PI - Estrutura de Dados

Faculdade de Tecnologia SENAC Goiás
Segurança da informação
Estrutura de Dados
Prof. Marcelo Faustino

Alunos:
Cristhian Lopes de Souza
Matheus do Carmo
Matheus Oliveira
Pablo de Almeida

Goiânia,
06 de dezembro de 2016.

Sumário

1 Introdução:

Este trabalho faz parte do conjunto de estudos formadores do Projeto Integrador do terceiro módulo de Segurança da Informação, trecho referente a matéria de Estrutura de Dados.

Em sistemas de informática, lidamos com diversos tipos de dados, e o resultado do processamento desses dados, depende intrinsicamente de como esses dados são organizados, armazenados e estruturados. Nesse trabalho, mostraremos alguns exemplos básicos de estruturas de dados, recorrentes em diferentes linguagens de programação: Fila, lista, pilha, arvore e grafos. Também daremos alguns exemplos de aplicação dessas estruturas.

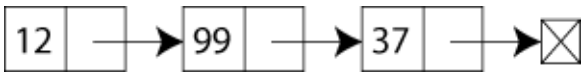
2 Estruturas de dados

2.1 Listas:

Listas são estruturas de dados que contém um conjunto de blocos de memória que armazenam dados. Esses blocos são encadeados (ligados) por ponteiros, formando uma espécie de “corrente”, onde as peças dessa corrente estão ligadas umas as outras. O encadeamento de listas pode ser de dois tipos: simplesmente encadeada e duplamente encadeada.

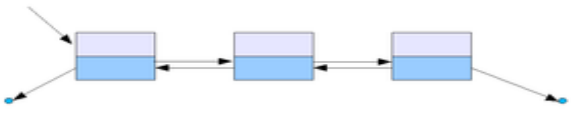
2.1.1 Lista simplesmente encadeada:

A lista simplesmente encadeada consiste de um ponteiro inicial que a principio aponta para NULL, sendo um objeto novo inserido, esse ponteiro passa a apostar para o primeiro objeto da lista, que por sua vez, possui um ponteiro para o próximo objeto, portanto, cada nó da lista possui sempre o ponteiro para o próximo objeto da lista e assim por diante.



2.1.2 Lista duplamente encadeada:

Já na lista duplamente encadeada, cada objeto possuiu não só o ponteiro para o próximo objeto (NULL no caso de ser o primeiro ou ultimo objeto), como também para o anterior (NULL no caso de ser o primeiro objeto), facilitando percorrer a lista em duas vias.



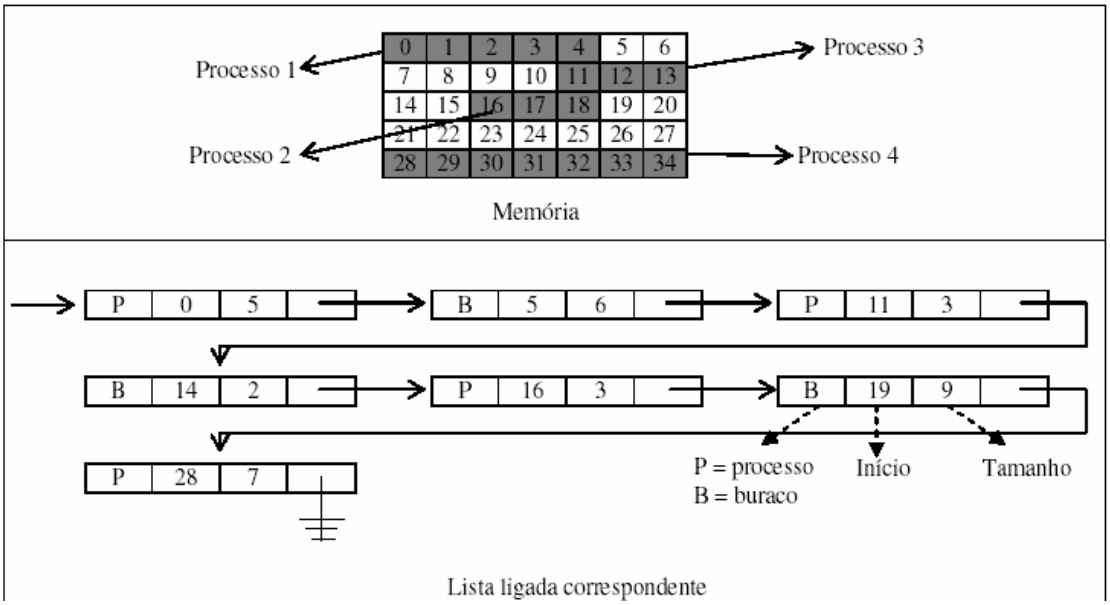
2.1.3 Lista circular

As listas circulares são como as listas duplamente encadeadas, porém, existe um vinculo entre o primeiro e o ultimo item da lista, portanto ponteiro do ultimo item sempre apostará para o primeiro e o ponteiro anterior do primeiro item apontará para o ultimo item da lista.

2.1.4 Usos de lista:

Se a aplicação precisa de armazenamento rápido, de um numero indefinido de elementos, a lista encadeada é a melhor opção, percorrer a lista é rápido e eficiente, pois como lidamos com ponteiros, nenhum item de memoria precisa ser movido, tal como o armazenamento tem tempo constante, pois o ponteiro está sempre apontado para a ultima posição livre, porém o tempo de busca continua sendo lento, pois os elementos são encadeados, portanto ainda é necessário que se percorra toda a lista. Alguns exemplos de uso de listas encadeadas:

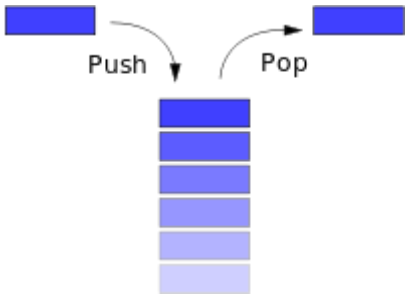
- Bancos de dados relacionais – Itens e chaves podem ser linkadas entre tabelas.
- Gerenciamento de disco – Em sistemas operacionais, listas ligadas são usadas para gerenciar espaços de memoria livre.



.

2.2 Pilha:

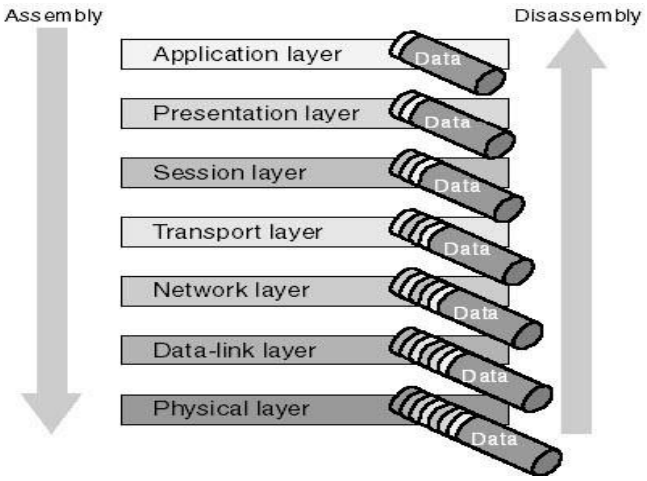
O funcionamento de uma pilha (ou *Stack*) consiste numa estratégia chamada LIFO (*last in, first out* – último a entrar, primeiro a sair). Dessa forma, o único elemento que se pode acessar na pilha é o elemento do topo da mesma, ou seja, o último a ser empilhado. Pense numa pilha de livros. Para se ter acesso ao ultimo livro, o de baixo, tem-se que mover antes todos os livros acima dele. É basicamente assim que esse tipo de estrutura funciona.



2.2.1 Usos de Pilha

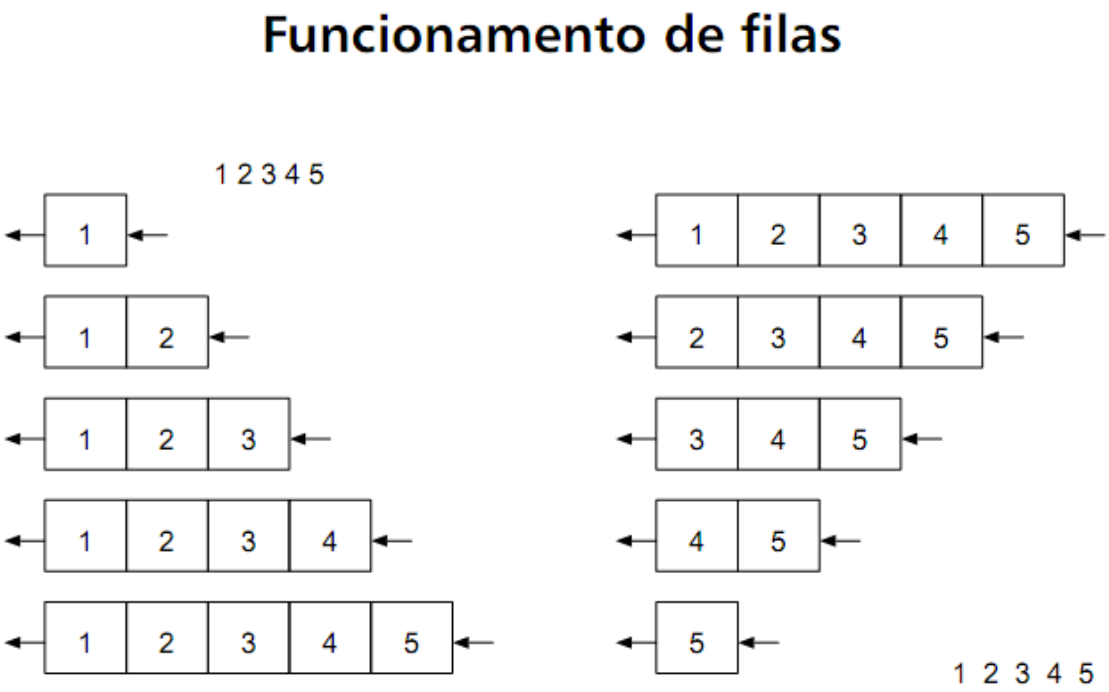
O conceito de pilha, embora simples, ainda é amplamente usado em diversas áreas da informática, alguns exemplos são:

- Editores de material em geral – A função “desfazer” por exemplo, armazena em uma pilha as condições anteriores do arquivo, e as recupera sempre pela ultima armazenada, como uma pilha LIFO(Last In First Out)
- Protocolo TCP/IP- O encapsulamento de pacotes, quadros e dados do sistema TCP funciona como um método de pilha, no envio o pacote é encapsulado varias vezes de forma sobreposta, e no recebimento esse mesmo dado deve ser desencapsulado na ordem inversa com que foi encapsulado.



2.3 Fila:

O Diferente da pilha, onde os dados só podem ser retirados ou inseridos de um único lado da pilha, a fila trabalha no método FIFO (First in, First out – primeiro a entrar, primeiro a sair). Funcionando exatamente como as filas de banco, onde o primeiro a chegar se posiciona no começo da fila, e os próximos sem sequencia atrás deste, e o caixa irá chama-los por ordem de chegada. A implementação pode realizar-se com ajuda de vetores, assim como através do uso de ponteiros. Se a fila é implementada com o uso de vetores, o número máximo de elementos armazenados deve ser estabelecido no código do programa antes da compilação (fila estática) ou durante sua execução (fila pseudo-estática).



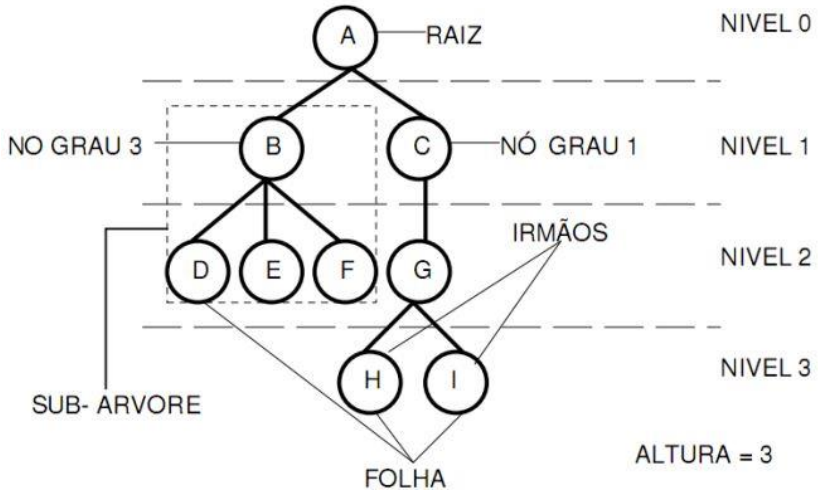
2.3.1 Usos de Fila:

Se a aplicação precisa de inserção rápida de dados, fila é a melhor solução, porem se os dados forem ordenados, a busca pode ser mais lenta, e por isso a remoção de itens específicos também pode ser mais lenta, mas se o numero de elementos for previamente conhecido, essa estrutura pode ser a solução mais pratica. Alguns exemplos de uso de pilha:

- Escalonamento de processos – Em sistemas operacionais, processos a espera de recursos, são colocados em filas de espera, e processos a prontos para serem processados são também organizados com o uso de filas, como no escalonamento FCFS.
- Servidores - Requisições feitas em servidores web são previamente armazenados em filas de espera, no caso de a demanda ser maior do que a quantidade de pedidos

2.4 Árvore:

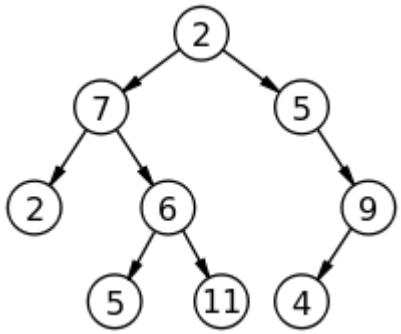
A árvore é uma das estruturas mais avançadas, diferentemente das filas e pilhas que tem em comum uma forma de armazenamento linear ou circular, a árvore, como o nome pressupõe, armazena os dados de forma hierárquica, como os galhos de uma árvore e suas folhas. Os galhos em uma árvore são arestas, e as folhas, são os nós.



Uma raiz principal no nível 0 leva a seus nós filhos, que formarão o próximo nível da arvore, e cada um desses nós podem por sua vez ter nós filhos, nós com uma raiz em comum são chamados de nós irmãos e serão parte do próximo nível, e assim por diante. Nós sem filhos são chamados de folhas.

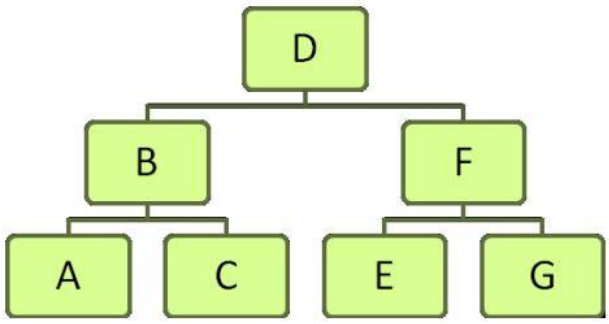
2.4.1 Árvores binárias:

Árvores binarias são as formas mais comuns de arvore encontrada em algoritmos, é comumente usada como arvore de busca, sua estrutura consiste de um nó raiz e um máximo de dois nós filhos (por isso o nome “binária”), a raiz e seus nos deverão conter dois ponteiros, um apontando para o nó da esquerda e um para o nó da direita.



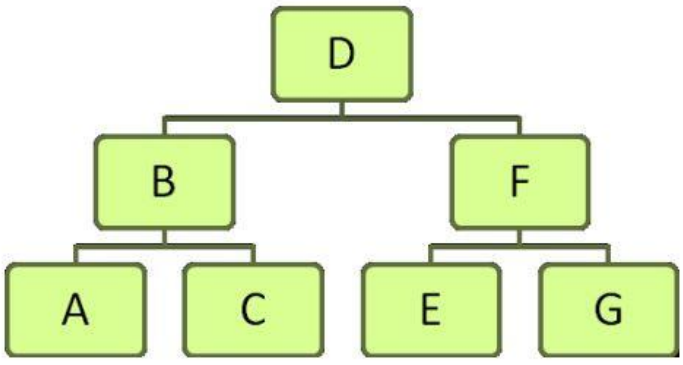
As arvores trazem benefícios em inserções, remoções e buscas, por ter uma estrutura pré definida, diferentemente das listas, em que todos os nós devem ser percorridos todos os momentos, a estrutura hierárquica da arvore, permite um acesso mais rápido saltando entre nós, Veremos agora algumas formas de definição dessa hierarquia.

Pré-Ordem: é tratada a raiz, em seguida se percorre o braço esquerdo, e por fim o braço direito, como segue:



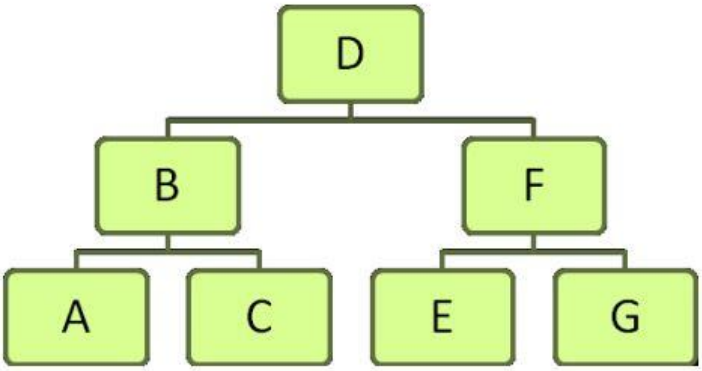
***Resultado**: D, B, A, C, F, E e G*

Em-Ordem: é primeiramente percorrido o braço esquerdo, em seguida pe tratada a raiz, e por fim o braço direito como segue:



***Resultado**: A, B, C, D, E, F e G.*

Pós-Ordem: é percorrido o braço esquerdo, seguido do braço direito e apenas por ultimo é tratada a raiz, como segue:

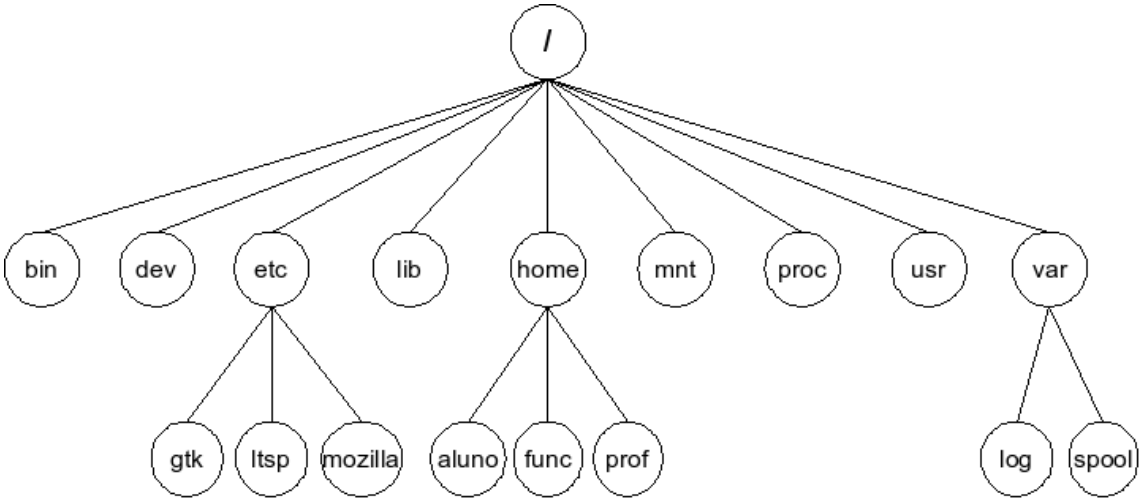


Resultado: *A, C, B, E, G, F e D.*

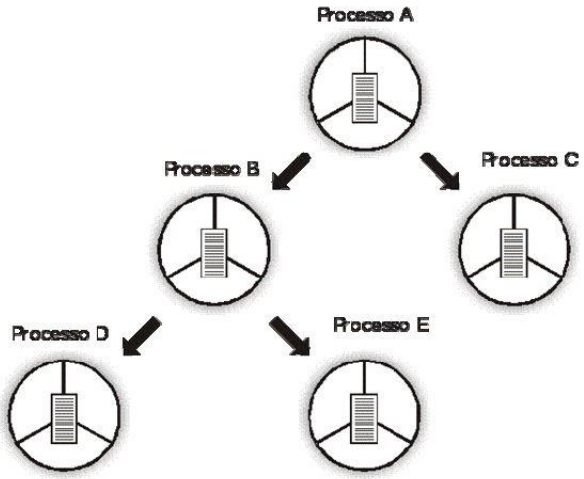
2.4.2 Usos de Arvores:

Arvores tem os tempos de acesso mais eficientes em quase todos os quesitos, inserção, remoção, busca, são uma solução muito procurada para armazenamento de dados ordenados, porem o tempo de busca pode ser afetado se a arvore se tornar desbalanceada, isso exige cuidado constante na forma de inserção e manutenção da mesma. Alguns exemplos do uso de arvores são:

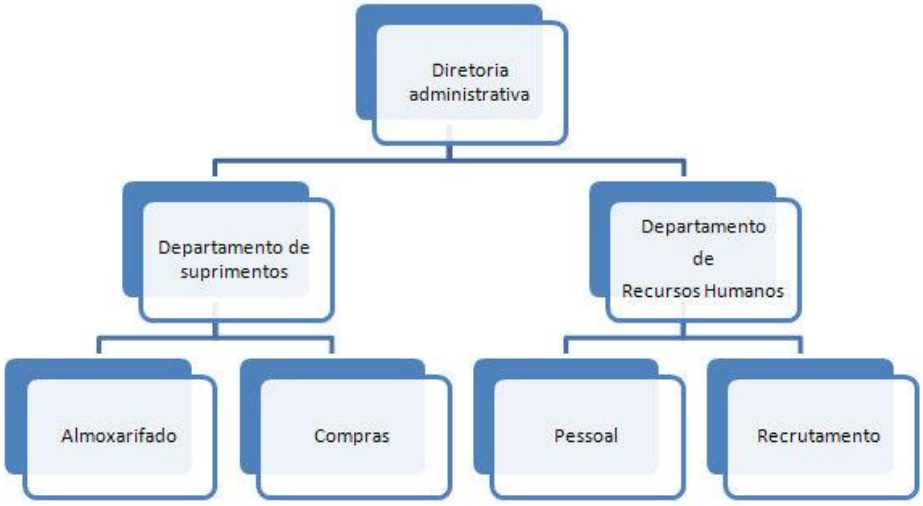
- Sistemas Operacionais – A hierarquia de pastas em sistemas operacionais é gerenciada com o uso de arvores



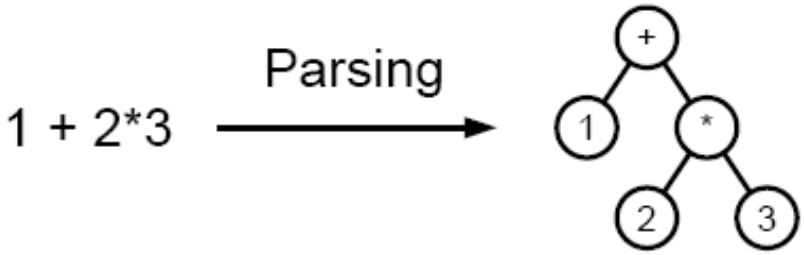
- Processos – O sistema de subprocessos em sistemas multi thread também é gerido com o uso de arvore



- Organogramas – Sistemas organizacionais, tanto empresariais, como acadêmicos ou pessoais, são feitos em forma de arvore



- Matemática – A Analise sintática (parsing) em equações e mesmo em compiladores, é feita em forma de arvore

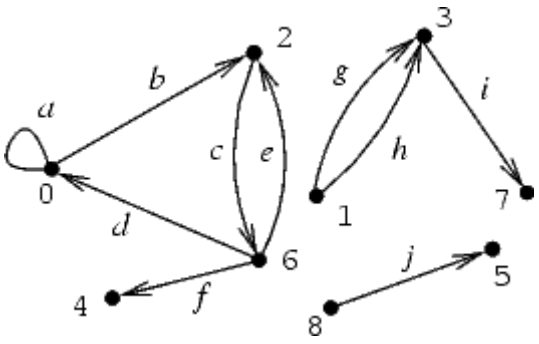


2.5 Grafos:

Enquanto nas arvores, existe uma raiz que leva a nos subsequentes, de forma acíclica, onde só existe um caminho distinto entre dois nós, em grafos, a estrutura se torna complexa e surgem laços e ligações múltiplas entre nós. Um grafo é uma estrutura formada por dois conjuntos: um conjunto de coisas chamadas vértices e um conjunto de coisas chamadas arcos; cada arco está associado a dois vértices: o primeiro é a ponta inicial do arco e o segundo é ponta final.

EXEMPLO 1: os vértices de nosso grafo são 0, 1, 2, 3, 4, 5, 6, 7, 8 e os arcos são a, b, c, d, e, f, g, h, i, j. Então a seguinte tabela define um grafo:

ponta inicial	0	0	2	6	6	6	1	1	3	8
arco	a	b	c	d	e	f	g	h	i	j
ponta final	0	2	6	0	2	4	3	3	7	5



Se um arco a tem ponta inicial v e ponta final w, dizemos que vai de v a w. Dizemos também que a sai de v e entra em w. Às vezes, um arco com ponta inicial v e ponta final w será denotado por (v,w) ou por v—w ou ainda por vw. Como se vê, os arcos são dirigidos; há quem goste de enfatizar esse fato dizendo que o grafo é dirigido (= directed).

De maneira mais formal, podemos dizer que um grafo é um terno (V, A, f), onde V e A são conjuntos finitos arbitrários e f é a função que associa a cada elemento de A um par ordenado de elementos de V. Às vezes, a função f é subentendida e dizemos simplesmente o grafo (V, A). Se o grafo como um todo é denotado por G, o seu conjunto de vértices será denotado por VG e o seu conjunto de arcos por AG. Se o grafo for denotado por g, os conjuntos de vértices e arcos serão denotados por Vg e Ag respectivamente.

2.5.1 Laços e arcos paralelos

Um arco é um laço (= loop = self-loop) se sua ponta inicial coincide com sua ponta final, ou seja, se o arco é da forma (v,v). Dois arcos são paralelos se têm a mesma ponta inicial e a mesma ponta final, ou seja, se os dois arcos são da forma (v,w). A

propósito, dois arcos são antiparalelos se a ponta inicial de um é ponta final do outro, ou seja, se um arco é da forma (v,w) enquanto o outro é da forma (w,v).

Um grafo é simétrico (= symmetric) se para cada arco da forma (v,w) existe um arco da forma (w,v).

- **Grafo simples** é um grafo não direcionado, sem laços e existe no máximo uma aresta entre quaisquer dois vértices (sem arestas paralelas). Para um grafo simples, o número de vizinhos de um vértice é igual à sua valência.
- **Grafo completo** é o grafo simples em que, para cada vértice do grafo, existe uma aresta conectando este vértice a cada um dos demais. Ou seja, todos os vértices do grafo possuem mesmo grau. O grafo completo de *n* vértices é frequentemente denotado por K_n. Ele tem $n(n-1)/2$ arestas (correspondendo a todas as possíveis escolhas de pares de vértices).
- **Grafo nulo** é o grafo cujo conjunto de vértices é vazio.
- **Grafo vazio** é o grafo cujo conjunto de arestas é vazio.
- **Grafo trivial** é o grafo que possui apenas um vértice e nenhuma aresta.
- **Grafo regular** é um grafo em que todos os vértices tem o mesmo grau.
- **Multigrafo** é um grafo que permite múltiplas arestas ligando os mesmos vértices (arestas paralelas).
- **Pseudografo** é um grafo que contém arestas paralelas e laços.
- **Grafo conexo** um grafo é *conexo* se for possível estabelecer um caminho de qualquer vértice para qualquer outro vértice de um grafo. Se for sempre possível estabelecer um caminho de qualquer vértice para qualquer outro vértice mesmo depois de remover *k*-1 vértices, então diz-se que o grafo está ***k*-conexo**. Note que um grafo está *k*-conexo se, e somente se, contém *k* caminhos independentes entre qualquer par de vértices. O grafo de exemplo acima é conexo (e portanto 1-conexo), mas não é 2-conexo. Em um grafo genérico *G*, o **corte** associado a um conjunto *X* de vértices é o conjunto de todas as arestas que têm uma ponta em *X* e outra em *V(G) - X*, onde *V(G)*é o conjunto de todos os vértices pertencentes ao grafo *G*.
- **Ponto de articulação** ou **Vértice de corte** é um vértice cuja remoção desliga um grafo. Uma ponte é uma aresta cuja remoção desliga um grafo. Um componente biconectado é um conjunto máximo de arestas tal que qualquer par de arestas do conjunto fazem parte de um ciclo simples comum. O contorno de um grafo é o comprimento do ciclo simples mais curto no grafo. O contorno de um grafo acíclico é, por definição, infinito.
- **Árvore** é um grafo simples acíclico e conexo. Às vezes, um vértice da árvore é distinto e chamado de *raiz*. As árvores são muito usadas como estruturas de dados em informática.
- **Floresta** é um conjunto de árvores; equivalentemente a uma floresta, em algum grafo acíclico.
- **Subgrafo** de um grafo *G* é um grafo cujo conjunto dos vértices é um subconjunto do conjunto de vértices *G*, cujo conjunto de arestas é um subconjunto do conjunto de arestas de *G*, e cuja função *w* é uma restrição da função de *G*
- **Subgrafo gerador** é aquele obtido pela remoção de uma ou mais arestas de um outro grafo, dizemos então que este novo grafo obtido é gerador do primeiro,
- **Subgrafo induzido** é obtido pela remoção de vértices e consequente das arestas relacionadas com ele de um outro grafo, dizemos que este novo grafo é um grafo induzido do original.
- **Grafo parcial** de um grafo *G* é um subgrafo com o mesmo conjunto de vértices que *G*. Uma **árvore parcial** é um grafo parcial que é árvore. Todo grafo tem pelo menos uma árvore parcial.

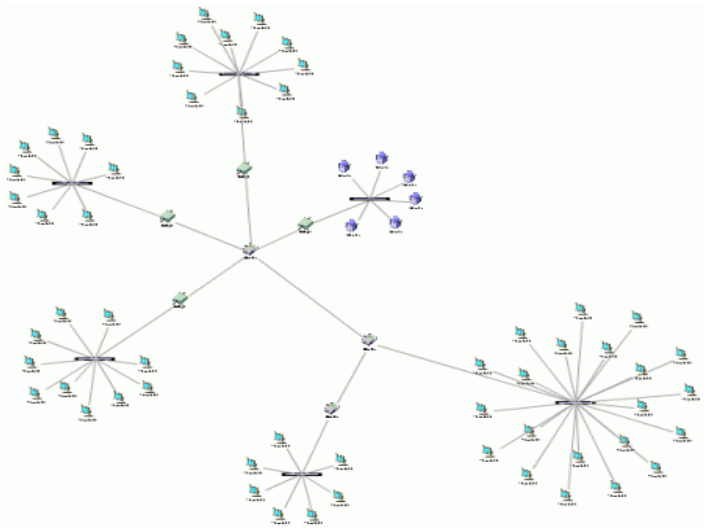
- **Clique** em um grafo é um subgrafo que também é um grafo completo. No grafo do exemplo acima, os vértices 1, 2 e 5 formam um clique.
- **Conjunto independente** em um grafo é um conjunto de vértices não adjacentes entre si. No exemplo acima, os vértices 1, 3 e 6 formam um conjunto independente e 3, 5 e 6 são outro conjunto independente.
- **Grafo planar** é aquele que pode ser representado em um plano sem qualquer intersecção entre arestas. O grafo do exemplo é planar; o grafo completo de *n* vértices, para *n*> 4, não é planar.
- **Grafo bipartido** é o grafo cujos vértices podem ser divididos em dois conjuntos, nos quais não há arestas entre vértices de um mesmo conjunto. Para um grafo ser bipartido ele não pode conter circuitos de comprimento ímpar.
    1. **Se um grafo G é bipartido, todo o circuito de G possui comprimento par.**Sejam V1 e V2 os dois conjuntos em que, de acordo com a definição de grafo bipartido, se particiona V(G). Toda a aresta de G conecta um vértice em V1 com outro em V2. Assim sendo, se X for um vértice de V1, para "voltar" a esse vértice terá de se ir a V2 e voltar a V1 um número indeterminado de vezes, e de cada vez serão percorridas duas arestas, uma de um vértice em V1 para um vértice em V2 e outra de um vértice em V2 para um vértice em V1. Logo, o número de arestas a percorrer será par, ou seja, o comprimento do circuito é par.
    2. **Se todo o circuito de um grafo G possui comprimento par, então o grafo é bipartido.**Seja G um grafo em que todo o circuito tem comprimento par, e seja X um vértice de G. Denotemos por V1 o conjunto formado por X e por todos os vértices cuja distância a X é par. Seja V2 = V(G)\V1 (isto é, o conjunto formado pelos vértices de G que não pertencem a V1). Pretende mostrar-se que não existe qualquer aresta que conecte vértices de V1 ou vértices de V2. Suponhamos a existência de tal aresta, isto é, suponhamos a existência de dois vértices em V1 (ou V2), digamos Xi e Xj, conectados por uma aresta. Ora existe já um caminho de comprimento par entre Xi e Xj, já que existem caminhos, ambos de comprimento par (ou ímpar, no caso de Xi e Xj pertencerem a V2), entre Xi e X e entre X e Xj. Se a esse caminho juntarmos a aresta {Xi;Xj} obtemos um circuito de comprimento ímpar o que contraria a hipótese de apenas existirem circuitos de comprimento par.
- **Grafo bipartido completo** é o grafo bipartido, cujo qualquer vértice do primeiro conjunto é adjacente a todos vértices do segundo conjunto
- **Grafo k-partido** ou **grafo de *k*-coloração** é um grafo cujos vértices podem ser particionados em *k* conjuntos disjuntos, nos quais não há arestas entre vértices de um mesmo conjunto. Um grafo 2-partido é o mesmo que grafo bipartido.
- **Emparelhamento de grafos** consiste em partir o grafo em conjuntos de vértices a qual não compartilham nenhuma aresta entre eles.

2.5.2 Usos de grafos:

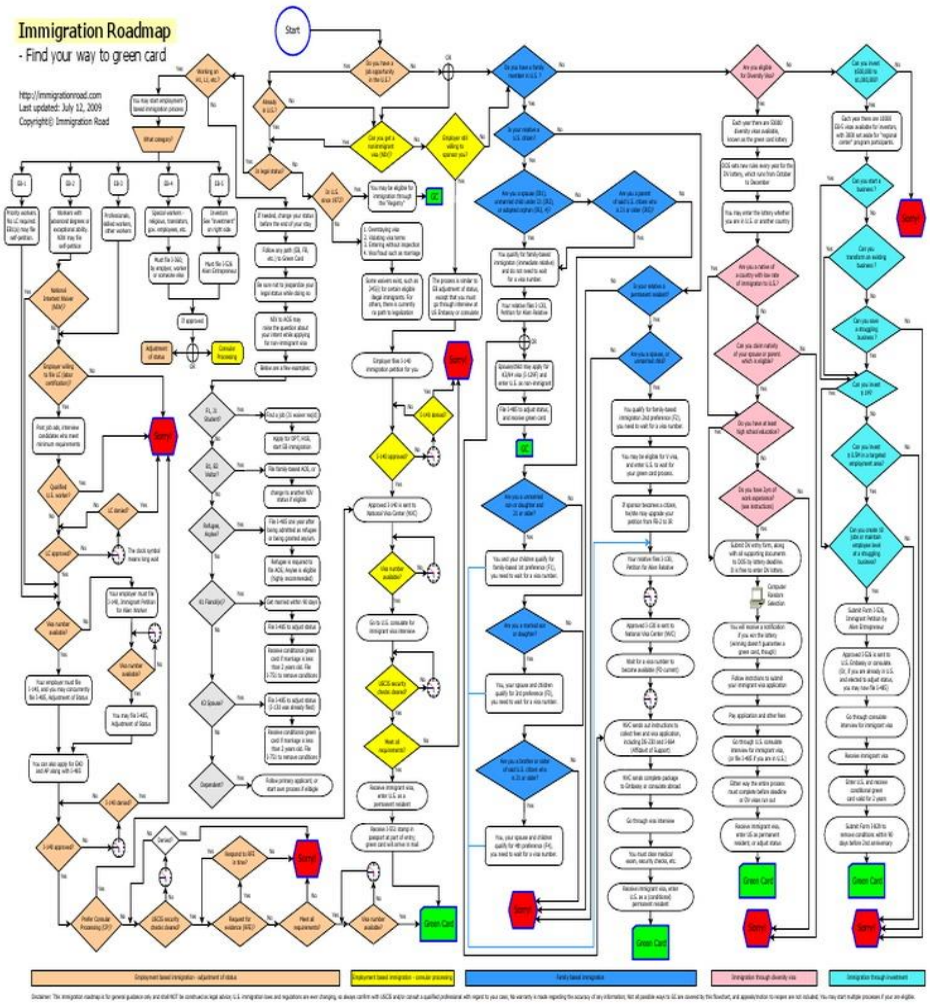
Grafos podem ser visto como a exposição de dados em um plano matemático, é uma forma de armazenar dados onde pode haver ligações complexas entre os dados, possibilitando operações avançadas que não eram possíveis com as estruturas estudadas até então. Um exemplo de aplicação é a instalação de fibras ópticas no campus de uma universidade. Cada trecho de fibra óptica entre os prédios possui um custo associado. Um grafo fornece todas as conexões possíveis entre os prédios. Uma árvore geradora fornece um modo de conectar todos os prédios sem redundância.

Uma árvore geradora mínima desse grafo nos daria uma árvore com o menor custo para fazer essa ligação.
Alguns outros exemplos da aplicação de grafos são:

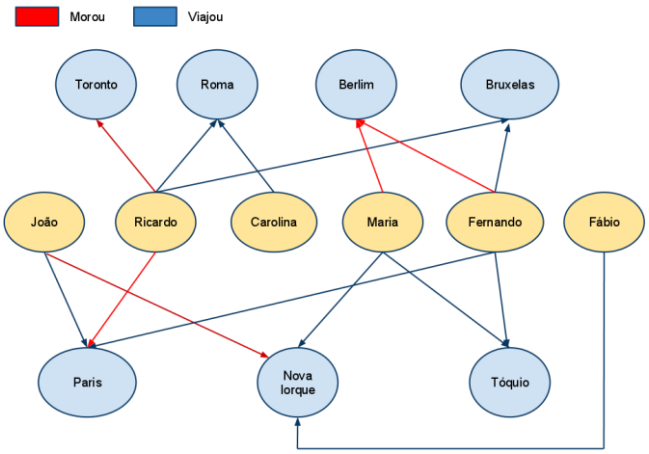
- Roteamento – Sistemas de telefonia, redes e internet WAN.



- Flowcharts – Graficos, diagramas e estudos de cenários.



- Inteligência Artificial - Redes Neurais e tomadas de decisão.
- Bancos de Dados – bancos relacionais podem ser modelados como grafos.



- Data-Search - Ranking de páginas do Google, ou orientação do Google Maps.
- Redes Sociais – Relações de preferencia, uso, interesse, dispersão.
- Circuitos Elétricos – Impressão de circuitos avançados.
- Sistemas Operacionais - Analise de estabilidade em aplicações multi-tarefa.

3 Referências:

<https://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/>

<http://www.cprogressivo.net/2013/10/Estrutura-de-dados-dinamica-em-C-Listas-Filas-Pilhas-Arvores.html>

<http://forum.clubedohardware.com.br/topic/646067-estruturas-de-dados-pilhas-filas-listas-encadeadas/>

<http://www.guj.com.br/t/filas-pilhas-e-listas/>

<http://stackoverflow.com/>