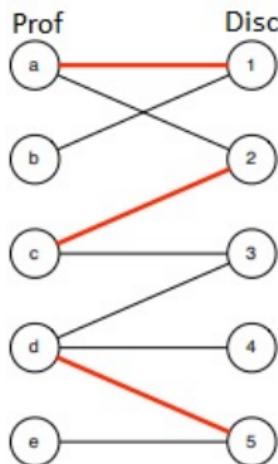


- Essa situação, de encontrar um conjunto de arestas que associe a cada vértice de um conjunto um vértice de outro conjunto de modo que cada vértice seja adjacente no máximo a uma aresta aparece em várias situações:
 - Associar tarefas a pessoas habilitadas a faze-las
 - Professores a disciplinas
 - Monitores a salas ou horários
 - Verificação de similaridades (pontos de uma impressão digital com uma base de dados)

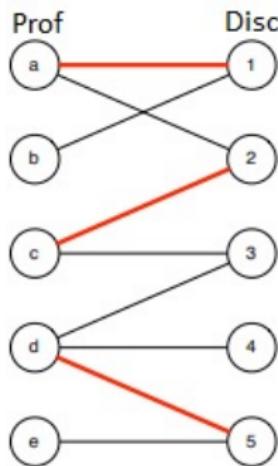
- Suponha que temos uma lista de professores e disciplinas a alocar
- Cada professor está capacitado a ministrar algumas disciplinas
- Pode ser modelado como um grafo bipartido



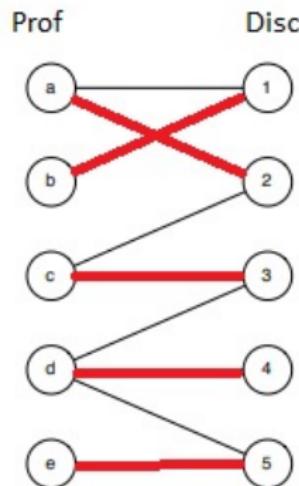
- Uma **associação** (matching) associa a cada professor uma disciplina
- Deseja-se associar o máximo de disciplinas a professores
- Busca-se uma **associação máxima**
- A associação ao lado não é máxima



- Uma associação é dita **maximal** se não é possível acrescentar arestas à mesma sem que ela deixe de ser uma associação (conjunto de arestas não adjacentes entre si)
- A associação ao lado é maximal, mas não é máxima



- Uma associação é dita **perfeita** se todos os vértices de ambos os conjuntos são adjacentes a alguma aresta.
- A associação ao lado é perfeita



- Uma **associação máxima** (maximum matching) em um grafo bipartido de vértices $V = X \cup Y$ é um conjunto de arestas máximo em que cada vértice é adjacente a no máximo uma aresta. Por exemplo, considere o grafo abaixo:

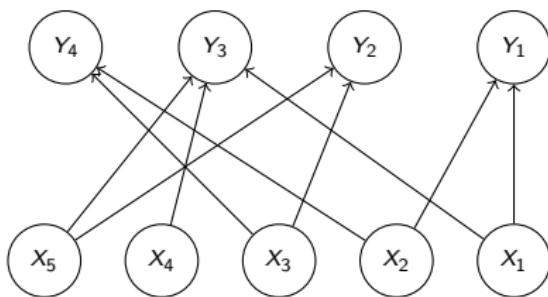
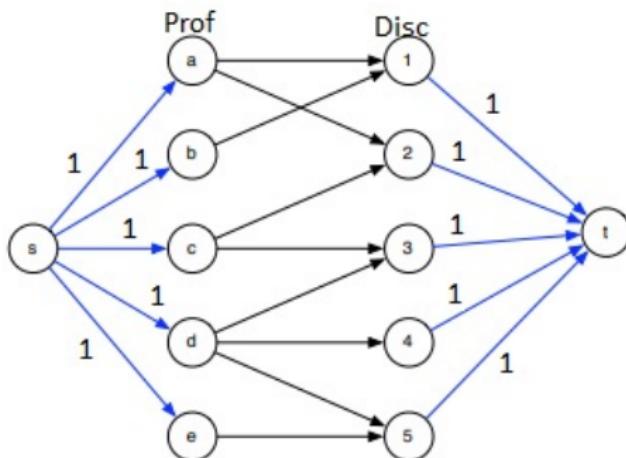


Figura: Problema de associação máxima

- Um problema de associação máxima em grafo bipartido pode ser resolvido transformando-o em um problema de fluxo máximo em redes, como um problema produtor x consumidor.



- O número de arestas em uma associação máxima em um grafo G bipartite é igual ao fluxo máximo de uma rede correspondente $N(G)$ onde foram acrescentados dois vértices s e t , e arestas de s a cada vértice de X e arestas de cada vértice de Y a t , de forma análoga a um modelo produtor x consumidor.
- Atribuindo-se capacidade máxima de fluxo 1 a cada aresta, a associação máxima é dada pelas arestas (x, y) que entram no fluxo máximo da rede equivalente. A rede $N(G)$ equivalente ao grafo ao lado é mostrada abaixo, bem como a associação máxima.

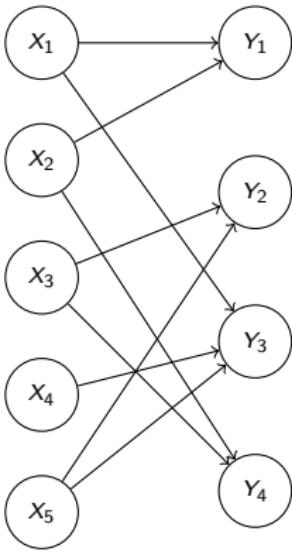


Figura: Rede correspondente ao grafo

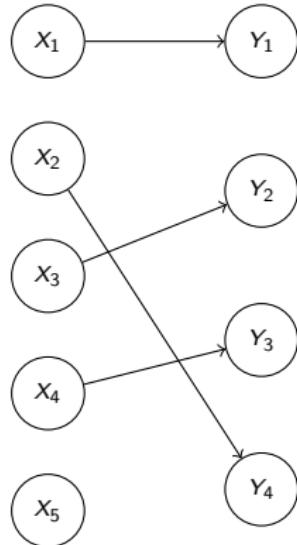


Figura: Associação máxima

- Ex. A UCS dispõe de duas salas para monitoria, G41 e G42, cada uma podendo ser utilizada em 4 horários diferentes, segunda-feira no vespertino, segunda-feira à noite, terça feira no vespertino e sábado pela manhã. Os monitores André, Emilia, Julio e Ionara precisam de dois horários cada um para dar atendimento. A disponibilidade de horários de cada um é dada pela tabela a seguir:

	Seg.Vesp.	Seg.Noite	Ter.Vesp.	Sab.Manhã
André	X		X	X
Emilia	X	X		
Julio		X		X
Ionara			X	X

- Deve-se procurar uma atribuição de salas e horários aos monitores respeitando sua disponibilidade de horários e considerando que cada sala só pode ser utilizada por um monitor em cada horário.
- Explique como esse problema (e qualquer outro problema desse tipo) pode ser resolvido utilizando teoria de grafos e mostre um grafo que modela esse problema. E no caso em que as salas têm horários de funcionamento diferentes?

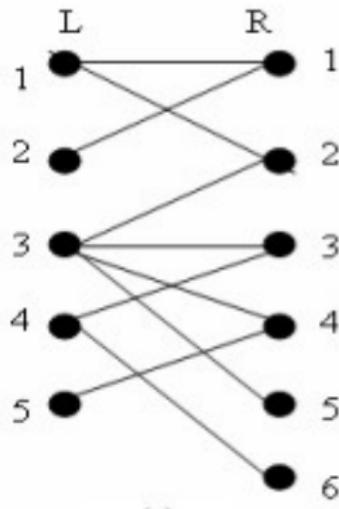
[Voltar para o índice](#)

[Exercícios](#)

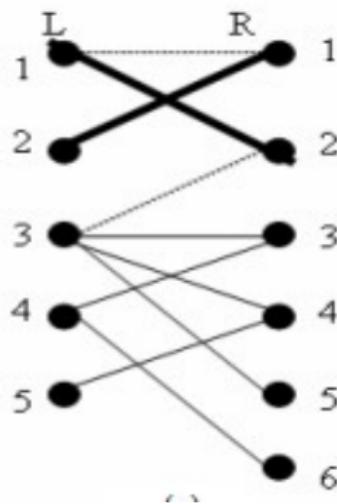
Algoritmo de Hopcroft-Karp para Associação Máxima em Grafos Bipartidos

- Na identificação de associação máxima em grafos bipartidos por Ford-Fulkerson, a cada iteração é encontrado um caminho aumentante e o número de arestas da associação é incrementado de 1 a cada iteração, resultando em um algoritmo de custo $O(|V|.|E|)$.
- O algoritmo de Hopcroft-Karp reduz a complexidade do algoritmo para $O(|E|\sqrt{|V|})$ encontrando vários caminhos aumentantes a cada iteração.

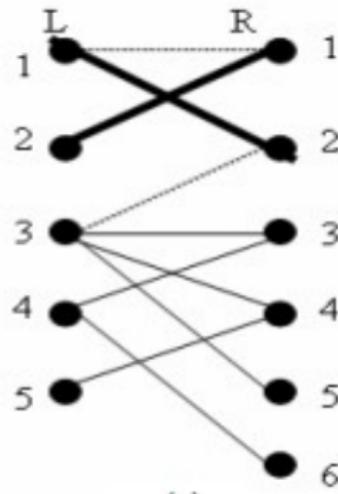
- Algumas definições:
- Considere um grafo bipartido formado por dois conjuntos U e V de vértices.



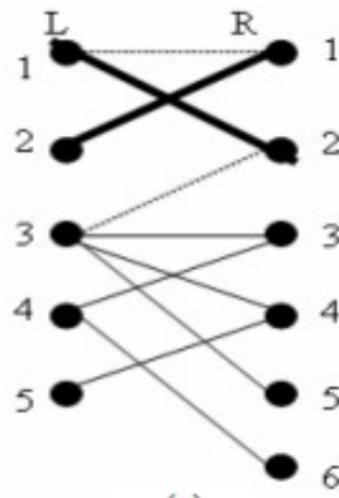
- Uma associação é um subconjunto de arestas independentes, ou seja, que não compartilham nenhum vértice em comum.
- No grafo ao lado, as arestas mais escuras compõem uma associação.



- Um **vértice livre** é um vértice que não é adjacente a nenhuma aresta da associação.
 - No grafo ao lado, os vértices 3, 4 e 5 do conjunto L são vértices livres.
 - Aos vértices adjacentes às arestas da associação chama-se **vértices associados**.
 - No grafo ao lado, os vértices 1 e 2 do conjunto L e os vértices 1 e 2 do conjunto R são vértices associados.



- Um **caminho extensor** (ou **caminho aumentante**) é um caminho que inicia e termina em um vértice livre, iniciando por uma aresta não associada, e alternadamente passando por arestas associadas e não associadas.
- Um caminho extensor pode ser constituído de uma única aresta ligando um vértice livre em L a um vértice livre em R.
- No grafo ao lado, cada aresta ligando dois vértices livres é um caminho extensor.



- No algoritmo de Hopcroft-Karp busca-se, a cada passo, todos os caminhos aumentantes de comprimento mínimo.
- Isso pode ser obtido a partir de uma busca em largura iniciando por todos os vértices livres em L, até alcançar algum vértice livre em R.
- Todos os vértices percorridos nessa busca são marcados com o nível obtido na bfs.
- A busca pode ser implementada por uma fila, em que no início todos os vértices livres de L são colocados na fila.

- No pseudo-código apresentado na wikipedia, o grafo é composto de dois conjuntos de vértices U e V , e para cada vértice em U é guardada a sua lista de adjacências.
- A associação é guardada nos vetores Pair_u e Pair_v , onde para cada vértice é armazenado o vértice a ele ligado pela associação.
- A seguir, uma tentativa de explicar o pseudo-código da Wikipedia:

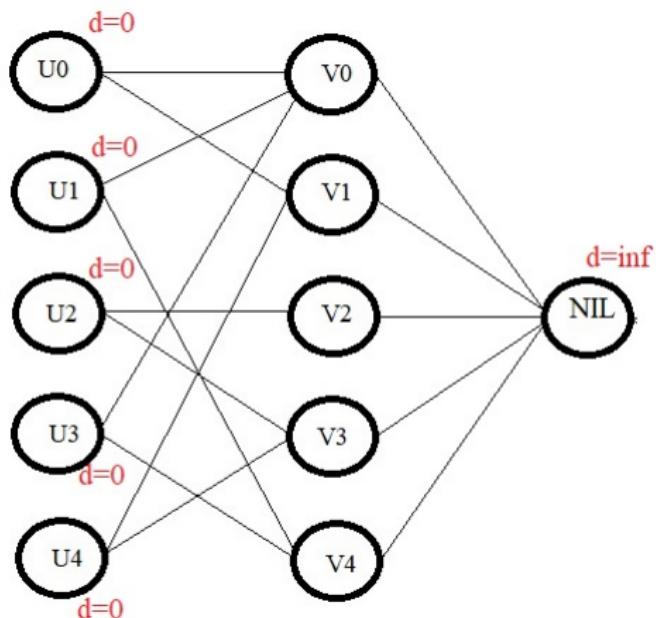
```
1 function Hopcroft-Karp is
2     for each u in U do
3         Pair_U[u] := NIL
4     for each v in V do
5         Pair_V[v] := NIL
6     matching := 0
7     while BFS() = true do
8         for each u in U do
9             if Pair_U[u] = NIL then
10                if DFS(u) = true then
11                    matching := matching + 1
12    return matching
```

- O algoritmo inicia marcando todos os vértices de ambos os lados (U e V) como não associados (linhas 2 a 5)
- **matching** é o contador de arestas da associação máxima
- A busca em amplitude ($BFS()$) varre o grafo até encontrar um vértice livre no conjunto V (procura todos os caminhos aumentantes) de tamanho mínimo, marcando cada vértice com o nível em que foi encontrado.
- Se foi encontrado um caminho aumentante, é efetuada uma busca em profundidade a partir de cada vértice livre seguindo os níveis definidos no BFS , a procura de um caminho aumentante.
- Em cada caminho aumentante encontrado é aplicada a diferença simétrica e o tamanho da associação aumenta em uma unidade.

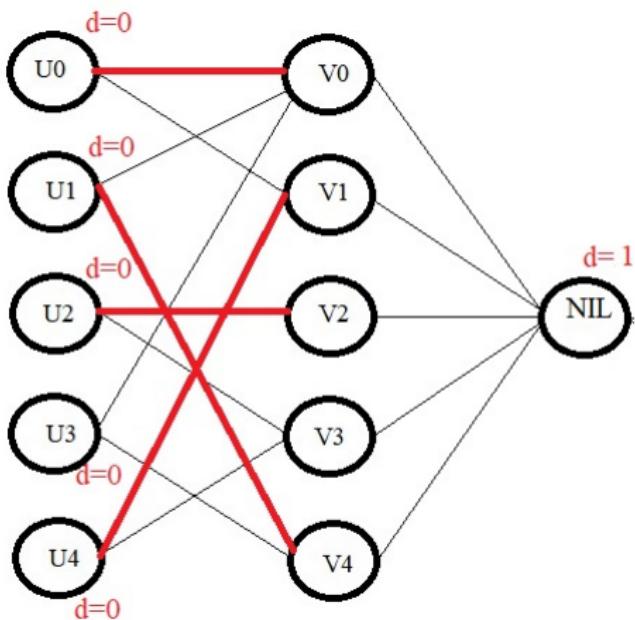
```
1 function BFS() is
2     for each u in U do
3         if Pair_U[u] = NIL then
4             Dist[u] := 0
5             Enqueue(Q, u)
6         else
7             Dist[u] :=  $\infty$ 
8         Dist[NIL] :=  $\infty$ 
9         while Empty(Q) = false do
10            u := Dequeue(Q)
11            if Dist[u] < Dist[NIL] then
12                for each v in Adj[u] do
13                    if Dist[Pair_V[v]] =  $\infty$  then
14                        Dist[Pair_V[v]] := Dist[u] + 1
15                        Enqueue(Q, Pair_V[v])
16        return Dist[NIL]  $\neq \infty$ 
```

```
1 function DFS(u) is
2     if u ≠ NIL then
3         for each v in Adj[u] do
4             if Dist[Pair_V[v]] = Dist[u] + 1 then
5                 if DFS(Pair_V[v]) = true then
6                     Pair_V[v] := u
7                     Pair_U[u] := v
8                     return true
9             Dist[u] := ∞
10            return false
11        return true
```

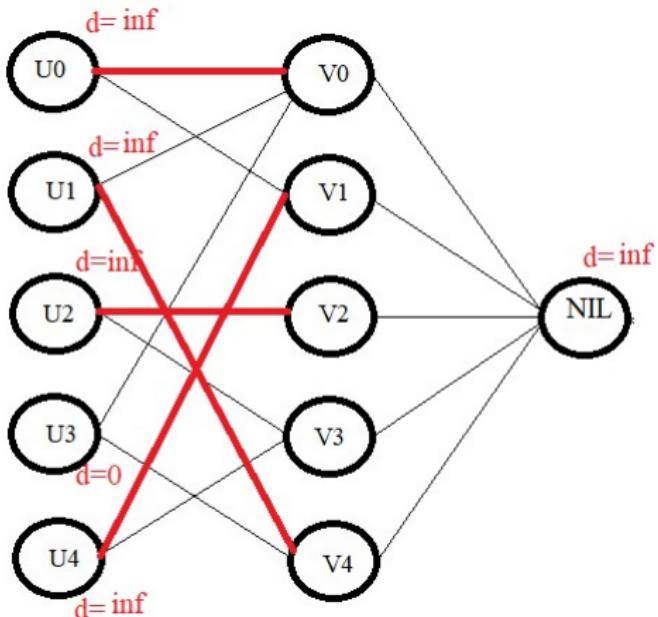
- A figura abaixo mostra a inicialização no BFS. E a fila conterá $Q = \{U_0, U_1, U_2, U_3, U_4\}$



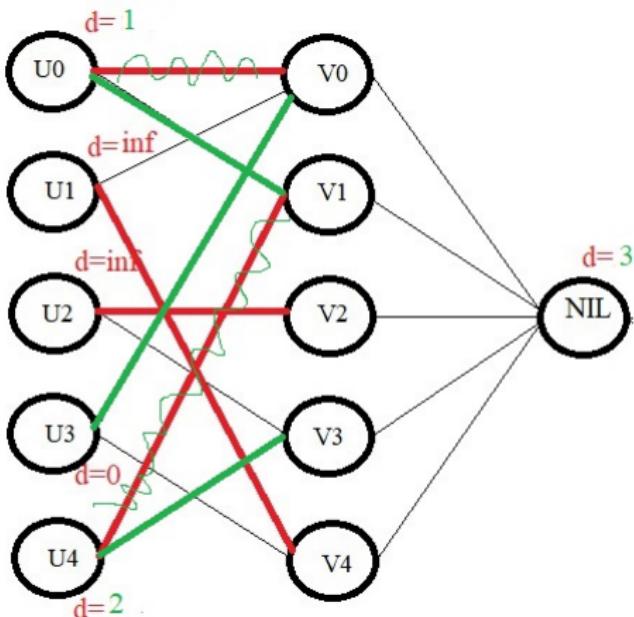
- A figura abaixo mostra a associação após o primeiro passo (BFS+DFS)



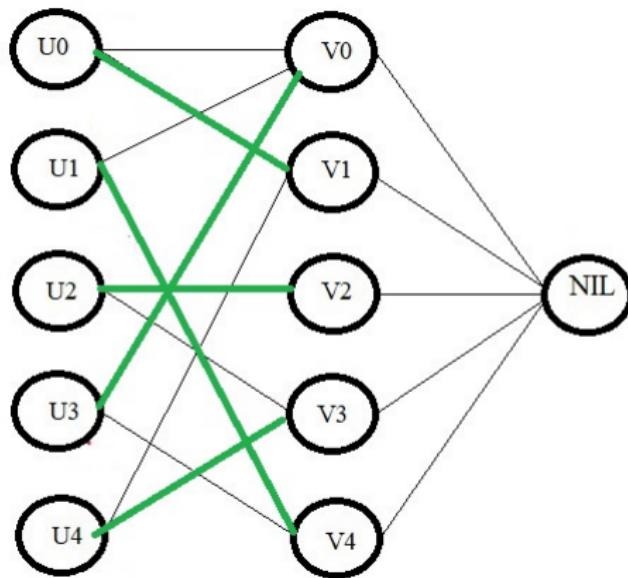
- A figura abaixo mostra a inicialização do BFS no segundo passo



- A figura abaixo mostra ao final do segundo BFS. As arestas vermelhas riscadas foram retiradas da associação anterior, as verdes passaram a ser associadas.



- E a associação final:



Associação máxima em grafos não bipartidos

- Os conceitos de associação máxima em grafos bipartidos também podem ser aplicados a grafos não bipartidos.
- **Def:** Uma associação (**matching**) em um grafo é um conjunto de arestas, nenhuma das quais é adjacente a outra.
- Uma **associação maximal** é uma associação que não está propriamente contida em outra.
- Uma **associação máxima (maximum matching)** é uma associação de máxima cardinalidade.
- Uma **associação perfeita** é uma associação que cobre todos os vértices.

[Voltar para o índice](#)

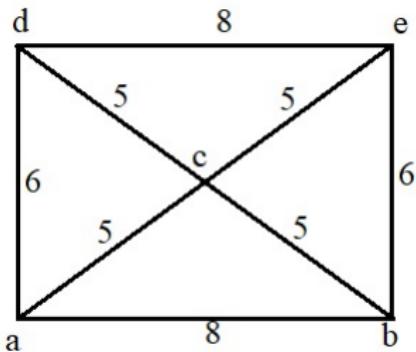
- Os amigos André, Ricardo, Luis, Daniel, Eliseo e Mauricio sempre se encontram para jogar conversa fora e às vezes jogar damas, xadrez e dominó. As preferências de cada um são as seguintes:
 - André só joga xadrez;
 - Ricardo joga xadrez e damas;
 - Luis joga damas e dominó;
 - Daniel só joga dominó;
 - Eliseo só joga xadrez;
 - Mauricio joga damas e dominó.
- É possível encontrar um emparelhamento dos 6 amigos em 3 duplas respeitando suas preferências? Em caso afirmativo, dê uma solução para o problema. Como esse problema pode ser modelado e resolvido com teoria de grafos?

- Ex2: Organizar o emparceiramento em um torneio de xadrez:
 - A cada passo pode-se buscar o emparceiramento de forma a minimizar ou maximizar a força dos oponentes (associação máxima de peso mínimo ou peso máximo).

- A identificação de uma associação máxima em grafos não bipartidos aparece como parte da solução de vários problemas em teoria de grafos.
 - Por exemplo, uma parte importante do algoritmo de Christofides, para encontrar uma aproximação do problema do caixeiro viajante é a identificação de uma associação máxima entre os vértices de grau ímpar.

- Outra aplicação do algoritmo de associação máxima em grafos não bipartidos é o problema do carteiro chinês, que consiste em encontrar o menor custo para percorrer todas as arestas de um grafo.
- Se o grafo é euleriano, qualquer caminho euleriano é a solução.
- Se o grafo é não-euleriano, pode-se montar um grafo completo com todos os vértices que tem grau ímpar no grafo original, e arestas com valores iguais às distâncias entre os vértices, e procurar uma associação máxima de custo mínimo.

- No grafo abaixo, os vértices **a**, **b**, **d** e **e** tem grau ímpar (3).
- E pode-se fazer os empareiramentos embaixo à direita
- O custo corresponde à distância entre os vértices de cada par.
- Após obter a associação de menor custo, duplica-se as arestas da associação tornando o grafo euleriano.



pares	custo
ab,de	8+8
ad,be	6+6
ae,bd	10+10

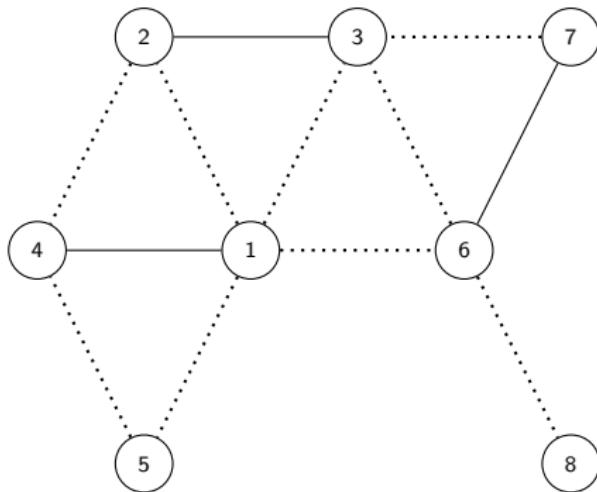


Figura: As três arestas contínuas formam uma associação

- **Def:** Um vértice é dito **livre** em relação a uma associação se ele não é adjacente a nenhuma aresta. No exemplo acima, os vértices 5 e 8 são livres.
- **Def:** Um caminho é dito **alternante** em relação a uma associação M se suas arestas estão alternadamente em M e não em M .

Existem três tipos de caminhos alternantes:



Figura: 1. Um caminho alternante entre dois vértices associados



Figura: 2. Um caminho alternante entre dois vértices livres



Figura: 3. Um caminho alternante entre um vértice associado e um vértice livre

Def: Um **caminho aumentante** é um caminho simples alternante entre dois vértices livres.

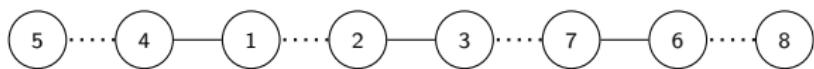


Figura: Um caminho aumentante no grafo anterior

Teorema (Berge's Lemma): M não é uma associação máxima se e somente se existe um caminho aumentante com relação a M .

Como resultado do teorema acima, pode-se chegar no algoritmo a seguir:

Algoritmo 1: Associação Máxima(G)

1 **início**

2 $M \leftarrow \emptyset$

3 **repita**

4 **se** *há um caminho aumentante sobre M* **então**

5 Faça P o caminho aumentante sobre M

6 $M \leftarrow M \oplus P$

7 **senão**

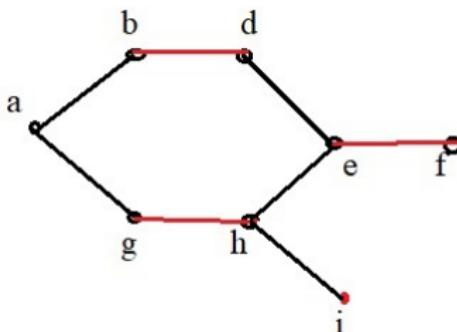
8 **retorne** M

9 **até** ;

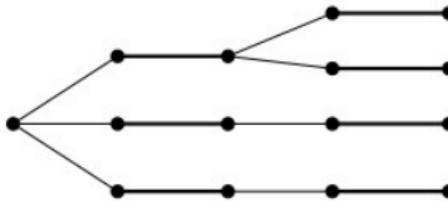
10 **fim**

- onde a operação \oplus representa a **diferença simétrica** entre M e P (é um ou exclusivo, entram as arestas que estão somente em P ou somente em M).
 - O que equivale a inverter o estado de todas as arestas em P .
- O caminho aumentante pode ser buscado por uma busca em amplitude ou em profundidade alterada para, a cada nível, alternar entre arestas livres (nos vértices de nível par) e arestas associadas (nos vértices de nível ímpar).
- Se o grafo é uma árvore, a DFS pode encontrar os caminhos aumentantes de forma eficiente.

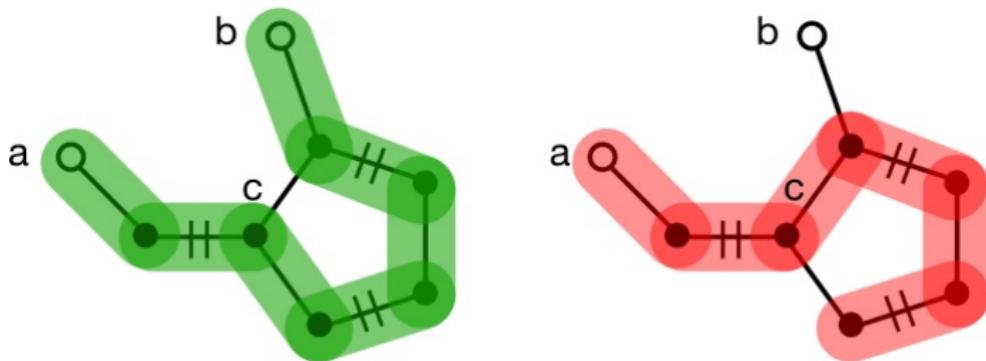
- Se o grafo tem ciclos, a busca em profundidade pode aumentar o custo computacional ao fazer com que um ramo do grafo seja explorado mais de uma vez por caminhos diferentes.
- Na figura abaixo (arestas vermelhas são as associadas) são explorados os caminhos alternantes a-b-d-e-f, a-g-h-e-f e a-g-h-i e o caminho e-f (e o que houvesse depois) é examinado mais de uma vez.



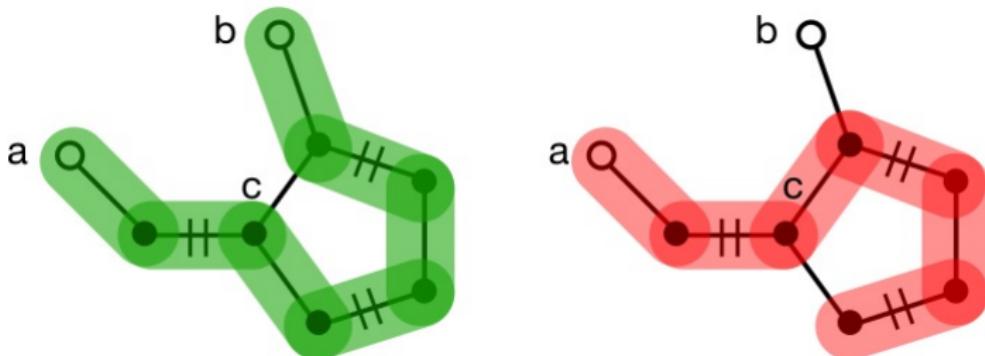
- Uma forma de evitar a repetição de caminhos é utilizar a marcação de vértices já visitados e isso funciona bem se o grafo tem somente ciclos de comprimento par (ou, de forma equivalente, é bipartido).
- Ao invés da DFS pode-se usar uma BFS para explorar todos os caminhos alternantes a partir de cada vértice livre, como na figura abaixo.
- A busca a partir de cada vértice gera uma árvore
M-alternante



- Se o grafo tem somente ciclos de comprimento par, o algoritmo descrito acima encontra caminhos aumentantes que utilizem arestas do ciclo.
- Se o grafo tem ciclos de comprimento ímpar, a situação abaixo pode ocorrer.



- A figura ilustra a busca de um caminho aumentante a partir do vértice **a**.
- Dependendo do caminho seguido a partir do vértice **c**, o caminho aumentante até **b** pode ser encontrado ou não.
- Essa situação ocorre somente em ciclos de comprimento ímpar, em que o ponto de saída do ciclo para o final do caminho aumentante pode ser alcançado tanto por uma aresta livre quanto por uma aresta associada.



- Em 1965 Jack Edmonds propôs a primeira abordagem eficiente para o problema do caminho aumentante através de ciclos ímpares.
- Jack Edmonds tem importantes contribuições à análise de tratabilidade ($P=NP?$) e foi o primeiro a usar o termo "polinomial" para algoritmos eficientes.



Figura: "Jack Edmonds"

- O passo chave do algoritmo "blossom" é o colapso do ciclo ímpar (o "blossom") em um único vértice, que é atravessado pelo caminho alternante.
- Após, o "blossom" é expandido, bem como o caminho aumentante.

ENTRADA: Grafo G , associação M sobre G

SAÍDA: caminho aumentante P em G ou caminho vazio se não houver caminho aumentante
função $\text{encontra_caminho_aumentante}(G, M) : P$

$F \leftarrow$ floresta vazia

desmarque todos os vértices e arestas em G , marque todas as arestas de M
para cada vértice livre v faça

crie uma árvore com $\{v\}$ e adicione a árvore a F

fimpara

enquanto houver um vértice não marcado v em F com distância ($v, \text{raiz}(v)$) par faça

enquanto houver uma aresta não marcada $e = \{v, w\}$ faça

se w não está em F então // w é associado, então adicione e e o vértice associado a w a F

$x \leftarrow$ vértice associado a w em M

adicione arestas $\{v, w\}$ e $\{w, x\}$ à árvore de v

senão

se distância ($w, \text{raiz}(w)$) é ímpar então // não faz nada

senão

se $\text{raiz}(v) \neq \text{raiz}(w)$ então // Encontrou um caminho aumentante em $F \cup e$.

$P \leftarrow$ caminho ($\text{raiz}(v) \rightarrow \dots \rightarrow v \rightarrow (w \rightarrow \dots \rightarrow \text{raiz}(w))$)

retorne P

senão // Contrae um blossom em G e procura um caminho no grafo contraído.

$B \leftarrow$ blossom formado por e e arestas no caminho $v \rightarrow w$ em T

$G', M' \leftarrow G$ e M após contração de B

$P' \leftarrow \text{encontra_caminho_aumentante}(G', M')$

$P \leftarrow$ expansão de P' em G

retorne P

fimse

fimse

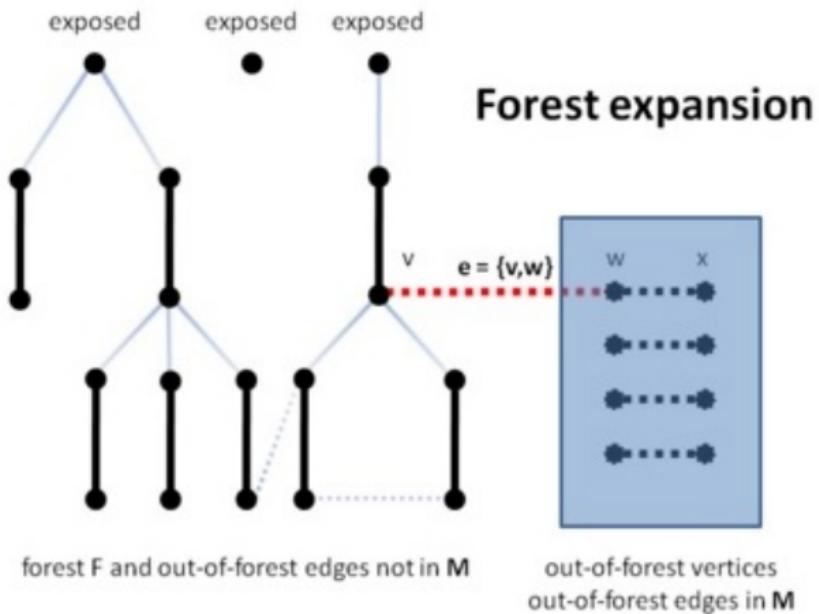
fimse

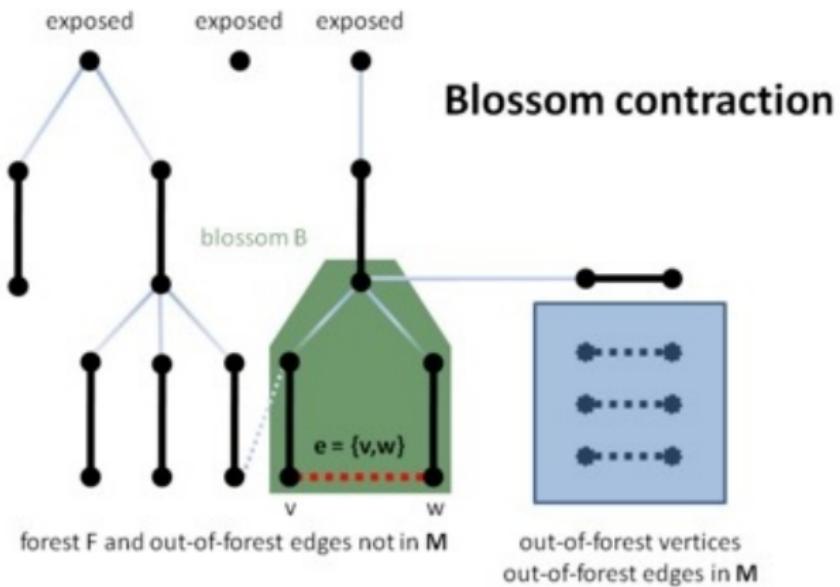
marque a aresta e

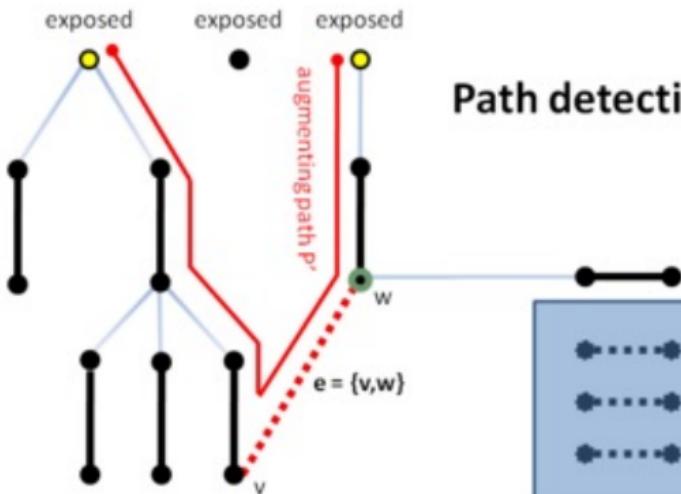
fim enquanto

marque o vértice v

fim enquanto





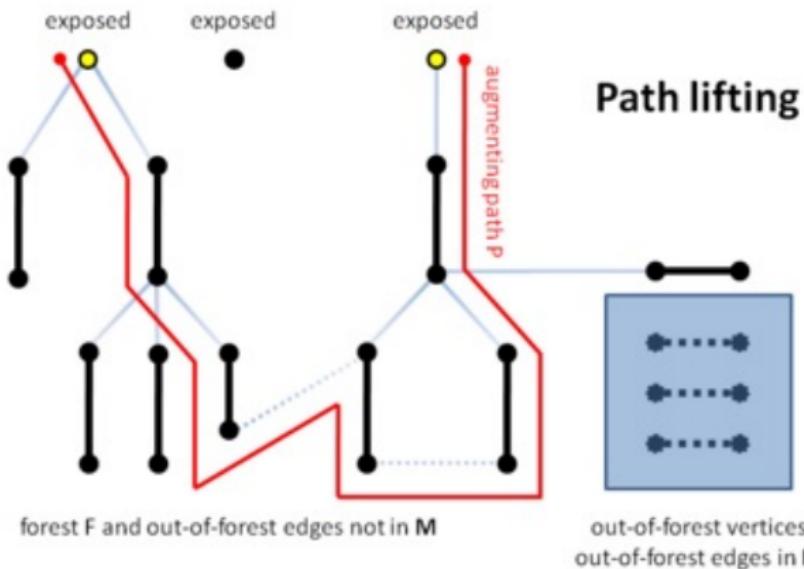


Path detection in G'

forest F' in G' and out-of-forest edges not in M'

out-of-forest vertices

out-of-forest edges in M'



Tópicos da Segunda Área

- Fluxo em Redes - Problemas Produtor x Consumidor
- Associação Máxima em grafos bipartidos e não bipartidos
- PERT/CPM
- Ordenação Topológica
- Caixeiro Viajante com programação dinâmica e soluções aproximativas pelo método guloso
- Cobertura de Vértices, Cobertura de Arestas e Conjunto dominante de vértices
- Identificação de componentes fortemente conexas
- Número cromático por alteração estrutural

Técnicas sobre grafos

Existem diversas técnicas de desenvolvimento de algoritmos que podem ser aplicadas a problemas envolvendo grafos. Algumas destas técnicas são:

- 1 Algoritmo guloso (greedy)
- 2 Alteração estrutural
- 3 Programação dinâmica

Aplicações:

- 1 Obter uma árvore geradora máxima
- 2 Determinar o número cromático
- 3 Particionar uma árvore de forma ótima

Ciclo Hamiltoniano e o Problema do Caixeiro Viajante

- Um **caminho hamiltoniano** é um caminho que percorre todos os vértices de um grafo, dirigido ou não, exatamente uma vez.
- O nome Hamiltoniano vem de William Hamilton, que estudou a existência desses ciclos em grafos no século XIX.

Sir William Rowan Hamilton



Sir William Rowan Hamilton (1805–1865)

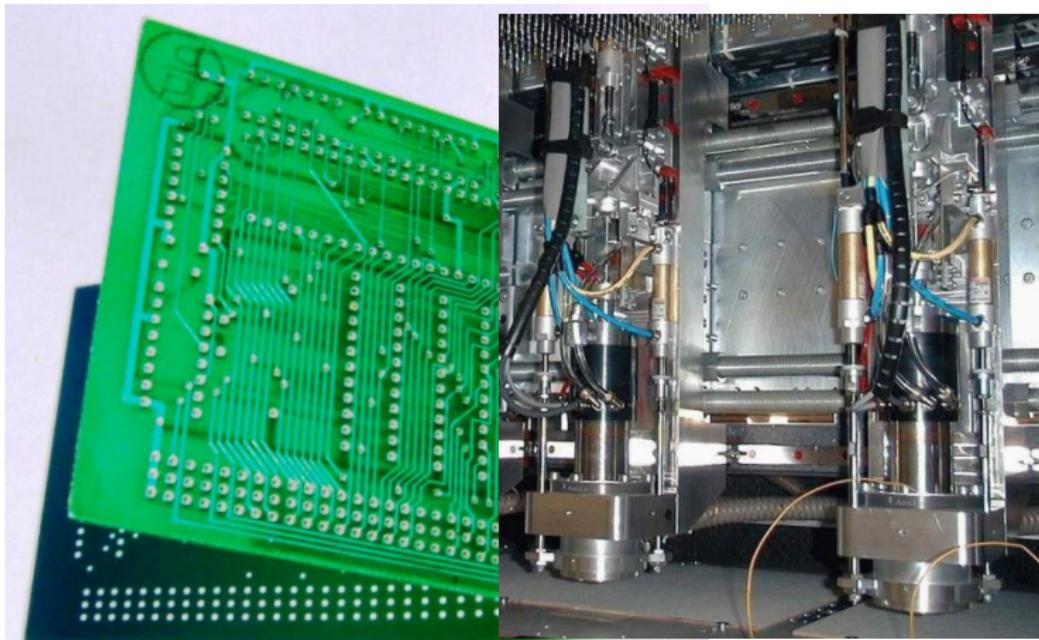
- Se o caminho inicia e termina no mesmo vértice, é dito um **ciclo hamiltoniano**. Um grafo que contém um ciclo hamiltoniano é dito um **grafo hamiltoniano**.
- Apesar da semelhança com o problema de identificar ciclos e caminhos eulerianos (que passam exatamente uma vez em cada aresta) que possuem algoritmos simples e de custo linear, a identificação de ciclos hamiltonianos ou a verificação de sua existência não é trivial, não havendo algoritmos polinomiais genéricos.
- Foi visto que um grafo contém um ciclo euleriano se e somente se todos os vértices tem grau par.
- Se o grafo tiver 2 vértices de grau ímpar, ele não tem ciclo euleriano, mas tem um caminho euleriano que inicia em um dos vértices de grau ímpar e termina no outro vértice de grau ímpar.

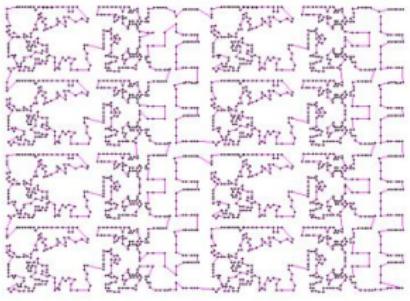
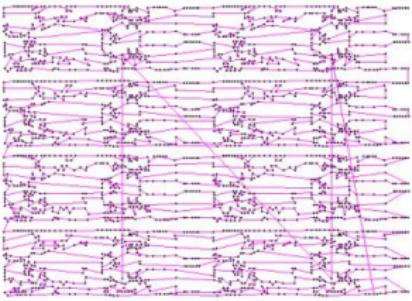
- Não há uma condição simples de verificar se um grafo é hamiltoniano ou não em todos os casos, mas há algumas condições suficientes (mas não necessariamente necessárias) para um grafo ser hamiltoniano ou não:
- Condição de **Ore**: Seja **G** um grafo de **n** vértices, onde $d(v)$ representa o grau do vértice v . Se para todo par de vértices não-adjacentes **u** e **v**, $d(u)+d(v)\geq n$, o grafo é hamiltoniano.

- Se o grafo é um DAG (grafo acíclico dirigido), um **caminho hamiltoniano** pode ser encontrado fazendo uma ordenação topológica (que será única) e verificando se cada par de vértices contíguos na ordenação topológica possuem uma aresta entre eles.
- Um problema relacionado ao problema do ciclo hamiltoniano, e de maior aplicação prática, é o **Problema do Caixeiro Viajante** (Traveling Salesman Problem - TSP).
- Esse problema consiste, em um grafo com valores associados às arestas, encontrar o ciclo de menor custo total.

- Algumas aplicações desse problema são:
 - Considere um caixeiro viajante que deve percorrer n cidades. Ele deve encontrar a seqüência a percorrer mais econômica entre as diversas possíveis.
 - Um caminhão de uma empresa de entregas deve entregar vários produtos ao longo do dia. Qual a ordem das entregas que minimiza o tempo / custo / distância total?
 - Considere um conjunto de máquinas que fabricam n produtos distintos de forma cíclica. Suponha que todas as máquinas podem estar fabricando um produto i e devem ser alteradas para produzir o produto j , pagando-se um custo c_{ij} . É o chamado custo de *setup* da máquina. Deseja-se determinar a ordem em que os produtos devem ser fabricados de forma a minimizar a soma dos custos de *setup* ex.

- Perfuração de placas de circuitos impressos. Qual a ordem de perfuração para minimizar o tempo total?





Algoritmo guloso aplicado ao problema do caixeiro viajante

- Como foi mencionado, não existe um algoritmo eficiente (polinomial) para encontrar a solução ótima para o Problema do Caixeiro Viajante.
- A verificação de todos os caminhos possíveis é inviável mesmo para grafos pequenos. Para um grafo com N vértices, o número de caminhos possíveis é da ordem de $(N-1)!$ (fatorial).
 - $10! = 3.628.800$
 - $11! = 39.916.800$
 - $12! = 479.001.600$
 - $13! = 6.227.020.800$ (6 bilhões de possibilidades?)
 - $14! = 87.178.291.200$ (87 bilhões?)
- Em todo caso, se o grafo for pequeno a solução pode ser obtida por uma busca em profundidade modificada para explorar todos os caminhos.

- Na impossibilidade de encontrar uma solução ótima, se uma solução aproximada é aceitável, algumas heurísticas podem ser utilizadas.
- Uma heurística que pode ser utilizada, se o grafo é completo, é a do **vizinho mais próximo** (**nearest neighbour**), que consiste em:
 - Começando do vértice v_1 , busca-se a aresta de menor valor adjacente a v_1 indo para v_2 .
 - A partir de v_2 busca-se a aresta de menor valor adjacente a v_2 que vá para um vértice ainda não visitado.
 - Repete-se isso até visitar todos os vértices, voltando-se então ao vértice inicial.
- Para cidades uniformemente distribuídas, essa abordagem leva a uma solução, em média, apenas 25% maior do que a solução ótima.

- Outra alternativa é escolher a cada passo, dentre todas as arestas, a aresta de menor peso que:
 - 1 Não cause um vértice a ter grau 3 ou mais
 - 2 Não forme um ciclo, a não ser quando o número de arestas selecionadas for igual ao número de vértices do problema
- Ambas heurísticas podem gerar soluções próximas à ótima, ou mesmo a ótima, mas também podem gerar soluções arbitrariamente ruins, dependendo dos pesos das arestas envolvidas.

Algoritmos Gulosos (Greedy)

- Ambas as heurísticas descritas acima se enquadram no paradigma chamado "algoritmo **guloso**" (**greedy**), que se caracteriza pela construção iterativa da solução, selecionando a cada passo da construção um elemento.
- Para alguns problemas a abordagem gulosa pode conduzir à solução ótima. Alguns algoritmos em que isso ocorre são:
 - Os algoritmos de Prim e Kruskal para obtenção da árvore geradora mínima.
 - O algoritmo de Dijkstra, para cálculo da distância entre dois vértices.
- Para outros problemas, como o do Cai xeiro Viajante, ele pode ser utilizado para obter uma solução que não é necessariamente (e provavelmente) a ótima.

Funções rand() e srand ()

- Para gerar valores randômicos utiliza-se a função rand(), da biblioteca stdlib.h.
- Cada chamada da função rand() retorna um valor inteiro entre 0 e RAND_MAX (constante definida em stdlib).
- Para inicializar o gerador de números randômicos no início do programa chama-se a função srand (time(NULL));

Ex:

```
#include < stdio.h >
#include < stdlib.h >
#include < time.h >
int main ()
{
    srand ( time(NULL) );
    printf ("Um numero entre 0 e RAND_MAX (%d): %d\n", RAND_MAX, rand());
    printf ("Um numero entre 0 e 99: %d\n", rand()%100);
    printf ("Um numero entre 20 e 29: %d\n", rand()%10+20);
    return 0;
}
```

- Medição de tempo em trechos de código pode ser feita utilizando a função time() da biblioteca time.h, que retorna o número de segundos transcorridos desde 1/1/1970.
- Pode ser utilizada da seguinte forma:

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    time_t t0, t1;
    t0=time(NULL);

    // trecho de código a ser medido

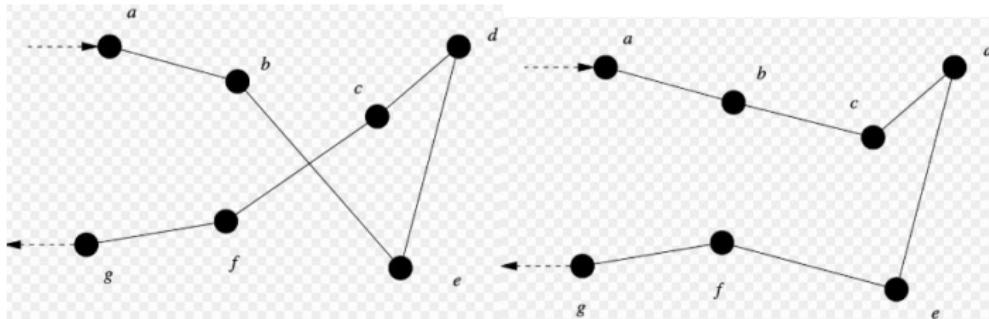
    t1 = time(NULL);
    printf("tempo:%d\n",t1-t0);
}
```

Ex: Aplique as duas heurísticas acima no grafo a seguir:

```
int mat[10][10]={{00, 18, 61, 74, 20, 99, 31, 18, 28, 92},  
                  {15, 00, 31, 78, 54, 87, 35, 85, 32, 68},  
                  {38, 82, 00, 49, 82, 70, 62, 10, 46, 77},  
                  {92, 36, 66, 00, 81, 95, 61, 42, 52, 64},  
                  {44, 73, 32, 14, 00, 70, 49, 49, 25, 97},  
                  {63, 72, 63, 14, 83, 00, 14, 48, 35, 51},  
                  {18, 37, 11, 17, 42, 14, 00, 33, 13, 63},  
                  {86, 75, 25, 41, 49, 51, 35, 00, 47, 68},  
                  {56, 60, 16, 71, 78, 10, 38, 41, 00, 36},  
                  {25, 17, 15, 27, 31, 17, 27, 73, 53, 00}};
```

- As heurísticas acima não garantem a qualidade da solução, podendo resultar em uma solução arbitrariamente ruim. Um outro algoritmo aproximativo é o seguinte:
 - 1 Encontre uma árvore geradora mínima T de G .
 - 2 Efetue uma busca em profundidade em T associando um número $L(v)$ a cada vértice v , correspondendo à ordem de visitação.
 - 3 Retorne o ciclo correspondente à ordem de visitação.

- Algumas estratégias podem ser utilizadas para tentar melhorar a qualidade de uma solução. Uma delas é a 2-opt, em que busca-se conjuntos de 4 vértices em que há cruzamento do caminho entre eles, e reordenar os caminhos de modo a diminuir o custo total no subconjunto de 4 vértices considerados.
- No caminho considerado na figura abaixo, as arestas b-e e f-c podem ser trocadas pelas arestas b-c e f-e com evidente redução no custo total.



Greedy + 2-opt

- A visualização abaixo no youtube (Traveling Salesman Problem Visualization [Botão](#)) mostra o ganho obtido sobre uma solução inicial greedy:



Traveling Salesman Problem Visualization

Algoritmo de Christofides para o TSP

- Para grafos completos que obedecam à desigualdade triangular (para quaisquer 3 vértices a, b, c , $I(a, b) < I(a, c) + I(c, b)$), esse algoritmo garante que a solução obtida não é maior que o dobro da solução ótima. O algoritmo a seguir obtem uma solução garantidamente menor que 1.5 vezes a solução ótima:
 - 1 Encontre uma árvore geradora mínima T de G .
 - 2 Construa o conjunto V' de vértices de grau ímpar em T e encontre uma associação perfeita M de V' .
 - 3 Construa o grafo Euleriano G' obtido pela adição das arestas de M a T (obs: as arestas de M serão acrescentadas a T mesmo que já existam em T . Nesse caso elas ficarão duplicadas).
 - 4 Encontre o ciclo euleriano C_0 de G' , associando um número $L(v)$ a cada vértice v , correspondendo à ordem de sua primeira visitação.
 - 5 Retorne o ciclo correspondente à ordem de visitação.

Ex: Aplique os 2 algoritmos vistos no grafo a seguir: (a solução ótima para este grafo é 220. Ciclo 0-4-8-9-2-7-6-5-3-1)

```
int mat[10][10]={{00, 18, 61, 74, 20, 99, 31, 18, 28, 92},  
                 {15, 00, 31, 78, 54, 87, 35, 85, 32, 68},  
                 {38, 82, 00, 49, 82, 70, 62, 10, 46, 77},  
                 {92, 36, 66, 00, 81, 95, 61, 42, 52, 64},  
                 {44, 73, 32, 14, 00, 70, 49, 49, 25, 97},  
                 {63, 72, 63, 14, 83, 00, 14, 48, 35, 51},  
                 {18, 37, 11, 17, 42, 14, 00, 33, 13, 63},  
                 {86, 75, 25, 41, 49, 51, 35, 00, 47, 68},  
                 {56, 60, 16, 71, 78, 10, 38, 41, 00, 36},  
                 {25, 17, 15, 27, 31, 17, 27, 73, 53, 00}};
```

Ex: Implemente uma função em C para encontrar a solução ótima para uma matriz 10x10 usando backtracking. Sugestão de parâmetros:

```
int TSP_back(int inicio, // vértice a partir de onde
              int nível, // nível de profundidade da
              int soma) // soma parcial até a chama
```

```
#include <stdio.h>
#include <conio.h>
int mat[10][10]={{00, 18, 61, 74, 20, 99, 31, 18, 28, 92},
                  {15, 00, 31, 78, 54, 87, 35, 85, 32, 68},
                  {38, 82, 00, 49, 82, 70, 62, 10, 46, 77},
                  {92, 36, 66, 00, 81, 95, 61, 42, 52, 64},
                  {44, 73, 32, 14, 00, 70, 49, 49, 25, 97},
                  {63, 72, 63, 14, 83, 00, 14, 48, 35, 51},
                  {18, 37, 11, 17, 42, 14, 00, 33, 13, 63},
                  {86, 75, 25, 41, 49, 51, 35, 00, 47, 68},
                  {56, 60, 16, 71, 78, 10, 38, 41, 00, 36},
                  {25, 17, 15, 27, 31, 17, 27, 73, 53, 00}};
```

```

int v[10]={0,0,0,0,0,0,0,0,0,0},ordem[10]={0,0,0,0,0,0,0,0,0,0};
int otima=1000;
int TSP_back(int inicio ,int nivel ,int soma)
{
    int i ,prim ,melhor ,aux;
    ordem[nivel]=inicio ;
    if (nivel==9)
    {
        if (soma+mat[inicio ][0]<otima )
        {
            otima=soma+mat[inicio ][0];
            for (i=0;i<10;i++)
                printf("%d_ ",ordem[i]);
            printf(" valor=%d\n" ,otima );
        }
        return soma+mat[inicio ][0];
    }
    prim=1;
    for (i=1;i<10;i++)
    {
        if (i!=inicio && v[i]==0)
        {
            v[i]=1;
            aux=TSP_back(i ,nivel+1,soma+mat[inicio ][i]);
            v[i]=0;
            if (prim) {melhor=aux ;prim=0;}
            else if (aux<melhor) melhor=aux ;
        }
    }
    return melhor;
}

```

```
int main()
{
printf ("%d\n" ,TSP_back (0 ,0 ,0));
getch ();
}
```

Programação Dinâmica

- Quando uma seqüência de decisões locais (como algoritmos gulosos) não gera uma solução ótima, podemos gerar todas as seqüências de decisão e escolher a melhor. Isto é ineficiente. A programação dinâmica é uma forma de diminuir o número de seqüências apoiando-se o princípio de otimalidade.

Princípio de Optimalidade

- Seja S_i o estado de um problema após a i -ésima decisão e seja $\Gamma_0 = r_1 r_2 \dots r_n$ uma seqüência ótima de decisão quando o problema está no estado inicial S_0 .
- Então $\Gamma_1 = r_2 r_3 \dots r_n$ é uma seqüência ótima com respeito a S_1 .
- Ex: Seja $i, i_1, i_2, \dots, i_k, j$ um menor caminho entre i e j . Então i_2, i_3, \dots, i_k, j é um menor caminho entre i_1 e j .

Programação Dinâmica

- Técnica através da qual a resolução de um problema P é obtida pela decomposição de P em subproblemas de natureza igual ou semelhante a P .
 - A resolução de cada um dos problemas define a resolução de P .
 - Mantém-se uma tabela de armazenamento dos resultados dos subproblemas.

Aplicação: Problema do caixeiro viajante (Terada 91)

- Considere um grafo orientado com custos c_{ij} associados a cada aresta.
- O Problema do caixeiro viajante consiste em encontrar o ciclo hamiltoniano de menor custo.
- Utilizando programação dinâmica é possível obter um algoritmo $O(n^2 \cdot 2^n)$, que apesar de exponencial, é mais rápido que o algoritmo de gerar todos os caminhos possíveis, que é da ordem de fatorial de n .
- Esse algoritmo chama-se algoritmo de Held-Karp, ou Bellman, Held e Karp.
- Considere que o percurso ótimo começa e termina no vértice 1. Qualquer ciclo é constituído de uma aresta $(1, k)$, $2 \leq k \leq n$, e um caminho M de k até 1. Se o ciclo é de custo mínimo, o caminho M deve ser de custo mínimo (princípio da optimalidade).

- Seja $f(i, C)$ o custo do caminho mínimo do vértice i até 1 , e que visita todos os vértices do conjunto C . Do parágrafo anterior tem-se
- $f(1, VG - 1) = \min\{c_{1k} + f(k, VG - \{1, k\})\}(1)$
- Se conhecemos os valores de $f(k, VG - \{1, k\})$ para todo $k, 2 \leq k \leq n$, pela equação acima podemos resolver o PCV.
- Podemos generalizar e obter
- $f(i, C) = \min_j\{c_{ij} + f(j, C - \{j\})\}(2)$

- isto é, o caminho mínimo do vértice i até 1 que passa por todos os vértices em C é constituído por (i,j) e um caminho de j até 1 que visita todos os vértices em C , para uma escolha adequada de j .
 - Os valores de f necessários em (1) para se resolver o PCV podem ser obtidos de (2) de uma forma ascendente, pelo algoritmo a seguir.
- 1 $|C| = 0, f(i, \emptyset) = c_{i1}$, para $1 < i \leq n$.
 - 2 Para $|C| = 1, f(i, \{j\}) = c_{ij} + f(j, \emptyset) = c_{ij} + c_{j1}$, para $1 < i, j \leq n$ e $i \neq j$
 - 3 Para $|C| = 2, 3, \dots, n - 2$, determinam-se os valores de $f(i, C)$ através de (2) utilizando-se os valores de f calculados nas iterações anteriores. Deve-se considerar $1 < i \leq n$ e conjuntos C tais que $C \cap \{1, i\} = \emptyset$

Ex: considere o grafo representado pela matriz de adjacências a seguir:

	1	2	3	4
1	0	2	10	7
2	20	0	9	1
3	1	5	0	15
4	7	12	3	0

Passo 1 - ($|C| = 0$):

$$f(2, \emptyset) = c_{21} = 20$$

$$f(3, \emptyset) = c_{31} = 1$$

$$f(4, \emptyset) = c_{41} = 7$$

Passo 2 - ($|C| = 1$):

$$f(2, \{3\}) = c_{23} + f(3, \emptyset) = c_{23} + c_{31} = 9 + 1 = 10$$

$$f(2, \{4\}) = c_{24} + f(4, \emptyset) = c_{24} + c_{41} = 1 + 7 = 8$$

$$f(3, \{2\}) = c_{32} + f(2, \emptyset) = c_{32} + c_{21} = 5 + 20 = 25$$

$$f(3, \{4\}) = c_{34} + f(4, \emptyset) = c_{34} + c_{41} = 15 + 7 = 22$$

$$f(4, \{2\}) = c_{42} + f(2, \emptyset) = c_{42} + c_{21} = 12 + 20 = 32$$

$$f(4, \{3\}) = c_{43} + f(3, \emptyset) = c_{43} + c_{31} = 3 + 1 = 4$$

Passo 3 - ($|C| = 2$):

$$f(2, \{3, 4\}) = \min(c_{23} + f(3, \{4\}), c_{24} + f(4, \{3\})) = \\ \min(9 + 22, 1 + 4) = 5(j = 4)$$

$$f(3, \{2, 4\}) = \min(c_{32} + f(2, \{4\}), c_{34} + f(4, \{2\})) = \\ \min(5 + 8, 15 + 32) = 13(j = 2)$$

$$f(4, \{2, 3\}) = \min(c_{42} + f(2, \{3\}), c_{43} + f(3, \{2\})) = \\ \min(12 + 10, 3 + 25) = 22(j = 3)$$

Passo 4 - ($|C| = 3$)

$$f(1, \{2, 3, 4\}) = \min(c_{12} + f(2, \{3, 4\}), c_{13} + f(3, \{2, 4\}), c_{14} + \\ f(4, \{2, 3\})) = \min(2 + 5, 10 + 13, 7 + 28) = 7(j = 2)$$

Assim, como $j = 2$ no cálculo de $f(1, \{2, 3, 4\})$ a primeira aresta do caminho mínimo é $(1,2)$. Prosseguindo, como $j = 4$ no cálculo de $f(2, \{3, 4\})$, a Segunda aresta é $(2,4)$. A terceira aresta é $(4,3)$ pois $j = 3$ no cálculo de $f(4, 3)$. Portanto, o ciclo de custo mínimo neste exemplo é $(1, 2, 4, 3, 1)$, com custo 7.

Alteração Estrutural

A técnica de alteração estrutural consiste em aplicar algum tipo de operação em G de modo a transformá-lo em outro grafo $G'(V', E')$. Suponha que se deseja resolver um problema P no grafo G e seja P' o problema equivalente no grafo G' , e que se possa resolver o problema P' . Se a partir da solução de P' se puder extrapolar a solução para P , o problema está resolvido. As alterações mais comuns de ocorrer são:

- Eliminação de vértices ou arestas (ex: demonstração de não planaridade por redução a um grafo de Kuratovski)
- Adição de vértices ou arestas
- Condensação de dois ou mais componentes (subconjuntos de vértices e arestas) em um só

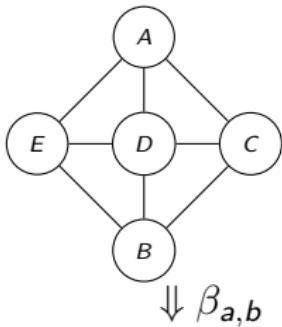
Aplicação : Obtenção do número cromático de um grafo

- Seja $G(V, E)$ um grafo não dirigido. Se G for o grafo completo K_n , o número cromático é igual a n , senão, há dois vértices v e w não adjacentes.
 - Denota-se $\alpha_{v,w}(G)$ ao grafo obtido pela adição da aresta (v, w) a G e
 - $\beta_{v,w}(G)$ ao grafo obtido pela condensação dos vértices v e w em um único vértice z , eliminando as arestas paralelas que aparecerem.
- O número cromático $\chi(G)$ é o mínimo entre $\chi(\alpha_{v,w}(G))$ e $\chi(\beta_{v,w}(G))$.
- Eventualmente a aplicação recursiva das adições e condensações chegará em um grafo completo. O número cromático de G é o número do menor grafo completo a ser alcançado a partir de sucessivas adições e condensações.

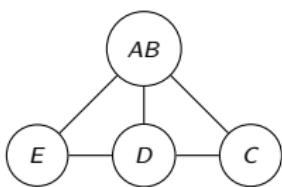
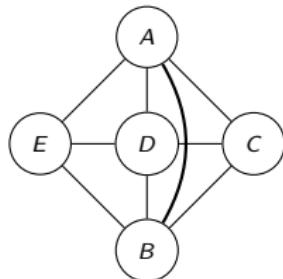
- A correção do método baseia-se no teorema seguinte:
 - Seja G um grafo não completo e v, w um par de vértices não adjacentes de G . Então o número cromático $\chi(G)$ satisfaz:
 - $\chi(G) = \min\{\chi(\alpha_{v,w}(G)), \chi(\beta_{v,w}(G))\}$
- **Prova:**
 - Suponha uma coloração ótima C de G . Se v, w possuem cores diferentes em C , então a aresta (v, w) pode ser acrescentada a G sem alterar C . Neste caso $\chi(G) = \chi(\alpha_{v,w}(G))$. Se v, w possuem a mesma cor em C então podem ser condensados em um único vértice z . Neste caso $\chi(G) = \chi(\beta_{v,w}(G))$. Logo, $\chi(G)$ deve ser o menor entre $\chi(\alpha_{v,w}(G))$ e $\chi(\beta_{v,w}(G))$.

- Algoritmo :
- Entrada: $G(V, E)$
- 1.Se o grafo for completo (K_n) então o número cromático é n .
- 2.Caso contrário, existem 2 vértices distintos, v e w , não adjacentes. Neste caso, definem-se outros dois grafos:
 - 2.1. Grafo obtido a partir de G pela adição de uma aresta entre v e w .
 - 2.2. Grafo obtido pela condensação dos vértices v e w em um único vértice z , eliminando-se as arestas paralelas.
- 3.Para cada um dos grafos obtidos aplica-se recursivamente a mesma estratégia. Por esta recursão, cada grafo será, em algum momento, um grafo completo, para o qual o número cromático é facilmente computável : é o menor número cromático obtido.

Ex1. Neste exemplo uma seqüência de duas condensações de vértice resulta em K_3 .



$$\alpha_{a,b} \\ \Rightarrow$$



$$\beta_{e,c} \\ \Rightarrow$$

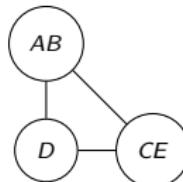


Figura: $\chi = 3$

$$\Downarrow \alpha_{e,c}$$

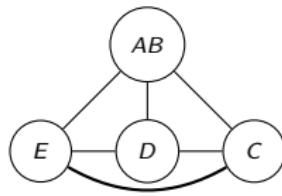
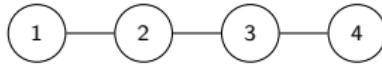


Figura: $\chi = 4$

- No grafo a seguir, se os vértices selecionados forem o 1 e 4, que , na coloração ótima tem cores distintas, a condensação de ambos em um único vértice resulta em K_3 , e a ligação dos mesmos por uma aresta vai resultar, depois de algumas operações, em K_2 . Este exemplo ilustra como a escolha dos vértices afeta o número de operações.



Problemas de Caminho Crítico - PERT/CPM

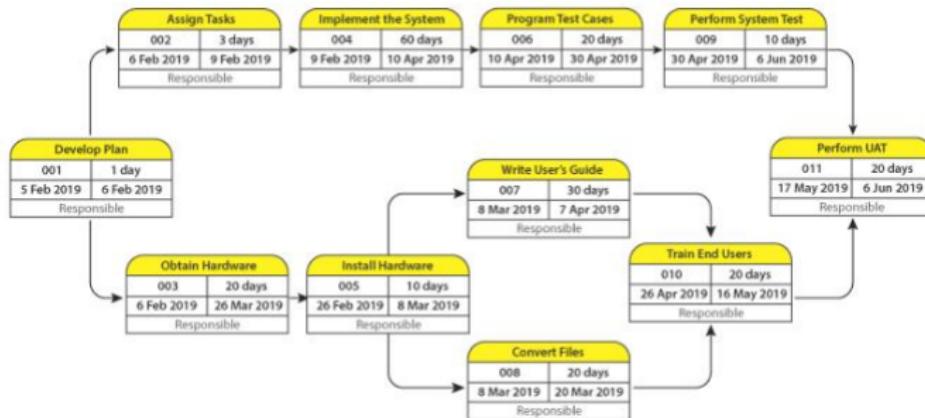
- Pert/CPM (Program Evaluation and Review Technique / Critical Path Method) é uma técnica de sequenciamento de atividades que permite a determinação do tempo total mínimo e a realocação de recursos no sentido de melhor aproveitamento. Ambas foram desenvolvidas na década de 50, e é comum que, pelas semelhanças que apresentam entre si, sejam consideradas como uma única técnica.

[Voltar para o índice](#)

- A partir da análise de um conjunto de tarefas que compõem um projeto, da estimativa da duração de cada tarefa e das dependências entre elas, pode-se extrair algumas informações como:
 - Qual o tempo total previsto para o projeto
 - Como a alteração no tempo de uma tarefa influenciará o tempo total do projeto
 - Onde devo investir recursos para reduzir o tempo total do projeto

[Voltar para o índice](#)

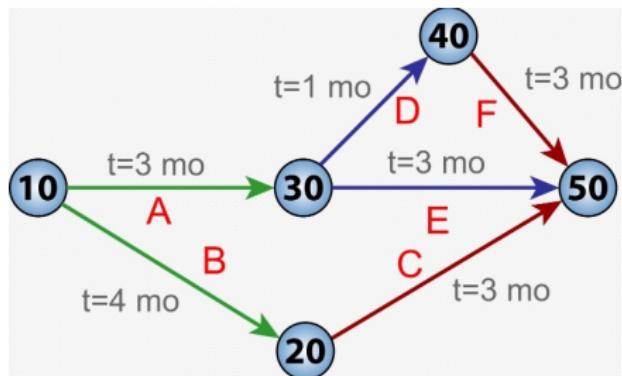
- A figura abaixo mostra um diagrama pert de um projeto de software. Cada bloco representa uma atividade com data de início e término, nome e identificação numérica da atividade e responsável.



- Um diagrama PERT é representado por uma rede $R(V,A)$, ou seja, um grafo orientado, sem laços e valorado, que contém exatamente uma fonte e um sumidouro.
- Nesta rede, os vértices representam eventos (milestones), ou etapas concluídas de um projeto, os arcos representam atividades e os valores associados aos arcos representam a duração das atividades.

[Voltar para o índice](#)

- A figura abaixo ilustra um diagrama PERT de um projeto com 6 atividades (A a F), com duração expressa em meses, e 5 eventos (10 a 50).
- Nela, a atividade 10 representa o instante inicial do projeto, a atividade 50 representa a conclusão do projeto e cada vértice representa o momento em que todas as atividades (arestas) dos quais ele depende já foram concluídas.



- A disposição dos eventos (vértices) e das atividades (arcos) no grafo expressa uma relação de dependência: um evento não ocorre antes que todas as atividades a ele imediatamente precedentes (arestas incidentes) sejam encerradas;
- de forma similar, uma atividade não pode ser iniciada antes que ocorra o evento que representa o início da atividade.

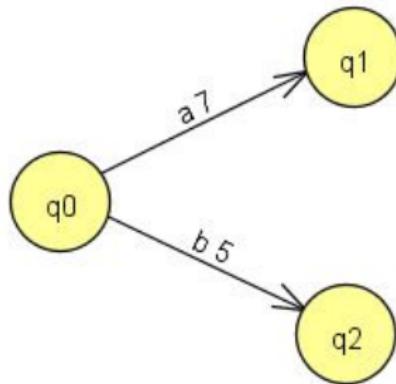
[Voltar para o índice](#)

- Nessa rede, o interesse maior é determinar o(s) caminho(s) do vértice inicial ao vértice final cuja soma das durações das atividades seja máxima.
- Este(s) caminho(s) é chamado **caminho crítico**.
- Ex: Seja o trabalho de fabricação de uma estante fechada com prateleiras, descrito pelas atividades abaixo discriminadas e pelas atividades que devem preceder a cada uma delas.
- A duração é expressa em dias de trabalho. Qual o tempo mínimo para concluir a estante?

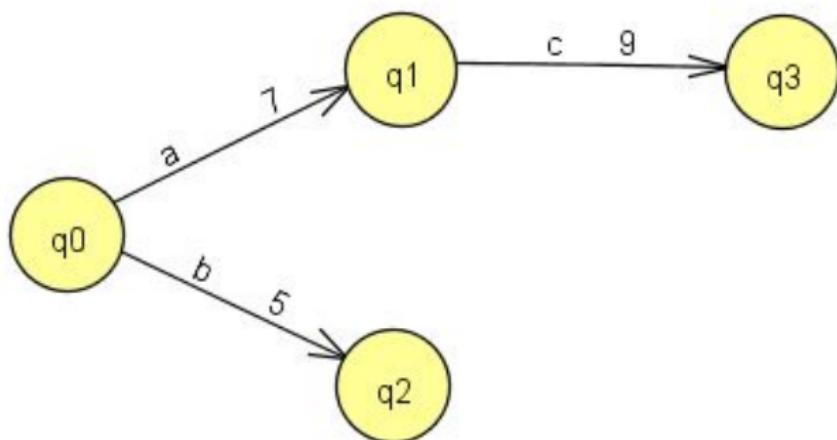
Atividade	Símbolo	Predec.	Duração
Corte da madeira p/ corpo	a	-	7
Idem, para prateleiras	b	-	5
Montagem do corpo	c	a	9
Preparar ajuste das portas	d	c	11
Fabricação das prateleiras	e	b	6
Ajuste das prateleiras	f	c,e	4
Acabamento do corpo	g	d	3
Acabamento das prateleiras	h	f	8
Lustro das prateleiras	i	g,h	6
Lustro do corpo	j	g,h	6
Entrega e montagem	k	i,j	7

Tabela: Etapas da fabricação de uma estante fechada

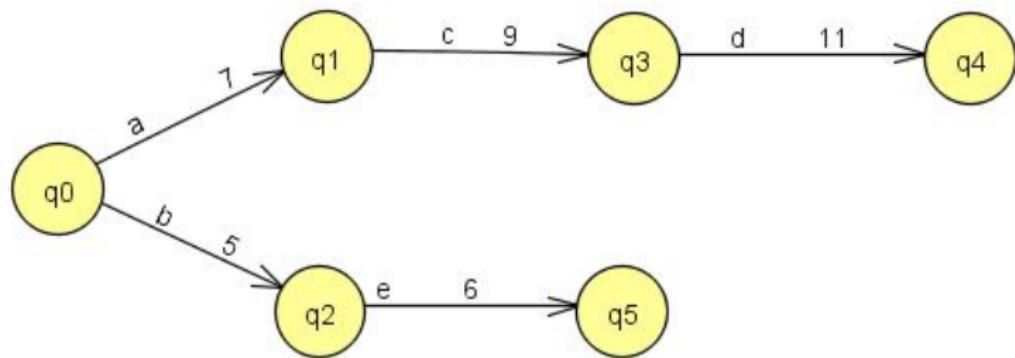
- No diagrama abaixo o vértice 0 representa o início do projeto.
- Como as atividades **a** e **b** não tem nenhuma predecessora (não dependem de nenhuma outra) elas iniciam no vértice 0.
- Cada aresta representa uma atividade e é identificada pelo identificador e duração da mesma.



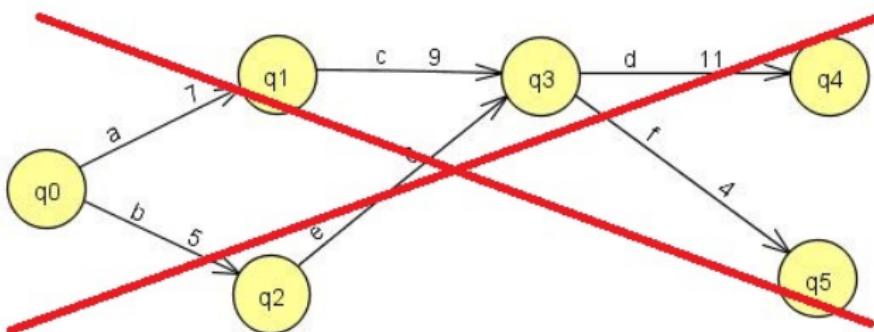
- A atividade **c** depende da atividade **a**, por isso a aresta que a representa parte do vértice q_1 , que representa a conclusão da atividade **a**.



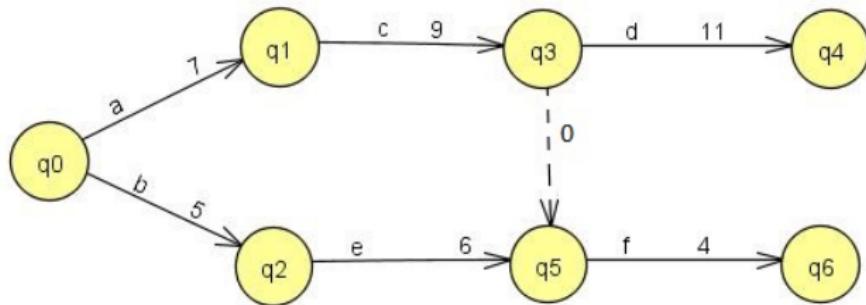
- Da mesma forma, a atividade **d** depende da atividade **c** e a atividade **e** depende da atividade **b**.



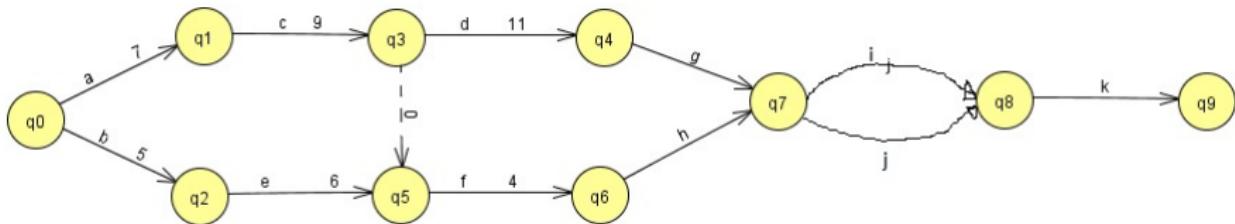
- A atividade **f** depende das atividades **c** e **e**. Quando uma atividade tem mais de uma predecessora, uma possibilidade é ligar todas as predecessoras em um mesmo vértice que representa a conclusão de todas as atividades predecessoras.
- Isso não é possível nessa situação, porque isso faria com que a atividade **d** dependesse também de **e**, dependência que não existe no projeto.



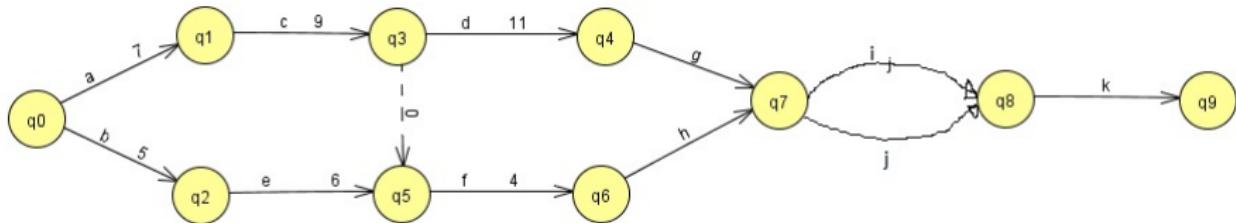
- A solução para isso é representar a dependência entre **f** e **c** por uma atividade-fantasma, representada por uma linha tracejada. Essa aresta pode ser interpretada como uma atividade de duração igual a 0.



- A atividade **g** depende da **d** e a **h** depende da **f**. Como não há nenhuma atividade que dependa somente da **g** ou da **h** (as atividades **i** e **j** dependem ambas de **g** e **h**) as atividades **g** e **h** podem terminar no mesmo vértice.



- As atividades **i** e **j** dependem de **g** e **h**, e a atividade **k** depende de **i** e **j**. Na figura, a conclusão do projeto é representada pelo estado **q9**.



- Na montagem do grafo parte-se de um evento e busca-se associar as atividades uma a uma, concatenando-as pelos seus eventos inicial e final, com base nas informações da lista.
- Na ausência de ambiguidades, o grafo está construído.
- Se elas ocorrerem, terão de ser resolvidas com a inclusão de atividades-fantasma, representadas por linhas tracejadas.
- Essas atividades possuem duração 0, e são utilizadas apenas para representar dependência.

Algoritmo PERT (Program, Evaluation and Review Technique)

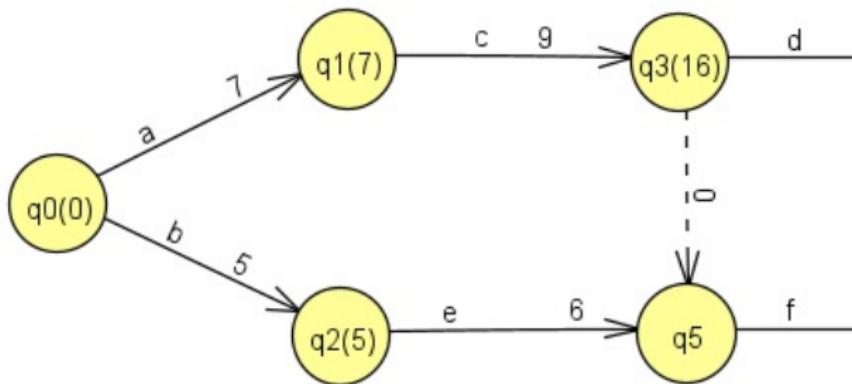
- É o algoritmo a ser aplicado ao grafo eventos-atividades.
- Envolve três etapas nas quais se determinam, de início, dois conjuntos de rótulos para os eventos (vértices): as *datas mais cedo* (denotadas t_i), as *datas mais tarde* (denotadas t'_i) e, em seguida, um caminho de comprimento máximo entre os eventos inicial e final: o **caminho crítico**.
- A **data mais cedo** para um evento é aquela antes da qual ela não pode ocorrer.

- Informalmente o algoritmo é o seguinte:
 - 1 Partir do vértice inicial (rotulado com zero para a data mais cedo) e rotular cada vértice com o valor do **maior caminho** que o atinge, fazendo as comparações que forem necessárias - os valores obtidos são as *datas mais cedo*;
 - 2 Tomar o valor obtido para o vértice final como rótulo para a data mais tarde e voltar, subtraindo os valores dos arcos e ficando, em cada vértice, com o **menor valor** obtido, após as comparações que forem necessárias - os valores obtidos são as "datas mais tarde";
 - 3 Todo vértice para o qual as etapas (1) e (2) produzirem o mesmo valor, faz parte do caminho mais longo (caminho crítico);

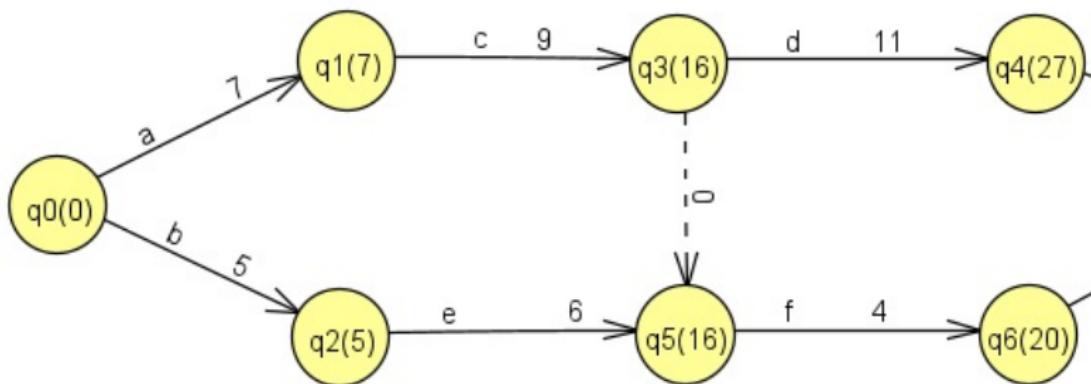
- A data mais cedo do vértice inicial é 0, significando que se o início do projeto for atrasado, isso acarretará em um atraso no final do projeto.
- A partir daí, calcula-se para cada vértice **w** a maior distância possível do vértice inicial até ele. Isso pode ser feito calculando-se a soma da data mais cedo de cada predecessor **v** de **w** com o custo do arco (v,w) . O maior valor obtido é a data mais cedo de **w**.
- Assim, para calcular a data mais cedo de um vértice, é necessário já ter calculado a data mais cedo de todos seus predecessores, o que caracteriza um problema de

ordenação topológica .

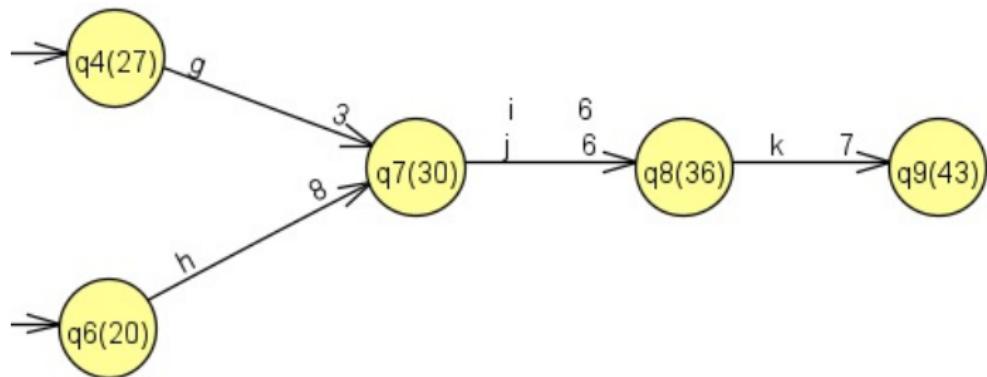
- Na figura abaixo, a data mais cedo de cada evento é representada entre parênteses.
- Como o evento **q1** depende da atividade **a**, de duração igual a 7, o tempo mínimo para o evento ser atingido é 7.
- Qual a data mais cedo para o evento **q5**?



- A data mais cedo do evento **q5** é 16, que é o maior valor entre $16+0$ e $5+6$.



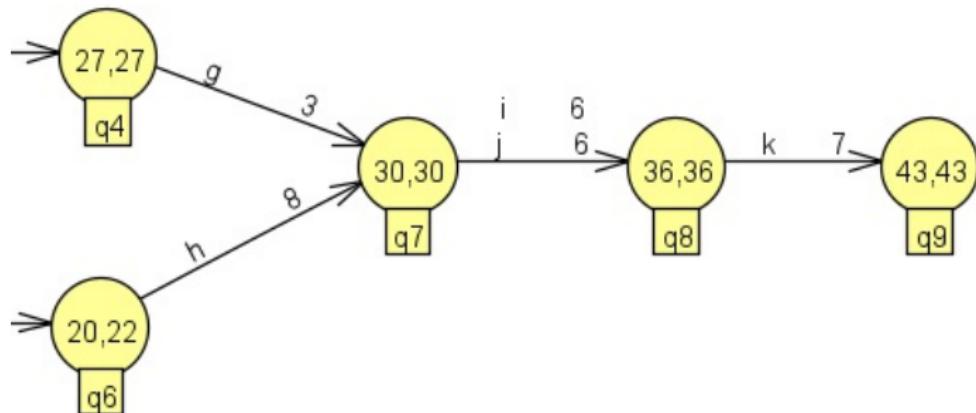
- E assim segue até calcular a data mais cedo de todos os eventos. No exemplo, a data mais cedo é 43, que significa que, a não ser que a duração de alguma atividade seja reduzida, não é possível completar o projeto antes de 43 unidades de tempo.



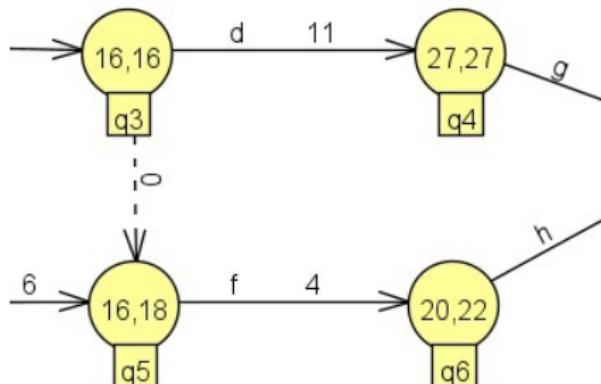
- Na continuação do algoritmo calcula-se para cada evento a "data mais tarde"
- A **data mais tarde** é aquela além da qual, se o evento não tiver ocorrido, acarretará um atraso no projeto como um todo.
- A diferença entre uma data mais cedo e a data mais tarde é a **folga** do evento.
- Se um evento tem 10 como data mais cedo e 15 como data mais tarde, isso significa que ele não tem como iniciar antes de 10, mas mesmo que inicie até o instante 15 isso não irá acarretar atrasos no projeto.

- O cálculo da data mais tarde de cada evento é feito do final para o início.
- Inicialmente toma-se a data mais cedo do vértice final como rótulo para a data mais tarde e voltar, subtraindo os valores dos arcos e ficando, em cada vértice, com o **menor valor** obtido, após as comparações que forem necessárias - os valores obtidos são as "datas mais tarde";

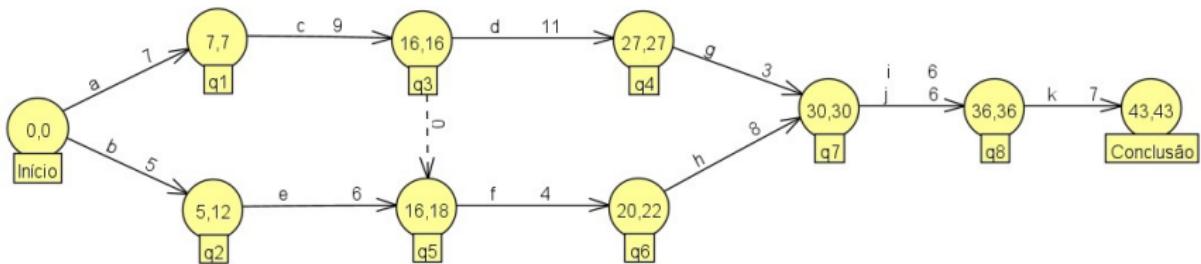
- Na figura abaixo o primeiro valor em cada evento representa a data mais cedo e o segundo valor é a data mais tarde.
- No evento q6 a diferença entre ambas é a folga do evento, significando que se o evento será alcançado em 20 unidades, mas que se for alcançado em 22 unidades não afetará a conclusão do projeto.



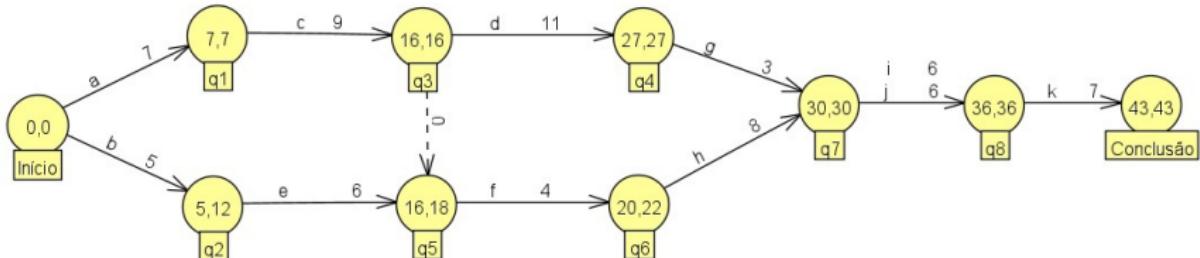
- A data mais tarde do evento q3 é calculada a partir das datas mais tarde dos eventos q4 ($27-11=16$) e q5 ($18-0=18$), já que ambos dependem de q3.
- Pega-se o menor dos valores. Se o evento q3 for alcançado em 18 ao invés de 16, isso atrasará a conclusão do projeto, pois isso atrasará os eventos q4, q7, q8 e q9 (conclusão do projeto).



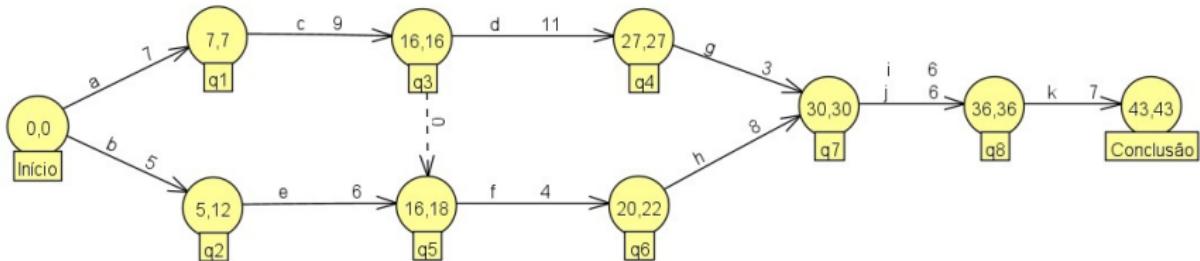
■ E assim por diante.



- A partir desse modelo pode-se identificar o **caminho crítico**, ou seja, as atividades e eventos que, se atrasarem, atrasarão todo o projeto.
 - Um **evento crítico** é um vértice do caminho crítico. Eventos críticos são eventos em que a data mais cedo e a data mais tarde são iguais.



- No exemplo em questão, o caminho crítico é formado pelos eventos q1, q3, q4, q7 e q8.
 - O algoritmo PERT pode ser utilizado para avaliar revisões de projeto. Pode-se, por exemplo, deslocar recursos das atividades onde ocorrem as maiores folgas para atividades críticas reduzindo o tempo total.



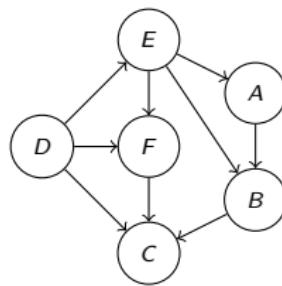
Ordenação Topológica

- Uma **ordenação topológica** de um dígrafo acíclico (DAG) é uma ordenação, possivelmente parcial (em que a ordem não está definida para todos os pares de vértices), linear de seus nós em que cada nó vem antes de todos nós para os quais este tenha arestas de saída.
- Cada DAG tem uma ou mais ordenações topológicas.
- Se um grafo tem ciclos, ele não possui uma ordenação topológica.
- Algoritmos de ordenação topológica começaram a ser estudados no início dos anos 1960 no contexto da técnica PERT.

[Voltar para o índice](#)

- Existem diversas ordenações topológicas para o mesmo grafo. Para o grafo a seguir as três ordenações mostradas são válidas

D, E, A, F, B, C
D, E, A, B, F, C
D, E, F, A, B, C



- A ordenação topológica é útil para enumerar as possíveis seqüências de execução de tarefas com dependência entre elas:
 - Escalonamento de instruções na geração de código em compiladores
 - Tarefas numa linha de montagem
 - Diagrama de Pert
 - Ordenamento da avaliação de células com dependências entre elas em planilhas eletrônicas
 - Determinação da ordem de compilação em makefiles
 - Resolução de dependências de símbolos em linkadores
- Uma forma de achar a ordenação topológica é remover, a cada iteração, um vértice com grau de entrada nulo, gerando um novo dígrafo, até que não hajam mais vértices a considerar.

Algoritmo:

Dados: dígrafo acíclico $D(V, E)$

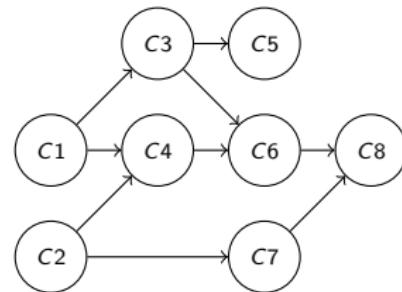
Para $j = 1, \dots, n$ efetuar

Escolher um vértice w com grau de entrada nulo em D

Retirar de D o vértice w e arestas dele divergentes

Definir $v_j = w$

Como exemplo, imagine um currículo de um curso de graduação onde os pré-requisitos são expressos por um grafo, como ilustrado na figura a seguir:



- Utilizando o algoritmo com o grafo ilustrado na figura, podemos obter a seguinte sequência: C1, C2, C3, C4, C5, C6, C7, C8.
- Note que como a cada etapa do algoritmo pode existir mais de um vértice de grau de entrada nulo, a resposta do algoritmo pode ser diferente.
- Veja por exemplo essa outra sequência que ele poderia retornar: C1, C3, C5, C2, C7, C4, C6, C8.

- É muito fácil mostrar que o algoritmo funciona corretamente, pois a cada etapa selecionamos um vértice que não tem nenhum predecessor, ou se ele tem, esse predecessor já foi processado anteriormente.
- Quanto ao tempo de execução, ele depende da maneira de implementar o grafo.
- O ponto crítico é a busca do vértice de grau de entrada nulo (a remoção pode ser realizada em tempo constante).

- Se for usada uma matriz de adjacência, o tempo para procurar um vértice de grau de entrada nulo é em $O(n^2)$.
- Como isso sera repetido n vezes, o tempo total é em $O(n^3)$.
- Se for usada uma estrutura de adjacência, a situação é melhor se o grafo é esparsa, senão, é igual.
- A cada etapa o tempo para buscar o vértice de grau nulo é no mínimo n , pois todo elemento do vetor deve ser visitado para ver se ele foi eliminado.
- Além disso, será preciso visitar todas as arestas do grafo.
- Então teremos um tempo de execução em $O(n + a)$ a cada iteração.
- No total, isso dá um tempo em $O(n^2 + na)$.

- É possível transformar o algoritmo de tal maneira que ele seja linear.
- É só utilizar um vetor $G[1..n]$ que memoriza o grau de entrada de cada vértice e uma lista L que contém todos os vértices de grau de entrada nulo.
- Eis uma versão em $O(n + a)$ do algoritmo:

procedimento Orden-Topo-Alt-Modif($G = (V, E)$): Grafo

$L := \emptyset$

Para $i := 1$ até n :

$G[i] := 0$

Para $i := 1$ até n :

 Para cada vértice j tal que existe uma aresta de i para j :

$G[j] := G[j] + 1$

Para $i := 1$ até n :

 Se $G[i] = 0$:

$L := L \cup i$

Enquanto L não vazia

$v :=$ um vértice de L

imprimir(v)

$L := L - v$

Para cada vértice w tal que existe uma aresta de v para w :

$G[w] := G[w] - 1$

Se $G[w] = 0$:

$L := L \cup w$

- Outra forma de resolver esse problema é a utilização de uma versão do algoritmo de busca em profundidade que imprime o vértice antes de retornar da chamada. Obteremos assim os vértices em ordem topológica invertida:

procedimento Orden-Topo(G : Grafo)

Para Cada vértice v de G :

 Marque v como não visitado

Para Cada vértice v de G :

 Se v não foi visitado:

 Busca-Prof-Orden-Topo(v)

fimprocedimento

procedimento Busca-Prof-Orden-Topo(v : vértice)

 Marque v como visitado

 Para cada vértice w adjacente a v :

 Se w não foi visitado:

 Busca-Prof-Orden-Topo(w)

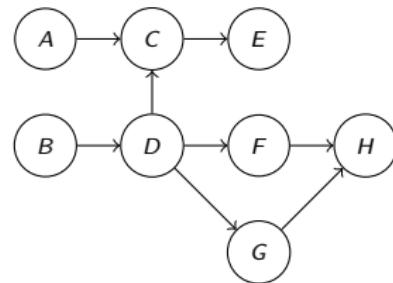
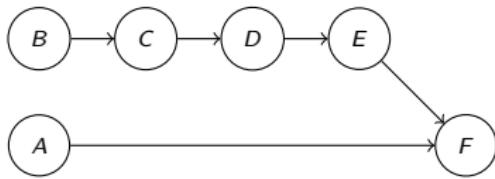
 Imprimir(v)

- Para verificar que esse algoritmo funciona, é só ver o que acontece quando chega ao final de uma recursão, no momento que ele imprime o vértice.
- Nesse caso, ele já foi o mais longe possível a partir do vértice inicial (isso é uma característica do algoritmo de busca em profundidade).
- Seja x o último vértice desse caminho.
- Não existe vértice v tal que $x < v$, porque se for o caso o algoritmo continuaria a busca no mínimo até v .
- Portanto, de todos os próximos vértices que serão imprimidos, nenhum sucede a v na ordem, e obteremos uma ordem topológica invertida.

- Imaginemos agora uma pessoa que pode se matricular somente em um curso por semestre.
- Nesse caso, a ordenação topológica retorna a sequência de cursos que ela deverá fazer.
- É fácil verificar que um resultado possível do algoritmo é a sequência C5, C8, C6, C3, C4, C1, C7, C2.
- Invertendo essa sequência, obtemos a ordem topológica C2, C7, C1, C4, C3, C6, C8, C5.
- Note que o resultado depende que qual vértice é selecionado a cada chamada recursiva da busca.
- Quanto ao tempo de execução desse algoritmo, é claro que é na mesma ordem que o algoritmo genérico de busca em profundidade.

Exercício

Dê três ordenações topológicas para cada um dos seguintes dígrafos:



[Voltar para o índice](#)

Identificação de componentes fortemente conexos

- Um grafo direcionado é **fortemente conexo** se existe um caminho de u até v e de v até u para qualquer par distinto de vértices u e v .
- Se um grafo não é fortemente conexo, podemos estar interessados em saber quais são os seus subgrafos que são fortemente conexos.
- Cada um desses subgrafos é chamado componente fortemente conexo.

- Considere o grafo ilustrado na figura abaixo. Podemos ver que ele contém três componentes fortemente conexos: $\{1, 2, 3, 4\}$, $\{6\}$, $\{5, 8, 9, 10\}$ e $\{7, 11\}$.

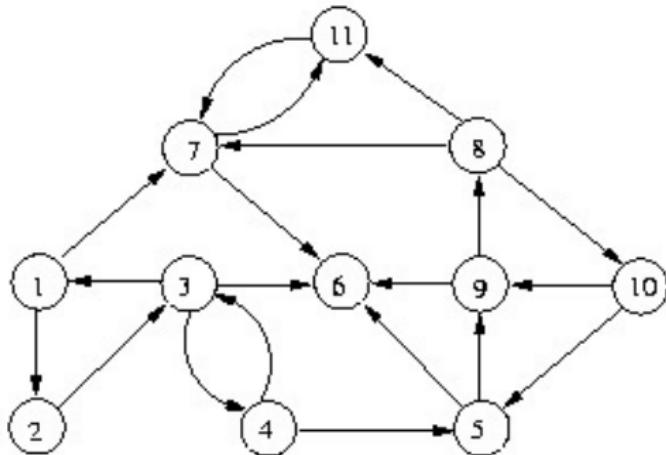
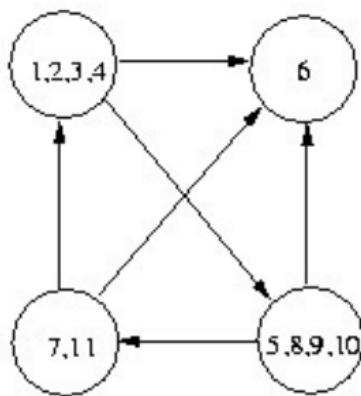


Figura: "Dígrafo com 4 componentes fortemente conexas"

- Existe uma maneira interessante de visualizar o grafo se considerarmos cada componente como um vértice.
- Isso é uma representação de como ir de um componente a outro componente.
- Veja na figura abaixo essa representação do grafo da figura anterior



Algoritmo de Kosaraju para identificação de componentes fortemente conexas

- Para identificar os componentes fortemente conexos de um grafo, é preciso primeiro modificar a busca em profundidade, para obter uma numeração pós-ordem dos nodos da árvore que resulta da busca.
- Para isso, é só incrementar um contador no momento da saída de uma chamada recursiva.

Índice

- Supondo uma variável global posnum[1..n], eis o algoritmo modificado:

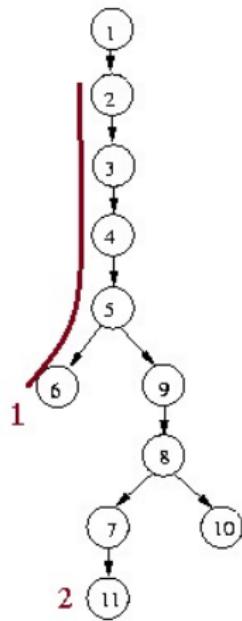
```
procedimento Busca(G: Grafo)
  nump := 0
  Para Cada vértice v de G:
    Marque v como não visitado
  Para Cada vértice v de G:
    Se v não foi visitado:
      Busca-Prof(v)
  retorna

procedimento Busca-Prof(v: vértice)
  Marque v como visitado
  Para Cada vértice w adjacente a v:
    Se w não foi visitado:
      Busca-Prof(w)
  nump := nump + 1
  posnum[v] := nump
  retorna
```

Índice

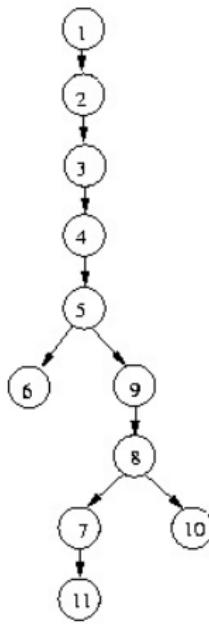
- A numeração resultante dá a ordem em que a varredura dos nodos é completada e é executado o return na busca em profundidade.
- No exemplo anterior, o primeiro nodo para o qual se encerra a varredura é o nodo 6.
- Seguindo a dfs, o segundo nodo a completar a varredura é o 11.
- E o terceiro é o nodo 7. E o quarto é o nodo 10. E assim vai...

Índice



- Agora, podemos apresentar o algoritmo que identifica os componentes fortemente conexos de um grafo G :
 - 1 Realizar uma busca em profundidade em G . Consideramos que depois dessa busca, cada valor $\text{posnum}[v]$ representa a posição do vértice v na sequência resultando de percurso pós-ordem.
 - 2 Construir um grafo G' que é igual a G , só que a orientação de cada aresta é invertida.
 - 3 Realizar uma busca em profundidade em G' , começando com o vértice que tem o maior valor em posnum . Recomeçar o processo com os vértices não visitados se essa busca não visitou todos. Assim por diante até que todos os vértices sejam visitados.
 - 4 Cada árvore da floresta que resulta da etapa precedente contém todos os vértices de um componente fortemente conexo de G .

- Vamos ver agora a aplicação desse algoritmo ao exemplo anterior. Suponhamos que a busca em profundidade corresponde à árvore ilustrada na figura a seguir.



- A figura a seguir ilustra o grafo G' . Ao lado de cada vértice é indicado um valor que representa a posição desse vértice no percurso pós-ordem da árvore obtida na busca em G .

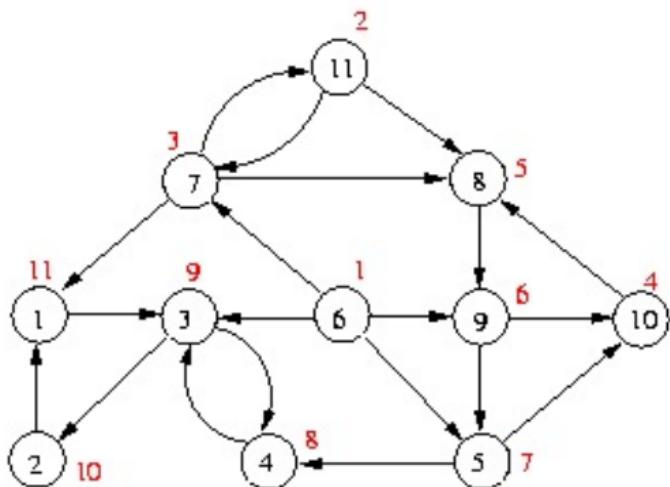


Figura: "Grafo G' "

Realizando a busca em profundidade no grafo G' ilustrado na figura 8, obtemos as árvores ilustradas na figura 9. Isso corresponde ao resultado esperado.

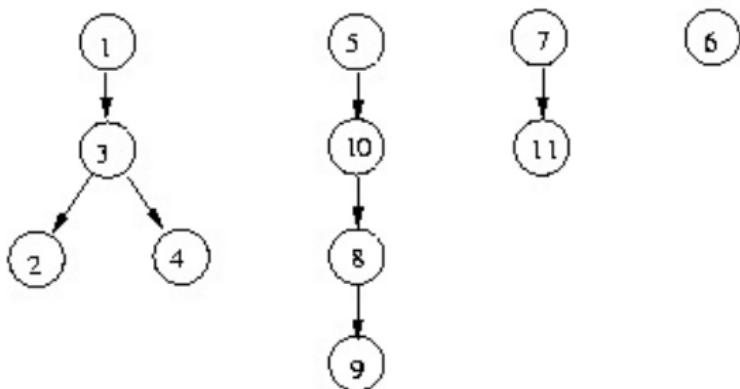


Figura: "Árvores geradas"

Por que inverter o grafo?

- Utilizar o grafo invertido e efetuar as buscas em profundidade na ordem obtida na dfs pós-ordem garante que se há um arco $A \rightarrow B$ (no grafo original) onde A e B são subgrafos simplesmente conexos, então na segunda dfs a busca em A terminará antes do início da busca em B. Se a busca for efetuada no grafo original, isso não é assegurado.
- Exemplo:

1->2

2->3

3->4

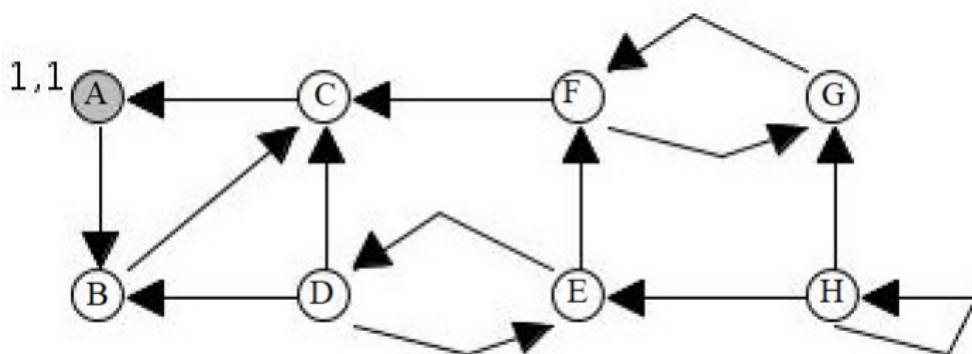
4->2

3->5

Algoritmo de Tarjan

- O algoritmo de Tarjan é da mesma classe de complexidade que o de Kosaraju, mas por efetuar apenas uma DFS no grafo, apresenta constantes menores, tendo normalmente um melhor desempenho.
- A ideia é identificar na DFS quando um vértice possui uma aresta para algum ancestral seu na árvore da DFS, identificando um ciclo e consequentemente parte de uma componente fortemente conexa.
- A cada vértice é mantida a numeração da DFS disparada a partir dele (index) e a numeração do vértice de menor numeração alcançado a partir dele (lowlink).
- Uma animação pode ser vista em [Aqui](#)

- No grafo abaixo, inicia-se a primeira DFS a partir do vértice A. O primeiro número refere-se à ordem de visitação, o segundo número refere-se ao menor número alcançado a partir dessa DFS.



Algoritmo de Tarjan

entrada: grafo $G = (V, E)$

saída: conjunto de componentes fortemente conexas (cada uma definida pelo conjunto de vértices)

// A função strongconnect encontra todas as componentes fortemente conexas alcançáveis a partir de v

index := 0 // Associa uma ordem de visitação a cada vértice

S := pilha vazia

para cada vértice v em V faça

 if v.index is undefined then

 strongconnect(v)

 end if

end for

```
function strongconnect(v)
    v.index := index
    v.lowlink := index
    index := index + 1 // contador global da ordem de
visitação
    S.push(v)
    v.onStack := true
    para cada w adjacente a v
        if w.index is undefined then
            strongconnect(w)
            v.lowlink := min(v.lowlink, w.lowlink)
        else if w.onStack then
            v.lowlink := min(v.lowlink, w.index)
        end if
    end for
■ (continua...)
```

```
// If v is a root node, pop the stack and generate an SCC
if v.lowlink = v.index then
    start a new strongly connected component
repeat
    w := S.pop()
    w.onStack := false
    add w to current component
    while w ≠ v
        output the current component
end if
end function
```

Exercícios de grafos fortemente conexos

- URI 1128 - Ir e Vir
- URI 2429 - Rodovia

Grafos semi-conexos

- Um grafo é dito **semi-fortemente conexo** (ou **sf-conexo** ou **semi-connected**) se, para todos pares de vértices u, v em V , há um caminho de u para v ou de v para u .
- A verificação pode ser feita em tempo $O(V+E)$ a partir da seguinte afirmação:
 - Um DAG (grafo dirigido acíclico) é semi-conexo se, em sua ordenação topológica há uma aresta de cada vértice i para o vértice $i+1$.
 - Ou, se o DAG tem uma única ordenação topológica.

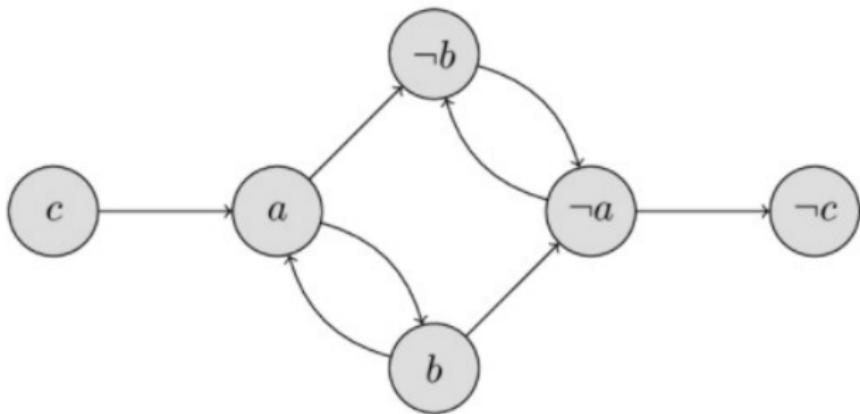
O problema do 2-SAT por componentes fortemente conexas

- O problema de Satisfatibilidade (SAT) consiste em atribuir valores booleanos às variáveis de forma a satisfazer uma dada expressão booleana.
- A expressão booleana normalmente estará na forma CNF, forma normal conjuntiva, que é uma conjunção (E lógico) de diversas cláusulas, onde cada cláusula é uma disjunção (OU lógico) de literais.

- 2-SAT (2-satisfatibilidade) é uma restrição do problema SAT, em que cada cláusula tem exatamente dois literais. P.ex: Encontrar atribuição de valores às variáveis a, b, c tal que a expressão a seguir é verdadeira:
- $(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$
- SAT é NP-completo, não havendo soluções eficientes para ele. Entretanto, 2SAT pode ser resolvido em $O(n+m)$ onde n é o número de variáveis e m é o número de cláusulas.

- O problema 2SAT pode ser resolvido por componentes fortemente conexas.
- O primeiro passo é transformar cada cláusula em duas implicações.
- Como a cláusula especifica que pelo menos uma das subexpressões é verdadeira, se uma delas é falsa então a outra deve ser verdadeira. P.ex: A cláusula $(a \vee b)$ é equivalente à expressão $(\neg a \implies b) \wedge (\neg b \implies a)$
- A expressão $(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$ resulta nas implicações: $(\neg a \implies \neg b)$, $(b \implies a)$, $(a \implies b)$, $(\neg b \implies \neg a)$, $(a \implies \neg b)$, $(b \implies \neg a)$, $(\neg a \implies \neg c)$, $(c \implies a)$

- A partir desse conjunto de implicações pode-se montar o grafo de implicações, em que os vértices são as variáveis negadas e não negadas, e cada aresta corresponde a uma implicação. Se $a \Rightarrow b$, há uma aresta de a para b. O grafo da expressão acima é:



- A partir do grafo de implicação pode-se verificar se o problema é satisfatível (se há uma atribuição de valores às variáveis que torne verdadeiras todas as cláusulas).
- Isso pode ser feito encontrando todas as componentes fortemente conexas do grafo. Se ocorrer de uma variável e sua negação estarem na mesma componente conexa, a expressão não é satisfatível, caso contrário, é satisfatível.
- Problemas: URI 1348 - X-Mart

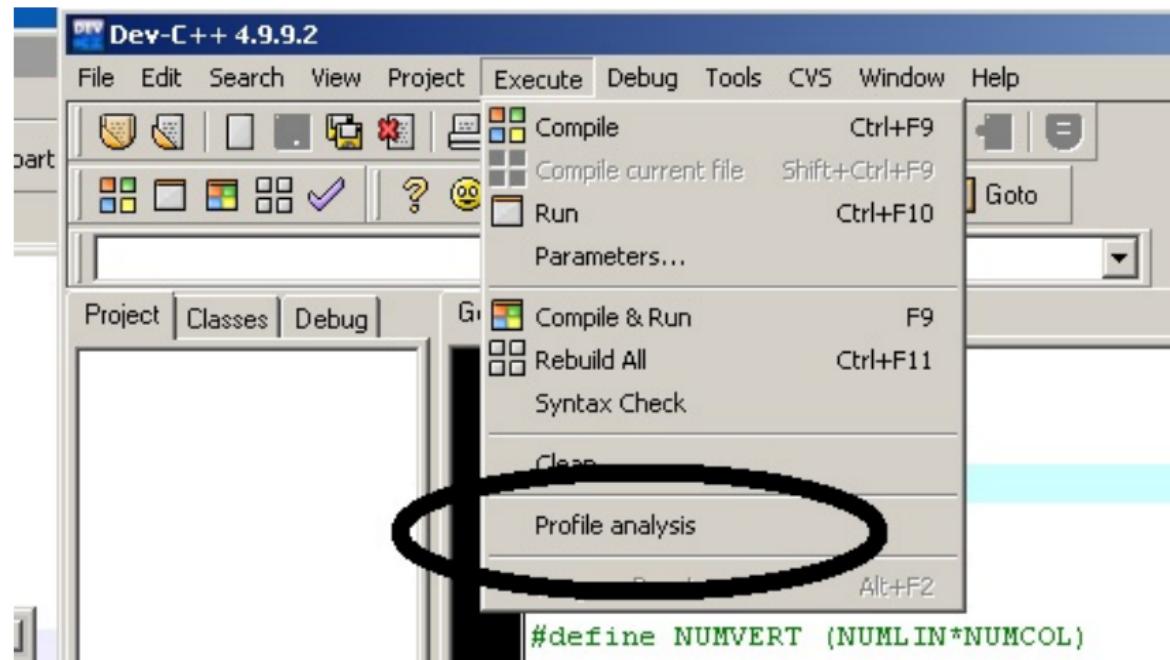
- Para achar uma atribuição de valores às variáveis, deve-se verificar para cada variável x se há algum caminho de x até $\neg x$ ou de $\neg x$ até x .
- Se houver um caminho de x até $\neg x$ o valor lógico da variável x é falso (se fosse verdadeiro, implicaria que $\neg x$ é verdadeiro, o que é uma contradição)
- Da mesma forma, se houver um caminho de $\neg x$ até x , seu valor lógico é verdadeiro.

Particionamento de Grafos

- O problema de particionamento de grafos reside em, dado um grafo, dividi-lo em subgrafos de forma que o número de vértices em cada subgrafo seja aproximadamente o mesmo e que o número de arestas entre vértices de subgrafos diferentes seja o menor possível.
- Achar a solução ótima é um problema de complexidade exponencial, sendo inviável, portanto, para grafos de um certo número de vértices.

- Existem diversas heurísticas para encontrar soluções próximas à ótima, podendo essas heurísticas ser classificadas como métodos globais (método inercial, método espectral, bissecção recursiva e várias outras) ou locais (kernighan-lin, fiduccia-mattheyses).
- Além disso, existem diversas bibliotecas de particionamento de grafos que são de domínio público (metis, chaco, jostle, scotch, party).

Perfilamento de Código no Dev C++



Profile analysis

Flat output | Call graph

Function name	% time	Cumul. secs	Self secs	Calls	Self ts/call	Total ts/call
amplitude(int)	53.13	0.17	0.17	1	170.00	170.00
main	46.88	0.32	0.15			
mostra_nivel()	0.00	0.32	0.00	1	0.00	0.00
enchemat()	0.00	0.32	0.00	1	0.00	0.00

```

%      the percentage of the total running time of the
time    program used by this function.

cumulative a running sum of the number of seconds accounted
seconds   for by this function and those listed above it.

self      the number of seconds accounted for by this
seconds   function alone. This is the major sort for this
           listing.

calls     the number of times this function was invoked, if
           this function is profiled, else blank.

self      the average number of milliseconds spent in this
ms/call   function per call, if this function is profiled,
           else blank.

total     the average number of milliseconds spent in this

```

Close



Opções para perfilador

Tools → Compiler Options → Code Generation/Optimization
→ Generate profiling information for analysis

Tools → Compiler Options → Linker → Generate debugging information

Execute → Rebuild All

Execute → Run

Execute → Profile analysis

- Senão, ao chamar a primeira vez por Profile analysis o dev seta as opções para gerar informações de debug.

- Qual o efeito no tempo e perfil de execução se a inicialização da matriz for feita com memset ao invés de elemento a elemento?

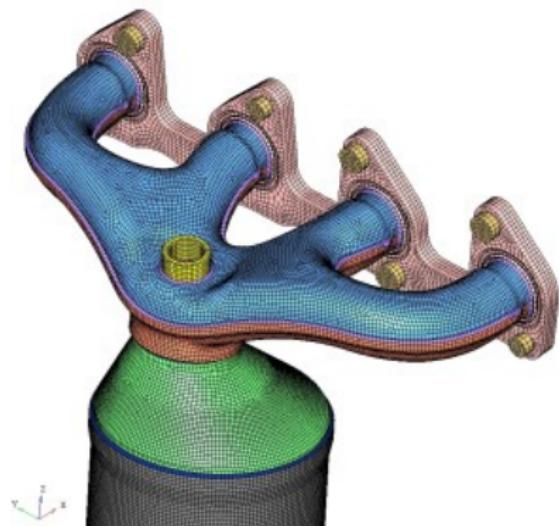
```
int main()
{
    int i, j, lin, col;
    //memset(m, 0, sizeof(m));
    for (i=0; i<NUMVERT; i++)
        for (j=0; j<NUMVERT; j++)
            m[i][j]=0;
    enchemat();
    amplitude(0);
    mostra_nivel();
    system(" pause");
}
```

- Qual o efeito no tempo e perfil de execução se a busca em amplitude for feita utilizando uma fila?

```
void amplitude(int v)
{
    int i, j, n_at, trocou;
    for (i=0; i<NUMVERT; i++)
        nivel[i]=-1;
    nivel[v]=0;
    n_at=0;
    do {
        trocou=0;
        for (i=0; i<NUMVERT; i++)
            if (nivel[i]==n_at)
                for (j=0; j<NUMVERT; j++)
                    if (m[i][j]==1 && nivel[j]==-1)
                    {
                        nivel[j]=n_at+1;
                        trocou=1;
                    }
        n_at++;
    } while (trocou==1);
}
```

- Qual o tamanho da maior grade que é possível montar?
100x100? 200x200? 300x300?
- $300 \times 300 \Rightarrow 90000$ vértices \Rightarrow Matriz de adjacências de $90000 \times 90000 \Rightarrow 8.100.000.000 \Rightarrow$ excede o limite de um inteiro e o tamanho máximo de uma estrutura.

Modelo Físico Discreto



Matriz de adjacências alocada dinamicamente

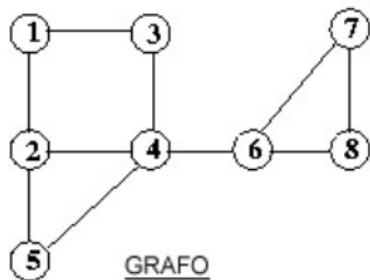
```
#define NUMLIN 200
#define NUMCOL 200
#define NUMVERT (NUMLIN*NUMCOL)
#define m(i,j) m[(i)*NUMVERT+(j)]

if ((m=(char *) malloc(NUMVERT*NUMVERT*sizeof(char)))
```

```
{
```

```
printf("Não deu...\n");
system("pause");
}
```

Lista de Adjacências



Vértice Grau Vértices adjacentes

1	2	2 → 3 /
2	3	1 → 4 → 5 /
3	2	1 → 4 /
4	4	2 → 5 → 3 → 6 /
5	2	2 → 4 /
6	3	4 → 7 → 8 /
7	2	6 → 8 /
8	2	6 → 7 /

LISTA DE ADJACÊNCIAS

Estruturas

```
typedef struct nodo {int valor; struct nodo *prox;} tnodeo;
tnodo *(m[NUMVERT]);
int nivel [NUMVERT];
void enchemat()
{
    int i,lin,col;
    for (lin=0;lin<NUMLIN;lin++)
        for (col=0;col<NUMCOL;col++)
    {
        i=lin*NUMCOL+col;
        if (lin>0)
            insere_aresta(i,i-NUMCOL);
        if (col>0)
            insere_aresta(i,i-1);
    }
}
memset(m,0 ,NUMVERT*sizeof(tnodo *));
```

- Altere o código da busca em amplitude para utilizar uma fila
- Perfile o código comparando com a versão anterior
- Altere o código da busca em amplitude substituindo o laço que testa a adjacência para todos os vértices do grafo, substituindo-o por um código que acessa os vértices adjacentes diretamente da lista de adjacências

Inclusão de Arestas

```
void insere_lista(tnodo **inic , int i)
{
    if (*inic==NULL || (*inic)->valor>i)
    {
        tnodo *ptaux=(tnodo *)malloc(sizeof(tnodo));
        ptaux->valor=i;
        ptaux->prox=*inic;
        *inic=ptaux;
        return;
    }
    if ((*inic)->valor==i) return;
    insere_lista(&(*inic)->prox , i);
}

void insere_aresta(int i , int j)
{
    insere_lista(&(m[i]) , j );
    insere_lista(&(m[j]) , i );
}
```

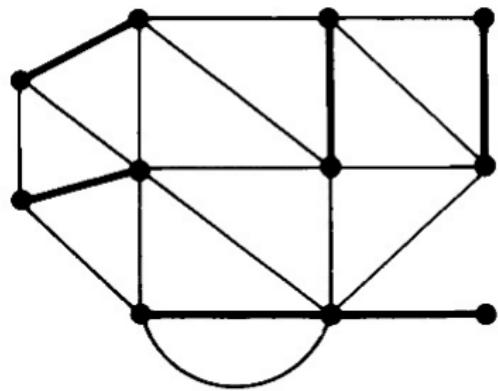
Teste de adjacência

```
int testa_lista(tnodo *inic, int j)
{
    if (inic==NULL) return 0;
    if (inic->valor==j) return 1;
    return testa_lista(inic->prox, j);
}

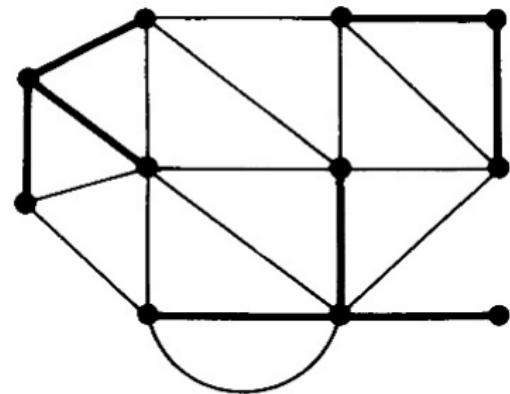
int testa_aresta(int i, int j)
{
    return testa_lista(m[i], j);
}
```

Cobertura de Arestas

- **Def:** Uma **cobertura de arestas** (ou simplesmente cobertura) (edge covering) é um subconjunto de arestas tal que cada vértice do grafo é adjacente a pelo menos uma aresta do subconjunto.
- Diz-se que o subconjunto de arestas **cobre** o grafo.
 - Uma árvore geradora é uma cobertura de arestas
 - Da mesma forma, um ciclo hamiltoniano é uma cobertura de arestas
 - Grafos com vértices isolados não possuem cobertura de arestas
 - Toda aresta adjacente a um vértice de grau 1 pertence a todas as coberturas de aresta.
- Uma cobertura minimal é uma cobertura em que não se pode remover nenhuma aresta.
- A figura abaixo mostra duas coberturas minimais para o mesmo grafo



(a)



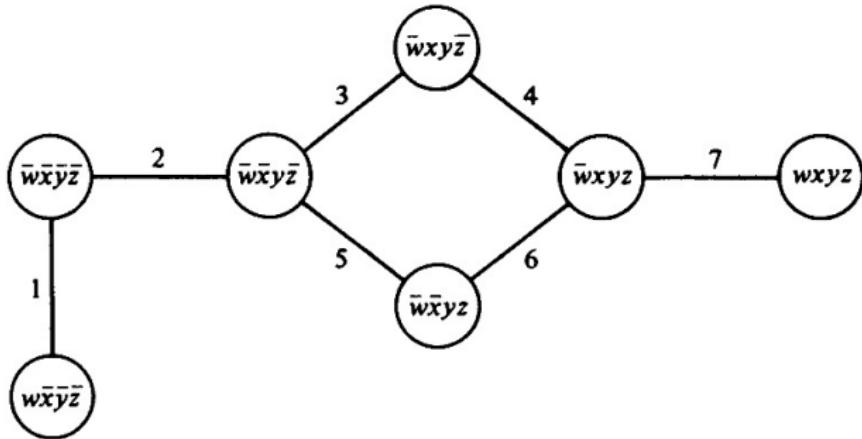
(b)

Graph and two of its minimal coverings.

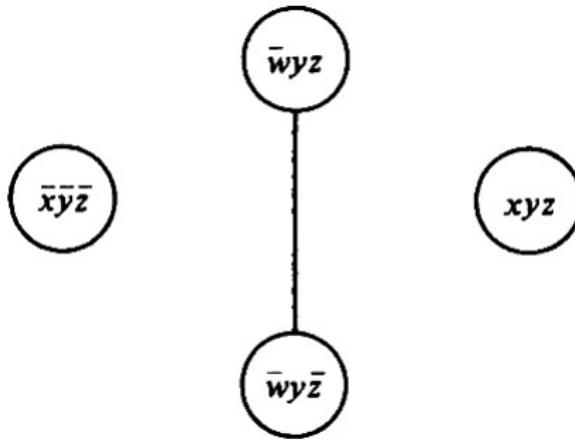
- Um vendedor de brinquedos educativos possui em estoque brinquedos de várias formas geométricas (cubos, pirâmides, etc.), cada qual fabricado em várias cores. O vendedor quer carregar consigo o menor número de objetos tal que cada cor e cada forma estejam representadas pelo menos uma vez.
- O vendedor constrói o seguinte grafo: cada forma e cada cor estão representados individualmente por um vértice e existe uma aresta ligando um vértice-forma a um vértice-cor, caso aquela forma geométrica seja fabricada naquela cor.
- O número mínimo de objetos que o vendedor precisa carregar é igual ao número de arestas numa cobertura de cardinalidade mínima: um

■ 0

- Uma aplicação de cobertura de arestas é em distribuição de pontos de vigilância.
- Uma aplicação menos óbvia é na minimização de funções booleanas. considere a expressão booleana abaixo, representada como uma soma de mintermos:
 - $F = \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z} + w\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}yz + \bar{w}xy\bar{z} + \bar{w}xyz + wxyz$
 - Cada termo pode ser representado por um vértice e arestas conectam vértices que diferem por apenas um termo.
 - Uma cobertura minimal do grafo representará uma expressão equivalente, mas com menos termos.

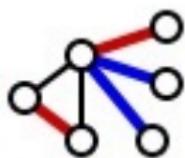


- As arestas 1 e 7 devem fazer parte da cobertura, uma vez que são as únicas que cobrem os vértices de grau 1.
 - Cada aresta da cobertura representará um termo na expressão simplificada.
 - Selecionando, para completar a cobertura, as arestas 3 e 6:
 - $F = \bar{x}\bar{y}\bar{z} + \bar{w}y\bar{z} + \bar{w}yz + xyz$



- Gerando um grafo para a expressão simplificada, os termos $\bar{w}\bar{y}\bar{z}$ e $\bar{w}yz$ só diferem por uma variável e são ligados por uma aresta.
 - Um novo passo de condensação gera a expressão:
 - $F = \bar{x}\bar{y}\bar{z} + \bar{w}y + xyz$
 - Que é a expressão mínima.

- Uma **cobertura mínima de arestas** é a cobertura de arestas de menor cardinalidade para o grafo. O número de arestas da cobertura mínima é denotada por $\rho(G)$
- Uma cobertura mínima de arestas pode ser encontrada em tempo polinomial encontrando uma associação máxima e acrescentando arestas até cobrir todos os vértices.
- Na figura abaixo, uma associação máxima está marcada com vermelho; as arestas adicionadas para completar a cobertura estão em azul.



Conjunto de vértices independentes

- Um **conjunto de vértices independentes** é um conjunto de vértices V' em que para todo $i, j \in V'$, i e j não são adjacentes.
- Se o conjunto V' não é subconjunto próprio de nenhum outro conjunto de vértices independentes, ou seja, não é possível acrescentar vértices a V' e mantê-lo independente, ele é dito **um conjunto maximal de vértices independentes (maximal independent set)**.
- Se não houver nenhum outro conjunto de vértices independentes de cardinalidade maior que V' , ele é dito **conjunto máximo de vértices independentes**.

- O problema de encontrar o conjunto máximo de vértices independentes é um problema NP-Completo, mas conjuntos maximais podem ser encontrados por diversas heurísticas.
- O algoritmo abaixo encontra um conjunto maximal I em tempo $O(M)$:

Inicializa I como um conjunto vazio

Enquanto V não está vazio:

 Escolha um vértice $v \in V$

 Adicione v ao conjunto I

 Remova de V o vértice v e todos seus adjacentes

Retorne I

- Para grafos pequenos pode-se encontrar um conjunto máximo através de um backtracking.

- [► URI 3043 - Festa Junina](#)

Cobertura de vértices

- Uma **cobertura de vértices (vertex cover)** é um subconjunto de vértices tal que cada aresta do grafo é adjacente a pelo menos um vértice do subconjunto.
- Diz-se que o subconjunto de vértices **cobre** o conjunto de arestas.
- Uma **cobertura mínima de vértices** é a cobertura de vértices de menor cardinalidade para o grafo.
- A figura abaixo mostra duas coberturas de vértices (em vermelho). São coberturas mínimas?



- Aplicações de cobertura de vértices incluem localização de câmeras de vigilância para selecionar os pontos onde posicionar câmeras de modo a cobrir todas as ruas e aplicações em bioinformática como modelar a eliminação de sequências de DNA repetitivas (??? - Wikipedia).
- Outra aplicação é no [algoritmo húngaro](#), para identificar o número mínimo de linhas e colunas que cobrem os zeros.

Algoritmo aproximativo

- O problema de encontrar a cobertura mínima é um problema NP-difícil mas uma abordagem que pode resultar uma aproximação de fator-2 (ou seja, no máximo o dobro de vértices da solução ótima) é, a cada passo, escolher uma aresta do grafo e remover os dois vértices adjacentes a essa aresta.
- Repete-se isso até que não haja mais arestas no grafo. O conjunto de vértices adjacentes às arestas selecionadas forma uma cobertura de vértices.
- Isso corresponde a todos os vértices de uma associação maximal (não necessariamente máxima).

- Uma abordagem para tratar o problema é a utilização de heurísticas. Uma heurística bastante utilizada é selecionar a cada passo o vetor de maior grau, incluí-lo na cobertura de vértices e remover todas as suas arestas adjacentes, atualizando os graus dos vértices associados.
- Essa heurística assegura um fator de aproximação $H(\Delta) = 1 + \frac{1}{2} + \dots + \frac{1}{\Delta} = O(\log \Delta)$
- onde Δ é o grau máximo do grafo.

Cobertura de vértices em grafos bipartidos

- Para grafos bipartidos, o **Teorema de König** afirma que o número de vértices da cobertura mínima é igual ao número de arestas de uma associação máxima, tornando o problema resolvível em tempo polinomial (possivelmente por Hopcroft-Karp).
- Pode-se obter uma cobertura mínima de vértices em um grafo bipartido cujo conjunto V de vértices está dividido em dois conjuntos X e Y a partir de uma associação máxima pelo processo a seguir:
- URI 1056 - Fatores e Múltiplos

- Encontre o conjunto U (possivelmente vazio) de vértices não associados em X .
- Construa o conjunto Z de vértices que estão em U , ou conectados a U por caminhos alternantes (caminhos que alternam entre arestas associadas e não associadas)
- $K = (X \setminus Z) \cup (Y \cap Z)$ é a cobertura mínima de vértices.
- Obs: O operador \setminus representa diferença de conjuntos.

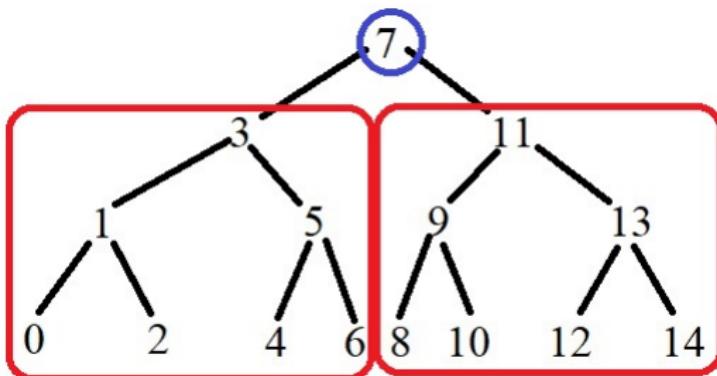
Cobertura de vértices em árvores

- Uma cobertura de vértices também pode ser encontrada em tempo linear em árvores.
- A ideia é considerar as duas possibilidades a seguir para a raiz e recursivamente para todos os nodos:
 - 1 A raiz faz parte da cobertura de vértices: Nesse caso a raiz cobre todas as arestas filhas. Recursivamente calcula-se o tamanho da cobertura de todos os filhos e soma-se 1 ao resultado.
 - 2 A raiz não faz parte da cobertura de vértices: Nesse caso, todos os vértices filhos da raiz devem ser incluídos na cobertura de vértices para cobrir todas as arestas filhas da raiz. Recursivamente calcula-se o tamanho da cobertura de vértices de todas os netos e soma-se o número de filhos.

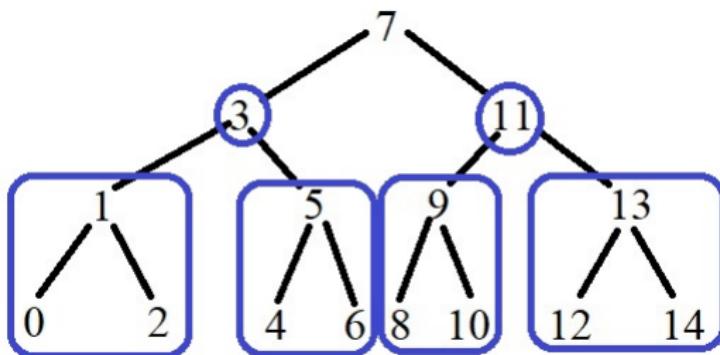
- A abordagem acima assume que a árvore é enraizada. Se não for, é só enraizá-la.
- A ideia acima pode ser expressa como uma recorrência e resolvida por programação dinâmica.
- Para cada vértice v , calcula-se dois valores para subárvore enraizada em v :
 - O custo da cobertura que **incluir** o vértice v ($\text{com}[v]$);
 - O custo da cobertura que **não incluir** o vértice v ($\text{sem}[v]$).
- $\text{com } [v] = \text{custo}(v) + \sum(\min(\text{com}(w), \text{sem}(w)))$ para w cada filho de v
- $\text{sem } [v] = \sum(\text{com}(w))$ para w cada filho de v

- Na formulação acima, a função $\text{custo}(v)$ é aplicada para problemas em que o grafo é valorado nos vértices e busca-se uma cobertura cuja soma dos vértices seja mínima (caso haja mais de uma cobertura com mesmo número de vértices)
- No cálculo de $\text{com}[v]$, como o vértice v faz parte da cobertura, ele cobre todos os vértices entre ele e os filhos, e os filhos não precisam necessariamente entrar na cobertura (mas podem entrar).

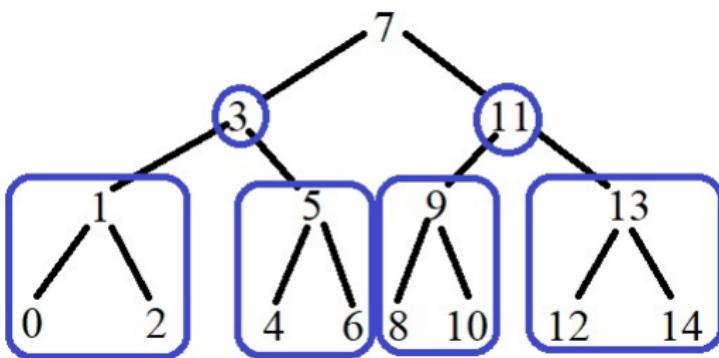
- Aplicado ao grafo abaixo, se a raiz entrar na cobertura de vértices a cobertura consistirá da raiz e da cobertura de cada uma das subárvore:



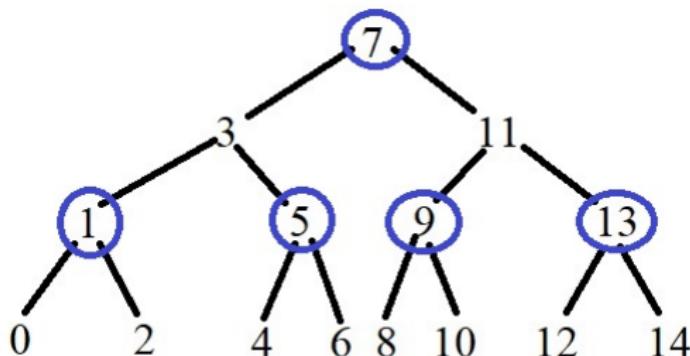
- Se a raiz não entrar na cobertura de vértices a cobertura consistirá dos filhos da raiz e da cobertura de cada uma das subárvores netas:



- Se a raiz não entrar na cobertura de vértices a cobertura consistirá dos filhos da raiz (3 e 11) e da cobertura de cada uma das subárvores netas resultando em uma cobertura de 6 vértices:



- O código VertexCoverTree, disponível no ava, gera a árvore desse exemplo e encontra uma cobertura de tamanho 5, que corresponde à solução abaixo:



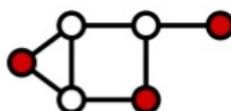
- O código VertexCoverTree chama a função vCover repetidas vezes para a mesma subárvore. P.ex. a sub-árvore enraizada por 1 será chamada como filha de 3 e como neta de 7.
- O código VertexCoverTreeAVL faz a inserção balanceada em uma AVL, e mostra o número de chamadas em função do número de nodos. Para 1000 nodos, a função é chamada 73.000 vezes.
- Uma forma de reduzir esse custo é armazenar, para cada sub-árvore, o tamanho da cobertura mínima de vértices. Isso pode ser feito colocando um campo adicional no nodo e atualizando-o a primeira vez que a cobertura da subárvore daquele vértice for calculada.

Conjunto dominante de vértices (dominating set)

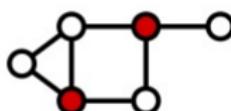
- Um conjunto dominante é um subconjunto de vértices tal que todo vértice do grafo está no conjunto ou é adjacente a um dos vértices.
- O número de dominação de um grafo G é a cardinalidade do menor conjunto dominante de G , notação $\gamma(G)$.
- Tal como no caso do número cromático $\chi(G)$, até hoje não existe um algoritmo polinomial para determinar $\gamma(G)$.
- No entanto, é possível verificar se um determinado conjunto de vértices é, ou não, dominante.

Ex: Um grafo e 3 conjuntos dominantes

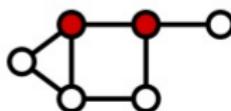
(a)



(b)



(c)



Conjuntos Dominantes

- O nome vem das aplicações, que têm a ver, habitualmente com controle ou vigilância.
- Procura-se um conjunto de cardinalidade mínima, cujos vértices podem estar associados.
- Exemplos:
 - Câmaras de segurança instaladas em lugares públicos.
 - Radares que devam cobrir uma determinada área

Problema da Distribuição de Pontos de Observação

- Em um parque ecológico, deseja-se colocar pontos de observação de modo que se tenha uma visão de todo o parque.
- Para isto, o parque foi dividido em várias áreas, sendo que para cada área foram identificados os pontos que mais propiciam a sua observação.
- Conhecida esta relação de vigilância, observou-se que em vários casos um mesmo ponto de observação permite a vigilância de mais de uma área.
- Sendo assim, a direção do parque optou por procurar instalar o menor número de pontos de observação de forma a interferir o mínimo possível com a vida do parque.
- Como se poderia identificar este conjunto mínimo de pontos de observação?

Problema da telefonia celular

- A empresa de telefonia celular AbTel pretende se instalar em Florianópolis.
- Em função dos turistas que vão para lá no verão, a AbTel considera importante que toda a ilha de Florianópolis esteja coberta por sinal de telefonia celular, de forma que em qualquer ponto da ilha um usuário possa obter sinal e falar normalmente.
- Isto significa distribuir diversas antenas de telefonia celular pela ilha.
- Mas dado que o custo de instalação de uma antena é relativamente alto, a empresa solicita ao departamento técnico um estudo para identificar o menor número de antenas (com as correspondentes localizações aproximadas) necessários para realizar esta cobertura.

Um algoritmo heurístico para determinar $\gamma(G)$

- Selecionar em sequência os vértices com maior grau (que cobrem uma maior quantidade de vértices), até que se obtenha um conjunto dominante. Ou...
- Selecionar em sequência os vértices que cobrem a maior quantidade de vértices **ainda não cobertos**, até que se obtenha um conjunto dominante.

Particionamento de Grafos

- Objetivo: Dividir o grafo em k subconjuntos de forma que o número de vértices em cada conjunto seja o mesmo (diferindo no máximo de um vértice) e o número de arestas entre eles (chamado corte de arestas) seja o menor possível.
 - É um problema NP-difícil.
 - Pode ser aplicado em computação paralela para dividir um modelo de simulação entre diversos processadores para que suas diversas partes sejam processadas em paralelo.
 - O objetivo de reduzir o corte de arestas é reduzir a comunicação entre os processadores melhorando o desempenho do modelo.
- Os algoritmos de particionamento podem ser classificados como globais ou locais:

Métodos Globais

- Usam informação de todo o grafo. São utilizados para gerar a partição inicial.
- Alguns métodos globais:
 - Particionamento em faixas: utilizado para grafos com formato de grade ortogonal (linhas e colunas). Nele, cada subgrafo recebe um determinado número de linhas do grafo. Garante um bom balanceamento mas não garante fronteiras pequenas.
 - Particionamento em blocos: O domínio é dividido em blocos. O balanceamento de carga é pior mas reduz o corte de arestas.
 - LND (Levelized Nested Dissection) ? Utiliza uma busca em amplitude. Escolhe-se uma raiz e vai-se visitando os vizinhos, como uma busca em amplitude, até marcar metade dos vértices.

Métodos locais

- Também chamados de métodos de refinamento, utilizam apenas informações do subgrafo e dos subgrafos vizinhos.
- São utilizados para melhorar a qualidade de uma partição. Normalmente utilizam trocas de vértices buscando reduzir o corte de arestas.
- Uma abordagem possível é selecionar a cada passo um par de vértices que melhore a qualidade do corte.
- Essa abordagem falha ao não conseguir superar mínimos locais, partições em que qualquer mudança piora a qualidade do corte, mas que as mudanças seguintes melhorariam.
- O algoritmo a seguir consegue superar isso:

Algoritmo de Kernighan-Lin (KL)

- O algoritmo parte de uma partição em que o conjunto N de vértices foi dividido em dois conjuntos A e B, sendo $|A| = |B|$
- O objetivo é encontrar subconjuntos X em A e Y em B, de mesmo tamanho, tais que a troca de X e Y reduza o custo total (T) das arestas que ligam A a B.

Algoritmo de Kernighan-Lin (KL)

- Definições:
- $E(a)$ - Custo externo do vértice a em A - número de arestas (a,a') tal que a' pertence a B .
- $I(a)$ - Custo interno do vértice a em A - número de arestas (a,a') tal que a' pertence a A .
- $D(a)$ - Custo do vértice a em $A = E(a) - I(a)$
- Ao trocar um vértice a por um vértice b , o novo custo T pode ser calculado por:
 - $\text{novo_}T = T - (D(a) + D(b) - 2 * w(a,b)) = T - \text{ganho}(a,b)$
 - onde $w(a,b)$ é o peso da aresta (a,b)
 - e os novos $D(a')$ (custo dos outros vértices do grafo) são dados por:
 - $\text{novo_}D(a') = D(a') + 2 * w(a',a) - 2 * w(a',b)$

O Problema do Corte Mínimo (min-cut)

- Em Teoria de Grafos, um **corte mínimo (min-cut)** é um corte, uma partição dos vértices do grafo em dois conjuntos disjuntos que minimiza alguma métrica.
- Está relacionado à identificação do corte de vértices, que é um conjunto de vértices que, removido, torna o grafo desconexo.
- Ou um corte de arestas, conjunto de arestas que, removidos, tornam o grafo desconexo.
- Uma métrica comum é minimizar o número de arestas entre as duas partições. Ou, se as arestas são valoradas, minimizar a soma das arestas entre as duas partições.
- Pode ser aplicado a grafos dirigidos ou não dirigidos.

- Se o grafo é uma rede (grafo dirigido com um único vértice inicial (**source**) e um único vértice final (**sink**), etc.) o **Teorema de Ford-Fulkerson** diz que o valor do corte mínimo (soma dos pesos das arestas do corte) é igual ao valor do fluxo máximo.
- Para encontrar o conjunto de arestas que compõem um corte mínimo, pode-se calcular o fluxo máximo aplicando o algoritmo de Ford-Fulkerson.
- Após encontrar a última rede residual, em que não há caminho aumentante de s a t , as arestas do corte são todas as arestas da rede original, que ligam algum vértice alcançável a partir de s na rede residual, a algum vértice não alcançável na rede residual.

- Se o grafo é não dirigido com arestas valoradas, e não há source e sink, e busca-se apenas tornar o grafo desconexo com um conjunto de arestas de soma total mínima, pode-se usar o algoritmo de Stoer-Wagner.
- Se as arestas são não valoradas, pode-se usar o algoritmo de Karger.

Algoritmo de Stoer-Wagner

- O algoritmo de Stoer-Wagner foi publicado

Exercícios da Área 1

1) Qual dos grafos a seguir não é isomorfo dos outros? Por que?

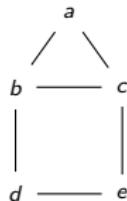


Figura: Grafo A

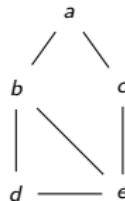


Figura: Grafo B

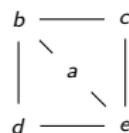


Figura: Grafo C

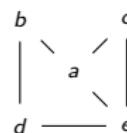


Figura: Grafo D

2) Desenhe 10 grafos não isomorfos de 4 vértices (existem 11 :-)).

Voltar para Isomorfismo

3) Os grafos G e H descritos a seguir são isomorfos? Justifique sua resposta.

$$V_G = \{a, b, c, d, e, f, g\}$$

$$E_G = \{ab, bc, cd, cf, fe, gf, ga, gb\}$$

$$V_H = \{h, i, j, k, l, m, n\}$$

$$E_H = \{hn, nj, jk, lk, lm, li, ij, in\}$$

[Voltar para Isomorfismo](#)

- 4) Quantas arestas possui K_{20} ?
- 5) É possível haver um grupo de 7 pessoas no qual cada uma conhece exatamente 3 outras pessoas? Justifique sua resposta utilizando teoria de grafos. ???
- 6) Qual é o maior número possível de vértices em um grafo com 19 arestas e todos os vértices tem pelo menos $\text{grau } 3$? Explique sua resposta.
- 7) É possível que um grafo bipartido com um número ímpar de vértices seja hamiltoniano (isso é, contenha um ciclo hamiltoniano)? E euleriano? Justifique sua resposta.

- 8) Suponha um conjunto de n números inteiros positivos. Diga uma condição necessária para que esses números possam representar os graus dos n vértices de uma árvore. Essa condição é suficiente?
- 9) Descreva um possível algoritmo para verificar se um dado grafo é **bipartido** ou não.
- 10) Uma *caterpillar* é um grafo conexo, acíclico e que possui um caminho ao qual todo vértice do grafo pertence ou é adjacente a um vértice deste caminho. Este caminho é chamado a *espinha* da *caterpillar* e pode não ser único. Descreva um algoritmo para identificar se um grafo é uma *caterpillar* ou não.

- 11) Qual o número máximo de arestas de um grafo bipartido com 30 vértices?
- 12) Quais dentre os grafos a seguir são bipartidos?

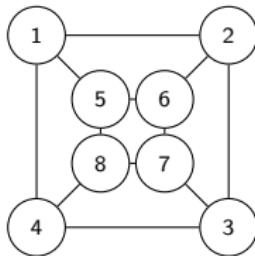


Figura: Grafo A

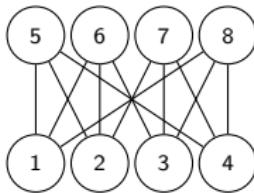


Figura: Grafo B

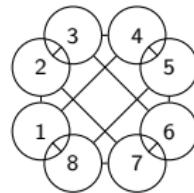
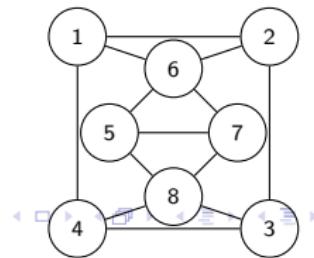
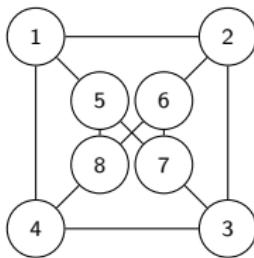
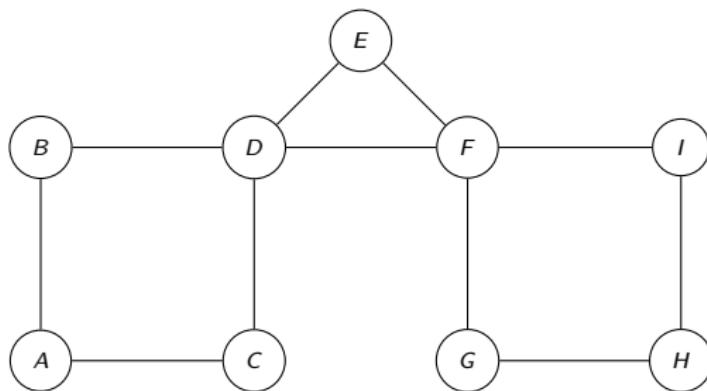


Figura: Grafo C

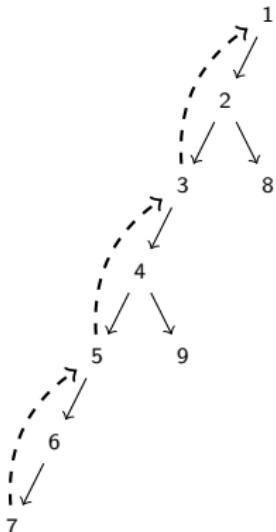


13) Decomponha o grafo a seguir em seus componentes biconexos, a partir do vértice e, utilizando o algoritmo dado. Mostre todos os passos do algoritmo (identificação das articulações, mostrando as arestas de retorno, identificação dos demarcadores e dos componentes a cada passo).

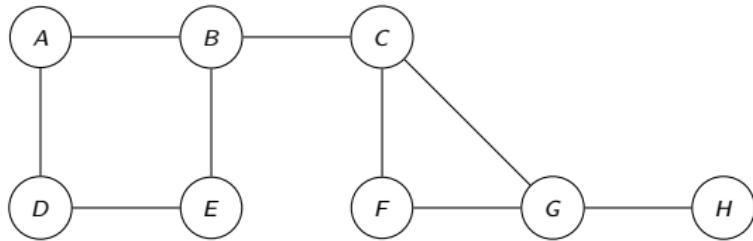


[Voltar para Componentes Biconexos](#)

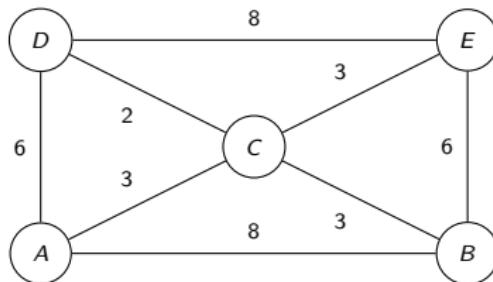
14) A figura a seguir representa a árvore resultante da busca em profundidade e as arestas de retorno de um grafo G . Identifique o lowpt de cada vértice, as articulações, demarcadores e componentes biconexas de G .



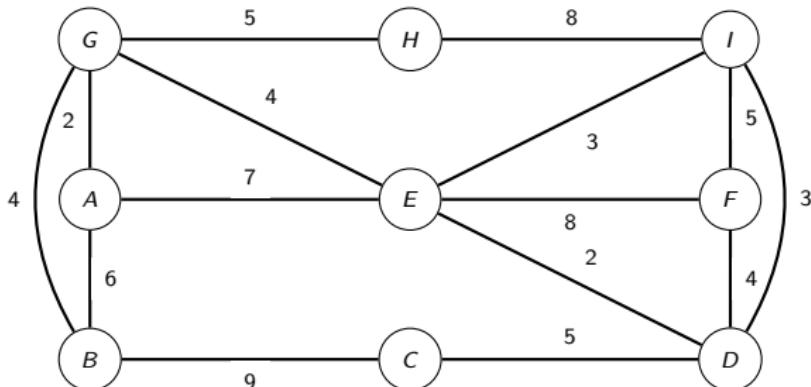
15) Decomponha o grafo a seguir em seus componentes biconexos, a partir do vértice e, utilizando o algoritmo visto em aula. Mostre todos os passos do algoritmo (árvore em profundidade mostrando as arestas de retorno, identificação das articulações, identificação dos demarcadores e dos componentes a cada passo).



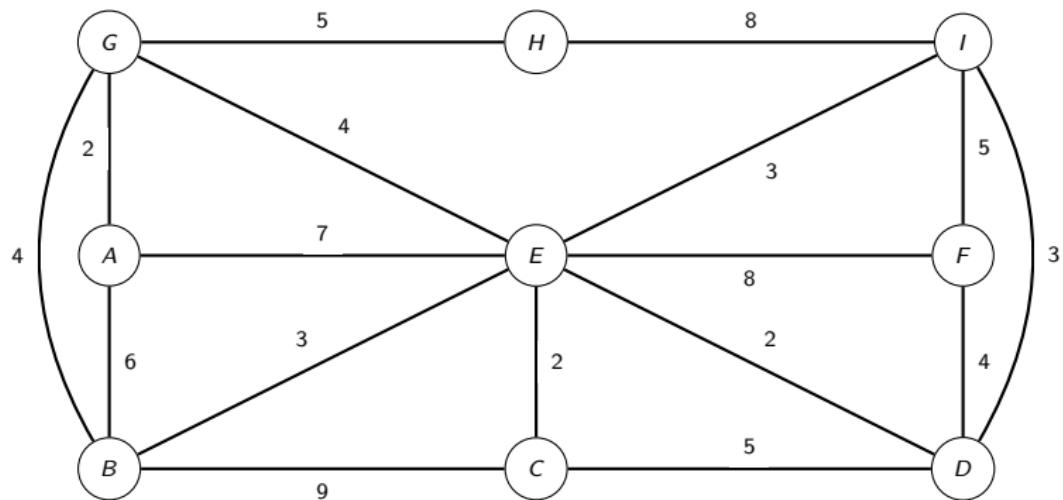
16) O problema do **carteiro chinês** consiste em, dado um grafo valorado (não necessariamente euleriano), encontrar o ciclo de menor custo total que percorra todas as arestas ao menos uma vez e volte para a aresta inicial. Descreva, em termos de teoria de grafos, como o problema pode ser resolvido. Mostre a solução para o grafo a seguir:



17) O grafo a seguir representa as esquinas e ruas de um bairro onde uma firma deve entregar iogurte a domicílio para um grande número de clientes. Sabendo que o depósito está situado no ponto (a) e que o grafo está valorado pela distância entre duas esquinas (em dezenas de metros), determinar o percurso mais curto que pode ser seguido pelo entregador, de modo que todas as entregas sejam feitas e que ele volte ao depósito.



18) O grafo a seguir representa as esquinas e ruas de um bairro onde uma firma deve entregar iogurte a domicílio para um grande número de clientes. Sabendo que o depósito está situado no vértice A e que o grafo está valorado pela distância entre duas esquinas (em dezenas de metros), determinar o percurso mais curto que pode ser seguido pelo entregador, de modo que todas as entregas sejam feitas e que ele volte ao depósito (parece o problema anterior mas o grafo é diferente). Considere que há iogurtes a serem entregues em todas as quadras.



- 19) Seja um jogo de dominó que contém 10 peças, cujas configurações são as seguintes:
 $(1,2), (1,3), (1,4), (1,6), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (5,6)$. É possível colocar as peças em uma fila circular (isso é, em que a primeira peça toca a última), de tal maneira que o número de uma peça sempre toca um número igual na outra peça? E se a fila não for circular? E se for removida a peça $(1,2)$?
- 20) Considere todos os movimentos possíveis para o cavalo em um tabuleiro de xadrez de 8×8 . Existe um percurso que o cavalo possa seguir onde ele execute todos os movimentos possíveis, executando cada movimento exatamente uma vez? Justifique sua resposta utilizando teoria de grafos.

21) Tertuliano Gonçalves havia prometido casamento a Josefina das Graças; o evento deveria se realizar, segundo ele, assim que acabasse o contrato de trabalho que acabava de assinar com uma empresa encarregada de pavimentar a rede de estradas que ligava Santana do Caixa Prego - cidade onde morava Josefina - às cidades vizinhas. O trabalho iria começar em Santana e prosseguir em continuidade, estrada após estrada, sem passar mais de uma vez na mesma estrada, terminando, segundo explicou Tertuliano - na própria Santana. A rede poderia ser representada pela matriz abaixo; Santana é a cidade número 1. Você que leu esta história, acha que Tertuliano está sendo sincero com Josefina? E se o itinerário 1-5-9-10 estivesse a cargo de outra empresa, estaria ele sendo sincero? Se sim, encontre uma ordem possível para a pavimentação das estradas.

	1	2	3	4	5	6	7	8	9	19
1		1	1		1					
2	1		1	1	1					
3	1	1			1	1				
4		1			1		1			1
5	1	1	1	1		1	1	1	1	
6			1		1			1		1
7				1	1			1		1
8					1	1	1		1	1
9					1			1		1
10				1		1	1	1	1	

- 22) A linha de produção de uma fábrica de tambores de aço está equipada para a produção de tambores de 100 e de 200 litros, ambos podendo ter tampa fixa com bujão ou tampa inteiramente removível.
- A mudança de tipo de tambor a ser produzido exige modificações na linha; em algumas máquinas isso só é necessário quando muda o tamanho do tambor, enquanto em outras a mudança do tipo de tampa implica também em modificações.
- A linha é formada de prensas para tampas e para fundos, soldadora, fresadora e recravadeira. As 4 opções de produção estão abreviadas abaixo da seguinte forma:
 - 1.tampa fixa, 100 litros
 - 2. tampa fixa, 200 litros
 - 3.tampa removível, 100 litros
 - 4. tampa removível, 200 litros

- Os valores, em homens-hora, do trabalho exigido para a passagem de uma máquina da opção i para a opção j são os seguintes, para a prensa de tampas e para a recravadeira:

	1	2	3	4
1	-	5	2	6
2	4.5	-	6.5	2.5
3	1.5	5.5	-	5.5
4	5	1.5	6	-

Tabela: Prensa de Tampa

	1	2	3	4
1	-	5	2	6
2	4.5	-	6.5	2.5
3	1.5	5.5	-	5.5
4	5	1.5	6	-

Tabela: Recravadeira

- Em relação às outras máquinas temos:

	Prensa fundos	Soldadora	Fresadora
100 para 200	4	6	0.5
200 para 100	3.5	6.5	1

- A fábrica dispõe de encomendas dos 4 tipos de tambores e deseja colocar a linha em funcionamento. Se nenhuma encomenda é mais urgente que as restantes, determine a seqüência de produção dos 4 tipos, mais econômica em relação ao custo do trabalho exigido pelas modificações:
 - a)Se a situação das encomendas é repetitiva;
 - b)Se não se pode considerá-la como repetitiva, isto é, se a fabricação dos 4 tipos ocorrerá apenas uma vez.

- 23) Uma fábrica faz oito variedades de palhetas de guitarras usando apenas uma máquina, a Presto Pick Puncher. Esta máquina precisa de diferentes ajustes a fim de fazer diferentes estilos de palhetas. A companhia deseja gastar 1 hora de cada dia de trabalho em cada estilo de palheta. No fim de cada hora os ajustes na máquina precisam ser convertidos para aqueles necessários para o próximo estilo de palheta. No fim do dia, a máquina deve ser convertida de volta para o estilo produzido no início do dia. O tempo necessário para conversão depende de cada estilo para outro, os quais são fornecidos a seguir. Desde que os tempos de conversões não são o mesmo, faz diferença em qual ordem os estilos são feitos. Você pode arranjar uma ordem, começando e finalizando com o estilo A, tal que nenhuma conversão leve mais do que 3 min? Se não, você pode fazê-la tal que nenhuma tarefa demore mais do que 4 min?

	A	B	C	D	E	F	G	H
A	5	2	6	6	5	2	5	
B	5		3	4	5	6	1	6
C	2	3		1	6	5	5	6
D	6	4	1		2	4	5	5
E	6	5	6	2		3	6	2
F	5	6	5	4	3		3	6
G	2	1	5	5	6	3		1
H	5	6	6	5	2	6	1	

Tabela: Custo de conversão de máquinas para fabricação de cada tipo de palheta

24) A turma de Teoria de Grafos planeja secretamente uma insurreição. Os sete líderes chaves estão listados na tabela abaixo. É decidido que durante o teste, os detalhes escritos serão passados entre os sete começando com Abe. O processo é complicado por dois fatores. Primeiro, dilemas ideológicos entre alguns deles impedem que eles troquem mensagens entre si. Segundo, no interesse de evitar a interceptação, é necessário passar o pedaço de papel, começando e finalizando com Abe, entre todos os sete sem ninguém receber duas vezes (exceto Abe). A tabela abaixo indica quais pares de estudantes têm relações amigáveis (A) e podem fazer contato para passar uma mensagem e quais pares não são amigos (NA) e não podem ter a mensagem passada diretamente entre eles. Pode a mensagem ser passada da forma definida? Em caso positivo, em que ordem?

	Abe	Tom	Dave	John	Lee	Gerry	Ren
Abe	-	A	NA	NA	A	NA	NA
Tom	A	-	A	A	NA	NA	A
Dave	NA	A	-	NA	A	A	A
John	NA	A	NA	-	A	A	NA
Lee	A	NA	A	A	-	A	A
Gerry	NA	NA	A	A	A	-	NA
Ren	NA	A	A	NA	A	NA	-

25) O professor descobriu a rebelião descrita no exercício anterior e deseja punir os sete líderes com um problema de teoria de grafos extra para fazer em casa. Seu plano é complicado pelo fato que estudantes amigos irão colaborar (assumindo que os inimigos não vão) se eles receberem o mesmo problema. Portanto, ele gostaria de ter certeza que estudante amigos receberão problemas diferentes. Ele gostaria de minimizar o número de problemas que ele atribuirá. Qual é o número mínimo?

26) Em um zoológico onde os animais vivem soltos existem 10 espécies, as quais estão listadas na tabela abaixo. As coisas ficam complicadas pelo fato que não é aconselhável permitir espécies que são inimigas naturais viver em um mesmo cercado. A tabela indica tais incompatibilidades pela letra X. Por razões de economia é desejado ter poucos cercados possíveis. Qual é o número mínimo de cercados? ????

	1	2	3	4	5	6	7	8	9	10
1			X							
2				X						
3	X			X						
4		X	X							
5					X				X	
6				X		X				
7					X		X		X	
8						X		X		
9							X		X	
10				X		X		X		

27) Um chefe de escoteiros está planejando levar oito membros de sua tropa em carros para um congresso de escoteiros. Para evitar problemas, somente os escoteiros que são amigos (A) vão no mesmo carro. Qual é o menor número de carros que são necessários para transportar os escoteiros? Se os carros pudessem carregar somente dois escoteiros, haveria uma resposta diferente?

????

	1	2	3	4	5	6	7	8
1	-	A	NA	NA	NA	A	NA	A
2	A	-	A	A	NA	NA	NA	A
3	NA	A	-	NA	NA	A	NA	NA
4	NA	A	NA	-	A	A	NA	NA
5	NA	NA	NA	A		A	A	NA
6	A	NA	A	A	A	-	A	A
7	NA	NA	NA	NA	A	A	-	NA
8	A	A	NA	NA	NA	A	NA	-

28)O Departamento de Informática pretende oferecer 7 disciplinas para o nono semestre do Curso de Computação no próximo semestre.

(C) Compiladores

(G) Grafos

(L) Lógica

(N) Análise de Algoritmos

(P) Probabilidade

(S) Engenharia de Software

(T) Arquitetura de

Computadores

Com base na pré-matrícula, verificou-se que cada estudante planeja cursar as disciplinas indicadas abaixo:

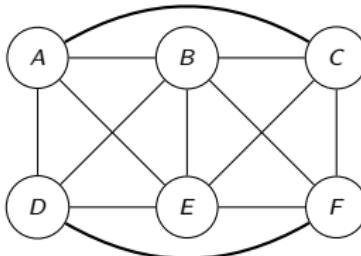
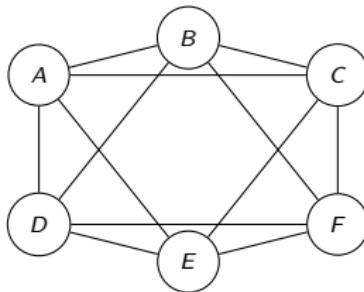
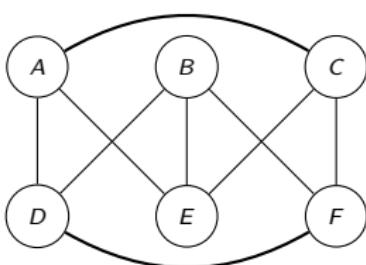
- | | | |
|-----------------|-----------------|----------------|
| ■ Aldo C,L,T | ■ Eduardo L,N | ■ Iracema C,T |
| ■ Batista C,G,S | ■ Frederico C,G | ■ Janete C,S,T |
| ■ Carlos C,L | ■ Giovana N,P | ■ Kevin P,S |
| ■ Daniel G,N | ■ Homero G,L | ■ Luis P,T |

Considerando que cada disciplina é dada num período, como se poderia determinar qual o menor número de períodos necessários para que os sete cursos possam ser oferecidos? Para os dados fornecidos, qual é esse número?

29) O grafo K_7 , após a exclusão das arestas $(1, 2), (1, 3), (1, 4), (2, 3)$ e $(2, 4)$ é planar?

30) O grafo resultante da remoção das arestas $(3,1), (3,2), (6,4), (6,5)$ e $(5,4)$ de K_6 é planar? Justifique sua resposta.

31) Verificar se os grafos a seguir são planares. Se forem, mostrar uma representação planar. Se não forem, mostrar um homeomorfismo de $K_{3,3}$ ou K_5 .



- a. É planar. Basta inverter os vértices B e E .
- b. É planar. Basta "puxar para fora" as arestas AC , CE e EA .
- c. Não é planar. $\{A,B,F\}$ e $\{D,E,C\}$ formam $K_{3,3}$. Além disso, ao remover as arestas BF e EF resulta um homeomorfismo de K_5 .

32) Idem (verifique a planaridade)

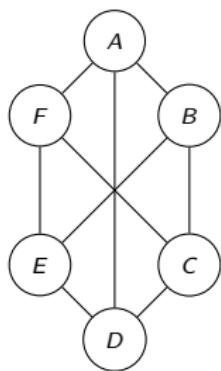


Figura: Grafo A

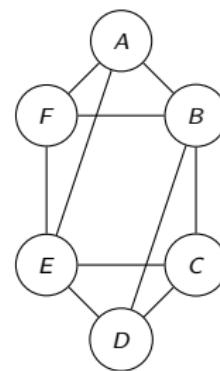


Figura: Grafo B

- a. É o próprio $K_{3,3}$.
- b. Planar. É só puxar para fora as arestas horizontais.

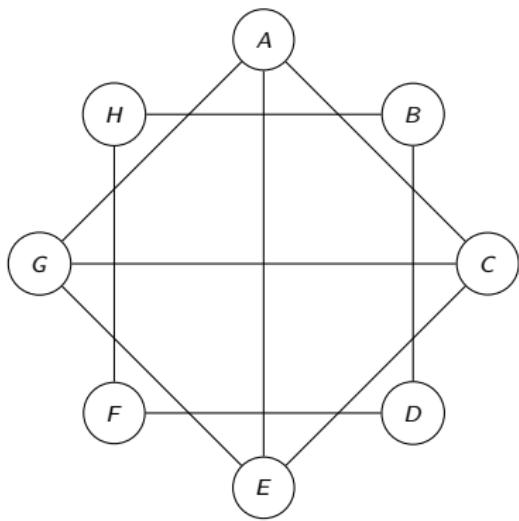


Figura: Grafo C

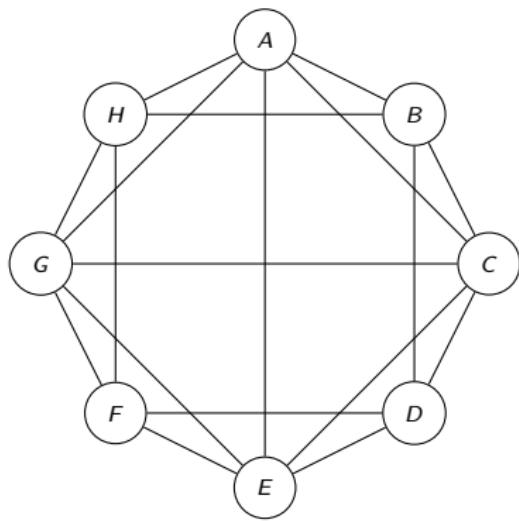


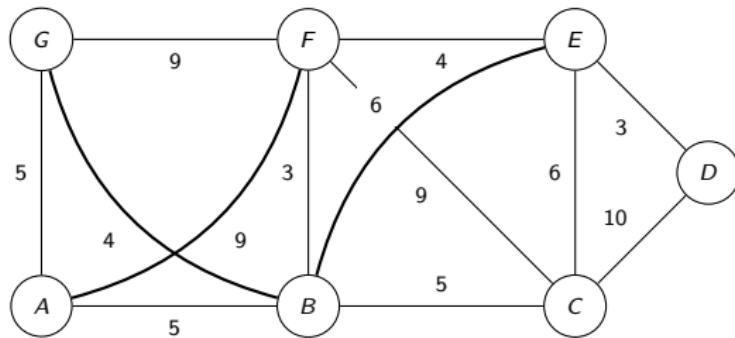
Figura: Grafo D

- c. Planar. Há dois subgrafos disjuntos.
- d. Não planar. Removendo as arestas AG , EF , FG , EG , EC resulta K_5 . E removendo as arestas EF e FG resulta $K_{3,3}$ ($\{G,A,D\}$ e $\{H,C,E\}$).

33) O grafo a seguir corresponde ao projeto de uma rede de computadores. Os vértices correspondem às máquinas e as arestas correspondem à possibilidade de conexão entre duas máquinas, juntamente com o correspondente custo de instalação. Essa rede deverá ser construída de forma que cada computador possa se comunicar (direta ou indiretamente) com cada um dos outros computadores, sempre a um custo mínimo, não importando o custo total da rede. Como a rede é bastante segura, não há a necessidade de haver mais de uma rota de comunicação entre dois computadores pois sempre será utilizada a de menor custo. Sendo assim, algumas conexões nunca serão utilizadas, de forma que não precisam ser implantadas.

a) O problema descrito pode ser resolvido através da geração da árvore geradora mínima?

b) Em caso contrário, como o problema pode ser resolvido?



34) Considere uma árvore composta de q vértices de grau 4 e p vértices de grau 1. Qual a relação matemática entre p e q ? (essa família de grafos representa hidrocarbonetos em que o vértice de grau 4 representa átomos de carbono e os vértices de grau 1 representam átomos de hidrogênio)

35) É possível existir um grafo com um número ímpar de vértices e um número par de arestas que contenha um ciclo euleriano? Justifique sua resposta.

36) Descreva duas formas possíveis de obter um ciclo qualquer a partir de um grafo sabidamente cíclico.

37) Considere o grafo a seguir $G(V, A)$:

$$V = \{A, B, C, D, E, F\}$$

$A =$

$$\{(A, D), (A, E), (B, D), (B, C), (C, D), (C, E), (D, E), (F, D), (F, E)\}$$

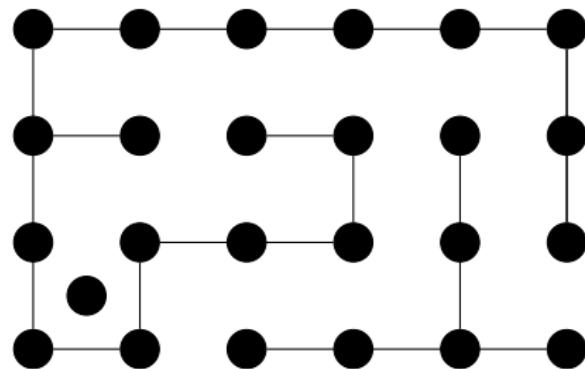
a) Descreva os percursos em profundidade (DFS) e em amplitude (BFS) para o grafo G , iniciando pelo vértice A e seguindo sempre pelo vértice de menor ordem alfabética quando houver mais de um vértice adjacente.

b) Considere o grafo G como um dígrafo (onde o arco sempre parte do vértice com letra de menor ordem alfabética para o maior). Defina a árvore geradora obtida a partir de um percurso em profundidade, iniciando pelo vértice A e seguindo sempre pelo vértice de menor ordem alfabética quando houver mais de um vértice adjacente.

38) Mostre uma árvore geradora mínima do grafo representado pela matriz de adjacências a seguir:

	A	B	C	D	E	F
A		3	2		5	4
B	3		3	4		4
C	2	3		5	2	4
D		4	5		4	6
E	5		2	4		2
F	4	4	4	6	2	

39) Considere um labirinto perfeito qualquer, como o mostrado abaixo, onde há uma saída do labirinto a partir de qualquer uma das casas. Deseja-se descobrir, dentro do labirinto, qual das posições dentro do labirinto é a mais afastada da saída mais próxima. Por exemplo, no labirinto mostrado, a casa mais afastada das duas saídas é a casa marcada com um círculo. Diga como esse problema pode ser resolvido com o uso da teoria de grafos, ou seja, como modelar o problema com um grafo e que algoritmo utilizar para chegar à solução do problema.



40) (busca em espaço de estados) Problema dos canibais e dos missionários - Três canibais e três missionários estão viajando juntos e chegam a um rio. Eles desejam atravessar o rio, sendo que o único meio de transporte disponível é um barco que comporta no máximo duas pessoas. Há uma outra dificuldade: em nenhum momento o número de canibais pode ser superior ao número de missionários, pois dessa forma os missionários estariam em grande perigo de vida. Como efetuar a travessia do rio?

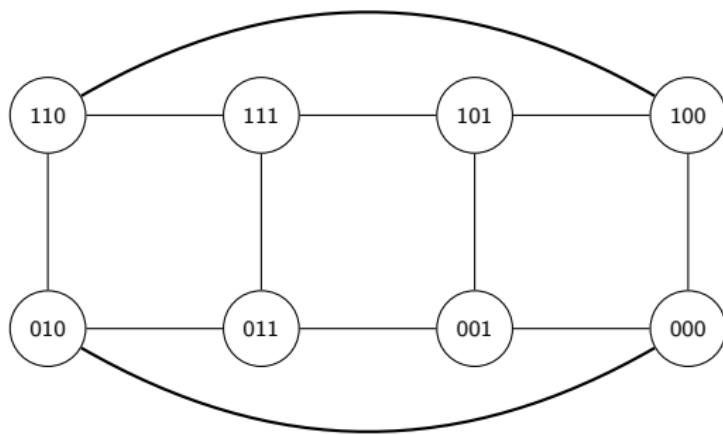
Situação inicial: cccmmm -

- Vão dois canibais (cmmm - cc), volta um canibal (ccmmm - c)
- Vão dois canibais (mmm - ccc), volta um canibal (cmmm - cc)
- Vão dois missionários (cm - mmcc), volta um canibal e um missionário (ccmm - cm)
- Vão dois missionários (cc - cmmm), volta o canibal (ccc - mmm)
- Vão dois canibais (c - ccmmm), volta um canibal (cc - cmmm)
- Vão dois canibais (- cccmmm)

41) Problema dos três maridos ciumentos - Três esposas e seus respectivos marido desejam ir ao centro da cidade em uma moto, a qual comporta apenas duas pessoas. Como eles poderiam deslocar-se até o centro considerando que nenhuma esposa deveria estar com um ou ambos os outros maridos, a menos que seu marido também esteja presente?

42) Problema dos potes de vinho - Considere que temos 3 potes com capacidades de 8, 5 e 3 litros, respectivamente, os quais não possuem qualquer marcação. O maior deles está completamente cheio enquanto que os outros dois estão vazios. Estamos interessados em dividir o vinho em duas porções iguais de 4 litros, tarefa esta que pode ser resolvida por transvasos sucessivos de um vaso no outro. Qual o menor número de transvasos necessários para completar a divisão?

43) Considere a sequência de números binários de 0 a 7. É possível colocar os 8 números em uma sequência de modo que, de um número para o outro, mude apenas um bit? O grafo da possibilidades de transição é planar?



44) Um grafo conexo é dito euleriano se possui um ciclo euleriano.
Desenho 3 grafos eulerianos não isomorfos com exatamente seis vértices (há 8)

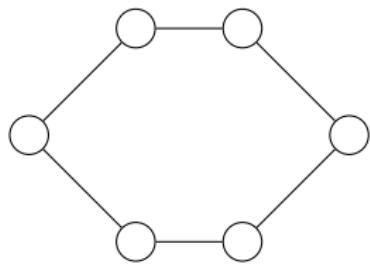


Figura: 222222

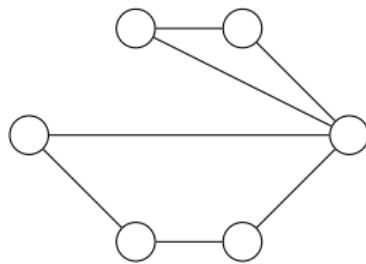


Figura: 222224

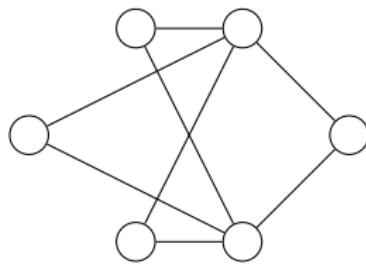


Figura: 222244

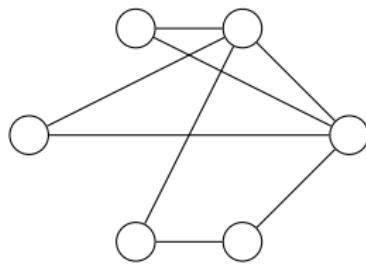


Figura: 222244

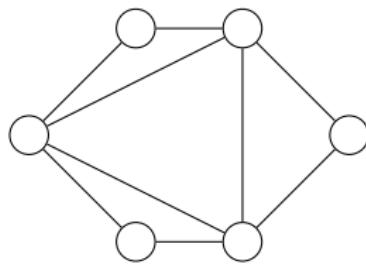


Figura: 222444

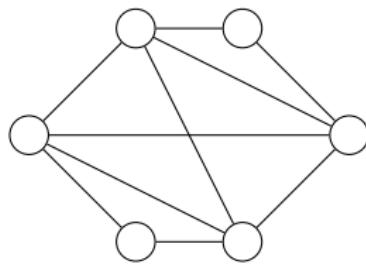


Figura: 22444

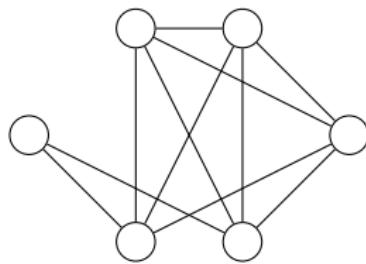


Figura: 24444

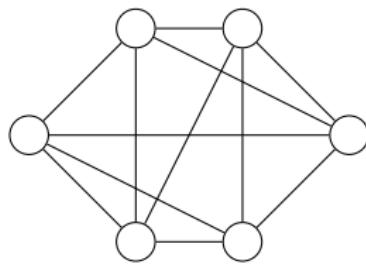
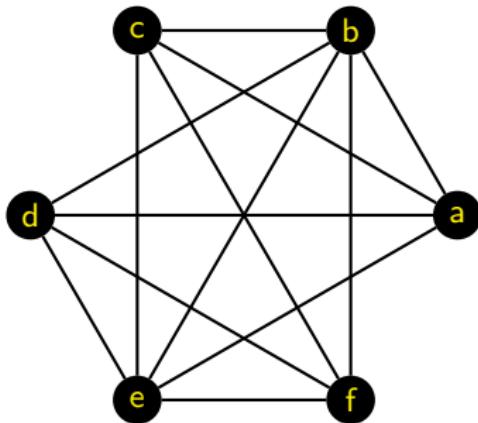


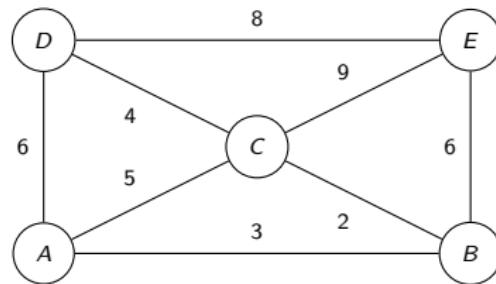
Figura: 444444

45) No grafo do exercício anterior, aplique o algoritmo de Tremaux, a partir do vértice A , mostrando como ficam ao final as marcações das passagens. Use como critério de escolha entre dois vértices, sempre o de menor ordem alfabética.

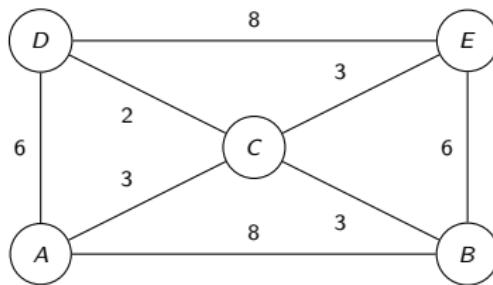
46) Aplique o teste de planaridade de Demoucron ao grafo a seguir para determinar se é planar ou não. Mostre o ciclo inicial e suas pontes. Mostre o desenho do grafo à medida que vão sendo acrescentados os caminhos a ele e indique as faces em que cada ponte restante pode ser inserida (para cada iteração do algoritmo).



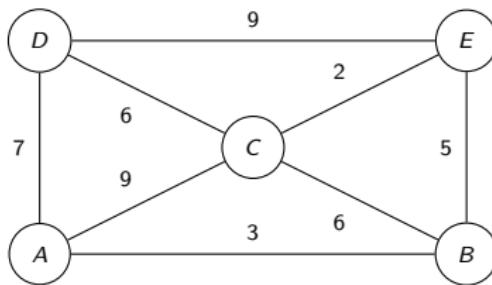
47) Aplique o algoritmo de **Floyd-Warshall** no grafo a seguir, mostrando a matriz de distâncias a cada iteração.



48) Aplique o algoritmo de Dijkstra ao grafo a seguir, para calcular a distância entre os vértices A e E . Mostre o vetor de distâncias ao final de cada iteração.



49) Aplique o algoritmo de Dijkstra ao grafo a seguir, para calcular a distância entre os vértices A e E . Mostre o vetor de distâncias ao final de cada iteração.



50) Para qualquer inteiro positivo k , um cubo de dimensão k (ou k -cubo) é o grafo definido da seguinte maneira: os vértices do grafo são todas as seqüências $b_1 b_2 \dots b_k$ de bits; dois vértices são adjacentes se e somente se diferem em exatamente uma posição. Por exemplo, os vértices do cubo de dimensão 3 são 000, 001, 010, 011, 100, 101, 110, 111; o vértice 000 é adjacente aos vértices 001, 010, 100 e a nenhum outro; e assim por diante.

Faça uma figura do 1-cubo, do 2-cubo e do 3-cubo.

Mostre a matriz de adjacências de um 3-cubo.

Quantos vértices tem o k -cubo?

Quantas arestas tem o k -cubo?

O 3-cubo é planar?

51) Descreva sucintamente qual a função de cada um dos 8 algoritmos a seguir, deixando bem claras as diferenças entre eles, caso haja algoritmos com funções semelhantes:

Floyd-Warshall

Prim

Kruskal

Demoucron

Tremaux

Moore

Dijkstra

Ford

52) Indique os grafos bipartidos completos que são:

- 1** eulerianos;
- 2** semi-eulerianos.

- 53) Um grafo conexo é dito *euleriano* se possui um ciclo euleriano. Desenhe 3 grafos eulerianos não isomorfos com exatamente seis vértices (há 8).
- 54) É possível que um grafo bipartido com um número ímpar de vértices seja hamiltoniano (isso é, contenha um ciclo hamiltoniano)? E euleriano? Justifique sua resposta.
- 55) Determine todos os grafos eulerianos (não isomorfos) com menos de seis vértices (há exatamente 7 grafos).
- 56) Sejam G_1 e G_2 dois grafos conexos, sem vértices em comum. Seja x um vértice de G_1 e y um vértice de G_2 . Seja G o grafo que se obtém de $G_1 \cup G_2$ acrescentando a aresta (x, y) . É possível que o grafo G seja euleriano? E semi-euleriano (contém um caminho euleriano mas não contém um ciclo euleriano)? Em que condições?

57) Determine os valores de n para os quais:

- 1** K_n é euleriano;
- 2** K_n é semi-euleriano.

58) Seja G um grafo com 14 vértices e 25 arestas. Se todo vértice de G tem grau 3 ou 5, quantos vértices de grau 3 o grafo G possui?

59) Um grafo é dito r -regular se todos os vértices tem grau r . Dê um exemplo de um grafo 3-regular que não seja completo.

60)Uma floresta é um grafo desconexo em que todas as componentes conexas são árvores. Determine o número de arestas de uma floresta com n vértices e k componentes conexos, em função de n e k .

- 61) Quantos subgrafos **induzidos** com 2 vértices possui um grafo de n vértices e m arestas?
- 62) Desenhe um grafo que contenha 4 **pontos centrais**.

Exercícios da Segunda Área

1) Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade.

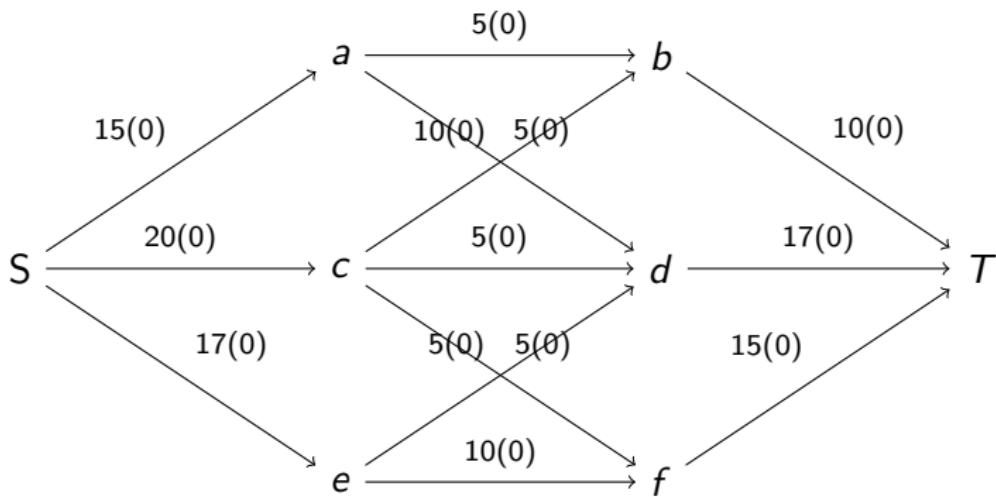


Figura: Encontre o fluxo máximo

2) Na rede a seguir, x_1 , x_2 e x_3 são fontes do mesmo produto. A quantidade disponível em x_1 é 5, em x_2 é 10 e em x_3 é 5. Os vértices y_1 , y_2 e y_3 são consumidores. A quantidade de produto necessária em y_1 é 5, em y_2 é 10 e em y_3 é 5. Descubra se é possível atender a estas condições (Dica: Uma forma de resolver este tipo de problema é introduzir uma fonte auxiliar s e um sumidouro t ; conectar s a x_i por uma aresta de capacidade igual à quantidade disponível em x_i ; conectar cada y_i a t através de uma aresta de capacidade igual à demanda de y_i ; encontrar um fluxo máximo na rede resultante e observar se todas as demandas são atendidas.

Produtor x Consumidor

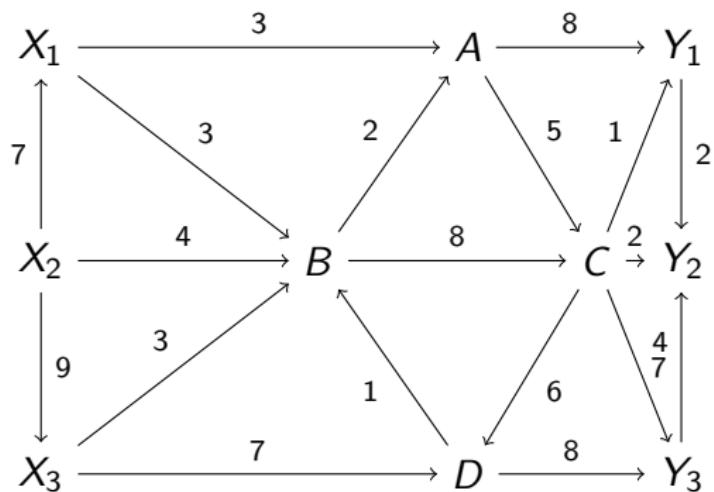


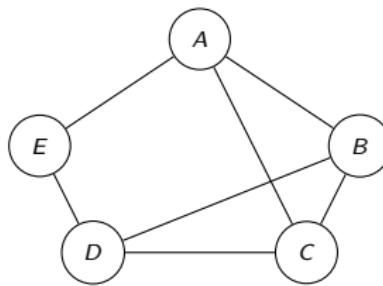
Figura: Problema produtor x consumidor

3) Dados os bom resultados obtidos no final do ano passado, a empresa de turismo Vemque Talimpo resolveu distribuir um lote de passagens (presente das companhias aéreas) entre seus funcionários. Com as passagens destinam-se a lugares específicos, o diretor da empresa viu-se diante de um dilema, pois não gostaria de presentear um funcionário com uma passagem para um lugar que não fosse de seu agrado. Sendo assim, ele resolveu fazer uma consulta aos funcionários solicitando que eles indicassem locais de sua preferência, dentre aqueles para os quais havia passagens. Como ele poderia, agora, obter uma distribuição de passagens que agradasse ao maior número possível de funcionários todos os funcionários? Para ilustrar sua solução, crie um exemplo com 5 passagens e 5 funcionários, cada um escolhendo 2 lugares.

prox

Associação Máxima

4) Mostre como seria utilizado o método de alteração estrutural para obtenção do número cromático no grafo a seguir, mostrando todos os grafos gerados em cada passo da aplicação do método:



5) Considere o grafo K_6 de vértices a,b,c,d,e,f. Mostre os grafos gerados na utilização do método da alteração estrutural para obtenção do número cromático se forem retiradas as arestas (b,c) e (a,d). Qual o número cromático obtido? E se, a partir do grafo original, forem retiradas as arestas (b,c) e (b,d)?

9) Considere que você é o Chefe do Departamento de Informática e você deve fazer a alocação das disciplinas aos professores para o próximo semestre. Você dispõe, para cada professor, da lista de disciplinas que ele é capaz de ministrar e do número máximo de disciplinas que o professor pode ministrar. Além disso você dispõe da lista de disciplinas que deverá ser oferecida no próximo semestre, e o número de turmas que deve ser oferecido para cada disciplina. Descreva como você pode resolver o problema, ou seja, chegar a uma atribuição de disciplinas aos professores que respeite suas competências e sua carga horária máxima de forma todas as turmas de todas as disciplinas sejam atendidas. Se for necessário, invente uma situação hipotética para ilustrar sua resposta.

Associação Máxima

10) Suponha que existem três geradores de energia a serem construídos em três das sete cidades mais populosas de certo país. As distâncias entre cada par de cidades é dada na tabela abaixo. O objetivo é construir os geradores tal que cada cidade está próxima dentro de 50 milhas da cada um dos geradores. Como este problema pode ser resolvido? Qual é a solução?

	B	C	D	E	F	G
A	80	110	15	60	100	80
B		40	45	55	70	90
C			65	20	50	80
D				35	55	80
E					25	60
F						70

11) Em uma nação emergente, cada uma das seis principais cidades receberá uma estação de televisão. Diferentes estações podem usar a mesma faixa de frequência, contanto que estejam distanciadas de mais de 80 quilômetros, para evitar problemas de interferência. Suponha que as cidades tenham os seguintes nomes: A, B, C, D, E e F, e as distâncias entre elas estão descritas na tabela a seguir:

	B	C	D	E	F
A	150	135	96	76	84
B		96	147	90	102
C			72	76	51
D				69	48
E					24

Qual o número mínimo de faixas de frequência necessárias, e como pode ser feita a distribuição entre as cidades? Como isso pode ser resolvido através de Teoria de Grafos?

PERT1) Construa um diagrama PERT da tabela de tarefas a seguir. Calcule também o tempo mínimo para completá-lo, os nós do caminho crítico e, para cada evento, a data mais cedo, a data mais tarde e a folga.

Tarefa	Predecessoras	Duração
a	Nenhum	10
b	Nenhum	5
c	a	10
d	a,b	5
e	c,d	15
f	d	10
g	e,f	12
h	d	10
i	g,h	7

PERT2) Construa um diagrama PERT da tabela de tarefas a seguir. Calcule também o tempo mínimo para completá-lo, os nós do caminho crítico e, para cada evento, a data mais cedo, a data mais tarde e a folga.

Tarefa	Predecessoras	Duração
a	e	3
b	c,d	5
c	a	2
d	a	6
e	nenhum	2
f	a,g	4
g	e	4
h	b,f	1

PERT3) Construa um diagrama PERT da tabela de tarefas a seguir. Calcule também o tempo mínimo para completá-lo, os nós do caminho crítico e, para cada evento, a data mais cedo, a data mais tarde e a folga.

Tarefa	Descrição da Tarefa	Pred.	Tempo (dias)
b	escavar e concretar sapatas	-	4
c	concretar a fundação	b	2
d	levantar estrutura de madeira, inclusive a base do telhado	c	4
e	assentar a alvenaria	d	6
f	instalar esgoto e tubulação do subsolo	c	1
g	concretar o piso do subsolo	f	2
h	instalar a canalização principal	f	3
i	instalar a rede elétrica geral	d	2
j	instalar aquecimento e ventilação	d,g	4
k	pregar as tábuas de revestimento e emboçar	i,j,h	10
l	assentar piso	k	3
m	instalar equipamento da cozinha	l	1
n	instalar a tubulação adicional	l	2
o	terminar carpintaria	l	3
p	terminar o telhado	e	2
q	instalar calhas e canais	p	1
r	colocar calhas para água da chuva	c	1
s	raspar e encerar o piso	o,t	2
t	pintar	m,n	3
u	terminar a instalação elétrica	t	1
v	terminar nivelamento do terreno	q,r	2
w	concretar as calçadas e completar o ajardinamento	v	5

PERT

PERT4) Construa um diagrama PERT da tabela de tarefas a seguir. Calcule também o tempo mínimo para completá-lo, as ATIVIDADES do caminho crítico e, para cada evento, a data mais cedo, a data mais tarde e a folga.

Tarefa	Predecessoras	Duração
a	Nenhum	10
b	Nenhum	5
c	a	5
d	a,b	10
e	b	5
f	c,d	10
g	d	7
h	d,e	16

PERT5) Construa um diagrama PERT da tabela de tarefas na página a seguir. Calcule também o tempo mínimo para completá-lo, os nós do caminho crítico e, para cada evento, a data mais cedo, a data mais tarde e a folga.

Tarefa	Predecessoras	Duração
a	Nenhum	10
b	Nenhum	20
c	Nenhum	30
d	Nenhum	40
e	a,b	15
f	b,c	10
g	c,d	12

PERT6) Suponhamos que se deseja levar a cabo um projeto de construção de locais comerciais para alugar, e para isto foram identificadas as seguintes atividades.

Descrição	Predecessoras	Dur.(semanas)
1 - Preparar desenho arquitetônico	Nenhuma	5
2 - Identificar novos arrendatários potenciais	Nenhuma	6
3 - Desenvolver prospectos de contrato para os arrendatários	1	4
4 - Selecionar construtor	1	3
5 - Preparar as licenças de construção	1	1
6 - Obter a aprovação das licenças de construção	5	4
7 - Levar a cabo a construção	4,6	14
8 - Formalizar os contratos com os arrendatários	2,3	12
9 - Entrada do arrendatários	7,8	2

Desenhe o diagrama PERT, e calcule a data mais cedo e a data mais tarde para cada evento e identifique as atividades que fazem parte do caminho crítico.

PERT7) Suponha que uma empreiteira ganhou uma concorrência para construir uma planta industrial e para isto foram identificadas as seguintes atividades.

Atividade	Descrição	Predecessoras	Dur.(semanas)
A	Escavação	Nenhuma	2
B	Fundação	A	4
C	Paredes	B	10
D	Telhado	C	6
E	Encanamento Exterior	C	4
F	Encanamento Interior	E	5
G	Muros	D	7
H	Pintura Exterior	E,G	9
I	Instalação Elétrica	C	7
J	Divisórias	F,I	8
K	Piso	J	4
L	Pintura Interior	J	5
M	Acabamento Exterior	H	2
N	Acabamento Interior	K,L	6

Desenhe o diagrama PERT, e calcule a data mais cedo e a data mais tarde para cada evento e identifique as atividades que fazem parte do caminho crítico.

3) Mostre todos os passos da utilização do algoritmo de
programação dinâmica para encontrar o ciclo hamiltoniano de custo
mínimo (problema do caixeiro viajante) para o grafo a seguir:

	1	2	3	4
1	0	3	∞	5
2	6	0	4	7
3	4	∞	0	2
4	2	8	3	0

15) Mostre todos os passos da utilização do algoritmo de programação dinâmica para encontrar o ciclo hamiltoniano de custo mínimo (problema do caixeiro viajante) para o grafo a seguir:

	1	2	3	4
1	0	6	2	5
2	6	0	4	∞
3	2	4	0	1
4	5	2	1	0

16) Problema das 5 damas - A dama é a peça mais poderosa do xadrez. Numa jogada ela pode mover-se tantas casas quantas quiser em qualquer direção (horizontal, vertical ou diagonal), desde que não haja nenhuma outra peça que obstrua sua passagem. O desenho que se segue mostra uma posição particular da dama na qual ela tem 27 possibilidades de movimento. Essas 27 casas (além daquela onde a dama está) estão sob o domínio da dama. Qualquer outra peça que estivesse numa destas casas estaria sob ataque da dama em questão. O problema consiste em procurar as posições a serem ocupadas por damas em um tabuleiro de xadrez de modo a que se possa controlar todas as casas do tabuleiro com o menor número de damas. Como esse problema pode ser modelado e resolvido com teoria de grafos?

			o				o
o			o			o	
	o		o		o		
		o	o	o			
o	o	o	D	o	o	o	o
		o	o	o			
	o		o		o		
o			o			o	

17) Problema da Distribuição de Pontos de Observação - Em um parque ecológico, deseja-se colocar pontos de observação de modo que se tenha uma visão de todo o parque. Para isso o parque foi dividido em várias áreas, sendo que para cada área foram identificados os pontos que mais propiciam sua observação. Conhecida esta relação de vigilância, observou-se que em vários casos um mesmo ponto de observação permite a vigilância de mais de uma área. Sendo assim, a direção do parque optou por procurar instalar o menor número de pontos de observação de forma a interferir o mínimo possível com a vida no parque. Como se poderia identificar esse conjunto mínimo de pontos de observação?

Questões do POSCOMP

(39 - Fundamentos - 2002) - O menor número possível de arestas em um grafo conexo com n vértices é:

1 1

2 $n/2$

3 $n - 1$

4 n

5 n^2

[Voltar para o índice](#)

Questões do POSCOMP

(39 - Fundamentos - 2002) - O menor número possível de arestas em um grafo conexo com n vértices é:

1 1

2 $n/2$

3 $n - 1$

4 n

5 n^2

(40 - Fundamentos - 2002) - Considere um grafo G satisfazendo as seguintes propriedades:

- G é conexo
- Se removermos qualquer aresta de G , o grafo obtido é desconexo.

Então é correto afirmar que o grafo G é:

- 1 Um circuito
- 2 Não bipartido
- 3 Uma árvore
- 4 Hamiltoniano
- 5 Euleriano

(40 - Fundamentos - 2002) - Considere um grafo G satisfazendo as seguintes propriedades:

- G é conexo
- Se removermos qualquer aresta de G , o grafo obtido é desconexo.

Então é correto afirmar que o grafo G é:

- 1 Um circuito
- 2 Não bipartido
- 3 Uma árvore
- 4 Hamiltoniano
- 5 Euleriano

(33 - Fundamentos - 2004) - Considere as seguintes afirmações sobre um grafo G com $n > 0$ vértices:

- I. Se G é conexo o número de arestas é maior que n ;
- II. G será acíclico somente se o número de arestas for menor que n ;
- III. Se G não tem triângulos então G é planar;
- IV. G é Euleriano se, e somente se, todo grau é par.

As afirmativas verdadeiras são:

- 1 I e II
- 2 I e III
- 3 II e III
- 4 II e IV
- 5 II, III e IV

(33 - Fundamentos - 2004) - Considere as seguintes afirmações sobre um grafo G com $n > 0$ vértices:

- I. Se G é conexo o número de arestas é maior que n ;
- II. G será acíclico somente se o número de arestas for menor que n ;
- III. Se G não tem triângulos então G é planar;
- IV. G é Euleriano se, e somente se, todo grau é par.

As afirmativas verdadeiras são:

- 1 I e II
- 2 I e III
- 3 II e III
- 4 II e IV
- 5 II, III e IV

(39 - Fundamentos - 2005) - Os grafos $G = (V_G, E_G)$ e $H = (V_H, E_H)$ são isomorfos. Assinale a alternativa que justifica essa afirmação.

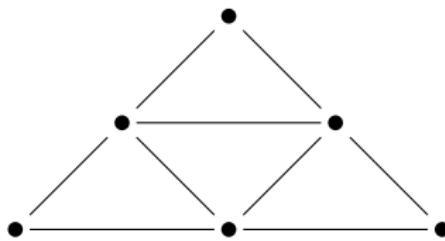
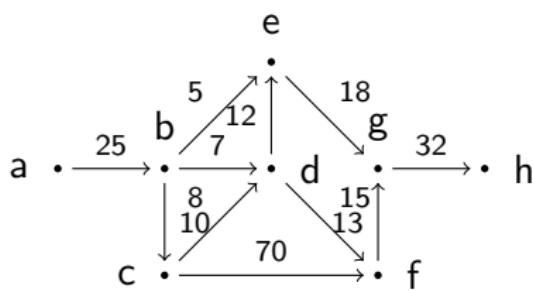


Figura: H

- 1** As seqüências dos graus dos vértices de G e H são iguais.
- 2** Os grafos têm o mesmo número de vértices e o mesmo número de arestas.
- 3** Existe uma bijeção de V_G em V_H que preserva as adjacências.
- 4** Cada vértice de G e de H pertence a exatamente quatro triângulos distintos.
- 5** Ambos os grafos admitem um circuito que passa por cada aresta exatamente uma vez.

(40 - Fundamentos - 2005) - Dadas as seguintes afirmações

- I. Qualquer grafo conexo com n vértices deve ter pelo menos $n - 1$ arestas.
- II. O grafo bipartido completo $K_{m,n}$ é Euleriano desde que m e n sejam ímpares.
- III. Em um grafo o número de vértices de grau ímpar é sempre par.

São verdadeiras:

- 1 Somente a afirmação (I).
- 2 Somente as afirmações (I) e (III).
- 3 Somente as afirmações (II) e (III).
- 4 Somente as afirmações (I) e (II).
- 5 Todas as afirmações.

(26 - Fundamentos - 2006) - A respeito da representação de um grafo de n vértices e m arestas é correto dizer que:

- 1 a representação sob a forma de matriz de adjacência exige espaço $\Omega(m^2)$.
- 2 a representação sob a forma de listas de adjacência permite verificar a existência de uma aresta ligando dois vértices dados em tempo $O(1)$.
- 3 a representação sob a forma de matriz de adjacência não permite verificar a existência de uma aresta ligando dois vértices dados em tempo $O(1)$.
- 4 a representação sob a forma de listas de adjacência exige espaço $\Omega(n + m)$.
- 5 todas as alternativas estão corretas.

(36 - Fundamentos - 2006) - Seja $G = (V, E)$ um grafo simples conexo não-euleriano. Queremos construir um grafo H que seja euleriano e que contenha G como subgrafo. Considere os seguintes possíveis processos de construção:

- I. Acrescenta-se um novo vértice, ligando-o a cada vértice de G por uma aresta.
- II. Acrescenta-se um novo vértice, ligando-o a cada vértice de grau ímpar de G por uma aresta.
- III. Cria-se uma nova cópia G' do grafo G e acrescenta-se uma aresta ligando cada par de vértices correspondentes.
- IV. Escolhe-se um vértice arbitrário de G e acrescentam-se arestas ligando este vértice a todo vértice de grau ímpar de G .
- V. Duplicam-se todas as arestas de G .

Quais dos processos acima sempre constroem corretamente o grafo H ?

- 1** Somente (II) e (IV)
- 2** Somente (II), (IV) e (V)
- 3** Somente (III), (V) e (VI)
- 4** Somente (II), (IV), (V) e (VI)
- 5** Somente (I), (III), (IV) e (V)

(31 - Fundamentos - 2007) - Considere o *problema do caixeiro viajante*, definido como se segue.

Sejam S um conjunto de n cidades, $n \geq 0$, e $d_{ij} > 0$ a distância entre as cidades i e j , $i, j \in S$, $i \neq j$. Define-se um *percurso fechado* como sendo um percurso que parte de uma cidade $i \in S$, passa exatamente uma vez por cada cidade de $S - \{i\}$, e retorna à cidade de origem. A distância de um percurso fechado é definida como sendo a soma das distâncias entre cidades consecutivas do percurso. Deseja-se encontrar um percurso fechado de distância mínima. Suponha um algoritmo guloso que, partindo da cidade 1, move-se para a cidade mais próxima ainda não visitada e que repita esse processo até passar por todas as cidades, retornando à cidade 1.

Considere as seguintes afirmativas.

- I Todo percurso fechado obtido com esse algoritmo tem distância mínima.
- II O problema do caixeiro viajante pode ser resolvido com um algoritmo de complexidade linear no número de cidades.
- III Dado que todo percurso fechado corresponde a uma permutação das cidades, existe um algoritmo de complexidade exponencial no número de cidades para o problema do caixeiro viajante.

Em relação a essas afirmativas, pode-se afirmar que

- (a) I é falsa e III é correta.
- (b) I, II e III são corretas.
- (c) apenas I e II são corretas.
- (d) apenas I e III são falsas.
- (e) I, II e III são falsas.

(39 - Fundamentos - 2007) - Seja $G = (V, E)$ um grafo simples e finito, onde $|V| = n$ e $|E| = m$.

Nesse caso, analise as seguintes afirmativas.

- I. Se G é hamiltoniano, então G é 2-conexo em vértices.
- II. Se G é completo, então G é hamiltoniano.
- III. Se G é 4-regular e conexo, então G é euleriano.
- IV. Se G é bipartite com partições A e B , então G é hamiltoniano se, e somente se, $|A| = |B|$.
- V. Se G é euleriano, então G é 2-conexo.

A análise permite concluir que são **FALSOS**

- 1 apenas os itens I e II.
- 2 apenas os itens I e V.
- 3 apenas os itens II e III.
- 4 apenas os itens III e IV.
- 5 apenas os itens IV e V.

40 - Fundamentos - 2007

Tem figurinhas. Mostrar no arquivo da prova.

(25 - Fundamentos - 2008) - Um grafo $G(V, E)$ é uma árvore se G é conexo e acíclico.

Assinale a **definição** que **NÃO** pode ser usada para definir árvores.

- 1 G é conexo e o número de arestas é mínimo.
- 2 G é conexo e o número de vértices excede o número de arestas por uma unidade.
- 3 G é acíclico e o número de vértices excede o número de arestas por uma unidade.
- 4 G é acíclico e, para todo par de vértices v, w , que não são adjacentes em G , a adição da aresta (v, w) produz um grafo contendo exatamente um ciclo.
- 5 G é acíclico, e o número de arestas é mínimo.

(26 - Fundamentos - 2008) - Em um grafo $G(V, E)$, o grau de um vértice v é o número de vértices adjacentes a v .

A esse respeito, assinale a afirmativa **CORRETA**.

- 1 Num grafo, o número de vértices com grau ímpar é sempre par.
- 2 Num grafo, o número de vértices com grau par é sempre ímpar.
- 3 Num grafo, sempre existe algum vértice com grau par.
- 4 Num grafo, sempre existe algum vértice com grau ímpar.
- 5 Num grafo, o número de vértices com grau ímpar é sempre igual ao número de vértices com grau par.

(28 - Fundamentos - 2008) - Seja $G(V, E)$ um grafo tal que $|V| = n$ e $|E| = m$.

Analise as seguintes sentenças:

- I. Se G é acíclico com no máximo $n - 1$ arestas, então G é uma árvore.
- II. Se G é um ciclo, então G tem n árvores geradoras distintas.
- III. Se G é conexo com no máximo $n - 1$ arestas, então G é uma árvore.
- IV. Se G é conexo e tem um ciclo, então para toda árvore geradora T de G , $E(G) - E(T) \neq \emptyset$

(24 - Fundamentos - 2009) - Assinalar a afirmativa correta, em relação a um grafo completo G com $n > 2$ vértices.

- 1 O grau de cada vértice é n .
- 2 O número cromático de G é igual a $n - 1$.
- 3 G não pode ser um grafo bipartido.
- 4 G não possui caminho hamiltoniano.
- 5 G possui ciclo euleriano.

(30 - Fundamentos - 2009) - Considere o algoritmo de busca em largura em grafos. Dado o grafo $V = A, B, C, D, E, F$ e $E = (A, C), (A, B), (B, D), (C, E), (C, D), (D, E), (D, F), (E, F)$ e o vértice A como ponto de partida, a ordem em que os vértices são descobertos é dada por:

- 1** $A \ B \ C \ D \ E \ F$
- 2** $A \ B \ D \ C \ E \ F$
- 3** $A \ C \ D \ B \ F \ E$
- 4** $A \ B \ C \ E \ D \ F$
- 5** $A \ B \ D \ F \ E \ C$

(43 - Fundamentos - 2010) Dados dois grafos não orientados $G_1(V_1, E_1)$ e $G_2(V_2, E_2)$:

- $G_1 : V_1 = \{a, b, c\} \quad E_1 = \{(a, b), (b, c), (a, c)\}$
- $G_2 : V_2 = \{d, e\} \quad E_2 = \{(d, e)\}$

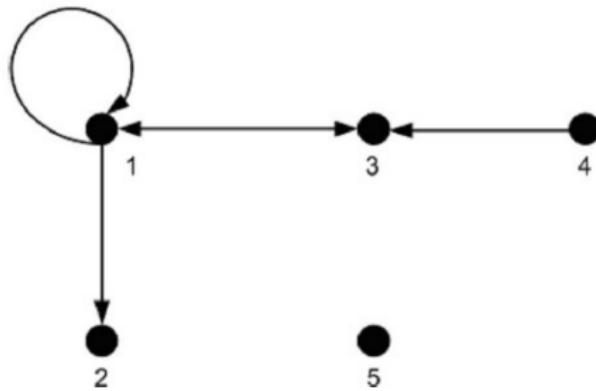
Qual alternativa apresenta corretamente o grafo $\text{Gr}(V, E)$ resultante da soma dos grafos G_1 e G_2 ?

- a) Gr: $V = \{a, b, c, d, e\} \quad E = \{(a, b), (b, c), (a, c), (d, e)\}$
- b) Gr: $V = \{a, b, c, d, e\} \quad E = \{(a, d), (a, e), (b, d), (b, e), (c, d), (c, e), (d, e)\}$
- c) Gr: $V = \{a, b, c, d, e\} \quad E = \{(a, b), (b, c), (a, c), (a, d), (a, e), (b, d), (b, e), (c, d), (c, e)\}$
- d) Gr: $V = \{a, b, c, d, e\} \quad E = \{(a, b), (b, c), (a, c), (a, d), (a, e), (b, d), (b, e), (c, d), (c, e), (d, e)\}$
- e) Gr: $V = \{a, b, c, d, e\} \quad E = \{(a, b), (b, c), (c, d), (d, e), (e, a)\}$

(2010 - Fundamentos - 46) Qual é o número cromático do grafo $K_{3,2}$?

- a) 2
- b) 3
- c) 4
- d) 5
- e) 6

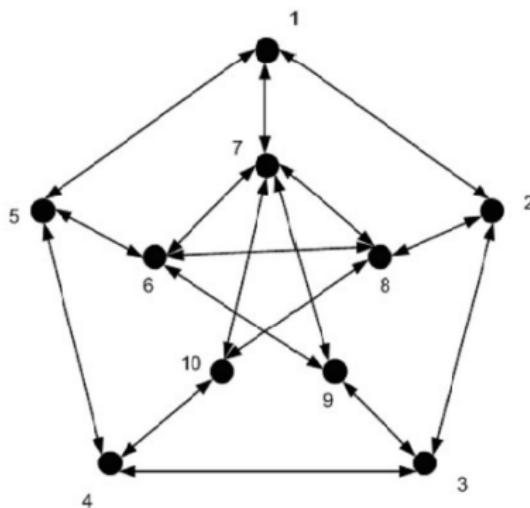
(2011 - Matemática - 13) Considere o grafo a seguir.



O grafo representa a relação:

- a) $R = \{(1, 1), (1, 2), (1, 3), (3, 1), (4, 3)\}$
- b) $R = \{(1, 1), (1, 2), (1, 3), (3, 1), (3, 4)\}$
- c) $R = \{(1, 1), (1, 3), (2, 1), (3, 1), (3, 4)\}$
- d) $R = \{(1, 1), (1, 2), (1, 3), (3, 4), (4, 3)\}$
- e) $R = \{(1, 1), (1, 3), (2, 1), (3, 1), (4, 3)\}$

(2011 - Matemática - 18) Sejam 10 cidades conectadas por rodovias, conforme o grafo a seguir.



Um vendedor sai de uma das cidades com o intuito de visitar cada uma das outras cidades uma única vez e retornar ao seu ponto de partida. Com base no grafo e nessa informação, considere as afirmativas a seguir.

- I. O vendedor cumprirá seu propósito com êxito se sair de uma cidade par.
- II. O vendedor cumprirá seu propósito com êxito se sair de uma cidade ímpar.
- III. O vendedor não cumprirá seu propósito com êxito se sair de uma cidade par.
- IV. O vendedor não cumprirá seu propósito com êxito se sair de uma cidade ímpar.

Assinale a alternativa correta.

- a) Somente as afirmativas I e II são corretas.
- b) Somente as afirmativas I e IV são corretas.
- c) Somente as afirmativas III e IV são corretas.
- d) Somente as afirmativas I, II e III são corretas.
- e) Somente as afirmativas II, III e IV são corretas.

Um vendedor sai de uma das cidades com o intuito de visitar cada uma das outras cidades uma única vez e retornar ao seu ponto de partida. Com base no grafo e nessa informação, considere as afirmativas a seguir.

- I. O vendedor cumprirá seu propósito com êxito se sair de uma cidade par.
- II. O vendedor cumprirá seu propósito com êxito se sair de uma cidade ímpar.
- III. O vendedor não cumprirá seu propósito com êxito se sair de uma cidade par.
- IV. O vendedor não cumprirá seu propósito com êxito se sair de uma cidade ímpar.

Assinale a alternativa correta.

- a) Somente as afirmativas I e II são corretas.
- b) Somente as afirmativas I e IV são corretas.
- c) **Somente as afirmativas III e IV são corretas.**
- d) Somente as afirmativas I, II e III são corretas.
- e) Somente as afirmativas II, III e IV são corretas.

(2011 - Fundamentos - 44) Qual a quantidade mínima de arestas que se deve remover do grafo completo com 6 vértices, K_6 , para se obter um grafo planar?

- a) 1
- b) 2
- c) 3
- d) 4
- e) 5

(2011 - Fundamentos - 47) Seja G um grafo conexo. Considere a notação a seguir.

- * c_v é o número cromático em vértices de G .
- * c_e é o número cromático em arestas de G .
- * g_{min} é o grau mínimo de G .
- * g_{max} é o grau máximo de G .
- * w é a quantidade de vértices do maior subgrafo completo de G .

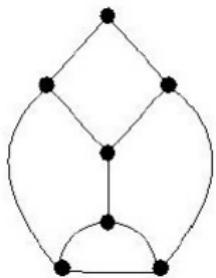
Assinale a alternativa correta.

- a) $c_v \leq c_e$
- b) $c_v \leq w$
- c) $c_e \leq g_{max}$
- d) $c_v \leq g_{max} + 1$
- e) $c_v \geq g_{min}$

(2011 - Fundamentos - 50) Seja G um grafo conexo com n vértices. Considere duas rotulações dos vértices de G obtidas por duas buscas em G , uma em largura, $I()$, e outra em profundidade, $p()$, ambas iniciadas no vértice v . Em cada rotulação, os vértices receberam um número de 1 a n , o qual representa a ordem em que foram alcançados na busca em questão. Assim, $I(v) = p(v) = 1$; enquanto $I(x) > 1$ e $p(x) > 1$ para todo vértice x diferente de v . Considere dois vértices u e w de G e denote por $d(u, w)$ a distância em G de u até w . Com base nesses dados, assinale a alternativa correta.

- a) Se $I(u) < I(w)$ e $p(u) < p(w)$, então $d(v, u) < d(v, w)$.
- b) Se $I(u) < I(w)$ e $p(u) > p(w)$, então $d(v, u) = d(v, w)$.
- c) Se $I(u) > I(w)$ e $p(u) < p(w)$, então $d(v, u) \leq d(v, w)$.
- d) Se $I(u) > I(w)$ e $p(u) > p(w)$, então $d(v, u) < d(v, w)$.
- e) Se $I(u) < I(w)$ e $p(u) > p(w)$, então $d(v, u) \leq d(v, w)$.

(2013) Qual o número cromático do grafo a seguir?



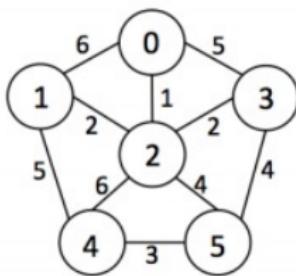
(2014) Considerando que um grafo possui n vértices e m arestas, assinale a alternativa que apresenta, corretamente, um grafo planar.

- a) $n = 5, m = 10$
- b) $n = 6, m = 15$
- c) $n = 7, m = 21$
- d) $n = 8, m = 12$
- e) $n = 9, m = 22$

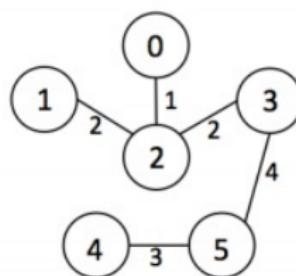
(2016 - Fundamentos - 32) A matriz de um grafo $G = (V, A)$ contendo n vértices é uma matriz $n \times n$ de bits, em que $A[i,j]$ é 1 (ou verdadeiro, no caso de booleanos) se e somente se existir um arco do vértice i para o vértice j . Essa definição é uma:

- (A) Matriz de adjacência para grafos não ponderados.
- (B) Matriz de recorrência para grafos não ponderados.
- (C) Matriz de incidência para grafos não ponderados.
- (D) Matriz de adjacência para grafos ponderados.
- (E) Matriz de incidência para grafos ponderados.

(2016 - Fundamentos - 36) A Figura (a) abaixo mostra o exemplo de um grafo não direcionado G com os pesos mostrados ao lado de cada aresta. Sobre a árvore T representada na Figura (b), é correto afirmar que:



(a)



(b)

- (A) T representa a árvore geradora mínima do grafo da Figura (a) cujo peso total é 12. T não é única, pois a substituição da aresta (3,5) pela aresta (2,5) produz outra árvore geradora de custo 12.
- (B) T representa a árvore de caminhos mais curtos entre todos os pares de vértices do grafo da Figura (a). T não é única, pois a substituição da aresta (3,5) pela aresta (2,5) produz caminhos mais curtos entre os mesmos pares de vértices do grafo.
- (C) T representa a árvore geradora mínima do grafo da Figura (a) cujo peso total é 12. A substituição da aresta (3,5) pela aresta (2,4) produz uma árvore geradora máxima cujo peso total é 14.
- (D) T representa a ordenação topológica do grafo da Figura (a). O peso da aresta (0,2) indica que ela deve ser executada antes da aresta (2,3) e o peso da aresta (2,3) indica que ela deve ser executada antes da aresta (4,5) e assim sucessivamente.
- (E) T representa a árvore de caminhos mais curtos do grafo da Figura (a) com origem única no vértice 2. T não é única, pois a substituição da aresta (3,5) pela aresta (2,4) produz caminhos mais curtos entre todos os pares de vértices do grafo.

(2016 - Fundamentos - 37) Em relação a Teoria dos Grafos, relate a Coluna 1 à Coluna 2.

Coluna 1

1. Grafo Completo.
2. Hipergrafo.
3. Árvore Livre.
4. Grafo Planar.
5. Grafo não direcionado antirregular.

Coluna 2

- Grafo não direcionado, no qual todos os pares de vértices são adjacentes entre si.
- Grafo não direcionado em que cada aresta conecta um número arbitrário de vértices, ao invés de conectar dois vértices apenas.
- Grafo não direcionado acíclico e dirigido.
- Grafo em que seu esquema pode ser traçado em um plano, de modo que duas arestas quaisquer se toquem, no máximo, em alguma extremidade.
- Grafo que possui o maior número possível de graus diferentes em sua sequência.

A ordem correta de preenchimento dos parênteses, de cima para baixo, é:

- (A) 1 - 2 - 3 - 4 - 5.
- (B) 2 - 3 - 4 - 5 - 1.
- (C) 3 - 4 - 5 - 1 - 2.
- (D) 4 - 5 - 1 - 2 - 3.
- (E) 5 - 1 - 2 - 3 - 4.

Questões do ENADE

- (Ciência da Computação 2014 - Questão 22) Considere o processo de fabricação de um produto siderúrgico que necessita passar por n tratamentos térmicos e químicos para ficar pronto. Cada uma das n etapas de tratamento é realizada uma única vez na mesma caldeira. Além do custo próprio de cada etapa do tratamento, existe o custo de se passar de uma etapa para outra, uma vez que, dependendo da sequência escolhida, pode ser necessário alterar a temperatura da caldeira e limpá-la para evitar a reação entre os produtos químicos utilizados. Assuma que o processo de fabricação inicia e termina com a caldeira limpa. Deseja-se projetar um algoritmo para indicar a sequência de tratamentos que possibilite fabricar o produto com o menor custo total. Nessa situação, conclui-se que:

- A A solução do problema é obtida em tempo de ordem $O(n \log n)$, utilizando-se um algoritmo ótimo de ordenação.
- B Uma heurística para a solução do problema de coloração de grafos solucionará o problema em tempo polinomial.
- C O problema se reduz a encontrar a árvore geradora mínima para o conjunto de etapas do processo, requerendo tempo de ordem polinomial para ser solucionado.
- D A utilização do algoritmo de Dijkstra para se determinar o caminho de custo mínimo entre o estado inicial e o final soluciona o problema em tempo polinomial.
- E Qualquer algoritmo conhecido para a solução do problema descrito possui ordem de complexidade de tempo não-polinomial, uma vez que o problema do caixeiro viajante se reduz a ele.

(Ciência da Computação 2014 - Questão 30) Uma fazenda possui um único poço artesiano que deve abastecer n bebedouros para o gado. Deseja-se determinar um projeto de ligação entre esses $n+1$ pontos através de encanamentos com a menor extensão total. Um algoritmo proposto para a solução do problema executa os seguintes passos:

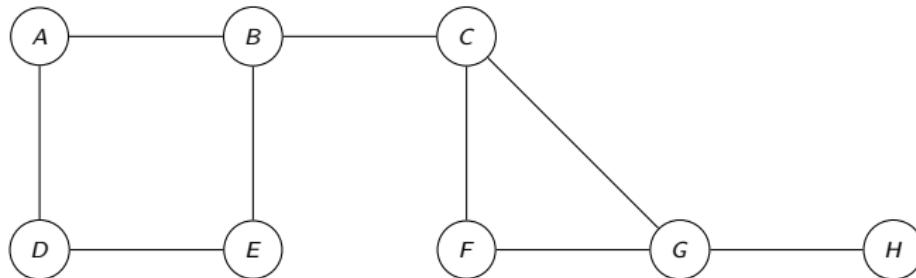
- 1 Crie $n+1$ conjuntos unitários, cada um contendo um dos pontos a serem ligados entre si e insira esses conjuntos em um conjunto C .
- 2 Crie um conjunto D contendo um registro para cada combinação possível de dois pontos distintos a serem ligados. Cada registro deve conter os campos c_i , c_j e d , em que c_i e c_j são os dois pontos a serem ligados e d é a distância entre eles.
- 3 Enquanto D não estiver vazio faça:
 - 1) Remova o registro de D com o menor valor de distância d .
 - 2) Se os valores de c_f e c_l do registro removido pertencerem a conjuntos distintos de C , então
 - a) Substitua estes dois conjuntos pela união entre eles.
 - b) Guarde o registro removido em um conjunto-solução.

Com base na descrição do problema e do algoritmo proposto, conclui-se que

- A O problema exemplifica a obtenção de uma árvore geradora mínima, portanto está no conjunto P.
- B O algoritmo é uma heurística para o Problema do Caixeiro Viajante, logo apresenta complexidade polinomial.
- C O problema descrito é de otimização, logo pertence ao conjunto NP-difícil, mas não ao conjunto NP-completo.
- D Uma alternativa para a solução do problema é usar o algoritmo de Dijkstra para obtenção do caminho mínimo entre dois pontos.
- E O passo de maior custo do algoritmo é a criação do conjunto D com as combinações de pontos, apresentando complexidade computacional $\Theta(n!)$.

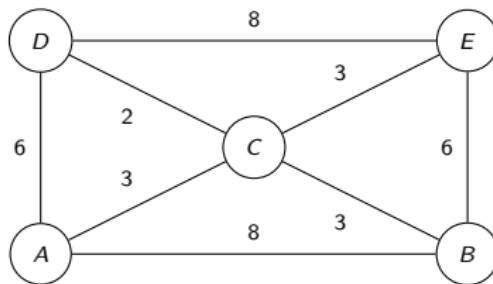
Últimas Provas - Primeira Área - 14/10/09

- 1) (2 pontos) Considere a sequência de números binários de 0 a 7. É possível colocar os 8 números em uma sequência de modo que, de um número para o outro, mude apenas um bit? Como isso pode ser resolvido através de teoria de grafos? Se necessário, desenhe o grafo associado ao problema.
- 2) (2 pontos) Decomponha o grafo a seguir em seus componentes biconexos, a partir do vértice e, utilizando o algoritmo visto em aula. Mostre todos os passos do algoritmo (árvore em profundidade mostrando as arestas de retorno, identificação das articulações, identificação dos demarcadores e dos componentes a cada passo).



[Voltar para o índice](#)

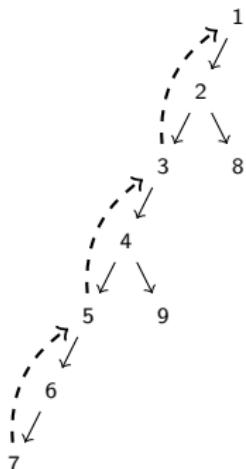
- 3) (2 pontos) No grafo do exercício anterior, aplique o algoritmo de Tremaux, a partir do vértice A , mostrando como ficam ao final as marcações das passagens. Use como critério de escolha entre dois vértices, sempre o de menor ordem alfabética.
- 4) (2 pontos) Aplique o algoritmo de Dijkstra ao grafo a seguir, para calcular a distância entre os vértices A e E . Mostre o vetor com os λ a cada vez que passar pelo passo 3.



- 5)(1 ponto) Um grafo conexo é dito euleriano se possui um ciclo euleriano. E semi-euleriano se possui um caminho euleriano mas não possui um ciclo euleriano. Determine os valores de n para os quais K_n é euleriano e semi-euleriano.
- 6)(1 ponto) É possível que um grafo bipartido com um número ímpar de vértices seja hamiltoniano (isso é, contenha um ciclo hamiltoniano)? E euleriano? Justifique sua resposta.

Primeira Prova - 6/10/10

- 1) (2.0 pontos) A figura a seguir representa a árvore resultante da busca em profundidade e as arestas de retorno de um grafo G . Mostre o lowpt de cada vértice. Identifique as articulações, demarcadores e componentes biconexas de G .



2) (2.0 pontos) Para qualquer inteiro positivo k , um cubo de dimensão k (ou k -cubo) é o grafo definido da seguinte maneira: os vértices do grafo são todas as seqüências $b_1 b_2 \dots b_k$ de bits; dois vértices são adjacentes se e somente se diferem em exatamente uma posição. Por exemplo, os vértices do cubo de dimensão 3 são 000, 001, 010, 011, 100, 101, 110, 111; o vértice 000 é adjacente aos vértices 001, 010, 100 e a nenhum outro; e assim por diante.

(0.4 pontos) Faça uma figura do 1-cubo, do 2-cubo e do 3-cubo.

(0.4 pontos) Mostre a matriz de adjacências de um 3-cubo.

(0.4 pontos) Quantos vértices tem o k -cubo?

(0.4 pontos) Quantas arestas tem o k -cubo?

(0.4 pontos) O 3-cubo é planar?

3) (2.0 pontos) Descreva sucintamente qual a função de cada um dos 8 algoritmos a seguir, deixando bem claras as diferenças entre eles, caso haja algoritmos com funções semelhantes (.25 cada um):

Floyd-Warshall

Prim

Kruskal

Demoucron

Tremaux

Moore

Dijkstra

Ford

4) (2.0 pontos) O Departamento de Informática pretende oferecer 7 disciplinas para o nono semestre do Curso de Computação no próximo semestre.

(C) Compiladores

(G) Grafos

(L) Lógica

(N) Análise de Algoritmos

(P) Probabilidade

(S) Engenharia de Software

(T) Arquitetura de

Computadores

Com base na pré-matrícula, verificou-se que cada estudante planeja cursar as disciplinas indicadas abaixo:

Aldo C,L,T

Batista C,G,S

Carlos C,L

Daniel G,N

Eduardo L,N

Frederico C,G

Giovana N,P

Homero G,L

Iracema C,T

Janete C,S,T

Kevin P,S

Luis P,T

Considerando que cada disciplina é dada num período, como se poderia determinar qual o menor número de períodos necessários para que os sete cursos possam ser oferecidos e todos os alunos possam cursar as disciplinas planejadas? Para os dados fornecidos, qual é esse número?

5)(2 pontos) Considere o problema 544 da ACM, descrito a seguir. Como esse problema poderia ser modelado e resolvido com teoria de grafos? Descreva em detalhes o grafo que modelaria o problema, o que se buscaria nesse grafo e qual o algoritmo utilizado. Caso seja necessário modificar o algoritmo, descreva o que deveria ser modificado no mesmo.

Problema 544 da ACM - Heavy Load (Carga Pesada)

O Problema

Big Johnsson Trucks Inc. é uma empresa especializada na fabricação de caminhões grandes. Seu último modelo, o V12 Godzilla, é tão grande que a quantidade de carga que pode transportar com ela nunca é limitado pelo próprio caminhão. É limitado apenas pelas restrições de peso que se aplicam às estradas ao longo do caminho que pretende conduzir.

Dadas a cidade de origem e de destino, seu trabalho é determinar a carga máxima do V12 Godzilla de forma que exista um caminho entre as duas cidades especificadas.

Entrada

O arquivo de entrada irá conter um ou mais casos de teste. A primeira linha de cada caso de teste contém dois números inteiros: o número N de cidades ($2 \leq n \leq 200$) e o número r e segmentos de estrada ($1 \leq r \leq 19.900$) que compõem a rede da rua.

Então r linhas seguirão, cada uma descrevendo um segmento rodoviário, contendo as duas cidades ligadas pelo segmento e o limite de peso de caminhões que usam esse segmento. Os nomes não tem mais de 30 caracteres e não contêm caracteres de espaço em branco. Os limites de peso são números inteiros no intervalo 0-10000. Estradas sempre podem ser percorrido em ambos os sentidos. A última linha do caso de teste contém os nomes da cidade de origem e da cidade de destino. A entrada será encerrado por dois valores de 0 para n e r .

Saída

Para cada caso de teste, imprimir três linhas:

- * Uma linha dizendo "Cenário x" onde x é o número do caso de teste
- * Uma linha dizendo "y toneladas" onde y é a carga máxima possível
- * Uma linha em branco

Exemplo de entrada

4 3

Karlsruhe Stuttgart 100

Stuttgart Ulm 80

Ulm Muenchen 120

Karlsruhe Muenchen

5 5

Karlsruhe Stuttgart 100

Stuttgart Ulm 80

Ulm Muenchen 120

Karlsruhe Hamburg 220

Hamburg Muenchen 170

Muenchen Karlsruhe

0 0

Exemplo de Saída

Cenário 1

80 toneladas

Cenário 2

170 toneladas

Primeira Prova - 09/05/11

1)(2.0 pontos) Considere a árvore obtida a partir de uma busca em profundidade em um grafo G , com raiz no vértice 3 e definida pelos seguintes arcos:

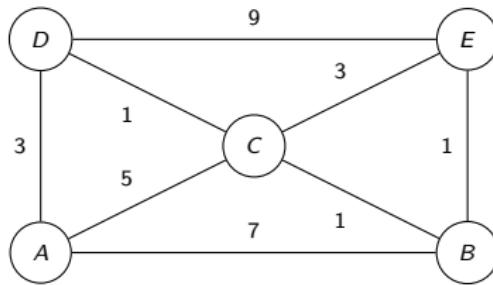
Arcos diretos: $\{(3,4),(4,5),(4,2),(5,6),(5,1),(6,9),(1,7),(1,8)\}$

Arcos de retorno: $\{(5,3),(9,5),(7,5)\}$

Mostre o lowpt de cada vértice. Identifique as articulações, demarcadores e componentes biconexas de G .

4)(1.0 ponto) Um grafo conexo é dito euleriano se possui um ciclo euleriano. Desenho 6 grafos eulerianos não isomorfos com exatamente seis vértices (há 8 grafos assim).

5)(2 pontos) Aplique o algoritmo de Dijkstra ao grafo a seguir, para calcular a distância entre os vértices A e E . Mostre o vetor com os λ a cada vez que passar pelo passo 3.

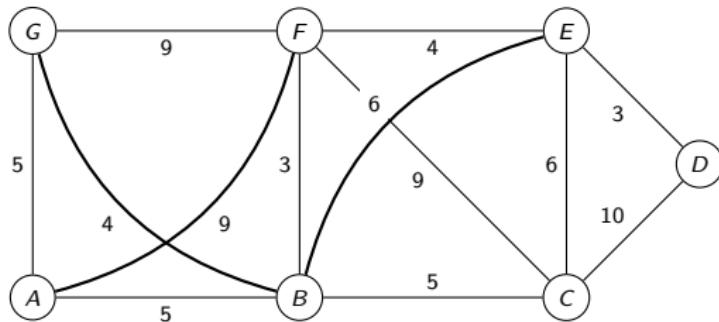


6)(1.0 ponto) Suponha um conjunto de n números inteiros positivos. Diga uma condição necessária para que esses números possam representar os graus dos n vértices de uma árvore. Essa condição é suficiente?

Primeira Prova - 3/10/11

- 1) (1.0 ponto) O grafo resultante da remoção das arestas $(3,1)$, $(3,2)$, $(6,4)$, $(6,5)$ e $(5,4)$ de K_6 é planar? Justifique sua resposta.
- 3) (1 ponto) Seja G um grafo com 14 vértices e 25 arestas. Se todo vértice de G tem grau 3 ou 5, quantos vértices de grau 3 o grafo G possui?
- 4) (1 ponto) Um grafo é dito r -regular se todos os vértices tem grau r . Dê um exemplo de um grafo 3-regular que não seja completo.
- 5) (1 ponto) Uma floresta é um grafo desconexo em que todas as componentes conexas são árvores. Determine o número de arestas de uma floresta com n vértices e k componentes conexos, em função de n e k .

8) (1 ponto) O grafo a seguir corresponde ao projeto de uma rede de computadores. Os vértices correspondem às máquinas e as arestas correspondem à possibilidade de conexão entre duas máquinas, juntamente com o correspondente custo de instalação. Essa rede deverá ser construída de forma que cada computador possa se comunicar (direta ou indiretamente) com cada um dos outros computadores, sempre a um custo mínimo, não importando o custo total da rede. Como a rede é bastante segura, não há a necessidade de haver mais de uma rota de comunicação entre dois computadores pois sempre será utilizada a de menor custo. Sendo assim, algumas conexões nunca serão utilizadas, de forma que não precisam ser implantadas. Descreva, do ponto de vista de Teoria de Grafos, como o problema pode ser resolvido.



Primeira Prova - 18/10/11

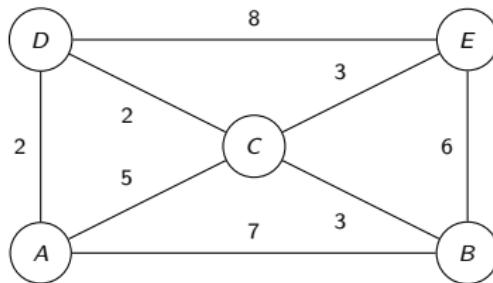
1) (1.5 pontos) Um cachorro, um gato e um rato estão na margem de um rio. Um barqueiro deseja levá-los até a outra margem. Uma vez que o barco é pequeno, ele apenas pode levar um deles por vez. Por razões óbvias, nem o cachorro e o gato, nem o gato e o rato podem ficar sozinhos. Como o barqueiro irá fazer para transportá-los? Descreva como esse problema pode ser resolvido utilizando teoria de grafos e desenhe (pelo menos parcialmente, mínimo de 7 estados) o grafo que modela o problema.

2)(1 ponto) Quantos grafos diferentes de N vértices existem? Encontre uma expressão matemática do número de grafos em função de N .

- 3)(1.5 pontos) Como se pode determinar quantos caminhos diferentes existem entre um dado par de vértices em um grafo?
- 4)(1 ponto) Um grafo conexo é dito euleriano se possui um ciclo euleriano. E semi-euleriano se possui um caminho euleriano mas não possui um ciclo euleriano. Determine os valores de n para os quais K_n é euleriano e os valores para os quais ele é semi-euleriano.

5)(1 ponto) É possível que um grafo bipartido com um número ímpar de vértices seja hamiltoniano (isso é, contenha um ciclo hamiltoniano)? E euleriano? Justifique sua resposta.

6)(2 pontos) Aplique o algoritmo de Dijkstra (o algoritmo está ao final da prova) ao grafo a seguir, para calcular a distância entre os vértices A e E . Mostre o vetor com os λ a cada vez que passar pelo passo 3.



Primeira Prova - 07/05/2012

2) (1.0 ponto) Para um número natural r , um grafo é r -regular se todos os vértices têm grau r . Para um grafo r -regular com n vértices e m arestas, expresse m em função de n e r .

6) (1.0 ponto) Considere um grafo G representado em uma matriz de adjacências M . Descreva um algoritmo baseado no produto matricial para verificar se o grafo G possui ciclos de comprimento igual a 3.

Primeira Prova - 05/10/2012

- 1)(1.0 ponto) Quantos subgrafos induzidos com 2 vértices possui um grafo de n vértices e m arestas?
- 4)(1.0 ponto) Seja T uma árvore com vértices $1, \dots, n$. Suponha que os graus dos vértices $1, 2, 3, 4, 5, 6$ são $7, 6, 5, 4, 3, 2$ respectivamente e que os vértices $7, \dots, n$ são folhas. Determine n .

- 6)(1.0 ponto) Considere um grafo G representado em uma matriz de adjacências M . Vimos em aula um algoritmo baseado em produto matricial para encontrar a quantidade de caminhos entre dois vértices. Descreva como o algoritmo poderia ser utilizado para verificar se o grafo G possui ciclos de comprimento igual a 3.
- 7)(1.0 ponto) O algoritmo de Dijkstra somente pode ser utilizado em grafos em que os pesos não positivos. Construa um grafo com pesos que podem ser negativos e no qual o algoritmo de Dijkstra devolve a resposta errada.
- 8)(1.0 ponto) Qual o menor número de arestas que devem ser retiradas de K_6 para que ele se torne planar?

- 2) (1.0 ponto) Para um número natural r , um grafo é r -regular se todos os vértices têm grau r . Para um grafo r -regular com n vértices e m arestas, expresse m em função de n e r .
- 4) (2.0 pontos) Um grafo valorado G com custo $c(i,j)$ entre cada aresta (i,j) satisfaz a desigualdade triangular se para todos os vértices $u, v, w \in V$:

$$c(u, w) \leq c(u, v) + c(v, w) \quad (1)$$

Escreva uma função (em C ou português estruturado) que receba uma matriz de adjacência e verifique se o grafo atende à desigualdade triangular, retornando: 0 - Se não atende; 1 - Se atende.

8) (1 ponto) Considere o problema de, em um grafo valorado em que são atribuídos pesos às arestas, encontrar o caminho que possui o menor valor para a aresta máxima do caminho. Que alterações devem ser feitas no algoritmo de Dijkstra (mostrado a seguir) para que encontre esse caminho?

- 2) (1 ponto) Considere um grafo com n vértices, m arestas e k componentes conexas. Quantas arestas é necessário acrescentar para torná-lo conexo?
- 3) (2 pontos) Uma forma de encontrar o centro de uma árvore é ir removendo, a cada iteração, todas as folhas, até que sobrem somente um ou dois vértices, que serão os pontos centrais. Faça uma função (em C ou português estruturado) que receba uma matriz de adjacências representando uma árvore de 10 vértices e escreva o número do vértice ou vértices que compõem o centro da árvore.

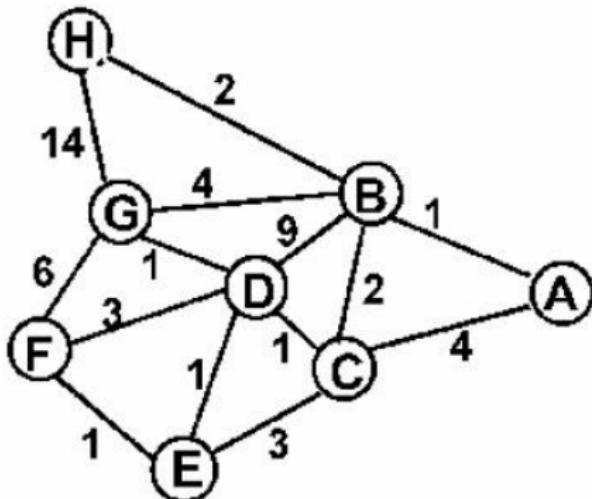
4) (1 ponto) Dadas as seguintes afirmações

- 1 Qualquer grafo conexo com n vértices deve ter pelo menos $n - 1$ arestas.
- 2 O grafo bipartido completo $K_{m,n}$ é Euleriano desde que m e n sejam ímpares.
- 3 Em um grafo o número de vértices de grau ímpar é sempre par.

São verdadeiras:

- 1 Somente a afirmação (1).
- 2 Somente as afirmações (1) e (3).
- 3 Somente as afirmações (2) e (3).
- 4 Somente as afirmações (1) e (2).
- 5 Todas as afirmações.

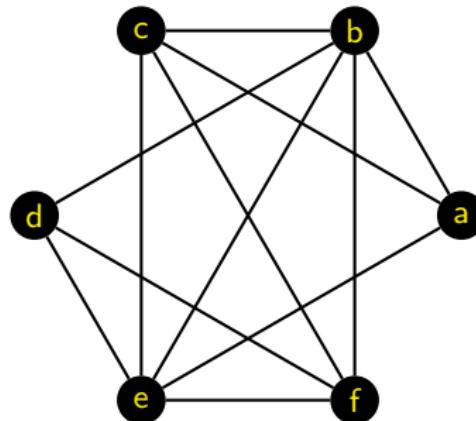
- 5) (1 ponto) Encontre uma expressão que relate a quantidade de grafos possíveis para N vértices.
- 6) (2 pontos) Aplique o algoritmo de Dijkstra ao grafo a seguir, para calcular a distância entre o vértice F e todos os outros.
Mostre o vetor com os λ a cada vez que passar pelo passo 3.



7) (2 pontos) O algoritmo de Floyd-Warshall encontra o caminho mínimo entre todos os pares de vértices do grafo. É baseado em programação dinâmica, de forma que a solução final vai sendo composta a partir das soluções parciais. A cada passo k o algoritmo calcula o caminho mínimo entre dois vértices que passa apenas por vértices de numeração menor ou igual a k .
Aplique o algoritmo de Floyd-Warshall no grafo representado pela matriz de adjacências a seguir, mostrando a matriz de distâncias a cada iteração.

0	3	8	∞
∞	0	∞	1
∞	4	0	∞
2	∞	5	0

4) (2.5 pontos) Aplique o teste de planaridade de demoucron ao grafo a seguir para determinar se é planar ou não. Considere o ciclo inicial como b-c-e-b. A cada iteração mostre as pontes e as faces, até o final do algoritmo.



Recuperação da Primeira Prova - 11/07/11

3) (1 ponto) Seja T uma árvore com $p + q$ vértices. Suponha que p dos vértices tem grau 4 e q são folhas. Qual a relação matemática entre p e q ?

Recuperação da Primeira Prova - 07/12/12

- 3) (2 pontos) No grafo do exercício anterior, aplique o algoritmo de Tremaux, a partir do vértice A , mostrando como ficam ao final as marcações das passagens. Use como critério de escolha entre dois vértices, sempre o de menor ordem alfabética.
- 4)(1 pontos) Como se pode determinar quantos caminhos simples (isso é, que passam somente uma vez em cada vértice) diferentes existem entre um dado par de vértices em um grafo?

Primeira Prova - 17/10/14

1)(1 ponto) Qual o número máximo de arestas que um grafo NÃO CONEXO simples (sem laços ou arestas paralelas) com 30 vértices pode conter?

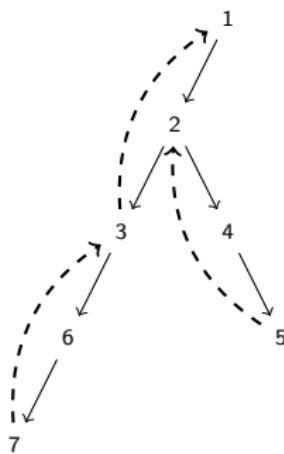
2)(1 ponto) É possível haver um grafo simples (sem laços ou arestas paralelas) com mais de um vértice em que todos os vértices tenham graus diferentes? Justifique sua resposta.

3)(1 ponto - POSCOMP 2014) Considerando que um grafo possui n vértices e m arestas, assinale a alternativa que apresenta, corretamente, um grafo planar.

- a) $n = 5, m = 10$
- b) $n = 6, m = 15$
- c) $n = 7, m = 21$
- d) $n = 8, m = 12$
- e) $n = 9, m = 22$

4)(1 ponto) É possível construir um grafo bipartido planar com no mínimo 5 vértices de grau 3?

5)(2 pontos) A figura a seguir representa a árvore resultante da busca em profundidade e as arestas de retorno de um grafo G . Identifique o lowpt de cada vértice, as articulações, demarcadores e componentes biconexas de G .



6)(2 pontos) Dado um grafo D com arestas valoradas com pesos entre 0 e 1, representando a probabilidade de uma aresta não falhar; a probabilidade de um caminho em D estar operacional é o produto das probabilidades das arestas desse caminho (i.e., estamos supondo independência das probabilidades de não falhar). Mostre que alterações devem ser feitas no algoritmo de Dijkstra (ao final dessa prova) para descobrir o caminho mais seguro entre dois vértices.

7) (2 pontos) Em teoria dos grafos, o complemento ou inverso de um grafo G é um grafo H que tem os mesmos vértices de G , tais que dois vértices de H são adjacentes se e somente se eles não são adjacentes em G , isso é, o conjunto de arestas de H é formado por todas as arestas que não estão em G . Escreva uma função (na linguagem de sua escolha, incluindo pseudo-código) que receba duas matrizes de adjacências representando dois grafos de 10 vértices, e verifique se um grafo é o complemento do outro, retornando: 0 - Se não são complementares; 1 - Se são complementares.

- 1) (1.0 ponto) Seja G um grafo com 14 vértices e 25 arestas. Se todo vértice de G tem grau 3 ou 5, quantos vértices de grau 3 o grafo G possui?
- 2) (1.0 ponto) Para um número natural r , um grafo é r -regular se todos os vértices têm grau r . Para um grafo r -regular com n vértices e m arestas, expresse m em função de n e r .
- 3) (1.0 ponto) Como é possível mostrar que um grafo não é planar?
- 4) (1.0 ponto) Sejam G_1 e G_2 dois grafos conexos, sem vértices em comum. Seja x um vértice de G_1 e y um vértice de G_2 . Seja G o grafo que se obtém de $G_1 \cup G_2$ acrescentando a aresta (x, y) . É possível que o grafo G seja euleriano? E semi-euleriano (contém um caminho euleriano mas não contém um ciclo euleriano)? Em que condições?
- 5) (1.0 ponto) Quantos subgrafos induzidos com 2 vértices possui um grafo de n vértices e m arestas?

- 6) (1 ponto) Seja T uma árvore com vértices $1, \dots, n$. Suponha que os graus dos vértices $1, 2, 3, 4, 5, 6$ são $7, 6, 5, 4, 3, 2$ respectivamente e que os vértices $7, \dots, n$ são folhas. Determine n .
- 7) (1 ponto) Um grafo é dito r -regular se todos os vértices tem grau r . Dê um exemplo de um grafo 3-regular que não seja completo.
- 8) (1 ponto) Uma floresta é um grafo desconexo em que todas as componentes conexas são árvores. Determine o número de arestas de uma floresta com n vértices e k componentes conexas, em função de n e k .
- 9) (1 ponto) Como se pode determinar quantos caminhos diferentes existem entre um dado par de vértices em um grafo?
- 10) (1 ponto) Quantos grafos diferentes de N vértices existem? Encontre uma expressão matemática do número de grafos em função de N .

Recuperação da Segunda Prova - 11/07/11

1) (2.0 pontos) Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade. Os números entre parênteses nos vértices a e c representam o fluxo máximo através do vértice.

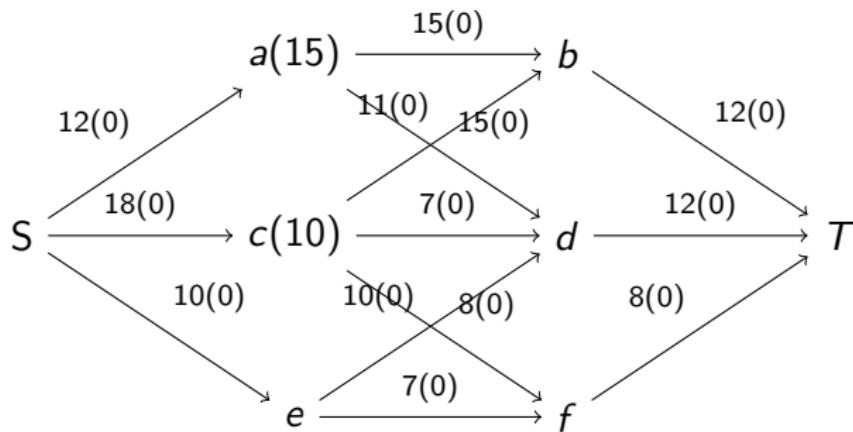


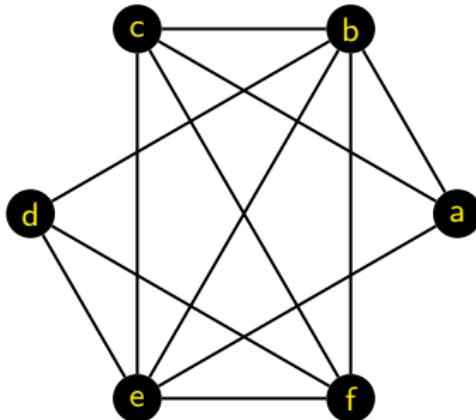
Figura: Encontre o fluxo máximo

2) (2.0 pontos) Utilizando o algoritmo por programação dinâmica visto em aula, encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

	1	2	3	4
1	0	3	∞	5
2	6	0	4	5
3	∞	4	0	2
4	2	7	3	0

4) (2.0 pontos) Apresente 5 ordenações topológicas para o dígrafo resultante do exercício 3.

5) (2.0 pontos) Aplique o teste de planaridade de demoucron ao grafo a seguir para determinar se é planar ou não. Considere o ciclo inicial como b-c-e-b. A cada iteração mostre as pontes e as faces, até o final do algoritmo.



Recuperação da Segunda Prova - 08/12/10

1) (2.5 pontos) Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade. Os números entre parênteses nos vértices a e c representam o fluxo máximo através do vértice.

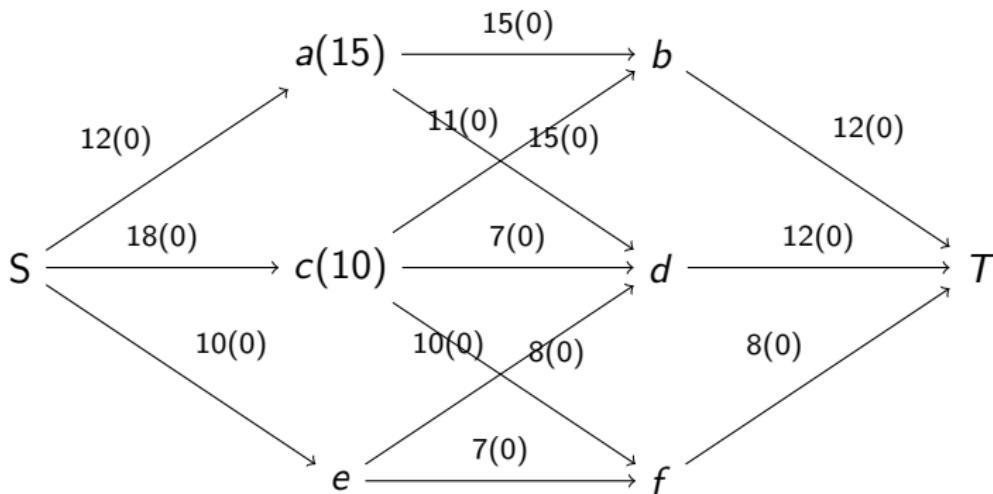
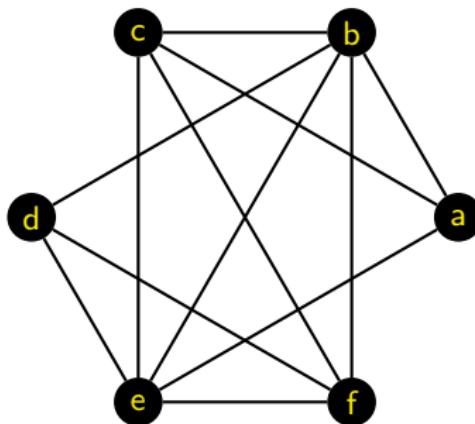


Figura: Encontre o fluxo máximo

2) (2.5 pontos) Utilizando o método de programação dinâmica encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir. Mostre todas as funções calculadas.

	1	2	3	4
1	0	13	∞	5
2	6	0	14	5
3	∞	14	0	2
4	12	7	13	0

4) (2.5 pontos) Aplique o teste de planaridade de demoucron ao grafo a seguir para determinar se é planar ou não. Considere o ciclo inicial como b-c-e-b. A cada iteração mostre as pontes e as faces, até o final do algoritmo.



Segunda Prova - 29/11/13

1) (2 pontos) Para a festa de fim de ano da UCS foram montados 6 comitês, o de Decoração, o de Comidas, o de Bebidas, o de Espetáculo, o de Fundos e o de Música. João está no de Decoração e de Comidas, José está no de Bebidas e de Fundos, Juca está no de Música, Joca está no de Música e de Espetáculo, Jeca está no de Fundos e de Música, Jaca está no de Decoração e de Espetáculo e Julio está no de Bebidas e de Fundos. Deseja-se montar uma comissão de 6 pessoas de modo que cada pessoa represente um comitê do qual participe, cada pessoa representando um comitê diferente. Mostre como é possível resolver (genericamente) esse problema utilizando teoria de grafos, e encontre a solução para o caso específico descrito nessa questão.

- 2) (2 pontos) Você foi encarregado de montar a tabela de um campeonato com 10 times, em que todos jogam contra todos, durante 9 rodadas. Descreva como a teoria de grafos pode auxiliar você nessa tarefa.
- 3) (2 pontos) Utilizando o método de programação dinâmica encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir. Mostre todas as funções calculadas.

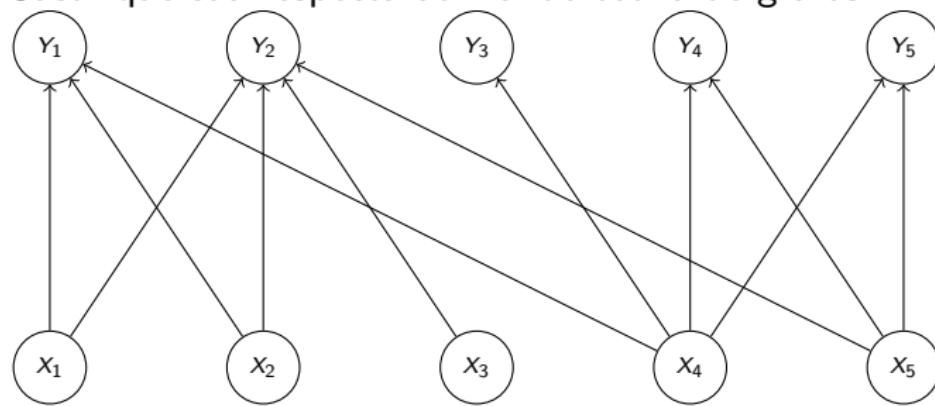
	1	2	3	4
1	0	10	∞	5
2	4	0	10	5
3	∞	12	0	2
4	12	9	13	0

5) (2 pontos) De três depósitos A, B e C, dispondo respectivamente de 20,10 e 35 toneladas de um dado produto, pretende-se fazer chegar a três destinos D, E e F, respectivamente 25, 20 e 20 toneladas do produto. As disponibilidades de transporte em caminhão entre os difentes pontos são os seguintes:

	D	E	F
A	15	10	-
B	5	-	10
C	10	5	5

Mostre como a teoria de grafos pode ser utilizada para estabelecer o melhor plano de transportes, e encontre a solução para a situação específica descrita nessa questão.

2) (2 pontos) O grafo a seguir possui uma associação perfeita?
Justifique sua resposta utilizando teoria de grafos.



- 3)(2 pontos) Qual o número cromático do grafo K_7 após excluídas as arestas $(1,2), (1,3), (1,4), (2,3)$ e $(2,4)$? Justifique sua resposta.
- 4) (2 pontos) Encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

	1	2	3	4	5
1	0	3	∞	5	3
2	6	0	4	∞	∞
3	∞	4	0	2	2
4	2	∞	3	0	2
5	2	∞	3	2	0

5)(2 pontos) Considere o problema 820 da ACM, descrito a seguir. Como esse problema poderia ser modelado e resolvido com teoria de grafos? Descreva em detalhes o grafo que modelaria o problema, o que se buscaria nesse grafo e qual o algoritmo utilizado. Caso seja necessário modificar o algoritmo, descreva o que deveria ser modificado no mesmo. Qual a resposta (numérica) para a instância do problema utilizada como exemplo do problema?

Problema 820 da ACM - Largura de Banda na Internet

O Problema

Na Internet, as máquinas (nós) são interconectados, e diferentes caminhos podem existir entre um dado par de nós. A capacidade de transferência total (largura de banda) entre dois nós é dada pela quantidade máxima de dados por unidade de tempo que pode ser transmitida de um nó para outro. Usando uma técnica chamada de comutação de pacotes, estes dados podem ser transmitidos ao longo de vários caminhos ao mesmo tempo.

Por exemplo, a tabela a seguir mostra a capacidade de transferência entre cada par de nós, para uma rede de 4 nós.

	1	2	3	4
1	0	20	10	0
2	20	0	5	10
3	10	5	0	20
4	0	10	20	0

No exemplo, a largura de banda entre o nó 1 e o nó 4 é o total de pacotes que pode ir do nodo 1 até o nodo 4 por unidade de tempo.

Entrada

O arquivo de entrada contém descrições de várias redes. Toda descrição começa com uma linha contendo um único inteiro $n(2 \leq n \leq 100)$, que é o número de nós na rede. Os nós são numerados de 1 a n . A próxima linha contém três números, s , t e c . Os números s e t são os nós de origem e destino, e c é o número total de conexões na rede. Após isto há c linhas descrevendo as ligações. Cada uma dessas linhas contém três inteiros: os dois primeiros são os números dos nós conectados, e o terceiro número é a largura de banda da conexão. A largura de banda é um número não-negativo não superior a 1000.

Pode haver mais de uma conexão entre um par de nós, mas um nó não pode ser conectado a si mesmo. Todas as conexões são bidirecionais, ou seja, os dados podem ser transmitidos em ambas as direções ao longo de uma conexão, mas a soma da quantidade de dados transmitidos em ambos os sentidos deve ser menor que a largura de banda.

Uma linha contendo o número 0 segue a descrição da última rede, encerrando a entrada.

Saída

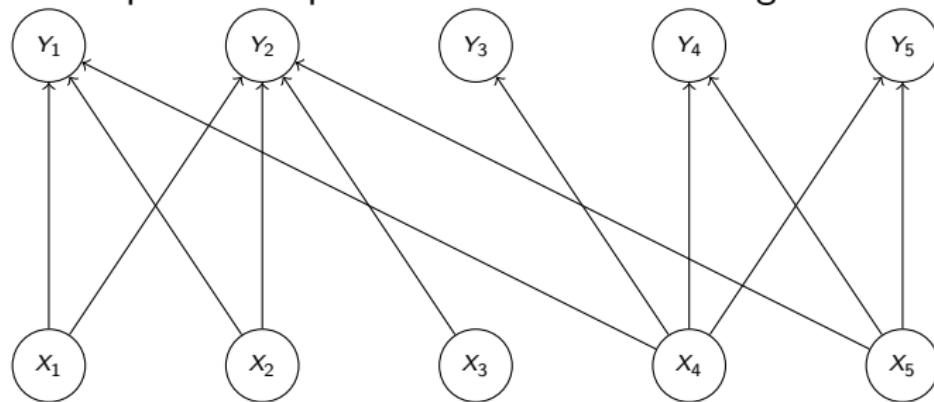
Para cada rede deve ser impresso o número da rede. Em seguida, imprima a largura de banda total entre o nó origem s e o nó destino t . Imprima uma linha em branco após cada caso de teste.

2) (2 pontos) Encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

	1	2	3	4	5
1	0	3	∞	5	3
2	6	0	4	∞	∞
3	∞	4	0	2	2
4	2	∞	3	0	2
5	2	∞	3	2	0

- 3) (2 pontos) Mostre todos os passos do método de alteração estrutural para a obtenção do número cromático do grafo K_5 (vértices identificados pelas letras de a a e) após excluídas as arestas (c, d) e (b, e) .
- 4) (2 pontos) Considere um tabuleiro de xadrez, de 8 por 8 casas. É possível cobrir todo o tabuleiro com peças de dimensão 1×2 ? E se forem removidas duas casas de cantos opostos? Esse problema pode ser resolvido através de teoria de grafos? Como?

1)(2 pontos) O grafo a seguir possui uma associação perfeita?
Justifique sua resposta utilizando teoria de grafos.



2) (2 pontos) Em uma nação emergente, cada uma das seis principais cidades receberá uma estação de televisão. Diferentes estações podem usar a mesma faixa de frequência, contanto que estejam distanciadas de mais de 80 quilômetros, para evitar problemas de interferência. Suponha que as cidades tenham os seguintes nomes: A, B, C, D, E e F, e as distâncias entre elas estão descritas na tabela a seguir:

	B	C	D	E	F
A	150	135	96	76	84
B		96	147	90	102
C			72	76	51
D				69	48
E					24

Qual o número mínimo de faixas de frequência necessárias, e como pode ser feita a distribuição entre as cidades? Como isso pode ser resolvido através de Teoria de Grafos?

4) (2 pontos) Encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

	1	2	3	4	5
1	0	3	∞	5	3
2	6	0	4	∞	∞
3	∞	4	0	2	2
4	2	∞	3	0	2
5	2	∞	3	2	0

5) (2 pontos) Considere que você é o Chefe do Departamento de Informática e você deve fazer a alocação das disciplinas aos professores para o próximo semestre. Você dispõe, para cada professor, da lista de disciplinas que ele é capaz de ministrar e do número máximo de disciplinas que o professor pode ministrar. Além disso você dispõe da lista de disciplinas que deverá ser oferecida no próximo semestre, e o número de turmas que deve ser oferecido para cada disciplina. Descreva como você pode resolver o problema, ou seja, chegar a uma atribuição de disciplinas aos professores que respeite suas competências e sua carga horária máxima de forma todas as turmas de todas as disciplinas sejam atendidas. Invente uma situação hipotética para ilustrar seu modelo, contendo ao menos 4 professores e 4 disciplinas.

Segunda Prova - 05/12/11

1)(2 pontos) O curso de inglês AILÓVIU pretende enviar 20 alunos/alunas para fazer um intercâmbio de um mês na Inglaterra. Os alunos ficarão em casas de família, dois alunos em cada casa. Cada aluno forneceu à coordenação do curso uma lista com suas preferências de quem gostaria de dividir residência. Observe, entretanto, que a relação de "preferência" não é necessariamente simétrica, isso é, o Juca gostaria de dividir residência com o Zé, mas o Zé não quer dividir residência com o Juca. Explique, DETALHADAMENTE, como o problema poderia ser modelado e resolvido com teoria de grafos. Se for necessário, invente um exemplo.

3)(2 pontos) Encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

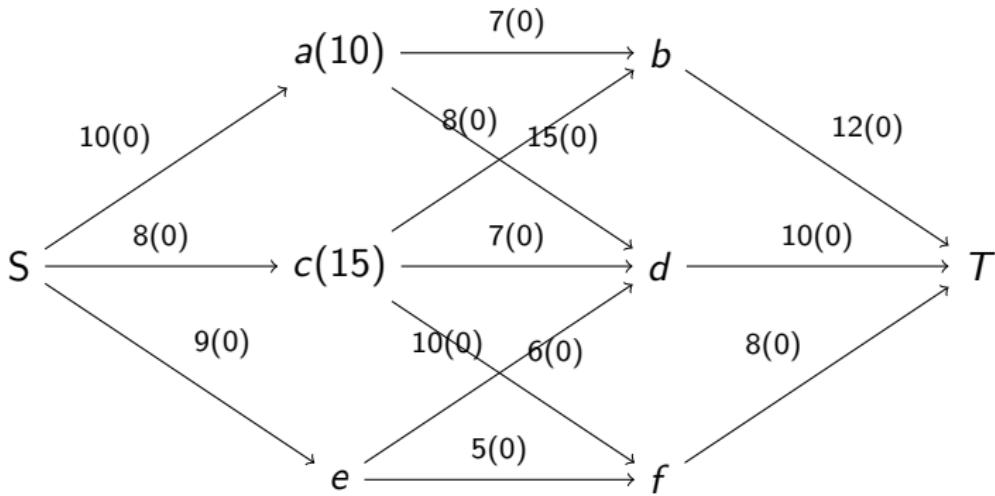
	1	2	3	4	5
1	0	3	∞	5	3
2	6	0	4	∞	∞
3	∞	4	0	2	2
4	2	∞	3	0	2
5	2	∞	3	2	0

4)(2 pontos) Descreva um algoritmo para obter uma ordenação topológica dos vértices de um dígrafo. Dê 5 ordenações topológicas para o dígrafo do exercício 2.

4)(2 pontos) Usando o algoritmo de programação dinâmica visto em aula, encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir:

	1	2	3	4
1	0	7	6	5
2	4	0	5	7
3	∞	2	0	2
4	2	7	3	0

5) (2 pontos) Encontre um fluxo máximo para a rede abaixo. O número na aresta representa sua capacidade. Os números entre parênteses nos vértices a e c representam o fluxo máximo através do vértice. Mostre pelo menos uma iteração do algoritmo de fluxo máximo, mostrando a rede residual gerada.

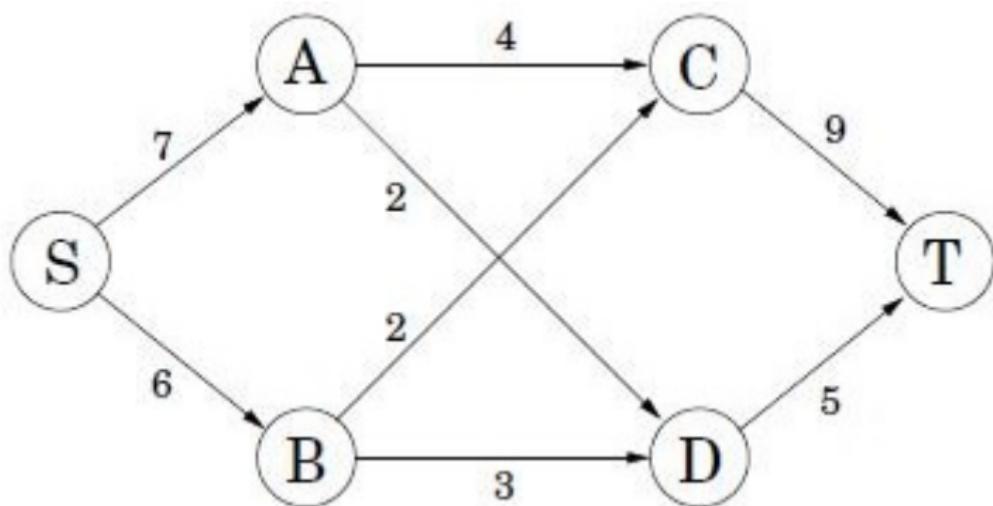


Segunda Prova - 05/12/14

1)(2 pontos) Considere o problema a seguir: Um hospital possui 10 médicos, cada um com uma lista de noites no mês (considere um mês de 4 semanas) em que ele tem disponibilidade para plantões. Como se poderia fazer a alocação dos médicos de modo que haja 2 médicos de plantão a cada noite? Mostre em detalhes como esse problema poderia ser modelado e resolvido utilizando teoria de grafos. Como se poderia acrescentar nesse modelo a restrição de que cada médico não pode atender mais de dois plantões na semana?

3)(2 pontos) Um conjunto dominante é um subconjunto de vértices tal que todo vértice do grafo está no conjunto ou é adjacente a um dos vértices. Escreva uma função (pode ser pseudo-código) que receba uma matriz de adjacências $G[10][10]$ e um vetor $Dom[10]$ representando um subconjunto de vértices de G (os vértices pertencentes ao subconjunto estão marcados com 1 no vetor, os que não pertencem estão marcados com 0) e verifique se o subconjunto descrito em $Dom[10]$ é um Conjunto Dominante de Vértices, retornando: 1 - Se Dom é um conjunto dominante; 0 - Se Dom não é um conjunto dominante.

4)(2 pontos) Utilizando o algoritmo de Ford-Fulkerson visto em aula, calcule o fluxo máximo da rede a seguir, mostrando a rede residual e o novo fluxo a cada passo. Os valores representados nas arestas são as capacidades das arestas.



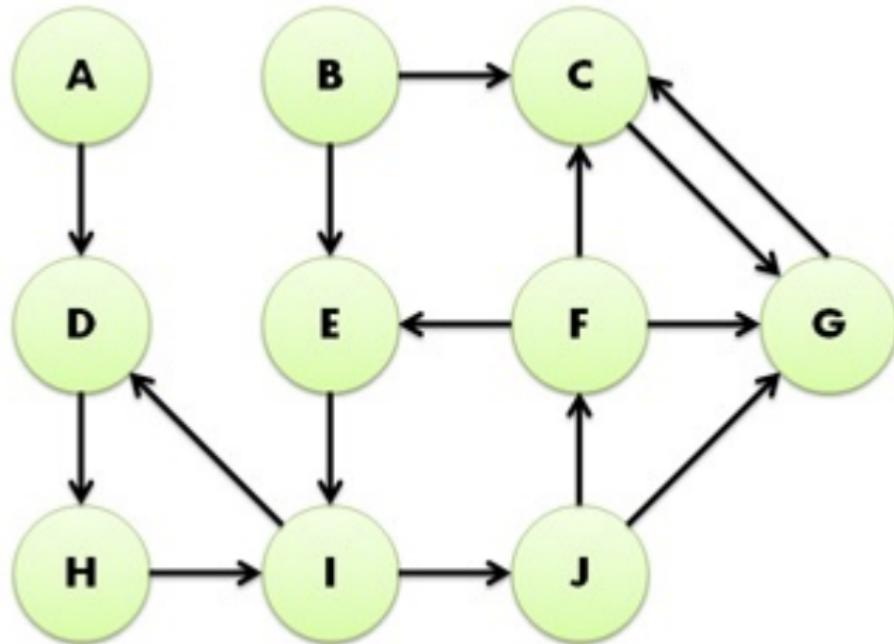
5) (2 pontos) Um vendedor de brinquedos educativos possui em estoque brinquedos de várias formas geométricas (cubos, pirâmides, etc.), cada qual fabricado em várias cores. O vendedor quer carregar consigo o menor número de objetos tal que cada cor e cada forma estejam representadas pelo menos uma vez. Explique como esse problema pode ser modelado e resolvido com teoria de grafos.

Recuperação da Segunda Área - 12/12/14

3) (2 pontos) Utilizando o método de programação dinâmica encontre o ciclo hamiltoniano de custo mínimo no grafo a seguir. Mostre todas as funções calculadas.

	1	2	3	4
1	0	10	∞	5
2	4	0	10	5
3	∞	12	0	2
4	12	9	13	0

5) (2 pontos) Identifique as componentes fortemente conexas do grafo abaixo:

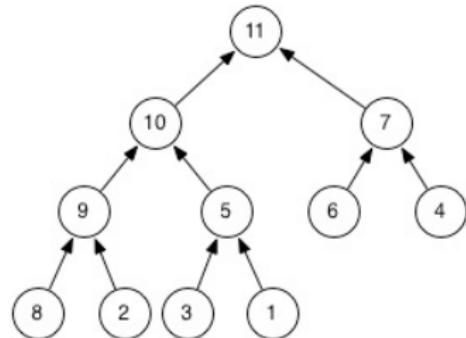


Exercícios de Grafos do URI Online Judge

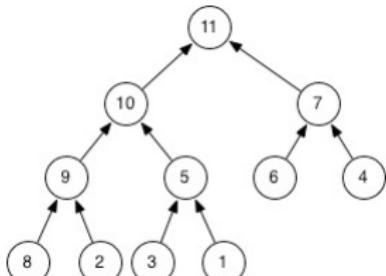
- Prim com fila de prioridades - 1152 - Estradas Escuras

Heaps, priority queues e Heapsort

- Um **heap** é uma estrutura de dados organizada como árvore binária em que todos os níveis até o penúltimo estão completos, e o último nível é preenchido "em ordem", da esquerda para a direita.
- Além disso, o valor da chave em cada nodo deve ser maior (maxheap) ou menor (minheap) que o valor da chave nos dois filhos, e a ordem da chave entre os filhos não interessa.



- Um heap pode ser implementado sobre um vetor, onde cada elemento $v[i]$ é pai dos elementos nas posições $v[2*i+1]$ e $v[2*i+2]$ e a raiz da árvore ocupa a primeira posição ($v[0]$)).
- Assim, dado o índice de um nodo no vetor, os índices do pai e dos filhos à esquerda e à direita podem ser obtidos por:
 - $\text{Pai}(i) : (i-1)/2$
 - $\text{Esquerda}(i) : i * 2 + 1$
 - $\text{Direita}(i) : i * 2 + 2$
- Assim, o heap abaixo poderia ser representado pelo vetor [11 10 7 9 5 6 4 8 2 3 1]



- As principais operações sobre heaps são:
 - cria-heap: cria um heap vazio
 - heapify: cria um heap a partir de um conjunto de elementos
 - busca-maior (em um heapmax) ou busca-menor(em um heapmin)
 - remove-maior: remove a raiz de um heapmax
 - altera-valor-chave
 - insere: adiciona elemento a um heap

Inserção em Heap

- Pode ser implementada adicionando o novo elemento no final do heap e depois "empurrando-o" para cima enquanto ele for maior que o pai (no caso de um Heap Max) até alcançar sua posição.
- Qual o custo?

```
void corrigeSubindo(int index)
{
    // Se index não é a raiz e o valor do index for maior do que o valor de seu pai,
    // troca seus valores (index e pai(index)) e corrige para o pai
    if (index > 0 && heap[index] > heap[pai(index)])
    {
        troca(heap[index], heap[pai(index)]);
        corrigeSubindo(pai(index));
    }
}
```

Defines

- No código anterior, uma forma de implementar o cálculo do índice do pai é através de uma função. Uma forma melhor ainda é através de um define.
- Normalmente defines são utilizados para melhorar a legibilidade e manutenção do código. Coisas como o código abaixo, para definir o tamanho de um vetor, de modo que, ao alterar o tamanho do vetor, não seja necessário inspecionar todo o código:
- `#define TAM 10`

- defines podem receber parâmetros, o que possibilita o seu uso no lugar de funções, com a mesma legibilidade de funções mas sem o custo de chamadas e retornos (o código do define é expandido no lugar do uso do define).
- Os parâmetros vão entre parênteses, como o define abaixo, que calcula o maior valor entre duas expressões
- `#define max(A,B) ((A)>(B)?(A):(B))`
- ou os defines abaixo, para o cálculo da posição do pai, filho esquerdo e filho direito no heap.
 - `#define pai(i) ((i-1)/2)`
 - `#define fesq(i) (i*2+1)`
 - `#define fdir(i) (i*2+2)`
- Pode-se passar expressões, mas o que resultaria no exemplo acima ao usar-se "fesq(1+2)" ?

Remoção

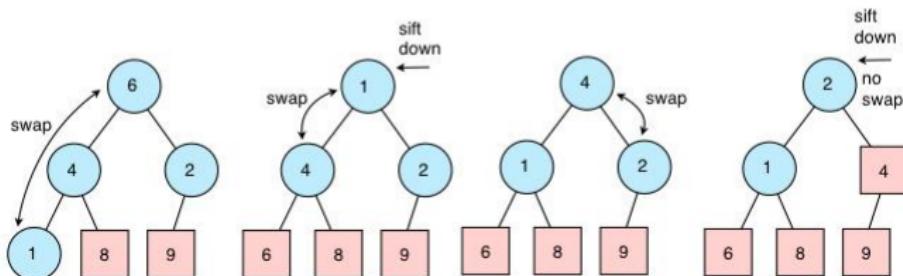
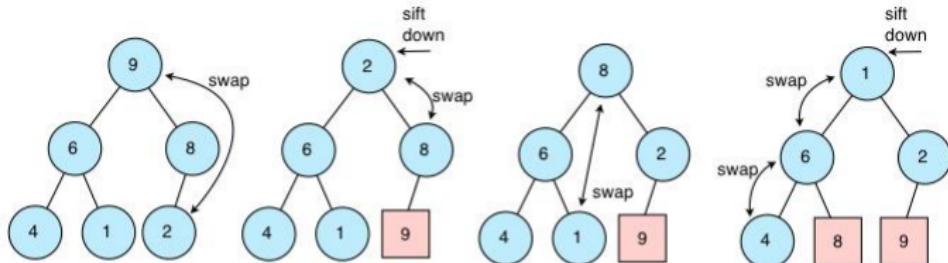
- A remoção do maior valor da raiz (no caso do Heap Max) pode ser implementada removendo a raiz e substituindo-a pelo último elemento do Heap e "empurrando-o" para baixo na árvore até que o heap esteja corrigido.
- Qual o custo?

```
void corrigeDescendo(int index)
{
    if (filhoE(index) < heap.size())      // Se index tem filho
    {
        // Seleciona o filho com menor valor (esquerda ou direita ?)
        int filho = filhoE(index);
        if (filhoD(index) < heap.size() && heap[filhoD(index)] > heap[filhoE(index)])
            filho = filhoD(index);
        // Se o valor do pai é maior do que o valor do maior filho , terminamos
        if (heap[index] > heap[filho])
            return;
        // Caso contrário , trocamos o pai com o filho no heap e corrigimos agora para o filho
        troca(heap[index], heap[filho]);
        corrigeDescendo(filho);
    }
}
```

- Heaps podem ser utilizados para implementar eficientemente filas de prioridades, já que o custo das principais operações (inserção, remoção) é $O(\log n)$ e o custo para obter o maior elemento é $O(1)$.
- Filas de prioridade são utilizadas em diversos algoritmos clássicos tais como:
 - Algoritmo de Dijkstra para caminho mínimo
 - Algoritmos de Prim e Kruskal para árvore geradora mínima
 - Algoritmo de Branch-and-bound para buscar nodo de melhores possibilidades

- Heaps também são utilizados para implementar o algoritmo de ordenação Heapsort que basicamente é:
 - Coloque os dados a serem ordenados em um heap (custo $O(n \log n)$)
 - A cada iteração remova a raiz do heap (maior valor) e coloque no final do vetor, e reorganize o heap ($O(\log n)$)

Heapsort



- Diversas linguagens tem bibliotecas prontas para a implementação de heaps. Em python há uma implementação no pacote **heapq**.
- O heap é implementado sobre uma lista. É um min-heap. Para implementar um max-heap pode-se inverter o sinal dos valores ao inserir e retirar, de modo que o maior passará a ser o menor.

```
import heapq
```

- As principais funções são:
 - heappush(heap,item) - Coloca o item no heap
 - heappop(heap) - retorna o menor elemento do heap removendo-o do mesmo. Se heap estiver vazio, gera um IndexError.
 - heappushpop(heap,item) - Coloca o item no heap e retorna o menor elemento.
 - heapify(x) - Transforma a lista x em um heap em tempo linear.

- Para acessar o menor elemento sem removê-lo do heap basta acessar a posição 0.
- Os itens do heap podem ser tuplas. Nesse caso a comparação é feita pelo primeiro valor e, em caso de empate, pelo segundo valor (e pelo terceiro, etc.)
- O código a seguir recebe um iterador e monta um min_heap a partir dele:

```
>>> def heapsort(iterable):
...     h = []
...     for value in iterable:
...         heappush(h, value)
...     return [heappop(h) for i in range(len(h))]
...
>>> heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Priority_queue (fila de prioridade) em C++

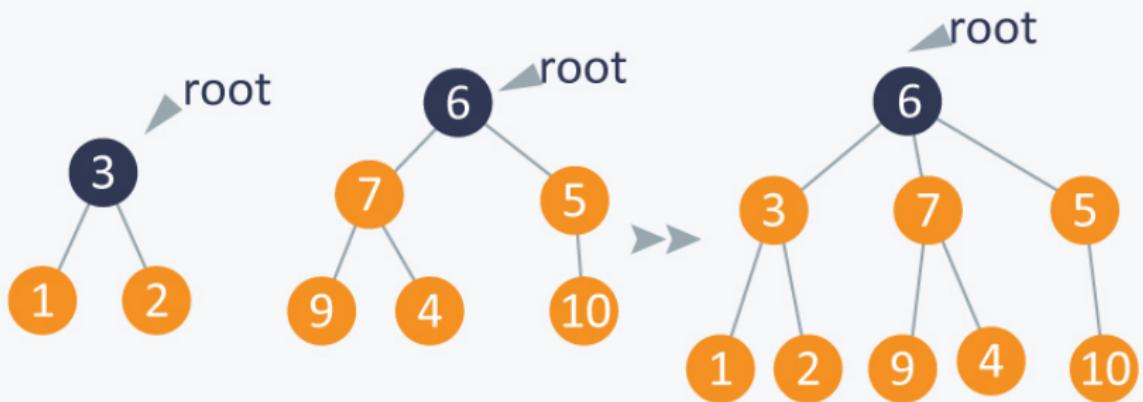
- O container **priority_queue** implementa uma fila de prioridade. Essa estrutura se caracteriza por permitir identificar o maior (ou menor) elemento em tempo $O(1)$, removê-lo em tempo $O(\log n)$ e inserir novos elementos em tempo $O(\log n)$.
- É útil em algoritmos como o de Prim ou o de Dijkstra, em que um passo importante, e custoso, do algoritmo é a identificação, a cada passo do elemento de menor valor em um conjunto (no caso, a distância de cada vértice ao vértice inicial, no Dijkstra, ou à sub-árvore já formada, no Prim)
- Ele está definido no módulo queue (include queue)

- A versão default implementa uma fila de prioridade max-heap, devolvendo a cada vez o maior valor da fila.
 - priority_queue <int> g;
- Os principais métodos para manipulação de filas de prioridade são :
 - empty() - retorna true se a fila está vazia
 - push(val) - empilha um valor
 - top() - retorna o valor no topo da pilha
 - pop() - remove (e descarta) o valor no topo da pilha
- Para implementar um min-heap, em que o MENOR elemento é mantido no topo, usa-se a sintaxe enigmática abaixo:
 - priority_queue < int, vector < int >, greater < int >> nome;

Disjoint-sets (Union-finds)

- Um disjoint-set (também conhecido como union-find ou união-busca) é uma estrutura de dados que mantém o controle de um conjunto de elementos particionados em subconjuntos disjuntos.
- Ele fornece de forma eficiente operações para adicionar novos conjuntos, para mesclar conjuntos existentes e para determinar se os elementos estão no mesmo conjunto.
- Além de muitos outros usos, os disjoint-sets desempenham um papel fundamental no algoritmo de Kruskal de busca de árvore geradora mínima.

- As principais operações sobre um disjoint-set são duas:
 - União: união de dois subconjuntos em um único;
 - Busca: determina em qual subconjunto um elemento em particular está. Esta operação também pode ser utilizada para determinar se dois elementos estão em um mesmo subconjunto.
- Um disjoint-set é representado como uma floresta, onde cada um dos subconjuntos forma uma árvore, na qual cada elemento aponta para seu pai, e a raiz da subárvore contém o elemento representativo do subconjunto.



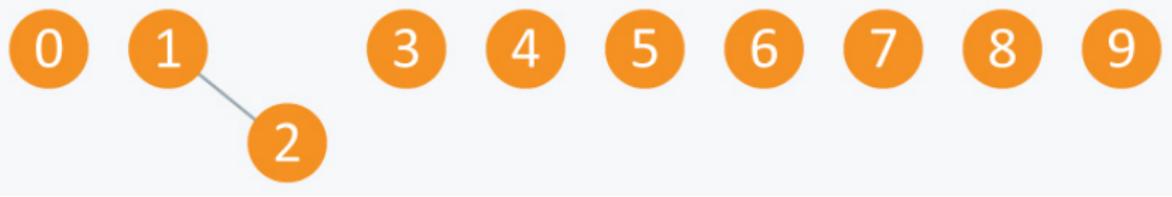
- Essa floresta pode ser representada por um vetor, onde o número da posição representa o vértice, e o conteúdo de cada posição contém o índice do pai. A posição do elemento que está na raiz da árvore contém seu próprio índice, permitindo identificar o elemento como raiz.
- Inicialmente cada elemento representa um subconjunto, sendo raiz da árvore de seu subconjunto, apontando para si mesmo
- Considere o conjunto de valores a seguir:



- Inicialmente cada elemento aponta para si mesmo e o conteúdo do vetor de "pais" é:

Arr	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

- Suponha que se faça a união dos conjuntos 1 e 2. Isso pode ser implementado fazendo com que o pai do conjunto 2 seja o nodo 1.



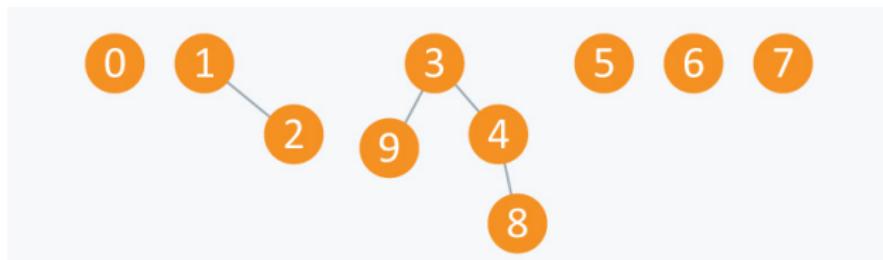
- E o vetor resultante após essa operação será:

Arr	0	1	1	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

- Supondo que se faça em seguida as operações união(4,3), união(8,4) e união(9,3), os subconjuntos resultantes serão:

Arr	0	1	1	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

- E o vetor resultante será:



- A função abaixo recebe o índice de um elemento e retorna o índice do elemento representativo de seu subconjunto, atualizando, no caminho, a informação do pai de cada nodo no caminho para a raiz:

```
int find(int x)
{
    if (pai[x] != x)
        pai[x] = find(pai[x]);
    return pai[x];
}
```

- E o código abaixo efetua a união dos subconjuntos dos elementos x e y . Se eles já fazem parte do mesmo subconjunto, a função não faz nada:

```
void union(int x, int y)
{
    int xset = find(x);
    int yset = find(y);
    if(xset==yset) return;
    pai[xset]=yset;
}
```

- A função anterior pode resultar em árvores bem desbalanceadas. Uma forma de melhorar o desempenho é armazenar junto à raiz da subárvore a altura da mesma, e "pendurar" a árvore mais baixa na árvore mais alta. Esse algoritmo é chamado de union-rank ("rank" é a altura da subárvore).
- O código do union-rank é mostrado na lâmina seguinte. Nele, cada elemento contém o índice do pai, e naqueles elementos que são raízes de árvores, o rank contém a altura da árvore:

```
typedef struct {int pai;int rank;} tnode;
tnodo ln[100001];
void union_rank(int x, int y)
{
    int xset = find(x);
    int yset = find(y);
    if(xset==yset) return;
    if (ln[xset].rank<ln[yset].rank)
        ln[xset].pai=yset;
    else
        ln[yset].pai=xset;
    if (ln[xset].rank==ln[yset].rank)
        (ln[xset].rank)++;
}
```

A STL (Standard Templates Library) do C++

- A **STL** é uma biblioteca que contem um conjunto de classes úteis para implementar alguns tipos abstratos de dados. Entre essas classes estão **containers**, que são tipos estruturados com métodos para executar diversas operações sobre eles. A STL oferece 5 containers básicos, 3 adaptadores de containers e alguns containers associativos, como dicionários e conjuntos.
- Vantagens do uso dos containers é que simplificam a codificação e algumas operações sobre eles são implementadas de forma muito eficiente.
- Para usar os componentes da stl é necessário colocar no início do código o comando "using namespace std;" para informar ao compilador que serão usadas funções ou classes da biblioteca padrão.
- Ou colocar "std:::" em cada uso de elemento da biblioteca.

- Os containers se diferenciam pela estrutura sobre a qual são implementados (vetor, lista encadeada, lista duplamente encadeada) e as operações que podem ser executadas sobre eles, definidas pelos métodos que oferecem.
- Os 5 containers básicos são:
 - vector
 - list
 - deque
 - arrays (a partir do C++11)
 - forward_list (a partir do C++11)

- Além dos tipos básicos, a STL provê também adaptadores de containers (container adaptors), implementados sobre outros containers, oferecendo operações adicionais.
- São eles:
 - Stack (pilha)
 - Queue (fila)
 - Priority Queue (fila de prioridade)

vector

- O vector é o container mais eficiente em relação a acesso a memória.
- Ele é implementado sobre uma área contínua de memória e permite acesso direto aos elementos, como um vetor, e inserção e remoção no final.
- A declaração de um vector é:
 - **vector** <tipo> nome;
- Ex:
 - **vector** <int> var1, var2;
- Para utilizar um vector é necessário colocar `#include <vector>`

- O acesso aos elementos de um vector pode ser feito diretamente como se fosse um vetor (ex: `vet[i]`), e nesse caso não é feita nenhuma verificação se o elemento existe no vector ou não.
- Ou pode ser feito pelo método `at()`, e no caso de ser feito um acesso a um elemento ainda não inserido é gerada uma exceção.

- Alguns dos principais métodos do vector são:
 - `front()` - devolve o valor do primeiro elemento. Se estiver vazio, resulta um comportamento indefinido.
 - `back()` - devolve o valor do último elemento
 - `empty()` - testa se o container está vazio
 - `push_back()` - insere um elemento no final
 - `pop_back()` - remove um elemento do final
 - `assign(int n, int v)` - preenche **n** posições do vetor com o valor **v**

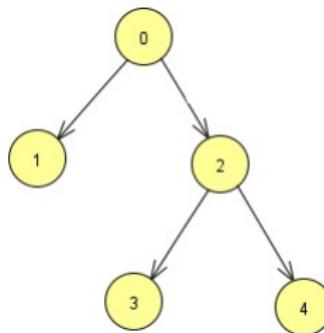
Iteradores

- Os containers possuem também a figura dos **iteradores**, que são objetos utilizados para percorrer sequencialmente os elementos de um container.
- A declaração de um iterador de nome **it** para um vector é feita por:
 - **vector<int>::iterator it;**
- em que o operador **::** é usado para referir um elemento de uma classe (no caso, a classe **vect<int>**).

- e os métodos `begin()` e `end()` retornam iteradores apontando para o primeiro elemento do vector, e para a posição seguinte ao último elemento do vector. Assim, um uso comum para um iterador sobre um vector chamado `vec` seria:
`for (it = vec.begin(); it!=vec.end(); ++it)`
- E o acesso ao elemento apontado pelo iterador é feito com o operador `*`, como se fosse um pointer.

- O código abaixo cria uma lista de adjacências para o grafo à esquerda:

```
#include <vector>
// declara vetor de
// listas de adjacências
using namespace std;
vector <int> adj[5];
int main()
{
    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[2].push_back(3);
    adj[2].push_back(4);
}
```

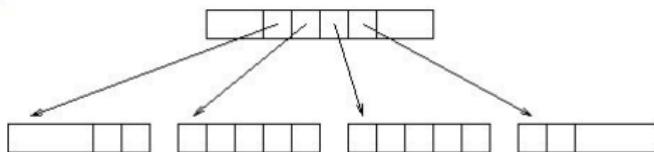


- E o código abaixo implementa uma DFS sobre o grafo:

```
int visit[5]={0};  
vector <int> adj[5];  
  
void dfs(int i)  
{  
    printf("%d\n", i);  
    visit[i]=1;  
    vector <int>::iterator it;  
    for (it=adj[i].begin(); it!=adj[i].end(); it++)  
        if (!visit[*it])  
            dfs(*it);  
}
```

Deque

- Operações de inserção no início ou no meio não são muito eficientes em um vector, já que todos os elementos após o ponto de inserção devem ser deslocados.
- Uma estrutura mais eficiente para inserção nas duas pontas é o deque (double-ended queue), que é implementado na forma a seguir.



- Essa estrutura de memória permite a inserção e remoção de elementos em qualquer posição sem precisar deslocar todos os posteriores.

list

- O tipo list implementa uma lista duplamente encadeada com as operações de inclusão e remoção nas duas pontas.
- Ao contrário do vector e deque, ela não permite acesso indexado a todos os elementos.
- A declaração de uma variável do tipo list é:
 - **list <tipo> nome;**
- Ex:
 - **list <int> lista1, lista2;**
- Para utilizar listas é necessário colocar `#include <list >`

- As principais operações sobre uma lista são:
 - `front()` - Retorna o valor do primeiro elemento da lista.
 - `back()` - Retorna o valor do último elemento da lista.
 - `push_front(g)` - Adiciona um novo elemento 'g' no início da lista.
 - `push_back(g)` - Adiciona um novo elemento 'g' no final da lista.
 - `pop_front()` - Remove e descarta o primeiro elemento da lista.
 - `pop_back()` - Remove e descarta o último elemento da lista

Queue (fila)

- O container **Queue** implementa uma fila, com as operações padrão de filas, de inserção e remoção em ambas as pontas.
- A queue é implementada sobre um deque ou uma list.
- A criação de uma fila é feita como qualquer outro container:
 - **queue <tipo> nome;**
- Não esquecer do `#include <queue>`

- Os principais métodos para trabalhar com queues são:
 - empty() - retorna true se a fila está vazia
 - push(valor) - insere elemento no final da fila
 - front() - retorna o primeiro elemento elemento na fila
 - back() - retorna o último elemento da fila
 - pop() - remove (e descarta) o próximo elemento da fila
- O código a seguir implementa uma BFS sobre o grafo do exemplo anterior, representado por um vetor de listas de adjacências:

```
void bfs(int i)
{
    int visit[5]={0};
    queue <int> fila;
    fila.push(i);
    visit[i]=1;
    while (!fila.empty())
    {
        int v=fila.front();
        printf("%d",v);
        fila.pop();
        vector <int>::iterator it;
        for (it=adj[v].begin(); it!=adj[v].end(); it++)
            if (!visit[*it])
            {
                visit[*it]=1;
                fila.push(*it);
            }
    } // do while
} // da função
```

Stack (pilha)

- O container **Stack** implementa uma pilha, com as operações padrão de pilhas de inserção e remoção no topo.
- Os principais métodos para manipulação de pilhas são:
 - `empty()` - retorna true se a fila está vazia
 - `push(val)` - empilha um valor
 - `top()` - retorna o valor no topo da pilha
 - `pop()` - remove (e descarta) o valor no topo da pilha
- O código a seguir implementa uma ordenação topológica no grafo, usando o método da DFS

```
#include <stdio.h>
#include <stack>
#include <vector>
using namespace std;
stack<int> pilha;
int visit[5]={0};
vector <int> adj[5];
void dfs(int i)
{
    visit[i]=1;
    vector <int>::iterator it;
    for (it=adj[i].begin();it!=adj[i].end();it++)
        if (!visit[*it])
            dfs(*it);
    pilha.push(i);
}
int main(){
    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[2].push_back(3);
    adj[2].push_back(4);
    for (int i=0;i<5;i++)
        if (!visit[i])
            dfs(i);
    while (!pilha.empty()){
        printf("%d\n", pilha.top());
        pilha.pop();
    }
}
```

Priority_queue (fila de prioridade)

- O container **priority_queue** implementa uma fila de prioridade. Essa estrutura se caracteriza por permitir identificar o maior (ou menor) elemento em tempo $O(1)$, removê-lo em tempo $O(\log n)$ e inserir novos elementos em tempo $O(\log n)$.
- É útil em algoritmos como o de Prim ou o de Dijkstra, em que um passo importante, e custoso, do algoritmo é a identificação, a cada passo do elemento de menor valor em um conjunto (no caso, a distância de cada vértice ao vértice inicial, no Dijkstra, ou à sub-árvore já formada, no Prim)
- Ele está definido no módulo queue (include queue)

- A versão default implementa uma fila de prioridade max-heap, devolvendo a cada vez o maior valor da fila.
 - `priority_queue <int> g;`
- Os principais métodos para manipulação de filas de prioridade são :
 - `empty()` - retorna true se a fila está vazia
 - `push(val)` - empilha um valor
 - `top()` - retorna o valor no topo da pilha
 - `pop()` - remove (e descarta) o valor no topo da pilha
- Para implementar um min-heap, em que o MENOR elemento é mantido no topo, usa-se a sintaxe enigmática abaixo:
 - `priority_queue < int, vector < int >, greater < int >> nome;`

- Containers podem conter tipos escalares, como int e float, mas também podem conter tipos estruturados, como structs.
- Quando o container deve armazenar pares de valores, pode-se utilizar o template pair $\langle \text{tipo1}, \text{tipo2} \rangle$ que agrupa dois valores como se fosse uma tupla.
- Ex: vector $\langle \text{pair} \langle \text{int}, \text{int} \rangle \rangle$ v;

- Na implementação do algoritmo de Dijkstra utilizando uma fila de prioridades para recuperar a cada passo o vértice mais próximo do vértice inicial, deve-se armazenar, na fila de prioridades, o número do vértice é a distância associada a ele, ou seja, cada posição da fila de prioridades conterá 2 valores, o número e o peso do vértice.
- Isso pode ser feito definindo um par de valores, pelo

Recursos online

- O site Data Structures Visualization (Botão) apresenta animações para uma variedade grande de algoritmos e estruturas de dados, incluindo árvores, ordenação, heaps, programação dinâmica, backtracking, grafos e outros.

Data Structure Visualizations

[About Algorithms](#)
[F.A.Q](#)
[Known Bugs / Feature Requests](#)
[Java Version](#)
[Flash Version](#)
[Create Your Own / Source Code](#)

Visualizing Algorithms

The best way to understand complex data structures is to see them in action. This visualization tool is written in javascript using the HTML5 canvas element, even the web browser in the Kindle! (The frame rate is low enough in the Kindle that Trees -- seem to work well enough)

Check the [Algorithms](#) menu for all of the latest javascript implementations.

- Especificamente de grafos, o site contem animações para:
 - Breadth-First Search
 - Depth-First Search
 - Connected Components
 - Dijkstra's Shortest Path
 - Árvore geradora mínima por Prim e Kruskal
 - Topological Sort (Using Indegree array)
 - Topological Sort (Using DFS)
 - Floyd-Warshall (all pairs shortest paths)

- O site VisuAlgo (Botão) apresenta também animações para uma variedade grande de algoritmos e estruturas de dados.
- Um diferencial é que ele mostra um pseudo-código do algoritmo durante a animação, destacando a linha em execução a cada passo.

The screenshot shows the VisuAlgo homepage. At the top, there's a navigation bar with a user icon, the text "VISUALGO.NET", and a language dropdown set to "en". Below this is a "Do You Know?" section containing a tip about cache loading for returning visitors. To the right of this is a "Next Random Tip" button. The main header "VISUALGO" is prominently displayed in large, bold, red letters, with ".NET/EN" in smaller text below it. A tagline "visualising data structures and algorithms through animation" is written in a smaller font. On the right side of the header is a search bar with the placeholder "Search...". Below the header, there's a "Featured story: Visualizing Algorithms with a Click" link. At the bottom of the page are various navigation icons for search and page navigation.

Lista de Problemas NP-Completos em Grafos

- Cobertura de vértices
- Caminho hamiltoniano
- Problema do caixeiro viajante (como problema de decisão)
- Isomorfismo de subgrafos
- Clique
- Conjunto independente máximo
- Conjunto dominante
- Coloração
- Lista mais completa ▶ Aqui