Algorithms and Data Structures

Graphs – Part III – Prim & Dijkstra

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

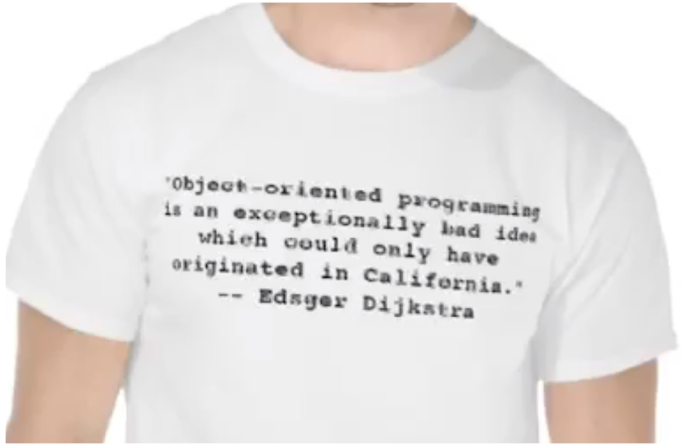
Program

- Shortest Paths Tree; **Dijkstra's Algorithm**, recap by exploration.
- Shortest Path: the **A* Algorithm**, student presentation.

  Benjamin has volunteered to present the A* Algorithm. Active student participation is encouraged. Thank you Benjamin!
- Some Exercises
- Quiz – where do we stand? (a self-service check)

  We're at a divide: should we fastforward, or fastreverse a bit from here?
- Time permitting: precendence scheduling, topological sort

Shortest Paths



1. Path Relaxation
2. Dijkstra's Algorithm – today for simple digraphs only.

The Shortest Paths Problem (single-source)

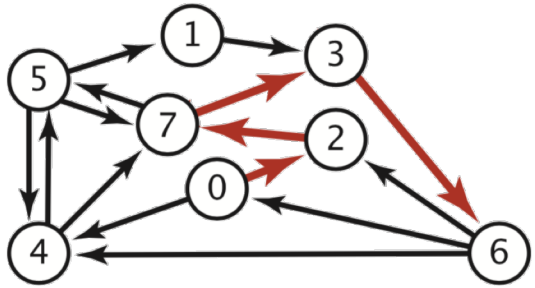
Given a digraph, find the shortest path between a vertex ‘s’ (source) and one other vertex, or several (all) other vertices.

Assume simple digraphs only:

- *Single source*: from one vertex *s* to every other vertex.
- Weights positive
- No negative cycles

edge-weighted digraph

```
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```

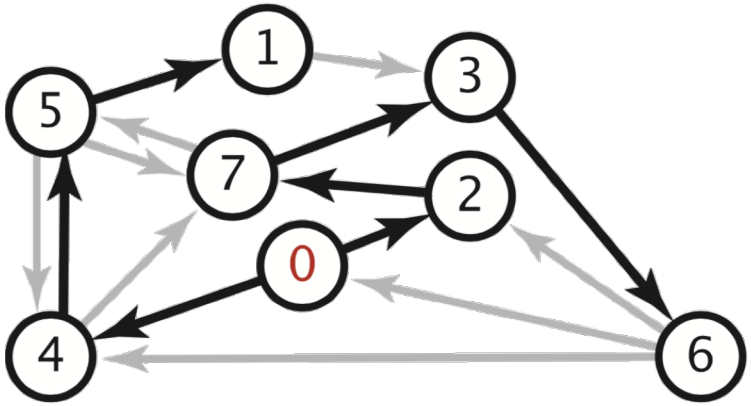


shortest path from 0 to 6

```
0->2  0.26
2->7  0.34
7->3  0.39
3->6  0.52
```

Shortest Path Tree

- Shortest Paths Tree, from computing all single-source paths from ‘s’.
- Represented by two arrays, `edgeTo[]` and `distTo[]`.

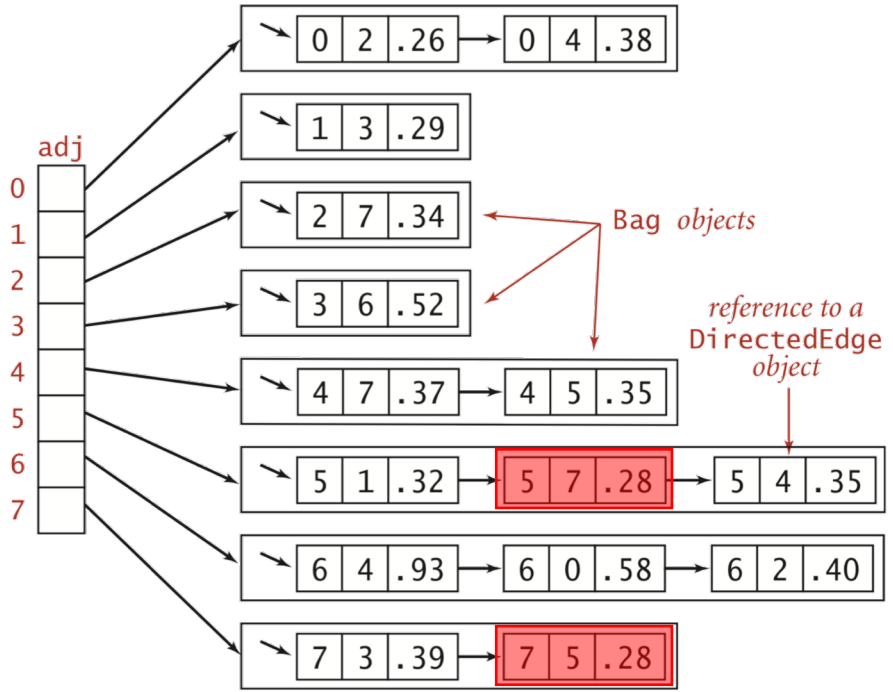


	edgeTo[]	distTo[]
0	null	0
1	5->1 0.32	1.05
2	0->2 0.26	0.26
3	7->3 0.37	0.97
4	0->4 0.38	0.38
5	4->5 0.35	0.73
6	3->6 0.52	1.49
7	2->7 0.34	0.60

What Data Structure for Dijkstra *et al.*?

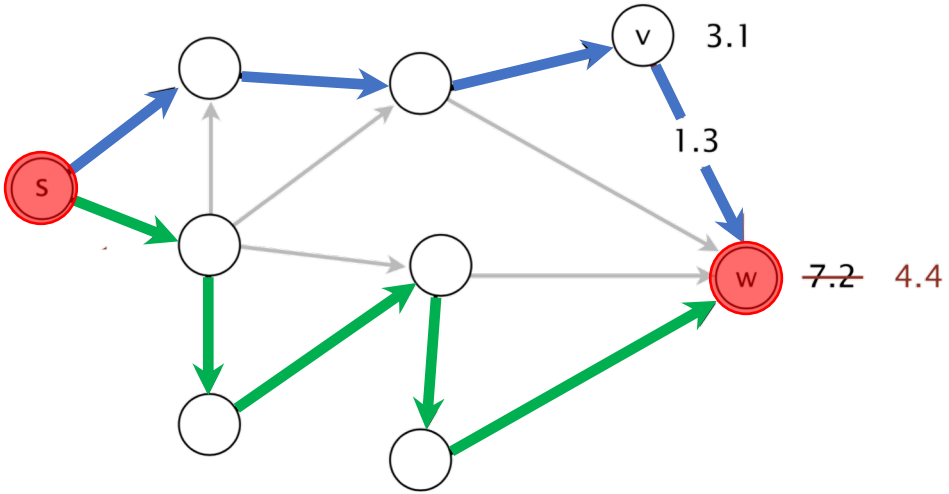
tinyEWD.txt

```
V → 8
    15 ← E
4 5  0.35
5 4  0.35
4 7  0.37
5 7  0.28
7 5  0.28
5 1  0.32
0 4  0.38
0 2  0.26
7 3  0.39
1 3  0.29
2 7  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```



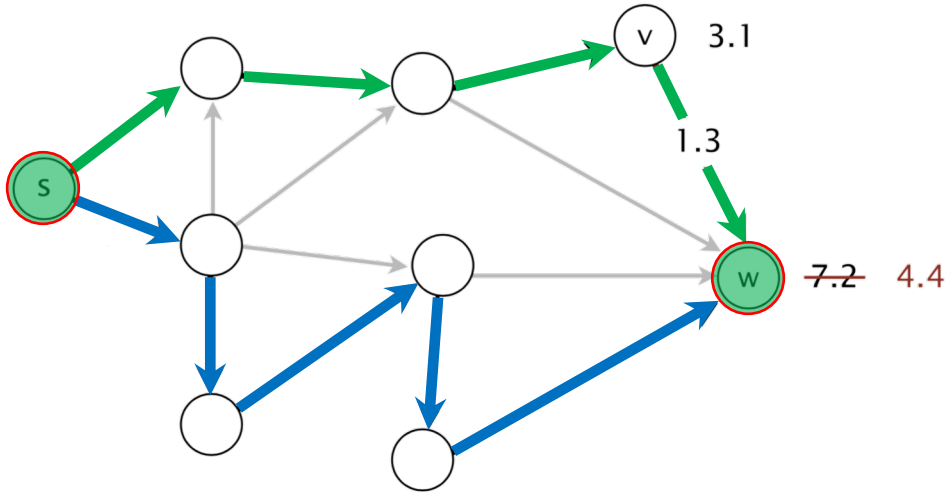
Edge Relaxation

If `e = v → w` gives shorter path to `w` through `v`, update both `distTo[w]` and `edgeTo[w]`.



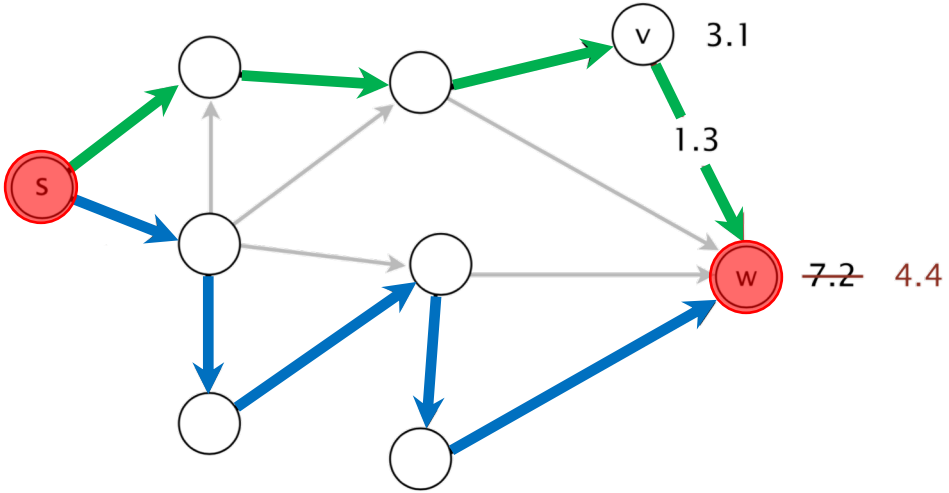
Edge Relaxation

If `e = v → w` gives shorter path to `w` through `v`, update both `distTo[w]` and `edgeTo[w]`.



A Generic SPT Algorithm

1. distTo[s] = 0
2. distTo[v] = ∞
3. While not all optimal:
   Relax edges
4. Done.



Exercise 4

- $e = v \longrightarrow w$ is an edge with weight 17.0.
- At some point, during the generic shortest paths algorithm, `distTo[v]` = ∞ and `distTo[w]` = 15.0. What will `distTo[w]` be after calling `relax(e)`?

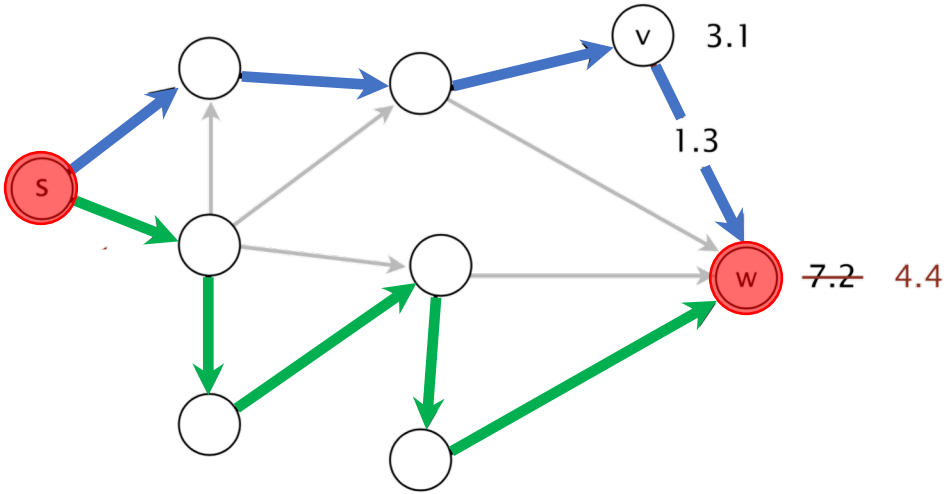
- ❑ The program will encounter a Java runtime exception
- ❑ 15.0
- ❑ 17.0
- ❑ ∞

Edge Relaxation HOWTO?

Three central algorithms exist:

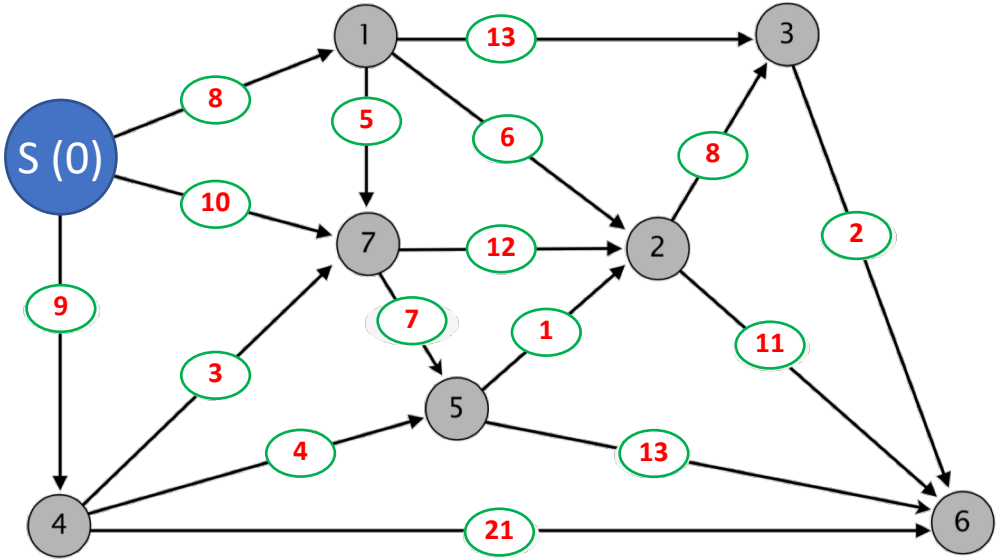
1. Dijkstra's Algorithm
2. Topological Sorting
3. Bellman-Ford Algorithm

We might just yet do something about number 2....



Exercise 5: Using Dijkstra's Algorithm

Find the SPT from source v=0 to all other vertices.



V	distTo[]	edgeTo[]
0	0.0	-
1	8.0	0 → 1
2		
3		
4		
5		
6		
7		

Trade-off hints:

1) [preferred!] you build the graph as a text file and run it though some software... requires some manual labor, though.

2) you can do it by brute force (paper/pen), keeping track like in the table here.. learning the algorithm first, though.

Dijkstra's Algorithm

- Keeping the queued vertices/edges on a minPQ (heap).

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

Exercise 6

For the five example graphs

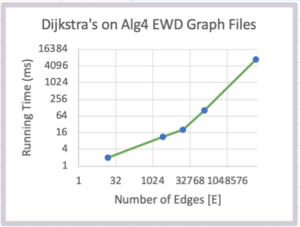
- `tinyEWG.txt` (V=8, E=15)
- `mediumEWG.txt` (V=250, E=2546)
- `1000EWG.txt` (V=1000, E=16866)
- `10000EWG.txt` (V=10000, E=123462)
- `largeEWG.txt` (V=1000000, E=15172126)

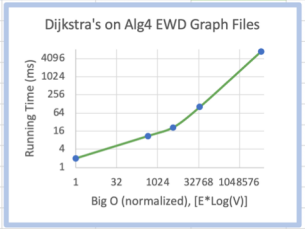
compare running times, using the `DijkstraSP()` class and the `SP()` class.

Time your computations. You can use fx the Book's `StopWatch()` class for this purpose. Or a system time diff, or you pick...

Exercise 6 – a few results.

V	E	Big O(Dijkstra, E*log(V))	norm, BigO	time 0	time 1	time 2	time 3
8	15	45	1	16	2	0,1	1
250	2546	20280,88679	450,6863731	16	11	3	8
1000	16866	168082,9177	3735,17595	16	21		0,006
10000	123462	1640527,546	36456,16768	109	101		
1000000	15172126	302404269,7	6720094,882	3208	7110	3158	





For the five example graphs

- `tinyEWG.txt` (V=8, E=15)
- `mediumEWG.txt` (V=250, E=2546)
- `1000EWG.txt` (V=1000, E=16866)
- `10000EWG.txt` (V=10000, E=123462)
- `largeEWG.txt` (V=1000000, E=15172126)

compare running times, using the `DijkstraSP()` class and the `SP()` class.

Time your computations. You can use fx the Book's `StopWatch()` class for this purpose. Or a system time diff, or you pick...

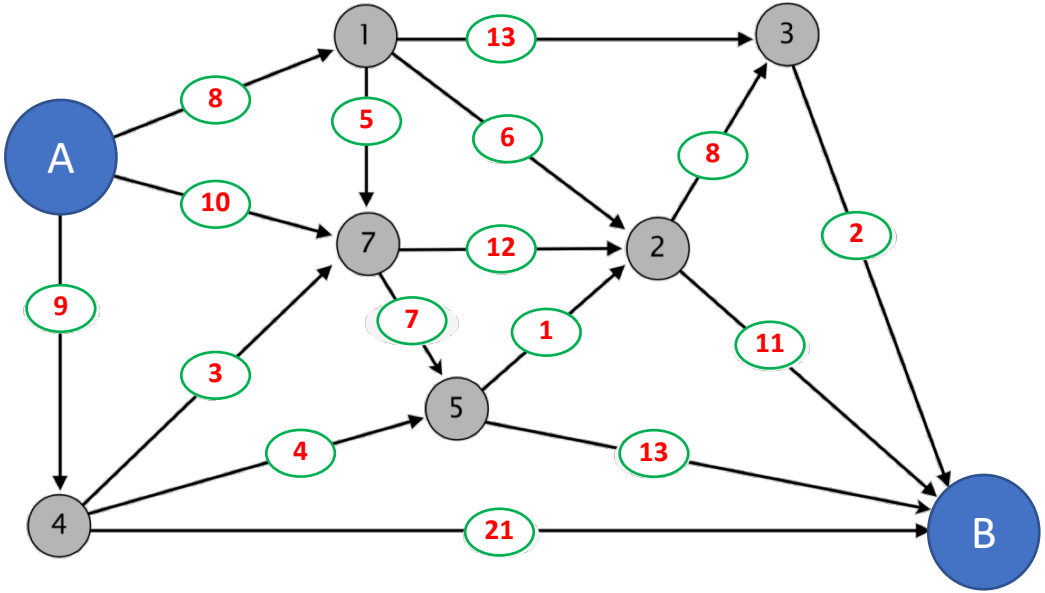
Exercise 7

Q: What is the BigO running time of Dijkstra's algorithm, using a binary heap?

- V
- E
- V^2
- $E \; Log \; V$

Benjamin presenting the A* Algorithm

Find the shortest path from A to B, with parametrized heuristic search algorithm.



V	distTo[]	edgeTo[]
0	0.0	-
1	10.0	0 → 1
2		
3		
4		
5		
6		
7		

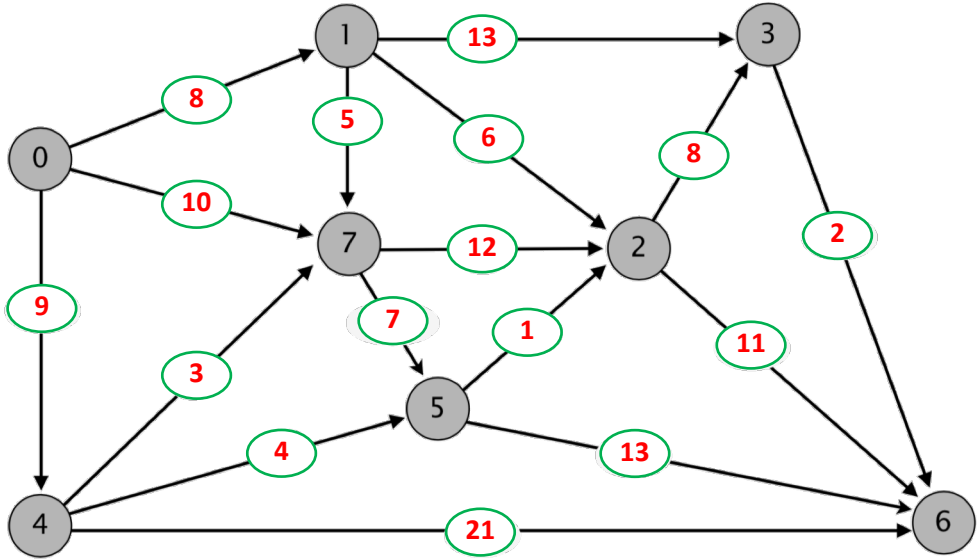
Trade-off methods:

1) [preferred!] you build the graph as a text file and run it though some software... requires some manual labor, though.

2) you can do it by brute force (paper/pen), keeping track like in the table here.. learning the algorithm first, though.

Exercise 8: Using the A* Algorithm

Find the shortest path from source v=0 to v=6.



V	distTo[]	edgeTo[]
0	0.0	-
1	10.0	0 → 1
2		
3		
4		
5		
6		
7		

Trade-off hints:

1) you build the graph as a text file and run it though some software... requires some manual labor, though.

2) you can do it by brute force (paper/pen), keeping track like in the table here.. learning the algorithm first, though.

Quiz – where do we stand?

A self-service check.

1) Take a piece of paper ...
2) ... and a pen ...
3) ... and score the following review in terms of what you think don't know/remember, per item on slide #....
4) let's collect and see what to review asap.
5) ☺

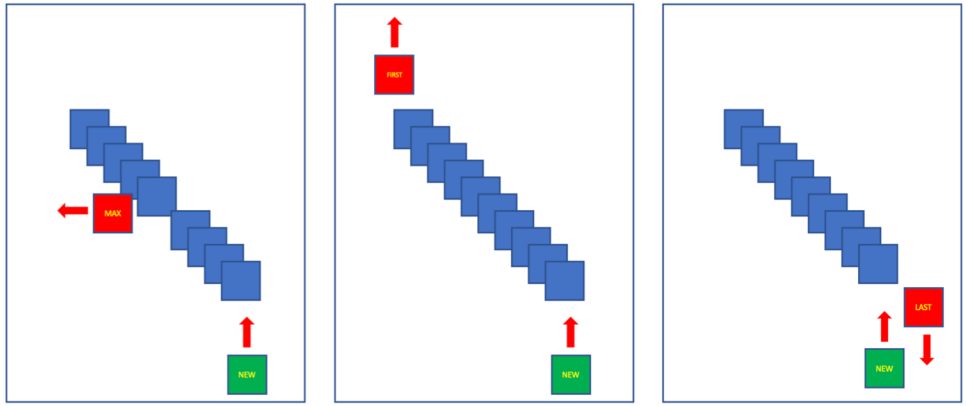
Quiz – Chapter 1: *'Fundamentals'*

Data structures

- Bags, queues, stacks

Analysis of Algorithms

- Running time
- Memory usage
- Big O notation



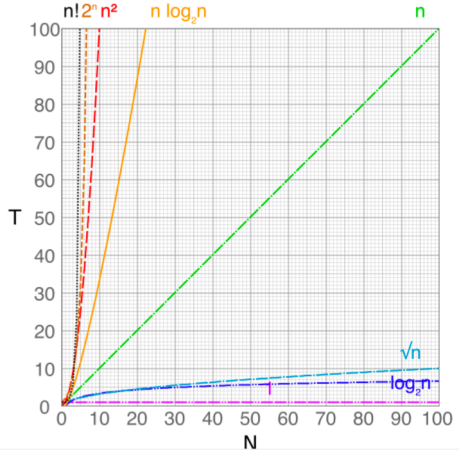
Quiz – Chapter 1: *'Fundamentals'*

Data structures

- Bags, queues, stacks

Analysis of Algorithms

- Running time
- Memory usage
- Big O notation



$$\mathcal{O}(2^N)$$

```
// We will measure the following in time units only,
// neglecting any movement of data and excess storage
// needs.
int count(int[] a) {
  int n = a.length;                          // ~ 1
  int count = 0;                             // ~ 1
  for (int i = 0; i < n; i++)                // ~ N
   for (int j = i + 1; j < n; j++)           // ~ N
    for (int k = j + 1; k < n; k++)          // ~ N
     if (a[i] + a[j] + a[k] == 0) count++    // ~ 1
  return count;
  }
```

Total units of time consumption:

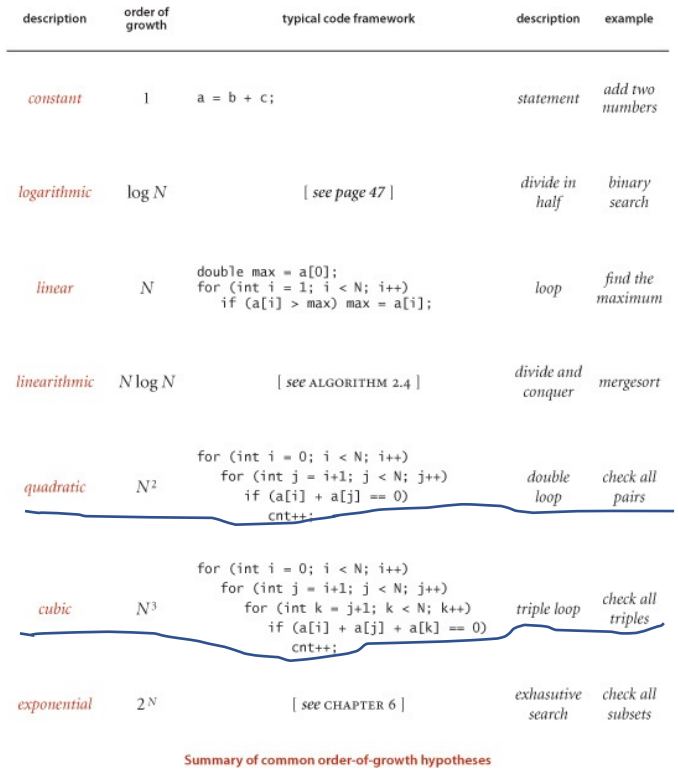
$$\mathcal{O}(1 + 1 + N + N^2 + N^3 + 1) \approx \mathcal{O}(N^3)$$

Exercise 3

- Show by any means (perhaps a summation series) that the running time complexity of computing the brute force 'Two-Sum-Zero' is $O(N^2)$:

$$C(twosumzero) \approx \frac{N(N-1)}{2} \backsim O(N^2).$$

- Extend you argument to show that the running time complexity of computing the brute force '**Three**-Sum-Zero' is $O(N^3)$.
- What is the worst case order of growth – in terms of Big O?



description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	statement	add two numbers
logarithmic	$\log N$	[see page 47]	divide in half	binary search

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	find the maximum

linearithmic	$N \log N$	[see ALGORITHM 2.4]	divide and conquer	mergesort

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	check all pairs

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`
			triple loop	check all triples

exponential	2^N	[see CHAPTER 6]	exhasutive search	check all subsets

Summary of common order-of-growth hypotheses

Quiz – Chapter 2: *'Sorting'*

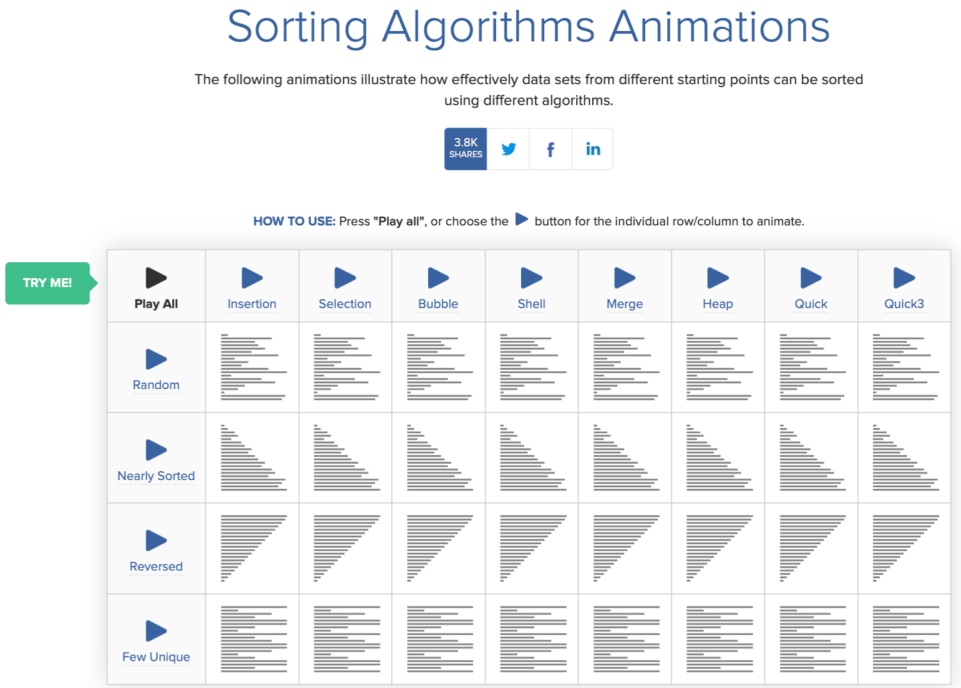
Elementary Sorts

- Insertion sort
- Selection sort

Mergesort

Quicksort

Priority Queues, heaps



Quiz – Chapter 2: *'Sorting'*

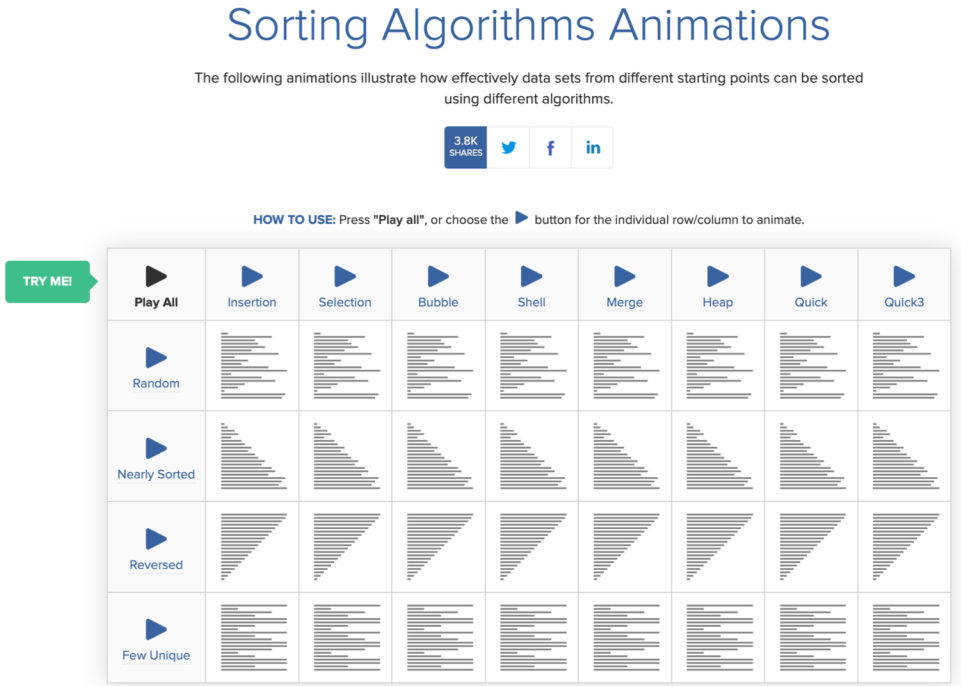
Elementary Sorts

- Insertion sort
- Selection sort

Mergesort

Quicksort

Priority Queues, heaps



Quiz – Chapter 2: *'Sorting'*

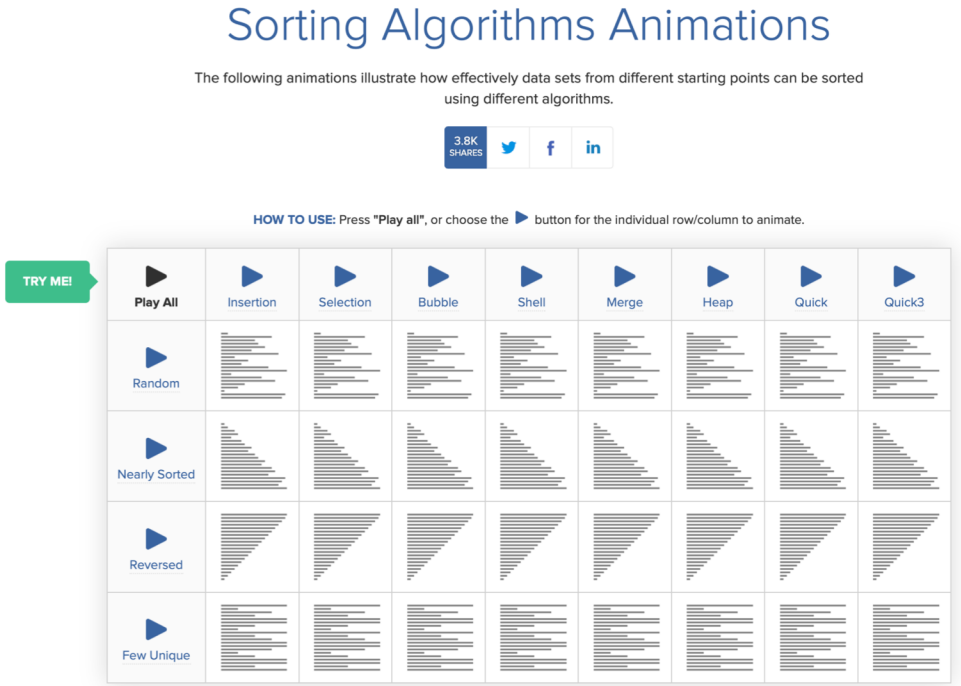
Elementary Sorts

- Insertion sort
- Selection sort

Mergesort

Quicksort

Priority Queues, heaps



Quiz – Chapter 2: *'Sorting'*

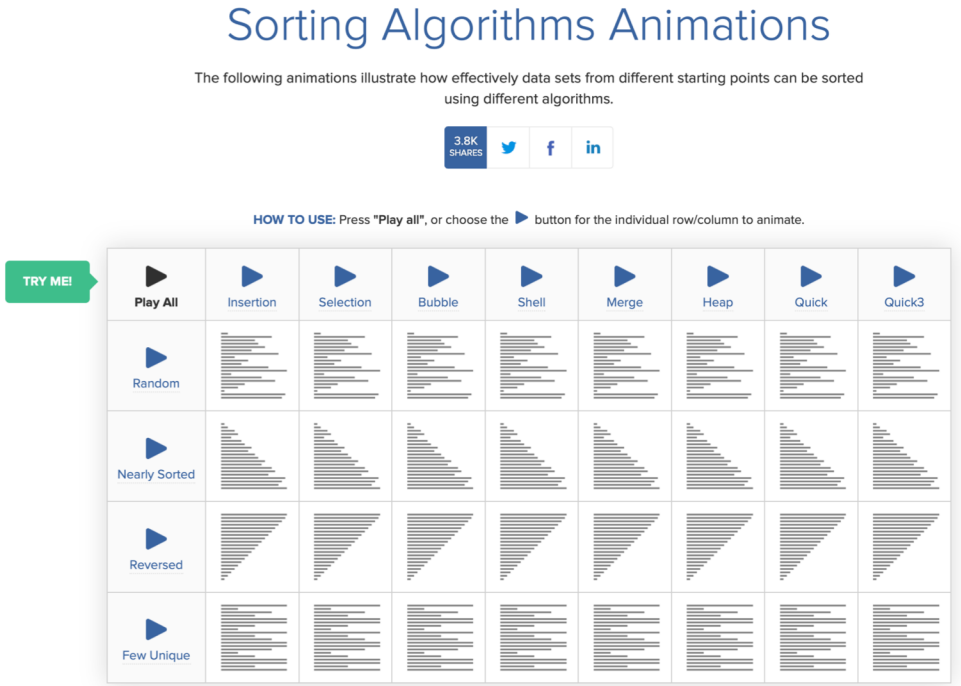
Elementary Sorts

- Insertion sort
- Selection sort

Mergesort

Quicksort

Priority Queues, heaps



Quiz – Chapter 3: *'Searching'*

Symbol Tables

- Sequential Search (un-ordered list)
- Binary Search (ordered list)

Binary Search Trees

Balanced Search Trees

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	$\frac{1}{2} N$	N	$\frac{1}{2} N$		equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2} N$	$\frac{1}{2} N$	✔	compareTo()
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	compareTo()
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	✔	compareTo()

Quiz – Chapter 3: *'Searching'*

Symbol Tables

- Sequential Search (un-ordered list)
- Binary Search (ordered list)

Binary Search Trees

Balanced Search Trees

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	$\frac{1}{2} N$	N	$\frac{1}{2} N$		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2} N$	$\frac{1}{2} N$	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	`compareTo()`
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	✔	`compareTo()`

Quiz – Chapter 3: *'Searching'*

Symbol Tables

- Sequential Search (un-ordered list)
- Binary Search (ordered list)

Binary Search Trees

Balanced Search Trees

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	$\frac{1}{2} N$	N	$\frac{1}{2} N$		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2} N$	$\frac{1}{2} N$	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	`compareTo()`
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	✔	`compareTo()`

Quiz – Chapter 4: *'Graphs'*

Un-directed Graphs

- Adjacency lists data structure representation
- Depth-first search (DFS, stack)
- Breadth-first Search (BFS, queue)
- Connected components

Directed Graphs

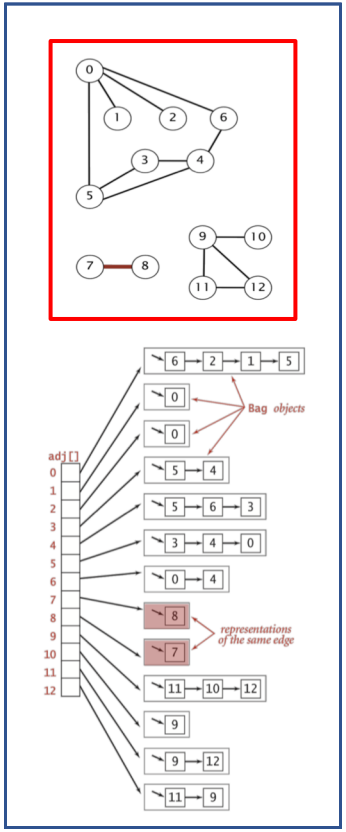
- Precendence-constrained Scheduling
- Topological Sort

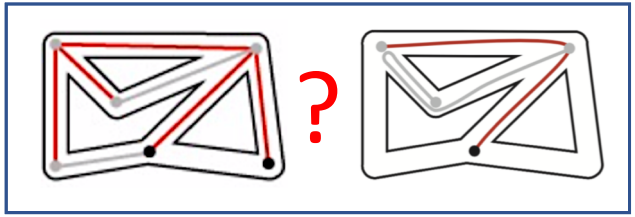
Minimum Spanning Trees

- Greedy Algorithm
- Prim's Algorithm

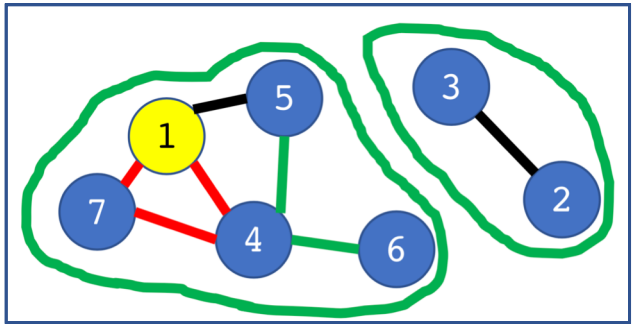
Shortest Paths

- Dijkstra's Algorihm
- A* Algorithm





representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	*degree*(v)	*degree*(v)



Quiz – Chapter 4: *'Graphs'*

Un-directed Graphs

- Adjacency lists data structure representation
- Depth-first search (DFS, stack)
- Breadth-first Search (BFS, queue)
- Connected components

Directed Graphs

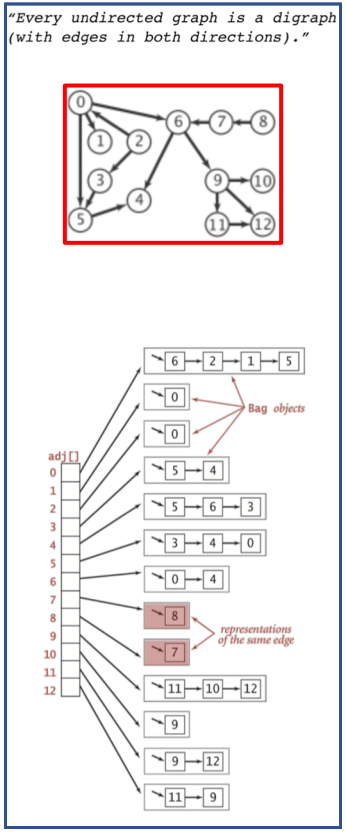
- Precendence-constrained Scheduling
- Topological Sort

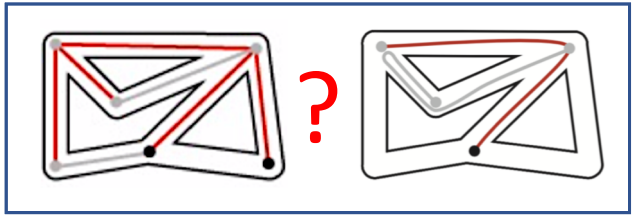
Minimum Spanning Trees

- Greedy Algorithm
- Prim's Algorithm

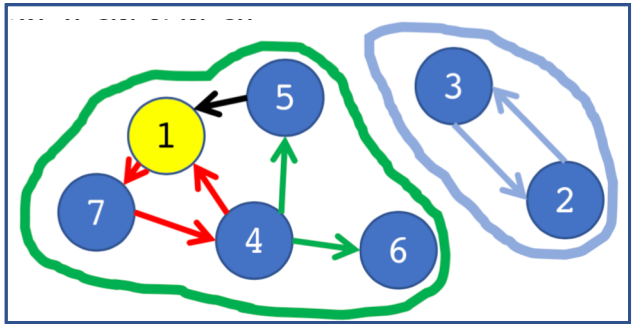
Shortest Paths

- Dijkstra's Algorihm
- A* Algorithm





representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	*degree*(*v*)	*degree*(*v*)



Quiz – Chapter 4: *'Graphs'*

Un-directed Graphs

- Adjacency lists data structure representation
- Depth-first search (DFS, stack)
- Breadth-first Search (BFS, queue)
- Connected components

Directed Graphs

- Precendence-constrained Scheduling
- Topological Sort

Minimum Spanning Trees

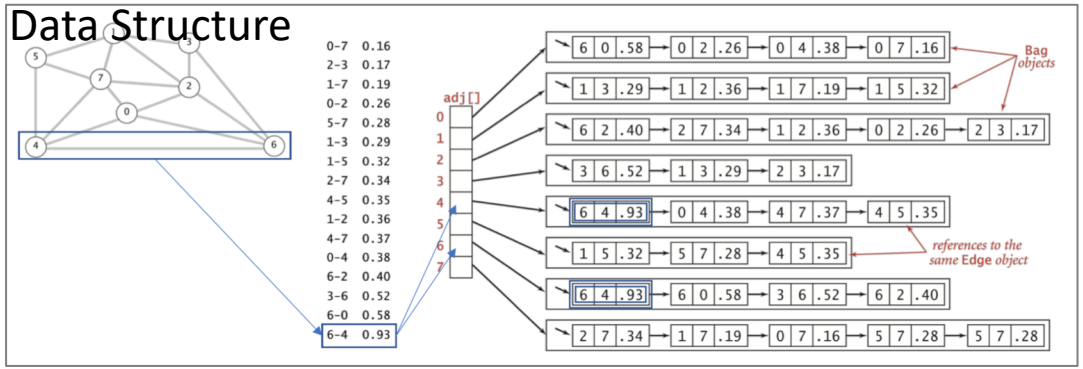
- Greedy Algorithm
- Prim's Algorithm

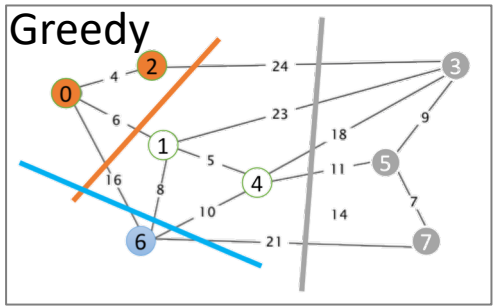
Shortest Paths

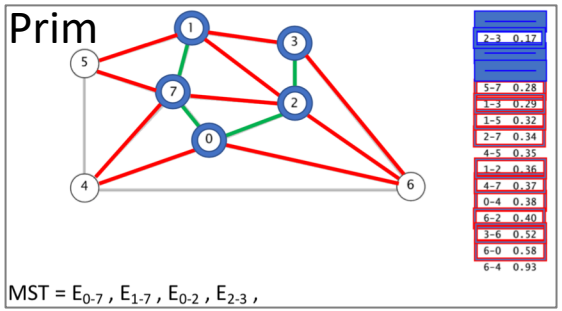
- Dijkstra's Algorihm
- A* Algorithm

MST Complexity; "All" Algorithms.

Algorithm	Space	Time
Lazy Prim	E	E Log E
Eager Prim	*V*	*E Log V*
Kruskal	*E*	*E Log E*







Quiz – Chapter 4: *'Graphs'*

Un-directed Graphs

- Adjacency lists data structure representation
- Depth-first search (DFS, stack)
- Breadth-first Search (BFS, queue)
- Connected components

Directed Graphs

- Precendence-constrained Scheduling
- Topological Sort

Minimum Spanning Trees

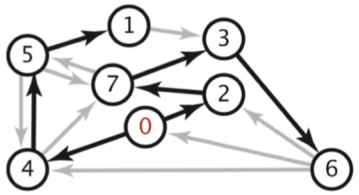
- Greedy Algorithm
- Prim's Algorithm

Shortest Paths

- Dijkstra's Algorihm
- A* Algorithm

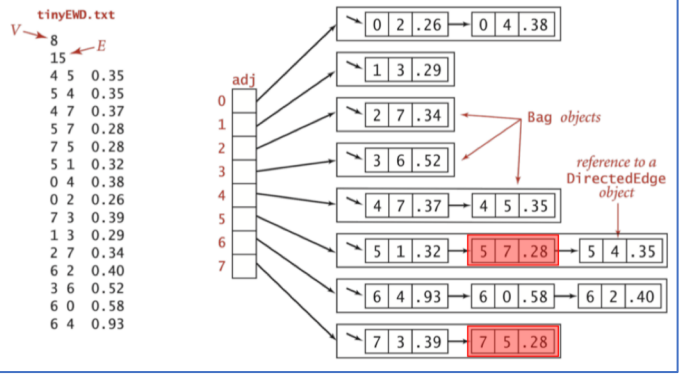
Shortest Path Tree

- Shortest Paths Tree, from computing all single-source paths from 's'.
- Represented by two arrays, `edgeTo[ ]` and `distTo[ ]`.



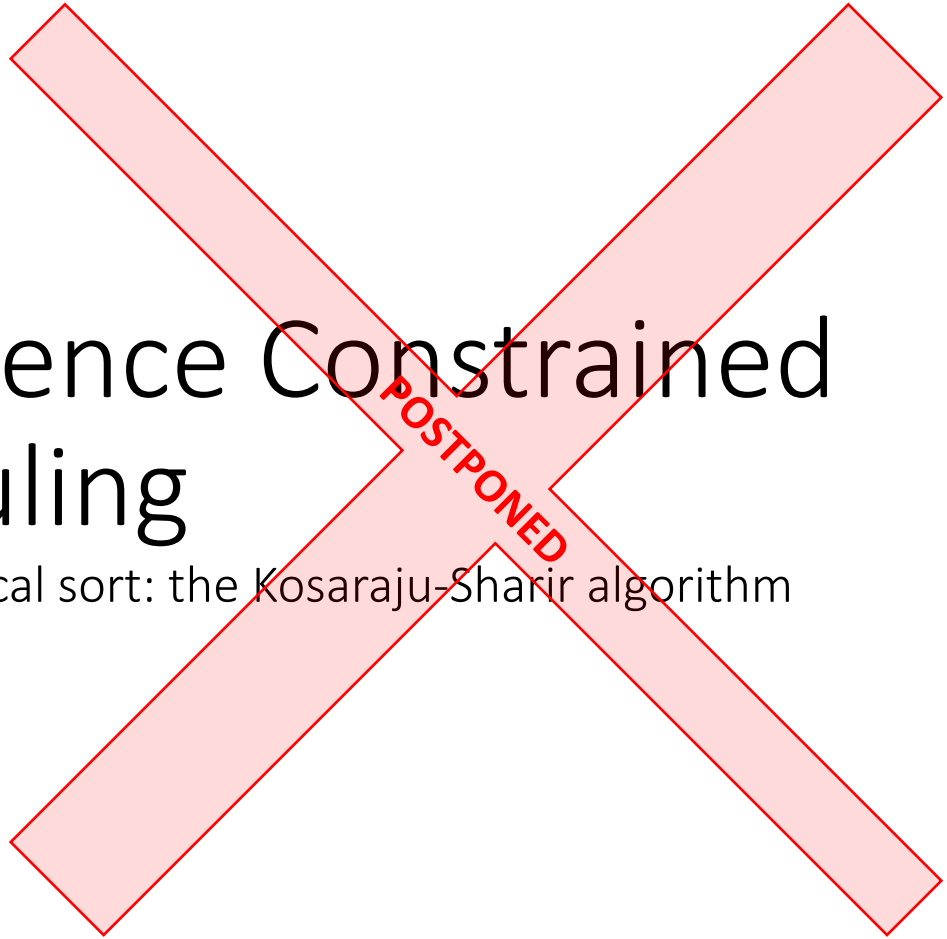
	edgeTo[]	distTo[]
0	null	0
1	5->1 0.32	1.05
2	0->2 0.26	0.26
3	7->3 0.37	0.97
4	0->4 0.38	0.38
5	4->5 0.35	0.73
6	3->6 0.52	1.49
7	2->7 0.34	0.60

What Data Structure for Dijkstra *et al.*?



Precedence Constrained Scheduling

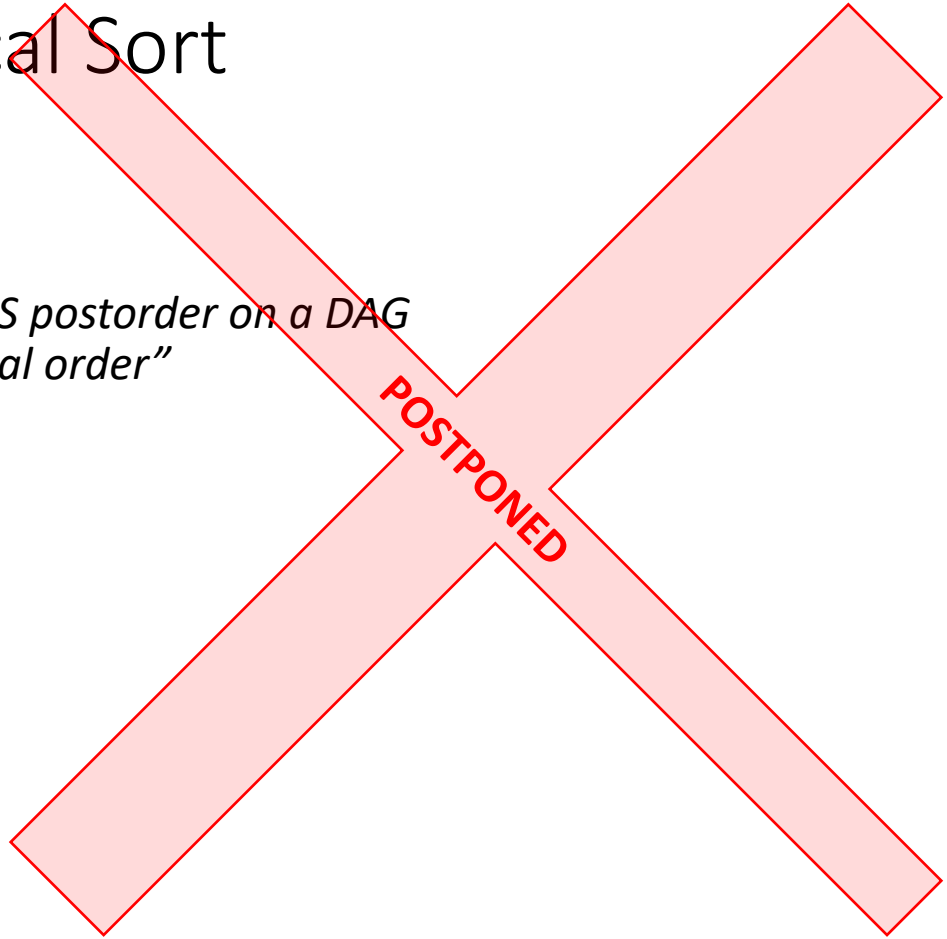
... and topological sort: the Kosaraju-Sharir algorithm



Topological Sort

- Theorem:

  "A reverse DFS postorder on a DAG is a topological order"



Topological Sort on DAGs

General problem of solving order of precedence for any system.

Must have a Directed Acyclic Graph (DAG), otherwise there is ambivalence.

Redraw all edges so they point in same direction.

Theorem: a reverse DFS postorder stack of a DAG is a topological order.

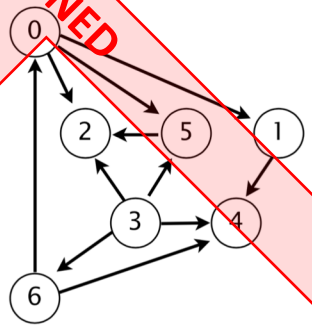
EXERCISE

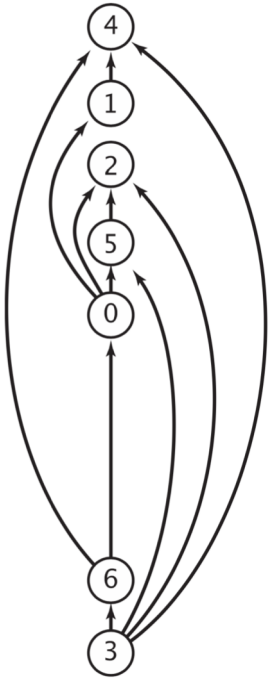
- **show by demo'ing.....on the white board.**
- **does the order of visits matter?**

POSTPONED

Example of use of Top.Sort. on DAGs is order of precedence scheduling.

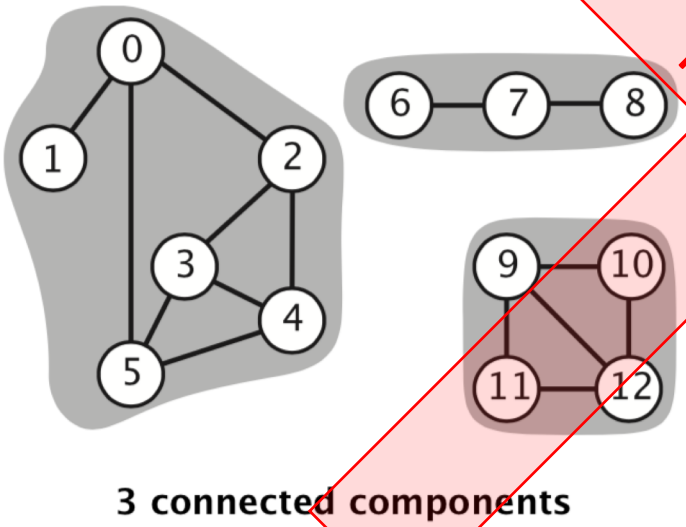
0. Algorithms
1. Complexity Theory
2. Artificial Intelligence
3. Intro to CS
4. Cryptography
5. Scientific Computing
6. Advanced Programming





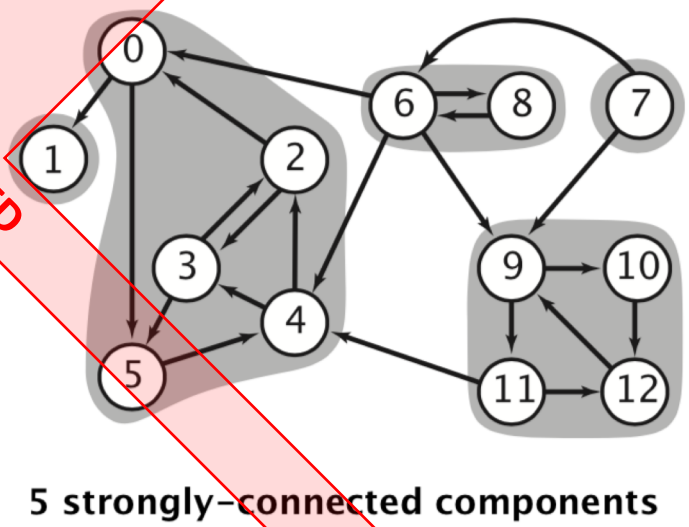
Strong connectivity in digraphs

Un-directed graphs



3 connected components

Digraphs



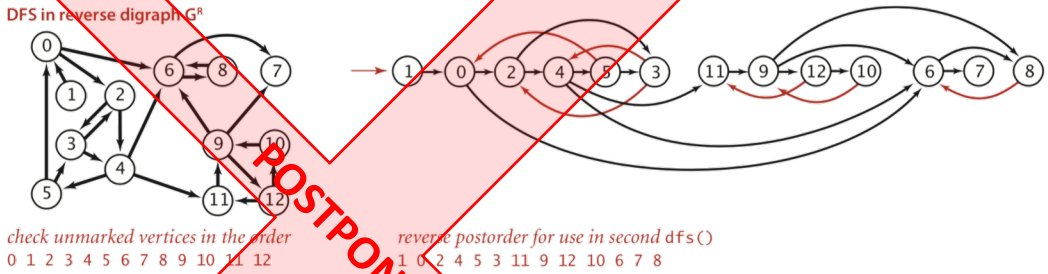
5 strongly-connected components

POSTPONED

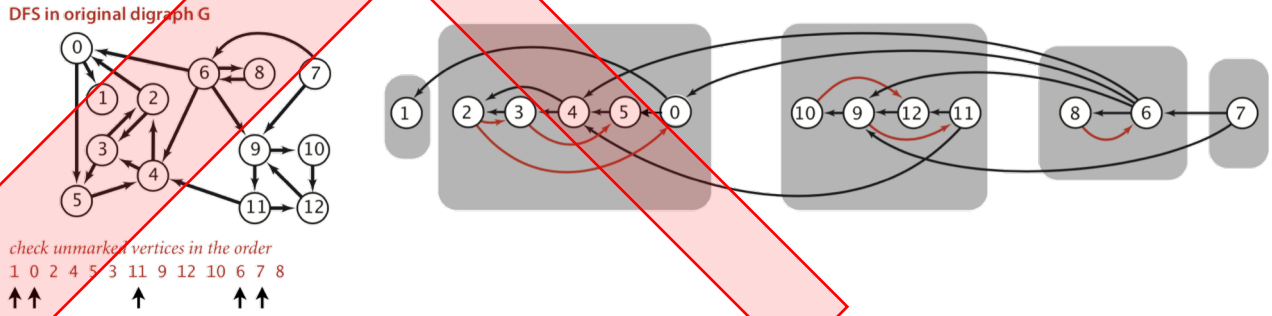
Kosaraju-Sharir algorithm: compute strong components using twice DFS.

POSTPONED

Step 1



Step 2



cphbusiness
COPENHAGEN BUSINESS ACADEMY

Acknowledgement

- Some material has been taken from the Princeton course 'Algorithms 4th Edition', by Sedgewick & Wayne.