Set 05: Abstract Data Types and Data Structures

CS240: Data Structures and Data Management

Jérémy Barbay

Outline

What's an Abstract Data Type? A Data Structure?

- Data Type
- Principle of Abstraction
- Abstract Data Type (ADT)
- Data Structure (DS)

Stack ADT

- ADT consisting of a container of objects
- Insertion/Deletion performed LIFO
- Required operations – `push( o )`, `pop()`, `isEmpty()`
- Helpful operations – `top()`, `size()`
- Applications:

Implementation in an Array

- Two instance variables:
  - Array, A, of size N
  - Integer, *size*
- Maximum size of stack is N
- Vector
- Running Times:
- Space Usage (after n items inserted):

Implementation in a Singular Linked List

- Use `Node` class:
  - Data fields – elem, next
  - Accessor/Mutators – getElem/setElem, getNext/setNext
- Two instance variables:
  - Reference to *top* Node of stack
  - Integer, *size*
- No maximum size of stack
- Running Times :
- Space Usage (after n items inserted):

Queue (FIFO) ADT

- ADT consisting of a container of objects
- Insertion/Deletion performed FIFO
- Required operations – `enqueue( o )`, `dequeue()`, `isEmpty()`
- Helpful operations – `front()`, `size()`
- Applications:

Implementation in an Array

- Four instance variables:
  - Array, A, of size N
  - Integers, *size*, *front*, *rear*
- Maximum size of queue is N
- Circular array
- Running Times:
- Space Usage (after n items inserted):

Implementation in a Singular Linked List

- Use `Node` class:
- Three instance variables:
  - Reference to *front*, *back* Nodes of queue
  - Integer, *size*
- No maximum size of queue
- Running Times :
- Space Usage (after n items inserted):

Outline

Simple Abstract Data Types
  Stack ADT
  Queue (FIFO) ADT

Graphs
  A family of ADTs
  Several Data Structures
  Algorithms for Search and Traversal

Definitions

- A graph G is a pair (V, E) such that
  - V is the set of vertices $\{v_1, v_2, \dots, v_n\}$
  - E is the set of edges $\{e_1, e_2, \dots, e_m\}$
- Undirected Graph

$$E = \{\{v_i, v_j\} : v_i \neq v_j \text{ and } v_i, v_j \in V\}$$

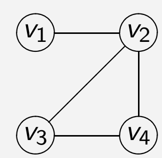
- Directed Graph

$$E = \{(v_i, v_j) : v_i, v_j \in V\}$$

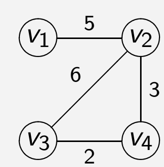
- Edges can also have a weight (cost) associated with it

Examples

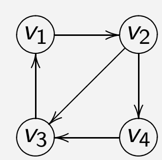
- Undirected Graph



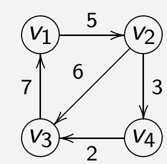
- Weighted Graph



- Directed Graph



- Directed Graph



Terminology

- **Simple Graph** – Finite, undirected graph with no self-loops or multi-edges
- **Dense Graph** – $m \in \Omega(n^2)$
- **Sparse Graph** – $m \in O(n)$
- **Path** (simple) – sequence of incident edges without repetition from u to v
- **Cycle** – path from u to u
- **non directed graph is connected** – $\forall u, v,\ \exists \text{path}(u, v)$
- **directed graph is connected** – $\exists u, \forall v,\ \exists \text{path}(u, v)$
- **directed graph is strongly connected** – $\forall u, v,\ \exists \text{path}(u, v)$

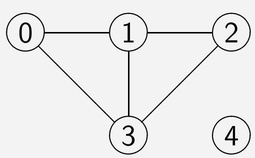
A family of Abstract Data Types

The most common operators are:

- Query Operators
  - `numVertices()`, `numEdges()`, `degree(v)`, `areAdjacent(` v_1`,` v_2 `)`,...
- Update Operators
  - `insertEdge(e)`, `removeEdge(e)`, `insertVertex(v)`, `removeVertex(v)`,...
- Iterators
  - `vertices()`, `edges()`, `adjacentVertices(v)`, `incidentEdges(v)`, ...

Data Structures for the Graph ADTs

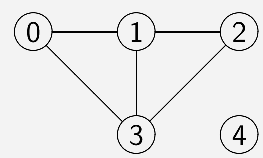
- Number the vertices $0, 1, \ldots, n-1$
- We will look at two possible data structures:
  1. Adjacency List
  2. Adjacency Matrix



- Ideas extend to directed and weighted graphs (**exercise**)
- **Note**: when analyzing iterator operators, we will determine the time to visit **all** the vertices or edges.

Adjacency List

- Maintain an unsorted list of neighbors, for each vertex in the array

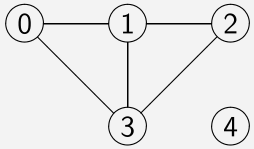


0 : 1 3
1 : 2 0 3
2 : 1 3
3 : 0 2 1
4 :

- Worst-case Runtimes:
  - `areAdjacent( v1, v2 )` –
  - `adjacentVertices(v)` –
  - `insertEdge(e)` –
  - `removeVertex(v)` –
- Space Usage –

Adjacency List (variant)

- Maintain a sorted list of neighbors, for each vertex in the array

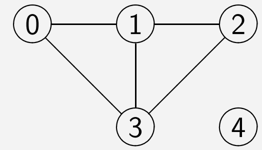


0 :	1	3	
1 :	(0)	2	3
2 :	(1)	3	
3 :	(0)	(1)	(2)
4 :			

- Worst-case Runtimes:
  - `areAdjacent( v1, v2 )` –
  - `adjacentVertices(v)` –
  - `insertEdge(e)` –
  - `removeVertex(v)` –
- Space Usage –

Adjacency Matrix

- Using a $n \times n$ matrix
- $A_{i,j} = 1$ if there is an edge



	0	1	2	3	4
0	–	1		1	
1	(1)	–	1	1	
2		(1)	–	1	
3	(1)	(1)	(1)	–	
4					–

- Worst-case Runtimes:
  - `areAdjacent( v1, v2 )` –
  - `adjacentVertices(v)` –
  - `insertEdge(e)` –
  - `removeVertex(v)` –
- Space Usage –

Many Other Implementations

- Many other implementations, but much more sophisticated, based on different trade-offs, and supporting more operators.
- An example: *static* succinct encoding of binary relations.

- Worst-case Runtimes:
  - `areAdjacent( v1, v2 )` – $O(\lg \lg n)$
  - `adjacentVertices(v)` – $O(1)$
  - `degree(v)` – $O(1)$
  - `insertEdge(e)`, `insertVertex(v)`, `removeVertex(v)`, `removeEdge(e)` – not supported
- Space Usage – $O(m \lg n)$ bits, i.e. $O(m)$ words on most machines

Some Algorithms on Graphs

There are many algorithms on graphs:

- Connex Component Search;
- Cycle search/detection;
- Flux optimisation;
- Shortest path:
  - between a pair of nodes,
  - between a node and all others.

We study here two of the simplest:

- Depth-First Search (and Traversal)
- Breadth-First Search (and Traversal)

Depth-First Search

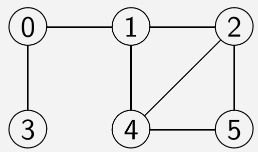
"Narrow" traversal of vertices

From a given start vertex v:

DFS-Graph(v)

```
Mark v
VISIT( v )
for each vertex, w, adjacent to v do
   if w is unmarked then
      DFS-GRAPH(w)
   end if
end for
```

Example:



What is the worst-case running time of the entire algorithm using adjacency lists?

Breadth-First Search

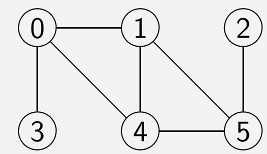
"Broad" traversal of vertices:

From a given start vertex v:

BFS-Graph(v)

```
if v has not been visited yet then
   VISIT(v)
   for w ← 1...n do
      BFS-GRAPH(w)
   end for
end if
```

Example:



What is the worst-case running time of the entire algorithm, when using adjacency lists?

Summary

Topic	Concept(s)
What's an ADT?	
Stacks (LIFO)	
Queue (FIFO)	
Graphs	
Adjacency list DS	
Adjacency matrix DS	
Algorithms on graphs	

Summary

Topic	Concept(s)
What's an ADT?	Abstract the "What" of the "How".
Stacks (LIFO)	Pile of plates
Queue (FIFO)	People waiting in line
Graphs	(non)directed, (un)weighted
Adjacency list DS	$O(m)$, good for sparse graphs.
Adjacency matrix DS	$O(1)$ or $O(n)$, good for dense graphs.
Algorithms on graphs	Depth First and Breadth First Traversal.