Abstract Data Type (ADT)

Tipos Abstractos de Datos (TADs)

Theory & Implementations

Course

Analysis and design of algorithms

Instructor

Acosta Bermejo Raúl

Lecture notes

Table of contents (outline)

Tabla de contenido

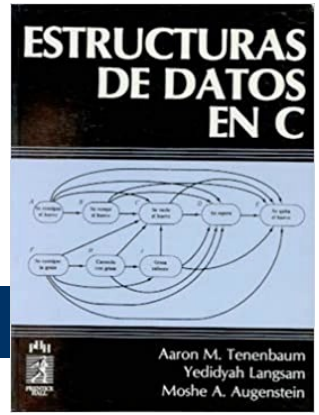
1. Introduction
2. Bibliografía
3. Librerias

1. Lists
2. Stacks
3. Queues
4. Trees
5. Hash
6. Heap

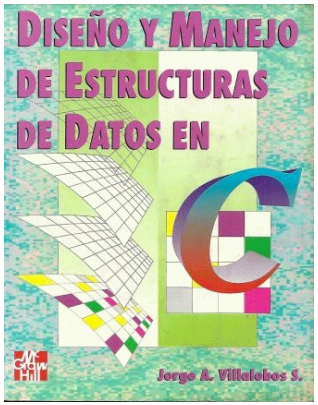
1. Árboles balanceados
2. Trie
3. Treap

Bibliografy

Bibliografía



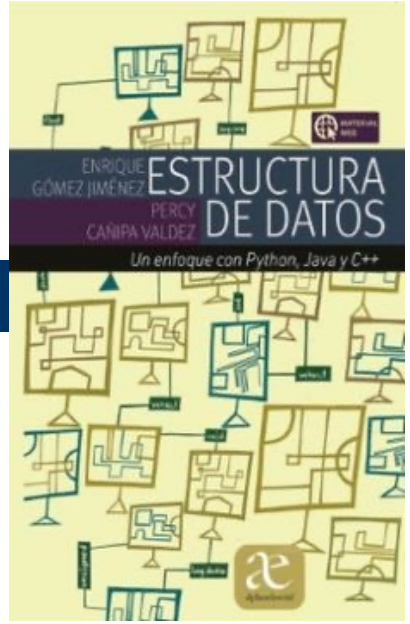
Los más conocidos (y los recomendados)



1. Estructuras de Datos en C, en Pascal, C y C++
   1. Ed. Prentice-Hall Hispanoamericana, 1993.
   2. Tanenbaum Aaron
2. Diseño y manejo de estructura de datos en C
   - Editor: Santafé de Bogotá: McGraw-Hill, 1996.
   - Jorge A Villalobos S.
3. Estructuras de Datos y Algoritmos
   1. Aho, Hopcroft, y Ullman
4. Solo son 3 de una EXTENSA lista de libros del tema.

Bibliografy

Bibliografía



En Python

Solo algunos ejemplos.

1. URLs
   1. Sitio oficial.
   2. https://docs.python.org/es/3/tutorial/datastructures.html
2. Libros
   - Algoritmos y estructuras de datos en Python.
   - https://editorial.uader.edu.ar/wp-content/uploads/2021/04/Algoritmos%20y%20estructuras%20de%20datos%20en%20Python%20-%20digital.pdf
   - Estructura De Datos. Un enfoque con Python, Java y C++
   - Enrique Gómez Jiménez, Jesús Percy Cañipa Valdez
   - https://books.google.com.mx/books/about/Estructura_de_datos.html?id=Xyv5EAAAQBAJ&redir_esc=y

Introduction

Introducción

Tipos de Datos Concretos

- Son tipos básicos que por lo mismo ya se tienen en los lenguajes de programación.
- Ejemplos:

char, Int, float, double

Tipos de Datos Abstractos

- Son tipos de datos considerados “NO básicos“ y por lo tanto no los incluye el lenguaje y se requiere que se agreguen al programa: librerias.
- ¿Pero cómo se crean nuevos tipos? ¿Que ofrece el lenguaje para crear los nuevos tipos?

Introduction

Introducción

Abstract Data Type (ADT)

It is a mathematical model for data types where a data type is defined by its behavior (semantics) from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the *behavior* of these operations.

- This contrasts with data structures, which are concrete representations of data, and are the point of view of an implementer, not a user.
- Formally, an ADT may be defined as a "class of objects whose logical behavior is defined by a set of values and a set of operations"; this is analogous to an algebraic structure in mathematics.

Introduction

Introducción

Implementation of ADTs

- An ADT may be implemented by:
  - Specific data types, or
  - Data structures (concrete data types),

    In many ways and in many programming languages; or described in a formal specification language.
- For example, integers as an ADT:
  - Defines as the values …, -2, -1, 0, 1, 2, ... and by
  - The operations of addition, substraction, multiplication, division, greather than, and less than.
  - The behavior (semantic) includes obeying various axioms (associativity and conmutatitvity of additions, etc.).

Introduction

Introducción

Data structures

- For example:
  - The type system in C is static and weakly typed (like ALGOL descendants such as Pascal)
  - There are built-in types for integers of various sizes, both signed and unsigned, floating-point numbers, characters, and enumerated types (enum).
  - There are also *derived types* including arrays, pointers, records (struct), and untagged unions (union).

Introduction

Introducción

Defining an ADT

- There are no standard conventions for defining them.
- A broad division may be drawn between "imperative" and "functional" definition styles (paradigms).
- Programming paradigm:
  - Imperative: Focuses on describing **HOW** a program operates. It uses statements (commands, control flow) that change a program'state (variables).

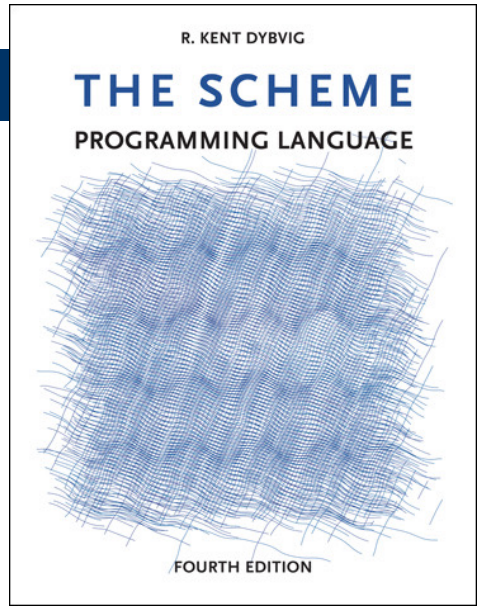
    Procedural, structures, OO programming
  - Declarative: Focuses on **WHAT** the program should accomplish without specifying *how* the program should achive the result.

    Often consideres programs as theories of a formal logic.
  - Functional: It treats computation as the evaluation of mathematical functions: avoids changing-state and mutable data.

    It is a declarative programming paradigm.

Introduction

Introducción



Functional Programming/Language

- Examples
  - Common Lisp
    - https://lisp-lang.org/
  - Scheme
    - https://www.scheme.com/tspl4/
  - Bigloo (dialecto de Scheme)
    - https://www-sop.inria.fr/mimosa/fp/Bigloo/index.html
  - Otros: Ocaml, Erlang, etc.

Actualmente la teoría que fundamenta este paradigma son
Las funciones lambda y el Cálculo lambda

Introduction

Introducción

Declarative o Logic Programming/Language

- Examples
  - Prolog (PROgramming in LOGic)
    - https://www.swi-prolog.org/
  - ASP (Answer Set Programming)
    - https://potassco.org/
  - Otros: pendiente.

Actualmente la teoría que fundamenta este paradigma es La Lógica de Predicados

Introduction

Introducción

Encapsulation

- Abstraction provides a promise that any implementation of the ADT has certain properties and abilities; knowing these is all that is required to make use of an ADT object.
- The user does not need any technical knowledge of how the implementation works to use the ADT.
- In this way, the implementation may be complex but will be encapsulated in a simple interface when it is actually used.

Introduction

Introducción

Operaciones básicas

Para almacenamiento persistente.

Existen varios acrónimos

- CRUD (Create, Read, Update, Delete)
- BREAD (Browse, Read, Edit, Add, Delete)
- ABC (Altas, Bajas, Cambios)

Introduction

Introducción

Según el lenguaje de programación los TAD se pueden usar de 2 formas:

1. Usar una librería que los implementa (aprender el API).
2. Usar las que ofrece el propio lenguaje (los tiene integrados).

En el primer caso existen varias librerías que implementan los TAD en cada lenguaje de programación. Por ejemplo:

- En el lenguaje C++:
  - https://en.wikipedia.org/wiki/Standard_Template_Library
- Para los demás lenguajes se deje que se investigue de forma opcional.

Tarea optativa

- Investigar las librerías más populares en cada lenguaje.

Lists

Listas

Lists

Modelo

Implementaciones: Tipos de listas

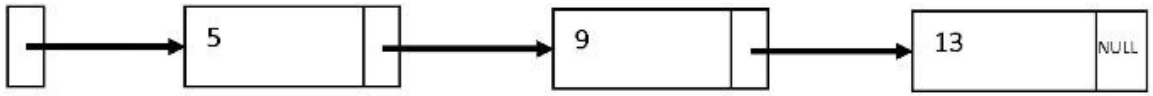
- Estática
  - Con arreglos simples
  - Con arreglos e indices
- Dinámica
  - Sin encabezado
    - Simplemente ligada
    - Doblemente ligada
  - Con encabezado
    - Simplemente ligada
    - Doblemente ligada

Lists

Modelo

Lista = <e_1, e_2, .., e_n>
= <5, 9, 13>
= <a, g, h, ..., g>
= “qwerty”

Representación gráfica

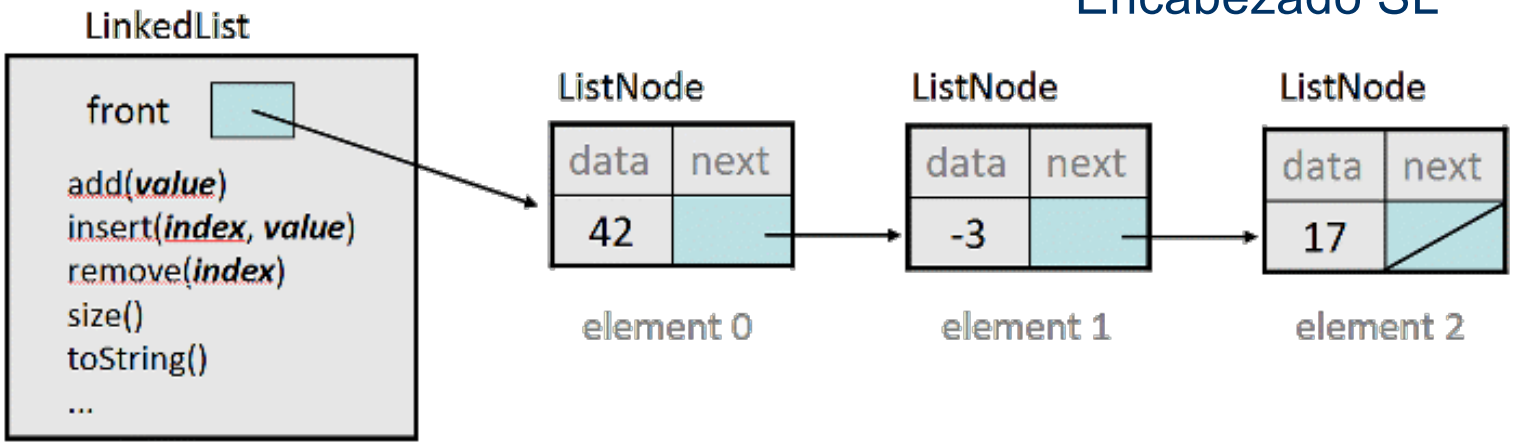



```
struct Node {
   int data;
   Node *nextPtr;
};

Node *head;
```

Implementación simplemente ligada (SL)

Encabezado SL

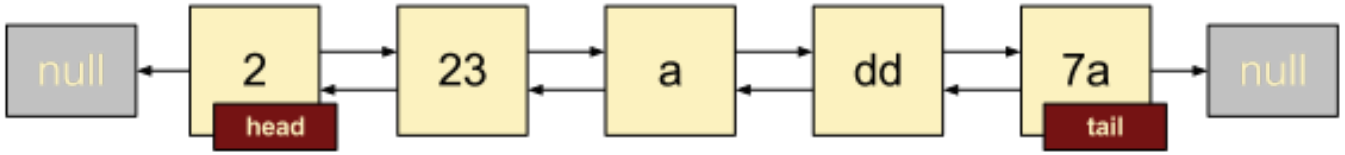


Lists

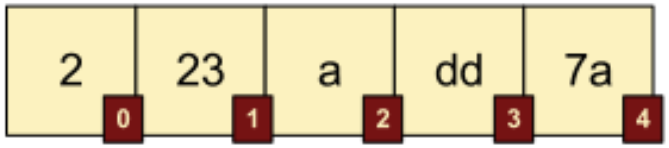
Modelo

Array vs Linked list

Linked list



Array



Operaciones

Hay muchas

Máxima flexibilidad

Ejemplos

- Leer(Posicion, Elemento)
- Agregar(Elemento, Posición)
- Borrar(Elemento, Posición)
- Calcular(tamaño)
- Transformar(cadena, sentido)

Ventajas (arreglo)

- Más rápido el acceso.

Desventajas (arreglo)

- Esta fijo el tamaño y
- Son caras las operaciones
  De agregar, borrar, cambiar el tamaño.

Stacks

Pilas

Stacks

Modelo

Implementaciones: tipos de pilas

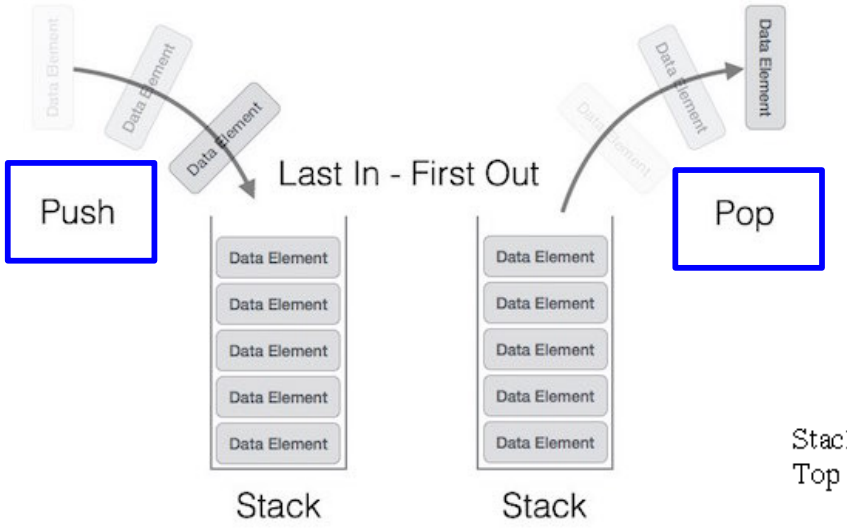
- Estática
  - Arreglo con índice
- Dinámica
  - Sin encabezado
    - Simplemente ligada
  - Con encabezado
    - Simplemente ligada

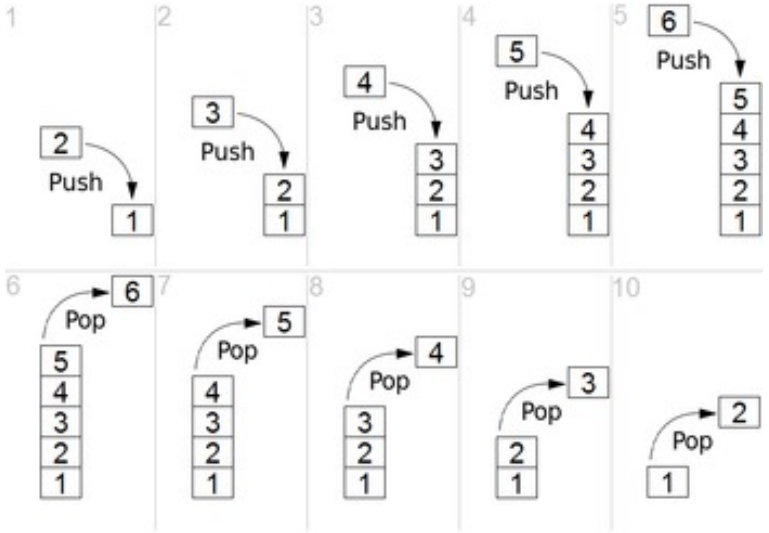
Doblemente ligada. No se utiliza por el tipo de operaciones.

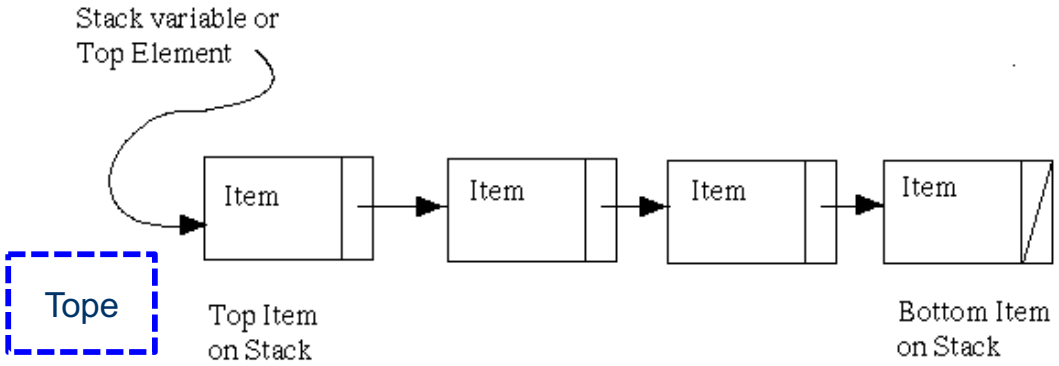
Stacks

Modelo

Representación gráfica







Operaciones

Push
Pop
Tope
Size

Queues

Colas

Queues

Modelo

Implementaciones: Tipos de listas

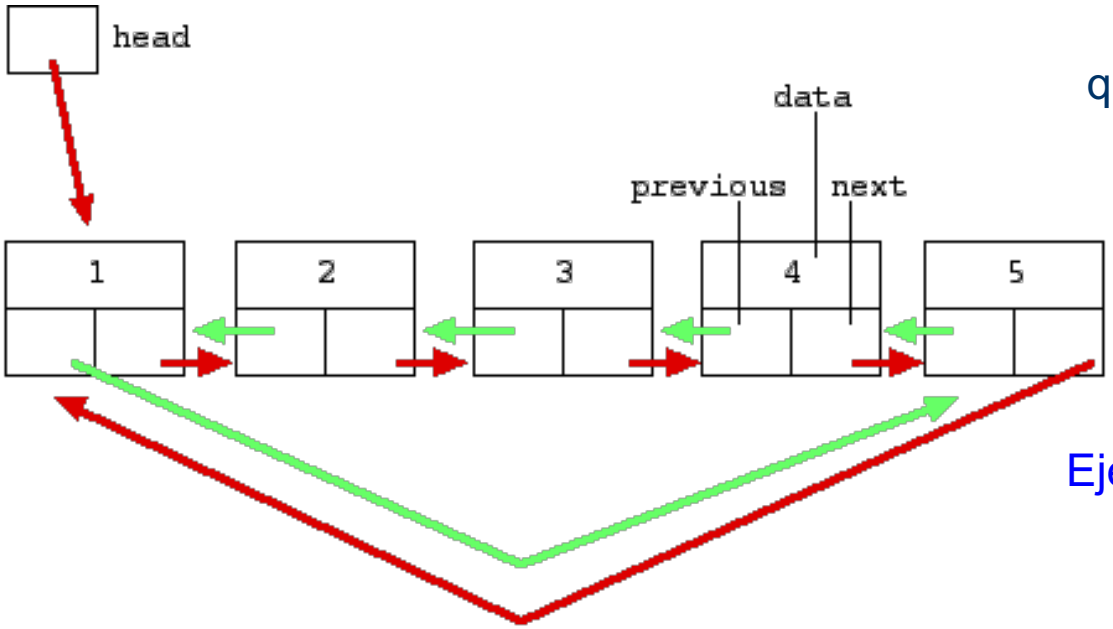
- Estática
  - Arreglo con índice
- Dinámica
  - Sin encabezado
    - Simplemente ligada
    - Doblemente ligada
  - Con encabezado
    - Simplemente ligada
    - Doblemente ligada
  - Circulares

Variantes
- Colas con prioridades

Queues

Modelo

Representación gráfica



Operaciones

Al igual que la pila se restringen la forma en que se manipula el objeto

Es como una **cadena**

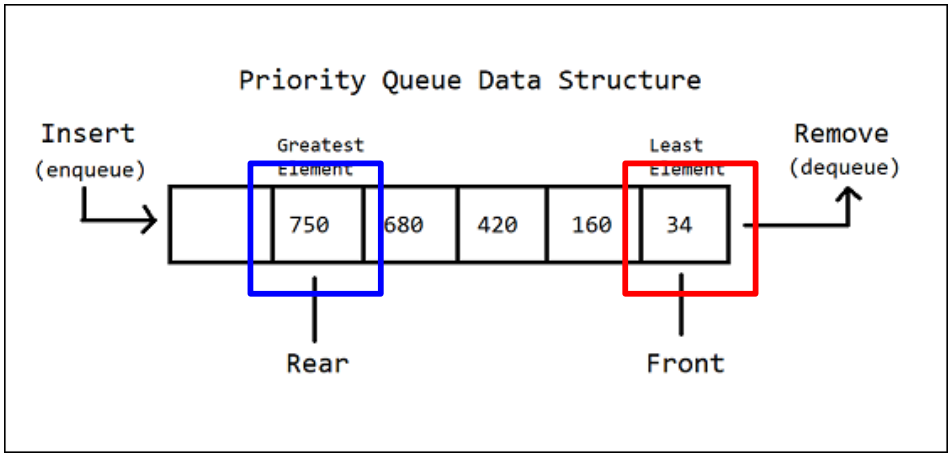
Ejemplos

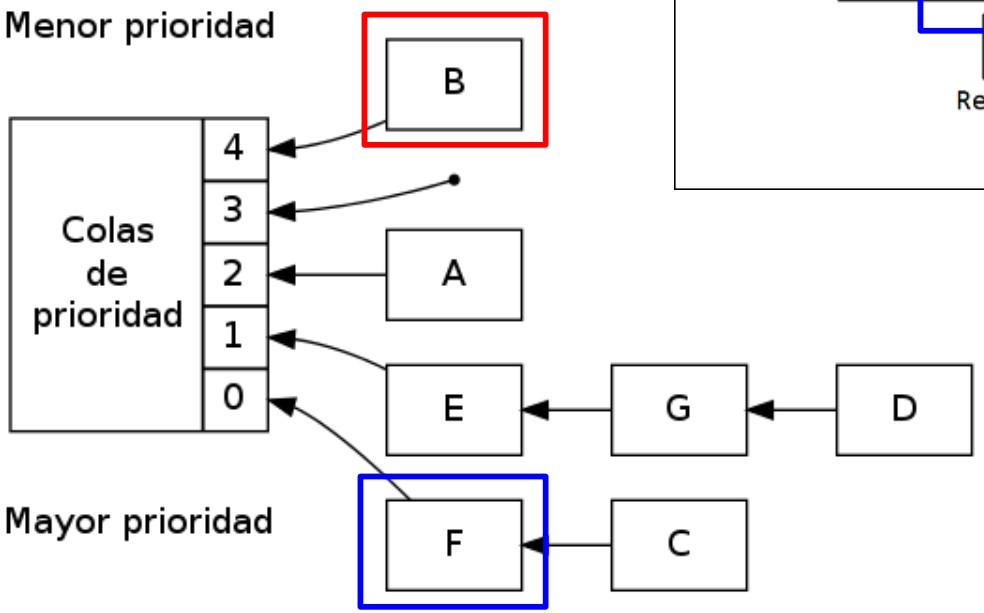
- agregarElemento
- eliminarElemento
- concatenarColas

Queues

Modelo

Cola con prioridades ordenada





El dato mas grande.
El más chico.

Trees

Árboles

Trees

Modelo

Tipos de árboles

- Árbol Binario (búsqueda)
- Árbol N-ario
- Árboles balanceados
  - Árbol Rojo-Negro (https://en.wikipedia.org/wiki/Red-black_tree)
  - Árbol AVL (**A**delson-**V**elsky, **L**andis, https://en.wikipedia.org/wiki/AVL_tree)
  - Árbol AA (**A**rne y **A**ndersson, https://en.wikipedia.org/wiki/AA_tree)
  - Árbol B, Árbol B+, Árbol B* (https://en.wikipedia.org/wiki/B+_tree)
    - Árbol 2-3, Árbol 2-3-4 (https://en.wikipedia.org/wiki/2-3_tree)
- Árbol splay (biselado)

Creados por

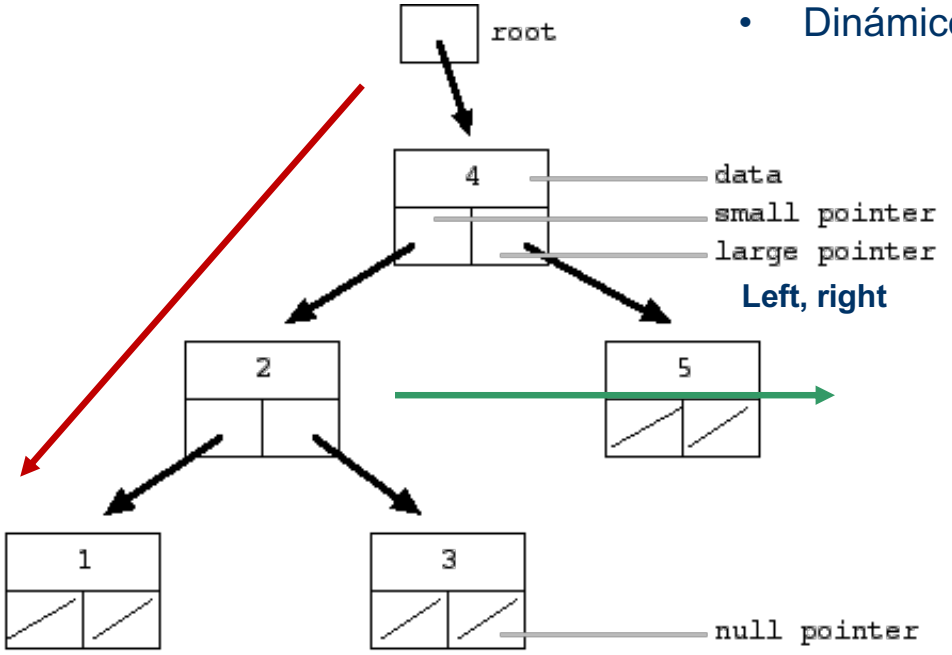
John Hopcroft

Premios
Turing, ACM
Jhon Von Neumann

Trees

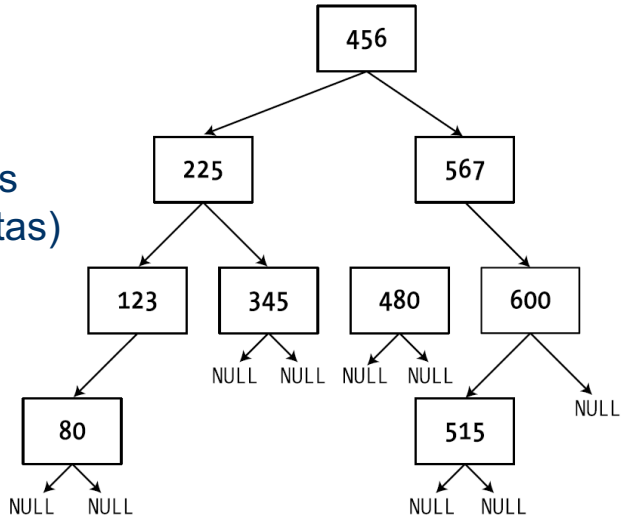
Modelo

Representación gráfica



Implementaciones

- Semi-estáticos: binarios
- Dinámicos: N-arios (listas)



Operaciones

- agregarElemento
- eliminarElemento
- Tamaño, altura, etc.

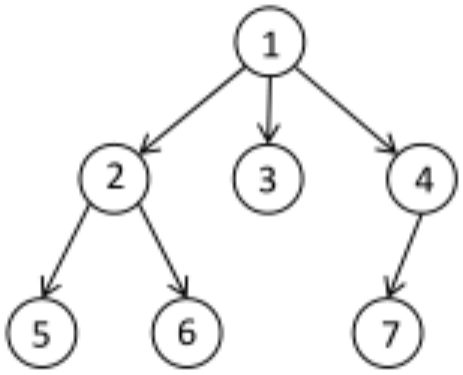
- Recorrido: profundidad, ancho.

Trees

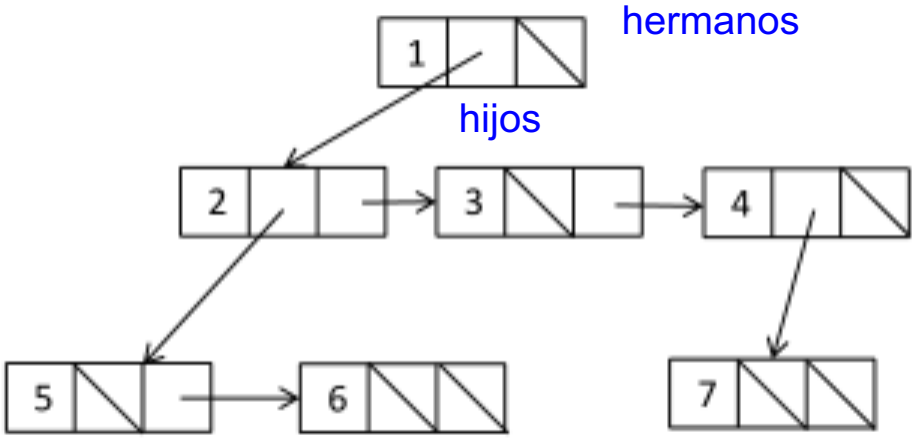
Modelo

Árbol n-ario

Implementación mediante listas.



Tree Structure



Linked List representation of A Tree

Trees

Modelo

Árbol binario Con arreglo o matriz

- Que valor se usa para el valor de NULL?
- Evitar guardar dato y referencia en la misma celda.

Esta implementación se usa mucho en BD.

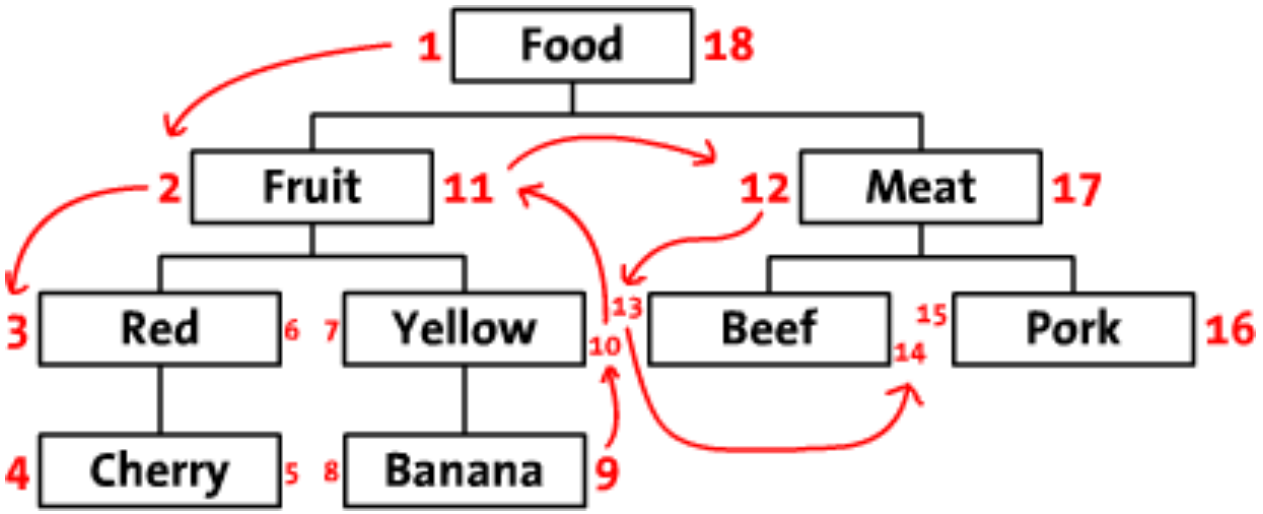
Id	Value	Left	Righ
0			
1	A	2	0 (cero)
2	B	3	4
3	C	5	0
4	D	0	6
5	E	0	0
6	F	0	7
7	G	0	0
…	…	…	…

Trees

Modelo

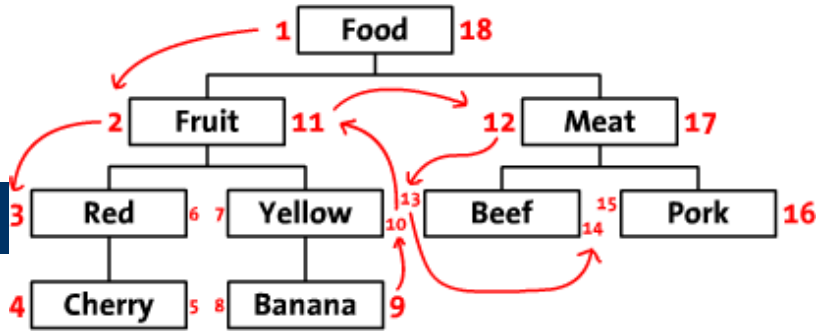
Modified Preorder Tree Traversal (MPTT)

- Implementaciones jerárquicas (árbol) en Bases de datos (tablas).
- Son estrucuturas cuya ventaja es una lectura rápida pero modificaciones lentas.



Trees

Modelo



MPTT

- Como leer sub-árbol Fruit?
- Registros con campo Left entre 2 y 11.
- Como se obtiene el camino a un nodo?

Parent	Title	Left	Righ
	Food	1	18
Food	Fruit	2	11
Fruit	Red	3	6
Red	Cherry	4	5
Fruit	Yellow	7	10
Yellow	Banana	8	9
Food	Meat	12	17
Meat	Beef	13	14
Meat	Pork	15	16

Trees

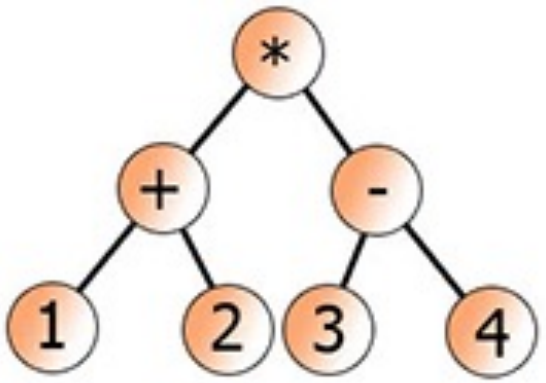
Modelo

Operaciones y Recorridos en árboles

- Notaciones
  - Infija
  - Prefia
  - Postfija.
- Recursividad y sus estados.

Clasic topics

Temas clásicos



((1+2)*(3-4))

Notaciones aritméticas

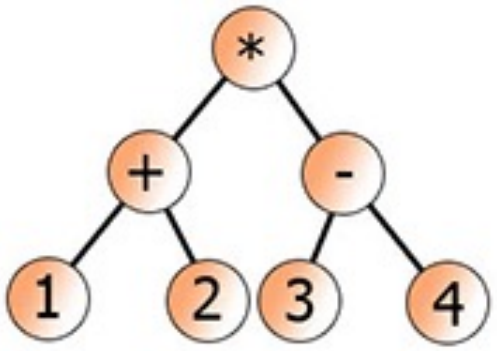
- Notación prefija
  - Operador Operando-1 Operadondo-2 = + A B
- Notación infija
  - Operando-1 Operador Operadondo-2 = A + B
- Notación postfija
  - Operando-1 Operadondo-2 Operador = A B +

- Si se implementan las expresiones con un árbol binario las notaciones se obtienen de un recorrido en profundidad DFS.
- Las notaciones posfija y prefija no necesitan paréntesis para indicar el orden de las operaciones, mientras la aridad del operador sea fija.

Implementación más eficiente (memoria)

Clasic topics

Temas clásicos



Recursive algorithm

```
string DFS(node n)
{
 string l, r;

     If( n==null) return “”;
     l= DFS(n->left);
     r= DFS(n->right);
     return strcat(strcat(l,r), n->value);
}
```

Si se tiene un árbol n-ario se usa un ciclo y una sola invocación (lista)

Variable	Value
node n	
string l	
string r	
uname vars	return
IP register	l= DFS...

Repaso de SO

Conceptos básicos

Proceso: es un programa (sólo el código) en ejecución.

- Significa que está cargado en RAM.
- Hay ciertos valores en los registros del procesador, y
- Las funciones se implementan con la pila.

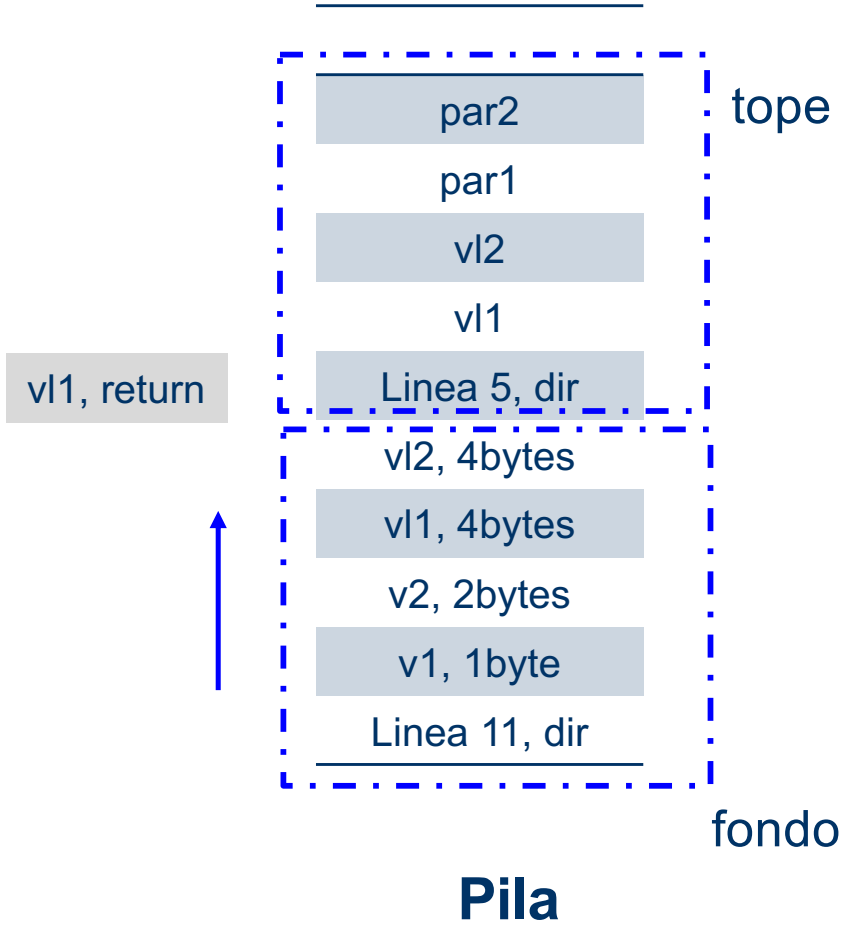
Segmentos Data Pointer (DP)	Datos (Data) Variables globales Heap malloc, new
Instruction Pointer (IP)	Código (Text) Funciones
Stack Pointer (SP)	Pila (Stack) Variables locales Parámetros

Base Pointer
Specific Pointer

Llamadas recursivas

Implementación

```
1   tipo func(char par1, int par2)
2   {
3    double vl1, vl2;
4         if( cond ) salir de la rec.
5         vl1= func(vl1, vl2);
6         return vl1;
7   }
8   main()
9   {
10     tipo var;
11        var = func(v1, v2);
12  }
```



Pila

En lenguaje C

NO se deben regresar variables locales (apuntadores).

Práctica

Modelo de función

Práctica optativa

- Verificar la forma en que se implementan las funciones con una pila:
  - Corroborar los datos de la pila y su orden.
  - Alterar dichos valores de manera indirecta.
  - Antes la pila tenia un tamaño fijo
    - Ahora se puede cambiar el tamaño máximo de la pila (según el compilador)?.
  - Etc.

Clasic topics

Temas clásicos

Notaciones polaca inversa (Reverse Polish Notation, RPN)

- Su nombre viene por analogía con la relacionada notación polaca, una notación de prefijo introducida en 1920 por el matemático polaco Jan Łukasiewicz.
- Es un método algebraico alternativo de introducción de datos donde cada operador está antes de sus operandos. En la notación polaca inversa es al revés.
- El esquema polaco inverso fue propuesto en 1954 por Burks, Warren y Wright y reinventado independientemente por Friedrich L. Bauer y Edsger Dijkstra a principios de los años 1960, para reducir el acceso de la memoria de computadora y para usar el stack para evaluar expresiones.
- La notación y los algoritmos fueron extendidos por el filósofo y científico de la computación australiano Charles Leonard Hamblin a mediados de los años 1960.
- Posteriormente, Hewlett-Packard lo aplicó por primera vez en la calculadora de sobremesa HP-9100A en 1968 y luego en la primera calculadora científica de bolsillo, la HP-35.
- Durante los años 1970 y 1980, el RPN tenía cierto valor incluso entre el público general, pues fue ampliamente usado en las calculadoras de escritorio del tiempo - por ejemplo, las calculadoras de la serie HP-10C.

Advanced topics

Sethi–Ullman algorithm (also known as Sethi–Ullman numbering)

- When generating code for arithmetic expressions, the compiler has to decide which is the best way to translate the expression in terms of number of instructions used as well as number of registers needed to evaluate a certain subtree.
  - Especially in the case that free registers are scarce, the order of evaluation can be important to the length of the generated code, because different orderings may lead to larger or smaller numbers of intermediate values being spilled to memory and then restored.
  - The Sethi–Ullman algorithm fulfills the property of producing code which needs the **fewest** instructions possible as well as the **fewest** storage references.
- NOTE
  - The algorithm succeeds as well if neither commutativity nor associativity hold for the expressions used, and therefore arithmetic transformations can not be applied.
  - The algorithm also does not take advantage of common subexpressions or apply directly to expressions represented as general directed acyclic graphs rather than trees.

Libraries

Librerías

Libraries

Librerias

La implementación de los TADs en cada lenguaje varia de 2 formas principales:

1. Son incluidos en el lenguaje como un tipo básico de dato.
2. Se requiere una librería.

En la mayoría de los lenguajes modernos se usa el concepto de collections para referirse a los diferentes tipos de TADs del tipo Lista, Cola, Pila, etc.

Algunos ejemplos de lo anterior son los siguientes:

1. Lenguaje C++ se usa STL.

Tarea optativa

Investigar como se implementan los TADs en varios lenguajes.

Libraries

Librerias

Conceptos / Modelo

- Namespace

  A namespace is a declarative region that provides a **scope** to the **identifiers** (the names of types, **functions**, **variables**, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

  Namespace in C++: nombre::variable

- SDK

  A **S**oftware **D**evelopment **K**it is a collection of software development tools in one installable package. They facilitate the creation of applications by having a:

  1. Editor
  2. Compiler
  3. Debugger and
  4. Perhaps a software framework.

  They are normally specific to a hardware platform and operating system combination.

Language C

Intro

Referencias

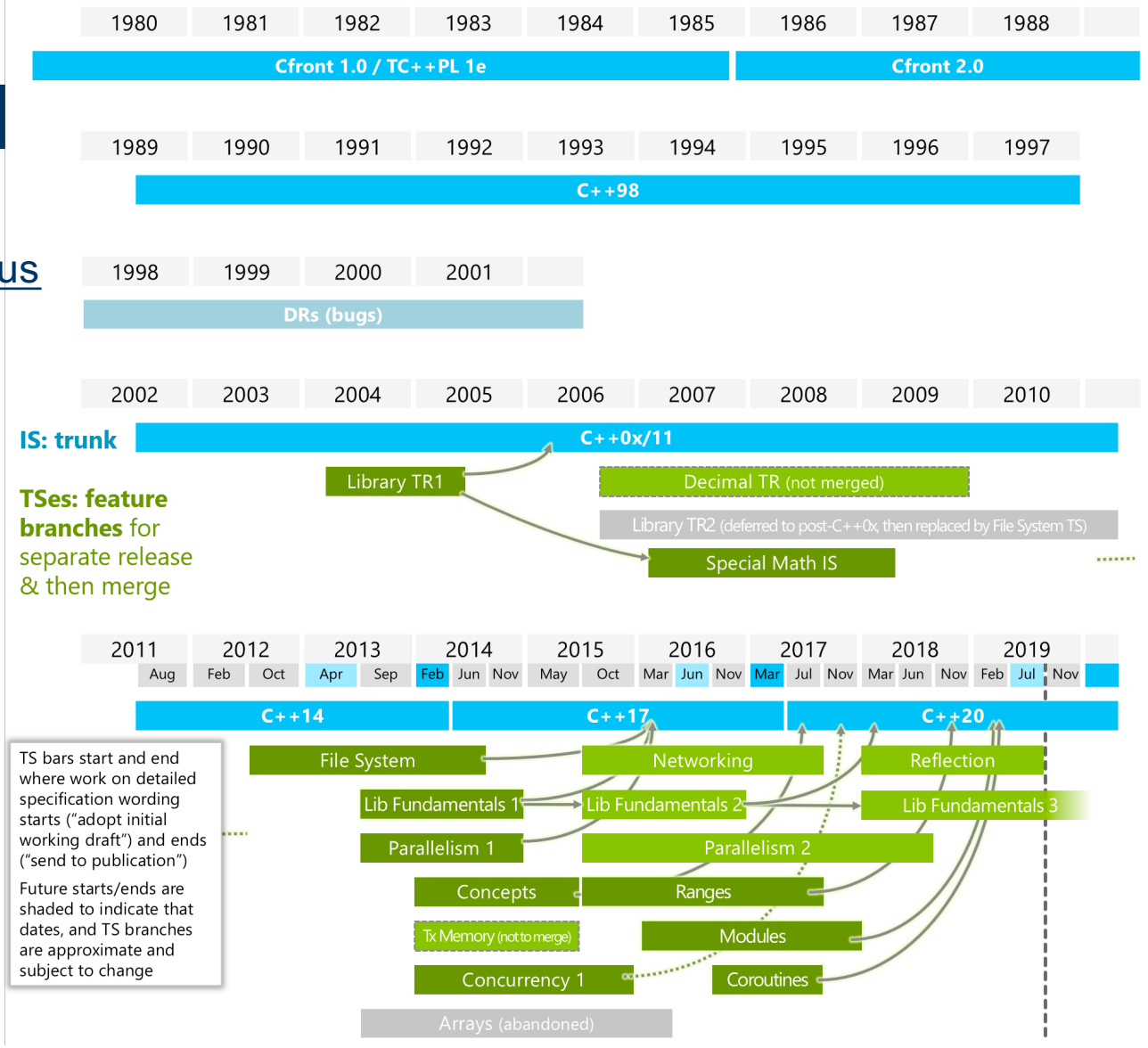
- Estándares del lenguaje conocidos como:
  C89, C90, C99 (*ISO/IEC 9899:1999*), C11, C17, C2X
- La librería principal
  https://www.gnu.org/software/libc/
  El manual en PDF tienen 1,210 páginas en su versión 2.34 del 2021
- TADs en Glibc
  #include <sys/queue.h>
    Referencia: https://linux.die.net/man/3/queue
  Investigar los demás..
- Librerías
  Cada quien implementa sus librerías y algunas las comparten:
  - http://mellowcandle.github.io/liblist/

Language C++

Intro

Referencias

https://isocpp.org/std/status



Language C++

Usando STL

Referencias

- https://isocpp.org/std/status
- https://users.cs.northwestern.edu/~riesbeck/programming/c++/stl-summary.html
  Tutorial básico

The Standard Template Library (**STL**) is a software library for the C++ programming language that influenced many parts of the C++ Standard Library. It provides four components called algorithms, containers, functional, and iterators.

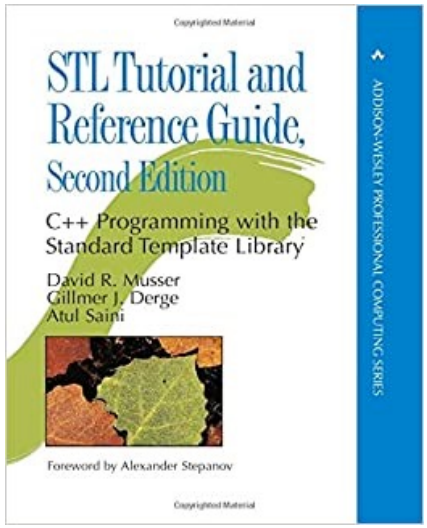
Collections?

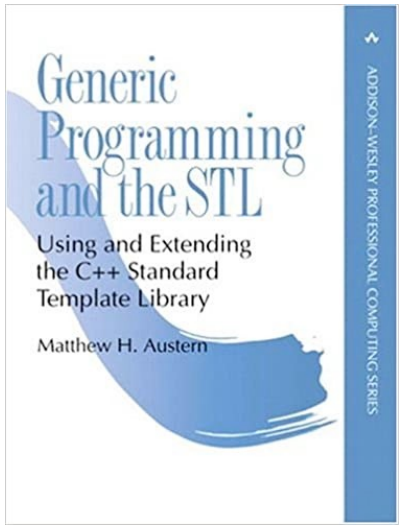
The STL provides a ready-made set of common classes for C++, such as containers and associative arrays, that can be used with any built-in type and with any user-defined type that supports some elementary operations (such as copying and assignment). STL algorithms are independent of containers, which significantly reduces the complexity of the library.

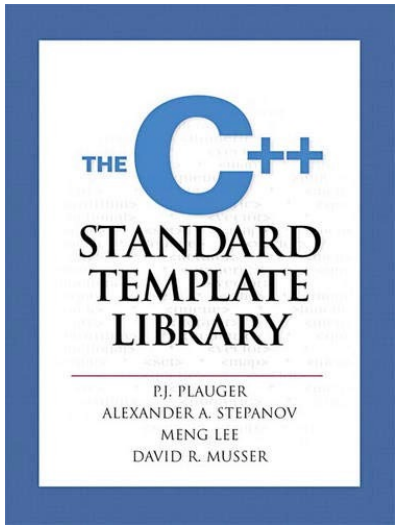
The STL achieves its results through the use of **templates**. This approach provides ***compile-time polymorphism*** that is often more efficient than traditional ***run-time polymorphism***. Modern C++ compilers are tuned to minimize any abstraction penalty arising from heavy use of the STL.

Language C++

Usando STL







Language C++

TADs

		Sequence cont		
Header		<array>	<vector>	<deque>
Container		array	vector	deque
	(constructor)	(implicit)	vector	deque
	(destructor)	(implicit)	~vector	~deque
	operator=	(implicit)	operator=	operator=
	assign		assign	assign
Iterators	begin	begin	begin	begin
	cbegin	cbegin	cbegin	cbegin
	end	end	end	end
	cend	cend	cend	cend
	rbegin	rbegin	rbegin	rbegin
	crbegin	crbegin	crbegin	crbegin
	rend	rend	rend	rend
	crend	crend	crend	crend
Element access	at	at	at	at
	operator[]	operator[]	operator[]	operator[]
	data	data	data	
	front	front	front	front
	back	back	back	back
Capacity	empty	empty	empty	empty
	size	size	size	size
	max_size	max_size	max_size	max_size
	resize		resize	resize
	capacity		capacity	
	reserve		reserve	
	shrink_to_fit		shrink_to_fit	shrink_to_fit
Modifiers	clear		clear	clear
	insert		insert	insert
	insert_or_assign			
	emplace		emplace	emplace
	emplace_hint			
	try_emplace			
	erase		erase	erase
	push_front			push_front
	emplace_front			emplace_front
	pop_front			pop_front
	push_back		push_back	push_back
	emplace_back		emplace_back	emplace_back
	pop_back		pop_back	pop_back
	swap	swap	swap	swap

Referencias

https://en.cppreference.com/w/

Container adaptors

Container adaptors provide a different interface for sequential containers.

stack	adapts a container to provide stack (LIFO data structure) (class template)
queue	adapts a container to provide queue (FIFO data structure) (class template)
priority_queue	adapts a container to provide priority queue (class template)

Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

array (C++11)	static contiguous array (class template)
vector	dynamic contiguous array (class template)
deque	double-ended queue (class template)
forward_list (C++11)	singly-linked list (class template)
list	doubly-linked list (class template)

Associative containers

Associative containers implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).

set	collection of unique keys, sorted by keys (class template)
map	collection of key-value pairs, sorted by keys, keys are unique (class template)
multiset	collection of keys, sorted by keys (class template)
multimap	collection of key-value pairs, sorted by keys (class template)

Unordered associative containers

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched ($O(1)$ amortized, $O(n)$ worst-case complexity).

unordered_set (C++11)	collection of unique keys, hashed by keys (class template)
unordered_map (C++11)	collection of key-value pairs, hashed by keys, keys are unique (class template)
unordered_multiset (C++11)	collection of keys, hashed by keys (class template)
unordered_multimap (C++11)	collection of key-value pairs, hashed by keys (class template)

Language C++

TADs

Librerias de uso especifico (varias incluyen TADs)

- **Boost** (https://www.boost.org/)

  It is a set of well-recognized C++ libraries, by various authors, that contain packages of code for the purpose of supporting linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, string and text processing, and unit testing.

  V/D. It is a large and complex library with a steep learning curve. Linking to Boost will also increase your executable size.

- **Poco** (https://pocoproject.org/)

  They are powerful cross-platform C++ libraries for building network- and internet-based applications that run on desktop, server, mobile, IoT, and embedded systems.

- Larga lista de librerías

  OpenSSL (criptografía), FFmpeg (multimedia – audio/video), SQLite (base de datos)
  Mosquito (MQTT broker), Tensorflow (machine learning)

  OpenCV (real-time computer vision, machine learning, and deep learning library for face recognition, object detection, and 3-D model extraction)

Language C#

Intro y TADs

Referencias

- https://www.ecma-international.org/publications-and-standards/standards/ecma-334/
- https://www.tutorialsteacher.com/csharp/csharp-collection

`System.Collections.Generic` namespace.

Generic Collections	Description
List<T>	Generic List<T> contains elements of specified type. It grows automatically as you add elements in it.
Dictionary<TKey,TValue>	Dictionary<TKey,TValue> contains key-value pairs.
SortedList<TKey,TValue>	SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default.
Queue<T>	Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection.
Stack<T>	Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values.
Hashset<T>	Hashset<T> contains non-duplicate elements. It eliminates duplicate elements.

Hash

Tablas hash

Funciones hash

Funciones Picadillo.

Hash

Modelo

The original form of storing data

Array

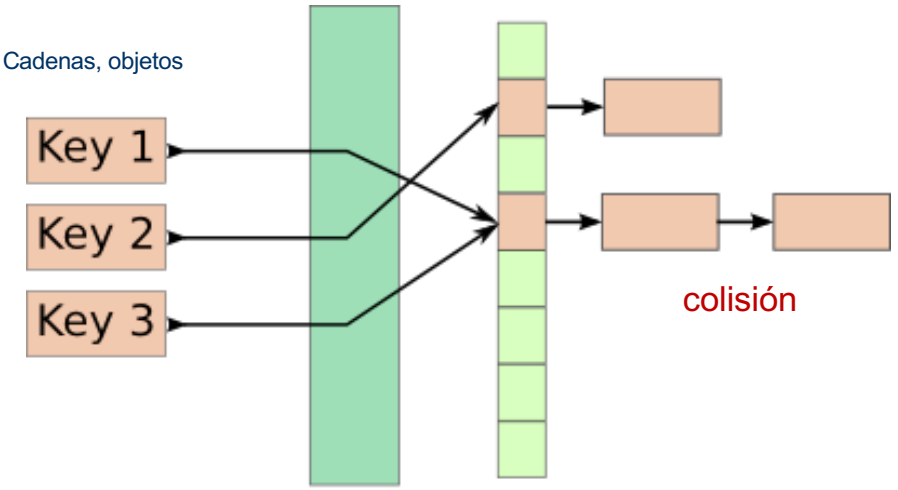
Index	Value
0	New York
1	Boston
...	Mexico
	Kansas
	Detroit
	California

O(1)

Hash Table

Key	Value
1	New York
2	Boston
3	Mexico
4	Kansas
5	Detroit
6	California

The new form where we generaliza key



Hash function

Ideally, the hash function will assign each key to a unique bucket, but it is possible that two keys will generate an identical hash causing both keys to point to the same bucket. Instead, most hash table designs assume that hash **collisions** - different keys that are assigned by the hash function to the same bucket - will occur and must be accommodated in some way (**list**).

f(key) = index

$n^{key\_long} > size(array)$

$27+10=37^{64} > 2^{64}$

Letras + números

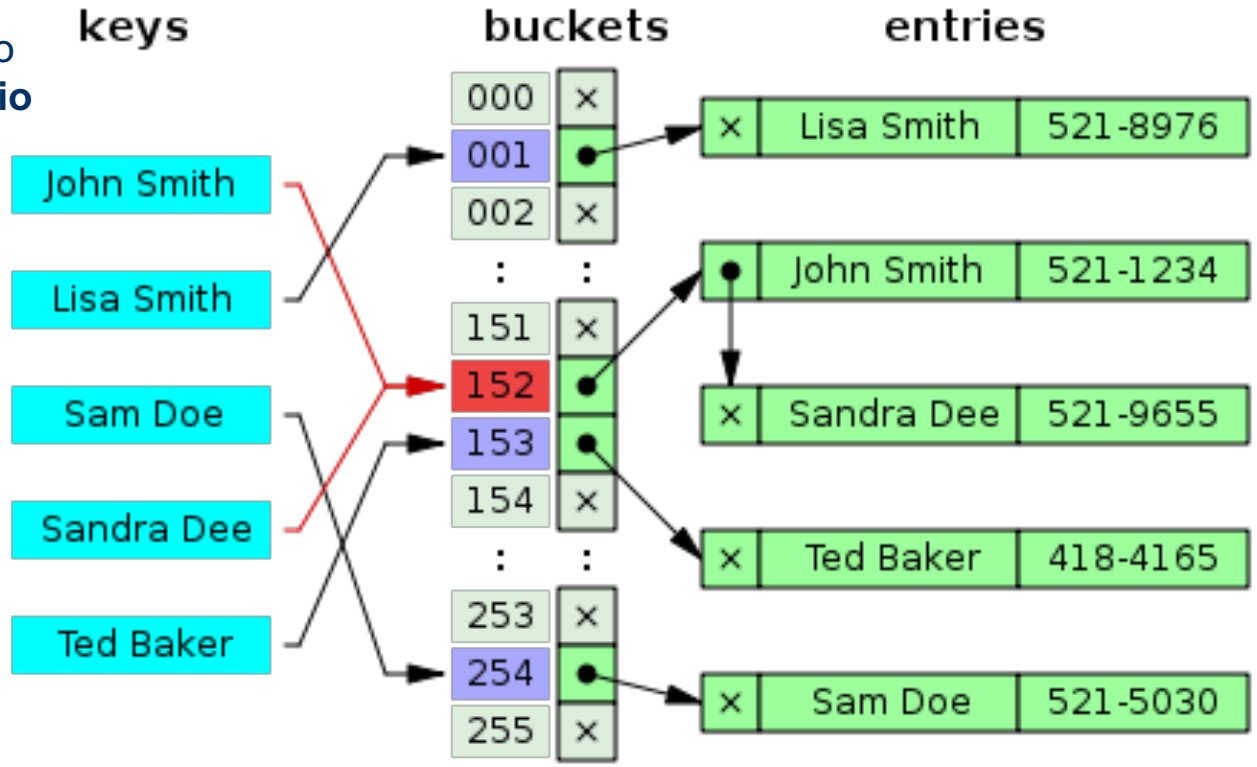
Hash

Modelo

Separate Chaining

- https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/

Hash collision resolved by separating chaining



Uso típico **diccionario**

$$O(1 + \frac{n}{k})$$

where $\frac{n}{k}$ is the load factor

Critical operations resizing

La función puede verse como algo que distribuye/dispersión.

Hash

Modelo

$$\frac{A}{B} = Q\ remainder\ R$$

Hash functions

$$A\ mod\ B = R$$

- Where keys are strings
- Converting to integers and divide (modulo) by a large number:
  - Lizzie McGuirre => 3,223 % 1000 = 3 remainder 0.223.
  - Index of array is 3.
- Modulo
  - n % M where M is the size of the array.
- For cryptography
  - $h(x) = a^x\ mod\ n$

Varios ejemplos de funciones hash

FNV-1 (**Fowler–Noll–Vo**)

It is a non-cryptographic hash function created by Glenn Fowler, Landon Curt Noll, & Kiem-Phong Vo.

dispersión

Cryptographic hash function

MD5, SHA1, SHA256 / 5 propiedades

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

Único

Tarea obligatoria

- Investigar las propiedades y formas típicas de la función hash según su uso.

Heap

Montículos

Heap

Modelo

- A **heap** is a specialized **tree**-based data structure that satisfies the *heap property:*

  If A is a parent node of B then the key of node A is ordered with respect to the key of node B with the same ordering applying across the heap

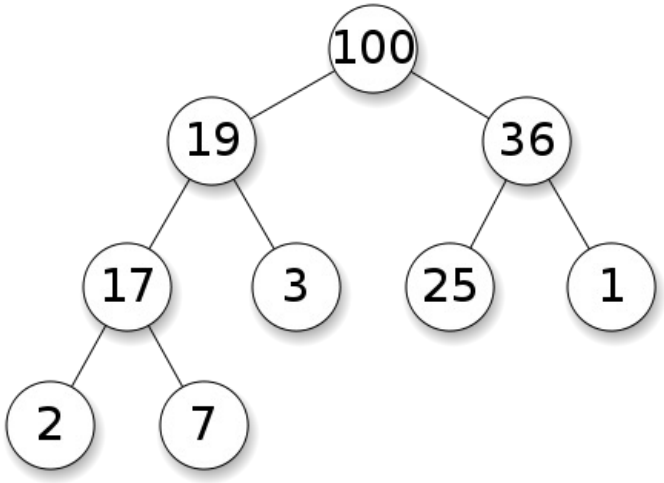
- A heap can be classified further as either:

Max heap.

The keys of parent nodes are always greater than or equal to those of the children and the highest key is in the root node.

Min heap.

The keys of parent nodes are less than or equal to those of the children and the lowest key is in the root node.



Max-heap

Heap

Variantes

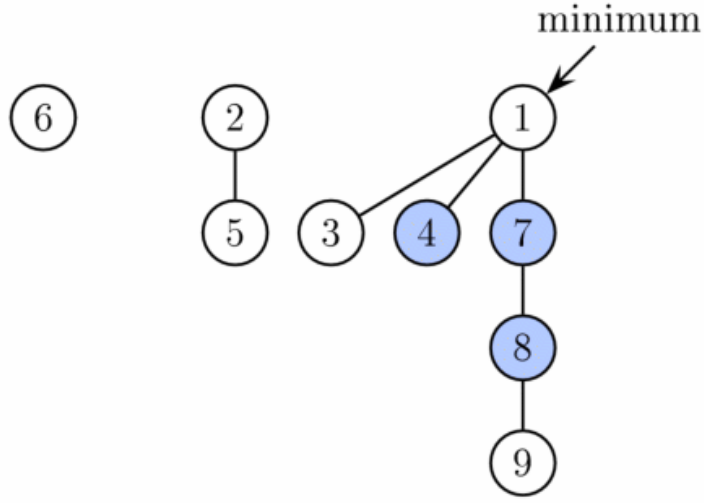
Variants

- Binary heap
- B-heap.
- Binomail heap
- Radix heap
- Fibonnaci heap
- 2-3 heap
- Etc.

A Fibonacci heap is a collection of trees satisfying the minimum-heap property.

Every node has degree at most O(log n) and the size of a subtree rooted in a node of degree k is at least F_k+2, where F_k is the kth Fibonacci number.

Sucesión de Fibonacci
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...



Ejemplo de f. heap que tiene:

- 3 árboles.
- De grados 0,1, y 3.

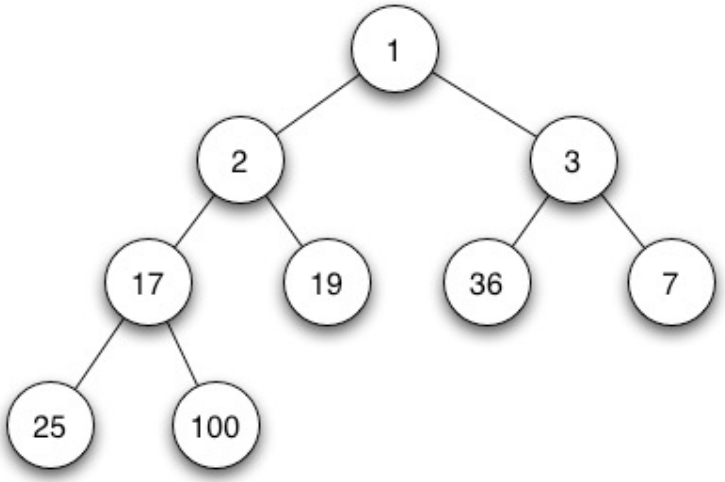
0 1 3

Heap

Modelo

- A common implementation of a heap is the **Binary heap**, in which the tree is a **complete** binary tree.
- A heap is **not a sorted structure** and can be regarded as partially ordered Previous example.
- When a heap is a complete binary tree, it has a smallest possible height - a heap with N nodes always has log N height.
- A heap is a useful data structure when you need **to remove** the object with the highest (or lowest) priority.

Min-heap



Heap

Operaciones

Basics

- find-max or find-min

  Find the maximum item of a max-heap or a minimum item of a min-heap (a.k.a. peek).

- Insert

  Adding a new key to the heap (a.k.a., push).

- delete-max or delete-min

  Removing the root node of a max- or min-heap, respectively.

- Replace

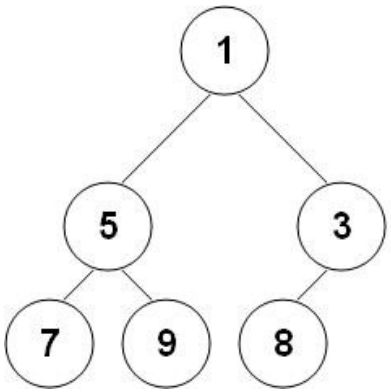
  Pop root and push a new key. More efficient than pop followed by push.

Heap

Implementación

- Heaps are commonly implemented with an array.
- Any binary tree can be stored in an array, but because a binary heap is always a complete binary tree, it can be stored compactly.
- No space is required for pointers; instead, the parent and children of each node can be found by arithmetic on array indices:

  Left=2i+1, Right=2i+2, Parent= floor[(i-1)/2]



Node	1	5	3	7	9	8
Index	0	1	2	3	4	5

Ej: nodo(5) e ind(1), Lef(2i+1=3), Right(2i+2=4)

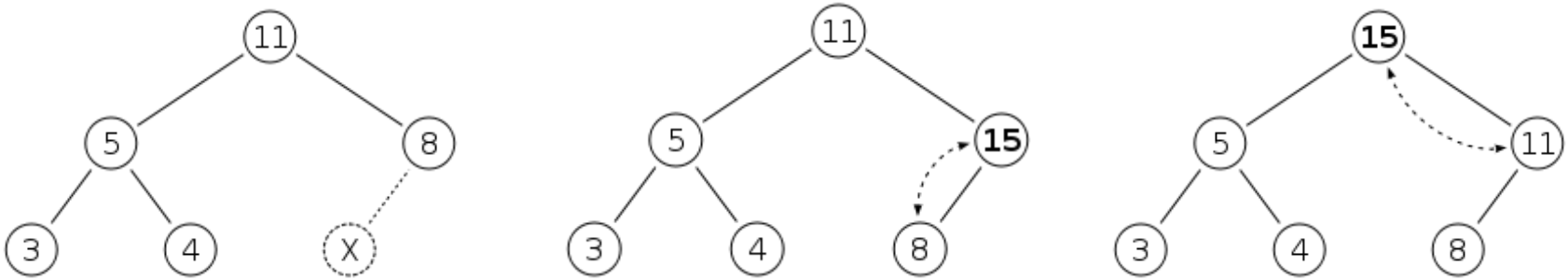
Investigar
Ahnentafel list.

Heap

Operaciones

Operation **insert**

- Adding a new key to the heap (a.k.a., push).
- Algorithm
  - Add a new element to the end of an array.
  - Sift up the new element, while heap property is broken.
  - Sifting is done as following: compare node's value with parent's value. If they are wrong order, swap them.



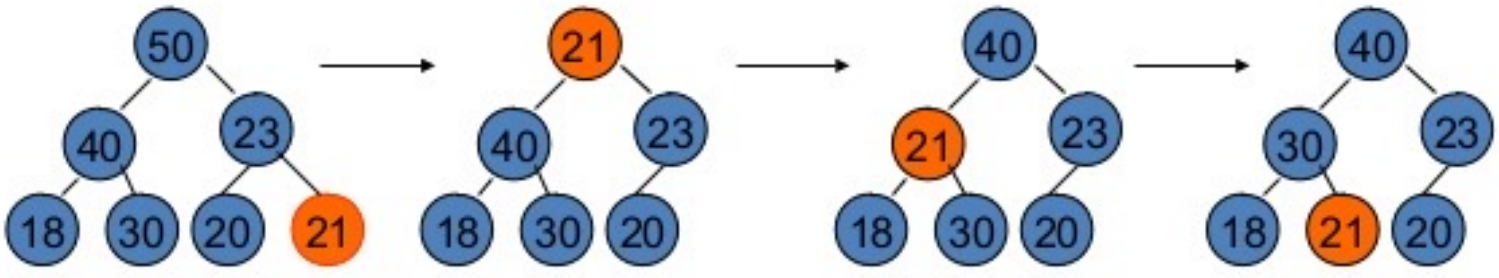
Insert (15) in a max-heap

Heap

Operaciones

Operation **delete-max**

- Removing the root node of a max- or min-heap, respectively.
- Algorithm
  - Replace the root node by the last node at bottom level maintaining completeness.
  - While (new root < its children) exchange it with the greater of its children.



Eliminamos 50 Remplaza 21 Reubicamos 21

Heap

Operaciones

Complexity

Binary heap:

- find-max or find-min: $\theta(1)$
- insert: $O(\log n)$
- delete-max or delete-min: $\theta(\log n)$

Referencias

- https://en.wikipedia.org/wiki/Heap_(data_structure).
- https://www.geeksforgeeks.org/heap-data-structure/
- https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

Disjoint Set - DAT

Conjuntos disjuntos / TADs

Implementación

Aplicaciones

Disjoint set (DAT)

TAD de conjuntos Disjuntos

Definición

- Es un TAD que mantiene un conjunto de elementos particionados en un número de conjuntos disjuntos (no se solapan los conjuntos).
  - Tambien se le conoce como: union–find, merge–find set.
- El algoritmo union-find realiza dos importantes operaciones en esta estructura de datos:
  - *Buscar*: Determina a cual subconjunto pertenece un elemento. Esta operación puede usarse para verificar si dos elementos están en el mismo conjunto.
  - *Union*: Une dos subconjuntos en uno solo.
  - La operación básica es *CrearConjunto* que crea el conjunto con un elemento dado. Implementación trivial.
- Links
  - https://en.wikipedia.org/wiki/Disjoint-set_data_structure
  - https://es.wikipedia.org/wiki/Estructura_de_datos_para_conjuntos_disjuntos

Disjoint set (DAT)

Conjuntos Disjuntos (CD)

Implementaciones

- Arreglo de CD
- Listas Enlazadas de CD
- Bosque de CD (*disjoint-set forest*)
  - Son estructuras de datos donde cada conjunto está representado por un árbol.

Aplicaciones

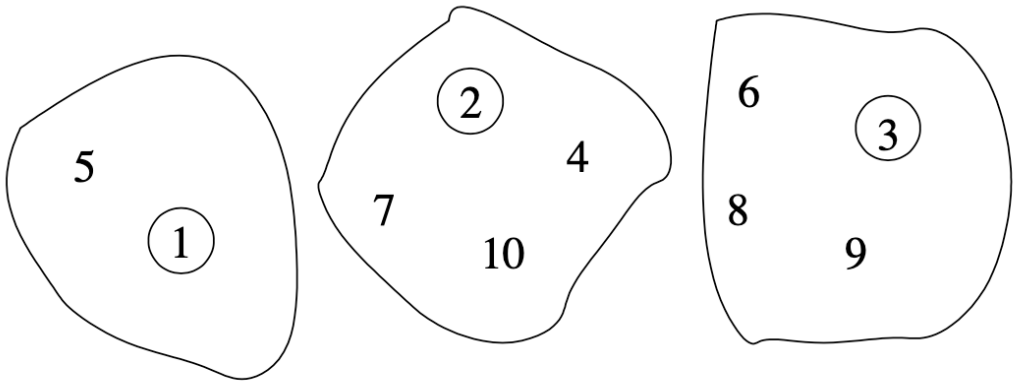
- La principal: Algoritmo de Kruskal para encontrar el MST.
- Otras son: computación simbólica, compiladores, y problemas de asignación de registros.

Disjoint set (DAT)

Conjuntos Disjuntos

Arreglo de CD

- Todos los elementos se numeran de 1 a *n*.
- Cada subconjunto toma su nombre de uno de sus elementos, lo va representar, p. ej. el valor más pequeño.
- Se mantiene en un vector el nombre del subconjunto disjunto de cada elemento.



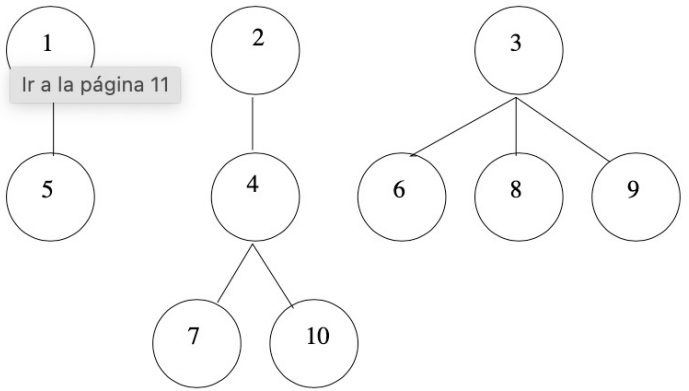
	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	2	3	3	2

Disjoint set (DAT)

Conjuntos Disjuntos

Árbol de CD

- Se utiliza un árbol para caracterizar cada subconjunto. La raíz nombra al subconjunto.
- La representación de los árboles es fácil porque la única información necesaria es un apuntador al padre. Cada entrada *p[i]* en el vector contiene el padre del elemento *i*.



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

Disjoint set (DAT)

Conjuntos Disjuntos

Temas avanzados

- Unión por alturas.
- Compresión de caminos.
- Uso de heurísticas
- Análisis de la complejidad

Artículos

- Estructuras de Datos para conjuntos disjuntos.
  - Santiago Zanella, 2008.

Independent Set

Conjuntos Independientes

Implementación

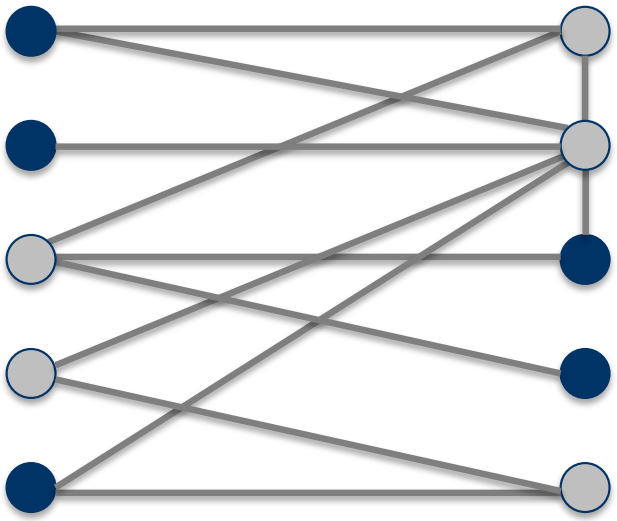
Aplicaciones

Independent set

Conjuntos Independientes

Definition

- A subset $S \subseteq V$ is independent if for every $(u, v) \in E$, either $u \notin S$ or $v \notin S$ (or both). No hay aristas entre los nodos del conjunto.
- *Problem*. Given a graph $G = (V, E)$, find a *Max Cardinality Independent set*. It is a set of vertices in a graph, where no two of which are adjacent.



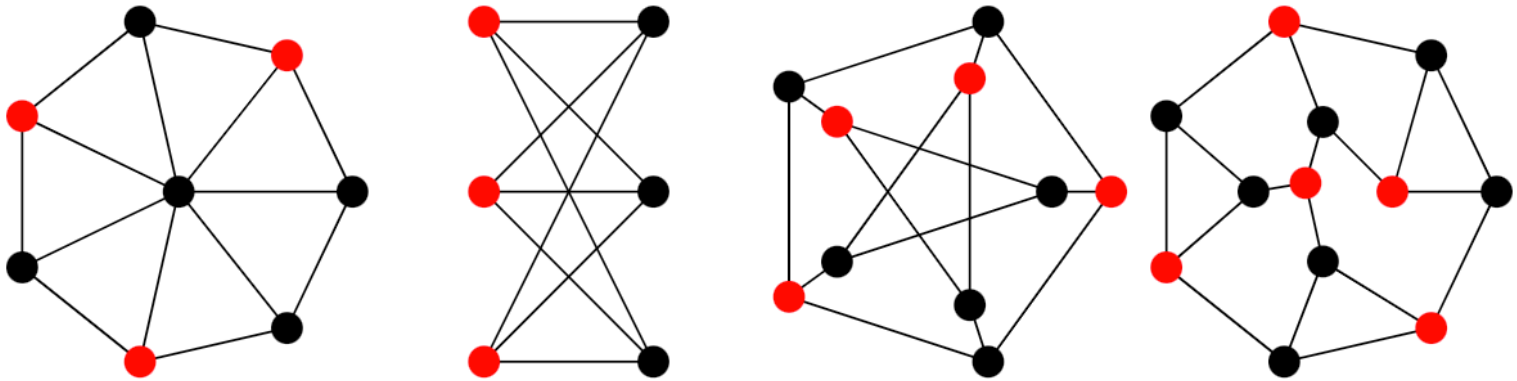
independent set

Independent set

Conjuntos Independientes

Los nodos rojos son el conjunto independiente.

Independent set



Wheel graph W_8, Utility graph $K_{3,3}$, Petersen graph, Frucht graph

A **Maximal Independent Set** (MIS) is an independent set which is a *maximal set*, i.e., an independent set that is not a subset of any other independent set.

La variante más compleja es encontrar todos los MIS.

Independent set

Conjuntos Independientes

Independent set

Definitions and properties

- Two sets A and B are said to be independent if their intersection is empty: $A \cap B = \emptyset$.

  For example, {A,B,C} and {D,E} are independent, but {A,B,C} and {C,D,E} are not.

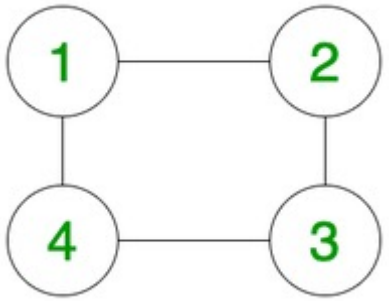
- Independent sets are also called disjoint or mutually exclusive.

Referencias

- https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/ballardmyer.pdf
- https://ac.informatik.uni-freiburg.de/teaching/ss_12/netalg/lectures/chapter6.pdf
- https://www.geeksforgeeks.org/maximal-independent-set-from-a-given-graph-using-backtracking/

Independent set

Conjuntos Independientes



{ }{ 1 }{ 1 3 }{ 2 }{ 2 4 }{ 3 }{ 4 }
{ 1 3 }{ 2 4 }

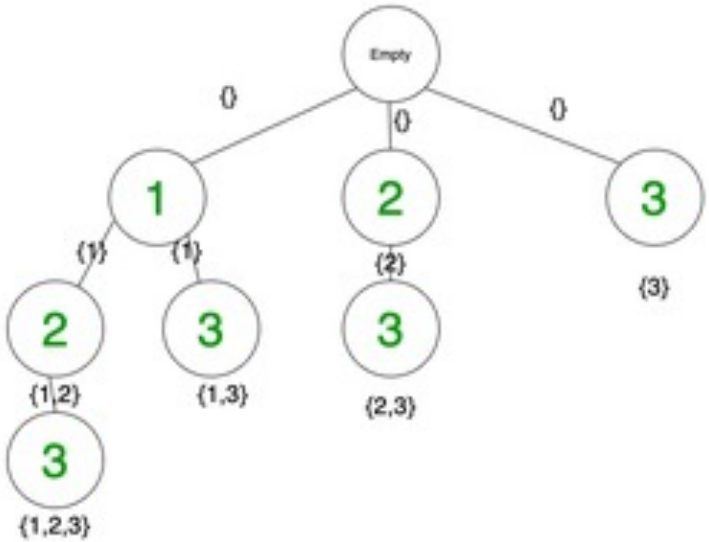
Solutions to MIS

Two main techniques or paradigms:

1. Backtracking

   At every step, we need to check whether the current node has any direct edge with any of the nodes already present in our independent set. If not, we can add it to our independent set and recursively repeat the same process for all the nodes

2. Greedy



Complejidad

- Time Complexity: $O(2^N)$
- Auxiliary Space: $O(2^N)$
- It is NP complete.

Independent set

Conjuntos Independientes

Solutions to MIS

Referencias

1. https://www.dharwadker.org/independent_set/
2. https://karlsruhemis.github.io/

Leer variantes

- https://www.tutorialspoint.com/graph_theory/graph_theory_independent_sets.htm
  - Independent Line Set
  - Independent Vertex Set

Leer artículos

- https://www.sciencedirect.com/science/article/pii/S0890540117300950

Balanced Trees

Árboles balanceados

Balanced Trees

Introducción

Tipos de árboles balanceados

- Árbol Rojo-Negro
- Árbol AVL
- Árbol AA
- Árbol B
  - Árbol B+, Árbol B*
- Árbol 2-3
  - Árbol 2-3-4
- Árbol splay (biselado)

Balanced Trees

Introducción

Árbol Rojo-Negro

- Referencias
  - https://en.wikipedia.org/wiki/Red-black_tree

Balanced Trees

Introducción

Árbol AVL

- Creados por (**A**delson-**V**elsky, **L**andis)
- Referencias
  - https://en.wikipedia.org/wiki/AVL_tree)

Balanced Trees

Introducción

Árbol AA

- Creados por **A**rne y **A**ndersson.
- Referencias
  - https://en.wikipedia.org/wiki/AA_tree

Balanced Trees

Introducción

Árbol B

- Variantes
  - Árbol B+, Árbol B*
- Referencias
  - (https://en.wikipedia.org/wiki/B+_tree)

Balanced Trees

Introducción

Premios
Turing, ACM
Jhon Von Neumann

Árbol 2-3, Árbol 2-3-4

- Creados por John Hopcroft
- Referencias
  - https://en.wikipedia.org/wiki/2-3_tree

Balanced Trees

Introducción

Árbol splay (biselado)

- Referencias

Trie (tree)

Árboles lexicográficos

Vs Diccionarios

Trie

Introducción

- El nombre de *trie* proviene de usar las letras centrales de la palabra "*retrieval*", recuperación (de información), y se pronuncia como "*try*" para distinguirlo del sonido de "*tree*".
- Un árbol lexicográfico o, en general, un *trie*, es un **árbol de prefijos**.
  - Es decir, los prefijos comunes a un subconjunto de claves se almacenan únicamente una vez.
  - Dicho de otra forma, **todas las hojas descendientes de un cierto nodo interno *n*, representan claves que tienen un prefijo común**, y ese prefijo sólo se almacena una vez.
- Una consecuencia de almacenar una única vez cada prefijo común a un subconjunto de claves es que **los *tries* requieren menos espacio para almacenar un conjunto de claves** que las otras soluciones.

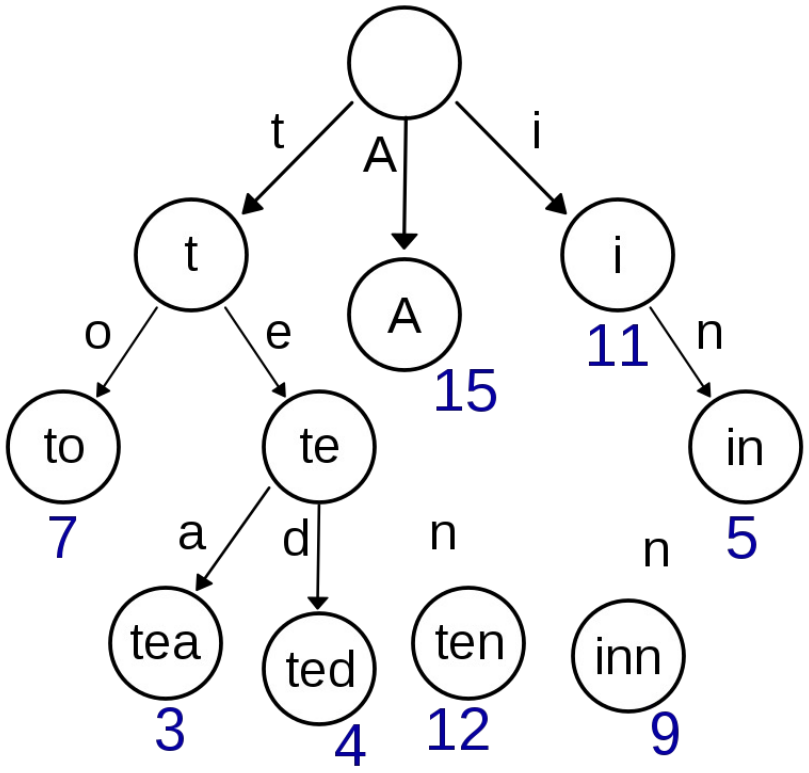
Trie

Ejemplo

A trie for keys:

"A", "to", "tea", "ted", "ten", "i", "in", and "inn".

Each complete English word has an arbitrary integer value associated with it.



Trie

Operaciones

A trie supports three fundamental operations:

- **FIND(*k*)** Determine whether *k* is present in the trie. Depending on the application of the ADT, **FIND** may either simply indicate whether or not *k* is present, or it may return a value *v* associated with the key *k*.

- **INSERT(*k*, *v*)** Add the key *k* to the trie. Depending on the application, a value *v* may or may not be supplied.

- **DELETE(*k*)** Remove the key *k* from the trie.

Trie

Definición Wikipedia

- A **trie**, also called **digital tree** or **prefix tree**, is a type of search tree, a tree data structure used for locating specific keys from within a set.
  - These keys are most often strings, with links between nodes defined not by the entire key, but by individual characters. In order to access a key (to recover its value, change it, or remove it).
  - The trie is traversed depth-first, following the links between nodes, which represent each character in the key.
- Unlike a binary search tree, nodes in the trie do not store their associated key.
  - Instead, a node's position in the trie defines the key with which it is associated. This distributes the value of each key across the data structure, and means that not every node necessarily has an associated value.
- All the children of a node have a common prefix of the string associated with that parent node, and the root is associated with the empty string. This task of storing data accessible by its prefix can be accomplished in a memory-optimized way by employing a radix tree.
- Though tries can be keyed by character strings, they need not be.
  - The same algorithms can be adapted for ordered lists of any underlying type, e.g. permutations of digits or shapes.
  - In particular, a **bitwise trie** is keyed on the individual bits making up a piece of fixed-length binary data, such as an integer or memory address.

Trie

Introducción

The notion of strings used usually is more general than the definition of strings in C.

> A *string* is defined to be to be a finite, ordered sequence of elements called *characters* of a finite set (called the *alphabet*).

However, other kinds of strings are quite useful and common (in the context of tries), but also in other contexts:

- In C, for example, the alphabet is the set of all possible chars, with the exception of the char 0, which is used to denote the end (but is not considered part of the string).
- Strings of bits, which have an alphabet of, and each character is a single bit.
- Strings of decimal numbers are often useful.

Treap

Árbol binario de búsqueda aleatorio

Treap

Modelo

- A **treap** is a *:*

- Implementación de Conjuntos ordenados con treaps.

Referencias

- https://es.wikipedia.org/wiki/Treap
- https://www.cs.cornell.edu/courses/cs312/2003sp/lectures/lec26.html

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx

http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx

racosta@cic.ipn.mx

57-29-60-00

Ext. 56652