

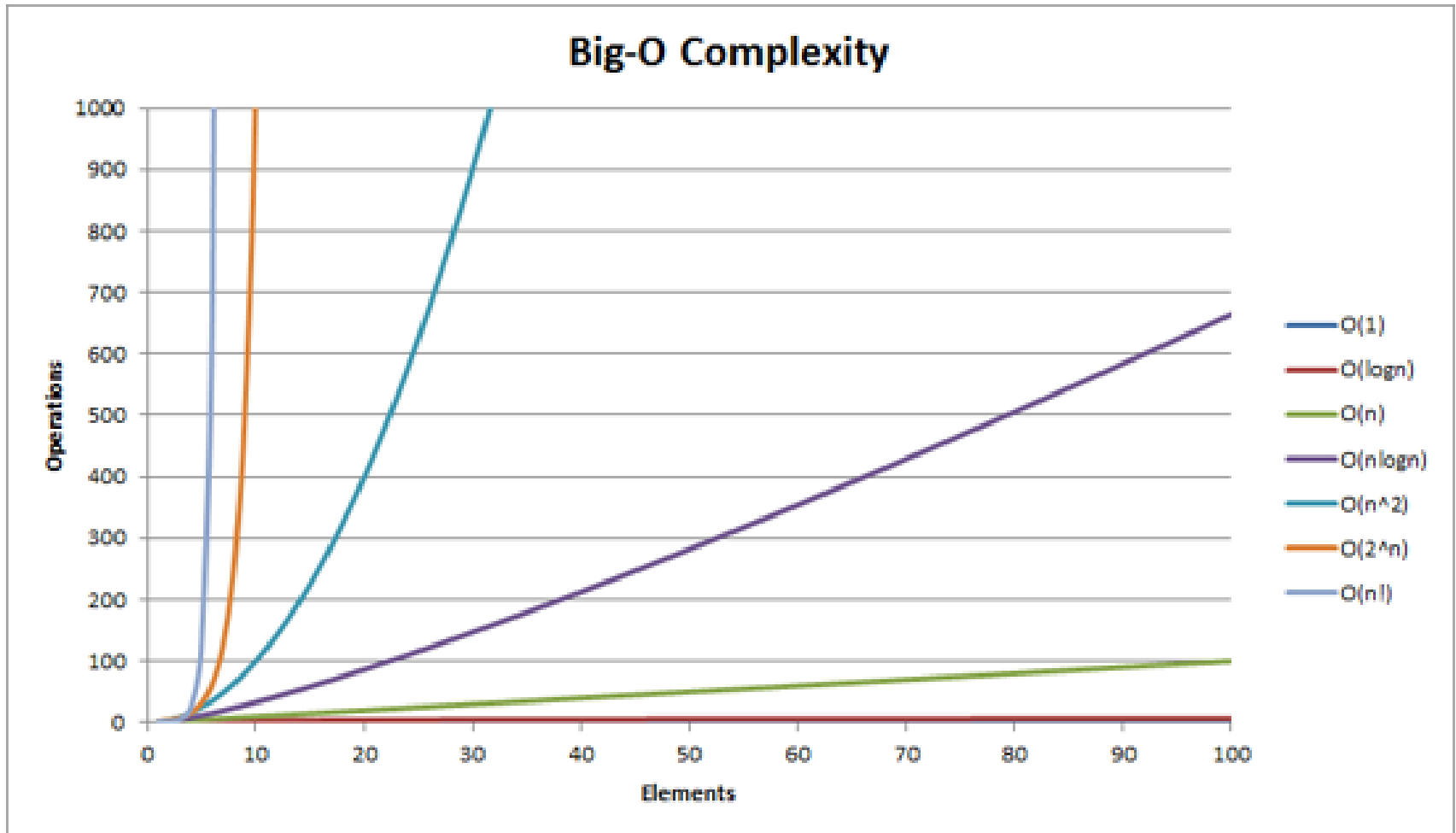
STL – Standard Template Library

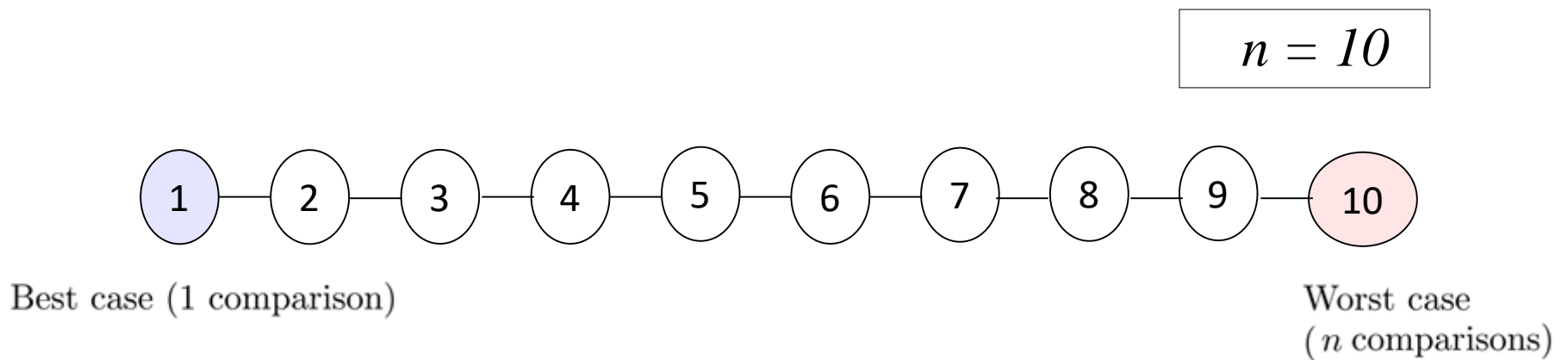
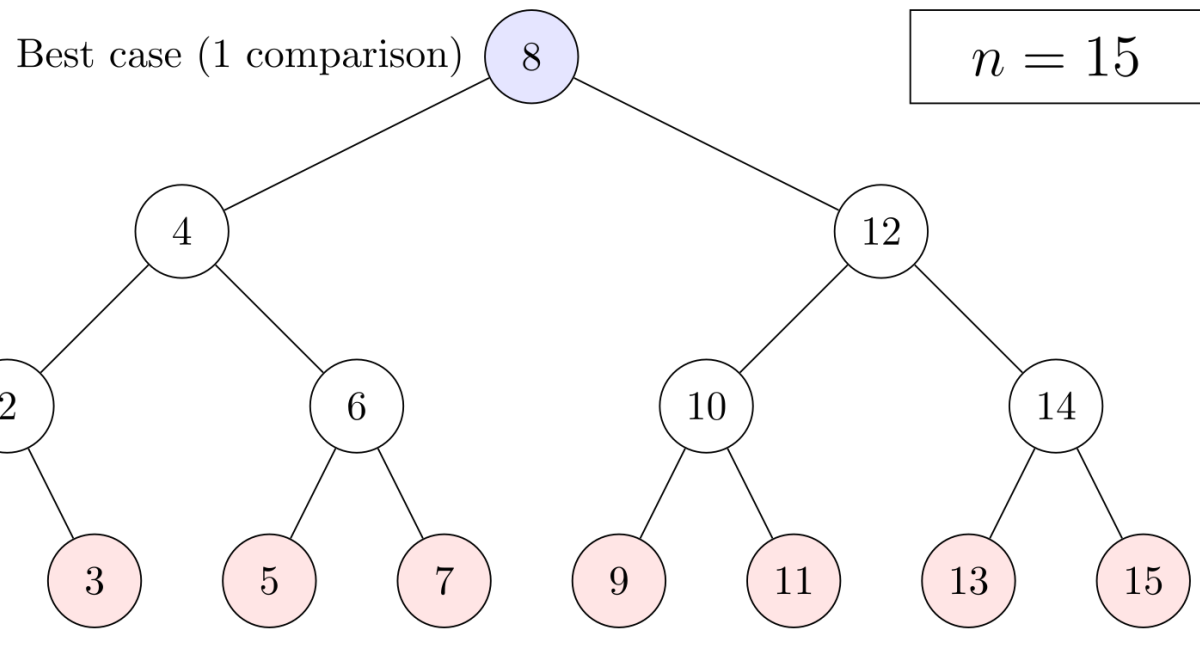
Prof. João Paulo R. R. Leite

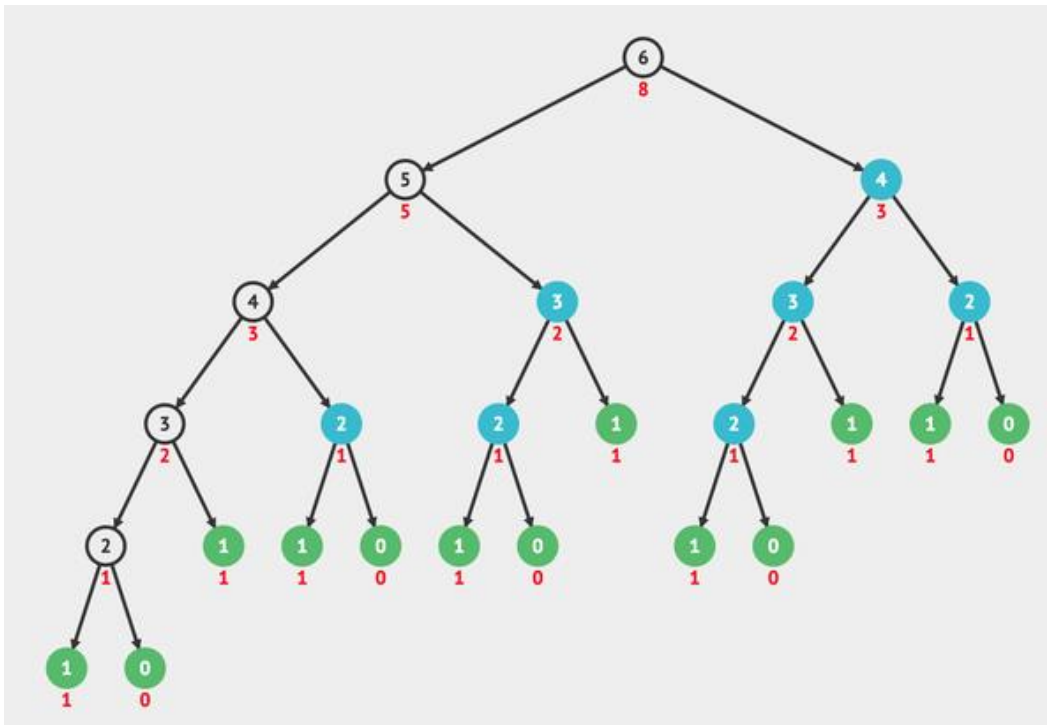
joaopaulo@unifei.edu.br

ECOE44 & ECOE45 – Maratona de Programação

Alguns Tópicos Importantes...







Fibonacci Recursivo:

Forma uma árvore de altura $n-1$, aproximadamente cheia. Portanto, são cerca de **$O(2^n)$** operações a serem realizadas.

0	1	2	3	4	5	6
0	1	1	2	3	5	8


Fibonacci Iterativo:

Forma vetor de comprimento igual a n . Portanto, são cerca de **$O(n)$** operações a serem realizadas.

Alguns Tópicos Importantes...

Um erro comum...

```
// Código meramente ilustrativo - Onde está o erro?  
for(int i = 0; i < m.size(); i++) {  
    int count = 0;  
    if(a == 0) count++;  
  
    for(int i = 0; i < b; i++) {  
        m[i] = b + 10;  
    }  
}
```



Saída de Caracteres

O operador << (“colocar em”) é utilizado como operador de saída, sendo que `cout` é o stream de saída padrão e `cerr` é o stream padrão para relato de erros. Podemos escrever:

```
void f()
{
    cout << 10; // coloque o literal 10 no stream padrão de saída

    int i = 10;
    cout << i; // coloque o conteúdo da variável i na saída

    // caso ocorra algum erro, podemos escrever no stream de erros
    cerr << "Ocorreu um erro no seu programa";
}
```

É possível ainda, combinar uma sentença inteira em uma única linha:

```
int idade {30};
cout << "O professor tem " << idade << " anos\n";
```

```
O professor tem 30 anos
```

Entrada de Caracteres

O operador `>>` (“coletar de”) é utilizado como operador de entrada, sendo que `cin` é o stream de entrada padrão. Por exemplo, podemos escrever:

```
void g()
{
    // entrando com um unico numero
    cout << "Entre com um inteiro: ";
    int i = 0;
    cin >> i;

    // entrando com dois numeros, separados por espaço ou enter
    cout << "Entre com dois doubles: ";
    double a = 0.0, b = 0.0;
    cin >> a >> b;
}
```

Com frequência, precisaremos ler valores textuais. Para isso, podemos utilizar o tipo `string`, do C++.

```
#include <string>

void h()
{
    cout << "Entre com seu nome: ";
    string str;
    cin >> str;
    cout << "Oi, " << str << "!\n";
}
```

Dicas:

Iremos **sempre** preferir a implementação de `string` da biblioteca padrão de C++ à implementação em **vetor de char de C**.

Por padrão, um caractere de espaço em branco termina a leitura da entrada. Portanto, caso o nome seja “Joao Paulo” (belo nome), somente “Joao” seria lido. Caso queira ler uma linha inteira de texto (incluindo o caractere de terminação de linha), há outra função que deve ser utilizada: `getline()`.

```
void h()
{
    cout << "Entre com seu nome: ";
    string str;
    getline(cin, str); // argumentos: stream de saída cin e a string a armazenar valor
    cout << "Oi, " << str << "!\n";
}
```

```
Entre com seu nome: Joao Paulo
Oi, Joao Paulo!
```

Em um exemplo anterior, foi utilizado o comando `std::endl` ao final da linha da seguinte maneira: `cout << "Voce parou..." << endl;`

Dicas:

Este comando também é adequado para terminar a linha. No entanto, além de acrescentar o final da linha, ele também realiza um `std::flush` na saída. Essa operação custa tempo. Portanto, sempre prefira `'\n'` a `endl`.

Processamento de strings

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout << "Entre com seu nome: ";

    string nome;
    cin >> nome;

    cout << "Agora entre com seu sobrenome: ";
    string sobre;
    cin >> sobre;

    // concatenando no novo
    string nome_completo = nome + " " + sobre;
    cout << "Nome completo: " << nome_completo << "\n";

    // concatenando no primeiro
    nome += " ";
    nome += sobre;
    cout << "Nome completo: " << nome << "\n";

    // comparando com um literal
    if(nome == "Joao Paulo")
        cout << "Ta tranquilo. Ta favoravel. \n";
}
```

Repare como é muito mais simples do que o que fazíamos em C. Atribuição pode ser realizada diretamente, utilizando o operador =, e comparações de igualdade podem ser feitas com ==.

Não é necessário utilizar strcmp, strcpy, etc.

Dicas: A biblioteca string é muito rica. Estude sua referência na internet e aprenda do que mais ela é capaz.

```
Entre com seu nome: Joao
Agora entre com seu sobrenome: Paulo
Nome completo: Joao Paulo
Nome completo: Joao Paulo
Ta tranquilo. Ta favoravel.
```

Alguns Tópicos Importantes...

Alguns critérios de parada...

- Em alguns casos, é **muito simples**.
- **Exemplo**: Ler até que a entrada contenha dois zeros separados por espaço.

Exemplo de Entrada

5 3
10 7 4 8 2
8 2
16 11 1 7 29 4 22 2
0 0

```
while(true) {  
    cin >> a >> b;  
    if(a == 0 && b == 0) break;  
  
    // .. continua código  
}
```

Alguns Tópicos Importantes...

Alguns critérios de parada...

- Outra maneira simples é quando a entrada contém a quantidade de casos de teste. Veja:

Exemplo de Entrada

2

50 4

11 20 -15 -13

50 5

30 30 30 40 -78

```
cin >> n_casos;
```

```
for(int i = 0; i < n_casos; i++) {
```

```
    // processa cada caso de teste
```

```
}
```

Alguns Tópicos Importantes...

Alguns critérios de parada...

- Às vezes, é preciso identificar que a entrada acabou, sem que haja nenhuma indicação pra isso.

```
while (scanf() != EOF){  
    // leia o caso de teste  
    // e o processe.  
}
```

```
while (cin >> x) {  
    // leia o caso de teste  
    // e o processe.  
}
```

```
while(getline(cin, str)) {  
    // leia o caso de teste  
    // e o processe.  
}
```

A STL é uma **caixa de ferramentas**, que auxilia e traz soluções para problemas de programação que envolvem estruturas de dados.

Estruturas de Dados

Uma estrutura de dados é uma forma de **armazenar e organizar dados**, preferencialmente provendo maneiras eficientes para a realização de inserções, consultas, buscas, atualizações e remoções.

Na STL, os “**containers**” estão divididos em categorias:

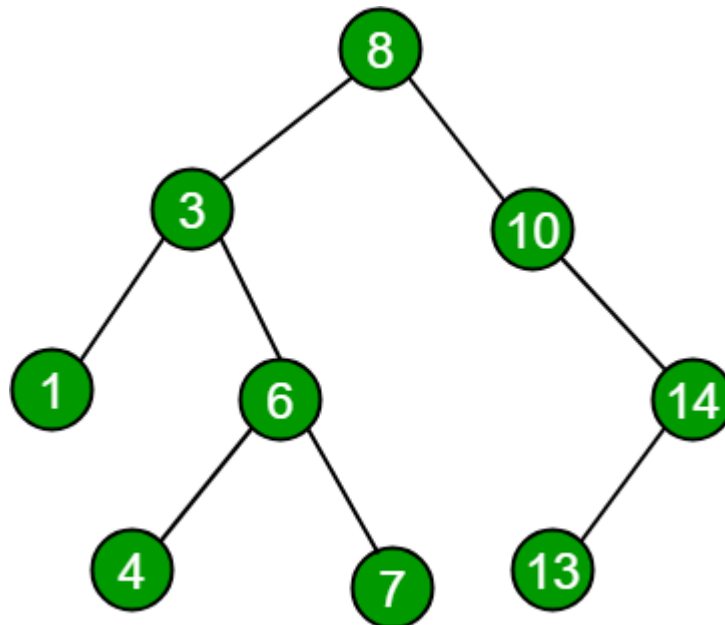
- **Sequenciais:** **vector**, **deque** e **list**.
- **Adaptadores:** **stack**, **queue** e **priority_queue**.
- **Associativos Classificados:** **set** e **map**.

Estruturas não-lineares

- Para alguns problemas, existem maneiras melhores de se representar os dados do que uma sequência linear de dados. Com as implementações da STL que iremos discutir a seguir, podemos realizar buscas mais rápidas e acelerar nossos algoritmos, quando as condições forem favoráveis.
- De quais estruturas estamos falando?
 - **Balanced Binary Search Tree (BST)**
 - STL: `<map>/<set>`
 - **Heap**
 - STL: `<queue>` : `priority_queue`

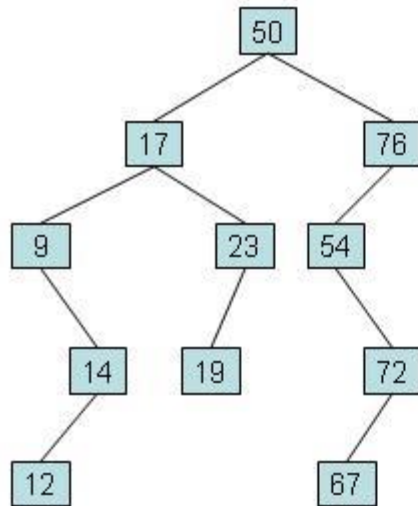
Árvores Binárias

- Organizam dados em forma de **árvore**.
- Como estão dispostos os dados em uma árvore binária de busca (BST)? Em cada subárvore com raiz em x , os itens na sub-árvore à **esquerda** de x são **menores** x e os itens na sub-árvore à **direita** de x são **maiores ou iguais** a x .

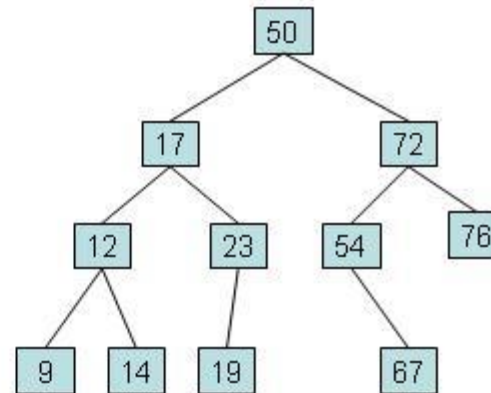


Árvores Binárias

- Mas quando uma BST é **balanceada**?
- Quando há um esforço para manter a sua altura (número de arestas entre a raiz e o nó folha mais distante) a menor possível, o que facilita a busca de valores (menos passos para encontrar qualquer valor).
- Operações realizadas em **$O(\log n)$** .



An unbalanced tree



The same tree after
being height-balanced

Árvores Binárias

- Este tipo de organização permite inserção, busca e remoção em **$O(\log n)$** , mas somente funciona se a árvore for balanceada.
- Exemplos de árvores que são “*self-balancing*”: **AVL**, **RB-Tree**.
- STL **<map>** e **<set>** são implementações de um tipo de árvore binária de busca balanceada chamada Árvore Rubro-negra, ou **Red-Black Tree** em inglês.
- Link da Wikipedia:
https://pt.wikipedia.org/wiki/Árvore_rubro-negra
- **Qual a diferença?**
 - **<map>** armazena pares (key, data)
 - **<set>** armazena apenas (key)

set

```
#include <iostream>
#include <set>

using namespace std;

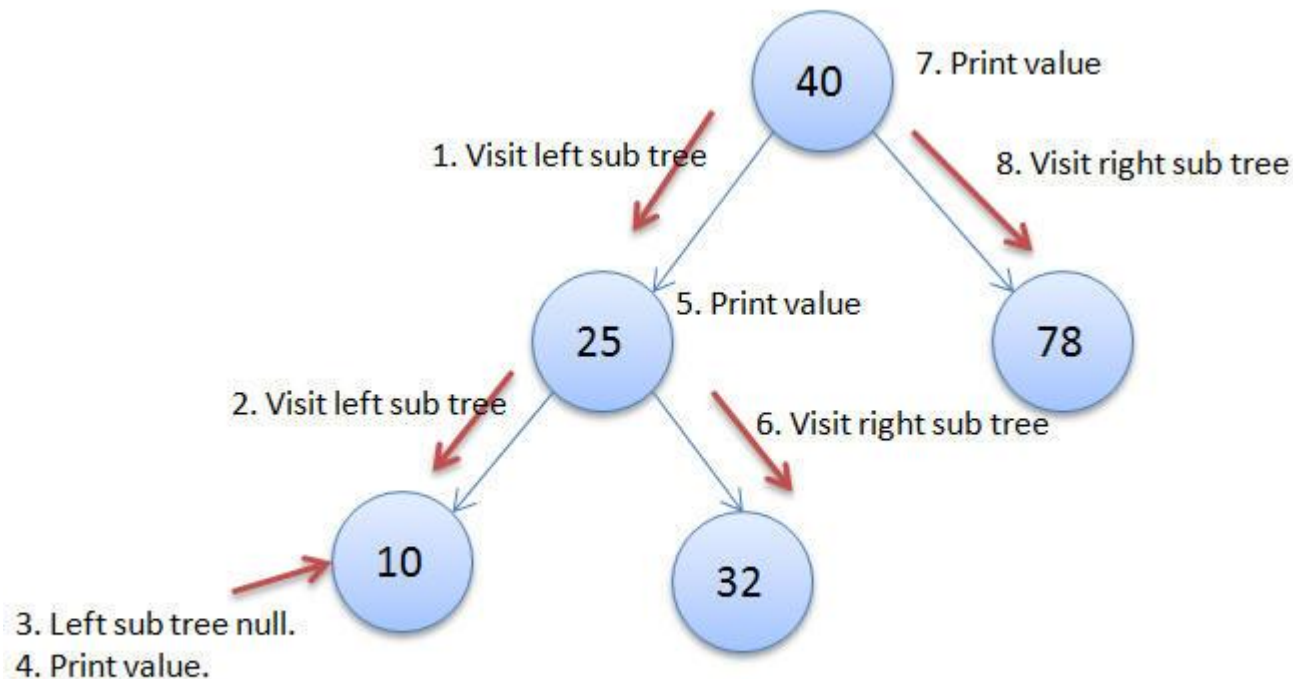
int main()
{
    set<int> conjunto;
    set<int>::iterator it;

    // Em um set, as chaves não podem ser duplicadas.
    // para isso, utilize um multiset (também de <set>
    conjunto.insert(30);
    conjunto.insert(20);
    conjunto.insert(10);
    conjunto.insert(10); // não será inserido
    conjunto.insert(20); // não será inserido
    conjunto.insert(40);
    conjunto.erase(10); // apaga um item, baseado no valor.

    cout << "Quantidade de elementos: " << conjunto.size() << endl;
    cout << "Elementos: ";
    // perceba que os elementos em um set estão sempre ordenados
    for(it = conjunto.begin(); it != conjunto.end(); ++it)
        cout << *it << " ";
    cout << endl;

    return 0;
}
```

Mas... a estrutura da BST (set/map) é não-linear e os dados são **visitados em ordem** pelo iterador? Como isso é realizado?



The above INORDER traversal gives: **10, 25, 32, 40, 78**

map

```
#include <map>
#include <iostream>

using namespace std;

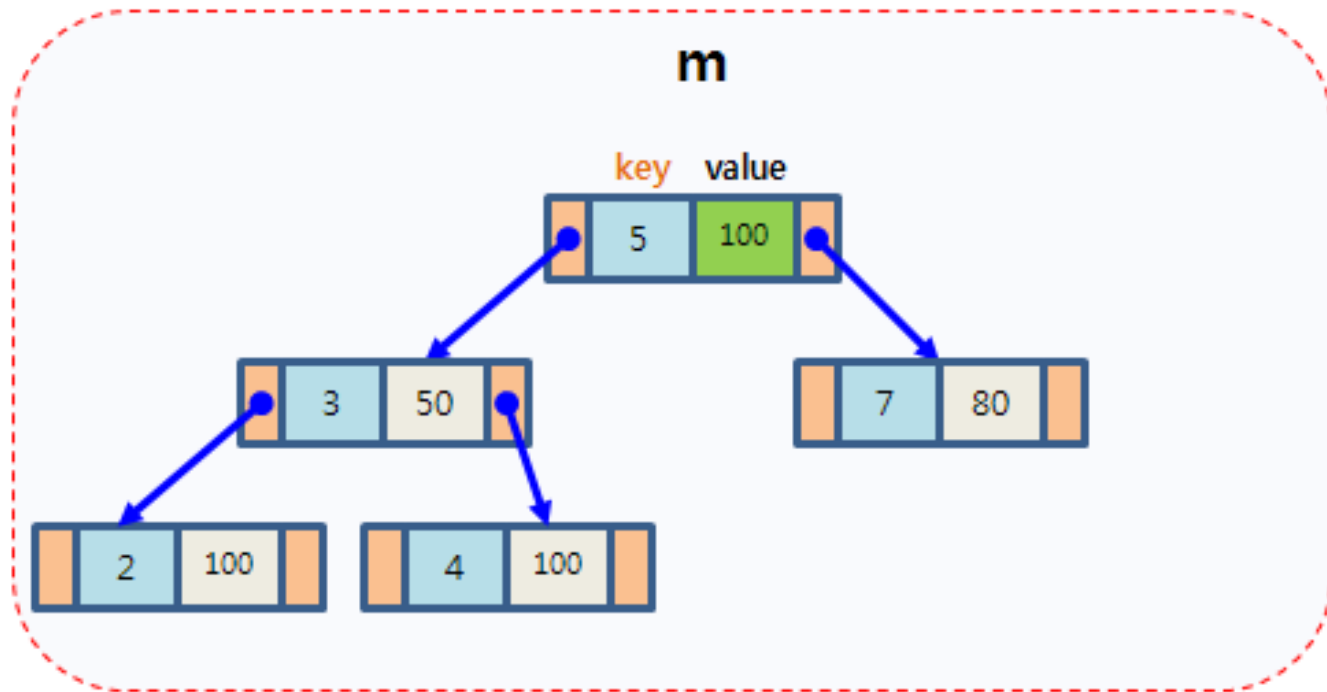
int main()
{
    map<int, string> alunos;
    map<int, string>::iterator it;

    // inserindo quatro alunos
    alunos.insert(make_pair(11984, "Joao Paulo"));
    alunos.insert(make_pair(23456, "Jose"));
    alunos.insert(make_pair(8541, "Carlos"));
    alunos.insert(make_pair(8541, "Edmilson")); // Não é inserido (multimap)
    alunos.insert(pair<int, string> (29546, "Maria"));

    // Removendo José
    alunos.erase(23456);

    // Imprimindo lista de alunos
    // Repare que o map está ordenado de acordo com a chave
    cout << "Lista de ALunos: " << endl;
    for(it = alunos.begin(); it != alunos.end(); ++it)
        cout << it->first << " - " << it->second << endl;

    return 0;
}
```



Exemplo de C++ STL map

Exercício A: Word Index

Esquemas de codificação são frequentemente usados em situações que requerem criptografia ou economia de transmissão/armazenamento. Aqui, faremos um esquema simples de codificação, que codifica tipos específicos de palavras com cinco ou menos letras minúsculas em inteiros.

Considere o alfabeto inglês $\{a, b, c, \dots, z\}$. Usando este alfabeto, um conjunto de palavras válidas podem ser formadas que estejam em ordem lexicográfica. Neste conjunto de palavras válidas, as letras de uma palavra estão sempre em ordem crescente, com relação à sua posição no alfabeto. Por exemplo, “abc aep gwz” são três palavras válidas, enquanto “aab are cat” não são.

Para cada palavra válida, associe um inteiro que dê a posição da palavra na lista alfabetizada de palavras. Ou seja: a -> 1, b -> 2, ... z -> 26, ab -> 27, ac -> 28, ... az -> 51, bc -> 52, ... vwxyz -> 83681

Seu programa precisa ler uma série de linhas de entrada. Para cada palavra lida, se a palavra for inválida, imprima o número 0. Se a palavra for válida, dê sua posição na lista de palavras descrita anteriormente.

Exercício A: Word Index

Entrada

A entrada consiste em uma série de palavras, uma por linha. As palavras tem pelo menos uma letra e, no máximo, cinco letras. Apenas letras minúsculas são utilizadas {a, b, ...,z}. A entrada termina no final do arquivo.

Saída

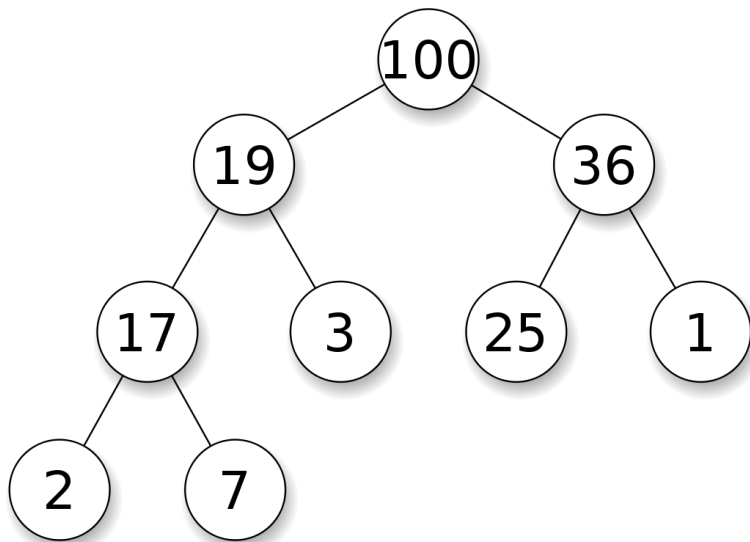
A saída é um único número inteiro maior ou igual a 0 e menor ou igual a 83681. Há uma linha de saída pra cada linha de entrada.

Exemplos de Entrada	Saída correspondente
z	26
a	1
cat	0
vWXYZ	83681

Gere todas as palavras, adicione a um *map* para auto ordenação.

Heap

- *Heap* é outra maneira de organizar dados em forma de árvore binária, mas com propriedades diferentes da BST.
- Como estão dispostos os dados em uma *heap*? Para cada subárvore com raiz em x , itens das subárvores esquerda e direita são menores do que x . Isso garante que o topo da *heap* (e de cada subárvore) terá o elemento de **valor máximo**.



Link na Wikipédia:

<http://pt.wikipedia.org/wiki/Heap>

Heap

- Em uma *heap*, a árvore montada precisa ser completa pelo menos até seu penúltimo nível. No último nível, os itens estão preenchidos da esquerda para a direita, mas não precisa ser completo (único).
- Normalmente não realizamos buscas na heap. Seus pontos fortes são a inserção e remoção de dados, que podem ser feitas em **$O(\log n)$** .
- Pode ser modelada como uma **fila de prioridade**.
- Em STL, está implementada como **priority_queue** <queue>.
- É importante em uma série de algoritmos como **Dijkstra**, **Kruskal**, e outros.

Heap

Priority queue (STL)

Filas de prioridade são um tipo de adaptador de container projetado especificamente para que seu **primeiro elemento seja sempre o maior entre todos os elementos**, de acordo com um critério de ordenação.

O contexto é, portanto, o de uma *heap*, onde elementos podem ser inseridos a qualquer momento, e somente o elemento máximo da heap pode ser obtido (aquele no topo da fila de prioridade, `pop_back`).

Funções membro:

- **empty**
- **size**
- **top**
- **push** (`push_back`)
- **pop** (`pop_back`)

priority_queue

```
#include <iostream>
#include <queue>

using namespace std;

int main()
{
    priority_queue<float> distancias;

    // insere valores de distancias
    distancias.push(1000.0);
    distancias.push(100.0);
    distancias.push(10.0);
    distancias.push(1001.0);
    distancias.push(900.0);

    cout << "Imprimindo na ordem de prioridade: " << endl;
    while(!distancias.empty())
    {
        cout << distancias.top() << endl;
        distancias.pop();
    }

    return 0;
}
```

Exercício B: I Can Guess

Há uma estrutura parecida com uma sacola, que suporta duas operações básicas::

1 x : Coloca um elemento x na sacola.

2 : Tira um elemento da sacola.

Dada uma sequência de operações com valores de retorno, você precisará adivinhar qual é a estrutura de dados em questão. Ela pode ser uma pilha (Last-In, First-Out), uma fila (First-In, First-Out), uma fila de prioridade (sempre tira o maior elemento primeiro) ou alguma outra que você nem imagina!

Exercício B: I Can Guess

Entrada

Há vários casos de teste. Cada caso começa com uma linha contendo um único inteiro (entre 1 e 1000). Cada uma das próximas n linhas é um comando do tipo 1, ou um comando do tipo dois seguido de um inteiro x. Isso significa que o elemento x foi retornado pela chamada do comando 2. O valor de x é sempre um inteiro positivo menor ou igual a 100. A entrada termina com o fim d arquivo (EOF).

Exercício B: I Can Guess

Saída

Para cada caso de teste, imprima uma das respostas::

`stack` se for definitivamente uma pilha.

`Queue` se for definitivamente uma fila.

`priority queue` se for definitivamente uma fila de prioridade.

`impossible` não pode ser pilha, fila ou fila de prioridade.

`not sure` pode ser mais de uma das estruturas mencionadas.

Exemplo de Entrada	Exemplo de Saída
6	queue
1 1	not sure
1 2	impossible
1 3	stack
2 1	priority queue
2 2	impossible
2 3	
6	
1 1	
1 2	
1 3	
2 3	
2 2	
2 1	
2	
1 1	
2 2	
4	
1 2	
1 1	
2 1	
2 2	
7	
1 2	
1 5	
1 1	
1 3	
2 5	
1 4	
2 4	
1	
2 1	