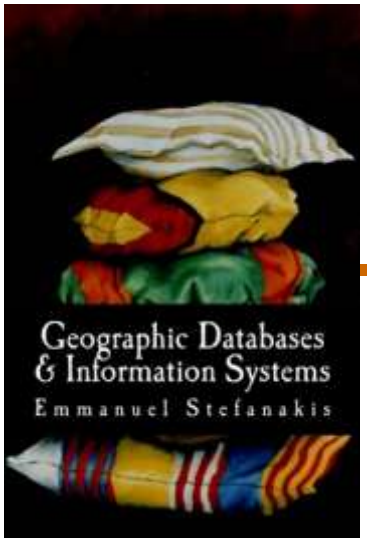


Stefanakis, E., 2014. *Geographic Databases and Information Systems*. CreateSpace Independent Publ. [In English], pp.386.

Get a copy from Amazon

Chapter 8

Data Structures and Algorithms

Emmanuel Stefanakis

http://www2.unb.ca/~estef/

Storing data elements

- Problem: Given two numbers find their sum

  x = 20 sum = x+y = 20+10 = 30

  y = 10

- Program (in C like)

```
int x;
int y;
int sum;
x=10; y=20;
sum = x+y;
printf("%d", sum);
```

→ x is a variable of type integer

built-in types (in programming languages)

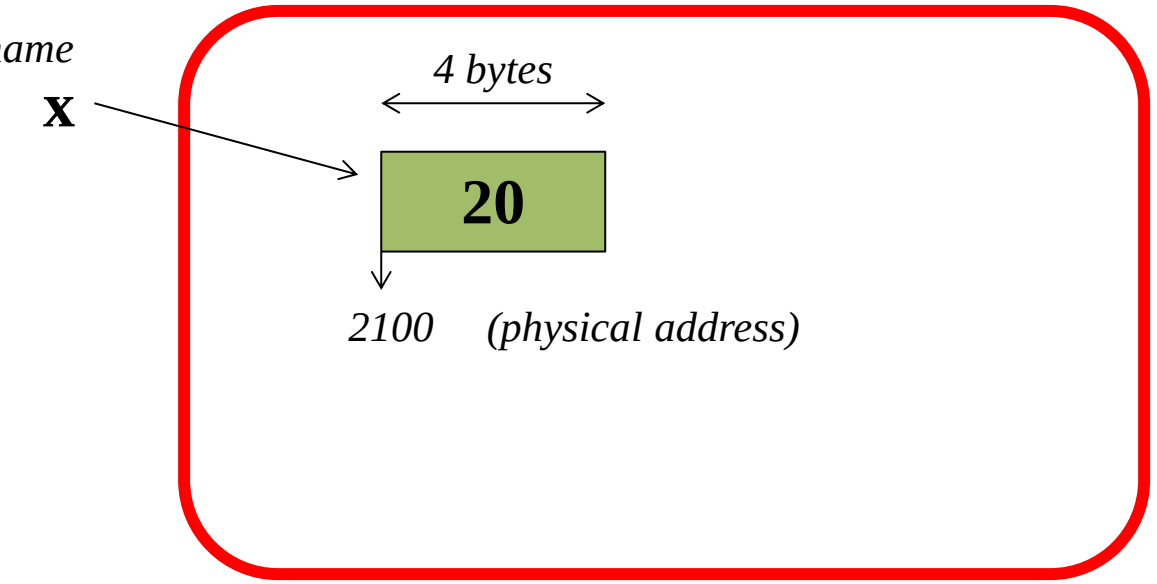
int	4-byte integer
float	4-byte real
char	1 byte character
double	8-byte real etc.

Storing data elements

(type) (variable name)

int x; → 4 bytes allocated in RAM by the operating system

x = 20; $20_{<10>} = 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ _{<2>}$




```
x = 20
&x = 2100
*(&x) = 20
```

address

content

Structure: Table

- Table:
  - Collection of data elements of the **same** type (e.g., of 5 integers)

int x[5]; → 5x4=20 bytes allocated in RAM by the operating system

x[0] = 2; stored in 3200-3

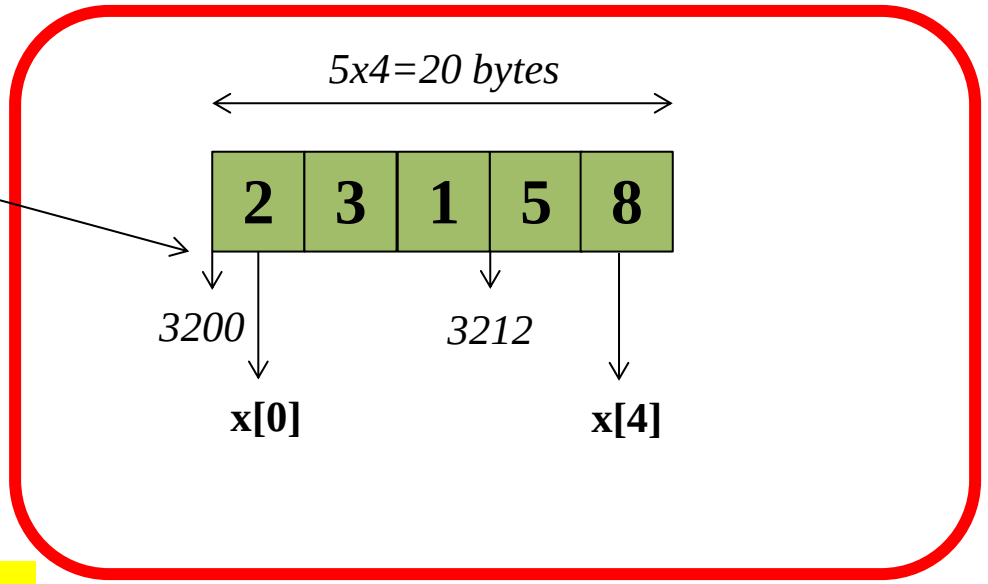
x[1] = 3;

x[2] = 1;

x[3] = 5; stored in 3212-5

x[4] = 8;

3200+3x4=3212



Main memory (RAM)

Direct access to all table elements through the calculated address

Searching the elements in a Table…

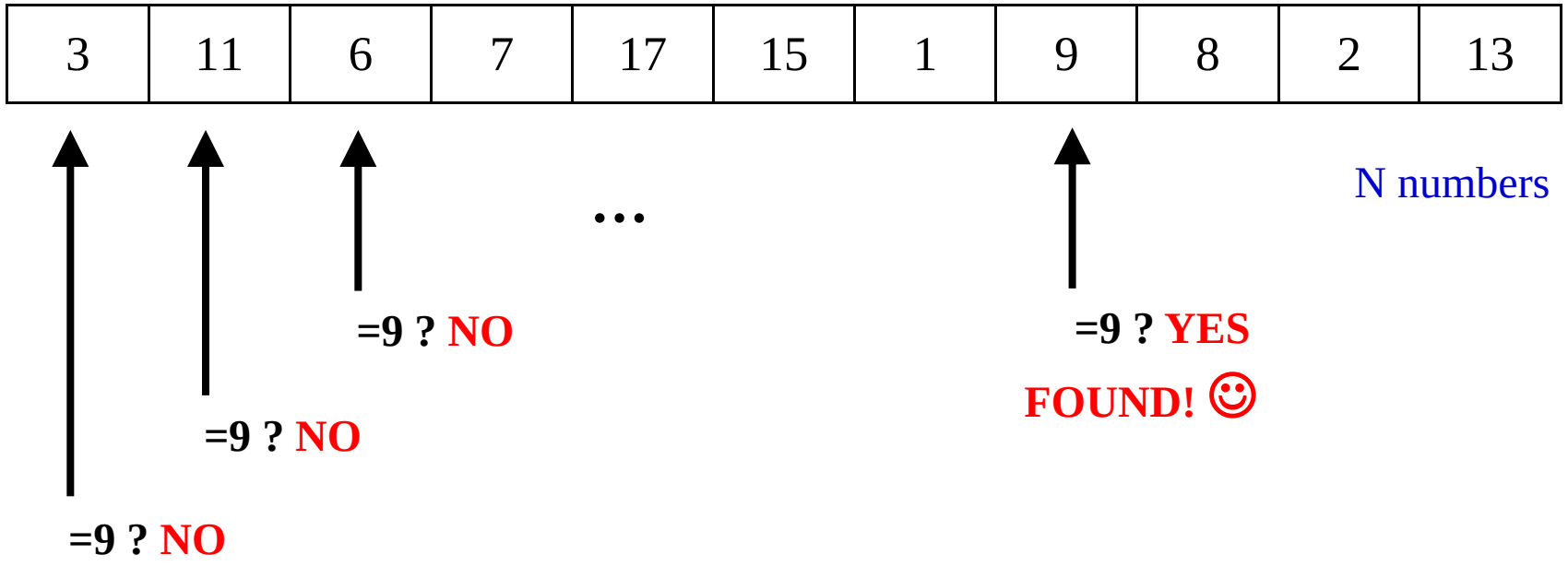
- Traditional databases…
  - Alphanumeric datasets
    - A set of N=11 atomic values

3	11	6	7	17	15	1	9	8	2	13

- Question: Is it “9” present in this set ?

Searching the elements in a Table…

- Algorithm 1: Sequential (serial) Search



– Very slow! ☹

» Complexity: O(N) (i.e., N comparisons)

Searching the elements in a Table…

- Algorithm 2: Binary Search
  - Similar to the phone book search
  - The numbers are **sorted** first

3	11	6	7	17	15	1	9	8	2	13

sorting algorithm

1	2	3	6	7	8	9	11	13	15	17

Searching the elements in a Table…

- Algorithm 2: Binary Search
  - Question: Is it "9" present in this set ?

1	2	3	6	7	8	9	11	13	15	17

Step 1: Go to the middle element

1	2	3	6	7	8	9	11	13	15	17

=9 ? NO

8 < 9

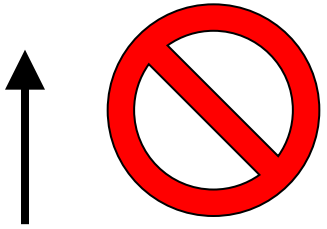
Searching the elements in a Table…

- Algorithm 2: Binary Search
  - Question: Is it “9” present in this set ?

1	2	3	6	7	8	9	11	13	15	17

Step 2: Continue recursively

9	11	13	15	17

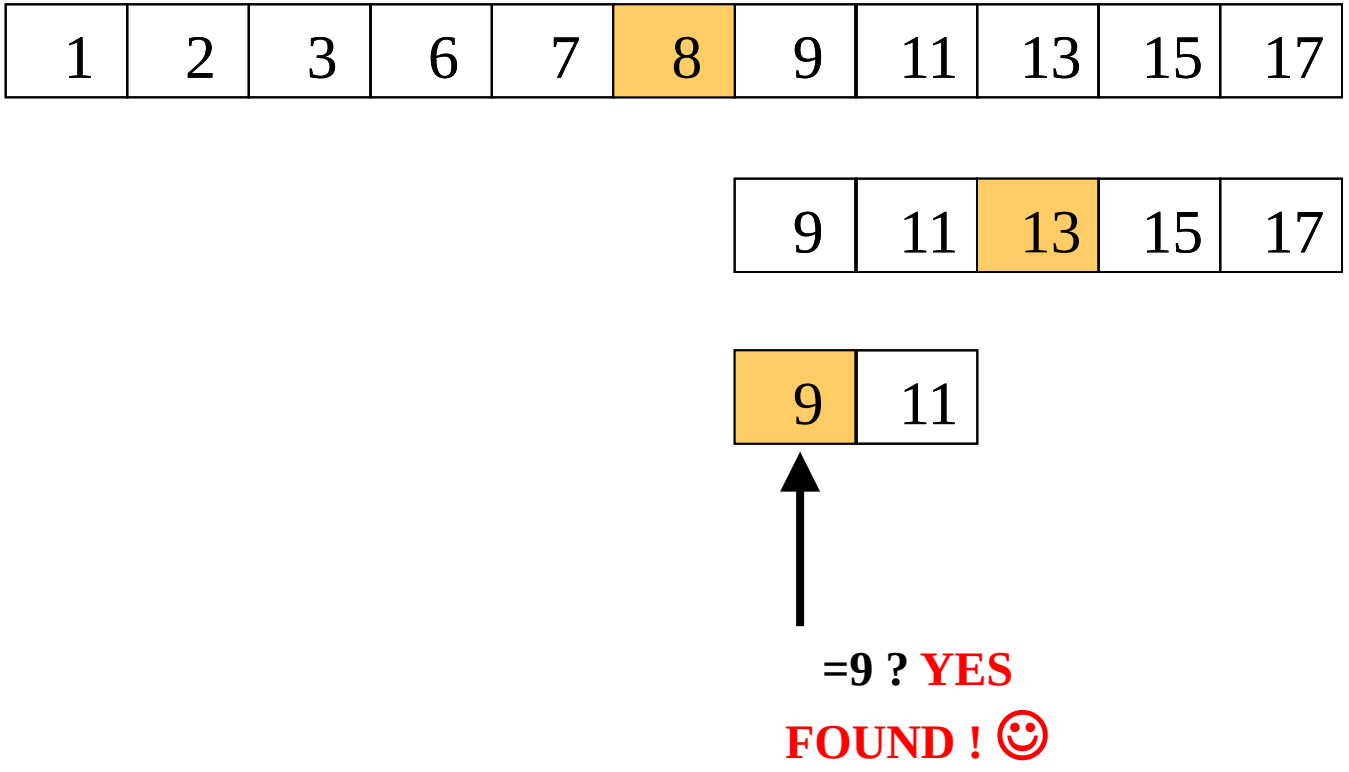


=9 ? NO

13 > 9

Searching the elements in a Table…

- Algorithm 2: Binary Search
  - Question: Is it “9” present in this set ?



Searching the elements in a Table…

- Algorithm 2: Binary Search
  - What's the complexity ?
    - In step **1** → N numbers $N/2^0$
    - In step **2** → N/2 numbers $N/2^1$
    - In step **3** → N/4 numbers $N/2^2$
    - …
    - In step **k** → 1 number (left) $\mathbf{N/2^{k-1} = 1}$

1	2	3	6	7	8	9	11	13	15	17

9	11	13	15	17

9	11

$1 = N/2^{k-1} \rightarrow 2^{k-1} = N \rightarrow$
$\rightarrow \log_2(2^{k-1}) = \log_2(N) \rightarrow$
$\rightarrow k-1 = \log_2(N) \rightarrow k = \log_2 N + 1$

Searching the elements in a Table…

- Algorithm 2: Binary Search
  - Very fast ! ☺
    - Complexity: $O(\log_2 N)$

	Serial search	Binary Search
N	$O(N)$	$O(\log_2 N)$
10	10	3
100	100	7
1,000	1,000	10
10,000	10,000	13
100,000	100,000	17
1,000,000	1,000,000	20

comparisons

Structure: Table

- Advantages:
  - Direct access to its elements
  - Fast search – binary search
    - if items sorted; complexity $O(N\log_2 N)$

- Disadvantage:
  - The structure is **not** appropriate for dynamic data
    - i.e., data that change often

Structure: Table

- It is **not** appropriate for dynamic data
  1. The size of the table must be defined in the beginning of a program

     int x[100]; → a table of 100 integers

     – what if only 10 elements are stored?

     » Waste of space

     – what if 100 elements are stored and a new element appears?

     » Table overflows

Structure: Table

- It is **not** appropriate for dynamic data

2. The binary search requires sorted elements
   - Hard to maintain the table sorted (costly)

1	2	3	6	7	8	9	11	13		

– what if a new element with value 5 arrives?

» All elements right of 3 must be shifted one spot right (costly)

– what if element 3 is deleted?

» All elements right of 3 must be shifted one spot left (costly)

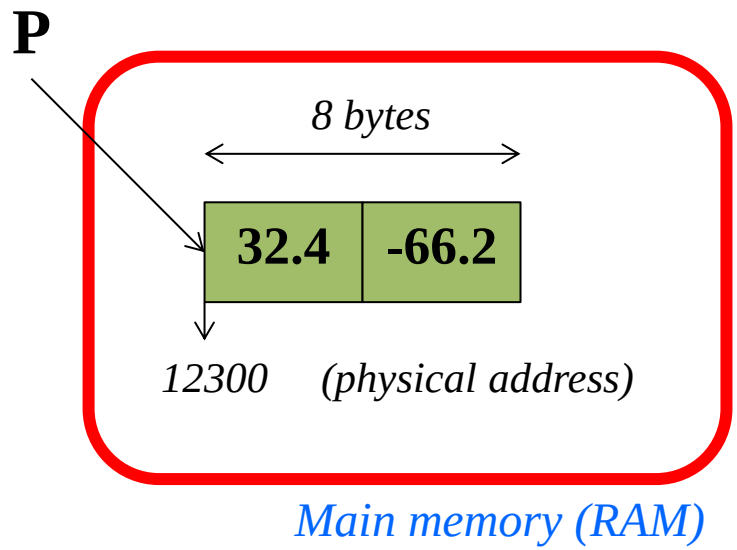
Abstract Data Types (ADT)

- Each programming language
  - offers some built-in data types
    - e.g., int, short, float, double, char
- Developer may define their own types
  - User-defined data types or ADT
    - e.g., point, line, polygon, etc.

Abstract Data Types (ADT)

- Definition of an ADT (in C-like)

```
struct point {
    float x;        ← 4 bytes
    float y;        ← 4 bytes
}
point P;            ← 8 bytes
P.x =32.4;          ← stored in the first 4 bytes
P.y = -66.2;        ← stored in the second 4 bytes
```



Abstract Data Types (ADT)

- Owners table in the Cadastre Database

OWNERS							
ID	**SURNAME**	**NAME**	**BIRTH_DATE**	**STREET**	**NUMBER**	**P_CODE**	**CITY**
46419735	SMITH	JOHN	15/08/1952	EL.VENIZELOU	45	11562	ATHENS
56712945	COOPER	MARY	23/11/1945	DEMOKRATIAS	22	74100	RETHYMNO
...	...	...	...	...	...	...	...

```
struct owner {
    int ID;                 4
    char SURNAME[20];       20
    char NAME[12];          12
    date BIRTH_DATE;        8
    char STREET[30];        30
    int NUMBER;             4
    char P_CODE[5];         5
    char CITY[12];          12
}
```

```
struct date {
    short day;      2
    short month;    2
    int year;       4
}

owner O[100];       definition of a table of records

sizeof(owner) = 95 bytes (each record)
```

Abstract Data Types (ADT)

- Owner's record in the Cadastre Database

OWNERS							
ID	SURNAME	NAME	BIRTH_DATE	STREET	NUMBER	P_CODE	CITY
46419735	SMITH	JOHN	15/08/1952	EL.VENIZELOU	45	11562	ATHENS
56712945	COOPER	MARY	23/11/1945	DEMOKRATIAS	22	74100	RETHYMNO
...	...	...	...	...	...	...	...

owner O[100]; table of records → 9500 bytes or 9.5KB

Populate the first row in the table:

O[0].ID = 46419735;

O[0].SURNAME = "SMITH";

O[0].NAME = "JOHN";

O[0].BIRTH_DATE.day = 15;

O[0].BIRTH_DATE.month = 8;

O[0].BIRTH_DATE.year = 1952;

O[0].STREET[30] = "EL. VENIZELOU";

O[0].NUMBER = 45;

O[0].P_CODE = "11562";

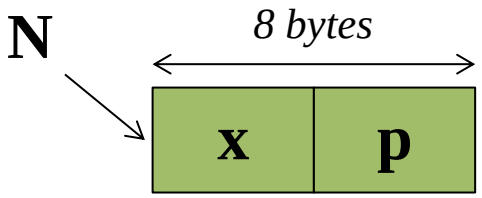
O[0].CITY = "ATHENS";

95 bytes needed

Structure: Linked-List

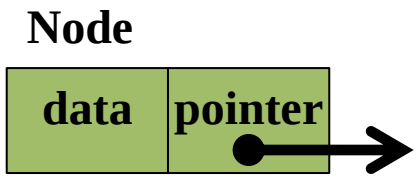
- This is an alternative structure …
  - for storing a collection of data elements of the same type (built-in or ADT)
    - data are maintained in a list
  - The linked-list makes use of an ADT called node:

```
struct node {
        int x;      ← data element (4 bytes)
        int *p;     ← address of the next element in RAM (4bytes)
}
node N;             ← definition of a new node
```



Structure: Linked-List

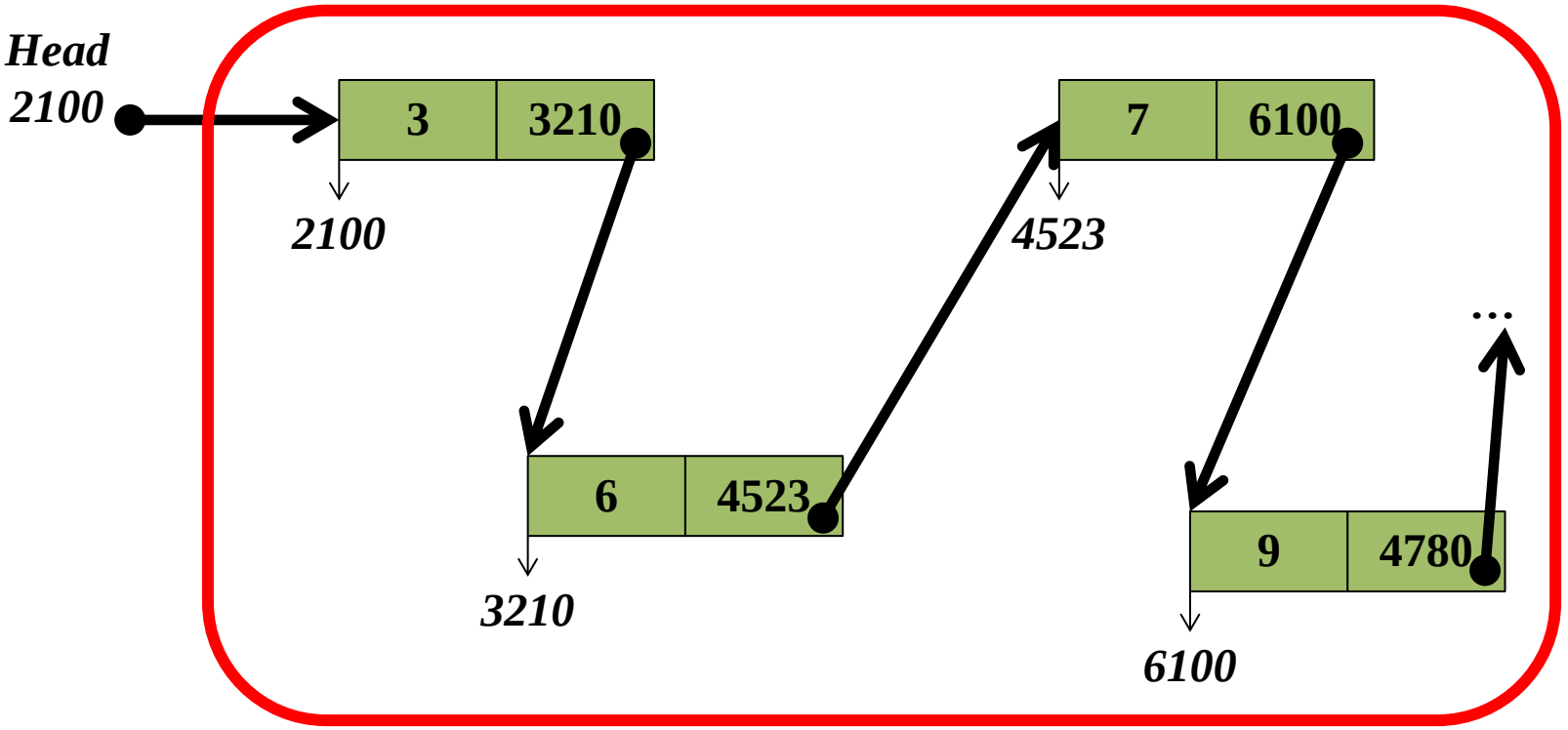
- A node accommodates…
  - one **data** element, and
  - the address of the next node (one **pointer**)



- To store N data elements…
  - N nodes are required
  - these nodes are stored **scattered** in the RAM
    - any new node is stored randomly in the RAM and pointed by the previous node in the list
  - all nodes can be accessed through the **head node**

Structure: Linked-List

- A linked list example…



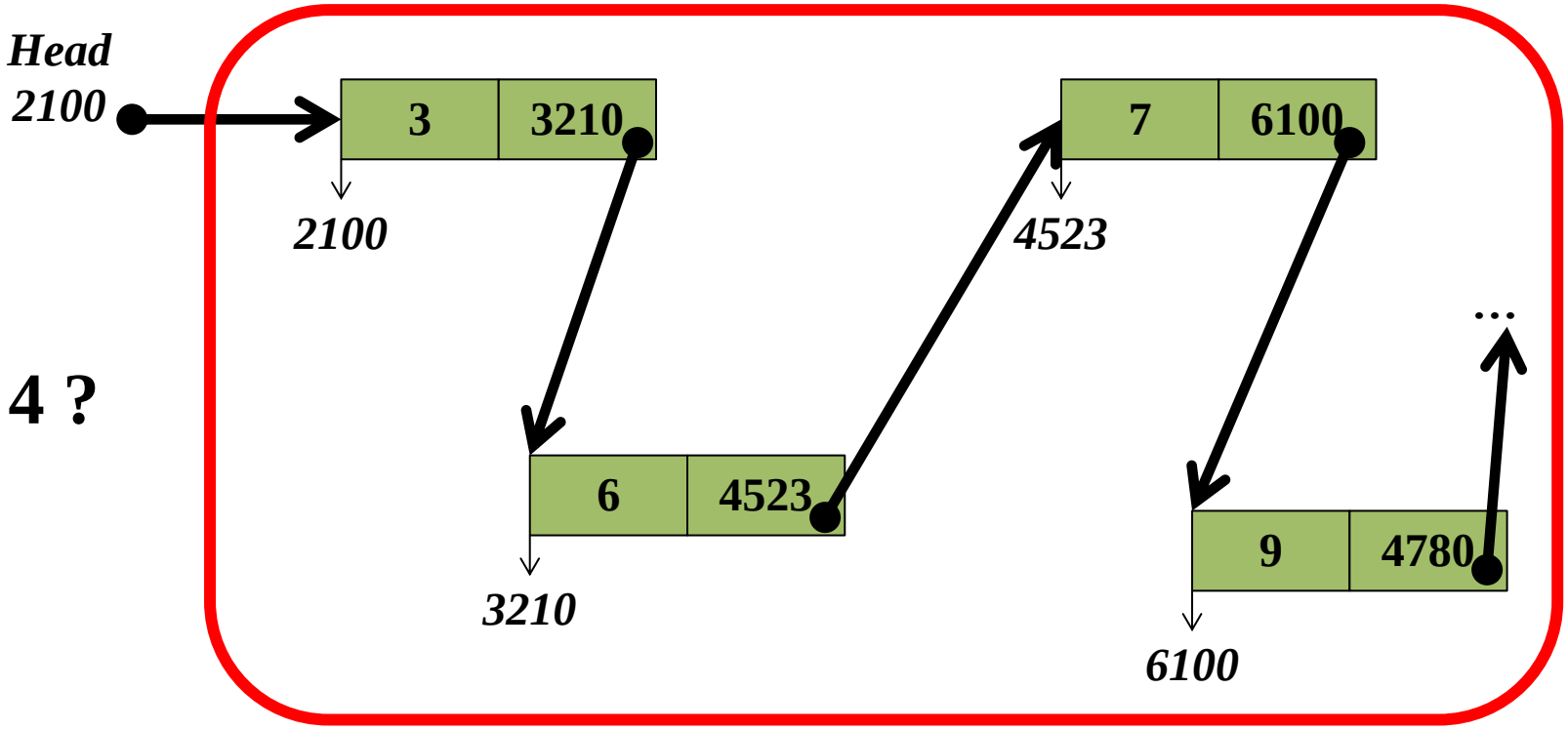
Main memory (RAM)

Structure: Linked-List

- Advantages
  - Insertions and deletions of nodes (i.e., data elements are easily handled)
  - A sorted list can be easily maintained
    - through appropriate arrangement of pointers
  - The list size is always appropriate and dependent on the stored list elements
    - i.e., 10 elements → 10 nodes
      100 elements → 100 nodes
      no overflow occurs

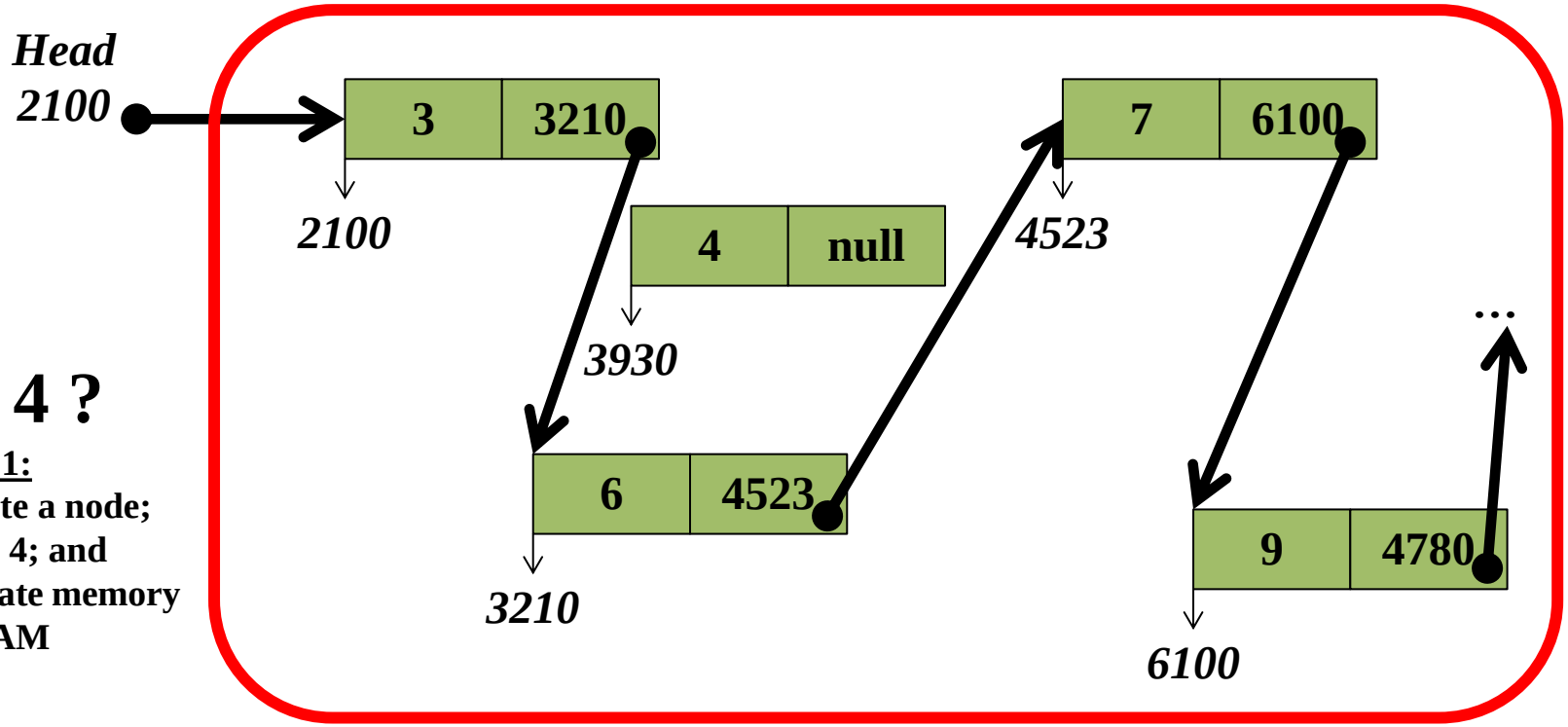
Structure: Linked-List

- Example: insert data element 4…
  - **The list must be maintained sorted**



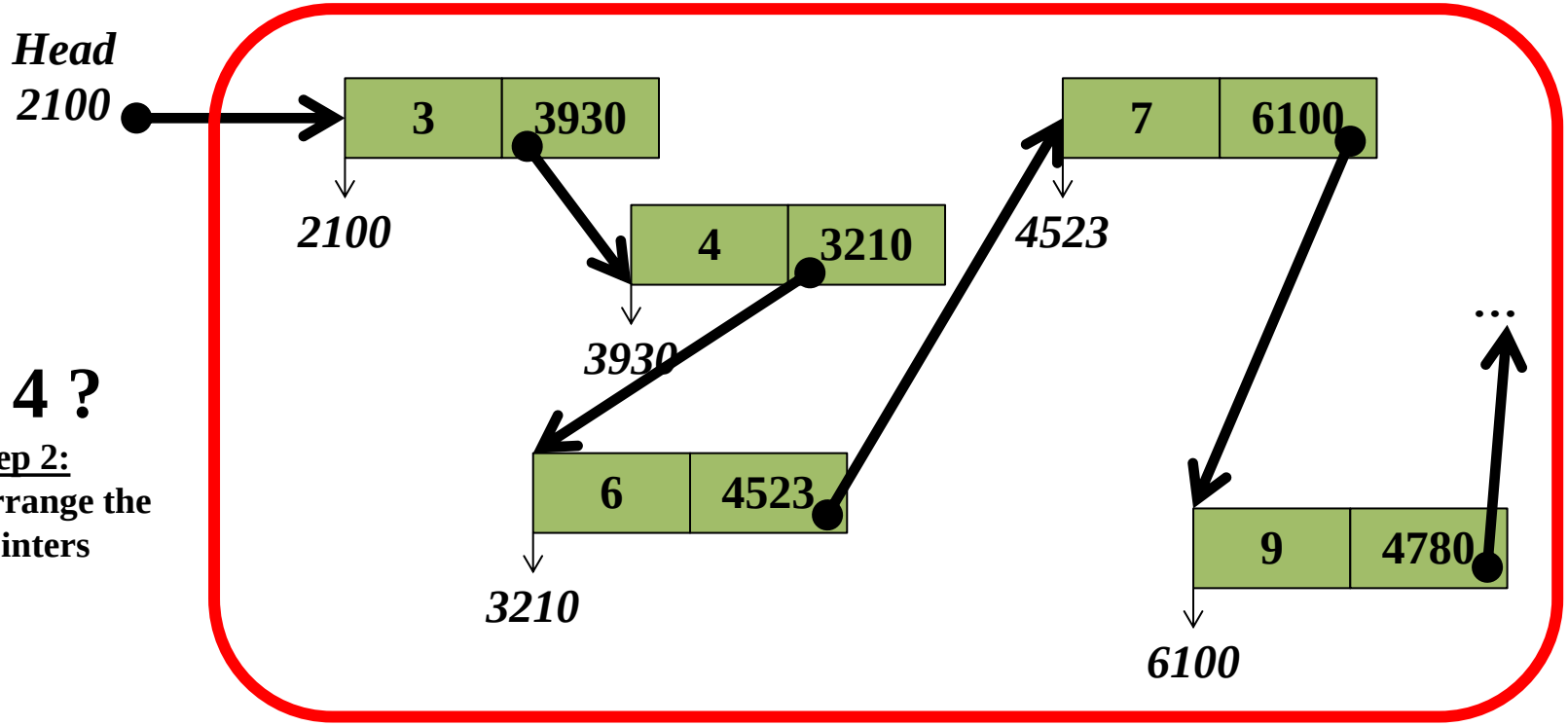
Structure: Linked-List

- Example: insert data element 4…
  - **The list must be maintained sorted**



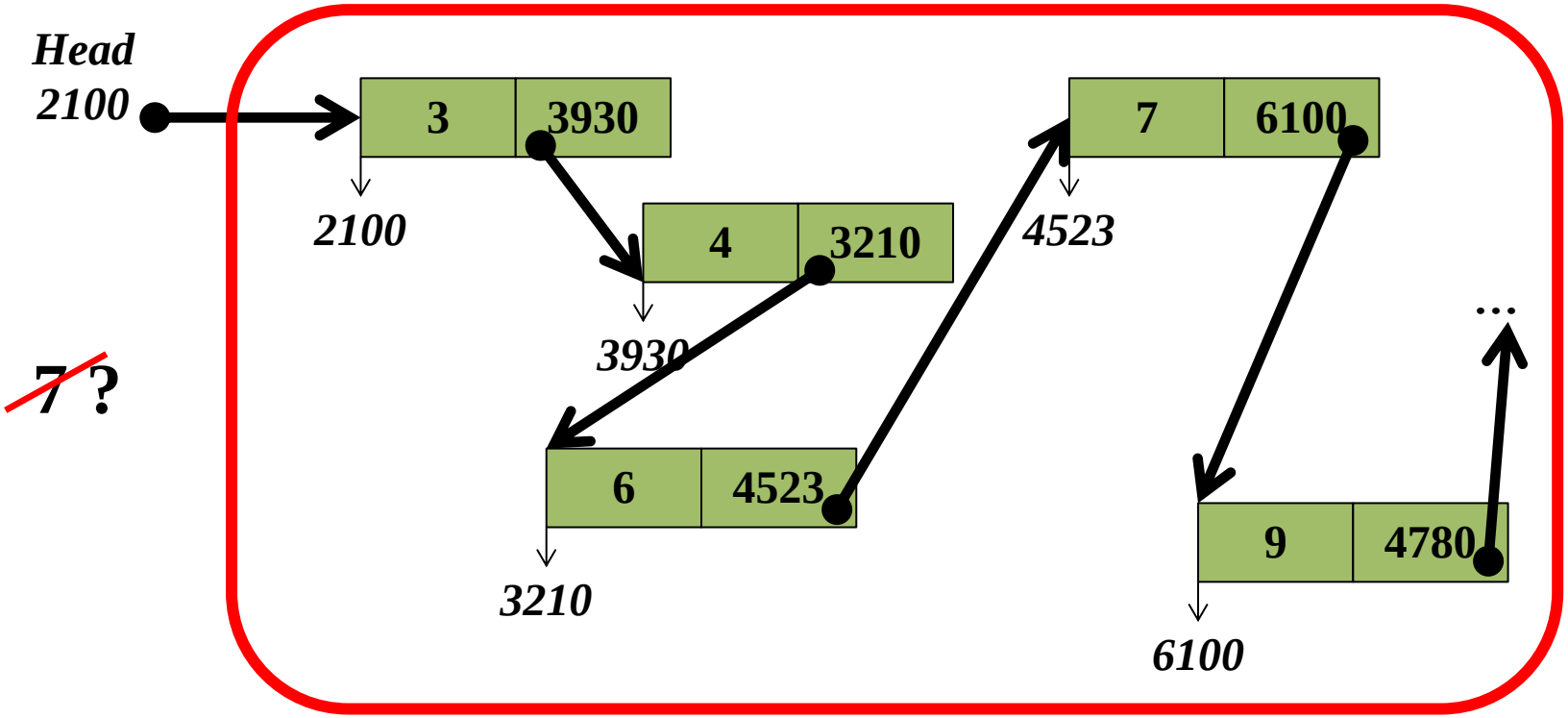
Structure: Linked-List

- Example: insert data element 4…
  - **The list must be maintained sorted**



Structure: Linked-List

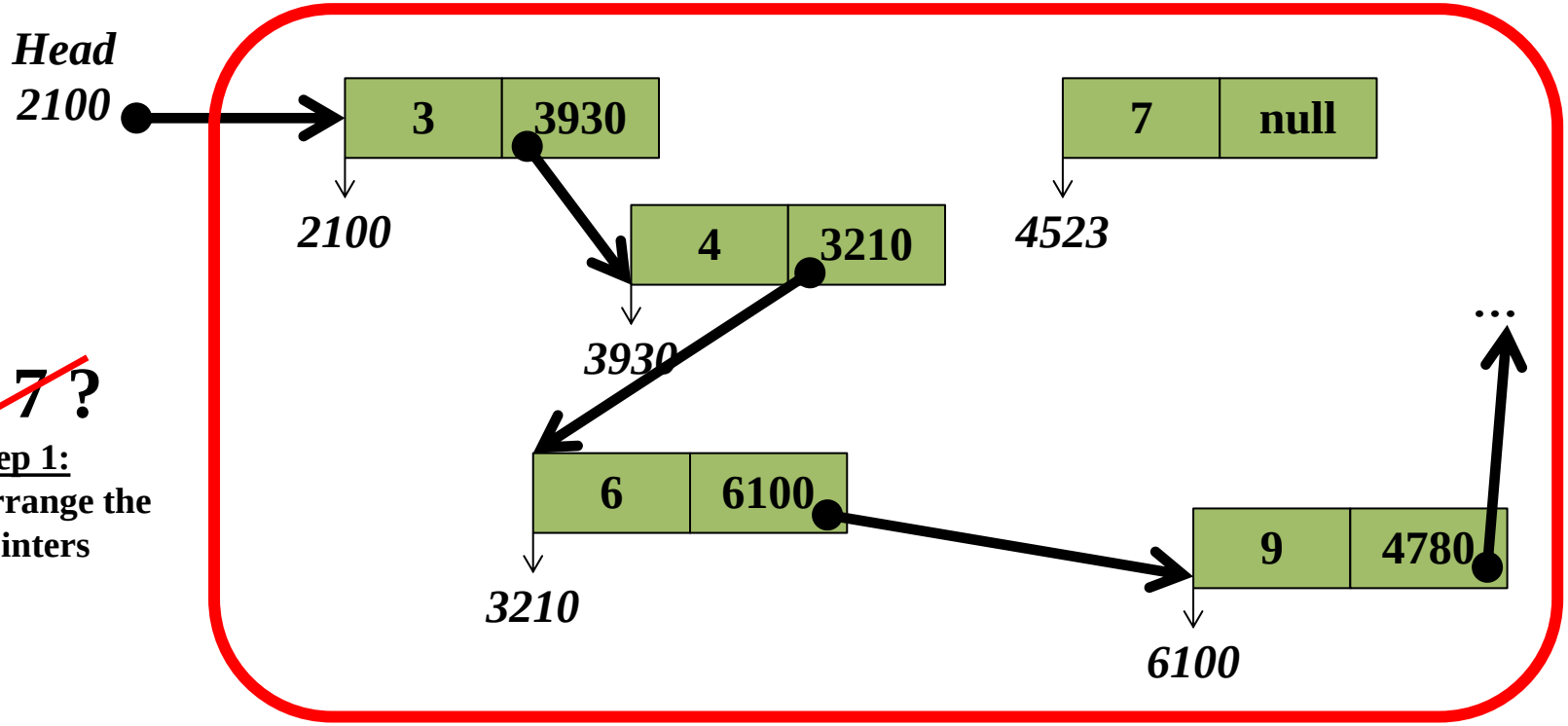
- Example: delete data element 7…
  - **The list must be maintained sorted**



Main memory (RAM)

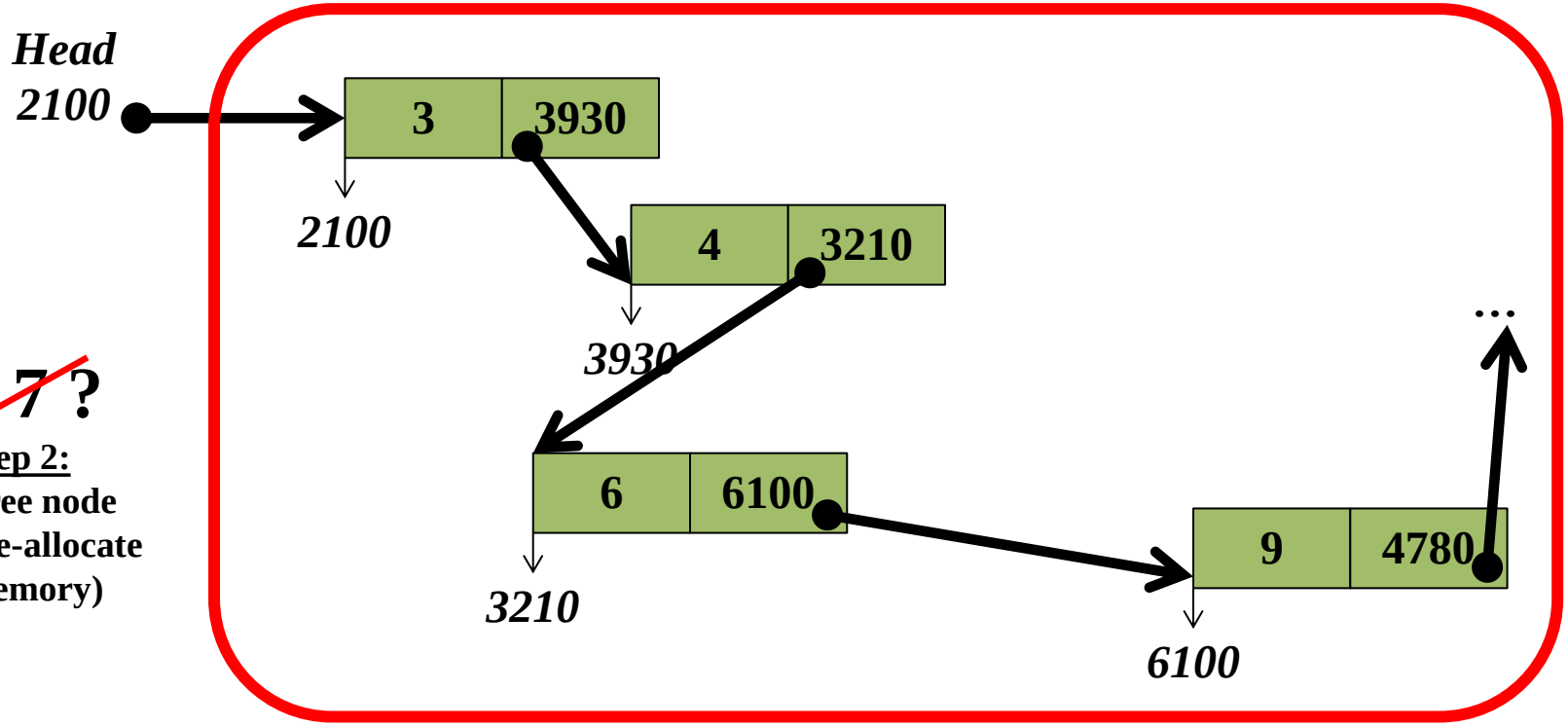
Structure: Linked-List

- Example: delete data element 7…
  - **The list must be maintained sorted**



Structure: Linked-List

- Example: delete data element 7…
  - **The list must be maintained sorted**



Structure: Linked-List

- Disadvantage…
  - the binary search cannot be applied…
    - … although the list is sorted
    - … any node is reachable through head only
  - only serial search !

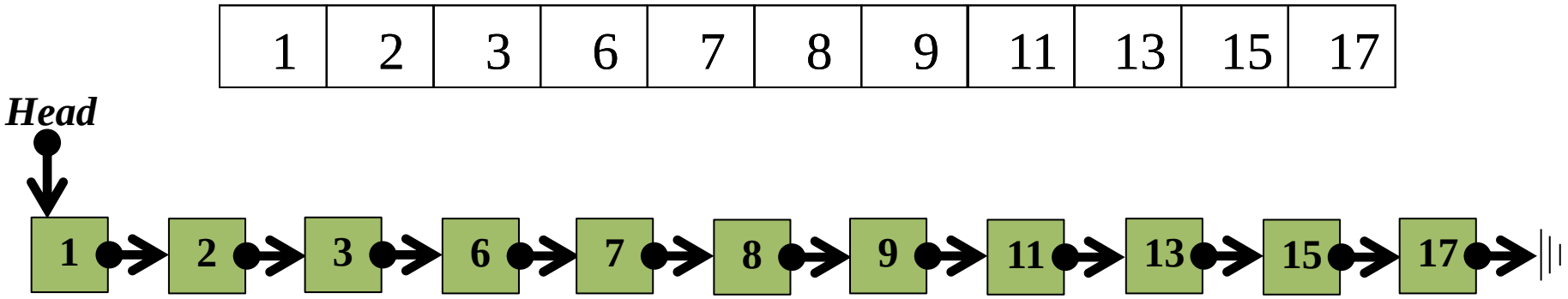


Table (vs) Linked-List

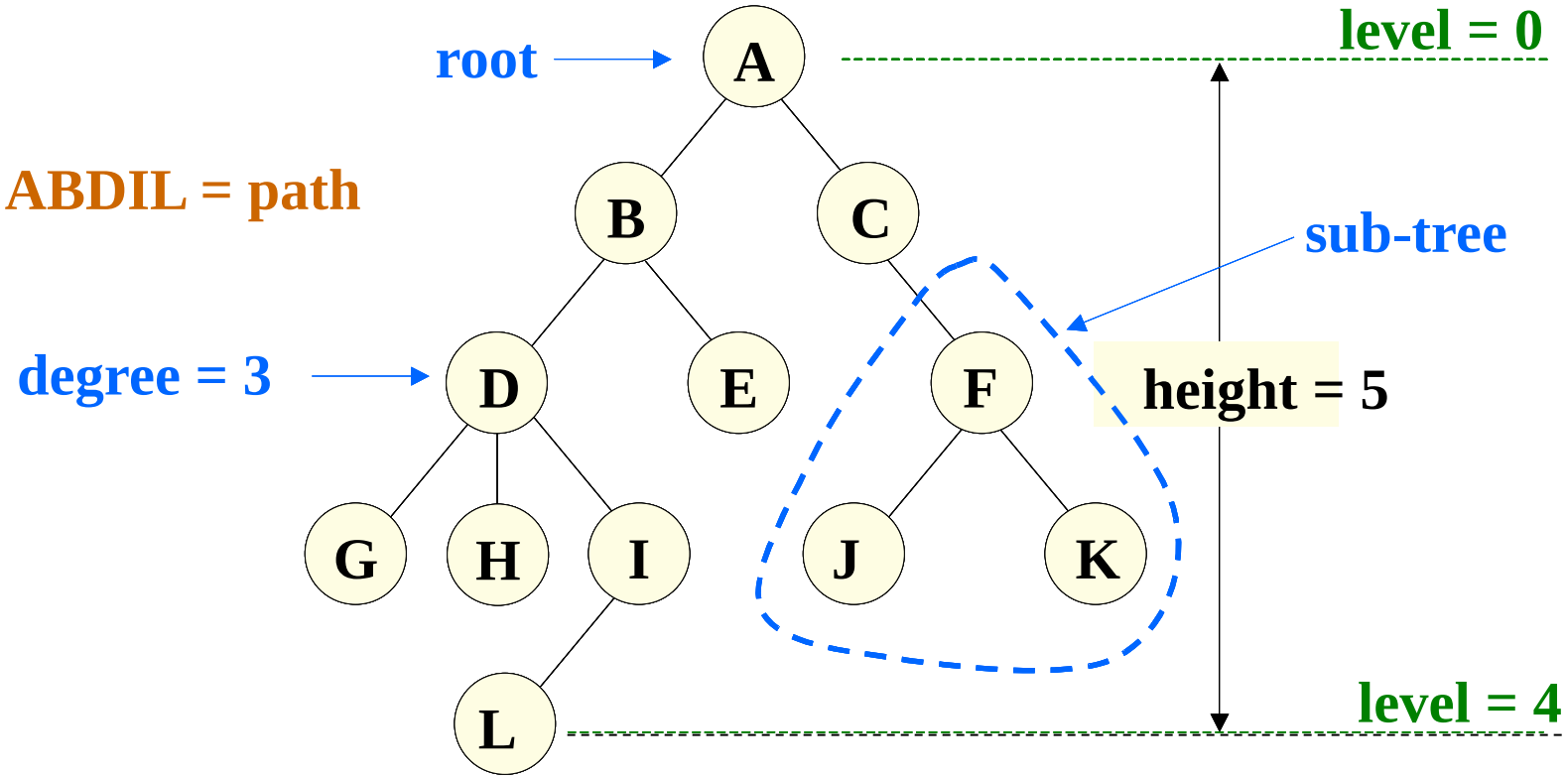
- Tables
  - Support fast search (binary search)
  - Inefficient dynamic data handling
- Linked-Lists
  - Efficient dynamic data handling
  - Fast search not supported
- Need to combine the advantages
  - i.e., fast search and efficient handling → **Trees**

Structure: Tree

- Trees are…
  - Hierarchical and non-linear (multi-dimensional) linked-lists
  - Consist of nodes
    - Each node has
      - One precedent
      - Zero, one or more descendant
  - All nodes are accessed through the root (head) only

Structure: Tree

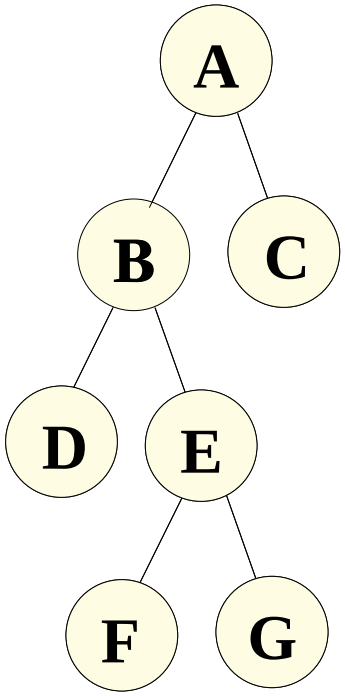
- Example tree…



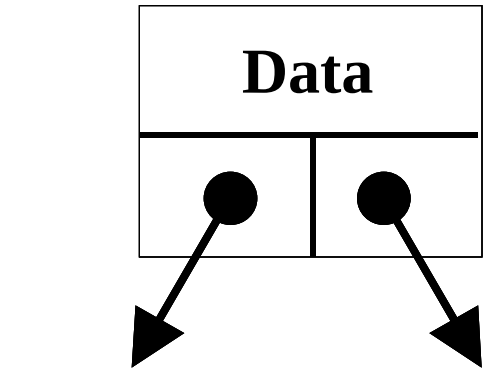
Leaf-nodes: { G,H,L,E,J,K }

Structure: Tree

- A binary tree…



Node structure



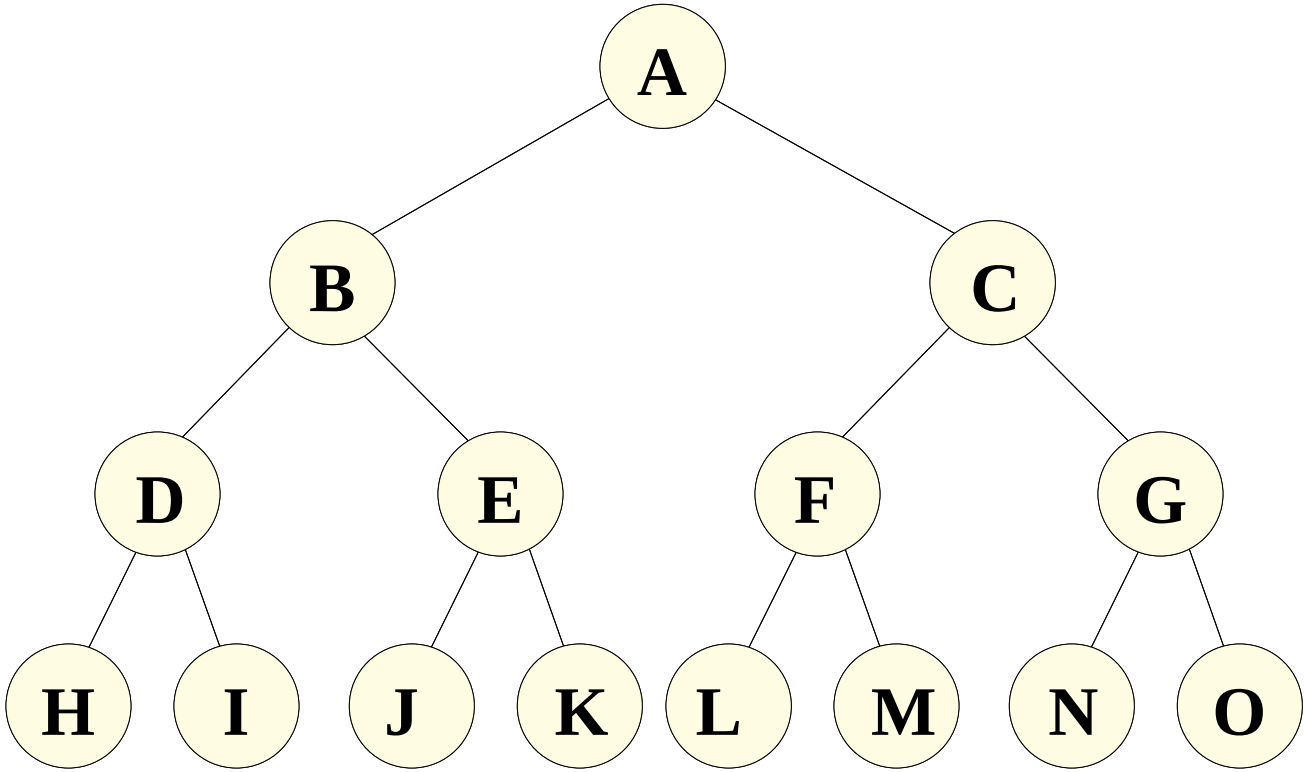
Pointer to the Left child

Pointer to the Right child

All nodes of degree 2; i.e., 2 children per node (maximum)

Structure: Tree

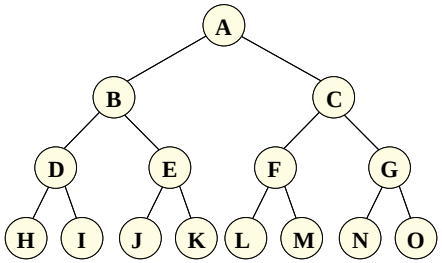
- A **full** and **balanced** binary tree…



All leaf-nodes at the same level. All non-leaf nodes have two children.

Structure: Tree

- A **full** and **balanced** binary tree…
  - Height (h) vs Number of nodes (N)

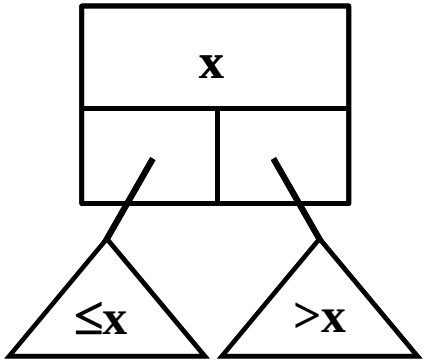


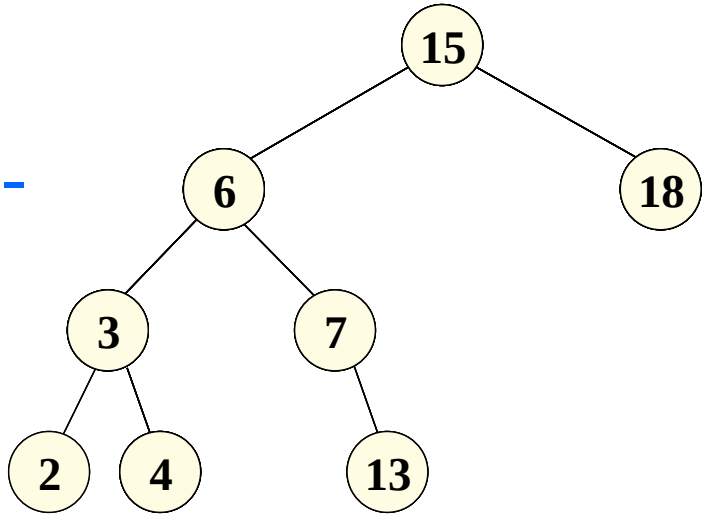
Tree level	Nodes in this level	Sum of nodes from the root to this level	Height
0 (root)	$2^0 = 1$	2^0	0
1	2^1	$2^1+(2^1-1)$	1
2	2^2	$2^2+(2^2-1)$	2
...	...		...
h	2^h	$2^h+(2^h-1)$	h

$$\mathbf{2^h+(2^h-1) = N \Rightarrow 2^{h+1} = N+1 \Rightarrow h = \log_2(N+1)-1}$$

Structure: Tree

- The binary search tree…
  - It is a binary tree
  - For each node,
    - data values in the **left** sub-tree are less than x (15)
    - data values in the **right** sub-tree are greater than x (15)

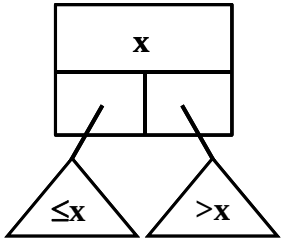


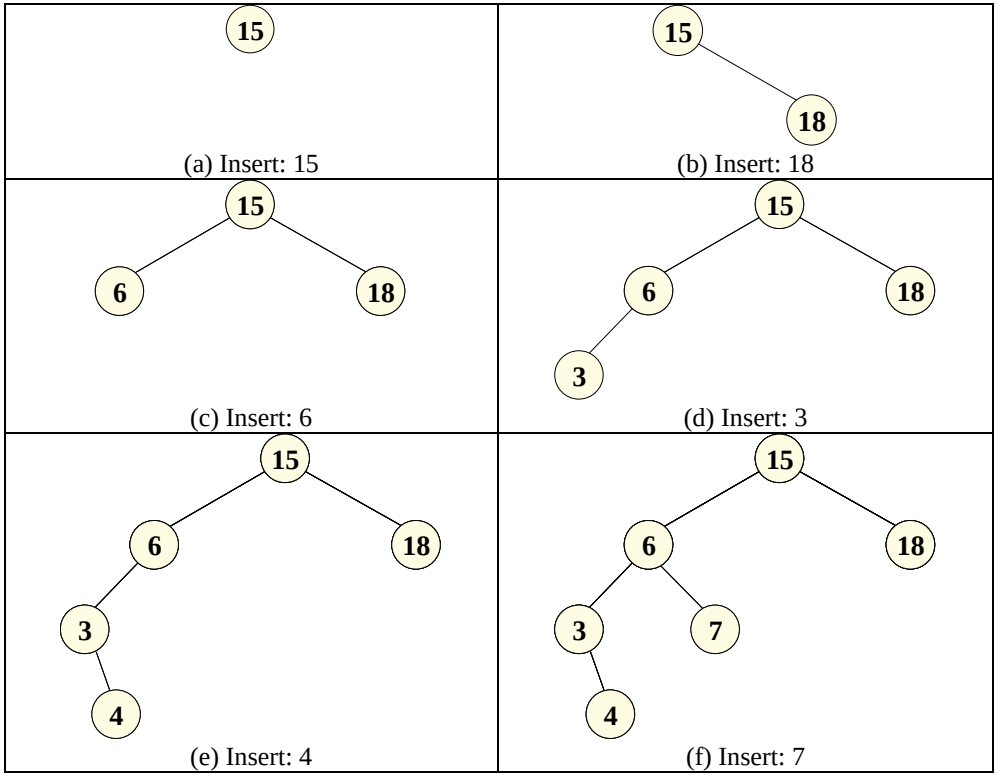


Structure: Tree

- Building the binary search tree…

  Insert: { 15, 18, 6, 3, 4, 7, 13, 2, 20, 9, 17 }

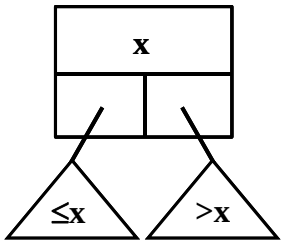


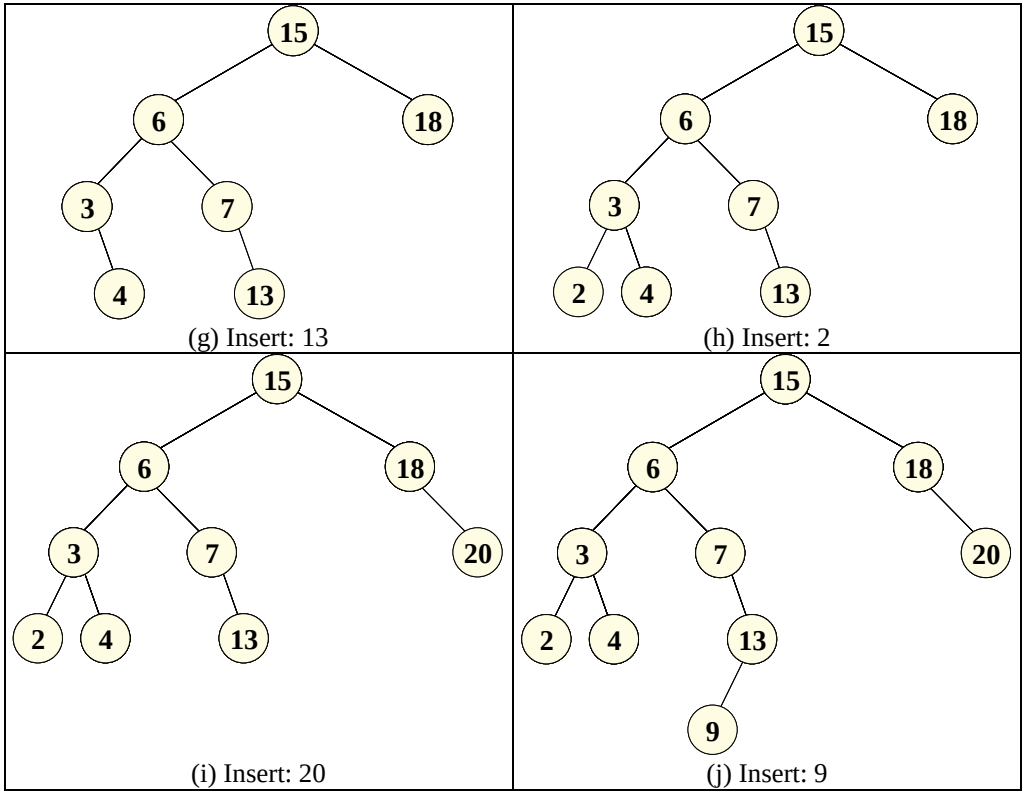


Structure: Tree

- Building the binary search tree…

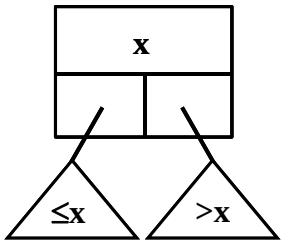
  Insert: { 15, 18, 6, 3, 4, 7, 13, 2, 20, 9, 17 }



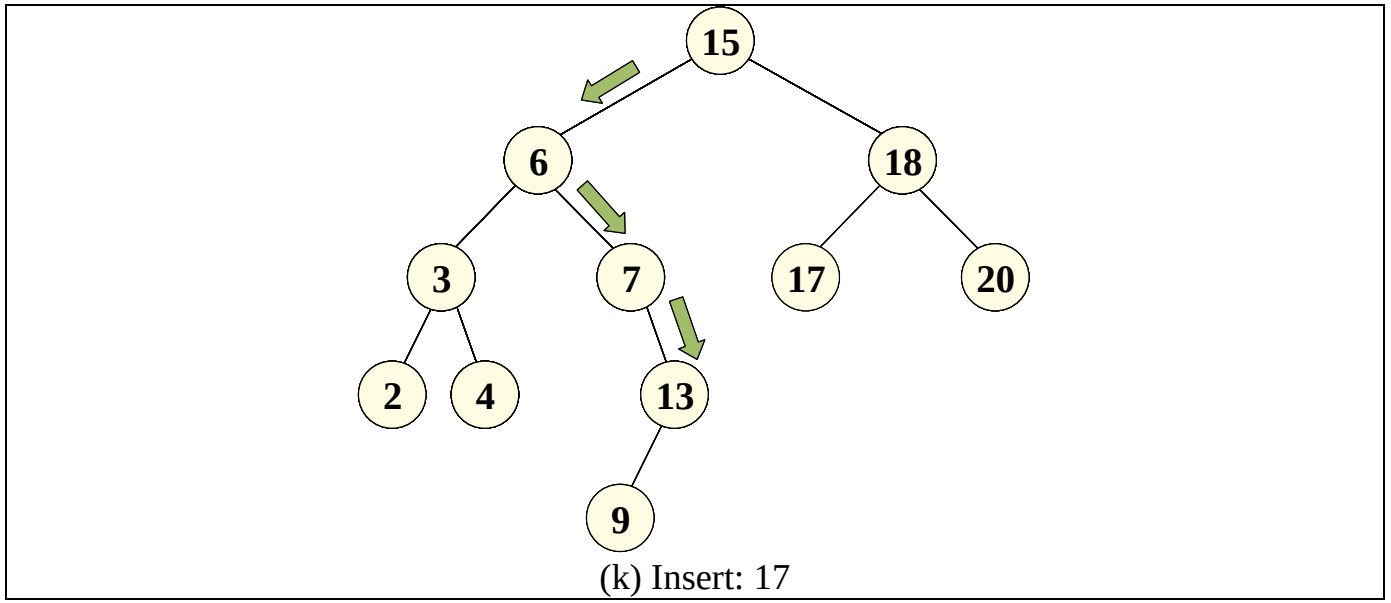


Structure: Tree

- Building the binary search tree…



Insert: { 15, 18, 6, 3, 4, 7, 13, 2, 20, 9, 17 }



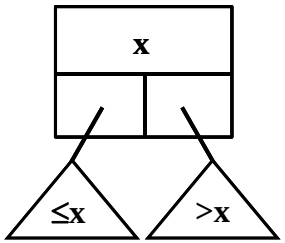
(k) Insert: 17

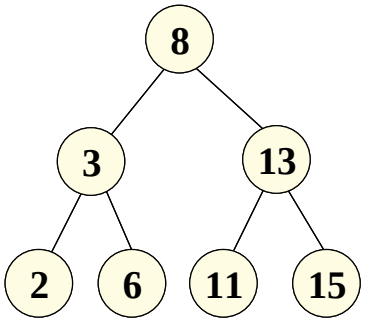
Search for: 13 (follow the arrows)

if full and balanced → search in $O(\log_2 N)$

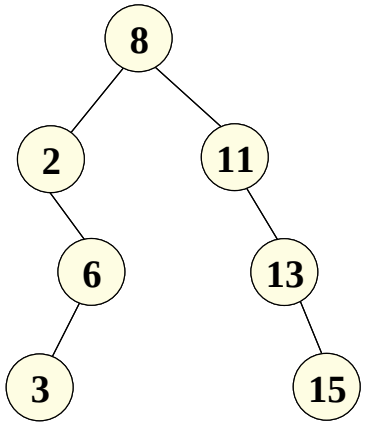
Structure: Tree

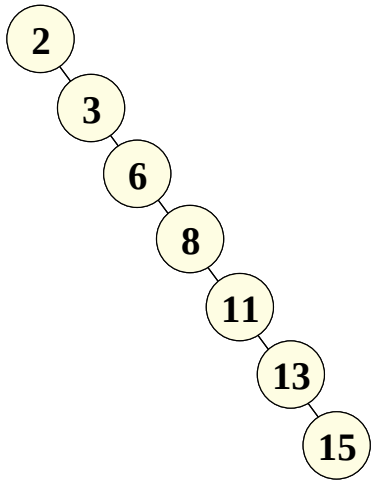
- Building the binary search tree…
  - depends on the order the elements are inserted





best structure





  - these three trees contain the same elements:

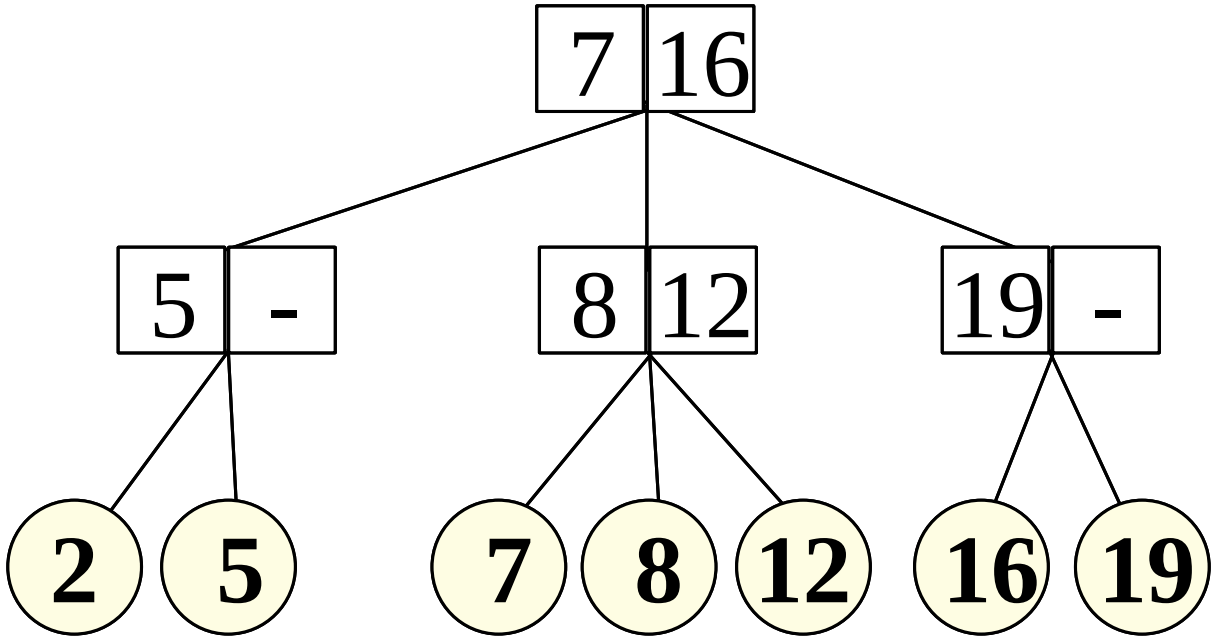
    {2,3,6,8,11,13,15}

Structure: Tree

- Tree 2-3:
  - Always balanced
  - Logarithmic search guaranteed – $O(\log_2 N)$
  - Data elements reside in the leaf-nodes only
  - Non-leaf nodes are index nodes
  - Each non-leaf node may have 2 or 3 children
  - Each leaf node may accommodate 2 or 3 elements

Structure: Tree

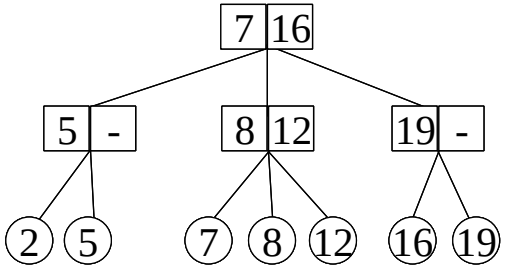
- Tree 2-3:



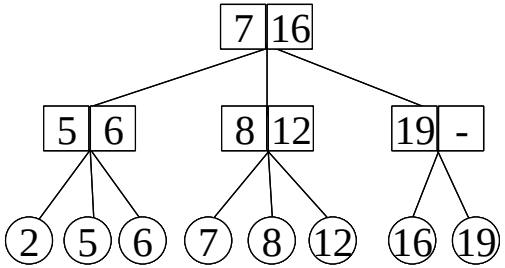
$\log_3 N$ < height < $\log_2 N$

Structure: Tree

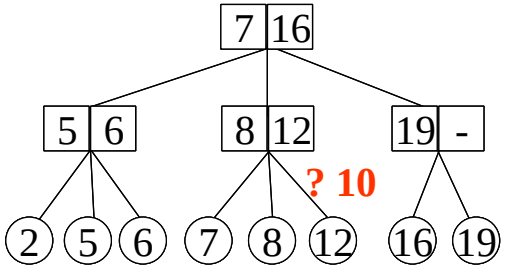
- Tree 2-3:
  - the tree grows from the top to remain balanced



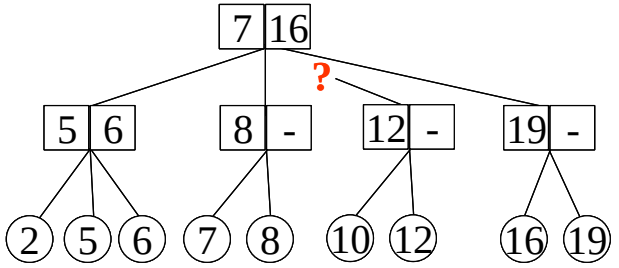
Initial tree



Insert 6



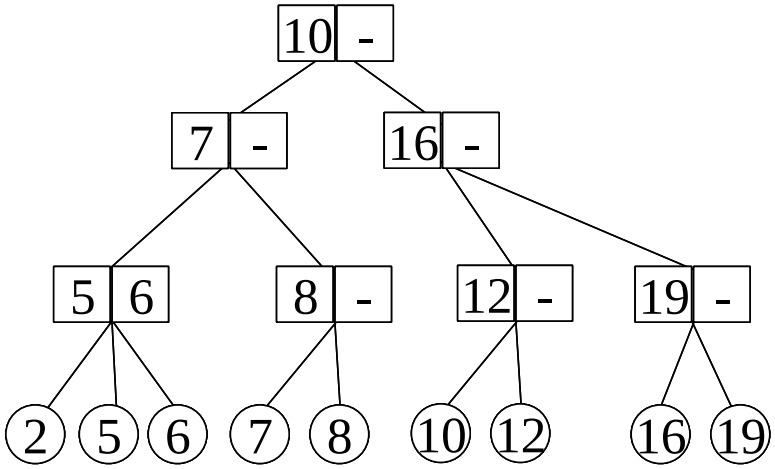
Insert 10



Node split to insert 10

Structure: Tree

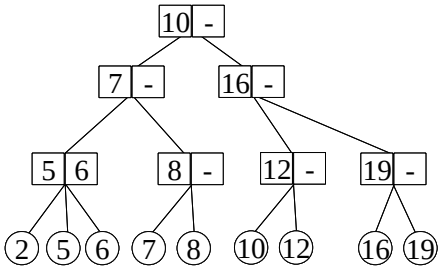
- Tree 2-3:
  - the tree grows from the top to remain balanced



10 inserted; the tree has grawn one level

Structure: Tree

- Tree 2-3:
  - Tree height → complexity < $O(\log_2 N)$



    - Worst case: all nodes have only 2 children

In level 0 (leaves) $N/2^0 = N$ nodes

In level 1 $N/2^1 = N/2$ nodes

In level 2 $N/2^2$

...

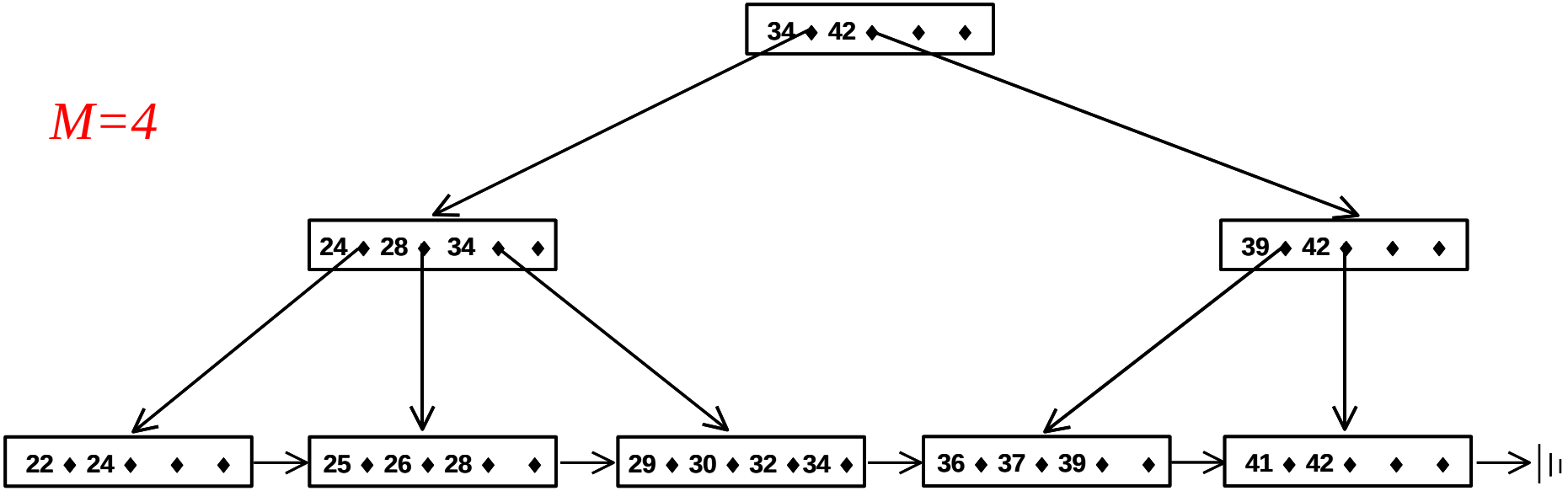
In level h (root) $N/2^h = 1$ node

$N/2^h = 1 \rightarrow h = \log_2 N$

Structure: Tree

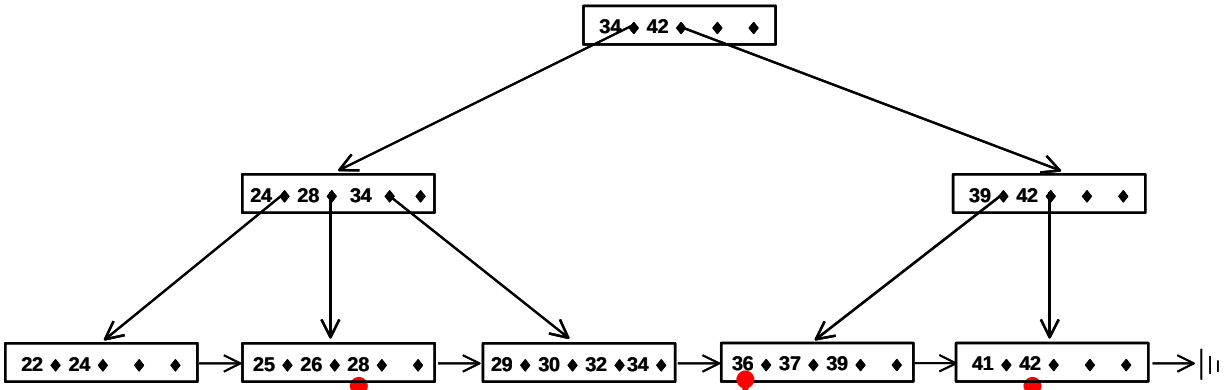
- B+tree:
  - Extending the idea of tree 2-3.
  - Used in DBMS to index alphanumeric data
  - Each node has between M and M/2 children (e.g., M=100)

M=4



B+Tree Index Structure

- Indexing the attribute table…



OWNERS

ID	NAME	AGE
22	LOLA	34
41	NINI	23
39	KIKI	24
24	ZIZI	67
26	PAPA	21
42	MAMA	76
39	LALA	45
37	WIWI	34
32	RIRI	67
36	TOTO	24
34	SASA	39
30	PEPE	19
28	BOBO	49
25	ZOZO	72

pointer to the table record

An index is built on attribute: ID

UNB

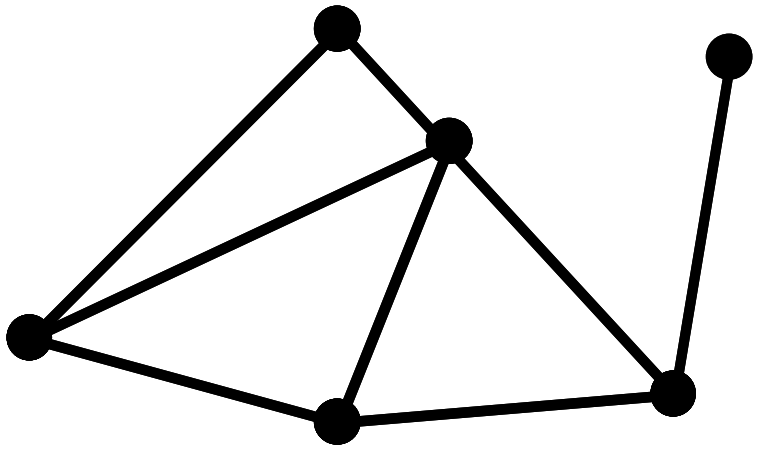
Introduction to
Graph Theory

Emmanuel Stefanakis

estef@unb.ca

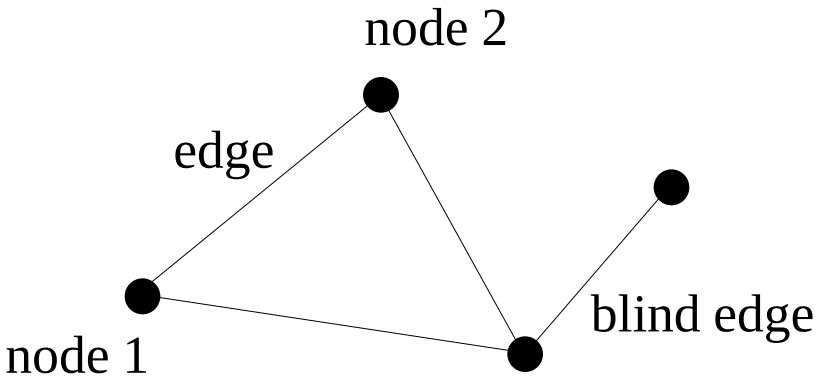
Graphs…

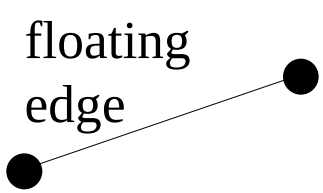
- Definition…
  - A graph G(N,E)
    - is a collection of
      - Nodes N, and
      - Edges E
    - can simulate…
      - a road network
      - the countries of a continent (nodes) and their topological relationships (edges)

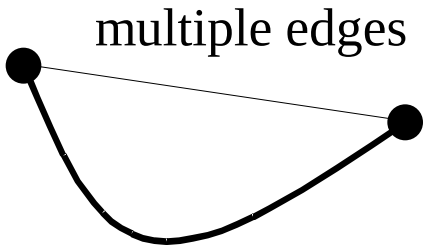


Graphs…

- Types of nodes and edges…



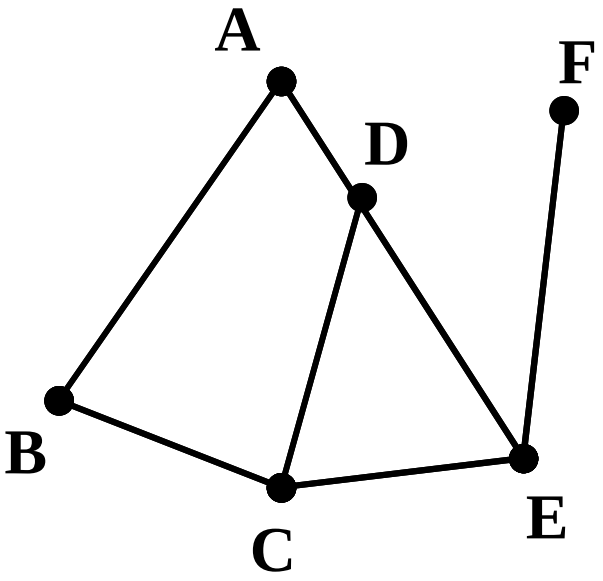




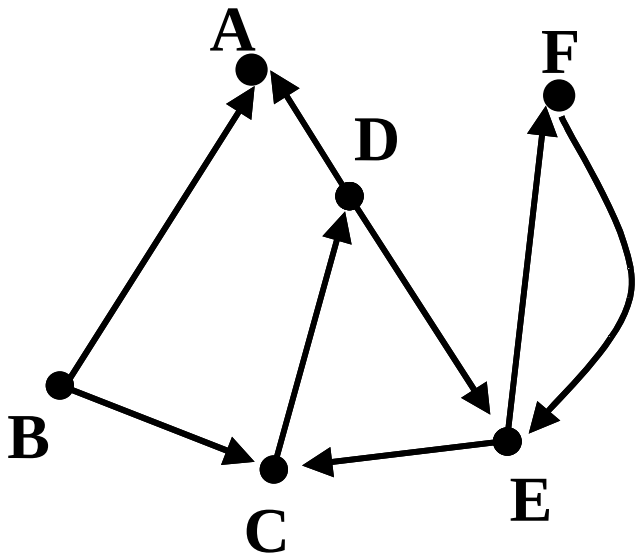
node 1 is “adjacent” to node 2
(connected through a single edge)

Graphs…

- Types of graphs…



A simple “non-directed” graph

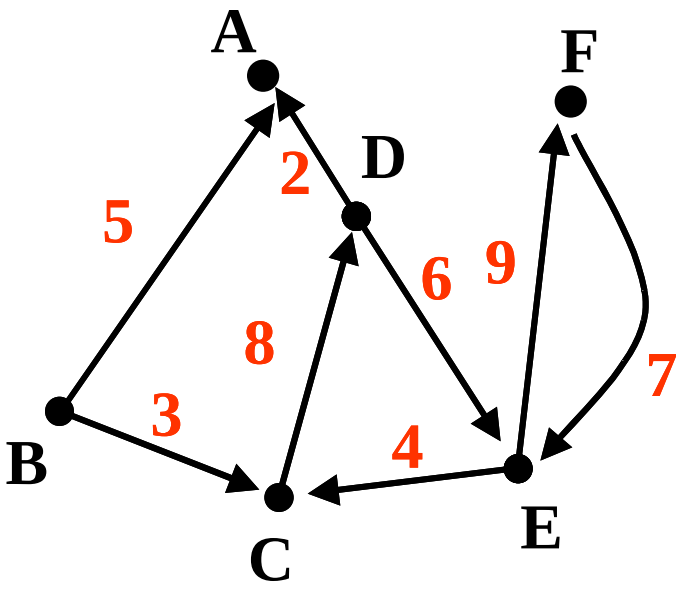


A “directed” graph

Graphs…

- Types of graphs…

The weights may represent: time, distance, fuel consumption, etc.

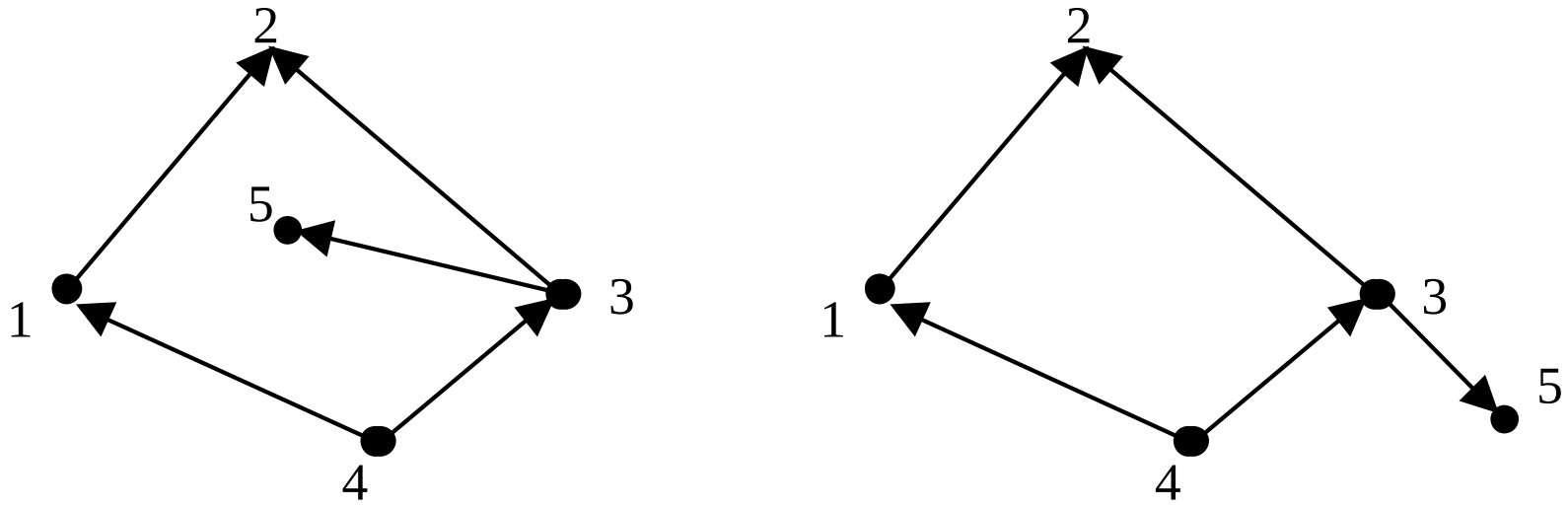


A “directed and weighted” graph

moving from node B to E (through nodes C and D) costs: 3+8+6=17

Graphs…

- The definition of a graph…
  - does not imply its graphical representation



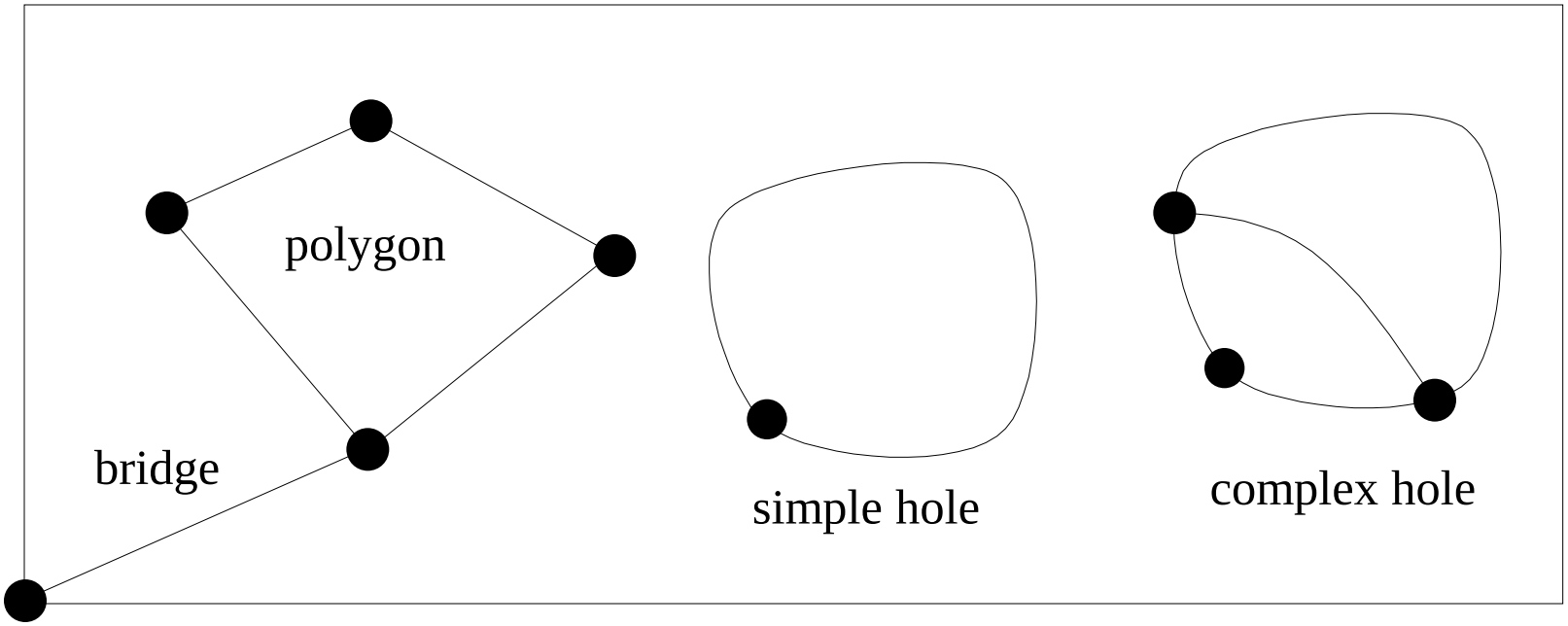
Both graphs G(N,E) are defined as:
N: {1, 2, 3, 4, 5}; E (directed): {(1,2), (3,2), (4,3), (4,1), (3,5)}

Graphs…

- The concept of "Polygon"
  - A closed chain of edges form a polygon
  - Any edge is the border of two polygons
  - A directed edge has a left and a right polygon
  - A polygon may have holes (or islands)
  - A hole may be simple (if it consists of one edge with the same start and end node) or complex (if it consists of more than one edges
  - An edge is called bridge edge, if when deleted a hole is created → non-connected graphs

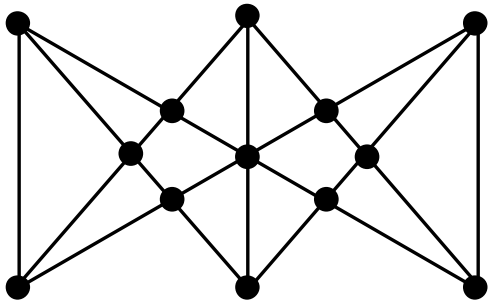
Graphs…

- The concept of “Polygon”

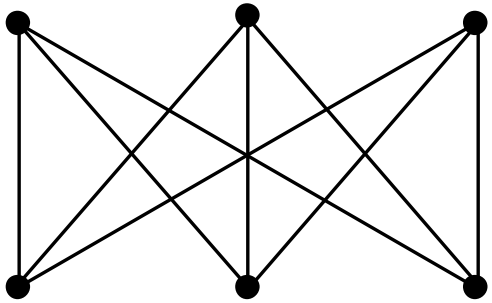


Graphs…

- Planar and non-planar graphs…
  - Planar graphs…
    - When two edges intersect → node



  - Non-planar graphs…
    - Two edges may intersect without any node



Planar Graphs…

- In Planar graphs…
  - The concept of polygon is meaningful…
  - Euler criterion
    - In planar graphs: **p – e + n = 2**
      - p: number of polygons
      - e: number of edges
      - n: number of nodes

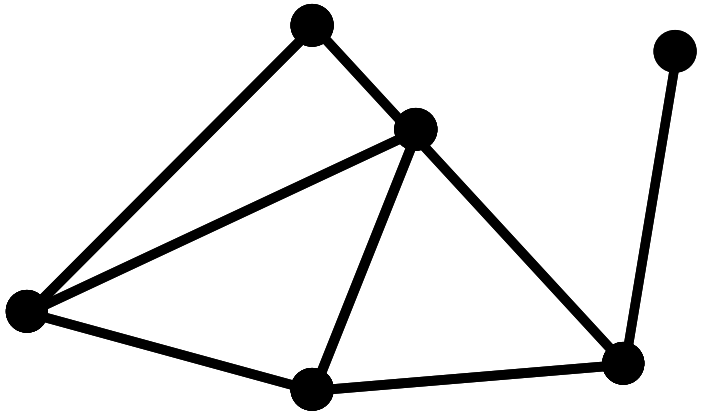
Planar Graphs…

- Euler criterion
  - In planar graphs: **p – e + n = 2**

    p = 4

    e = 8 → 4 – 8 + 6 = 2 (ok!)

    n = 6



Check the consistency of the topology
(a necessary but not sufficient condition)

Planar Graphs…

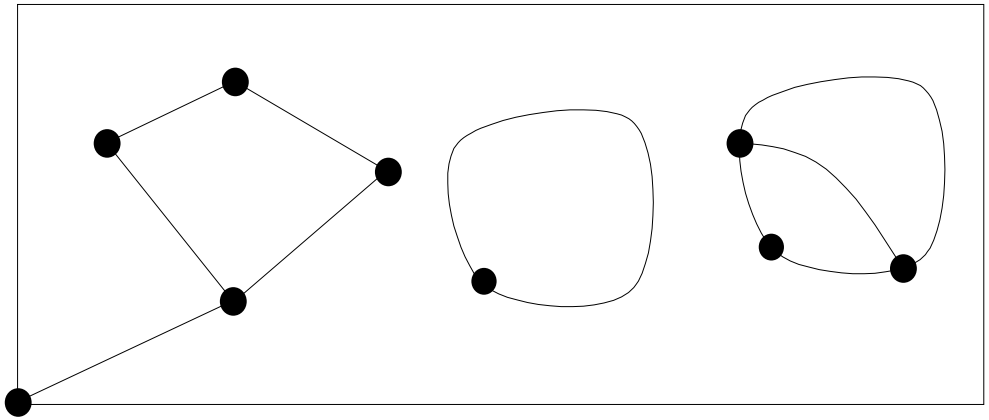
- Euler criterion (generalized version)…
  - for **c non-connected graphs**
  - In planar graphs: **p – e + n – c = 1**

p = 6

e = 11 → 6 – 11 + 9 – 3 = 1 (ok!)

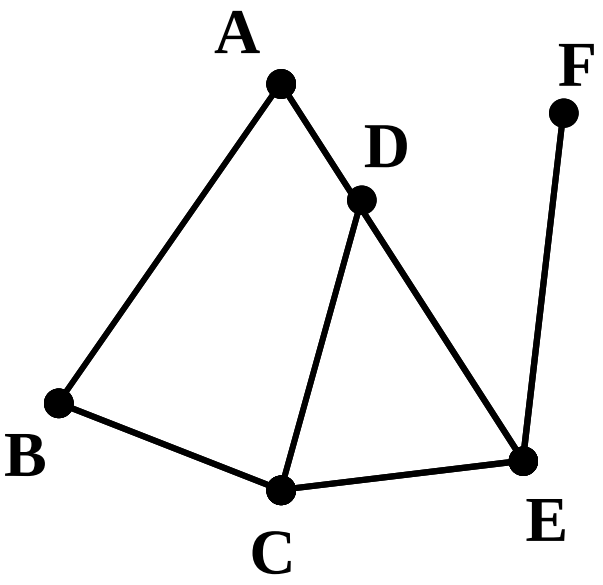
n = 9

c = 3



Representation of a Graph…

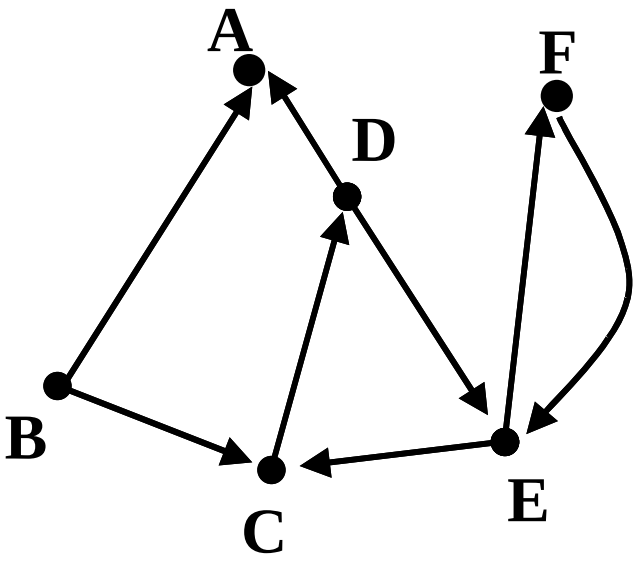
- Data structure: table
  - Simple graph (symmetric table)



	A	B	C	D	E	F
A	1	1	0	1	0	0
B	1	1	1	0	0	0
C	0	1	1	1	1	0
D	1	0	1	1	1	0
E	0	0	1	1	1	1
F	0	0	0	0	1	1

Representation of a Graph…

- Data structure: table
  - Directed graph

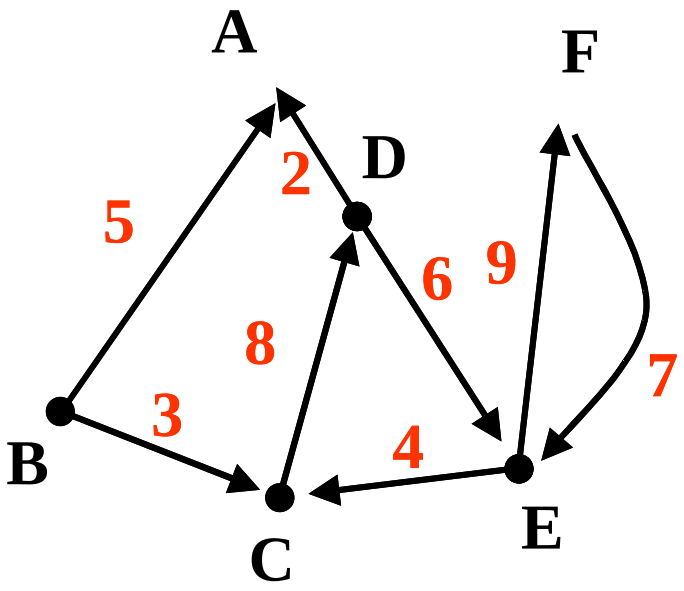


from / to

	A	B	C	D	E	F
A	1	0	0	0	0	0
B	1	1	1	0	0	0
C	0	0	1	1	0	0
D	1	0	0	1	1	0
E	0	0	1	0	1	1
F	0	0	0	0	1	1

Representation of a Graph…

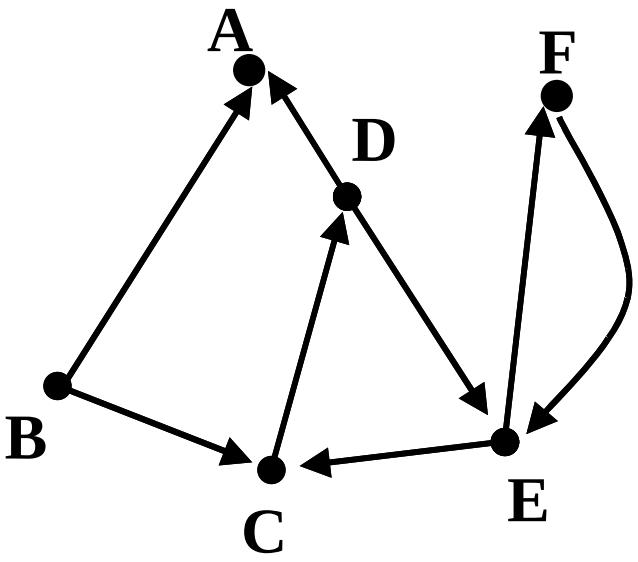
- Data structure: table
  - Directed and weighted graph

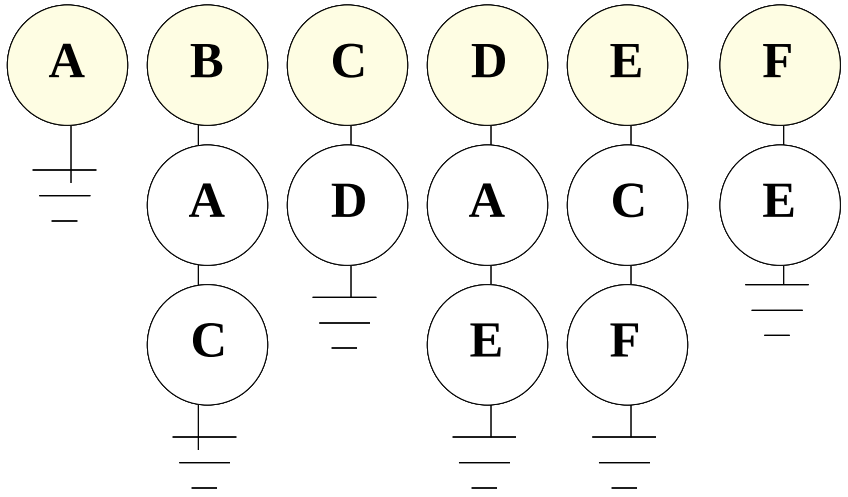


from \ to	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	5	0	3	∞	∞	∞
C	∞	∞	0	8	∞	∞
D	2	∞	∞	0	6	∞
E	∞	∞	4	∞	0	9
F	∞	∞	∞	∞	7	0

Representation of a Graph…

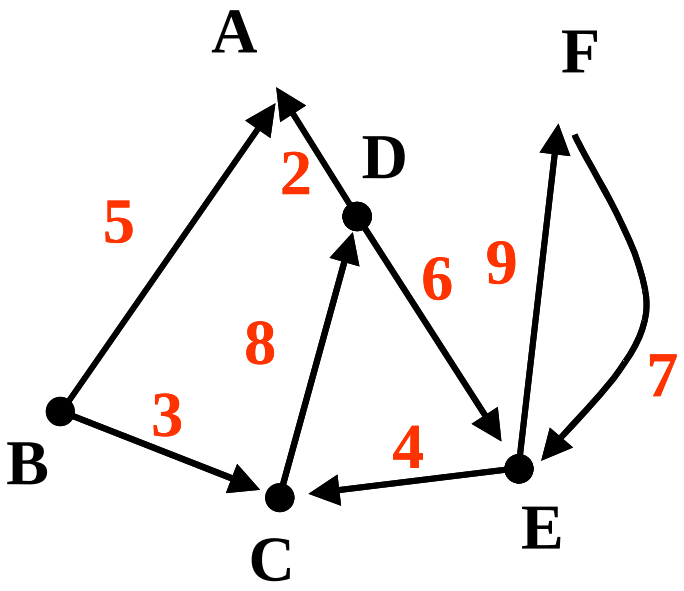
- Data structure: linked-list
  - Directed graph

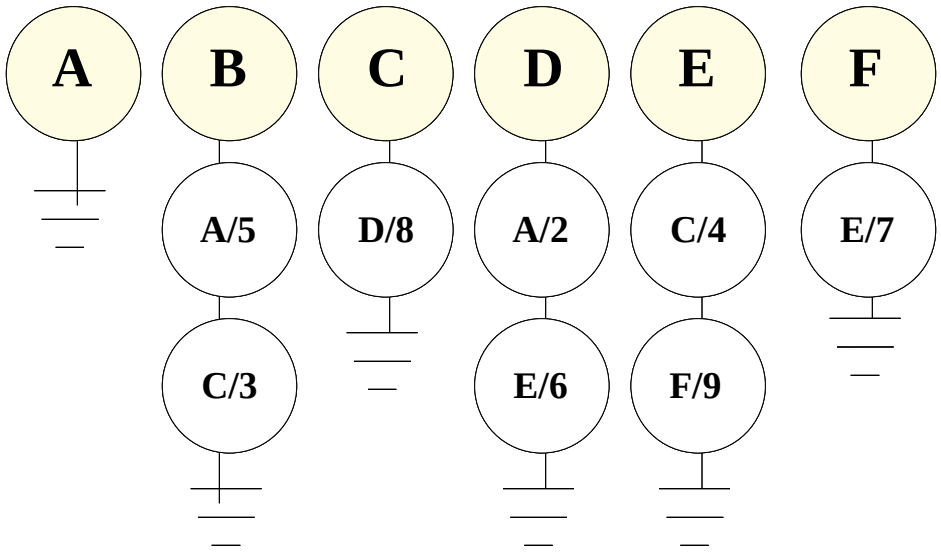




Representation of a Graph…

- Data structure: linked-list
  - Directed and weighted graph

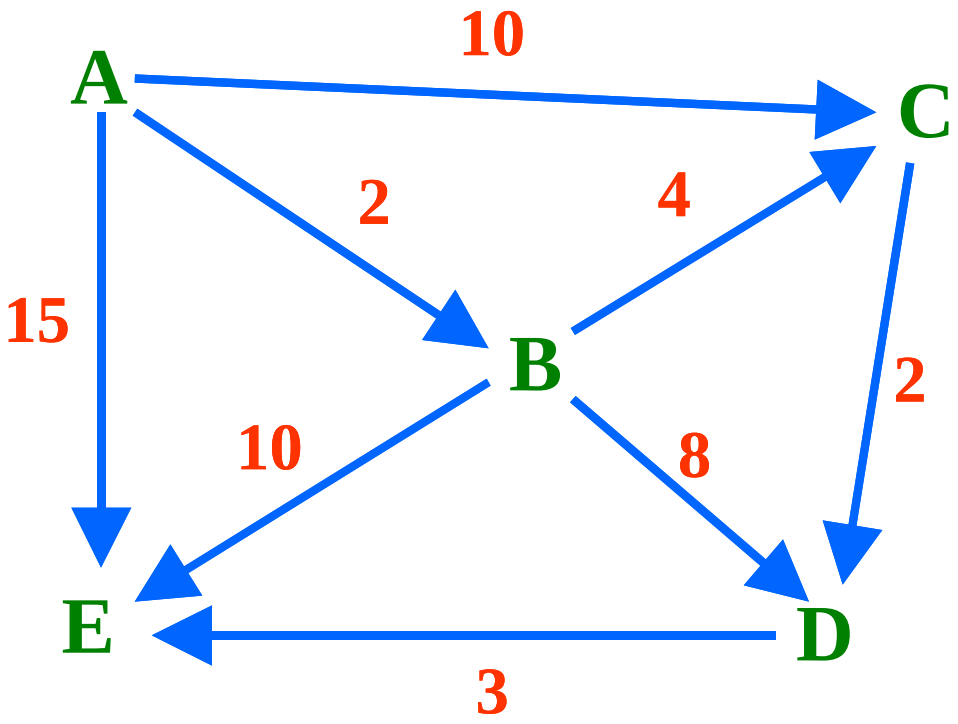




Shortest Path in a Network

- Dijkstra Algorithm
  - Finds the shortest path from A to all graph nodes

$N_o = A$

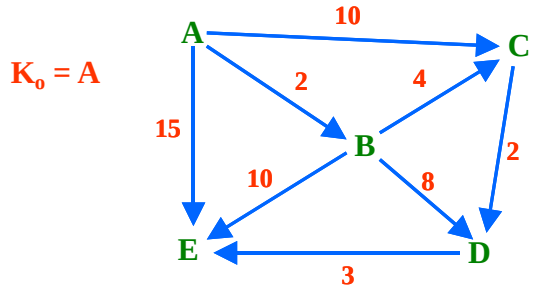


Shortest Path in a Network

- Dijkstra Algorithm

```
accumulated_travel_cost(Ur)
begin
   C[Ur] ← 0;
   for each U ∈ UM - {Ur} do C[U] ← ∞;
   UV ← {Ur};
   while UV ≠ 0 do
   begin
      for U ∈ UV : C[U] = min do
      begin
         UV ← UV  - {U};
         UN ← neighbors(U);
         for each Un ∈ UN do
         begin
            if accessible(Un) and C[Un] = ∞  then
            UV ← UV ∪ {Un};
            C[Un] ← min{C[Un], C[U]+travel_cost(U,Un)};
         end;
      end;
   end;
end;
```

Shortest Path in a Network



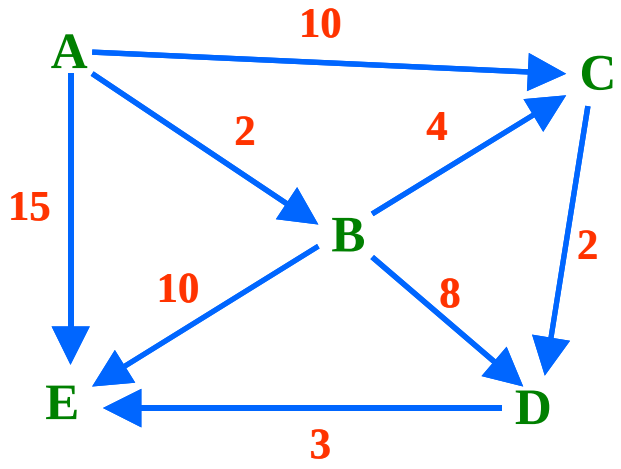
- Dijkstra Algorithm
  - Finds the shortest path from A to all graph nodes

Step	Visited VN	Next NVN	Curr CN	A	B	C	D	E
0	∅	A		0	∞	∞	∞	∞
1	A	BCE	A	0	2	10	∞	15
2	AB	CED	B	0	2	6	10	12
3	ABC	ED	C	0	2	6	8	12
4	ABCD	E	D	0	2	6	8	11
5	ABCDE	∅	E	0	2	6	8	11

Shortest Path in a Network

- Dijkstra Algorithm
  - Finds the shortest path from A to all graph nodes

$K_o = A$



Step	Visited	Next	Curr	A	B	C	D	E
0	∅	A		0	∞	∞	∞	∞
1	A	BCE	A	0	2	10	∞	15
2	AB	CED	B	0	2	6	10	12
3	ABC	ED	C	0	2	6	8	12
4	ABCD	E	D	0	2	6	8	11
5	ABCDE	∅	E	0	2	6	8	11

Find shortest path from A to E:
E (11-3=8) → D (8-2=6) → C (6-4=2) → B (2-2=0) → A.
Hence: A→B→C→D→E

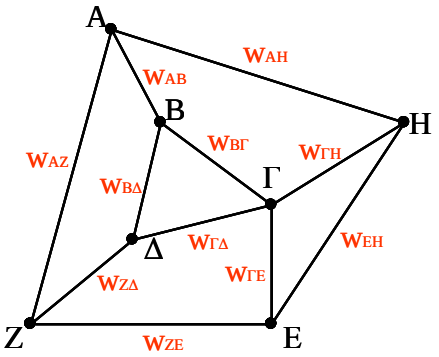
Shortest Path in a Network

- Dijkstra Algorithm
  - Complexity: $O(K^2)$
    - where K the number of nodes.
    - Very slow in big networks!

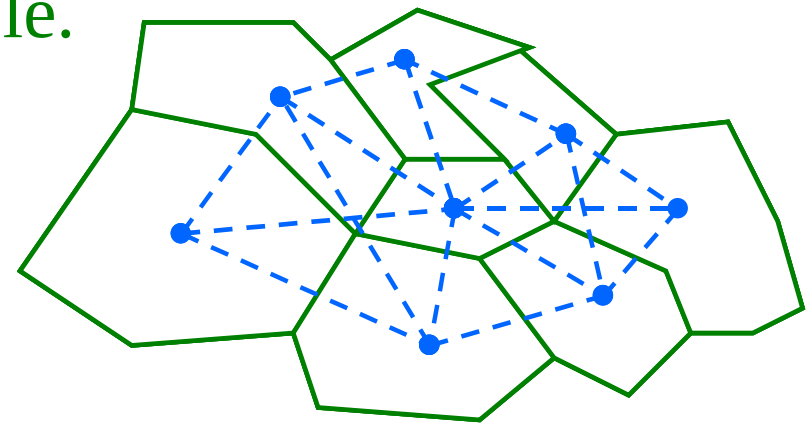
- Artificial Intelligence:
  - A* Algorithm (heuristics)

Graph problems

- Other problems solved using graphs…
  - TSP problem
    - Traveling salesman problem



  - Coloring the polygons of a thematic map using as less colors as possible.

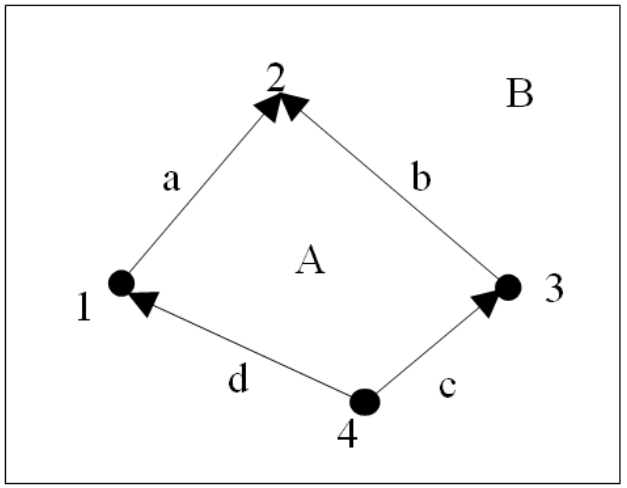


Topological Data Structures

- In GIS usually …
  - we build topological data structures to accelerate processing
  - Topological relationships can be extracted through processing…
    - however, it is time consuming and
    - this is why sometimes they are pre-calculated

Topological Data Structures

- A planar graph



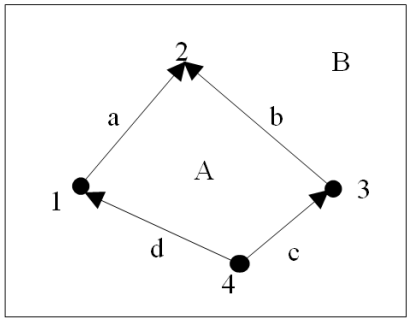
- The description of its topology (data structure)

edge	From node	To node	Left polygon	Right polygon
a	1	2	B	A
b	3	2	A	B
c	4	3	A	B
d	4	1	B	A

Topological Data Structures

- Some typical queries
  - Supported by the topological data structure

GENERAL QUERY	EXAMPLE
Find the neighbors of a polygon.	Which parcels meet the park?
Is there any hole in the polygon?	Which are the islands of this lake?
Are the two nodes connected?	Is there a road connecting the two towns?

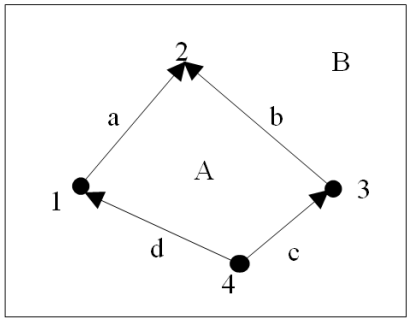


edge	From node	To node	Left polygon	Right polygon
a	1	2	B	A
b	3	2	A	B
c	4	3	A	B
d	4	1	B	A

Topological Data Structures

- Some typical queries
  - Supported by the topological data structure

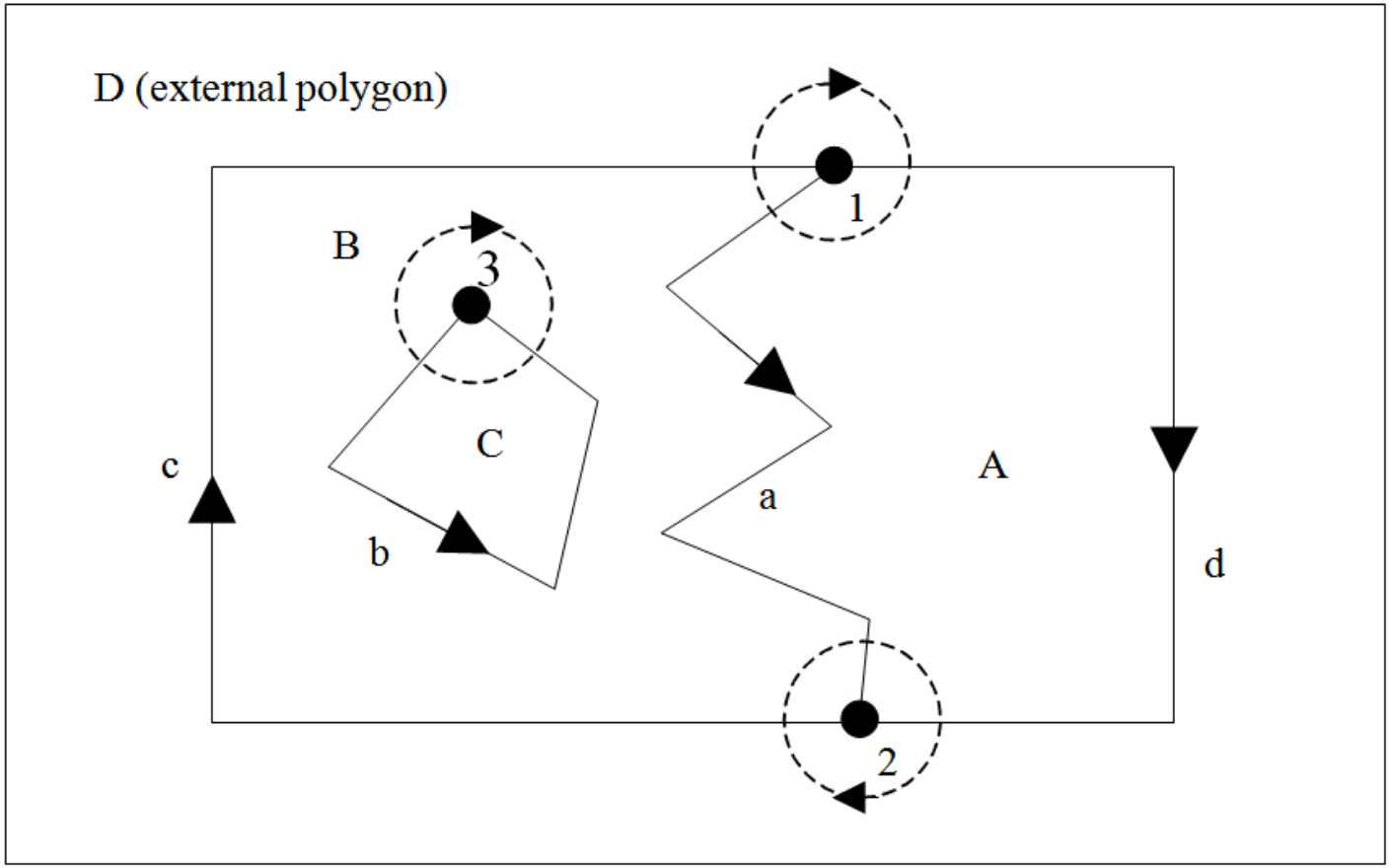
GENERAL QUERY	EXAMPLE
Which is the shortest path from node A to node B?	Which is the shortest route from town A to town B?
Which edges meet at this node?	Which roads end at this square?
Which is the intersection of two polygons A and B?	Which area has a annual average temperature higher than 10^{o}C and is a forest?



edge	From node	To node	Left polygon	Right polygon
a	1	2	B	A
b	3	2	A	B
c	4	3	A	B
d	4	1	B	A

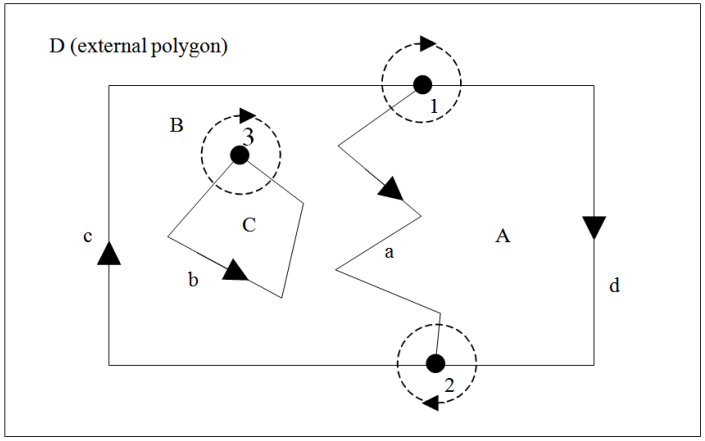
Topological Data Structures

- A example structure (in CARIS)



Topological Data Structures

- A example structure (in CARIS)

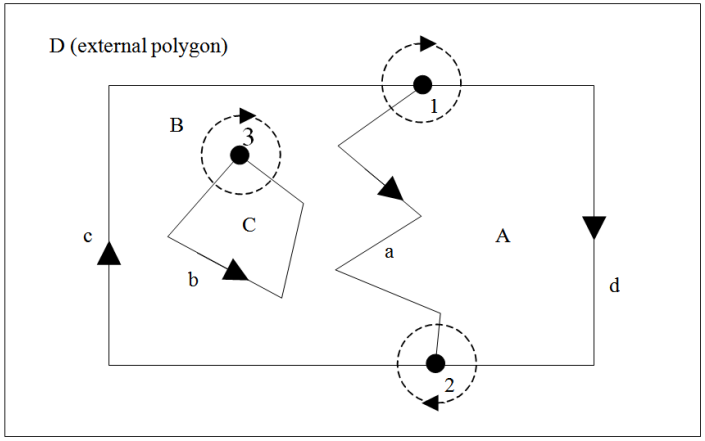


File of Edges:

edge	From node	To node	Next edge on start	Next edge on end	Left polygon	Right polygon
a	1	2	c	d	A	B
b	3	3	b	b	C	B
c	2	1	a	d	D	B
d	1	2	a	c	D	A

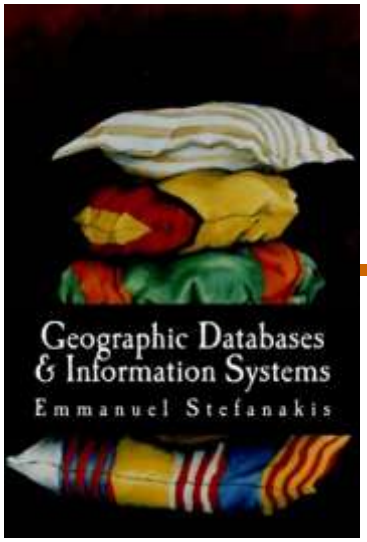
Topological Data Structures

- A example structure (in CARIS)



File of Polygons:

polygon	edge	Island (hole)	Surrounded by
A	a	--	D
B	c	C	D
C	b	--	B



Stefanakis, E., 2014. *Geographic Databases and Information Systems*. CreateSpace Independent Publ. [In English], pp.386.

Get a copy from Amazon

Chapter 8

Data Structures and Algorithms

Emmanuel Stefanakis

http://www2.unb.ca/~estef/