ÁRVORES

Conceito:

- estrutura NÃO linear
- armazena elementos de maneira hierárquica
- conjunto finito de um ou mais nodos
    - raiz (root)
    - sub árvore (subtree) – parent (pai), filho (child)
- floresta: conjunto de uma ou mais árvores disjuntas

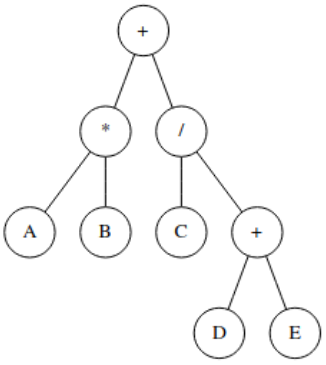
Relacionamento entre nodos:

- Nodo **Externo** (folha): se não tem filhos
- Nodo **Interno** (galho): um ou mais filhos
- **grau** (degree): número de filhos de um nodo
- **nível** (depth): número de linhas que liga o nodo a raiz (raiz é o nível zero)
- **altura** (height): nível mais alto da árvore
- a raiz de uma árvore é chamada de pai de suas sub árvores
- variáveis: info, father, child1, child2…

Aplicações:

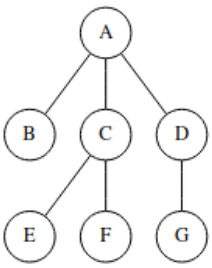
- expressões aritiméticas

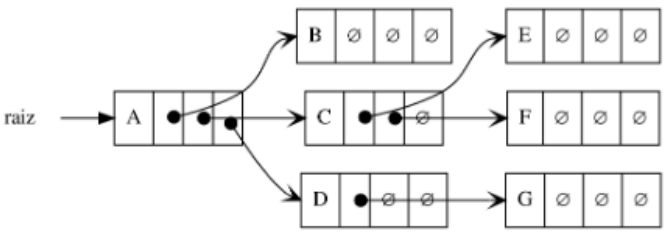
Exemplo: `A * B + C / ( D + E )`



Representação na memória:

- **contiguidade** (arranjo)
- **encadeamento**





- **construtor**: inicializa os nodos = vazio
- **destrutor**: verificar se os nodos != vazio, para então esvaziar
- **void clean**: recursiva e similar ao destrutor

*e = new Node('E'); // criar nodos **folha**

*b = new Node('B',e); // criar nodos **intermediários**

*root = new Node ('A', b, e); // criar nodo **raiz**

TAD

MÉTODOS GENÉRICOS

- contains()
- se o dado do nodo for = i não segue a execução
- percorre a quantidade de nodos-filho
- verifica se o filho atual está apontado para o método(i)
- find()
- se o dado do nodo for = i não segue a execução
- percorre a quantidade de nodos-filho
- referência a classe constante filho = filho atual apontado para método(i)
- verificar se filho não está vazio para retornar filho

MÉTODOS DE CONSULTA

- isExternal() // se a quantidade de filhos está vazia
- isInternal() // verificar se pai != vazio e quantidade de filhos != zero
- isRoot() // se pai estiver vazio

ÁRVORE BINÁRIA

Conceitos:

- **grau** de cada nodo é menor/igual 2
- sub árvore esquerda (prioridade)
- sub árvore direita
- **própria** se cada um de seus nodos internos tiver dois filhos
- máximo 2 filhos por nodo
- referenciar os campos pai dos filhos apontaram para nodo que está inicializando usa-se a autorreferência *this*
- variáveis: info, right,left,parent

TAD

MÉTODOS GENÉRICOS

- degree()
- verificar se subtree -> left/right != vazio
- incrementar filho em cada verificação
- depth()
- criar nodo auxiliar apontando a sub árvore para pai
- loop while caso o auxiliar não esteja vazio
- incrementar nível
- auxiliar = ele mesmo apontando para pai
- size()
- verificar se a sub árvore está vazia
- recursivo: retornar 1 + somatória de cada sub árvore apontada para left/right
- height()
- verificar se a sub árvore está vazia
- criar variável nova esq/dir
- verificar (if) se a sub árvore apontada left/right está vazia,caso contrário 1 + recursiva(sub árvore apontada left/right)
- caso esq > dir priorizar o lado esq

ÁRVORE GENÉRICA

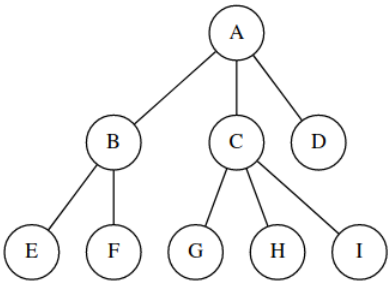
- usar <vector> pois o nodo pode ter número variável de filhos

TAD

- addSubtree()
- verificar se a sub árvore está vazia
- insercao do filho (push_back)
- sub árvore apontada para pai = this
- removeSubtree()
- percorrer a quantidade de nodos-filho
- verifica se o filho atual combina com a sub árvore fornecida
- remoção do filho (erase) percorrendo (filho.begin() + i)

CAMINHAMENTO - ordem de visita dos nodos

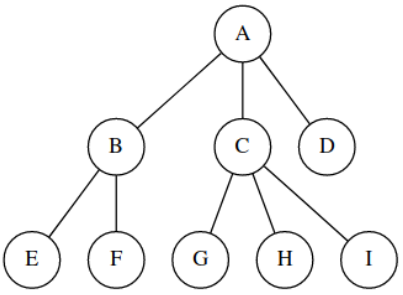
- preorder()



1. A
2. B
3. E
4. F
5. C
6. G
7. H
8. I
9. D

- imprime info
- verifica se left/right nao esta vazio para imprimir left/right apontada para método

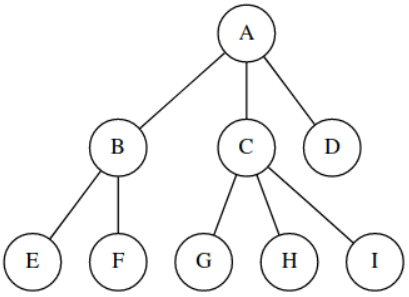
- posorder()



1. E
2. F
3. B
4. G
5. H
6. I
7. C
8. D
9. A

- verifica se left/right nao esta vazio para imprimir left/right apontada para método
- imprime info

- levelorder()

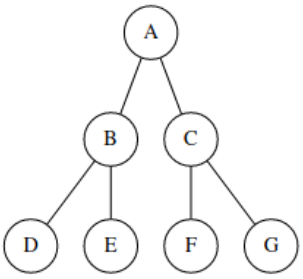


1. A
2. B
3. C
4. D
5. E
6. F
7. G
8. H
9. I

- cria uma variável = height(raiz)
- percorre a variável
- método(raiz, i)

*CAMINHAMENTO DA ÁRVORE BINÁRIA GENÉRICA

- inorder()



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

- verifica se left nao esta vazio para imprimir left apontada para método
- imprime info
- verifica se right nao esta vazio para imprimir right apontada para método

AVL

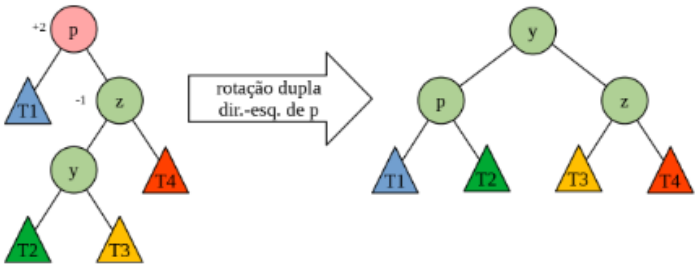
- elementos a **esquerda menores** que a raiz
- elementos a **direita maiores** que a raiz
- distribuição uniforme por meio da altura

- BALANCEAMENTO
- fator: FB(nodo) = altura(nodo -> dir) - altura(nodo -> esq)
- fator precisa ser **-1, 0 ou +1**
- O(n) não balanceada
- O(log n) balanceada

- OPERAÇÕES
- rotação simples direita **(sinal igual e negativo)**
    + FB = −2 e filho com FB = −1 ou FB = 0

- rotação simples esquerda **(sinal igual e positivo)**
    + FB = 2 e filho com FB = 1 ou FB = 0

- rotação dupla direita - esquerda **(sinal contrário com pai positivo e filho negativo)**
    + FB = 2 e filho com FB = -1



- rotação dupla esquerda - direita **(sinal contrário com pai negativo e filho positivo)**
    + FB = −2 e filho com FB = 1