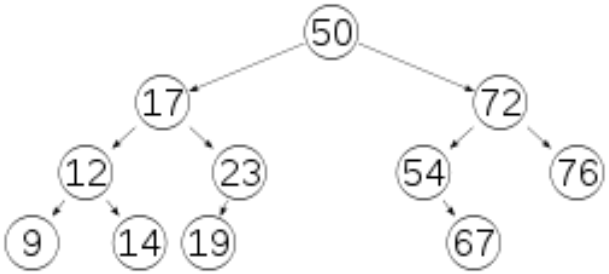
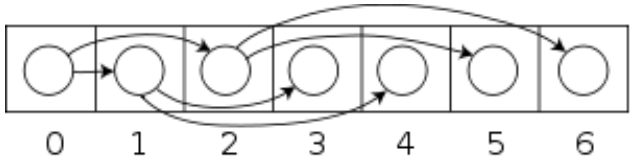




#include <algorithm>

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

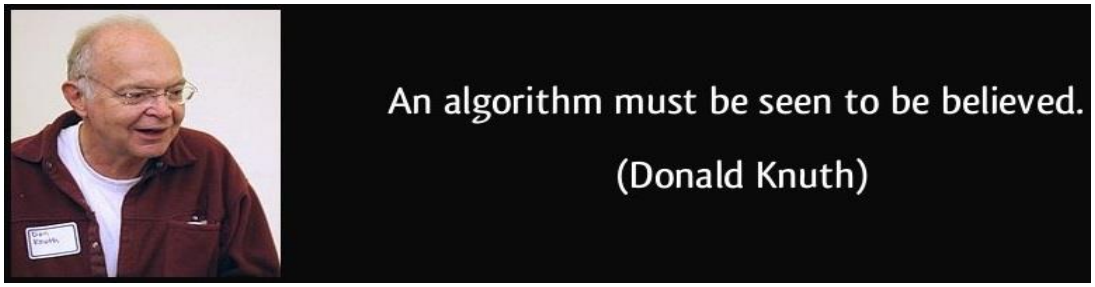
@FabioGaluppo

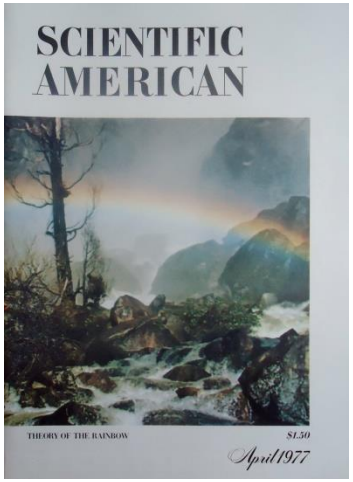
O que é Algoritmo?

Algorithms

An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined it can be translated into computer language and executed by machine

by Donald E. Knuth





https://www.scientificamerican.com/magazine/sa/1977/04-01/#article-algorithms

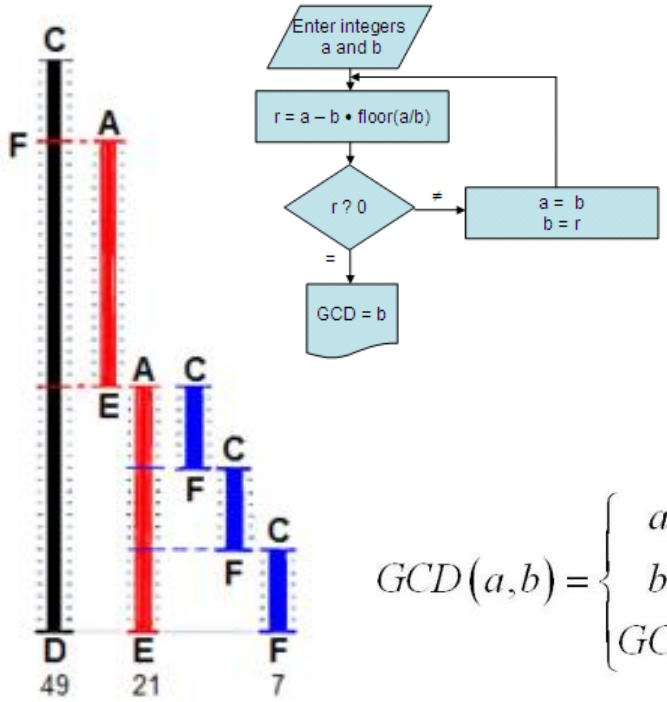
Algorithms

WARM UP

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Algoritmos e Linguagens de Programação

- Algoritmo é um conceito mental que existe independemente de qualquer representação



$$7854 = 1 \cdot 4746 + 3108$$
$$4746 = 1 \cdot 3108 + 1638$$
$$3108 = 1 \cdot 1638 + 1470$$
$$1638 = 1 \cdot 1470 + 168$$
$$1470 = 8 \cdot 168 + 126$$
$$168 = 1 \cdot 126 + 42$$
$$126 = 3 \cdot 42 + 0.$$

$$GCD(a,b) = \begin{cases} a & \text{if } b=0 \\ b & \text{if } a=0 \\ GCD(b, a \bmod b) & \text{otherwise} \end{cases}$$

```
1: procedure EUCLID(a, b)
2:     r ← a mod b
3:     while r ≠ 0 do
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b
9: end procedure
```

https://en.wikipedia.org/wiki/Euclidean_algorithm

Algoritmos e Linguagens de Programação

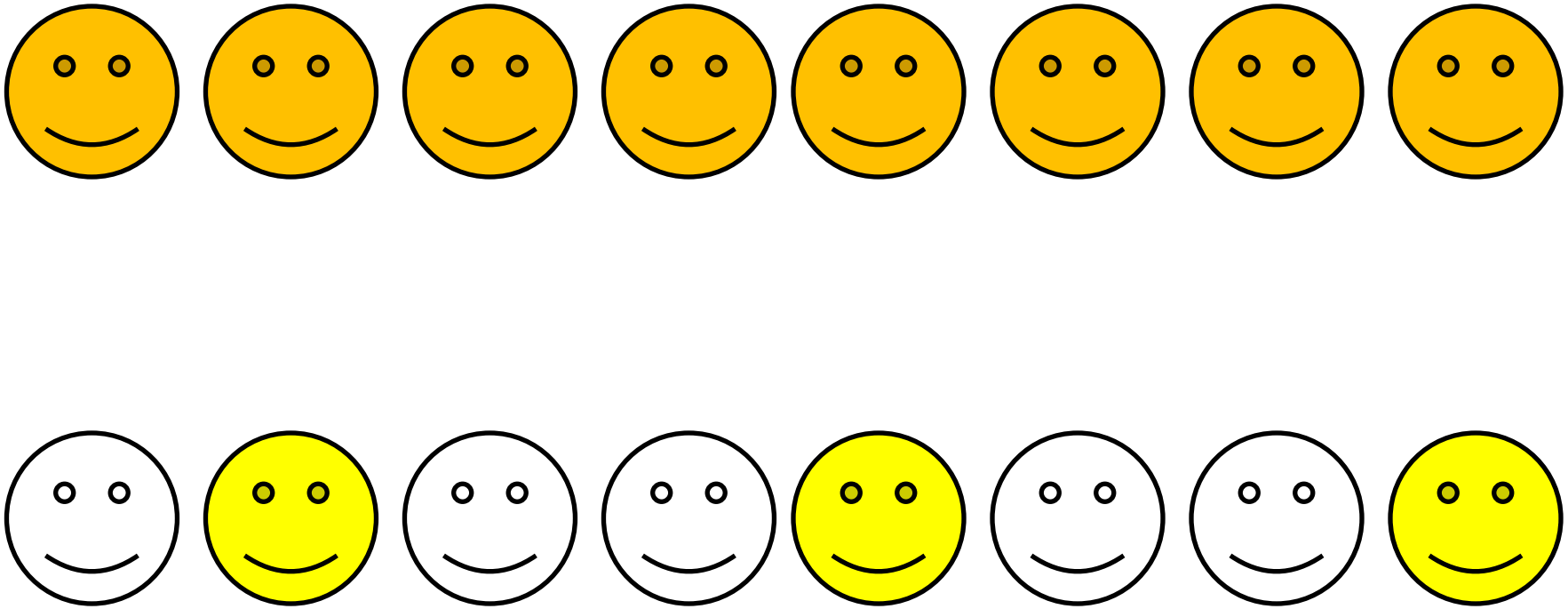
- Greatest Common Divisor (GCD) in C++:

```
template <typename T>
T gcd_iterative(T a, T b)
{
    T temp = a % b;
    while (!(temp == T(0)))
    {
        a = b;
        b = temp;
        temp = a % b;
    }
    return b;
}
```

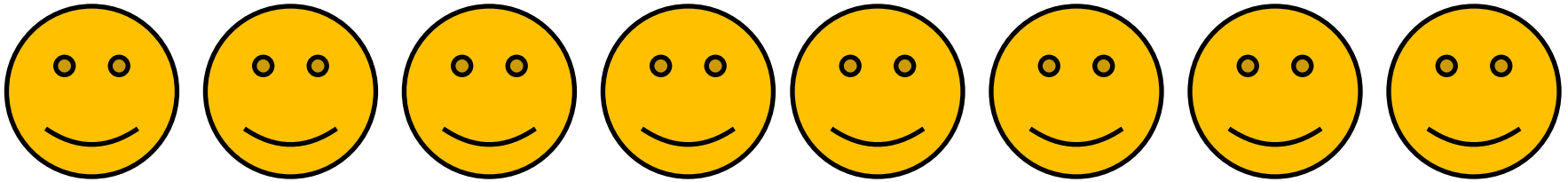
```
template <typename T>
T gcd_recursive(T a, T b)
{
    if (b == T(0)) return a;
    if (a == T(0)) return b;
    return gcd_recursive(b, a % b);
}
```

- Você utiliza um código para dizer ao computador o que ele deve fazer. No entanto, antes você precisa elaborar um algoritmo

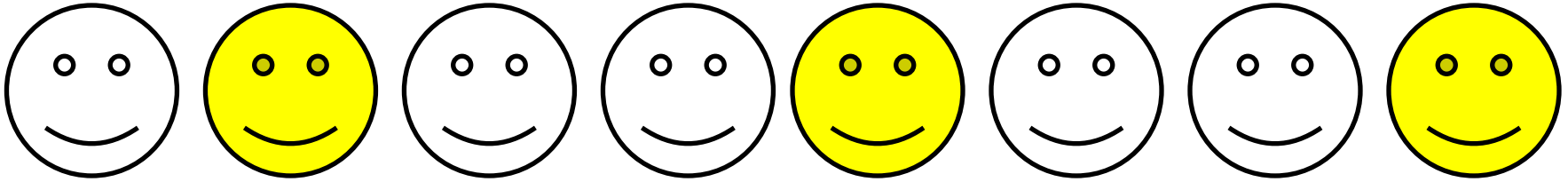
Sobre esforço computacional



Sobre esforço computacional



$$N = 8 \rightarrow Contagem = 8$$



$$N = 8 \rightarrow Contagem = 3$$

$$3 = log_2 8$$

Análise de desempenho

- Assintótica – limite tendendo ao infinito

```
std::max_element(xs.begin(), xs.end());
```
$O(N)$

```
for (int i = 0; i < n - 1; ++i)
    for (int j = 0; j < n - i - 1; ++j)
        if (xs[j] > xs[j + 1])
            std::swap(xs[j], xs[j + 1]);
```
$O(N^2)$

- Empírica – experimentos ou observações

```
Estimated runnning time is 2.4438e-06 x N^1.0000 ms
```

```
Estimated runnning time is 1.9701e-07 x N^2.2192 ms
```

Algoritmo e Memória

(versão imperativa)

- Contar a incidência de um valor em uma estrutura de dados

```
size_t count_occurrences(const std::vector<int>& xs, int target)
{
    size_t counter = 0;
    for (size_t i = 0; i < xs.size(); ++i)
        if (xs[i] == target)
            ++counter;
    return counter;
}
```

index-based

0	1	2	3	4	5	6	7	8	9	10
1	2	2	3	1	4	5	1	2	5	2

Algoritmo e Memória

(versão imperativa)

- Contar a incidência de um valor em uma estrutura de dados

pointer-based

```
size_t count_occurrences(node* ptr, int target)
{
    size_t counter = 0;
    while (ptr != nullptr)
    {
        if (ptr->value == target)
            ++counter;
        ptr = ptr->next;
    }
    return counter;
}
```

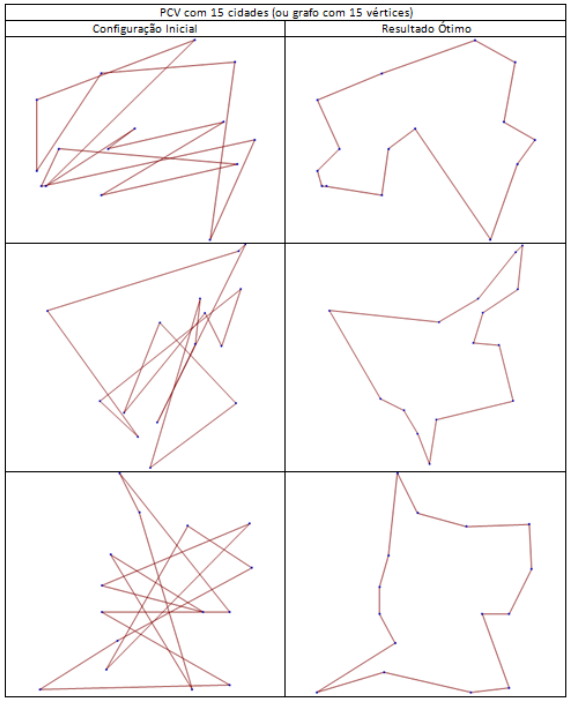
```
struct node
{
    node(int value, node* next = nullptr) :
        value(value), next(next) {}
    int value;
    node* next;
};
```

Sobre eficiência e utilidade

- Um algoritmo é plenamente útil se e somente se for eficiente!

	Tempo (ms)
Número de Cidades	Força Bruta
13	743691
12	53093
11	4056
10	331
9	39
8	2
7	1

PCV com 15 Cidades		
	Força Bruta	
Solução Ótima	Tempo (ms)	Tempo (h)
359,399	165340592	45,928
317,232	165590540	45,997
368,79	165517424	45,977



15!

Result:

1 307 674 368 000

WolframAlpha computational intelligence.

Number name:

1 trillion 307 billion 674 million 368 thousand

$47!/2 = 1{,}29311 \times 10^{59}$ $47! = 2{,}58623 \times 10^{59}$ $48! = 1{,}24139 \times 10^{60}$

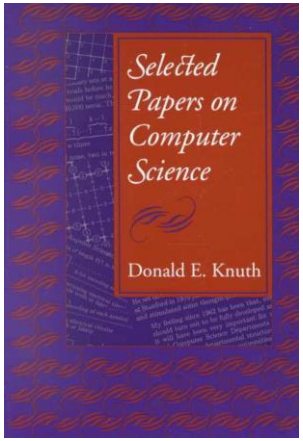
Algorithms

SHOW ME THE C++ CODE

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Algorithms

```
1: Searching a Computer's Memory
2: The Advantage of Order
3: Binary Tree Search
4: Hashing (linear probing)
5: Improving Unsuccessful Searches
```



I find that I don't understand things unless I try to program them.

Donald E. Knuth
Professor Emeritus at Stanford University

https://www-cs-faculty.stanford.edu/~knuth/cs.html

Searching a Computer's Memory

```
static inline std::size_t sequencial_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t N = words.size();
    while (N)
    {
        if (words[--N].word == word)
            return N;
    }
    return NOT_FOUND;
}
```

```
static inline const std::vector<word_count>& words()
{
    //monotonically decreasing order by count
    static std::vector<word_count> _words
    {
        { "THE", 15568 }, { "OF", 9767 },   { "AND", 7638 },   { "TO", 5739 },
        { "A", 5074 },    { "IN", 4312 },   { "THAT", 3017 },  { "IS", 2509 },
        { "I", 2292 },    { "IT", 2255 },   { "FOR", 1869 },   { "AS", 1853 },
        { "WITH", 1849 }, { "WAS", 1761 },  { "HIS", 1732 },   { "HE", 1727 },
        { "BE", 1535 },   { "NOT", 1469 },  { "BY", 1392 },    { "BUT", 1379 },
        { "HAVE", 1344 }, { "YOU", 1336 },  { "WHICH", 1291 }, { "ARE", 1222 },
        { "ON", 1155 },   { "OR", 1101 },   { "HER", 1093 },   { "HAD", 1062 },
        { "AT", 1053 },   { "FROM", 1039 }, { "THIS", 1021 }
    };
    return _words;
}
```

The Advantage of Order

```
static inline std::size_t binary_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t l = 0, r = words.size(), mid;
    while (l != r)
    {
        mid = (r + l) / 2;
        int comp = word.compare(words[mid].word);
        if (comp == 0)
            return mid;
        else if (comp < 0)
            r = mid;
        else /* if (comp > 0) */
            l = mid + 1;
    }
    return NOT_FOUND;
}
```

The Advantage of Order
Achilles' hell of Binary Search

```
mid = (r + l) / 2;
```

```
79      std::size_t l_ = 0;
80      std::size_t r_ = std::numeric_limits<std::size_t>::max(); //4294967295
81  overflow_case:
82      std::size_t mid_ = (r_ + l_) / 2; //4294967295 + 2147483648 = 6442450943 / 2 = 3221225471
83      l_ = mid_ + 1; //2147483648 ≤ 1ms elapsed
84      goto overflow_case;
```

Name	Value	Type
l	0	unsi
l_	2147483648	unsi
mid	3435973836	unsi
mid_	1073741823	unsi
r		
r_		
word		
words		

Calculator
Scientific
1,073,741,823

Calculator
Scientific
3,221,225,471.5

F-E
M+ M- MS M˅

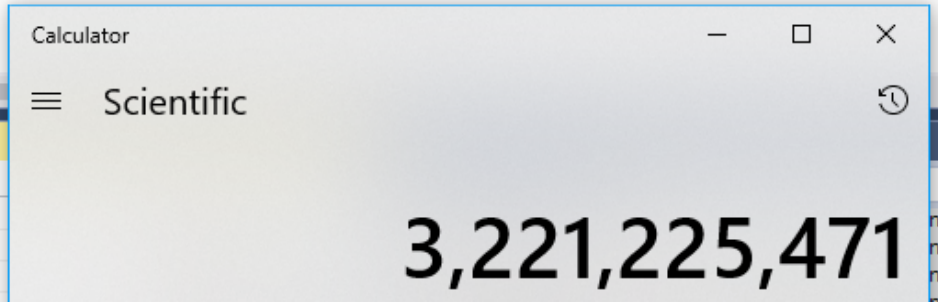
The Advantage of Order Fixing Binary Search

```
mid = l + (r - l) / 2;
```

```
79      /*
80      >>> from math import floor
81      >>> def lerp(begin, end, percent):
82      ...     return begin + floor(percent * (end - begin))
83      ...
84      >>> lerp(2147483648, 4294967295, 0.5)
85      3221225471
86      */
87      std::size_t l_ = 0;
88      std::size_t r_ = std::numeric_limits<std::size_t>::max(); //4294967295
89  no_overflow_case:
90      std::size_t mid_ = l_ + (r_ - l_) / 2; //lerp(2147483648, 4294967295, 50%) = 3221225471
91      l_ = mid_ + 1; //2147483648 ≤ 1ms elapsed
92      goto no_overflow_case;
93
```

ls

me	Value
l_	2147483648
mid	3435973836
mid_	3221225471

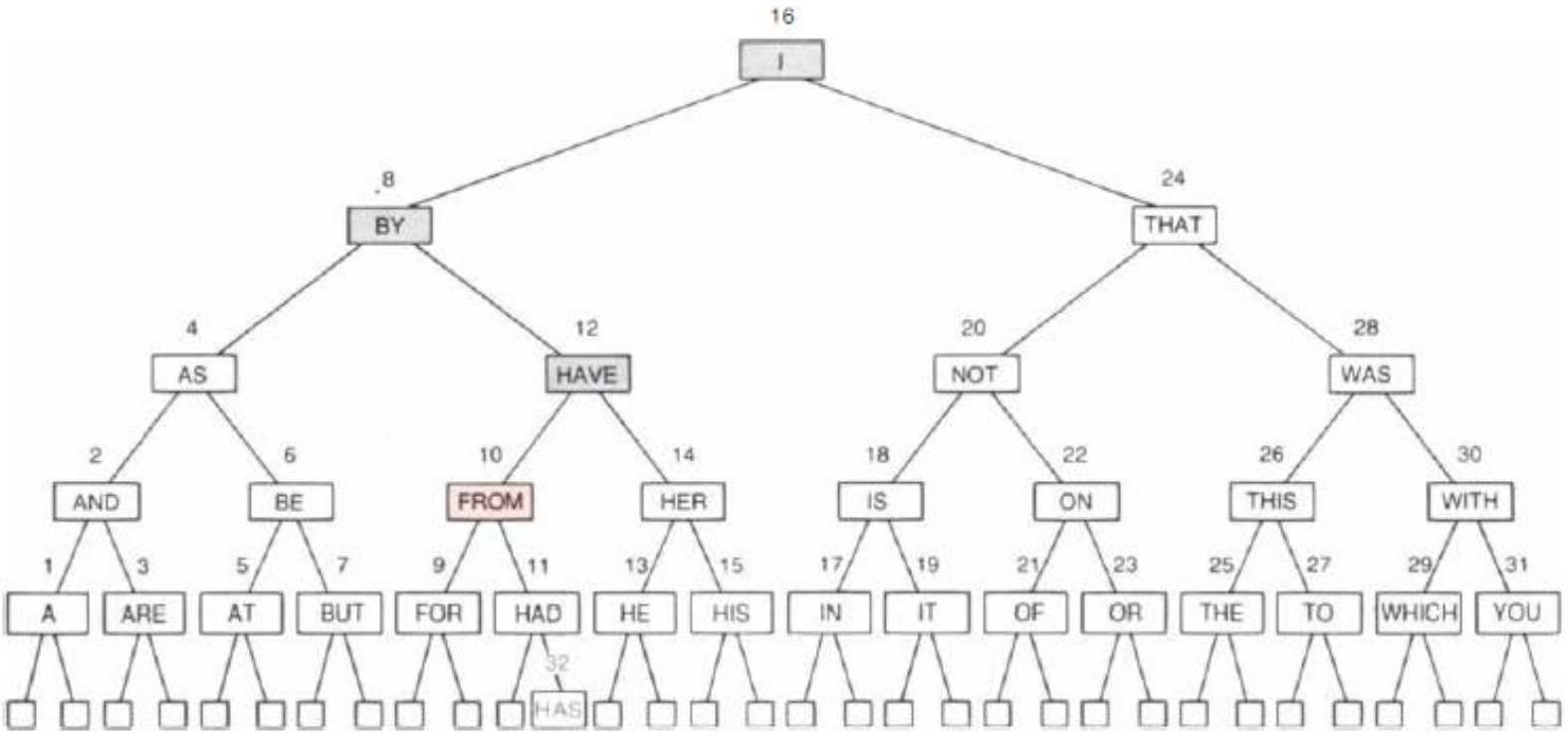


The Advantage of Order

```
static inline std::size_t binary_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t l = 0, r = words.size(), mid;
    while (l != r)
    {
        mid = l + (r - l) / 2;
        int comp = word.compare(words[mid].word);
        if (comp == 0)
            return mid;
        else if (comp < 0)
            r = mid;
        else /* if (comp > 0) */
            l = mid + 1;
    }
    return NOT_FOUND;
}
```

Binary Tree Search

```
struct bst_node final
{
    bst_node(size_t index) : index(index) {}
    std::size_t index = NOT_FOUND;
    std::unique_ptr<bst_node> left = nullptr;
    std::unique_ptr<bst_node> right = nullptr;
};
```



https://www.scientificamerican.com/magazine/sa/1977/04-01/#article-algorithms

Binary Tree Search

```
static std::size_t bst_search_recursive
(const bst_node* ptr, const std::vector<word_count>& words, const std::string& word)
{
    if (ptr == nullptr)
        return NOT_FOUND;
    std::size_t index = ptr->index;
    int comp = word.compare(words[index].word);
    if (comp == 0)
        return index;
    if (comp < 0)
        return bst_search_recursive(ptr->left.get(), words, word);
    return bst_search_recursive(ptr->right.get(), words, word);
}
```

Hashing

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[THIS,23]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[FROM,19]
14:[AT,20]
15:[NOT,16]
```

```
static inline std::size_t hash
(const std::string& word, std::size_t bucket_size)
{
    std::size_t h = 0;
    for (char ch : word)
        h += (ch - 'A' + 1);
    return --h % bucket_size;
}
```

```
16:[THAT,16]
17:[WHICH,18]
18:[AND,18]
19:[AS,19]
20:[OF,20]
21:[ON,28]
22:[IN,22]
23:[ARE,23]
24:[YOU,28]
25:[BY,26]
26:[WITH,27]
27:[IS,27]
28:[IT,28]
29:[HER,30]
30:[OR,0]
31:[A,0]
```

Hashing

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[THIS,23]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[FROM,19]
14:[AT,20]
15:[NOT,16]
```

```
static inline std::size_t hash_table_search
(const std::vector<word_count>& dic, const std::string& word)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(word, bucket_size);
    while (!dic[h].word.empty())
    {
        if (word == dic[h].word)
            return h;
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    return NOT_FOUND;
}
```

```
16:[THAT,16]
17:[WHICH,18]
18:[AND,18]
19:[AS,19]
20:[OF,20]
21:[ON,28]
22:[IN,22]
23:[ARE,23]
24:[YOU,28]
25:[BY,26]
26:[WITH,27]
27:[IS,27]
28:[IT,28]
29:[HER,30]
30:[OR,0]
31:[A,0]
```

Improving Unsuccessful Searches

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[AND,18]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[ARE,23]
14:[AS,19]
15:[NOT,16]
```

```
static inline void ordered_hash_table_insert
(std::vector<word_count>& dic, word_count wc)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(wc.word, bucket_size);
    while (!dic[h].word.empty())
    {
        if (wc.word.compare(dic[h].word) > 0)
            std::swap(dic[h], wc);
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    dic[h] = wc;
}
```

```
16:[THAT,16]
17:[AT,20]
18:[WHICH,18]
19:[FROM,19]
20:[OF,20]
21:[BY,26]
22:[IN,22]
23:[THIS,23]
24:[IS,27]
25:[IT,28]
26:[ON,28]
27:[WITH,27]
28:[YOU,28]
29:[A,0]
30:[HER,30]
31:[OR,0]
```

Improving Unsuccessful Searches

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[AND,18]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[ARE,23]
14:[AS,19]
15:[NOT,16]
```

```
static inline std::size_t ordered_hash_table_search
(const std::vector<word_count>& dic, const std::string& word)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(word, bucket_size);
    while (!dic[h].word.empty())
    {
        int comp = word.compare(dic[h].word);
        if (comp > 0)
            break;
        if (comp == 0)
            return h;
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    return NOT_FOUND;
}
```

```
16:[THAT,16]
17:[AT,20]
18:[WHICH,18]
19:[FROM,19]
20:[OF,20]
21:[BY,26]
22:[IN,22]
23:[THIS,23]
24:[IS,27]
25:[IT,28]
26:[ON,28]
27:[WITH,27]
28:[YOU,28]
29:[A,0]
30:[HER,30]
31:[OR,0]
```

Algorithms

COOL DOWN

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Programas

- Eles são rodados pelo computador para executar tarefas específicas
  - Solucinar problemas

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

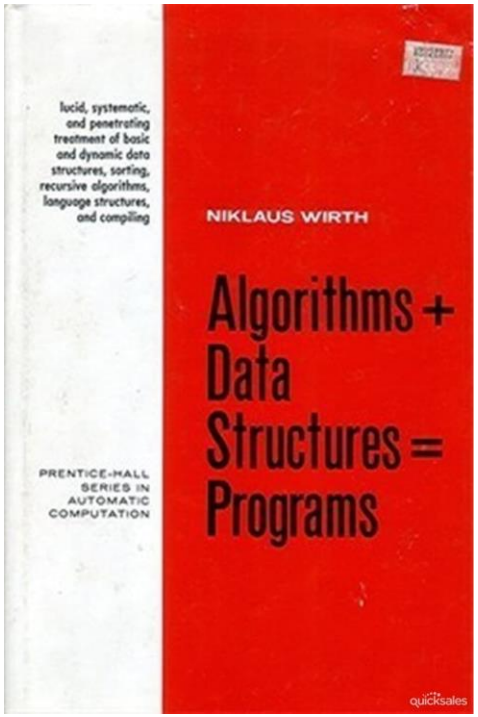
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮



- Algoritmos + Estrutura de Dados

https://algs4.cs.princeton.edu/lectures/00Intro-2x2.pdf

#include <algorithm>

Table 100 — Algorithms library summary

	Subclause	Header(s)
28.5	Non-modifying sequence operations	<algorithm>
28.6	Mutating sequence operations	
28.7	Sorting and related operations	
28.8	C library algorithms	<cstdlib>

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf

- Existem mais de 100 algoritmos padrões bem testados
- Serve como base para novos algoritmos
- Compreensível e mais simples do que um *raw loop*
- Mantém *side-effects* dentro de uma interface bem definida
- Facilita o raciocínio sobre o problema
- Atua em conjunto com *iterators* ou *left-closed interval* $[begin, end)$

Algoritmo

(versão declarativa com STL)

- Contar a incidência de um valor em uma estrutura de dados

```
std::vector<int> ys{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(ys), std::end(ys), 2) << "\n";

std::forward_list<int> zs{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(zs), std::end(zs), 2) << "\n";


std::count_if(std::begin(ys), std::end(ys),
              [](int y) { return !(y & 0x1) /* is even? */; })
```

Algorithms

`#include <algorithm>`

- Searching a Computer's Memory

  `std::find`
- The Advantage of Order

  `std::lower_bound`
- Binary Tree Search

  `std::map` (Red-Black Tree → Balanced BST)
- Hashing

  `std::unordered_map`

  – *Separate Chaining* instead of *Linear Probing*
- Improving Unsuccessful Searches

Algorithms

#include <algorithm>

- between

```
std::tuple<std::vector<std::string>::const_iterator, std::vector<std::string>::const_iterator>
between(const std::vector<std::string>& words, const std::string& lhs, const std::string& rhs)
```

```
const std::vector<word_count> wcs = words_ordered_by_word();
std::vector<std::string> words;
std::transform(wcs.begin(), wcs.end(),
    std::back_inserter(words), [](const word_count& wc) { return wc.word; });
std::vector<std::string>::const_iterator first, last;

std::tie(first, last) = between(words, "HAVE", "NOTHING");
for (auto iter = first; iter != last; ++iter)
    std::cout << *iter << "  ";
std::cout << "\n";
```

```
HAVE  HE  HER  HIS  I  IN  IS  IT  NOT
```

Algorithms

#include <algorithm>

- between

```
std::tuple<std::vector<std::string>::const_iterator, std::vector<std::string>::const_iterator>
between(const std::vector<std::string>& words, const std::string& lhs, const std::string& rhs)
{
    //assert(lhs < rhs);
    //assert(std::is_sorted(words.begin(), words.end()));
    auto first = std::lower_bound(words.begin(), words.end(), lhs);
    auto last  = std::upper_bound(words.begin(), words.end(), rhs);
    return std::make_tuple(first, last);
}
```

Algorithms

`#include <algorithm>`

- Generic Programming
  - 1988

Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software. For example, a class of generic sorting algorithms can be defined which work with finite sequences but which can be instantiated in different ways to produce algorithms working on arrays or linked lists.

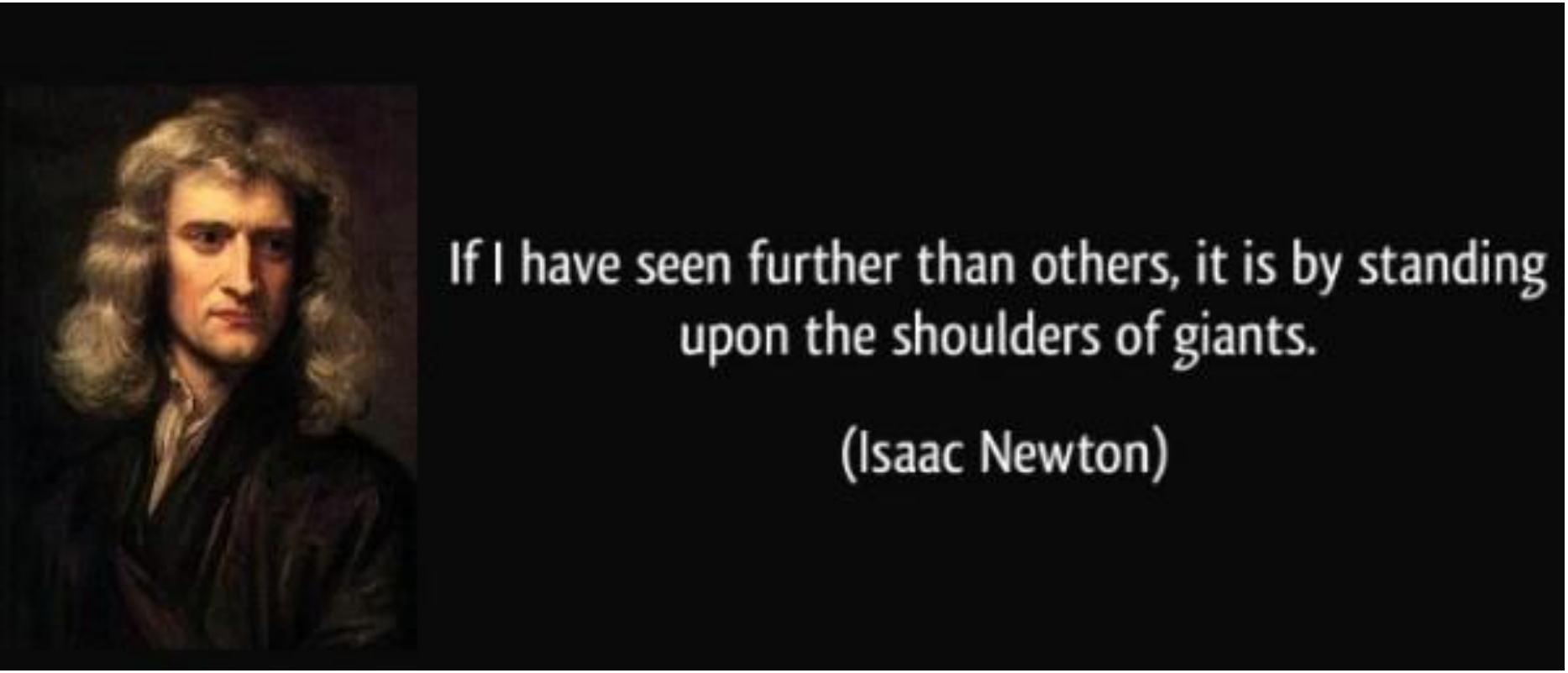
```
std::vector<int> vec{ 33, 55, 11, 99, 88, 44, 22 };
std::list<int>   lst{ 33, 55, 11, 99, 88, 44, 22 };

std::stable_sort(vec.begin(), vec.end());
std::stable_sort(lst.begin(), lst.end());
```

http://stepanovpapers.com/genprog.pdf

https://link.springer.com/book/10.1007/3-540-51084-2

On the Shoulders of Giants



Stepanov on the Shoulders of Knuth

- Generic Programming is about abstracting and classifying algorithms and data structures
- It gets its inspiration from Knuth

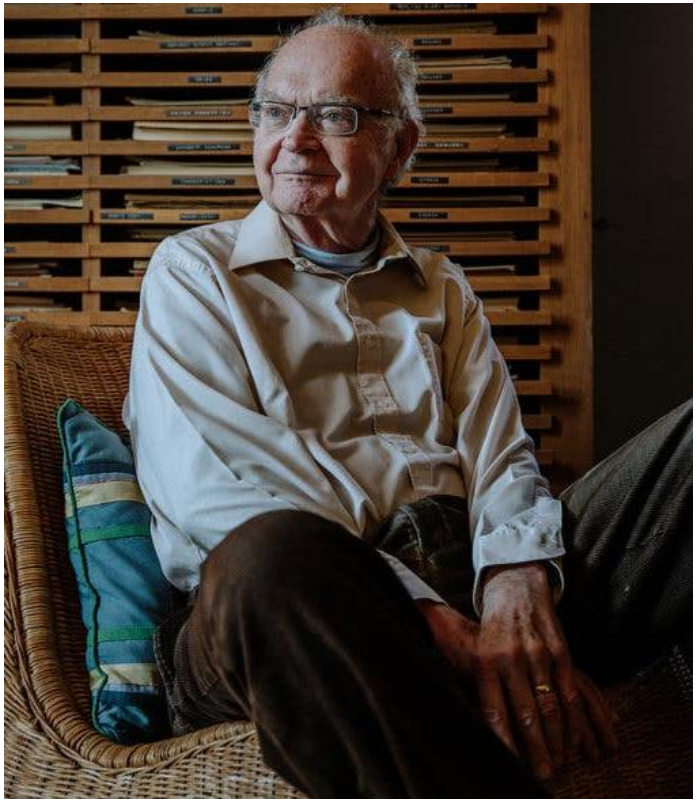
STL is only a limited success. While it became a widely used library, its central intuition did not get across. People confuse generic programming with using (and abusing) C++ templates. Generic programming is about abstracting and classifying algorithms and data structures. It gets its inspiration from Knuth and not from type theory. Its goal is the incremental construction of systematic catalogs of useful, efficient and abstract algorithms and data structures. Such an undertaking is still a dream.

Short History of STL

http://stepanovpapers.com/history%20of%20STL.pdf

On the Shoulders of Giants

Donald Knuth



https://en.wikipedia.org/wiki/Donald_Knuth

Alex Stepanov

https://en.wikipedia.org/wiki/Alexander_Stepanov

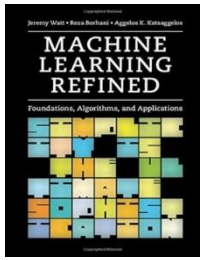
Algorithms

HYPE

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

Tipos de algoritmos de *Machine Learning*

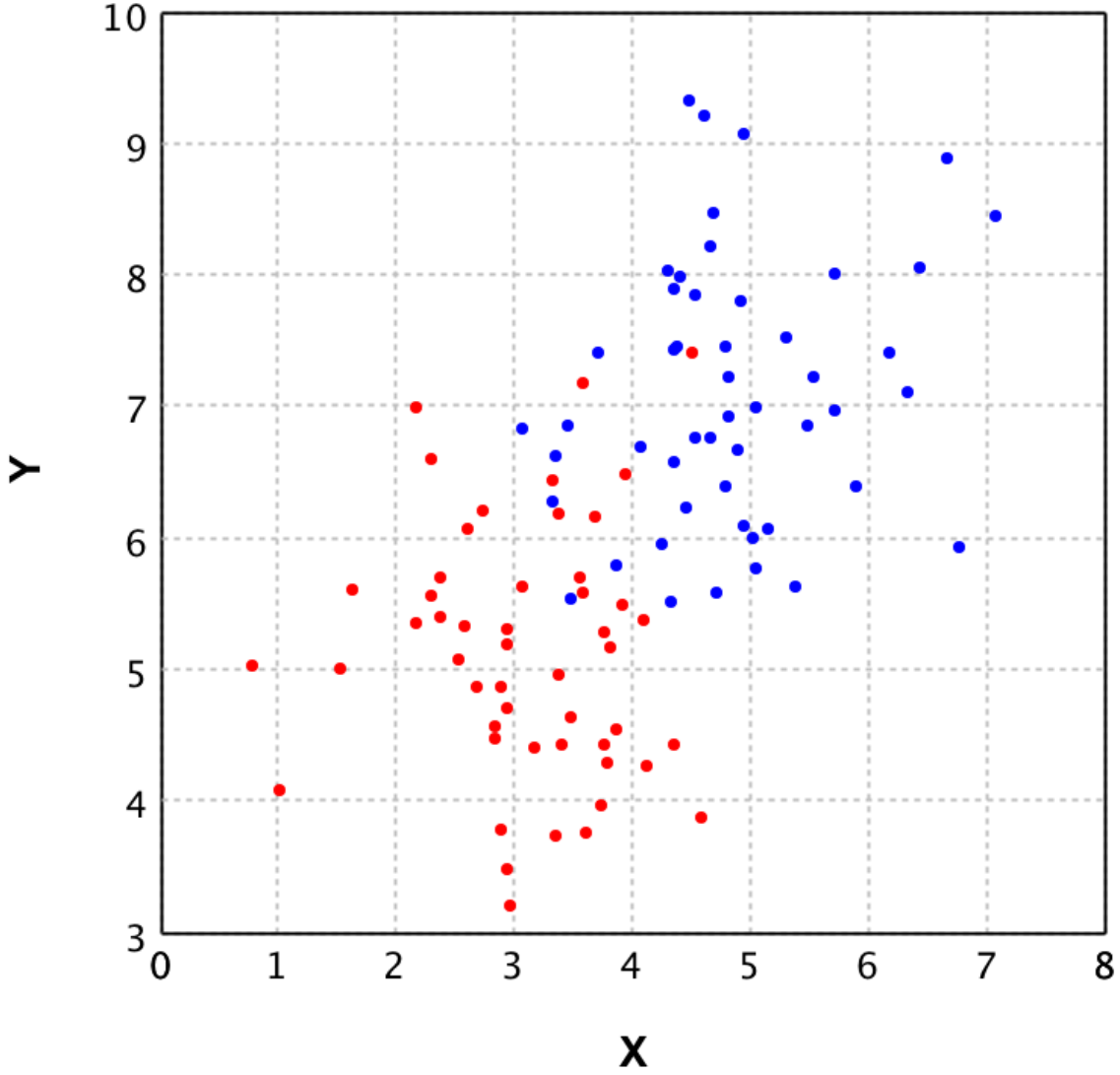
- ***Machine learning*** is a rapidly growing field of study whose primary concern is the ***design and analysis of algorithms*** which ***enable computers to learn***. Machine Learning Refined



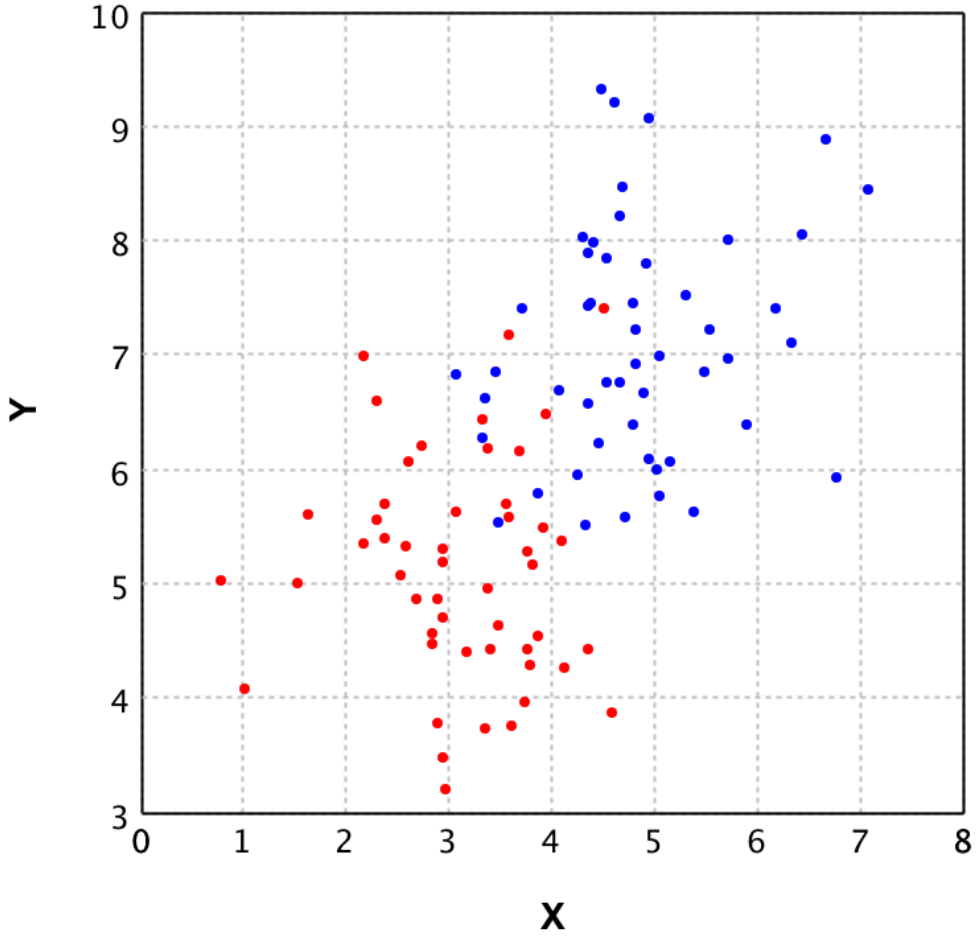
- ***Supervised Learning***
  - algorithms make predictions based on a set of examples
    - ***Classification, Regression, Forecasting***
- ***Unsupervised Learning***
  - It is asked to discover the intrinsic patterns that underlies the data
    - ***Clustering, Dimension reduction***

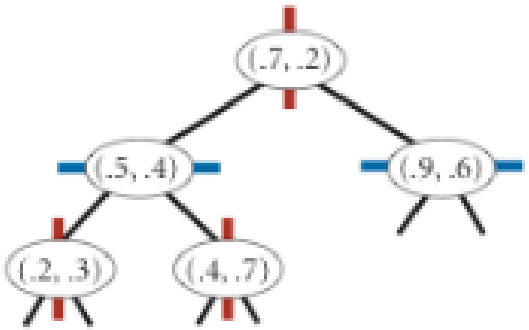
https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

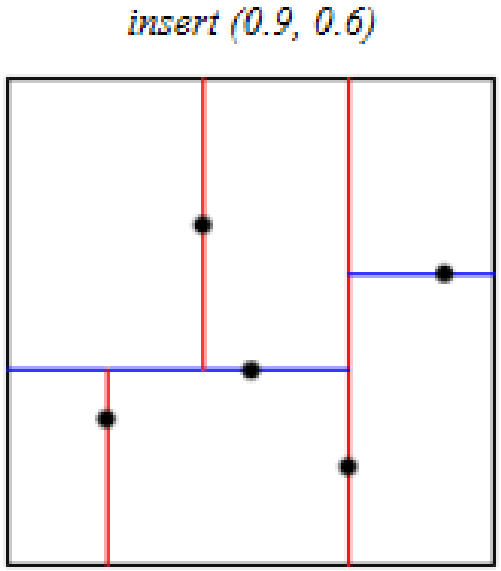
Classification Problem



KdTree and Classification







http://coursera.cs.princeton.edu/algs4/assignments/kdtree.html

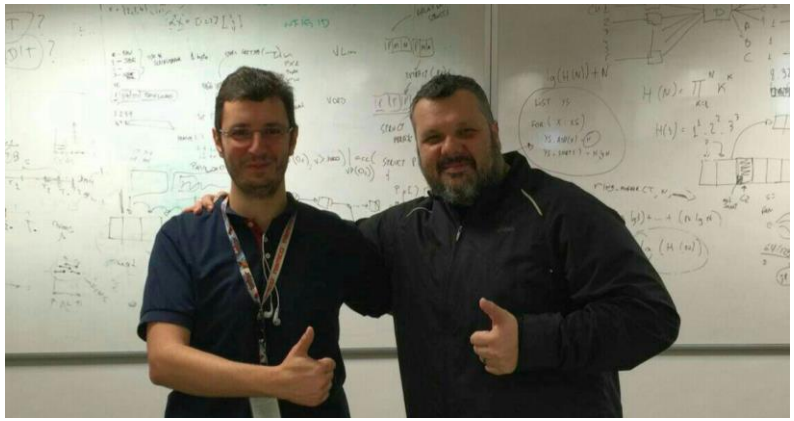
https://xyclade.github.io/MachineLearning/#labeling-isps-based-on-their-downupload-speed-k-nn-using-smile-in-scala

Se quiser saber mais sobre:

C++
Programação Genérica
Iterators
Policy-based Design
Algoritmos
...

visite:

www.simplycpp.com



http://www.simplycpp.com

Se quiser aprender mais sobre:

Algoritmos e C++

CURSO ALGORITMOS COM C++

Perfil do curso:

Introdução de Algoritmos com C++: Fundamentos e Prática para Soluções de Problemas

Objetivo: Capacitar o aluno na aplicação de algoritmos e estrutura de dados fundamentais, bem como, construí-los de forma efetiva na linguagem de programação C++ e adequados ao estilo da Standard Template Library (STL). Simultaneamente, o aluno estará apto a aplicar ou transferir os conceitos adquiridos sobre programação genérica, estrutura de dados fundamentais, algoritmos de busca, algoritmos de ordenação e análise de algoritmos independentemente da linguagem de programação.

Carga horária: 20 horas em sala de aula, com 5 aulas de 4 horas (existirá horas de atividades para estudo em casa).

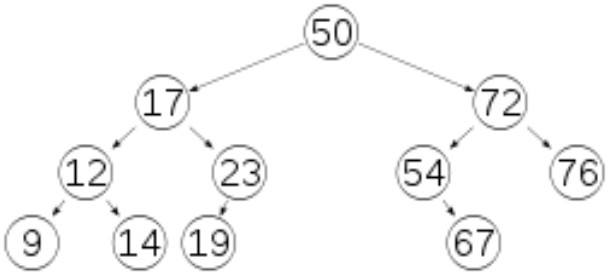
20 horas de carga horária

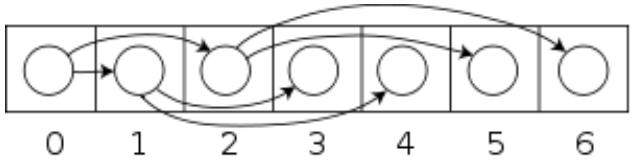
Conteúdo completo Saiba +

Calendário do curso Saiba +

Valores e Descontos Saiba +

https://www.agit.com.br/cursoalgoritmos.php





#include <algorithm>

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

@FabioGaluppo