Algorithms & Data Structures

Outline

Lecture I

1. Motivation
2. Sorting algorithms
3. Linear data structures

Lecture II

5. Nonlinear data structures
6. Abstract data types
7. Dijkstra’s algorithm
8. Summary

Motivation

- Algorithms
- Data Structures

1. **Everything** running on your computer is an **algorithm**
2. Analysing them is paramount to **writing**, **maintaining** and **improving** them
3. Several **tools** exist to help achieve this

Motivation

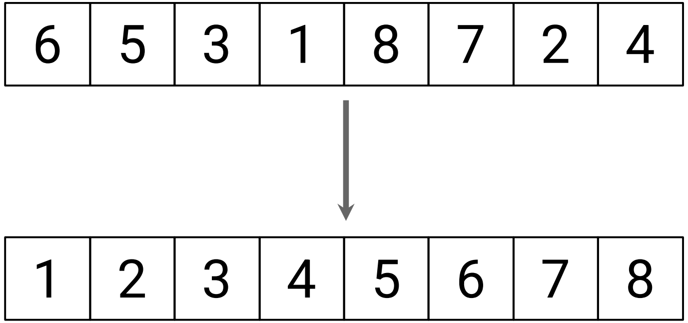
- Algorithms
- Data Structures

1. Data Structures define how data is **stored** in **RAM**
2. Many variations, each with **advantages** and **disadvantages**
3. Strongly coupled to algorithmic **complexity**

2. Sorting algorithms

Sorting

- Suppose we have some **unsorted list**
- We want to make it **sorted**



Insertion sort

- In pseudocode:

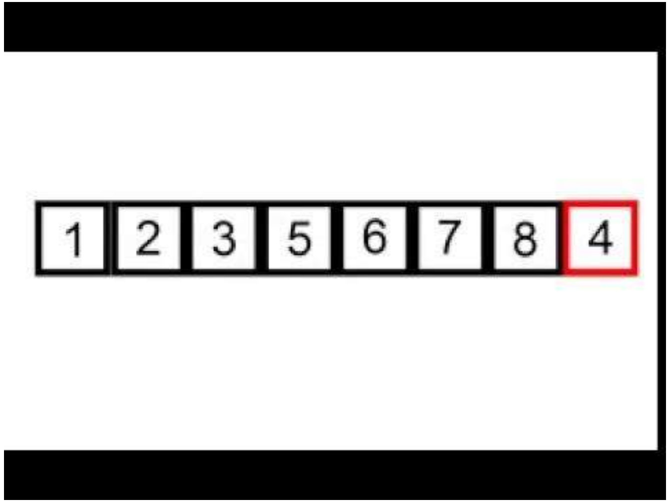
```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

- “Naive” sorting algorithm
- One-by-one, take each element and move it
- When all elements have been moved, list is sorted!

***N* steps**

***N* steps**

Total: ***N*² steps**



By Swfung8 (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Bubble sort

- Traverse the list, taking **pairs** of elements
- **Swap** if order incorrect
- **Repeat** N times
- Now it's **sorted**!

***N* steps**

***N* steps**

By Swfung8 (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Total: ***N^2* steps**

Intermezzo: Divide and conquer

- Generic algorithm **strategy**
  - **Divide** the problem into smaller parts
  - **Solve** (conquer) the problem for each part
  - **Recombine** the parts
- Straightforward to **parallelise**
- Closely related to **map-reduce**
- Has been advocated by Caesar, Machiavelli, Napoleon...

Merge sort

- Much **smarter** sort
  - **Split** the dataset into chunks
  - **Sort** each chunk
  - **Merge** the chunks back together
- Example of **divide-and-conquer**
- Splitting & sorting takes **$\log_2(N)$ steps**
- Merging takes ***N* steps**

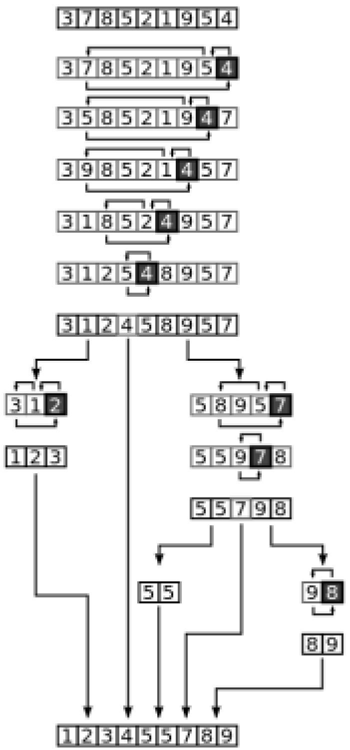
By Swfung8 (Own work) [CC BY-SA 3.0],
via Wikimedia Commons

Total: ***N* log(*N*) steps**

One more sorting example: Quicksort

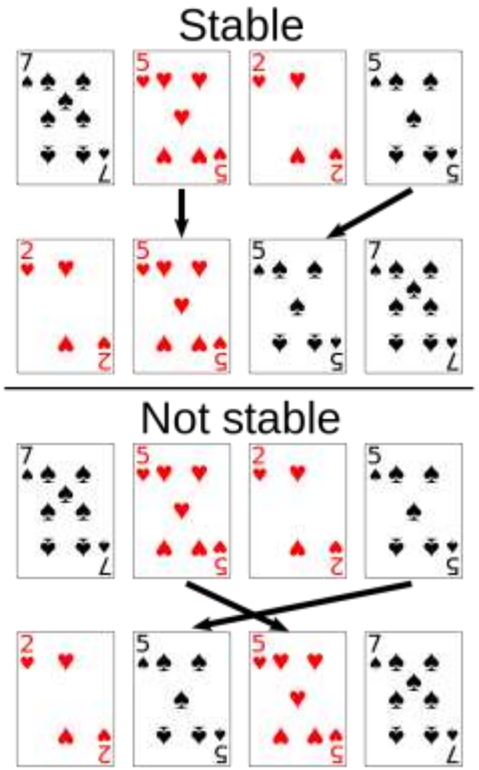
- Pick an element, called **pivot**
- **Partitioning**: reorder the array so that the pivot is in the correct place
- **Recursively** apply the above steps to the sub-arrays on either side of the pivot

- **Randomised-quicksort**: select the pivot **randomly**



Stable sorting

- A sorting algorithm is **stable** iff it **conserves the order** of equal elements



Comparison of algorithms

Algorithm	**Stable?**	**Complexity**
Insertion sort	✓	$O(N^2)$
Bubble sort	✓	$O(N^2)$
Merge sort	✓	$O(N \log(N))$
Quicksort	✗	?

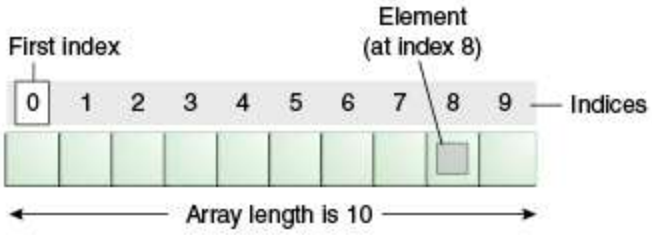
4. Linear data structures

Memory

Arrays

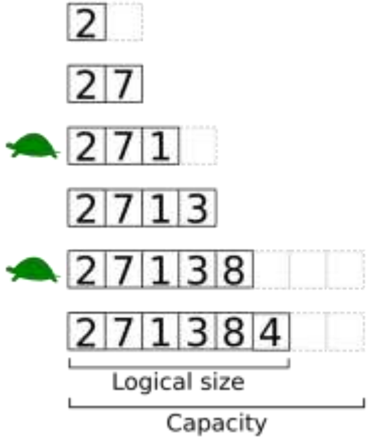
- **Linear**, **contiguous** list of data
- Accessible by **index**
- **Fixed-size**
  - $N * d$
- **Supported** by all major systems

- Back-insert/remove: **O(1)**
- Random insert/remove: **O(*N*)**
- Index-lookup: **O(1)**
- Lookup: **O(*N*)**



Dynamic arrays

- **Linear**, **contiguous** list of data
- Accessible by **index**
- **Resizable**

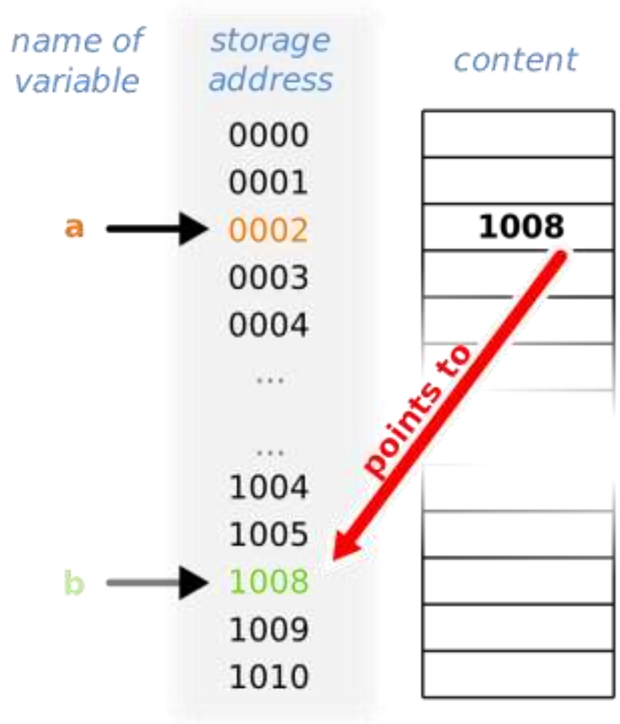


- Back-insert/remove: **O(1)***
- Random insert/remove: **O(*N*)**
- Index-lookup: **O(1)**
- Lookup: **O(*N*)**

*Amortised.

C++: **`std::vector`**
Python: **`list`**
C#: **`System.Collections.ArrayList`**
Java: **`java.util.ArrayList`**

Pointers

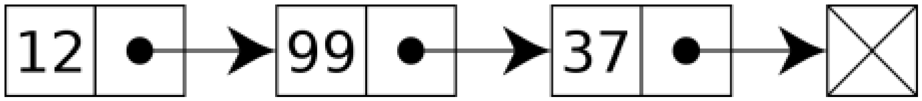


By Sven (Own work) [CC BY-SA 3.0],
via Wikimedia Commons

Linked list

- **Linear, contiguous** list of data
- Accessible by **iteration**
- **Resizable**

- Back-insert/remove: **O(1)**
- Random insert/remove: **O(*N*)**
- Index-lookup: **O(*N*)**
- Lookup: **O(*N*)**

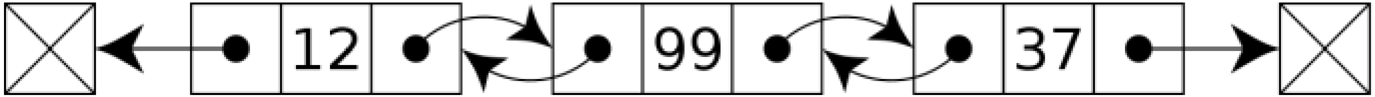


C++: `std::forward_list`

Doubly Linked list

- Pointers **both ways**
- Uses **more memory**, but allows **iteration** both ways

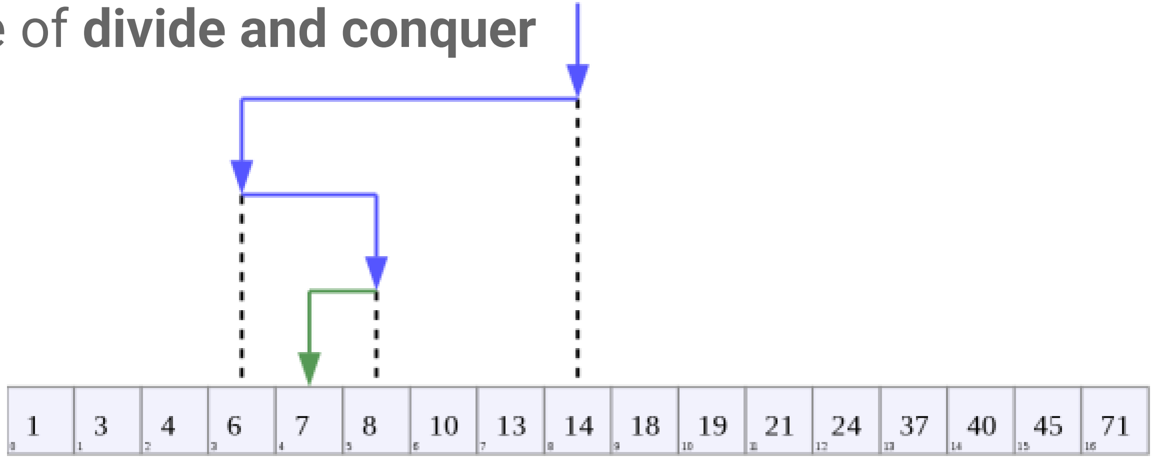
- Back-insert/remove: **O(1)**
- Random insert/remove: **O(*N*)**
- Index-lookup: **O(*N*)**
- Lookup: **O(*N*)**



C++: **`std::list`**
C#: **`System.Collections.Generic.LinkedList`**
Java: **`java.util.LinkedList`**

Binary search

- **Searches** a sorted linear data structure
- Takes **Θ(log(*N*))**
- Example of **divide and conquer**

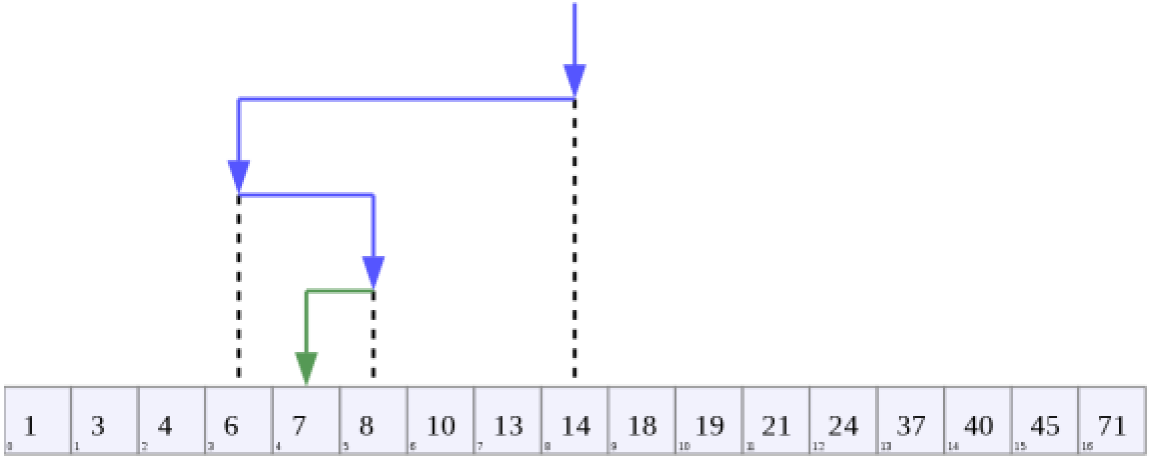


By AlwaysAngry (Own work) [CC BY-SA 4.0],
via Wikimedia Commons

5. Nonlinear data structures

Recall: Binary search

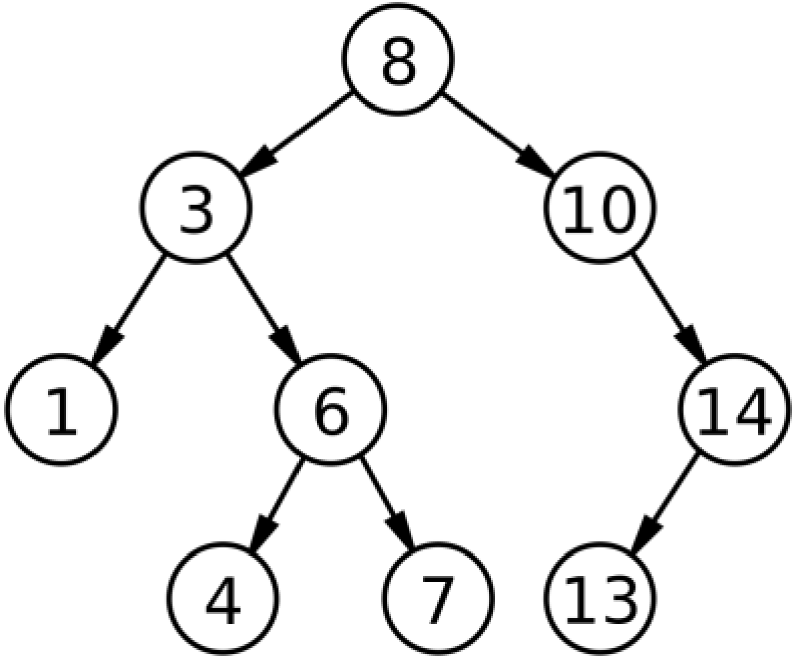
- **Searches** a sorted linear data structure
- Takes **Θ(log(*N*))**
- … let’s use this as inspiration for a **data structure**!



By AlwaysAngry (Own work) [CC BY-SA 4.0],
via Wikimedia Commons

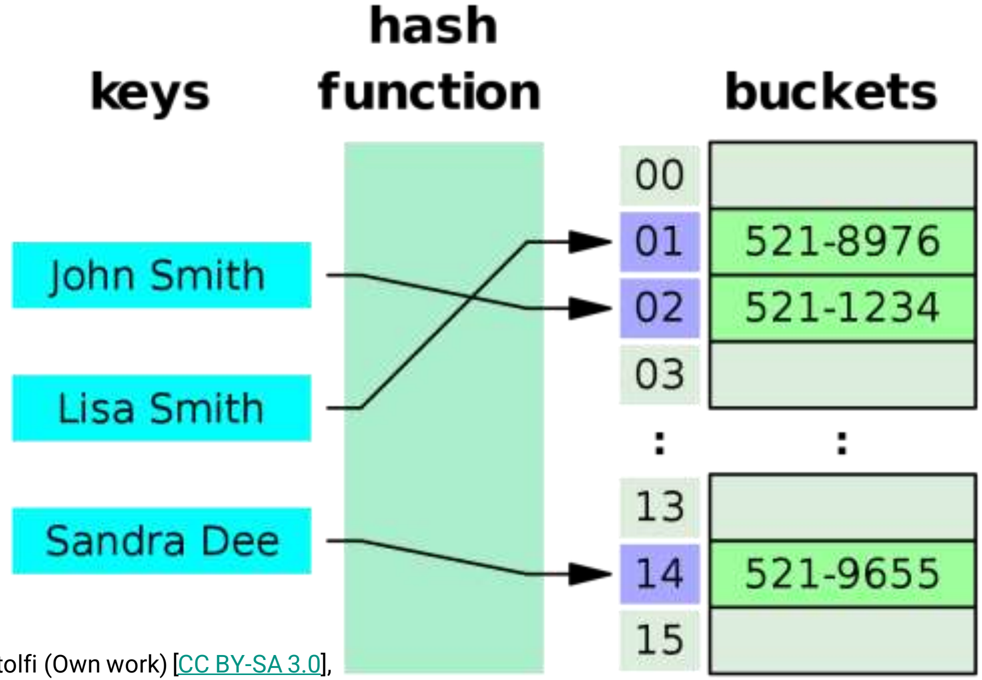
Binary search trees

- **Tree** structure
- **Pointers** between nodes
  - To the right: only **larger**
  - To the left: only **smaller**
- Allows easy **sorted iteration**
- Search/insert/delete:
  all **O(log(*N*))**



Hash tables

- Idea: create **buckets** numbered 1 to B
- For each item, **compute** in which bucket it belongs
- **Put** the item in that bucket
- Search/insert/delete: all **O(1)**

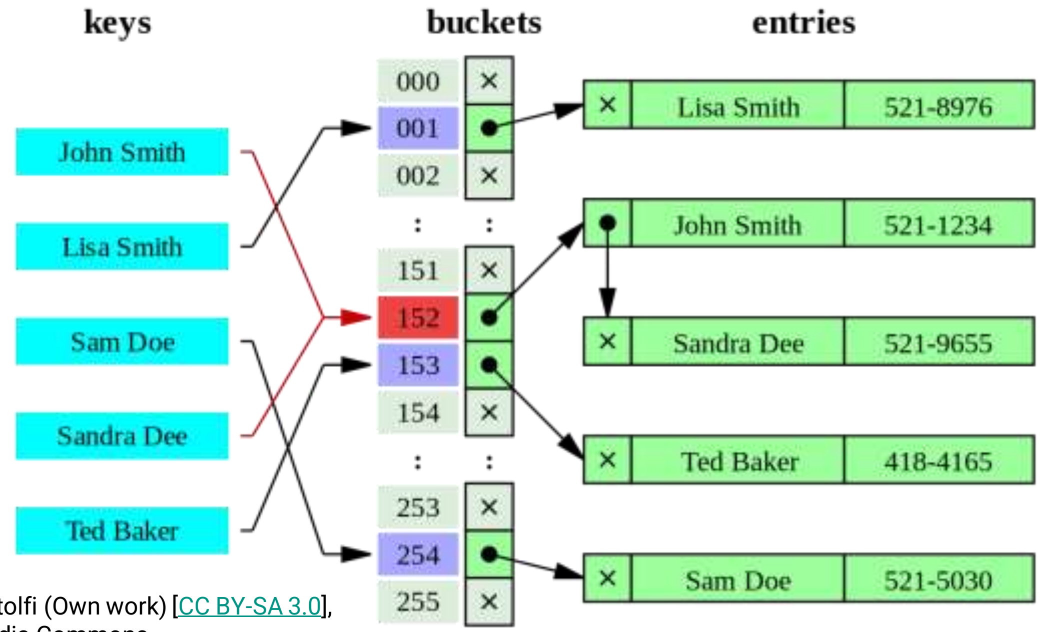


By Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Hash tables

- Problem: **clashing** hashes!
- Solution: replace entry with **linked list** (chaining)

- New problem: **load factor** can become **too high**!
- Solution: **copy** to new table with **more buckets**



By Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Comparing data structures

Data structure → Operation ↓	Dynamic array	Linked list	Binary search tree	Hash table
Lookup	O(N)	O(N)	O(log(N))	O(1)
Indexed lookup	O(1)	O(N)	N/A	N/A
Back-insert	O(1)*	O(1)	O(log(N))	O(1)*
Random insert	O(N)	O(N)	N/A	N/A
Remove	O(N)	O(N)	O(log(N))	O(1)*

*Amortised.

6. Abstract data types

Why “Abstract”?

- Abstract Data Type (**ADT**) does **not** define a real data structure
  - Only defines an **interface**
  - **Implemented** using one of the “real” data structures
- Usually **limits** operations compared to actual DS
- Enhances **flexibility**

Related to several **core programming principles**:

- Program against the **interface**, not the **implementation**!
- Use **high cohesion, loose coupling**
- **Separate** the **concerns**

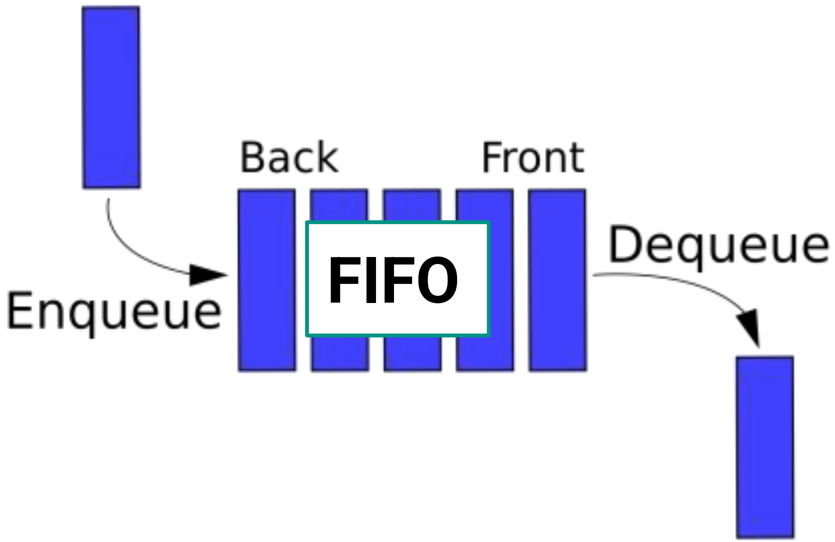
Queue

Operations:

- **Enqueue**: add item to beginning of queue
- **Dequeue**: retrieve and remove item from end of queue

Typical underlying data structure:

- **Linked list**
- Dynamic array



By Vegpuff (Own work) [CC BY-SA 3.0], via Wikimedia Commons

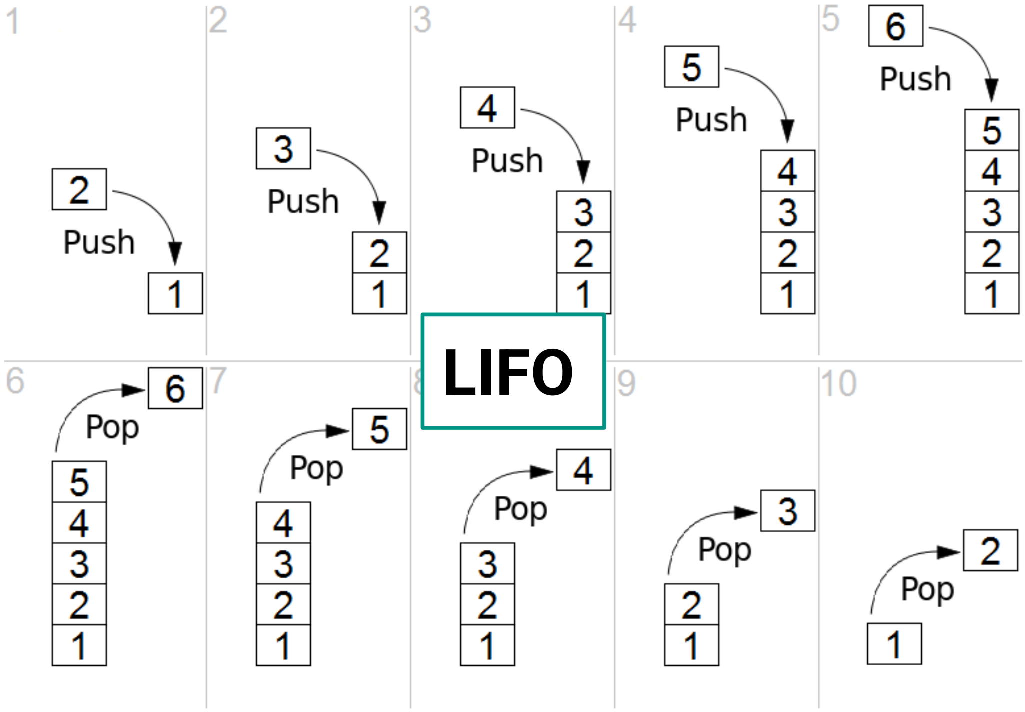
Stack

Operations:

- **Push**: add item to top of stack
- **Pop**: retrieve and remove item from top of stack

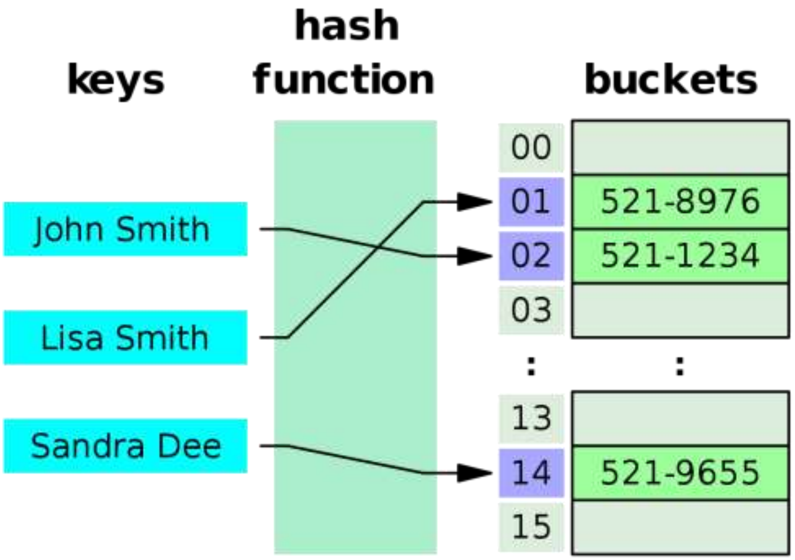
Typical underlying data structure:

- **Linked list**
- Dynamic array



Map

- Map: dataset that maps (associates) **keys** to **values**
- Keys are **unique** (values need not be)
- Values can be **retrieved** by key
- Not indexed…
  - ...although an array could be seen as a map with **integer keys**!

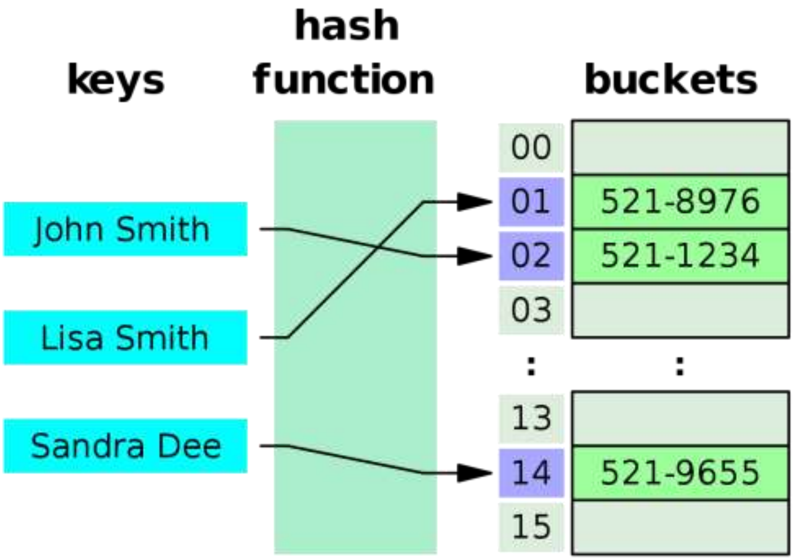


By Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Map

Operations:

- **Lookup**: retrieve value for a key
- **Insert**: add key-value pair
- **Replace**: replace value for a specified key
- **Remove**: remove key-value pair



By Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Map

Typical implementations:

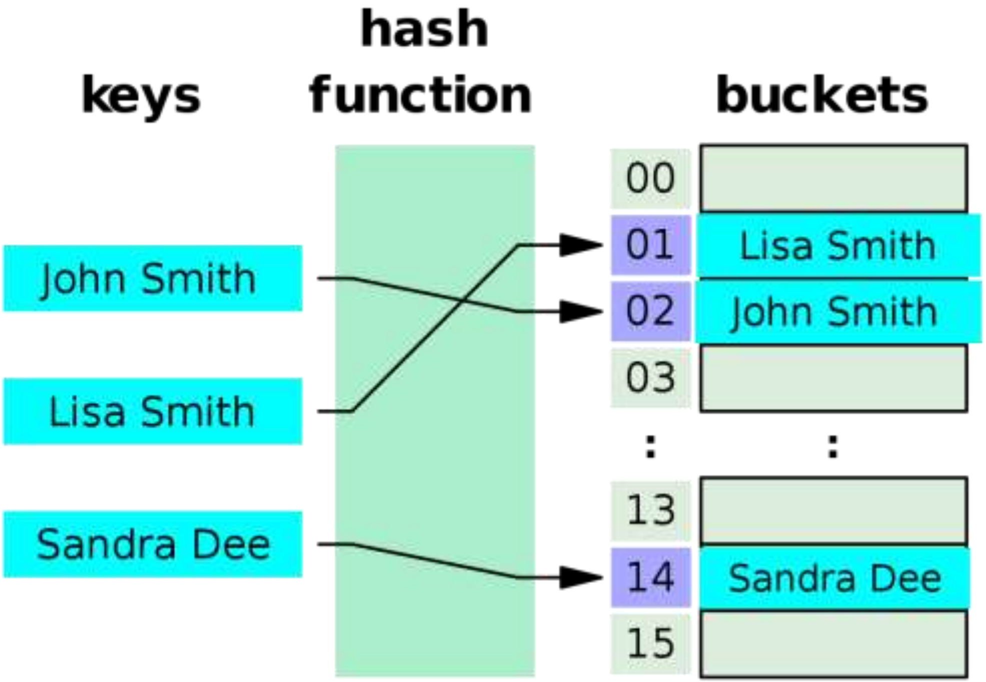
- Binary Search Tree
  - Requires **sortable keys**
  - Can do **indexed/range** queries!
  - Fast with many **insertions**
- Hash Table
  - Generally very **fast**
  - **Space-efficient**
  - Need to keep **load factor** under control...

C++: **`std::map`**
C#: **`System.Collections.Generic`**
`.SortedSet`
Java: **`java.util.TreeMap`**

Python: **`dict`**
C++: **`std::unordered_map`**
Java: **`java.util.HashMap`**

Set

- Set: dataset that contains certain **values**
- No **ordering**, no **multiplicity**
- A value is either **present** or **not**

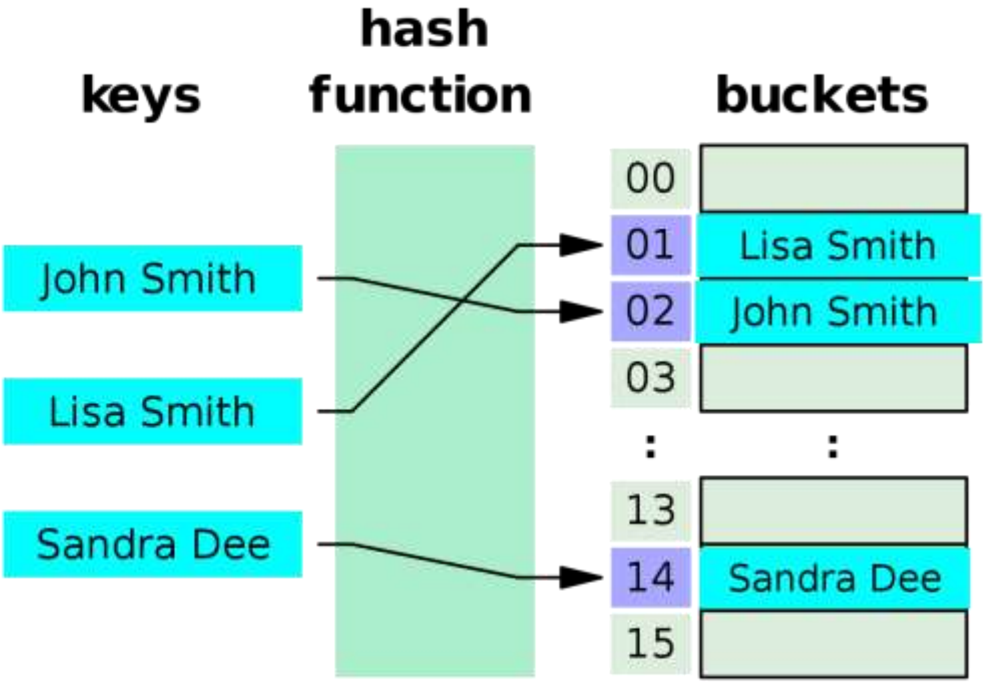


Adapted by L.J. Bel from Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Set

Operations:

- **Contains**: check whether a value is present
- **Add**: add a value
- **Remove**: remove a value



Adapted by L.J. Bel from Jorge Stolfi (Own work) [CC BY-SA 3.0], via Wikimedia Commons

Set

Typical implementations:

- Binary Search Tree
- Hash Table
- *Bloom filter*

C++: **`std::set`**
C#: **`System.Collections.Generic.SortedSet`**
Java: **`java.util.TreeSet`**

Python: **`set`** (and **`frozenset`**)
C++: **`std::unordered_set`**
C#: **`System.Collections.Generic.HashSet`**
Java: **`java.util.HashSet`**

Comparing ADTs

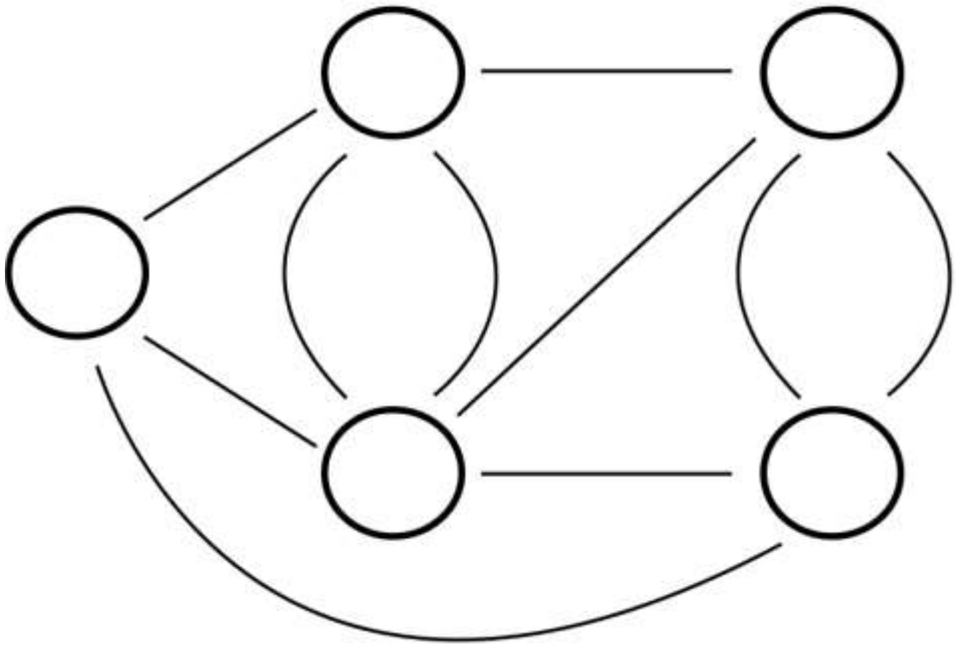
Abstract Data Type → Operation ↓	Queue	Stack	Map	Set
Lookup	N/A*	N/A*	By key	Contains
Add	Enqueue	Push	Key + value	Add
Replace	N/A	N/A	By key	N/A
Remove	Dequeue	Pop	By key	Remove

*Only by removing element
(some may support *peek*)

7. Dijkstra's algorithm

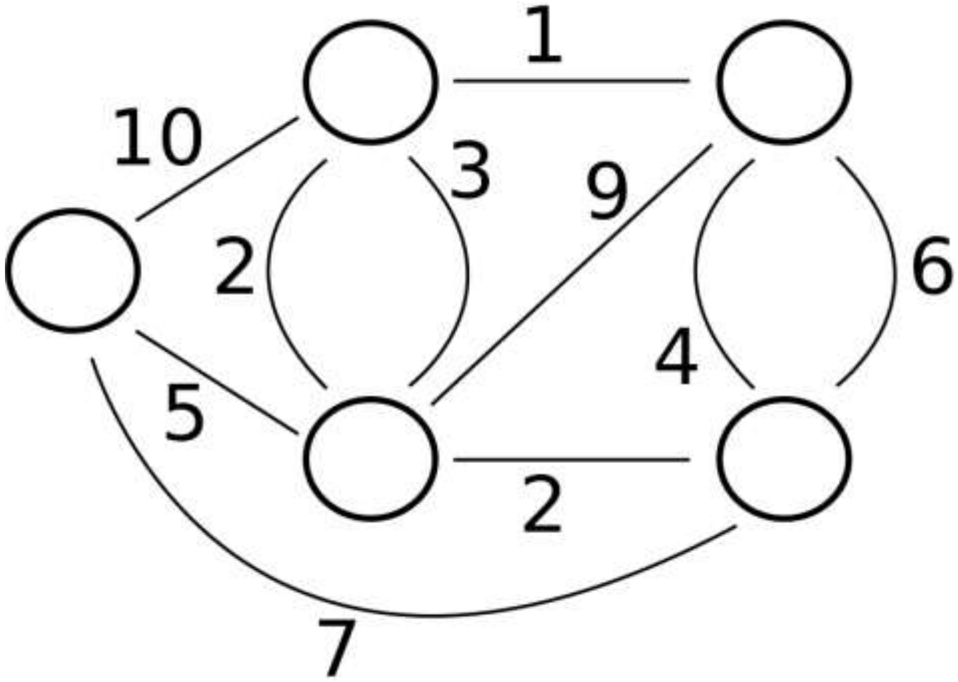
Graphs

- Another data structure!
- Consists of **vertices** (*V*) and **edges** (*E*)



Graphs

- Another data structure!
- Consists of **vertices** (*V*) and **edges** (*E*)
- Edges may carry a **weight**

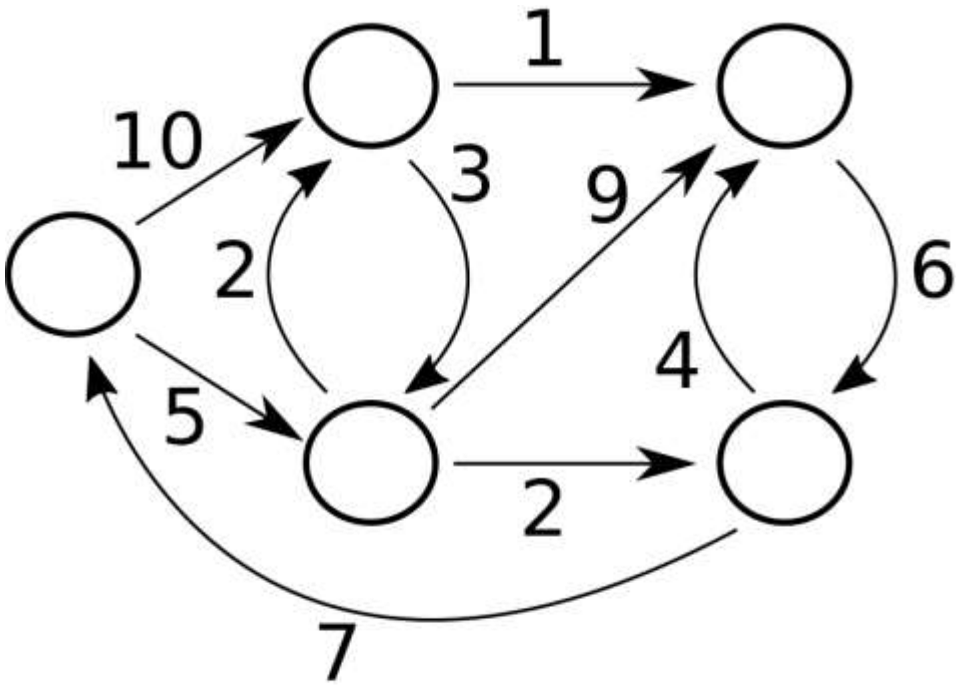


Graphs

- Another data structure!
- Consists of **vertices** (*V*) and **edges** (*E*)
- Edges may carry a **weight**

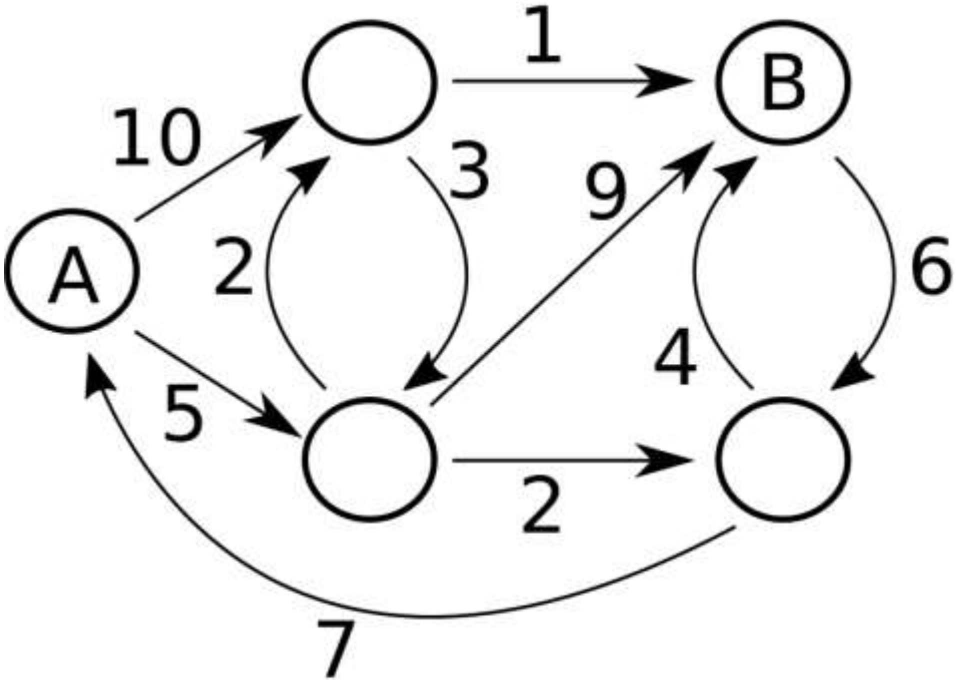
Directed graph:

- Edges are **directed**



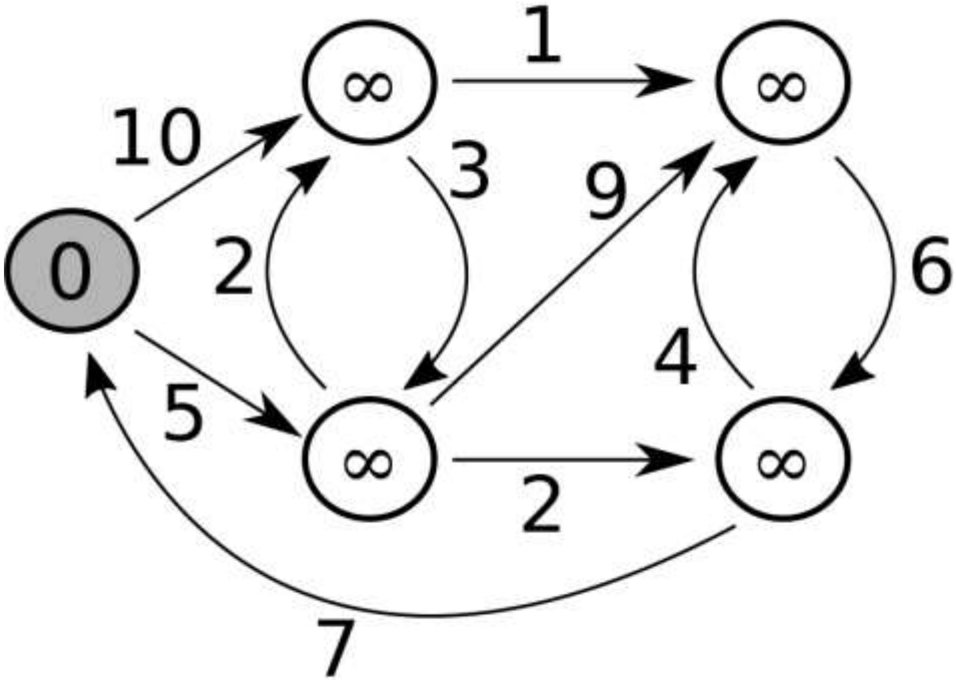
Pathfinding

- Problem: find **shortest path** from A to B
- Shortest is defined as **lowest** total **edge weights**



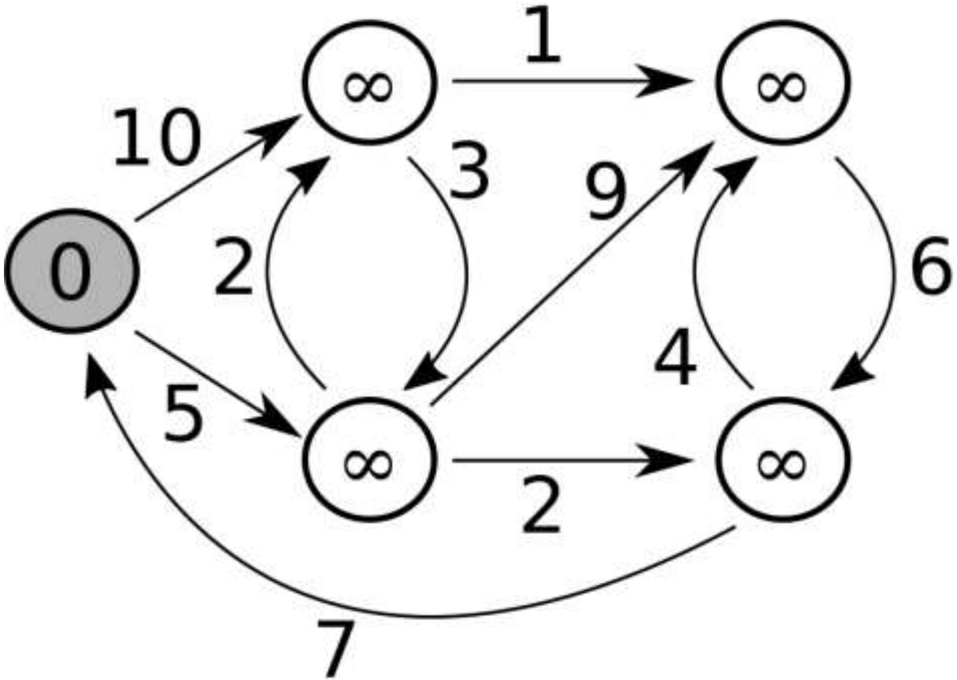
Dijkstra’s algorithm

- Algorithm to obtain **shortest path** from a given vertex to **any other** vertex
- Example of **greedy** algorithm
- Initially: set shortest-path **estimates** to **0** for start vertex and **∞** for the others



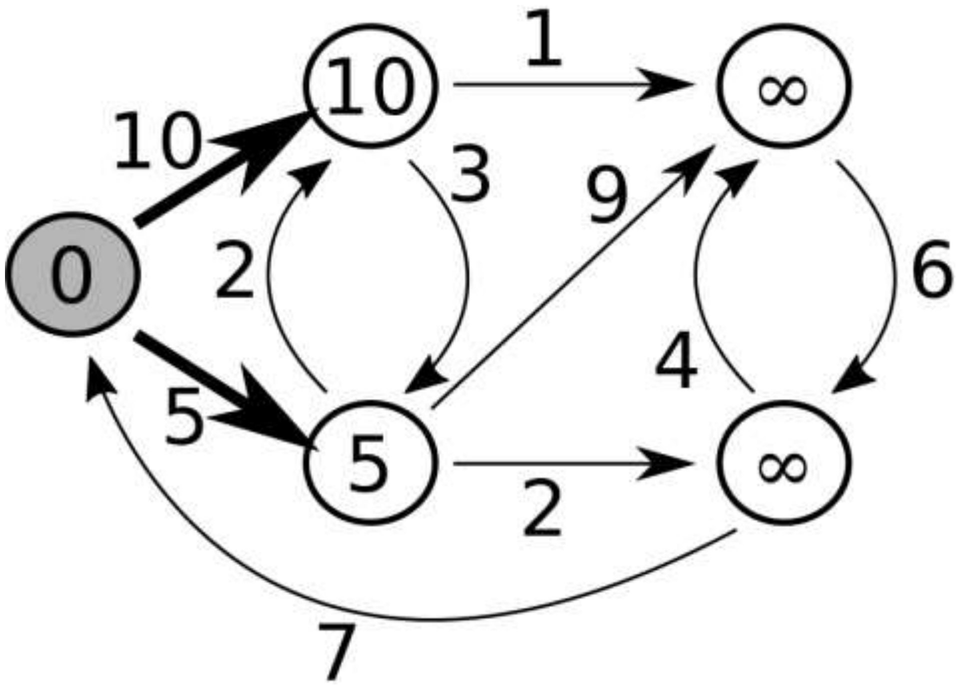
Dijkstra's algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



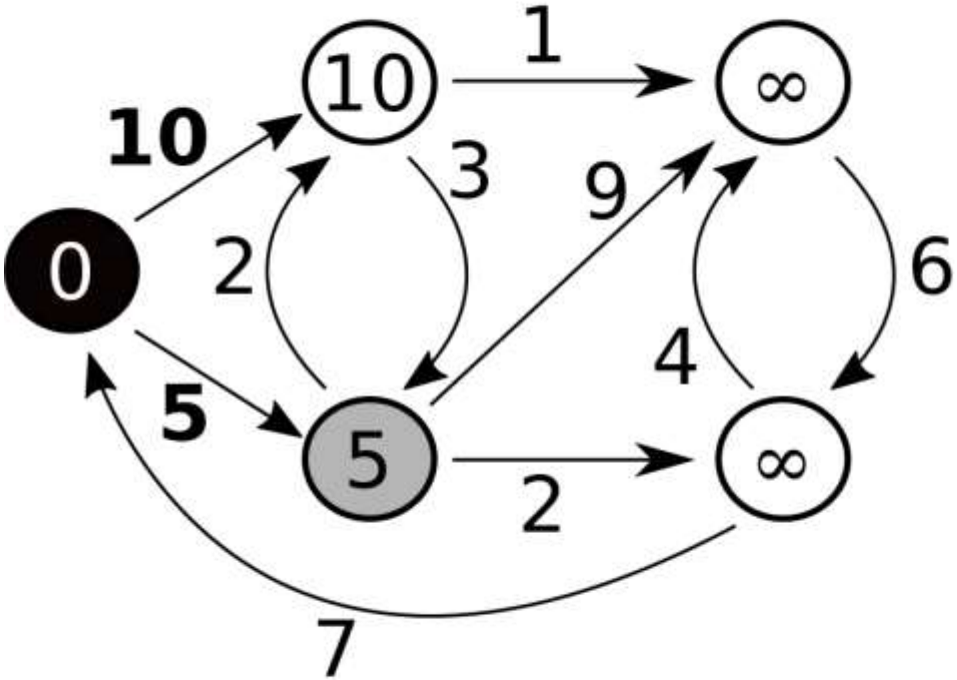
Dijkstra's algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



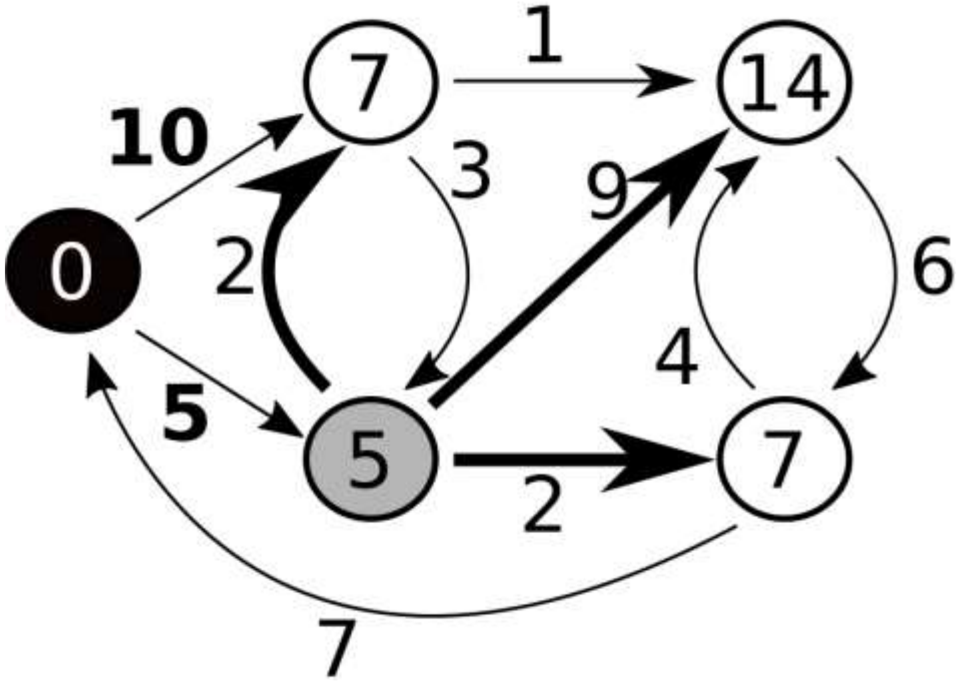
Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



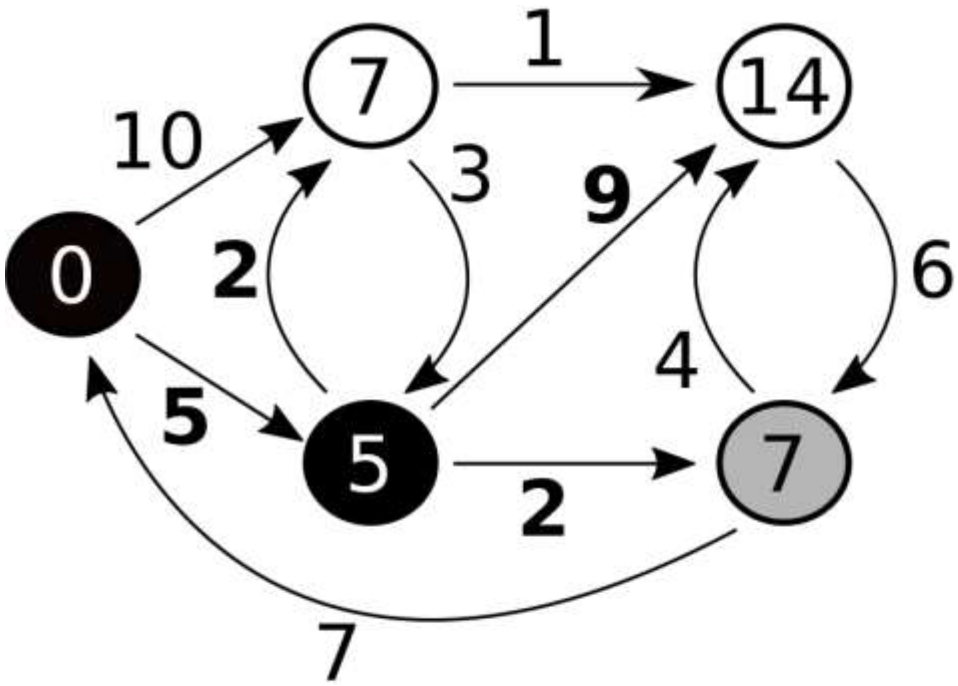
Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



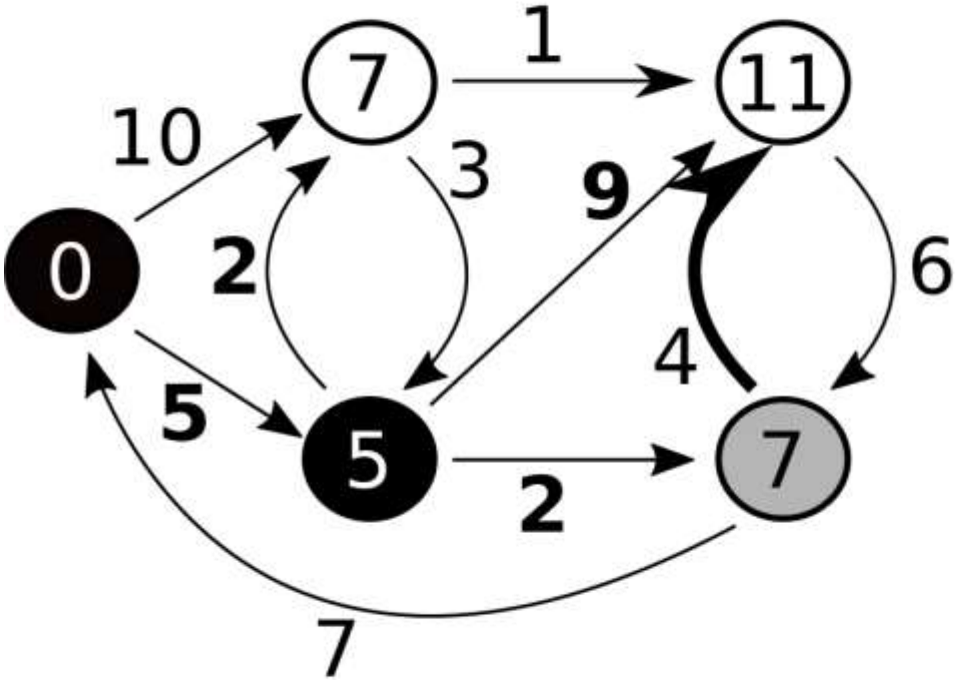
Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



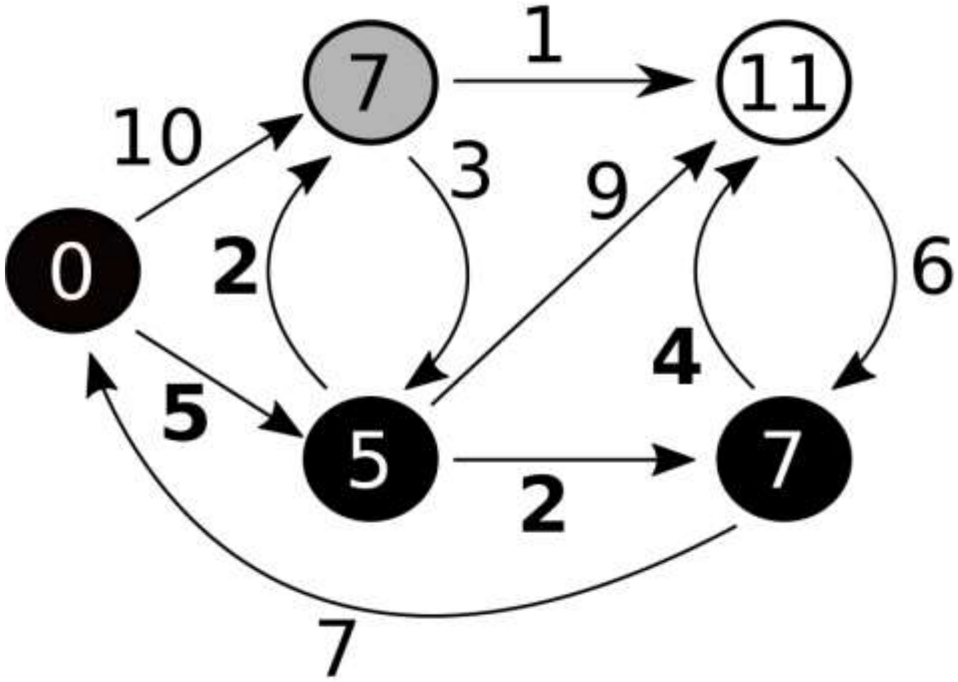
Dijkstra's algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



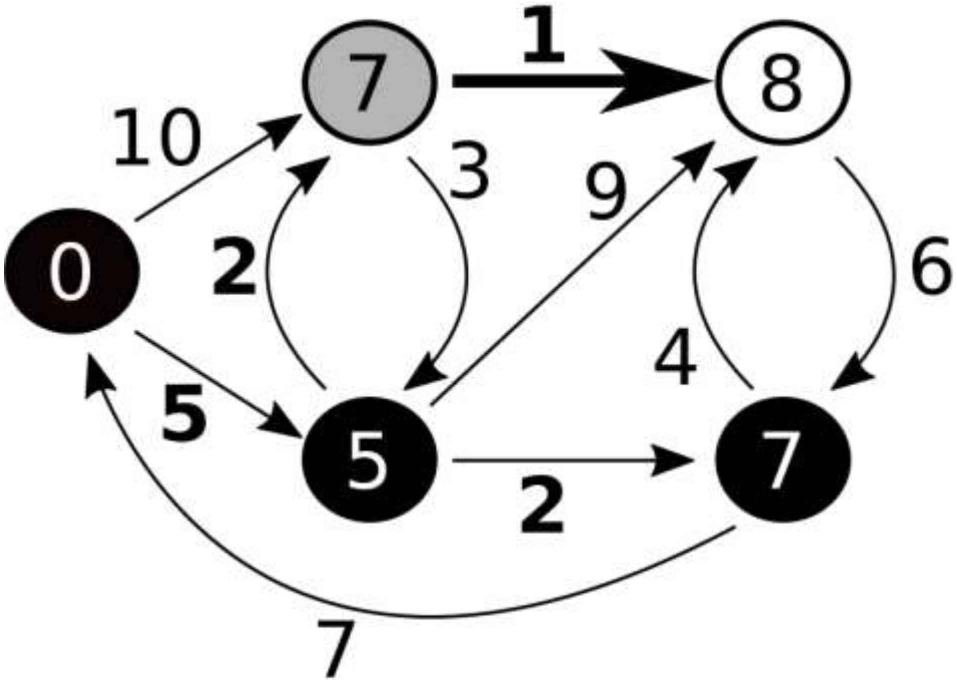
Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



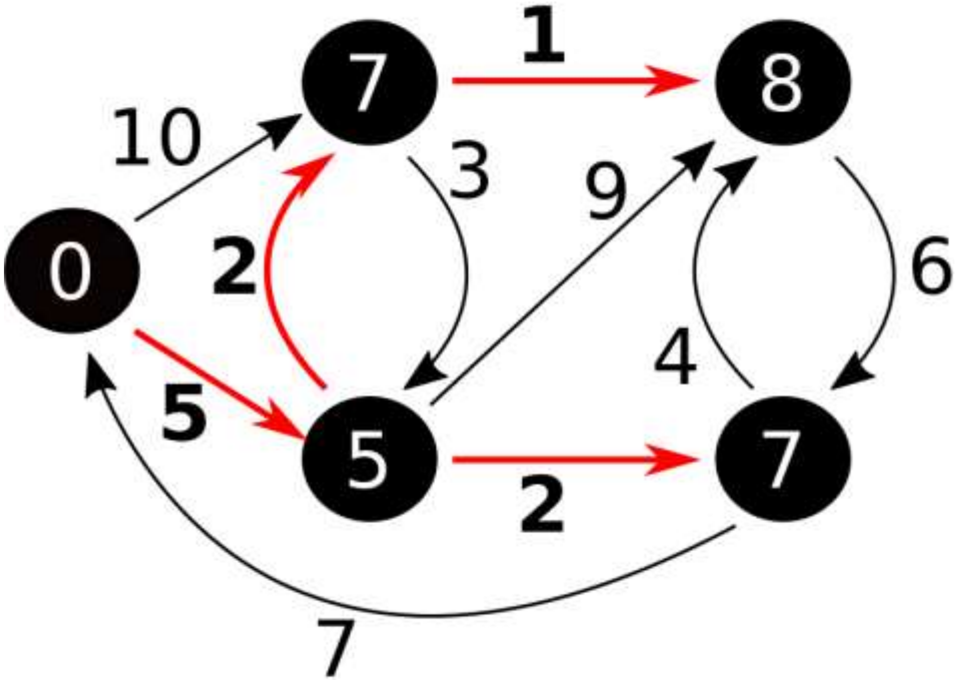
Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate



Dijkstra’s algorithm

- Repeat the following:
  - Select unvisited vertex with **lowest** estimate
  - Look at paths to **unvisited** nodes
  - Update estimates if **lower** than previous estimate
- Shortest paths indicated in red
- Complexity: **O(*E* + *V* log *V*)**



8. Summary

Summary

- Concepts

- Divide and conquer
- (Un)stable sorting
- Pointers

Summary

- Concepts
- Sorting algorithms

- Insertion sort
- Bubble sort
- Merge sort
- Quicksort

Summary

- Concepts
- Sorting algorithms
- Data structures

- Arrays
- Dynamic arrays
- (Doubly) linked lists
- Binary search trees
- Hash tables
- (Directed) graphs

Summary

- Concepts
- Sorting algorithms
- Data structures
- Abstract data types

- Queues
- Stacks
- Maps
- Sets

Summary

- Concepts
- Sorting algorithms
- Data structures
- Abstract data types
- Algorithms
  - Binary search
  - Dijkstra’s algorithm