Sets, Bags, Graphs

Anton Gerdelan <gerdela@scss.tcd.ie>

Sets and Bags (ADTs)

- Mirrors **finite set theory** from mathematics
    - usually **mutable sets** - allow deletion/insertion in set
- A set is a collection of unique items. Position in set is not important, except for display.
- Rather than return an element from a set

Set Operations

- **insert**(item) - fail if item is already in set
- **delete**(item)
- **test_for**(item) - return true if item is in set
- **union**() - combine 2 sets, return new set (OR)
- **intersection**() - returns a new set (AND)

Set Operations

- If set C contains { 6, 12, 9, 1 }
  and set D contains { 3, 6, 1, 5 }
- then set E = C union D
  contains {1, 3, 5, 6, 9, 12 }   - no duplicates
- and set F = C intersection D contains { 1, 6 }
- A **bag** is a set that can contain duplicates
  - B = { 3, 1, 22, 22, 3 } or
  - B = { 3(2), 1(1), 22(2) }

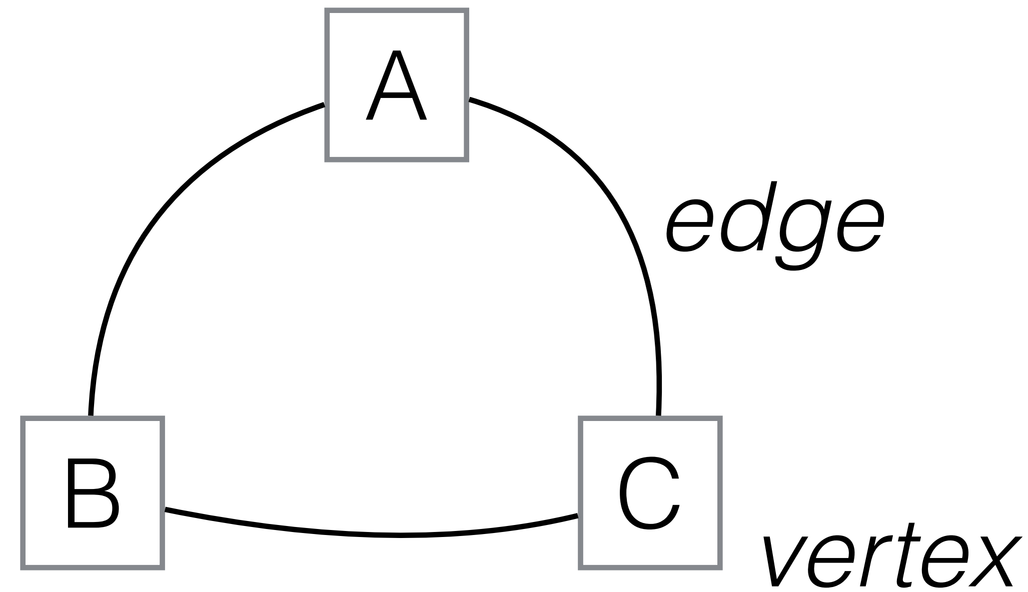
Set/bag Implementation

- **Arrays** or **linked lists** or…
- **bit-vectors** (sets only) - but very fast
- e.g. 32-bit integer can hold values 0 to 31 (or e.g. months of year)

  0000 0000 0000 0000 0000 0000 0000 0001
  this set holds { 0 }
  (note that in binary/hexedit this order is reversed)
- Union is just `E = C | D`
- Intersection is just `E =C & D`
- To insert an item, set its bit: `E = E | (1<<n)`

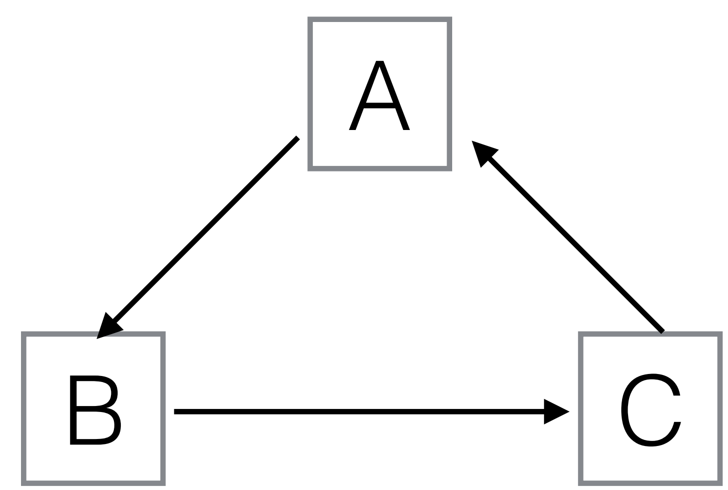
Graph ADT

- set of **vertices** (nodes)
- set of **edges** (like branches)
- similar to tree but
  - can contain **cycles**
- travel in any direction along edges
  - except in **directed graph**



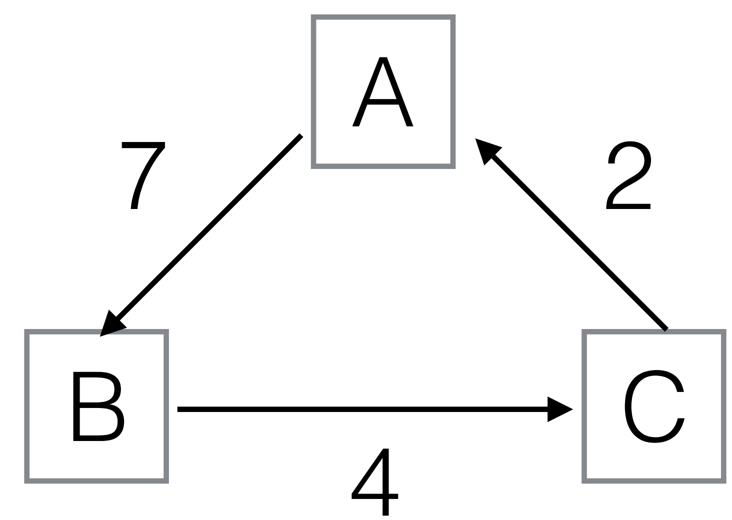
$v = \{A, B, C\}$

$e = \{(B,C), (C,A), (A,B)\}$



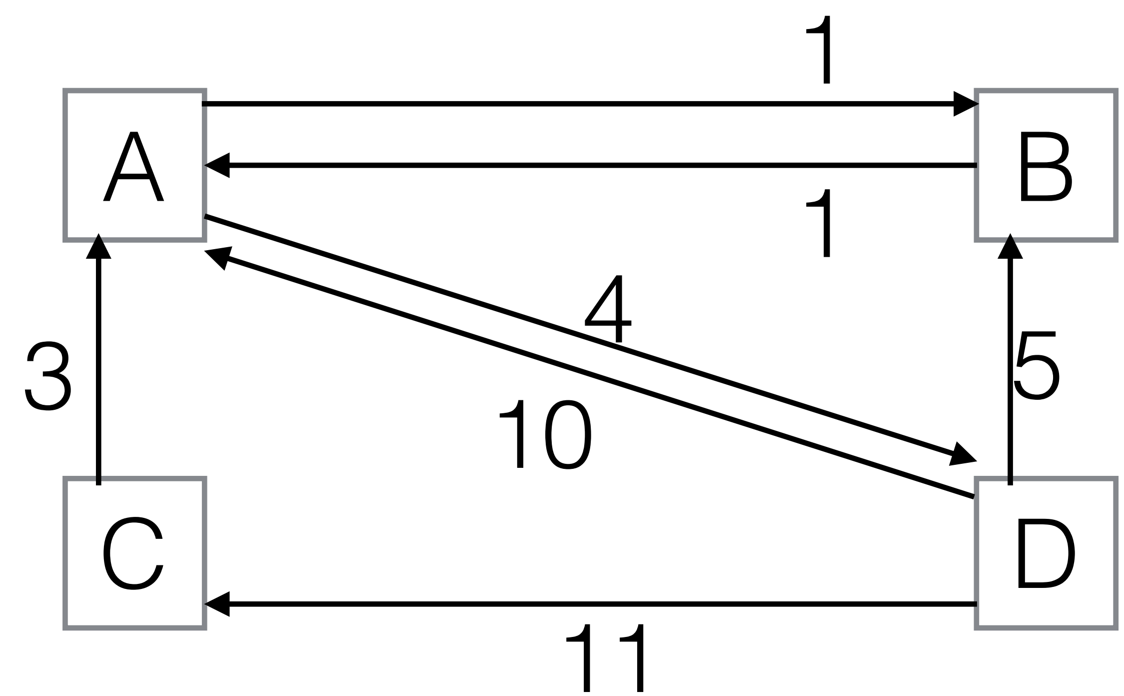
Graphs

- edges can have **weights**
  - represent cost or quantity of link
  - (or labels / words)
- **Q.** *what type of problems can we model with a graph?*
  - *what do the weights represent?*



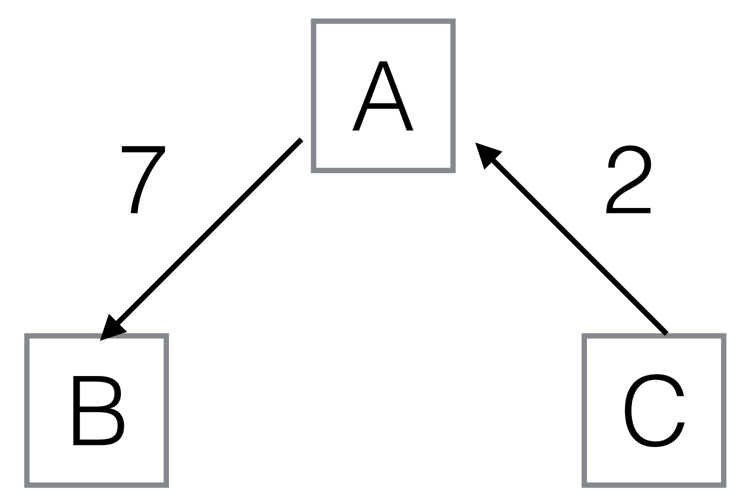
Paths

- Two vertices are adjacent if an edge links them directly
- A **path** between 2 vertices moves along a sequence of edges
  - A-B-A-D-C is a path
- Path **length** is the sum of weights on the path
  - A-B-A-D-C has length 17
- A **cycle** is a path with length > 0 from a vertex **to itself**
  - A-D-C-A is a cycle



Paths

- A **connected graph** has a path from every vertex to every other vertex
  - vertices don't need to be directly adjacent
- An **acyclic** graph has no cycles. **Cyclic** has 1+



Some Graph Operations

- **insert_vertex**() // insert new node into set of nodes
- **insert_edge**() // insert new edge into set of edges
- bool **is_adjacent**(vertex from, vertex to) // true if an edge from a to b exists
- int **weight**(vertex a, vertex b) // return weight of edge between a and b
- int **num_nodes**()
- int **num_edges**()
- **remove_node**() // remove nodes and any isolated edges
- **remove_edge**() // without removing nodes
- **edit_edge**() // alter weight or direction

Other Graph Operations

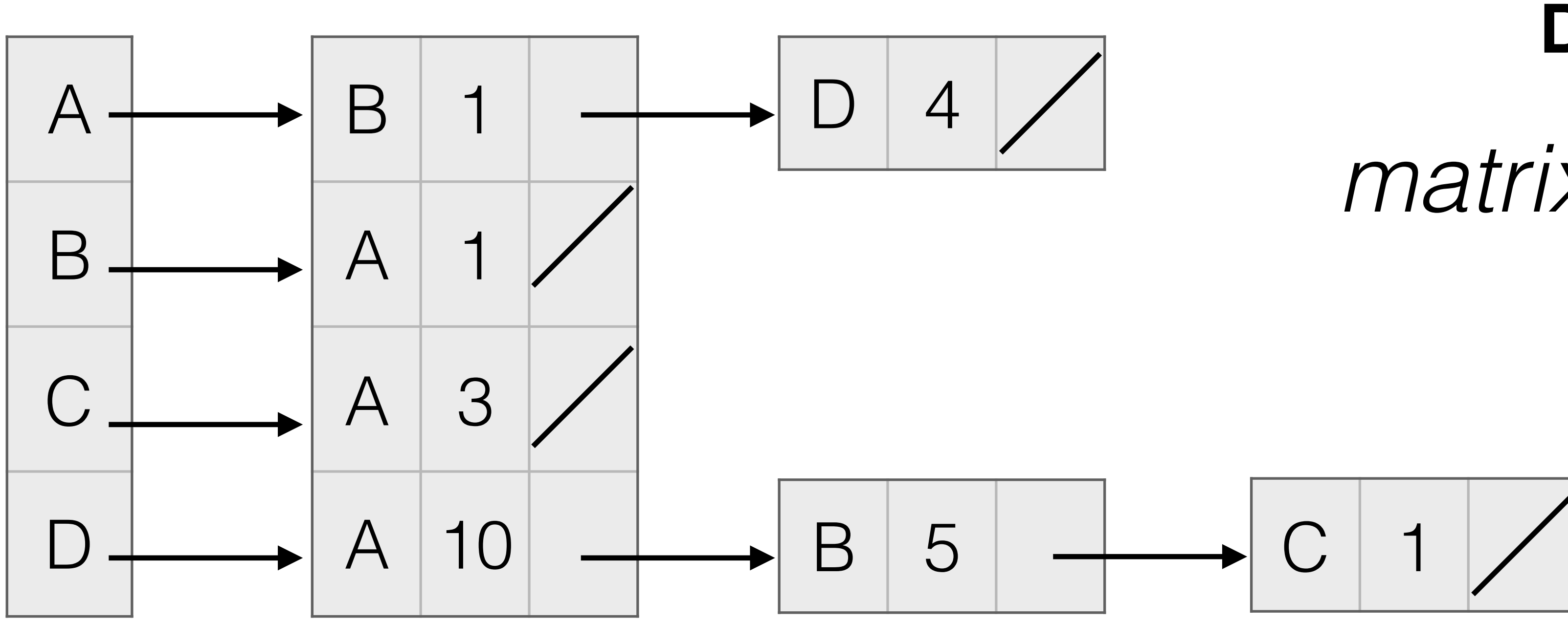
- find_path(vertex a, vertex b)
- find_shortest_path(vertex a, vertex b)
- …

Graph Implementation

- Two sets - *could* use sets to implement graphs
  - **G** = { Nodes, Edges }
  - **Nodes** = { A, C, D, B }
  - **Edges** = { (A, B, 1), (B, A, 1), (D, B, 5), (C, A, 3), (A, D, 4), (D, C, 11), (D, A, 10) }

Graph Implementation

- Usually more convenient to represent with matrices (sparse matrix - zero means "no edge")
- Or linked lists - an **adjacency list**



end node

start node

	A	**B**	**C**	**D**
A	-1	1	0	4
B	1	-1	0	0
C	3	0	-1	0
D	10	5	11	-1

matrix of edge weights