ESTRUCTURAS DE DATOS AVANZADAS

Aránzazu Jurío

ALGORITMIA

2018/2019

Índice

- Repaso. Estructuras de datos básicas
- Grafos
  - Repaso
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Montículos

Índice

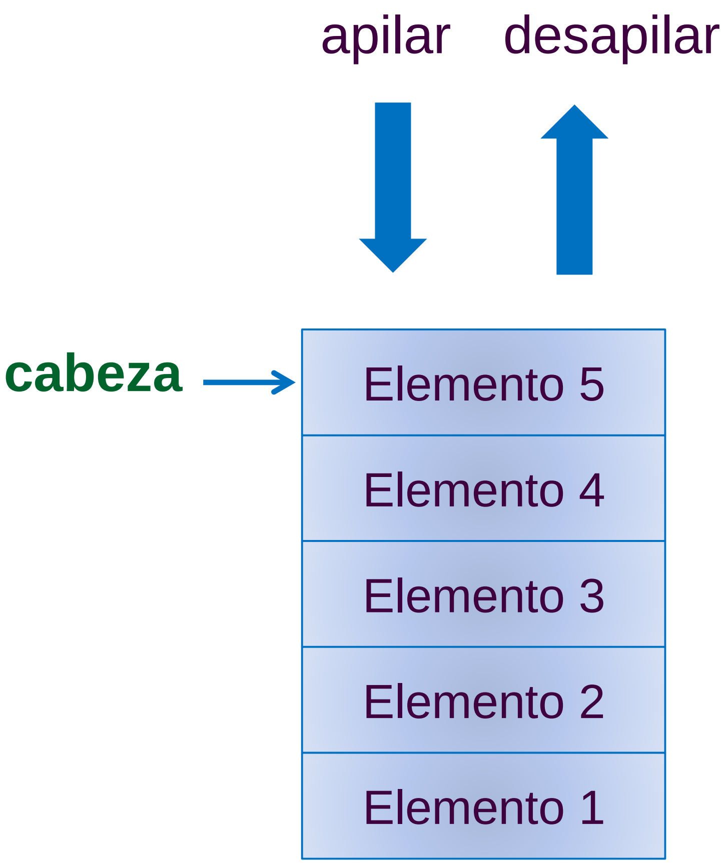
- Repaso. Estructuras de datos básicas
- Grafos
  - Repaso
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Montículos

Pilas

- Estructura LIFO (Last In First Out)

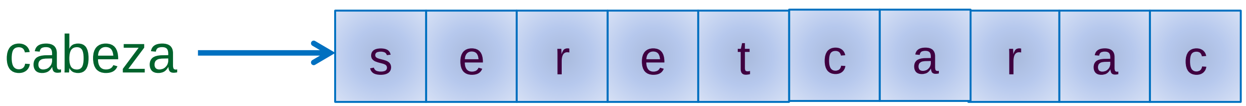
Pilas

- Apilar
  - Añade un elemento en la cabeza de la pila
- Desapilar
  - Elimina un elemento de la cabeza de la pila
- Cima
  - Muestra el elemento en la cabeza de la pila
- ¿Está vacía?
  - Comprueba si la pila contiene elementos o no
- Vaciar
  - Eliminan uno a uno todos los elementos de la pila



Pilas

- Aplicaciones:
  - Leer una secuencia de caracteres e imprimirla al revés



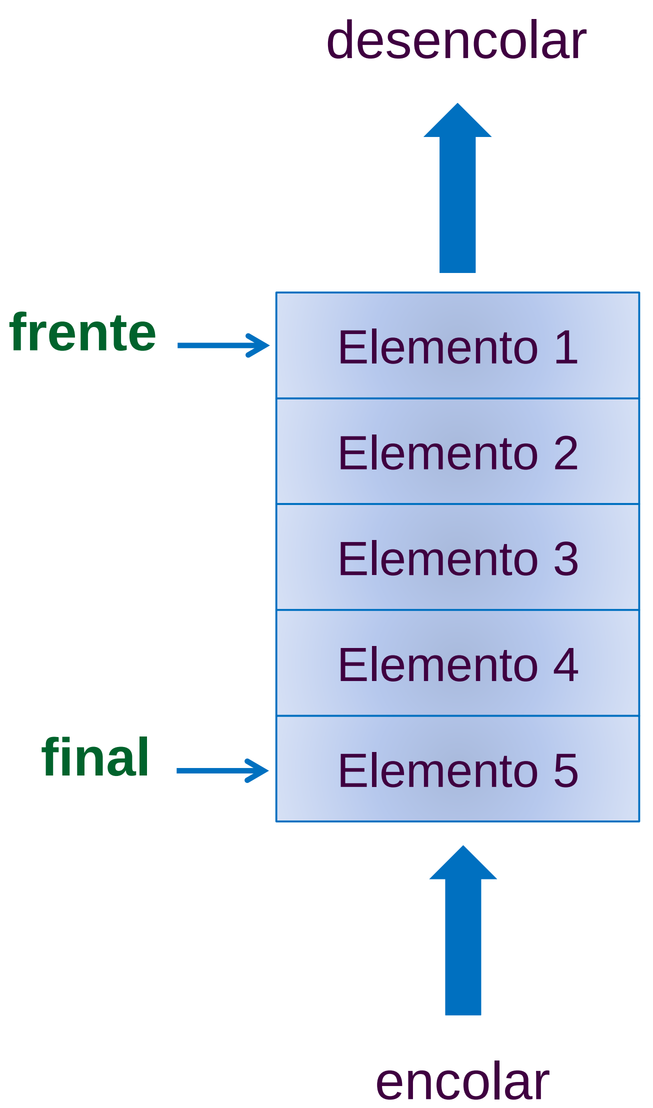
  - Comprobar si una cadena de caracteres tiene paréntesis balanceados
    - (abc(de)efg((hi)(j))kl) → BALANCEADOS
    - abc(de(fg(hi))))i(jkl → NO BALANCEADOS

Colas

- Estructura FIFO (First In First Out)

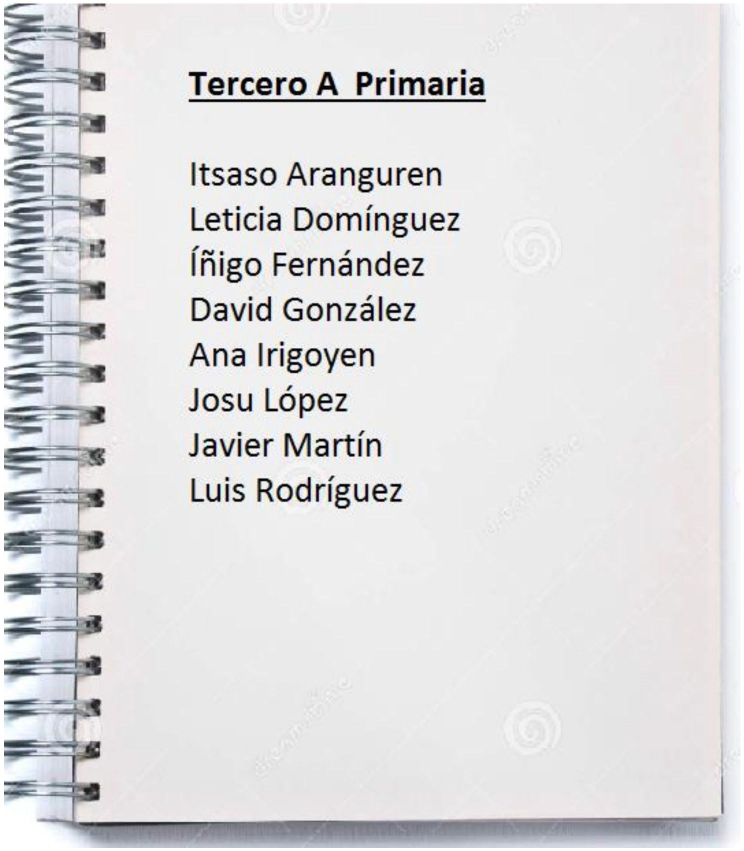

Colas

- Encolar
  - Añade un elemento en el final de la cola
- Desencolar
  - Elimina el elemento del frente de la cola
- Frente
  - Muestra el elemento del frente de la cola
- ¿Está vacía?
  - Comprueba si la cola contiene elementos o no
- Vaciar
  - Eliminan uno a uno todos los elementos de la cola



Listas


SHOPPING LIST
Leche
Aceite
Garbanzos
Salmón
Yogures

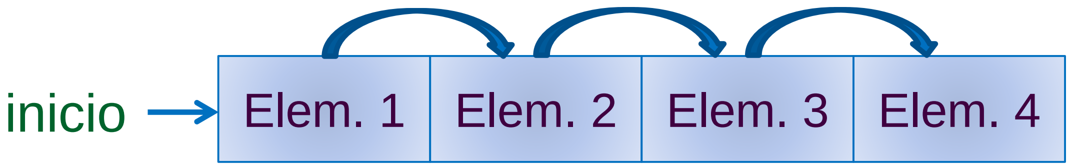

Tercero A Primaria
Itsaso Aranguren
Leticia Domínguez
Íñigo Fernández
David González
Ana Irigoyen
Josu López
Javier Martín
Luis Rodríguez

Listas

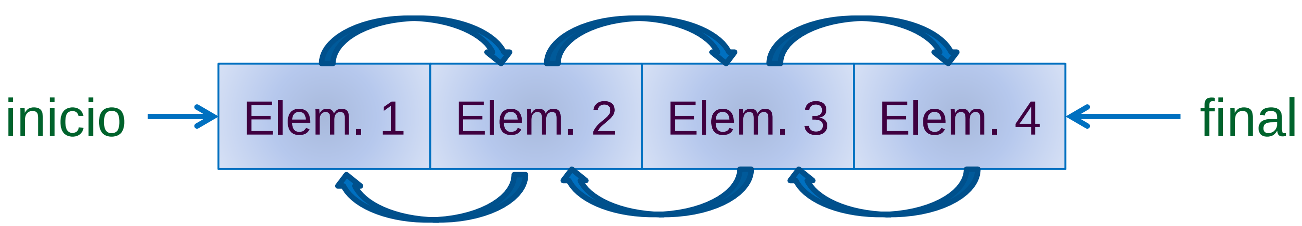
- Insertar
  - Añade un elemento a la lista en una posición dada
- Eliminar
  - Elimina un elemento de la lista. Puede ser por valor o por posición
- ¿Está vacía?
  - Comprueba si la lista contiene elementos o no

Listas

- Lista simple
  - A partir de cada elemento se puede acceder al siguiente



- Lista doble
  - A partir de cada elemento se puede acceder al siguiente y al anterior

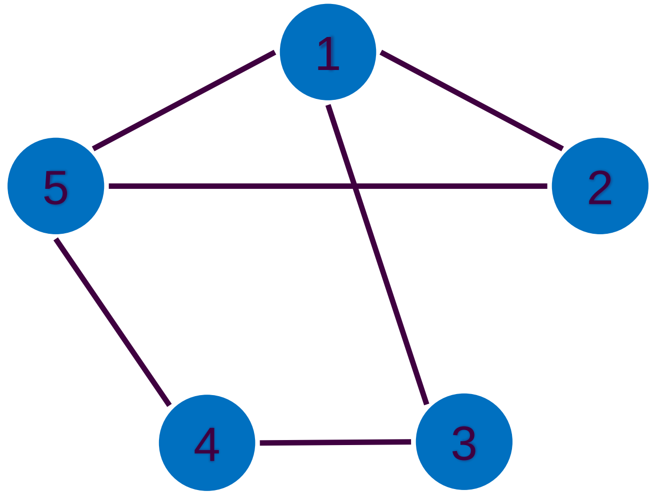


Índice

- Repaso. Estructuras de datos básicas
- Grafos
  - Repaso
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Montículos

Grafos

- Un grafo **G = ⟨N,A⟩** está formado por un conjunto finito y no vacío de vértices o nodos N y un conjunto A de pares de vértices a los que se llama aristas o arcos

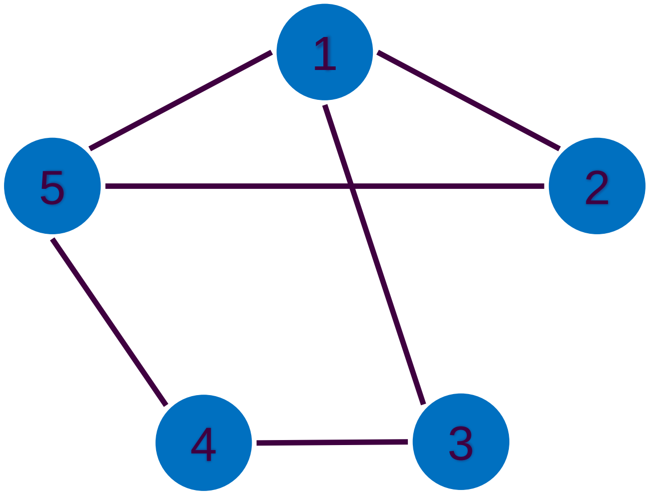


N={1, 2, 3, 4, 5}

A={{1,2}, {1,3}, {1,5}, {2,5}, {3,4}, {4,5}}

Grafos dirigidos y no dirigidos

- En un grafo no dirigido, el par de vértices que representa una arista está desordenado ({u,v} y {v,u} representan la misma arista)



N={1, 2, 3, 4, 5}

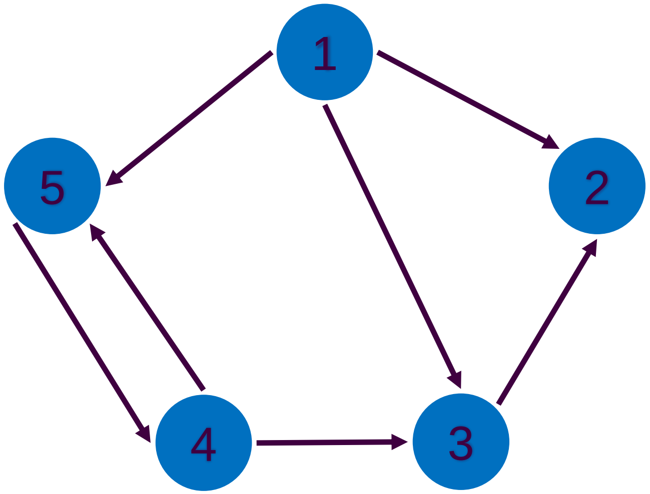
A={{1,2}, {1,3}, {1,5},
{2,5}, {3,4}, {4,5}}

=

A={{1,2}, {1,3}, {5,1},
{5,2}, {4,3}, {5,4}}

Grafos dirigidos y no dirigidos

- En un grafo dirigido las aristas o arcos se representan por pares de vértices ordenados (u,v)

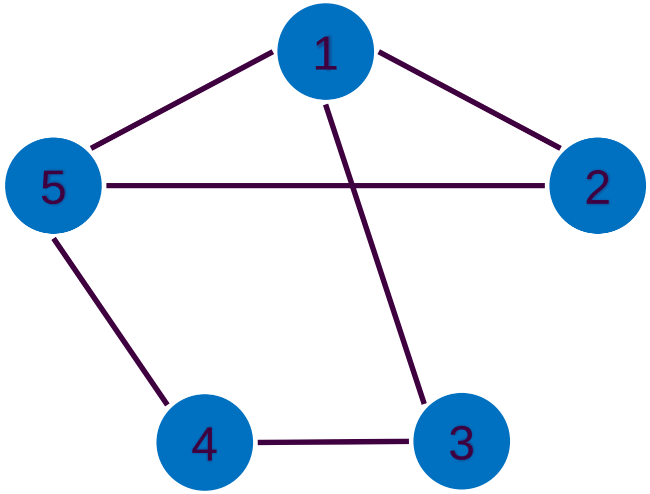


N={1, 2, 3, 4, 5}

A={(1,2), (1,3), (1,5), (3,2), (4,3), (4,5), (5,4)}

Grafos

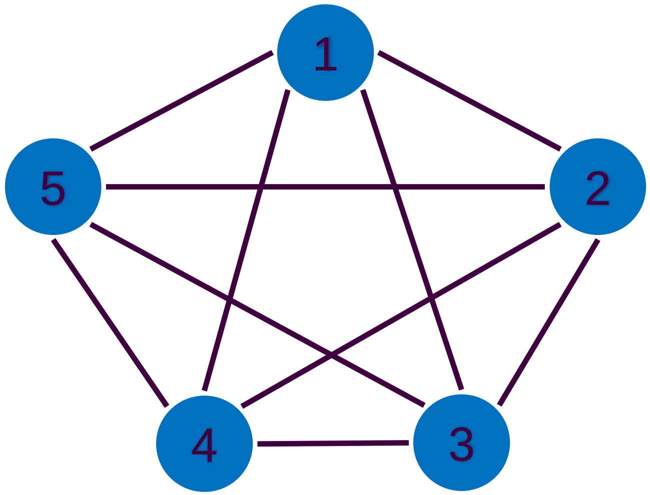
- Dos vértices u, v de un grafo G = ⟨N,A⟩ se dicen adyacentes si {u, v} ∈ A
- Dos aristas son adyacentes si tienen un mismo vértice como extremo
- Si a = {u, v} decimos que la arista a es incidente a los vértices u y v.
- El grado de un vértice es el número de arcos incidentes a él. El grado de un vértice u se denota gr(u).

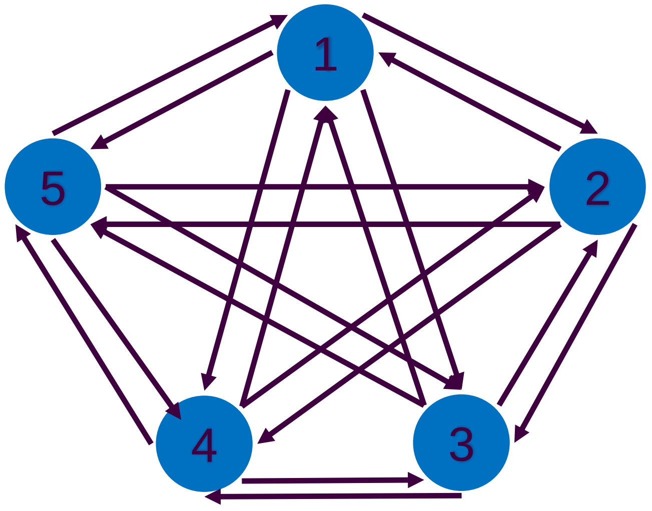


- 1 y 2 son vértices adyacentes porque {1,2}ϵA
- {1,2} y {1,3} son aristas adyacentes
- La arista {1,2} es incidente a los vértices 1 y 2
- gr(1)=3
- gr(4)=2

Grafos

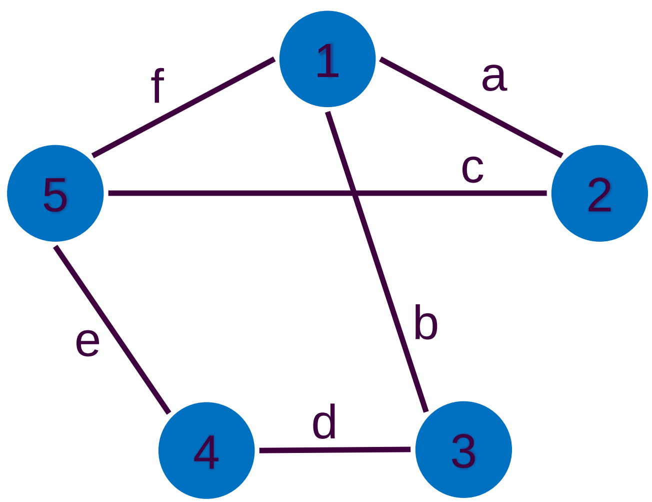
- Un grafo simple G = ⟨N,A⟩ se dice completo si cada vértice está conectado a cualquier otro vértice en G
  - Un grafo no dirigido de n vértices es completo si tiene exactamente n(n-1)/2 arcos
  - Un grafo dirigido de n vértices es completo si tiene exactamente n(n-1) arcos



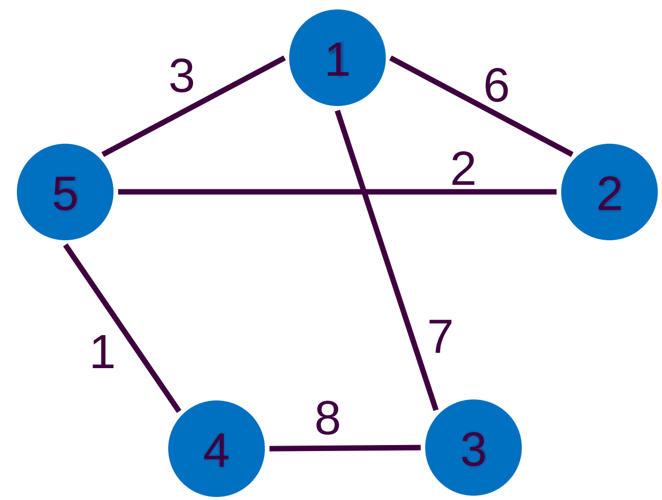


Grafos ponderados

- Un grafo G es un grafo etiquetado si sus aristas y/o vértices tienen asignado alguna identificación
- En particular, G es un grafo ponderado si a cada arista a de G se le asigna un número no negativo w(a) denominado peso o longitud de a



Grafo etiquetado



Grafo ponderado

Índice

- Repaso. Estructuras de datos básicas
- Grafos
  - Introducción a grafos
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Montículos

Implementación de grafos

- Existen diversas estructuras
- Cada una es apropiada en diferentes situaciones

- Matriz de adyacencia
- Lista de adyacencia
- Array de aristas

Matriz de adyacencia

- Sea G = ⟨N,A⟩ un grafo con n vértices, n ≥ 1. La matriz de adyacencia D es un array bidimensional n x n definido de la siguiente forma:

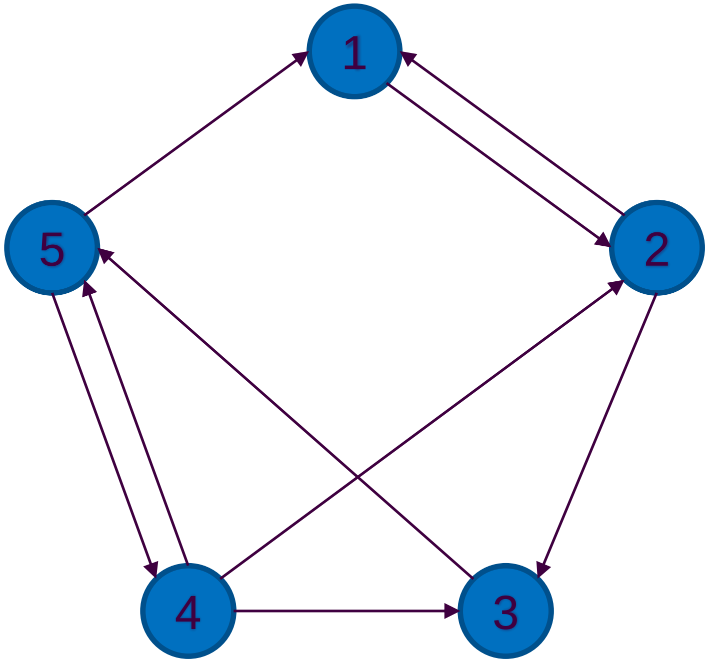
$$D(i,j) = \begin{cases} 1 & si\ \ \{i,j\} \in A\ \ ((i,j)\ \ en\ grafos\ dirigidos) \\ 0 & en\ caso\ contrario \end{cases}$$

- Si G es un grafo ponderado, la matriz de adyacencia se define de la siguiente forma:

$$D(i,j) = \begin{cases} w(i,j) & si\ \ \{i,j\} \in A\ \ ((i,j)\ \ en\ grafos\ dirigidos) \\ cte & en\ caso\ contrario \end{cases}$$

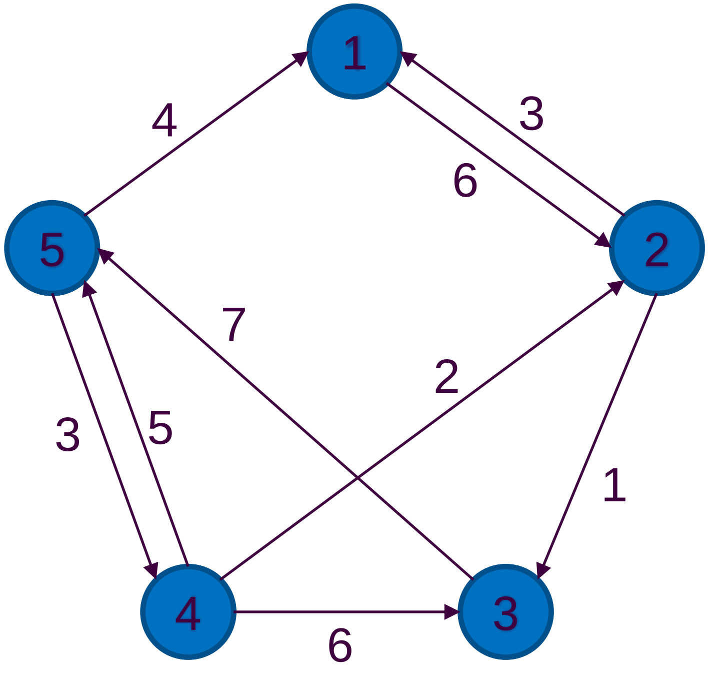
Donde cte es un valor constante (0, inf, -inf, …)

Matriz de adyacencia



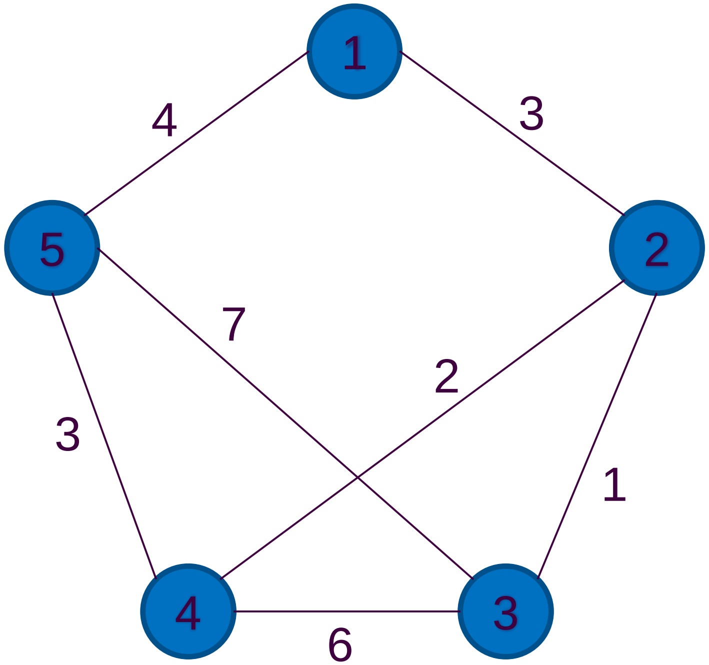
$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Matriz de adyacencia



$$M = \begin{pmatrix} 0 & 6 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 \\ 0 & 2 & 6 & 0 & 5 \\ 4 & 0 & 0 & 3 & 0 \end{pmatrix}$$

Matriz de adyacencia



$$M = \begin{pmatrix} 0 & 3 & 0 & 0 & 4 \\ 3 & 0 & 1 & 2 & 0 \\ 0 & 1 & 0 & 6 & 7 \\ 0 & 2 & 6 & 0 & 3 \\ 4 & 0 & 7 & 3 & 0 \end{pmatrix}$$

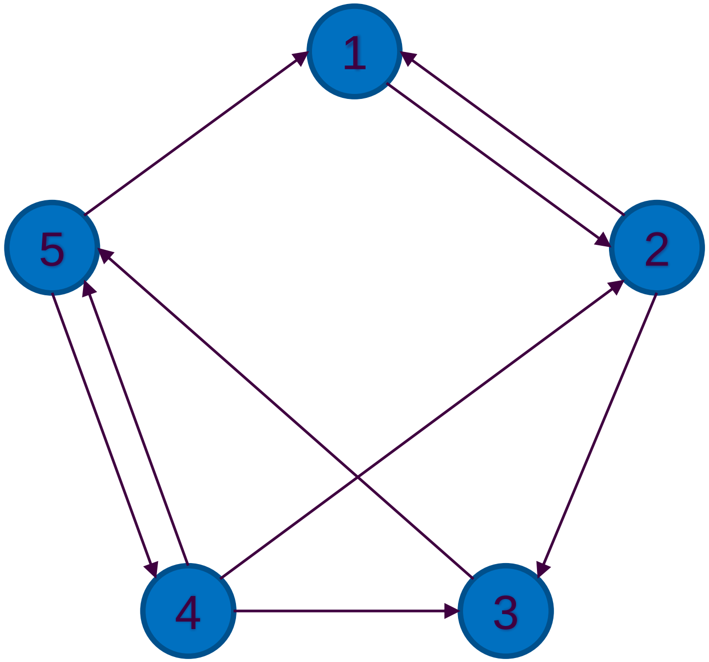
Matriz de adyacencia

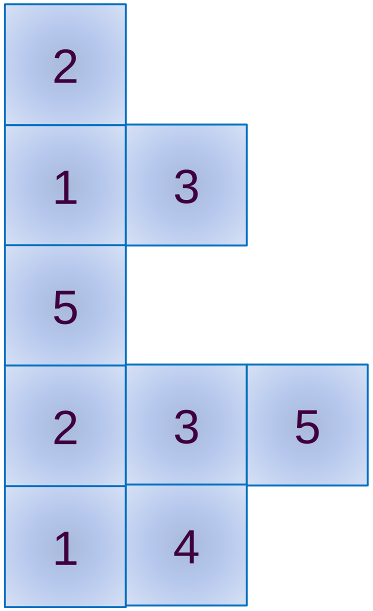
- Es inmediato ver si dos vértices están conectados. El orden de eficiencia de esta operación es independiente del número de vértices y arcos
- No es inmediato ver cuántos arcos tiene el grafo, o si éste es conexo
- Se utiliza mucha memoria extra en grafos poco densos (gran número de vértices y pocas aristas)

Lista de adyacencia

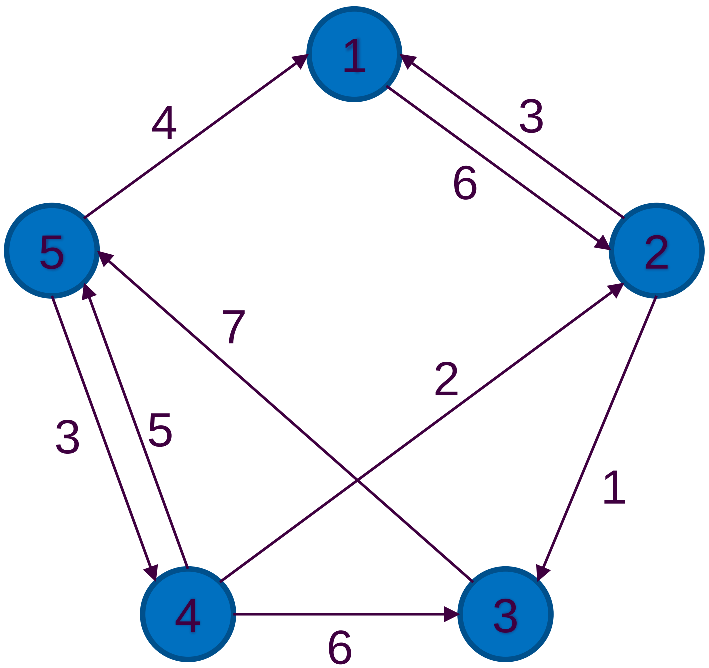
- Sea $G = \langle N,A \rangle$ un grafo con n vértices, $n \geq 1$. La lista de adyacencia es una lista de n listas enlazadas. Cada elemento de la lista está asociado con un vértice i, y representa mediante una lista los nodos destino de las aristas con origen en i.
- Si G es un grafo ponderado, entonces en cada lista se almacenan los nodos destino y los pesos de las aristas con origen en i.

Lista de adyacencia



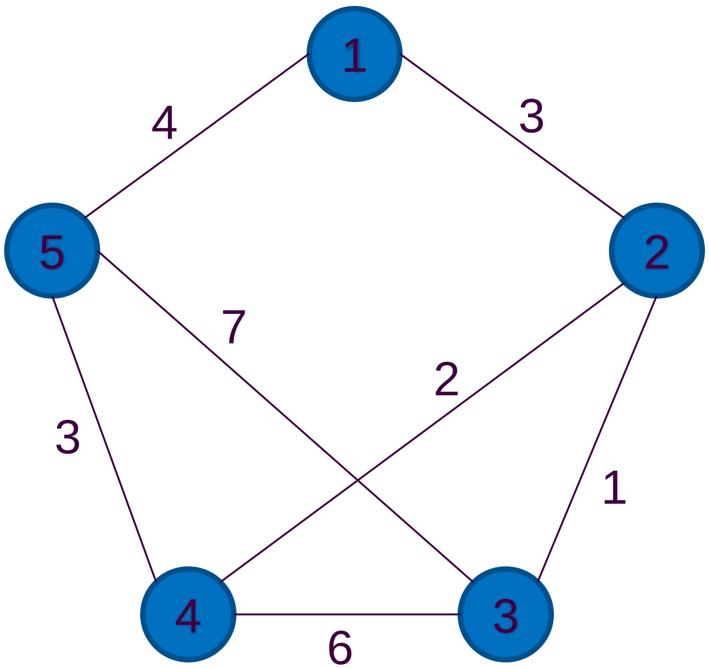


Lista de adyacencia



(2,6)		
(1,3)	(3,1)	
(5,7)		
(2,2)	(3,6)	(5,5)
(1,4)	(4,3)	

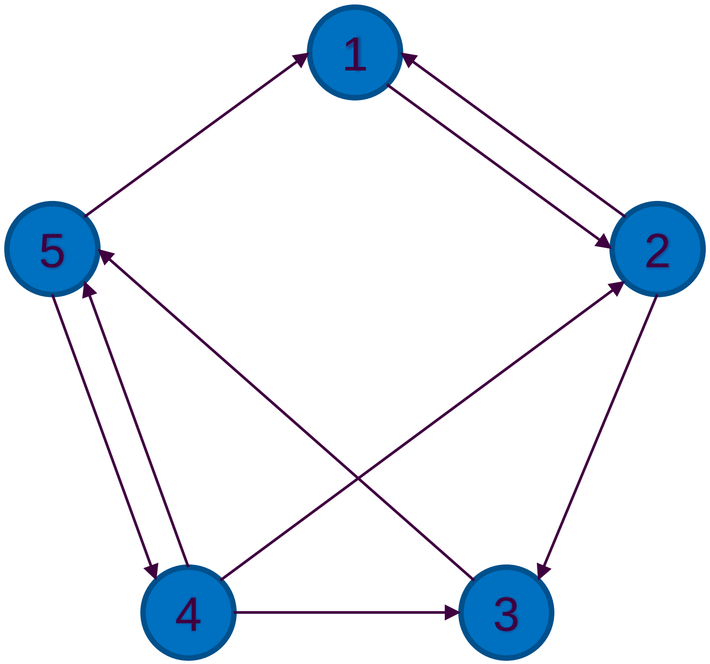
Lista de adyacencia

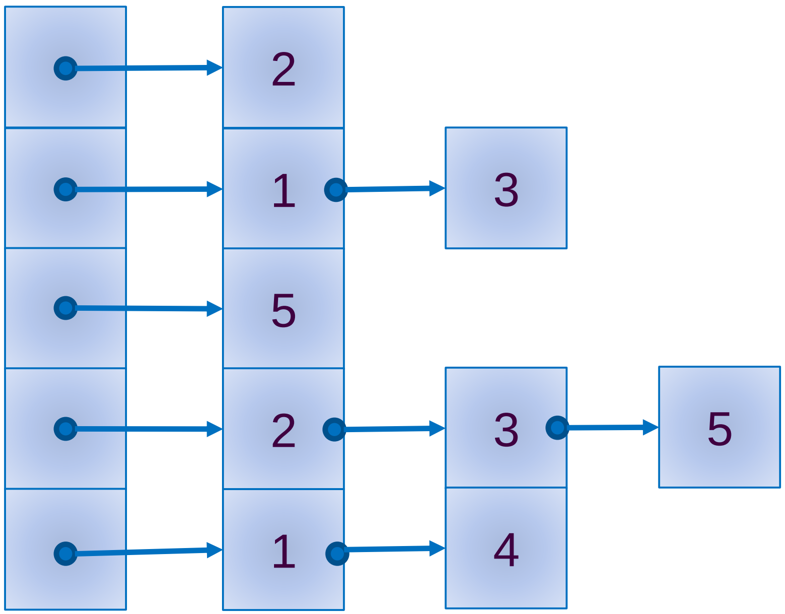


(2,3)	(5,4)	
(1,3)	(3,1)	(4,2)
(2,1)	(4,6)	(5,7)
(2,2)	(3,6)	(5,3)
(1,4)	(3,7)	(4,3)

Lista de adyacencia

- No podemos implementar una matriz con diferente número de columnas en cada fila
- Utilizamos punteros





Implementación de grafos

- Matrices de adyacencia vs. Listas de adyacencia
  - En general son mejores las matrices de adyacencia si los grafos son densos ($a \approx n^2$) y las listas de adyacencia para grafos poco densos.
  - Para cada problema hay que sopesar las diferentes opciones y elegir la más adecuada.

Ejercicio

- Sea G = ⟨N,A⟩ un grafo ponderado dirigido con 20 vértices. Suponemos que un elemento de una matriz ocupa una unidad de espacio; y un campo de una lista ocupa 3 unidades (una para almacenar el nodo destino, otra para el peso y otra para el puntero). ¿Hasta cuántas aristas puede tener el grafo para que ocupe menos espacio almacenado en una lista de adyacencia que en una matriz de adyacencia?
- Para el mismo grafo. Suponemos que se tarda lo mismo en recorrer un elemento de la matriz que un elemento de la lista; y queremos encontrar los arcos que tienen como nodo destino el vértice 5. ¿Cuántas aristas tiene que tener el grafo para que sea más rápido hacer la búsqueda en una lista de adyacencia que en una matriz de adyacencia?

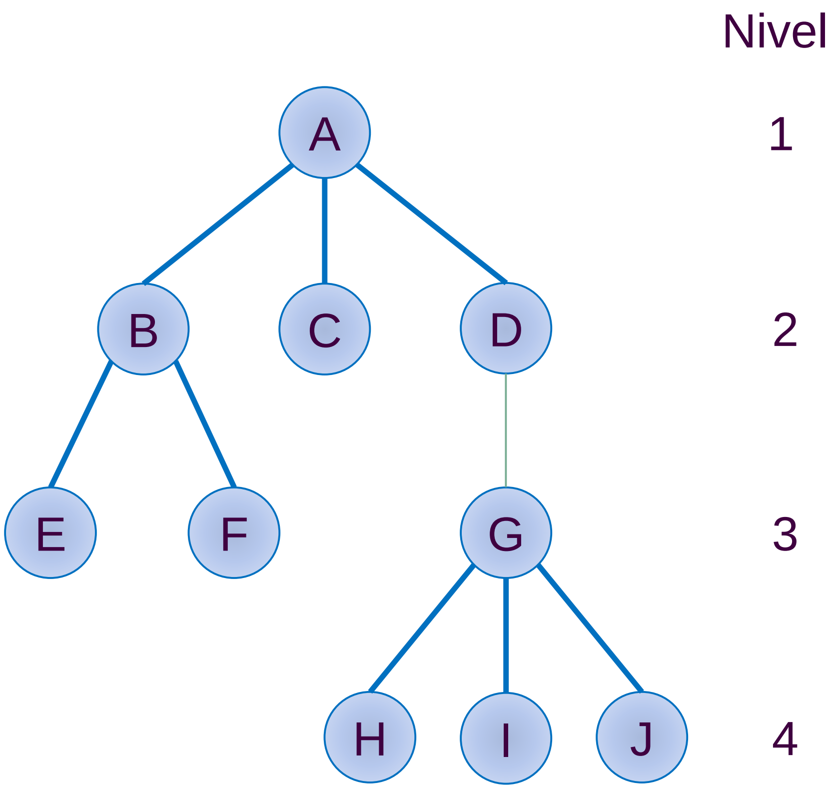
Índice

- Repaso. Estructuras de datos básicas
- Implementación de pilas
- Grafos
  - Introducción a grafos
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Implementación de árboles binarios de búsqueda
  - Montículos

Árboles

- Un árbol es un grafo no dirigido, conexo y acíclico.
- Un árbol T es un conjunto finito de uno o más nodos tal que hay un nodo especial llamado raíz y el resto están divididos en $n \geq 0$ subconjuntos disjuntos $T_1, T_2, \ldots, T_n$ de árboles a los que llamaremos subárboles de T.
- También el conjunto vacío es un árbol
- La representación más habitual de un árbol es mediante listas enlazadas

Árboles



- Elementos de un árbol
  - Hojas o nodos terminales; nodos no terminales
  - Hijos, padres, hermanos, antecesores de un nodo
  - Grado de un nodo: número de hijos
  - Nivel de un nodo: La raíz tiene nivel 1 y, si un nodo tiene nivel p, sus hijos tienen nivel p+1
  - Altura o profundidad: máximo nivel de cualquier nodo

Índice

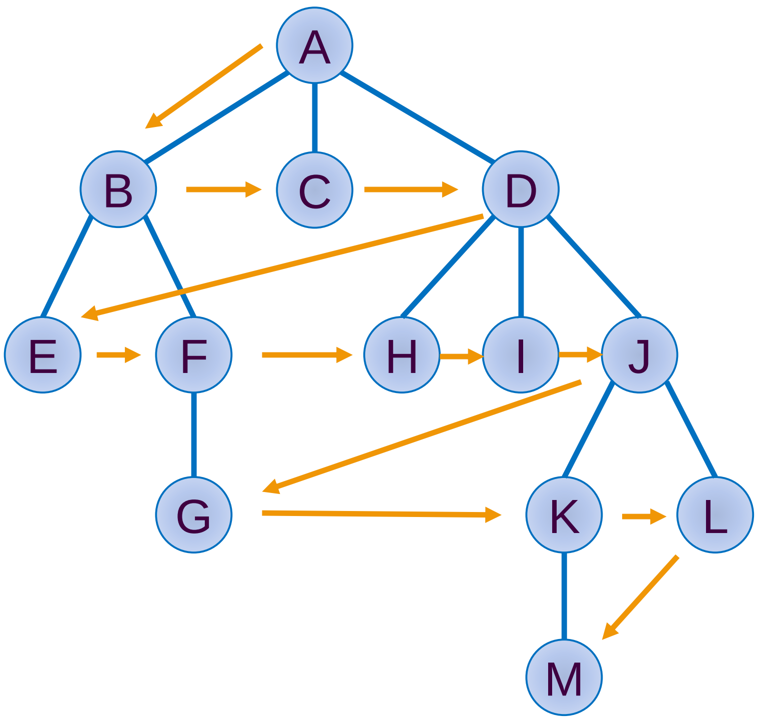
- Repaso. Estructuras de datos básicas
- Implementación de pilas
- Grafos
  - Introducción a grafos
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Implementación de árboles binarios de búsqueda
  - Montículos

Recorridos en anchura y profundidad

- Son dos métodos distintos de recorrer un grafo, para explorar todos sus vértices o para encontrar uno determinado.
- La diferencia fundamental es el orden en que se visitan los diferentes vértices.
- Para recorrer un grafo es necesario seleccionar un nodo inicial. En el caso de árboles, este nodo es la raíz del árbol.

Recorridos en anchura y profundidad

- Recorrido en anchura

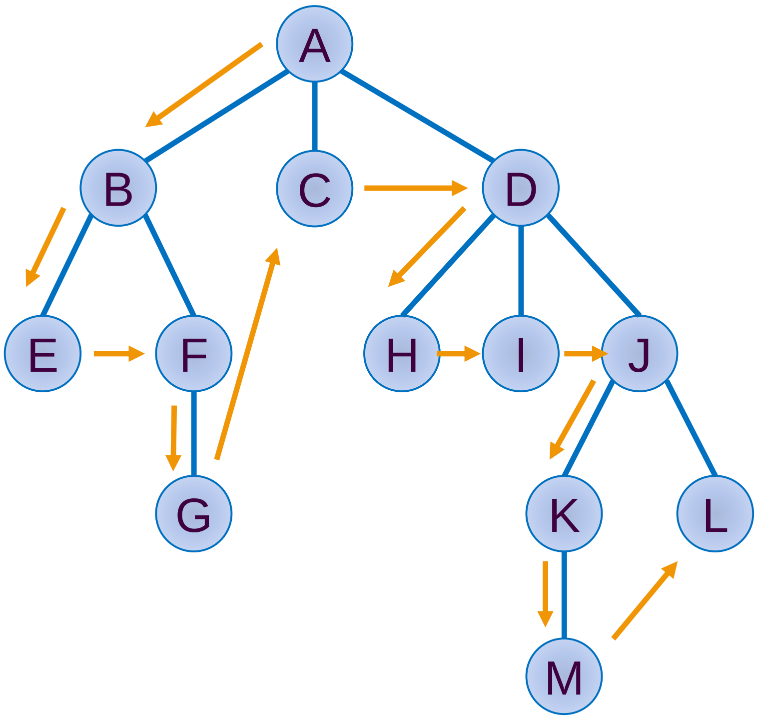


- A – B – C – D – E – F – H – I – J – G – K – L – M

- Se puede implementar mediante una cola
  - Mostar elemento
  - Encolar sus hijos

Recorridos en anchura y profundidad

- Recorrido en profundidad



- A – B – E – F – G – C – D – H – I – J – K – M – L

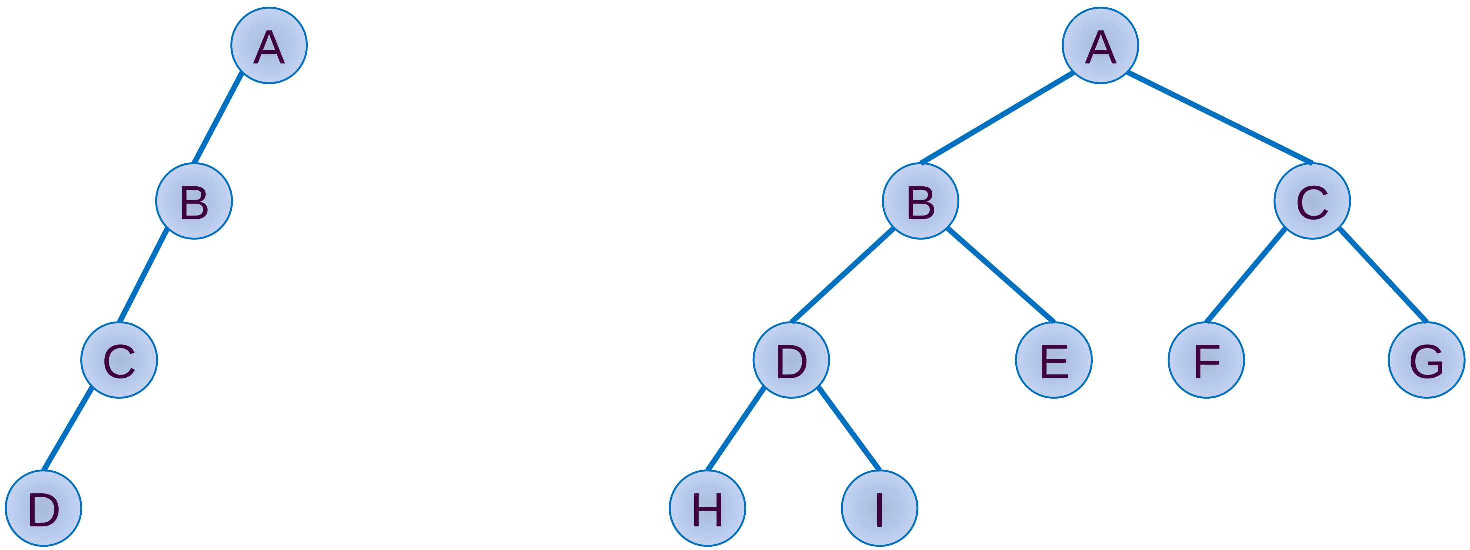
- Se puede implementar mediante una pila
  - Mostrar elemento
  - Apilar sus hijos en orden inverso

Índice

- Repaso. Estructuras de datos básicas
- Implementación de pilas
- Grafos
  - Introducción a grafos
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Implementación de árboles binarios de búsqueda
  - Montículos

Árboles binarios

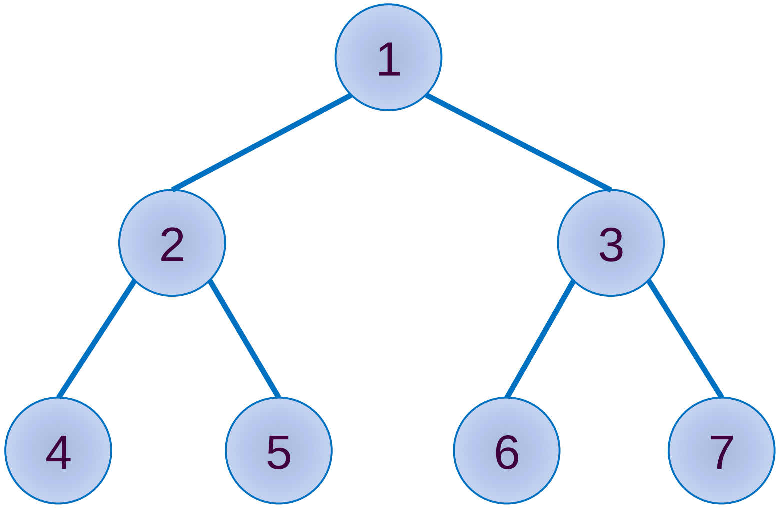
- Un árbol binario es un conjunto finito de nodos que puede estar vacío o formado por una raíz y dos subconjuntos disjuntos de árboles binarios llamados subárbol izquierdo y subárbol derecho



- En un árbol binario cada nodo puede tener 0, 1 o 2 hijos

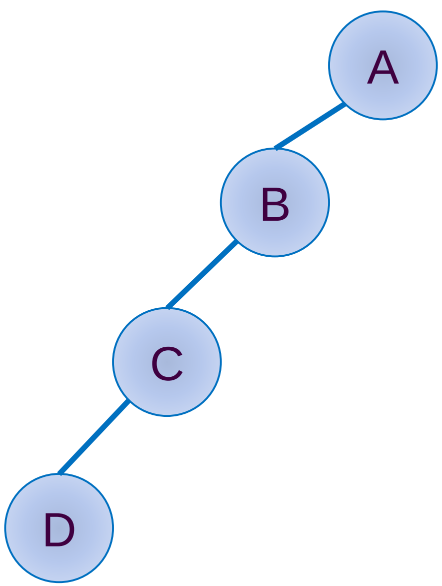
Árboles binarios

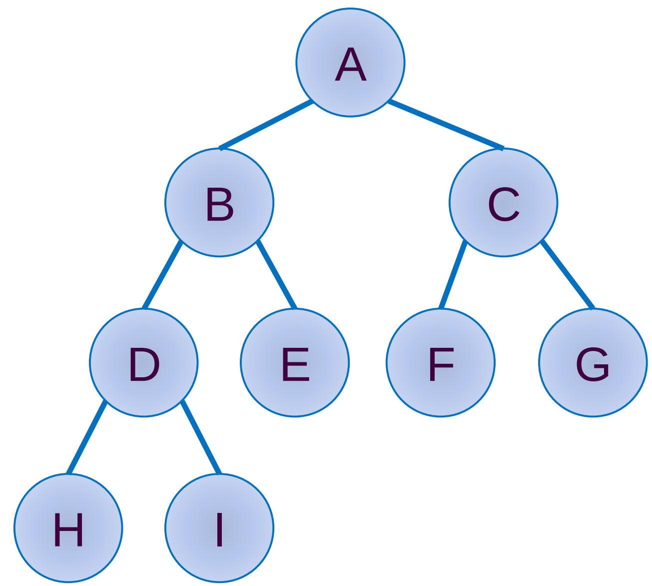
- Un árbol binario lleno de profundidad k tiene exactamente $2^k - 1$ nodos. Los numeramos de 1 a n ($n = 2^k$-1).
- Para cada cualquier nodo i (1≤i≤n):
- El padre de i es i/2, si i≠1 (la raíz no tiene padre)
- El hijo izquierdo de i es 2i, si 2i≤n (en otro caso no hay hijo izquierdo)
- El hijo derecho de i es 2i+1, si 2i+1≤n (en otro caso no hay hijo derecho)



Árboles binarios

- Un árbol binario de n nodos y profundidad k es completo si y sólo si sus nodos se corresponden con los nodos numerados de 1 a n en el árbol binario lleno de profundidad k.

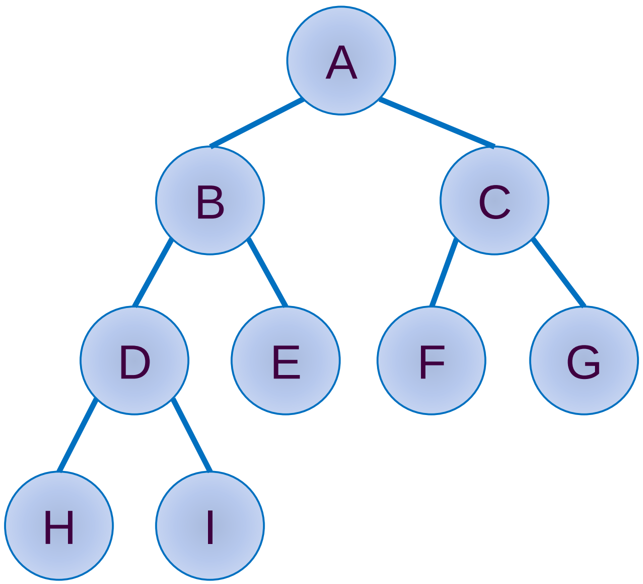




El segundo es completo (aunque no es lleno), pero el primero no.

Árboles binarios

- Los árboles binarios completos se pueden representar mediante vectores



A	B	C	D	E	F	G	H	I

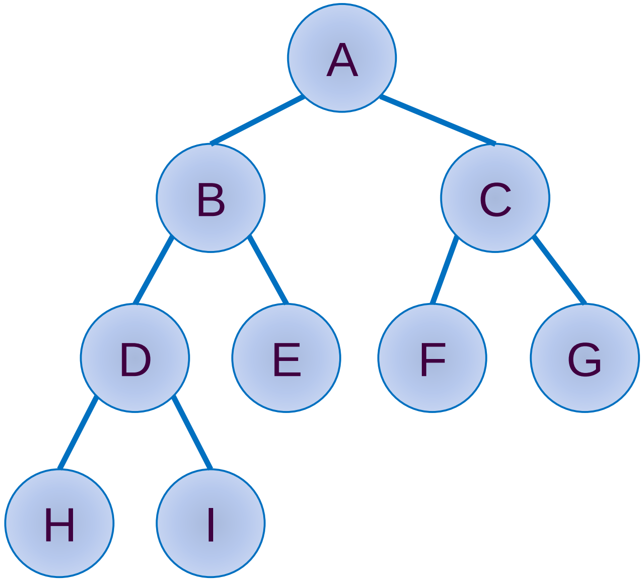
Árboles binarios

- Recorrido de un árbol binario
  - Permite acceder una sola vez a cada uno de los nodos de un árbol
  - Recorrido en pre-orden
  - Recorrido en in-orden
  - Recorrido en post-orden

Árboles binarios

- Recorrido en pre-orden
  - Visita la raíz
  - Recorrido del subárbol izquierdo en pre-orden
  - Recorrido del subárbol derecho en pre-orden

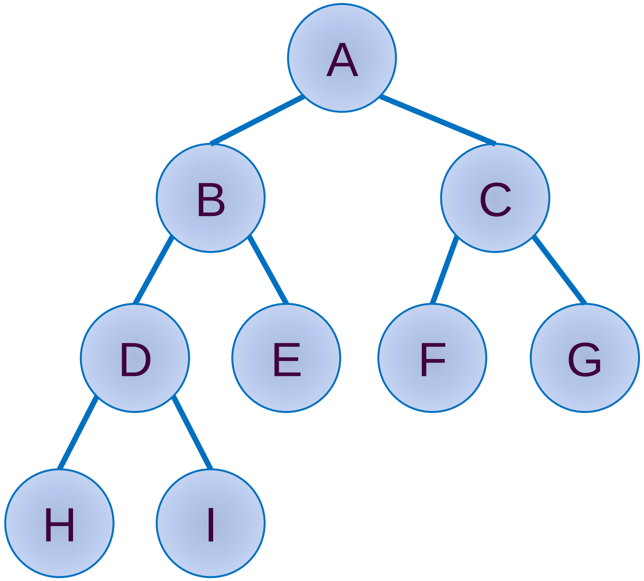
A – B – D – H – I – E – C – F - G



Árboles binarios

- Recorrido en in-orden
  - Recorrido del subárbol izquierdo en in-orden
  - Visita la raíz
  - Recorrido del subárbol derecho en in-orden

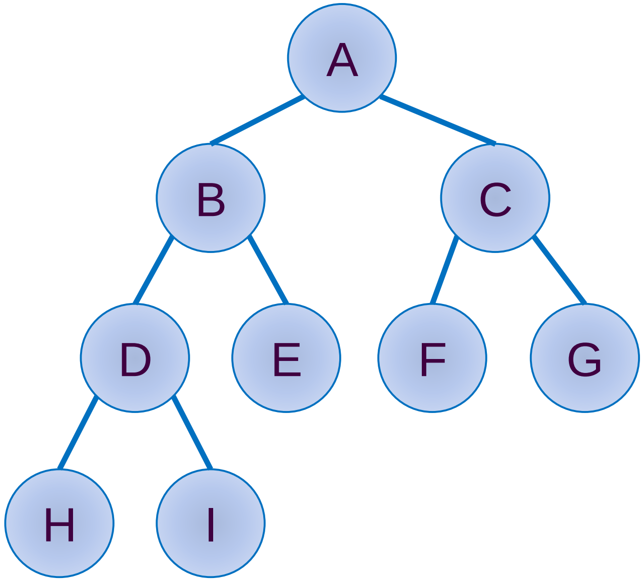
H – D – I – B – E – A – F – C - G



Árboles binarios

- Recorrido en post-orden
  - Recorrido del subárbol izquierdo en post-orden
  - Recorrido del subárbol derecho en post-orden
  - Visita la raíz

H – I – D – E – B – F – G – C - A



Ejercicio

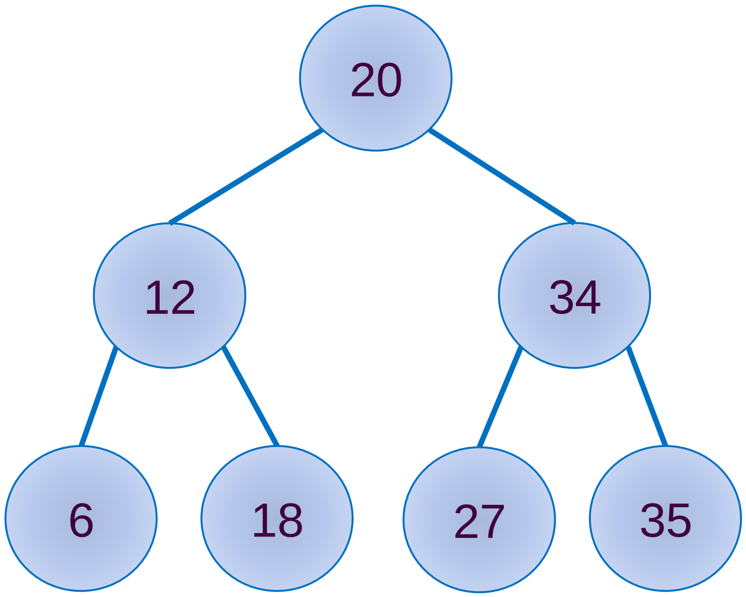
- Vamos a utilizar un árbol binario para almacenar operaciones matemáticas binarias (con dos argumentos). Estas operaciones nos llegan en notación postfija (primero los argumentos y después el operando) y debemos almacenarlas en un árbol. Posteriormente, a partir del árbol, debemos mostrar la operación en notación infija (primer argumento, operador, segundo argumento) utilizando paréntesis para agrupar suboperaciones.
- Si la operación a almacenar es 5 3 + 6 2 * - 4 -, ¿Cómo queda el árbol? ¿Qué tipo de recorrido hay que realizar para mostrarlo en notación infija: (((5+3)-(6*2))-4) ?

Ejercicio

- Seguimos con el mismo problema del árbol de operaciones matemáticas. Además del árbol propiamente, ¿qué otras estructuras de datos utilizarías para construir ese árbol? ¿De qué forma?

Árboles binarios de búsqueda

- Un árbol binario de búsqueda es un árbol binario que puede ser vacío o, en caso contrario, satisface las siguientes propiedades:
  - Cada elemento tiene una clave y no hay dos elementos con la misma clave
  - Las claves del subárbol izquierdo son menores que la clave de la raíz
  - Las claves del subárbol derecho son mayores que la clave de la raíz
  - Los subárboles izquierdo y derecho son también árboles binarios de búsqueda



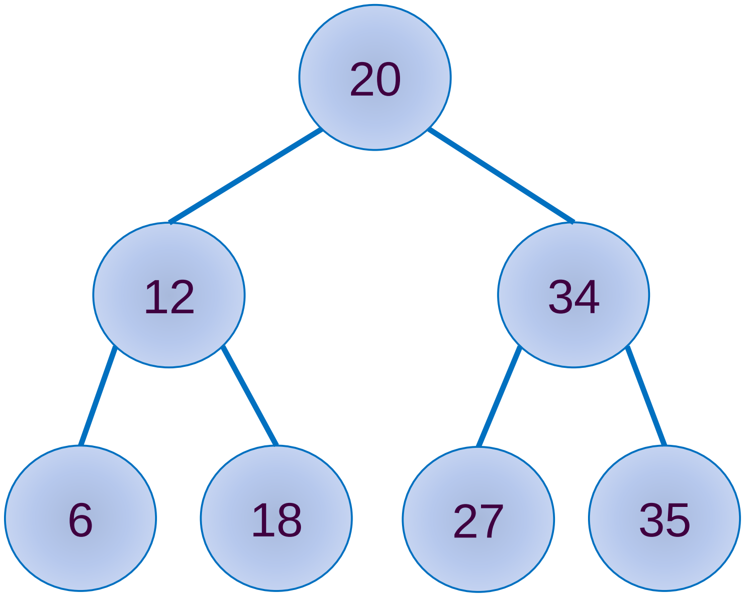
Árboles binarios de búsqueda

- Búsqueda recursiva en un árbol binario de búsqueda

- Si el árbol está vacío
  - Devolver NO
- Si no
  - Si buscado = elemento en raíz
    - Devolver raíz
  - Si no
    - Si buscado < elemento en raíz
      - Buscar en subárbol izquierdo
    - Si no
      - Buscar en subárbol derecho

Árboles binarios de búsqueda

- Búsqueda recursiva en un árbol binario de búsqueda



Buscamos el elemento 27
Buscar (20, 27)
Buscar (34, 27)
Buscar (27, 27) → ENCONTRADO

Buscamos el elemento 19
Buscar (20, 19)
Buscar (12, 19)
Buscar (18, 19)
Buscar (vacío, 19) → NO ENCONTRADO

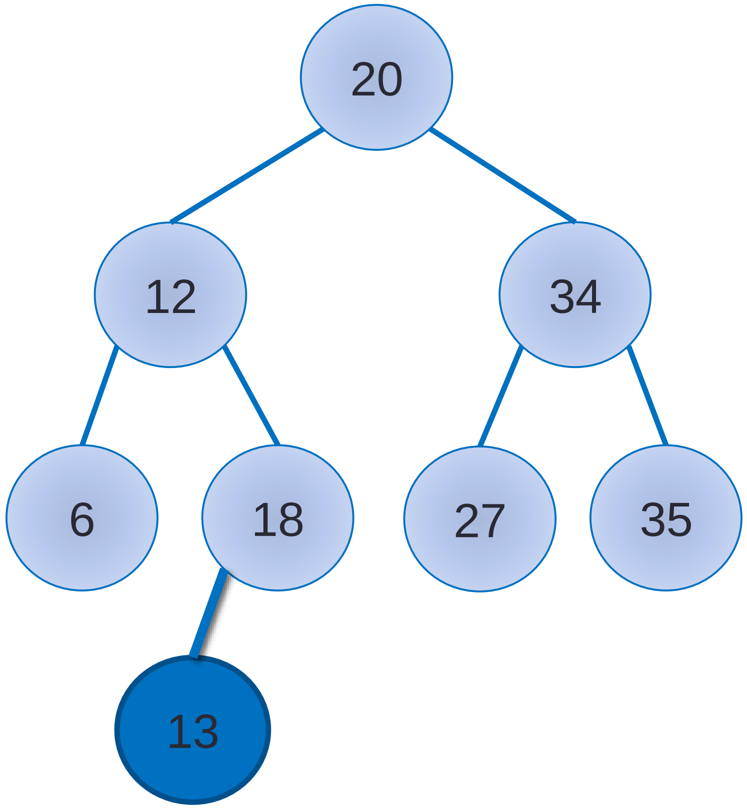
Árboles binarios de búsqueda

- Inserción en un árbol binario de búsqueda

- Si el árbol está vacío
  - Insertar número
- Si no
  - Si número = elemento en raíz
    - El número ya está en el árbol
  - Si no
    - Si número < elemento en raíz
      - Insertar número en subárbol izquierdo
    - Si no
      - Insertar número en subárbol derecho

Árboles binarios de búsqueda

- Inserción en un árbol binario de búsqueda



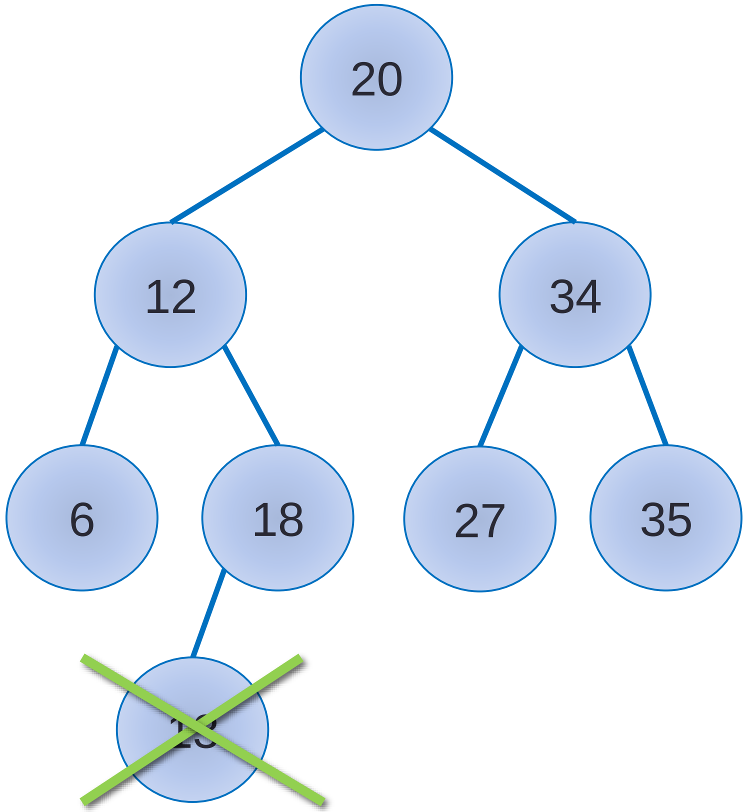
Buscamos el elemento 13
Insertar (20, 13)
Insertar (12, 13)
Insertar (18, 13)
Insertar (vacío, 13) → INSERTADO

Árboles binarios de búsqueda

- Borrado en un árbol binario de búsqueda
  - El elemento a eliminar es una hoja
    - Simplemente elimino el nodo
  - El elemento a eliminar sólo tiene un hijo
    - El padre del elemento se convierte en padre del hijo del elemento
  - El elemento a eliminar tiene dos hijos
    - Busco el sucesor del elemento a eliminar (elemento más pequeño del subárbol hijo derecho)
    - Coloco el sucesor en la posición del elemento a eliminar
    - Elimino el sucesor de su posición original

Árboles binarios de búsqueda

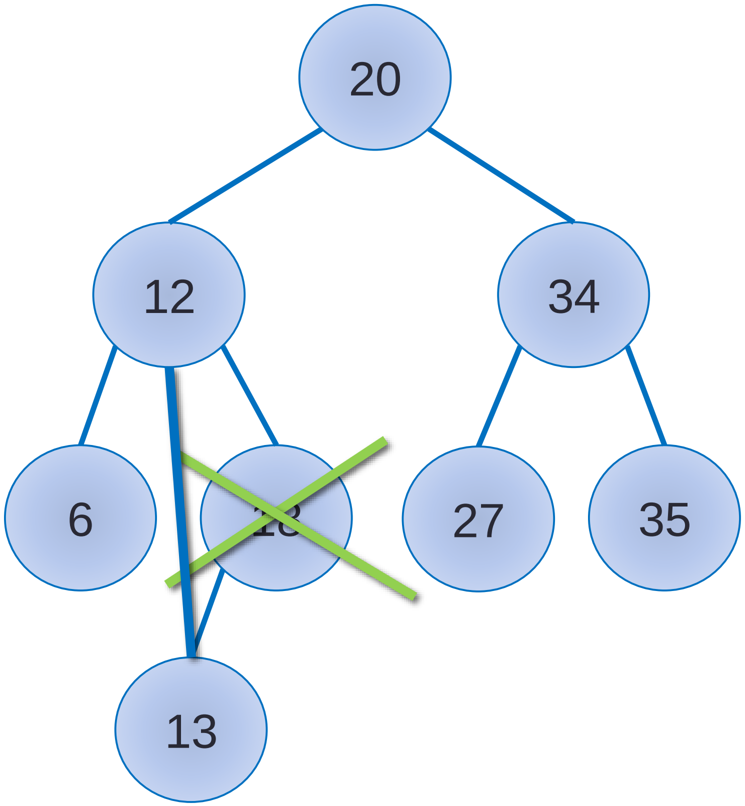
- Borrado en un árbol binario de búsqueda



Borramos el elemento 13
Eliminar 13

Árboles binarios de búsqueda

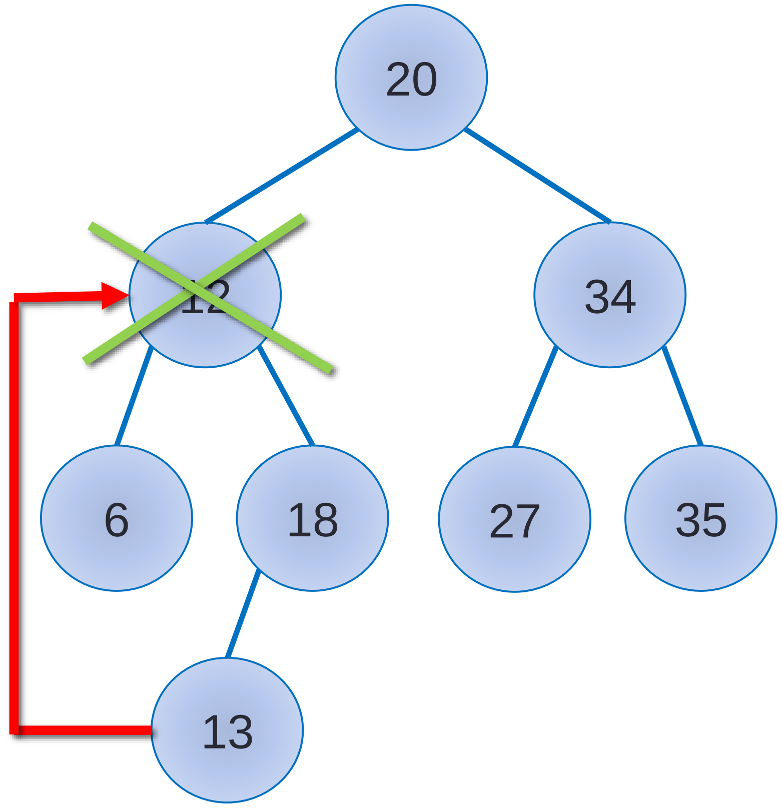
- Borrado en un árbol binario de búsqueda



Borramos el elemento 18
Unir 12 con 13
Eliminar 18

Árboles binarios de búsqueda

- Borrado en un árbol binario de búsqueda



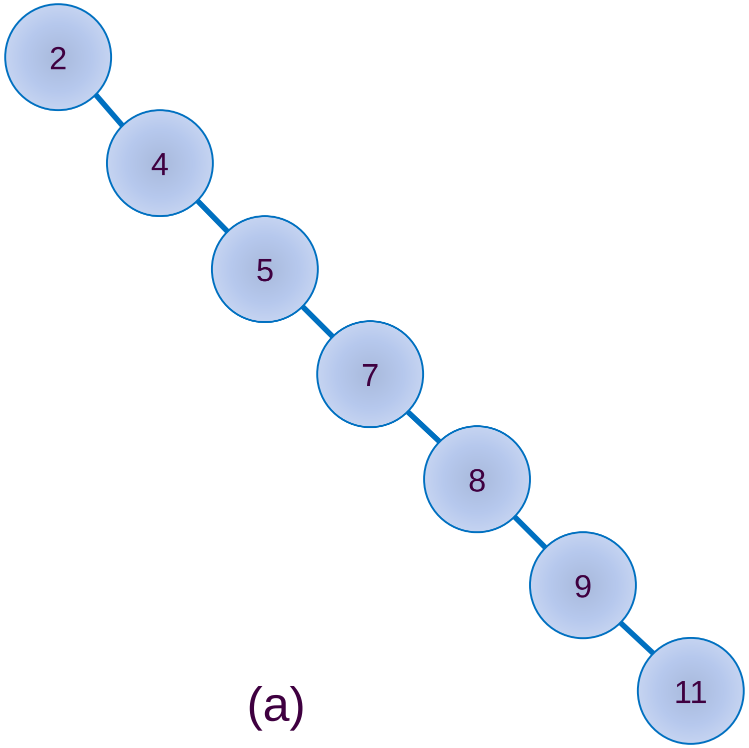
Borramos el elemento 12
Su sucesor es el elemento 13
Colocar 13 en la posición de 12
Eliminar 12 y 13 de sus posiciones originales

Árboles binarios de búsqueda

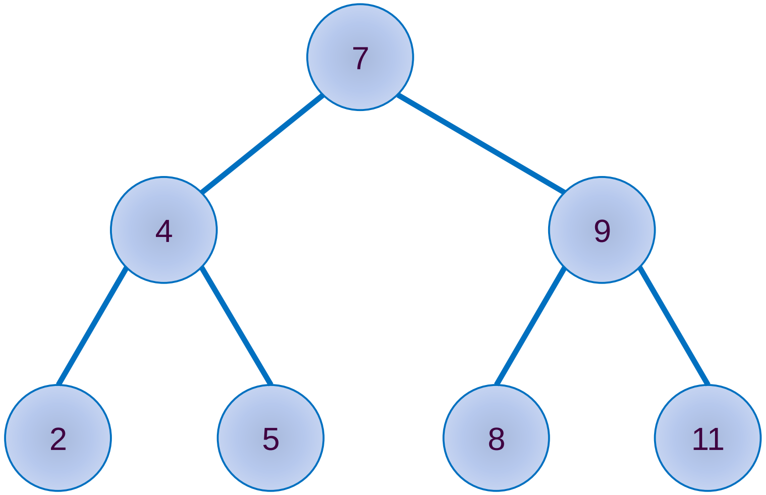
- Un mismo conjunto de datos puede crear diferentes árboles binarios de búsqueda, dependiendo del orden en que se introducen los elementos.
- Cuanto más balanceado esté el árbol, más eficiente serán las búsquedas que realicemos.
- Para que un árbol esté balanceado, la raíz debe coincidir con la mediana de los elementos.

Árboles binarios de búsqueda

- {2, 4, 5, 7, 8, 9, 11}
- {7, 4, 9, 8, 2, 5, 11}



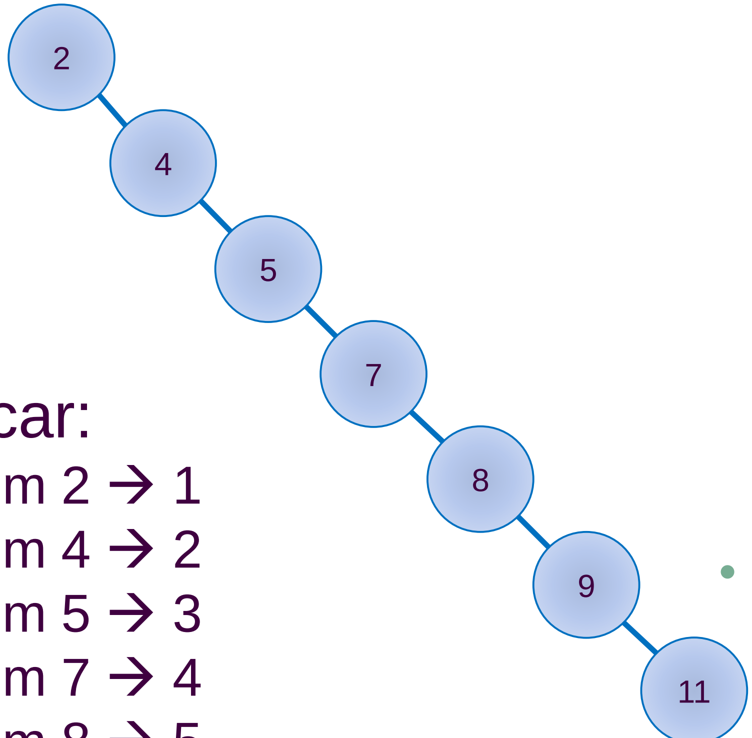
(a)



(b)

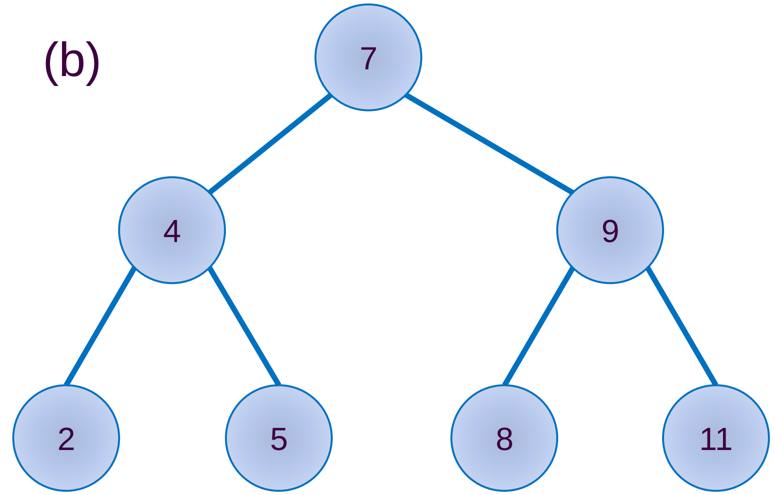
Árboles binarios de búsqueda

(a)



- Buscar:
  - Num 2 → 1
  - Num 4 → 2
  - Num 5 → 3
  - Num 7 → 4
  - Num 8 → 5
  - Num 9 → 6
  - Num 11 → 7
- Promedio:
  - 4 comparaciones

(b)



- Buscar:
  - Num 2 → 3
  - Num 4 → 2
  - Num 5 → 3
  - Num 7 → 1
  - Num 8 → 3
  - Num 9 → 2
  - Num 11 → 3

Promedio:

2,43 comparac.

Índice

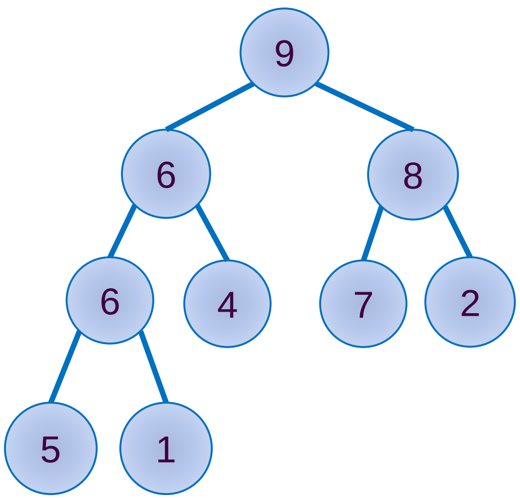
- Repaso. Estructuras de datos básicas
- Implementación de pilas
- Grafos
  - Introducción a grafos
  - Implementación de grafos
- Árboles
  - Recorridos en anchura y profundidad
  - Árboles binarios de búsqueda
  - Implementación de árboles binarios de búsqueda
  - Montículos

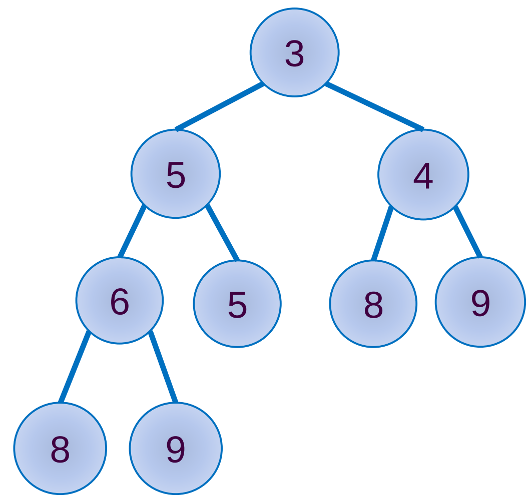
Montículos

- Un max-montículo es un árbol binario completo tal que el valor de cada nodo es mayor o igual que los valores de los nodos de sus hijos.
- Un min-montículo es un árbol binario completo tal que el valor de cada nodo es menor o igual que los valores de los nodos de sus hijos.
- La raíz de un max-montículo contendrá uno de los mayores valores (el mayor si todos son distintos) y la de un min-montículo uno de los menores valores.

Montículos

- Un montículo, como todo árbol completo, se puede representar mediante un vector unidimensional.



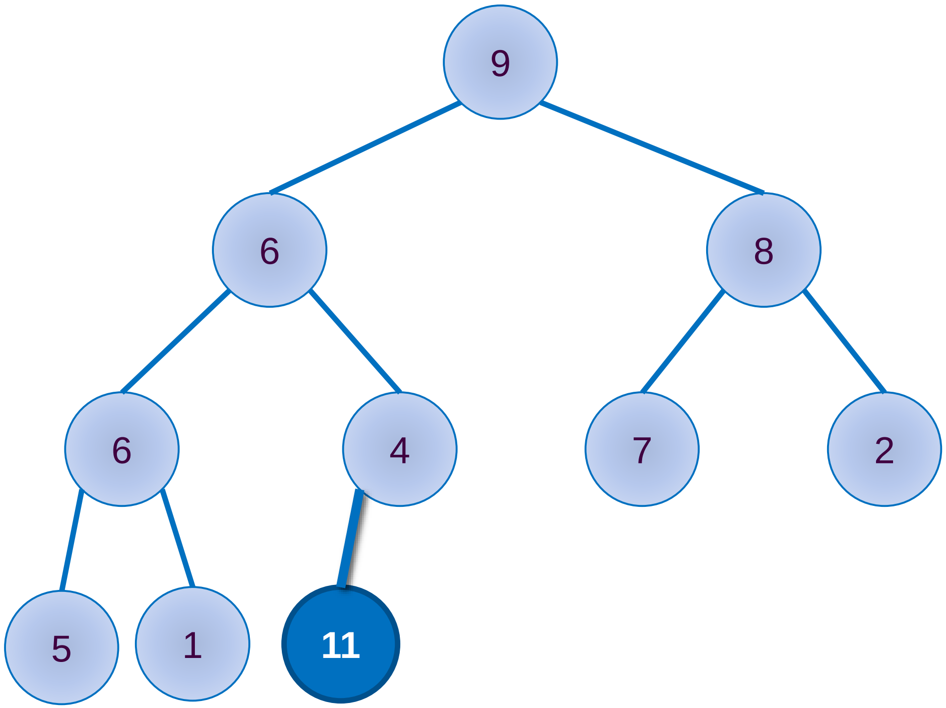


9	6	8	6	4	7	2	5	1

3	5	4	6	5	8	9	8	9

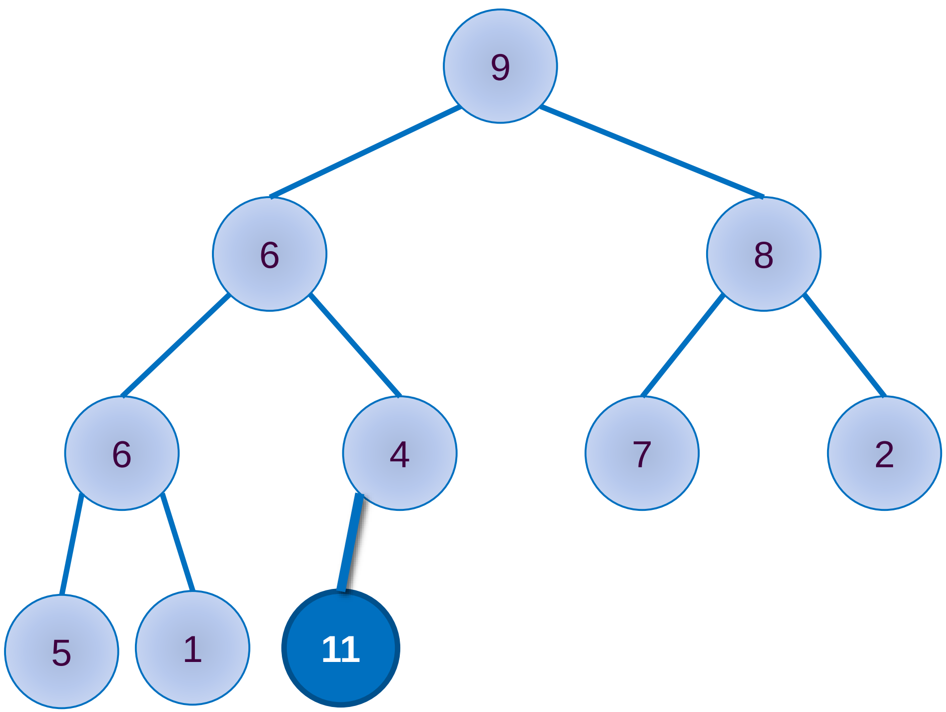
Montículos

- Inserción en un montículo
  - Insertamos el elemento en la siguiente posición



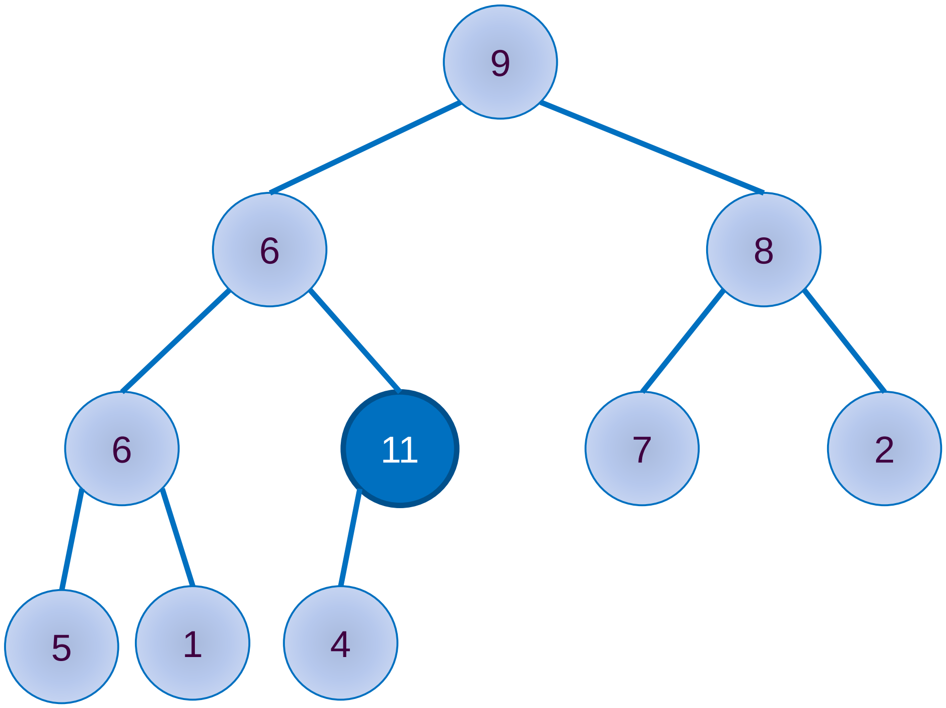
Montículos

- Inserción en un montículo
  - Insertamos el elemento en la siguiente posición
  - Mientras el nuevo elemento sea mayor que su padre (max-montículo)
    - Intercambiar elemento por su padre



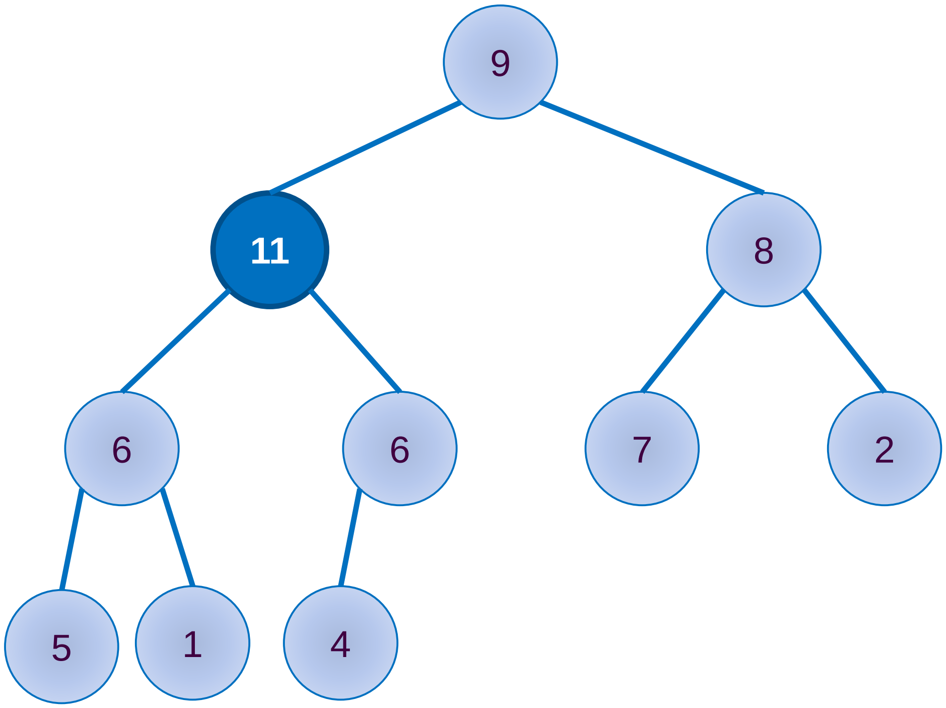
Montículos

- Inserción en un montículo
  - Insertamos el elemento en la siguiente posición
  - Mientras el nuevo elemento sea mayor que su padre (max-montículo)
    - Intercambiar elemento por su padre



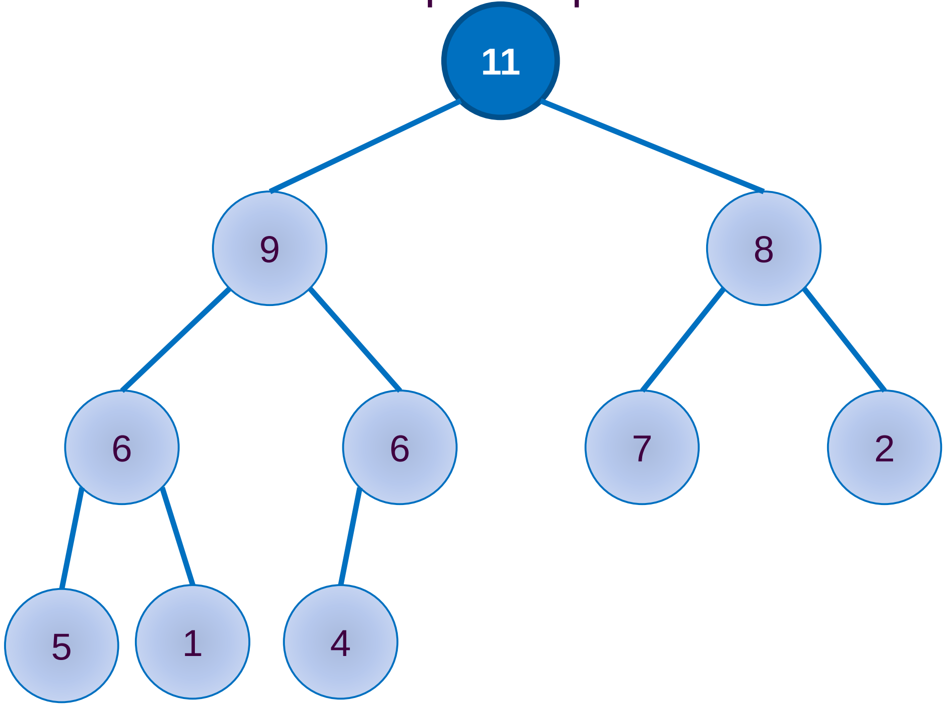
Montículos

- Inserción en un montículo
  - Insertamos el elemento en la siguiente posición
  - Mientras el nuevo elemento sea mayor que su padre (max-montículo)
    - Intercambiar elemento por su padre



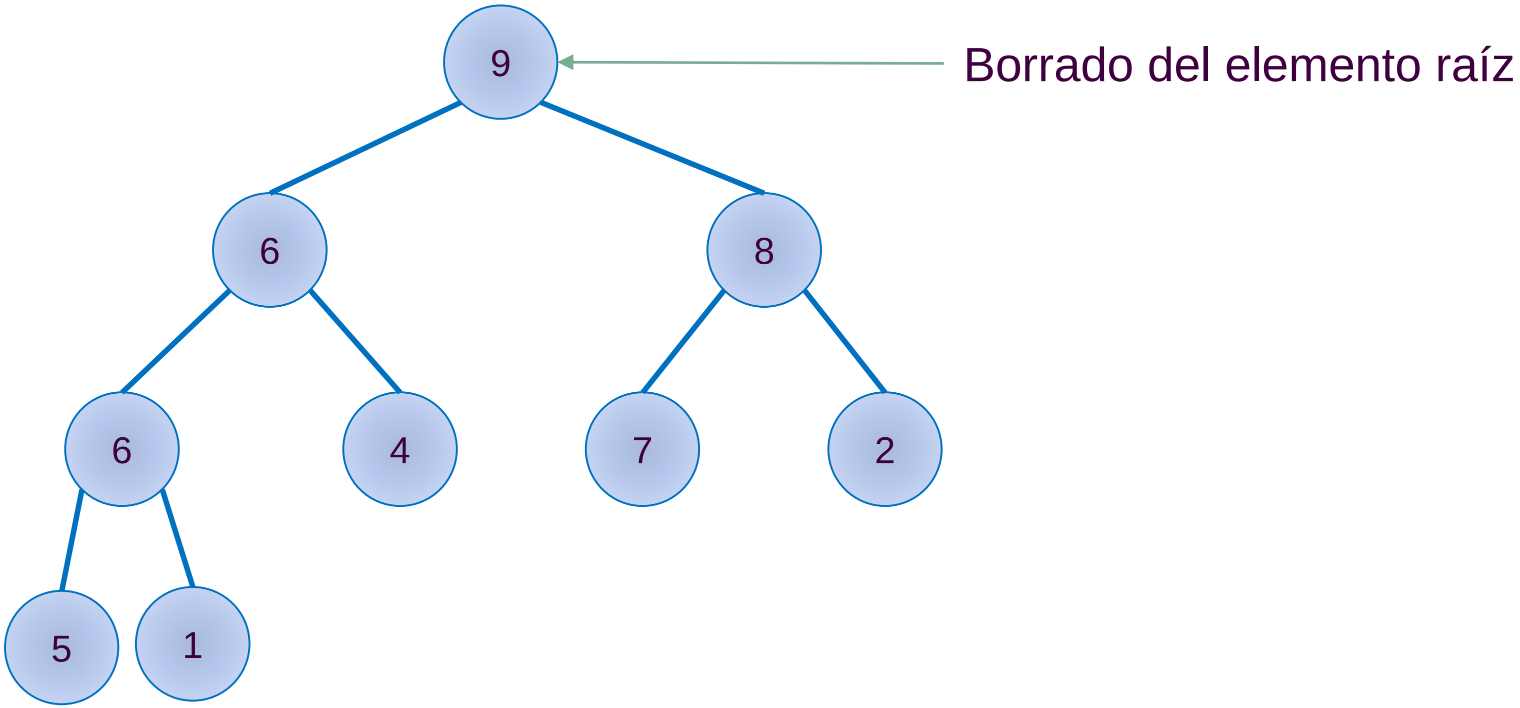
Montículos

- Inserción en un montículo
  - Insertamos el elemento en la siguiente posición
  - Mientras el nuevo elemento sea mayor que su padre (max-montículo) y no esté en la raíz
    - Intercambiar elemento por su padre



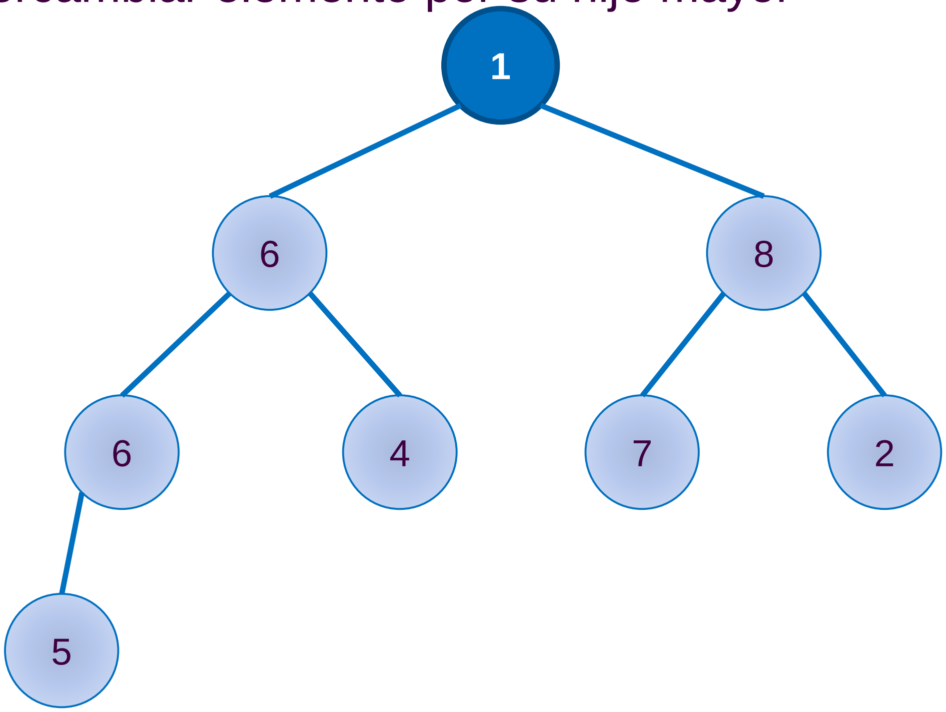
Montículos

- Borrado del elemento raíz en un montículo
  - Borramos el elemento y colocamos el último elemento en su posición



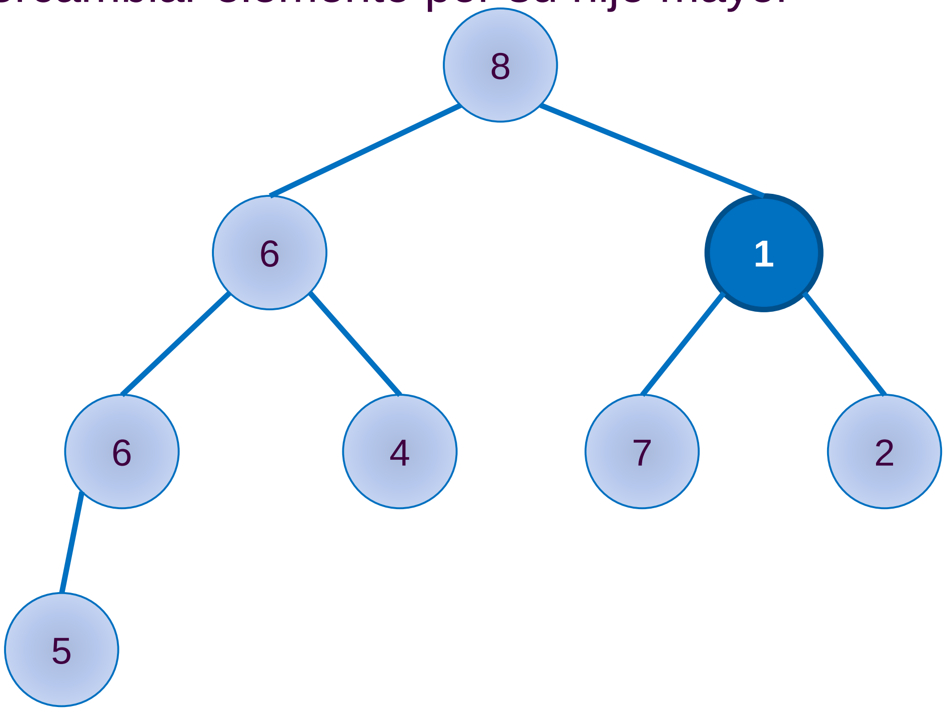
Montículos

- Borrado del elemento raíz en un montículo
  - Borramos el elemento y colocamos el último elemento en su posición
  - Mientras el elemento movido sea menor que alguno de sus hijos (max-montículo)
    - Intercambiar elemento por su hijo mayor



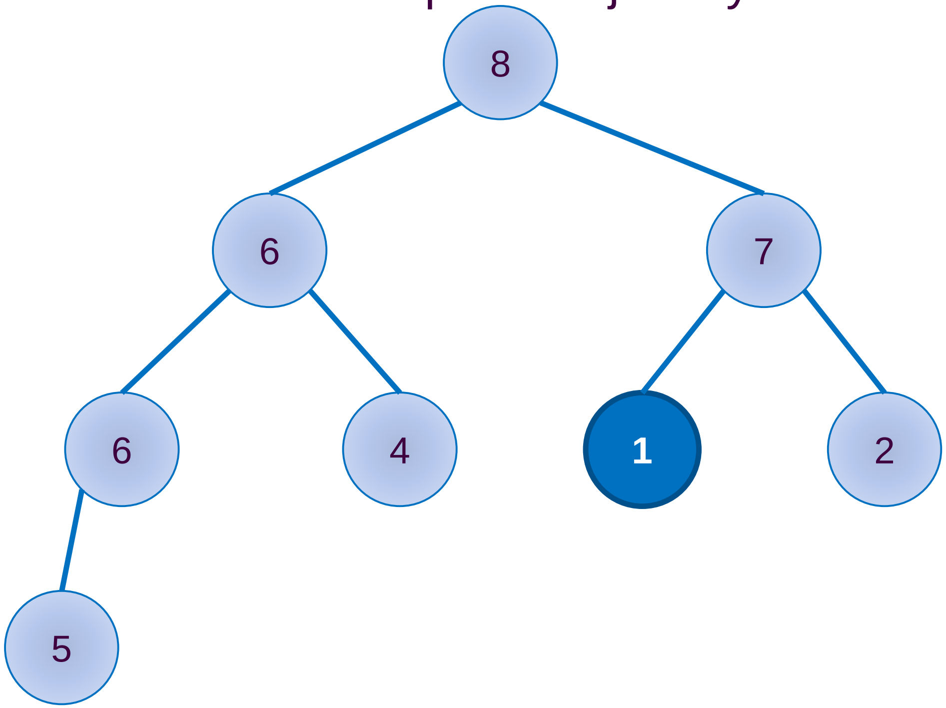
Montículos

- Borrado del elemento raíz en un montículo
  - Borramos el elemento y colocamos el último elemento en su posición
  - Mientras el elemento movido sea menor que alguno de sus hijos (max-montículo)
    - Intercambiar elemento por su hijo mayor



Montículos

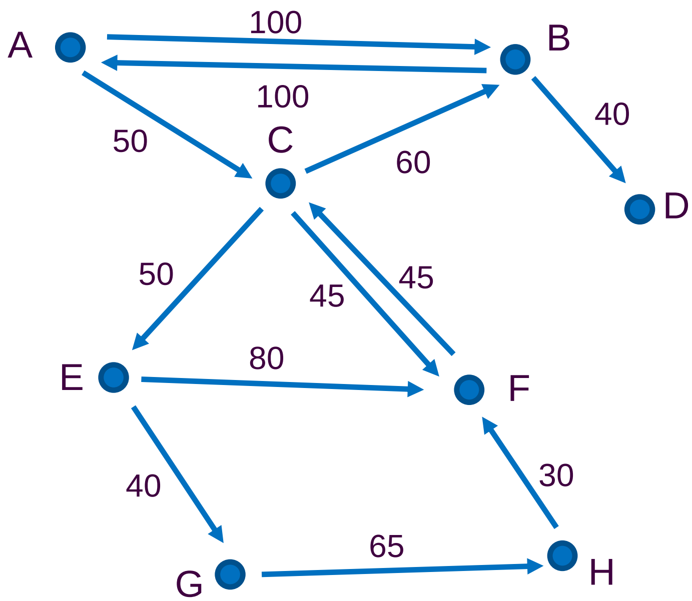
- Borrado del elemento raíz en un montículo
  - Borramos el elemento y colocamos el último elemento en su posición
  - Mientras el elemento movido sea menor que alguno de sus hijos (max-montículo)
    - Intercambiar elemento por su hijo mayor



Ejercicio

- Tenemos un conjunto de datos que queremos ordenar de menor a mayor. Una de las posibilidades existentes recurre a la utilización de un árbol.
- ¿Qué tipo de árbol es necesario utilizar? ¿Cuál sería esquema de ese algoritmo de ordenación?

Ejercicio



- Queremos almacenar en un grafo las conexiones que existen entre diferentes ciudades, atendiendo a si hay un servicio de tren directo que las une o no. En caso de existir, el peso de las aristas corresponde con los kilómetros de trayecto.
- Para cada uno de los siguientes casos, ¿en qué tipo de estructura sería más eficiente tener almacenado el grafo?

1) Conocer a cuántas ciudades hay tren directo desde la ciudad A.
2) Saber desde qué ciudad hay trenes directos a un mayor número de ciudades.
3) Mostrar desde qué ciudades se puede llegar a la ciudad C en tren directo.
4) Mostrar todas las conexiones existentes en orden decreciente de kilómetros (puedes utilizar un algoritmo de ordenación como el heapsort)

Ejercicio

- A partir de los siguientes datos (que llegan es este orden): 10 3 12 17 14 6 7 1 2
  - Construir un árbol binario de búsqueda
  - Construir un min-montículo
  - Construir un max-montículo

- Para el árbol de búsqueda, muestra los elementos en los siguientes órdenes:
  - Pre-orden
  - In-orden
  - Post-orden
  - Recorrido en anchura
  - Recorrido en profundidad

Ejercicio

- El recorrido en pre-orden de un árbol binario es:
  G – E – A – I – B – M – C – L – D – F – K – J – H
- El recorrido en in-orden del mismo árbol es:
  I – A – B – E – G – L – D – C – F – M – K – H – J
- Dibuja dicho árbol

Ejercicio

- Escribe un esquema de una función que devuelva la altura de un árbol binario.