Using the appropriate data structure

Vera Sacristán

Discrete and Algorithmic Geometry
Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

**Depending on what we want to do with our data,
it may be convenient to store them in different ways**

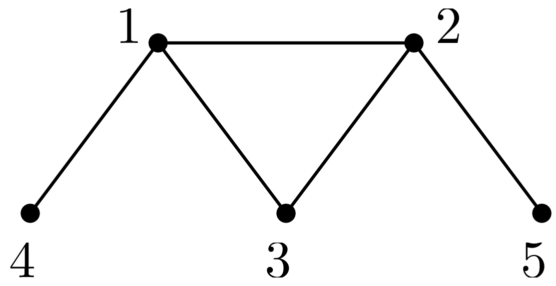
Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

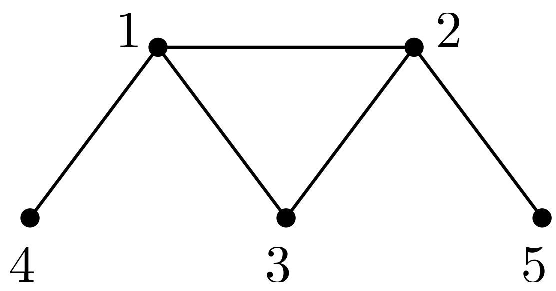
Consider the following graph G:



Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

Consider the following graph G:



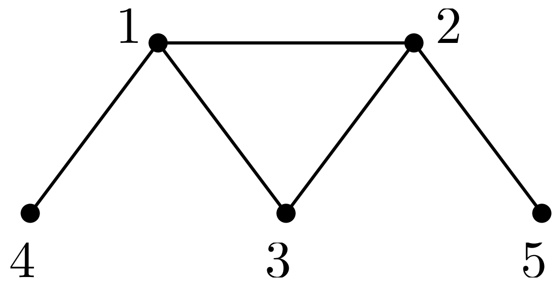
Incidence matrix

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

Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

Consider the following graph G:



Incidence matrix

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

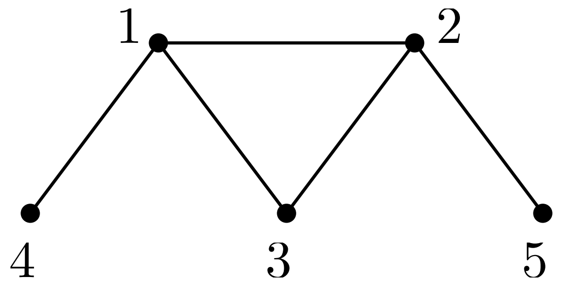
Adjacency list

$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

Depending on what we want to do with our data,
it may be convenient to store them in different ways

Example

Consider the following graph G:



Are v_i and v_j connected?

Incidence matrix

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

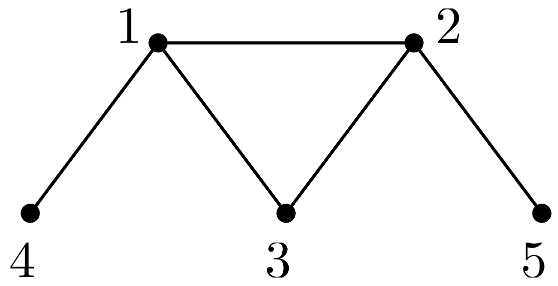
Adjacency list

$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

Depending on what we want to do with our data,
it may be convenient to store them in different ways

Example

Consider the following graph G:



Are v_i and v_j connected?

$O(1)$

$O(n)$

Incidence matrix

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

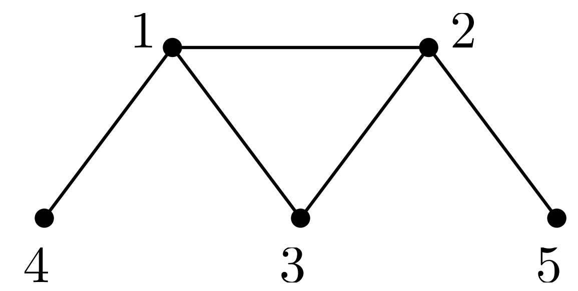
Adjacency list

$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

Consider the following graph G:



Are v_i and v_j connected?	What is degree(v_i)?
$O(1)$	
$O(n)$	

Incidence matrix

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

Adjacency list

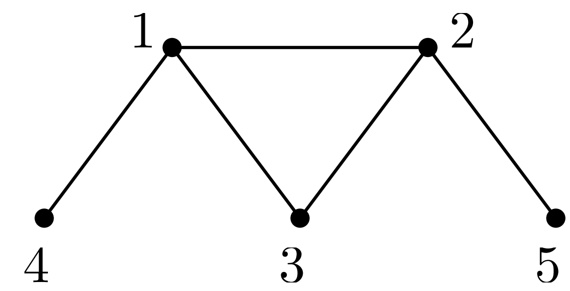
$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

USING THE APPROPRIATE DATA STRUCTURE

**Depending on what we want to do with our data,
it may be convenient to store them in different ways**

Example

Consider the following graph G:



Are v_i and v_j connected?	What is degree(v_i)?
$O(1)$	$O(n)$
$O(n)$	$O(degree)$

Incidence matrix

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

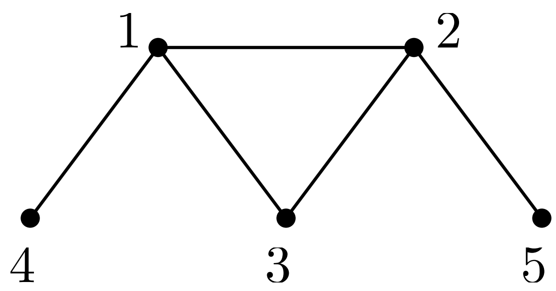
Adjacency list

$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

Depending on what we want to do with our data, it may be convenient to store them in different ways

Example

Consider the following graph G:



Are v_i and v_j connected?	What is degree(v_i)?	Storage space
$O(1)$	$O(n)$	
$O(n)$	$O(degree)$	

Incidence matrix

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

Adjacency list

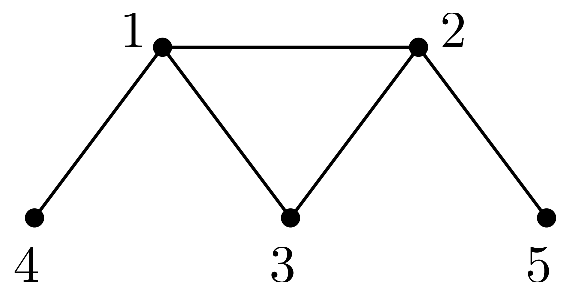
$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

USING THE APPROPRIATE DATA STRUCTURE

Depending on what we want to do with our data,
it may be convenient to store them in different ways

Example

Consider the following graph G:



Are v_i and v_j connected?	What is degree(v_i)?	Storage space
$O(1)$	$O(n)$	$O(n^2)$
$O(n)$	$O(degree)$	$O(edges)$

Incidence matrix

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

Adjacency list

$$((2,3,4),(1,3,5),(1,2),(1),(2))$$

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Any data structure representing S and allowing the following operations:

- query
- add
- delete

is called a dictionary

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Any data structure representing S and allowing the following operations:

- query
- add
- delete

is called a dictionary

If the universe $\mathcal{U}$ is totally sorted:

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Any data structure representing S and allowing the following operations:

- query
- add
- delete

is called a dictionary

If the universe $\mathcal{U}$ is totally sorted:

- locate
- insert
- delete

dictionary

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Any data structure representing S and allowing the following operations:

- query
- add
- delete

is called a dictionary

If the universe $\mathcal{U}$ is totally sorted:

- locate
- insert
- delete
- minimum

locate, insert, delete: dictionary

locate, insert, delete, minimum: priority queue

USING THE APPROPRIATE DATA STRUCTURE

Terminology

Let S be a set in a given universe $\mathcal{U}$.

Any data structure representing S and allowing the following operations:

- query
- add
- delete

is called a dictionary

If the universe $\mathcal{U}$ is totally sorted:

Operations			
• locate • insert • delete	dictionary	priority queue	augmented dictionary
• minimum		priority queue	augmented dictionary
• maximum • next • previous			augmented dictionary

Data structures to implement these operations

Data structures to implement these operations

Linked list

start $\longrightarrow$ x_1 $\longrightarrow$ x_2 $\longrightarrow$ $\ldots$ $\longrightarrow$ x_n $\longrightarrow$ end

Data structures to implement these operations

Linked list

$$\text{start} \longrightarrow x_1 \longrightarrow x_2 \longrightarrow \ldots \longrightarrow x_n \longrightarrow \text{end}$$

Doubly linked list

$$\text{start/end} \rightleftarrows x_1 \rightleftarrows x_2 \rightleftarrows \ldots \rightleftarrows x_n \rightleftarrows \text{start/end}$$

USING THE APPROPRIATE DATA STRUCTURE

Data structures to implement these operations

Linked list

$$\text{start} \longrightarrow x_1 \longrightarrow x_2 \longrightarrow \dots \longrightarrow x_n \longrightarrow \text{end}$$

Doubly linked list

$$\text{start/end} \rightleftarrows x_1 \rightleftarrows x_2 \rightleftarrows \dots \rightleftarrows x_n \rightleftarrows \text{start/end}$$

How are the operations done?

- Locate: explore the list in $O(i)$ time
- Insert/ delete: Once located, change pointers in $O(1)$ time
- Min/max: $O(1)$ time
- Next/previous: $O(1)$ time

Data structures to implement these operations

Linked list

$$\text{start} \longrightarrow x_1 \longrightarrow x_2 \longrightarrow \dots \longrightarrow x_n \longrightarrow \text{end}$$

Doubly linked list

$$\text{start/end} \rightleftarrows x_1 \rightleftarrows x_2 \rightleftarrows \dots \rightleftarrows x_n \rightleftarrows \text{start/end}$$

How are the operations done?

- Locate: explore the list in $O(i)$ time
- Insert/ delete: Once located, change pointers in $O(1)$ time
- Min/max: $O(1)$ time
- Next/previous: $O(1)$ time

Two particular lists are:

Stack: LIFO list (insert/delete only at the end)

Queue: LILO list (insert at the end, delete at the beginning)

Data structures to implement these operations

Binary-search trees

Balanced binary trees

Data structures to implement these operations

Binary-search trees

Balanced binary trees

↑ acyclic connected graphs

Data structures to implement these operations

Binary-search trees

Balanced binary trees

acyclic connected graphs

each node has at most two children

Data structures to implement these operations

Binary-search trees

Balanced binary trees

- trees: acyclic connected graphs
- binary: each node has at most two children
- Balanced: all branches have approximately the same length

Data structures to implement these operations

Binary-search trees

Balanced binary trees

acyclic connected graphs

each node has at most two children

all branches have approximately the same length

Example

$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in $\mathcal{U} = (\mathbb{R} \leq)$

Data structures to implement these operations

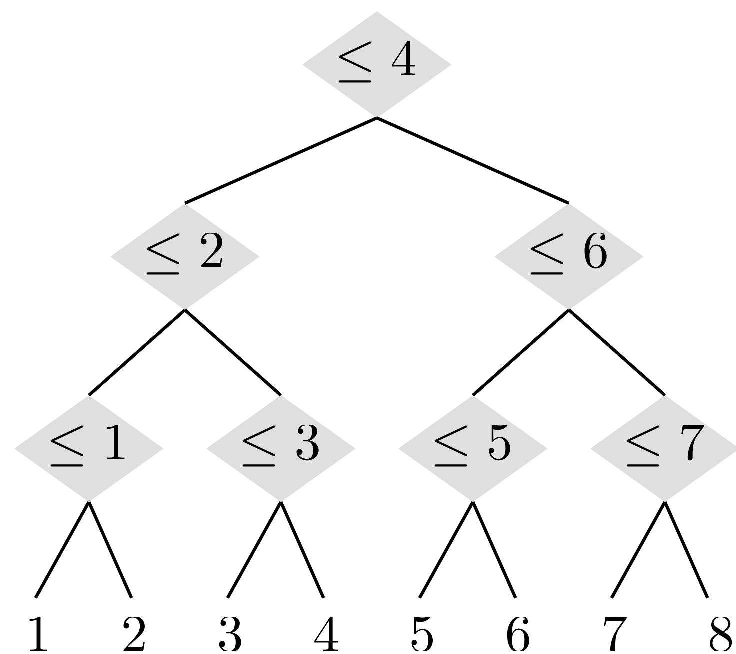
Binary-search trees

Balanced binary trees

- acyclic connected graphs
- each node has at most two children
- all branches have approximately the same length

Example

$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in $\mathcal{U} = (\mathbb{R} \leq)$



Data structures to implement these operations

Binary-search trees

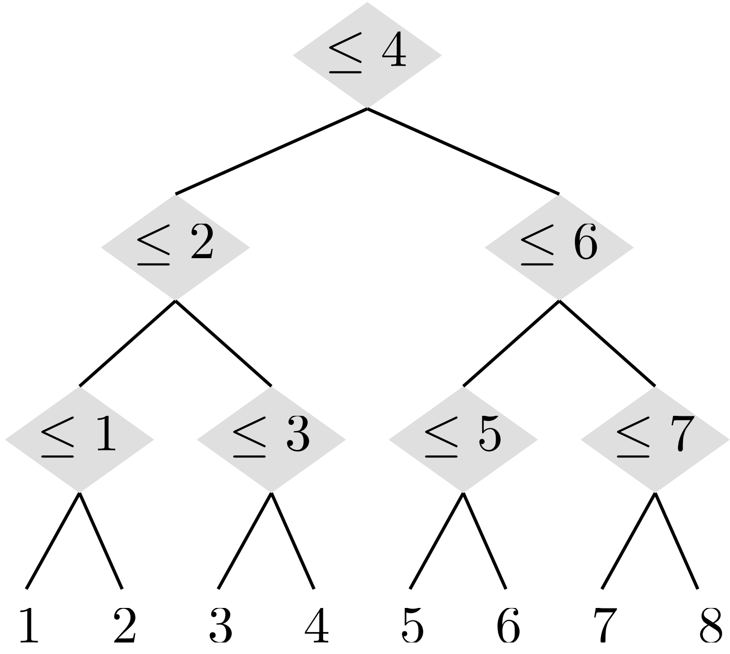
Balanced binary trees

- acyclic connected graphs (trees)
- each node has at most two children (binary)
- all branches have approximately the same length (balanced)

Example

$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in $\mathcal{U} = (\mathbb{R} \leq)$

- Locate(3)
- Locate(3.5)
- Min(S)
- Insert(3.5)
- Delete(3)



USING THE APPROPRIATE DATA STRUCTURE

Data structures to implement these operations

Binary-search trees

Balanced binary trees

- trees: acyclic connected graphs
- binary: each node has at most two children
- balanced: all branches have approximately the same length

Theorem

Given a set S of n elements ot a totally ordered univers $\mathcal{U}$, it is possible to build a balanced binary search tree representing S using $O(n)$ space and $O(n \log n)$ time.

Then:

- memberQ
- locate
- min/max
- insert/delete

run in $O(\log n)$ time

- next/previous runs in $O(1)$ time

INITIAL READING

F. P.Preparata and M. I Shamos
Computational Geometry: An Introduction
Springer-Verlag, 1985, pp. 6-15 and 26-35.

J-D. Boissonnat and M. Yvinec
Algorithmic Geometry
Cambridge University Press, 1997, pp. 3-31.

INITIAL READING

F. P.Preparata and M. I Shamos
Computational Geometry: An Introduction
Springer-Verlag, 1985, pp. 6-15 and 26-35.

J-D. Boissonnat and M. Yvinec
Algorithmic Geometry
Cambridge University Press, 1997, pp. 3-31.

FURTHER READING

T. H. Cormen, C. E. Leiserson, and R. L. Rivest
Introduction to Algorithms
The MIT Press - McGraw Hill Book Company, 1990 (3rd ed. 2009)

A. V. Aho, J. E. Hopcroft, and J. D. Ullman
Data structures and algorithms
Addison-Wesley, 1983