University of BRISTOL | Department of Computer Science

PROGRAMMING and ALGORITHMS II

INTRODUCTION TO

Greedy Algorithms

Dr Tilo Burghardt

Unit Code COMS10001

RECAP:
Algorithm Design Paradigm 1

DIVIDE & CONQUER (D&C)

Concept: Recursively *break down* a problem *into independent sub-problems* of related type and smaller size until these become simple enough to be solved directly.

Special Case: If only a single, simpler sub-problem is generated per step (e.g. binary search) the paradigm is also known as *`decrease & conquer'*...

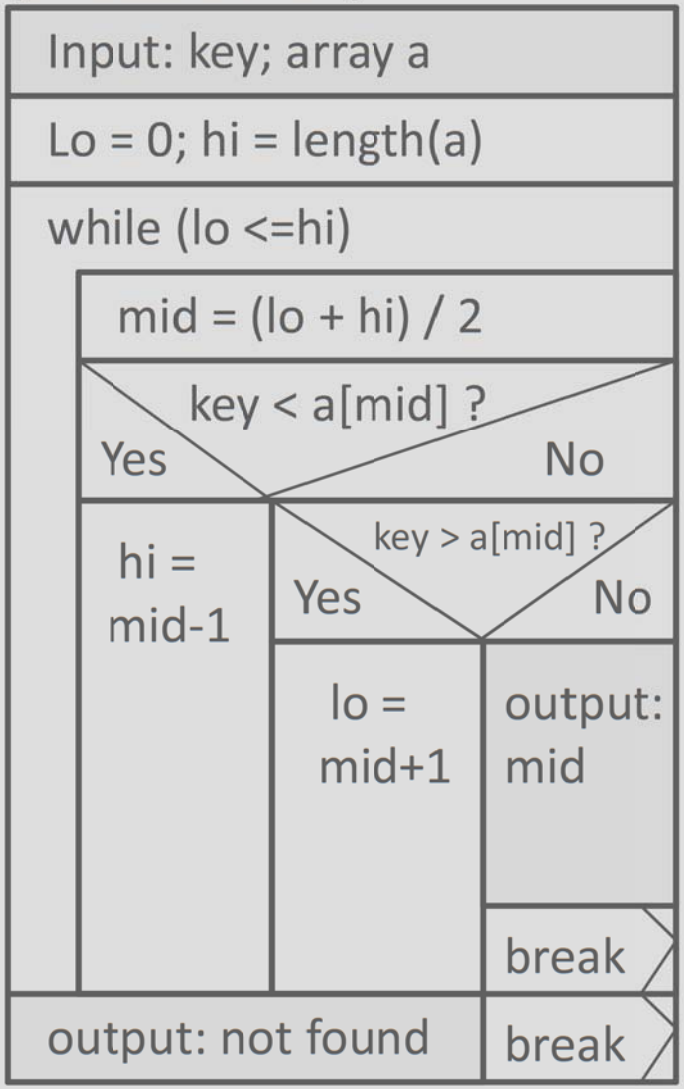
Recap: Simple Binary Search

University of BRISTOL | Department of Computer Science

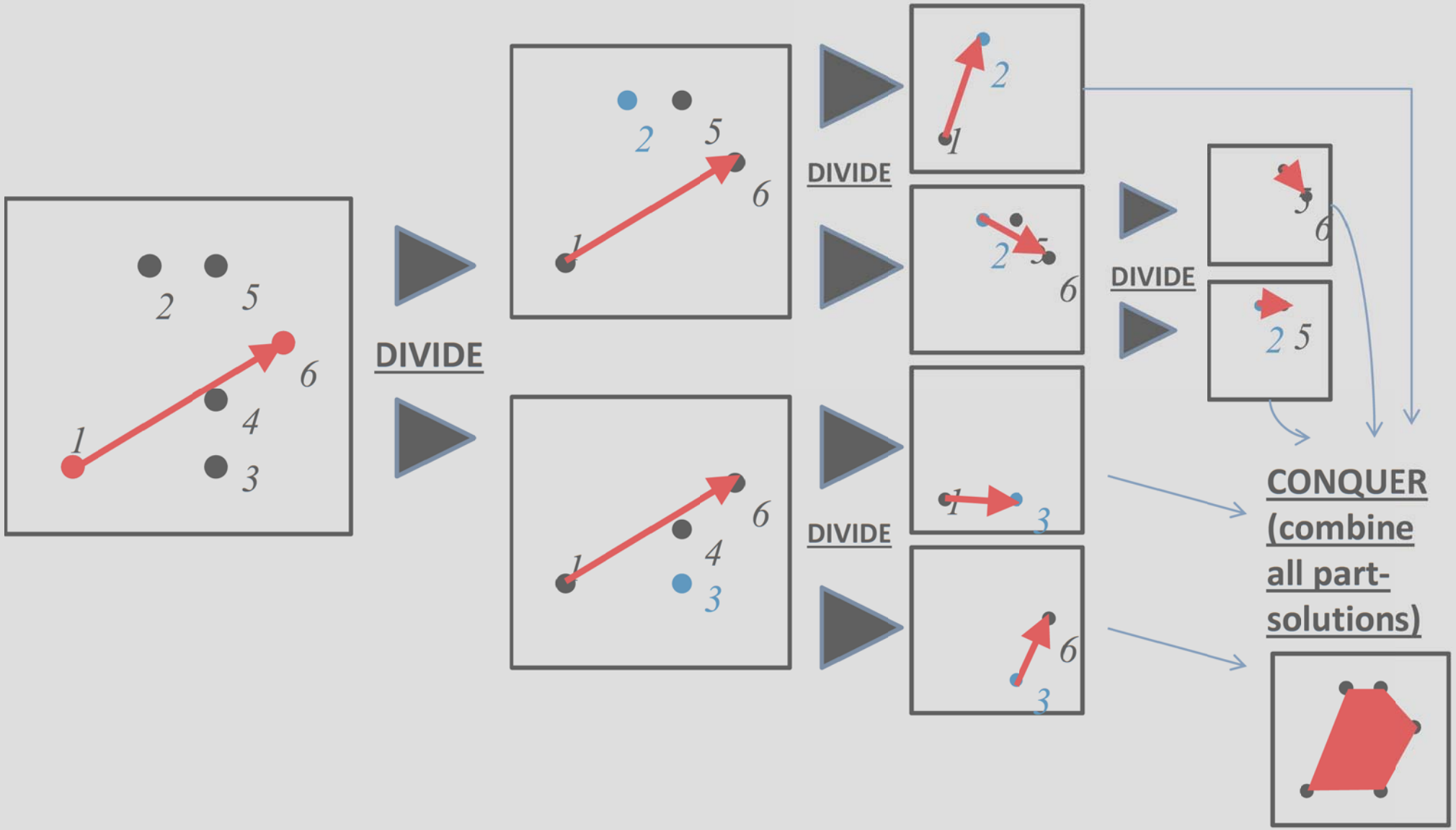
Code Fragment (Java)

```
class Library {
  int[] a; //book keys
  ...
  int find(int key) {
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      //discard upper array part
      if (key < a[mid]) hi = mid - 1;
      //discard lower array part
      else if (key > a[mid]) lo = mid + 1;
      //key found
      else return mid;
    }
    //key not in array
    return -1;
  }...}
```

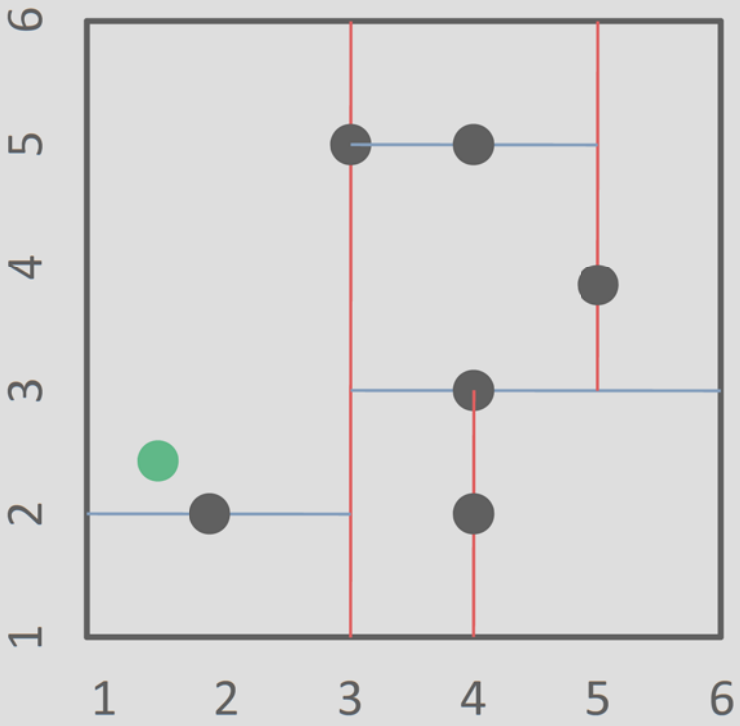
Nassi-Shneiderman-Diagram (structure chart)

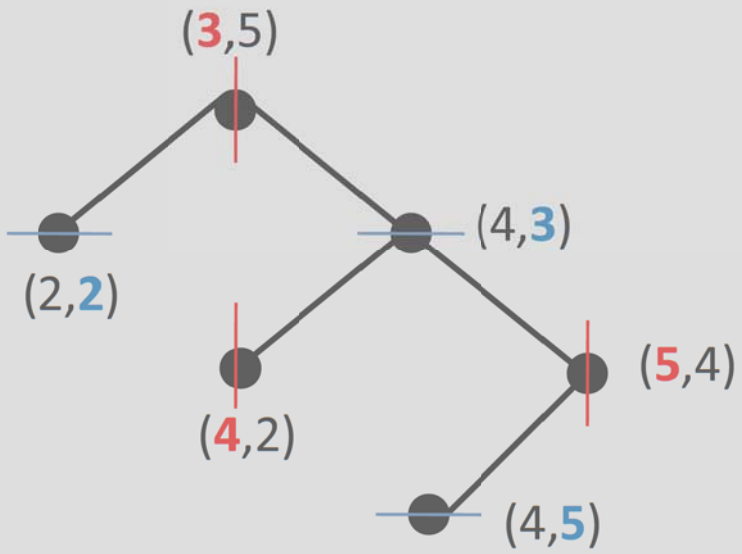


Recap: Quick Hull Algorithm



Task: Find Nearest Neighbour of **Search Node (3.5,3.3)**



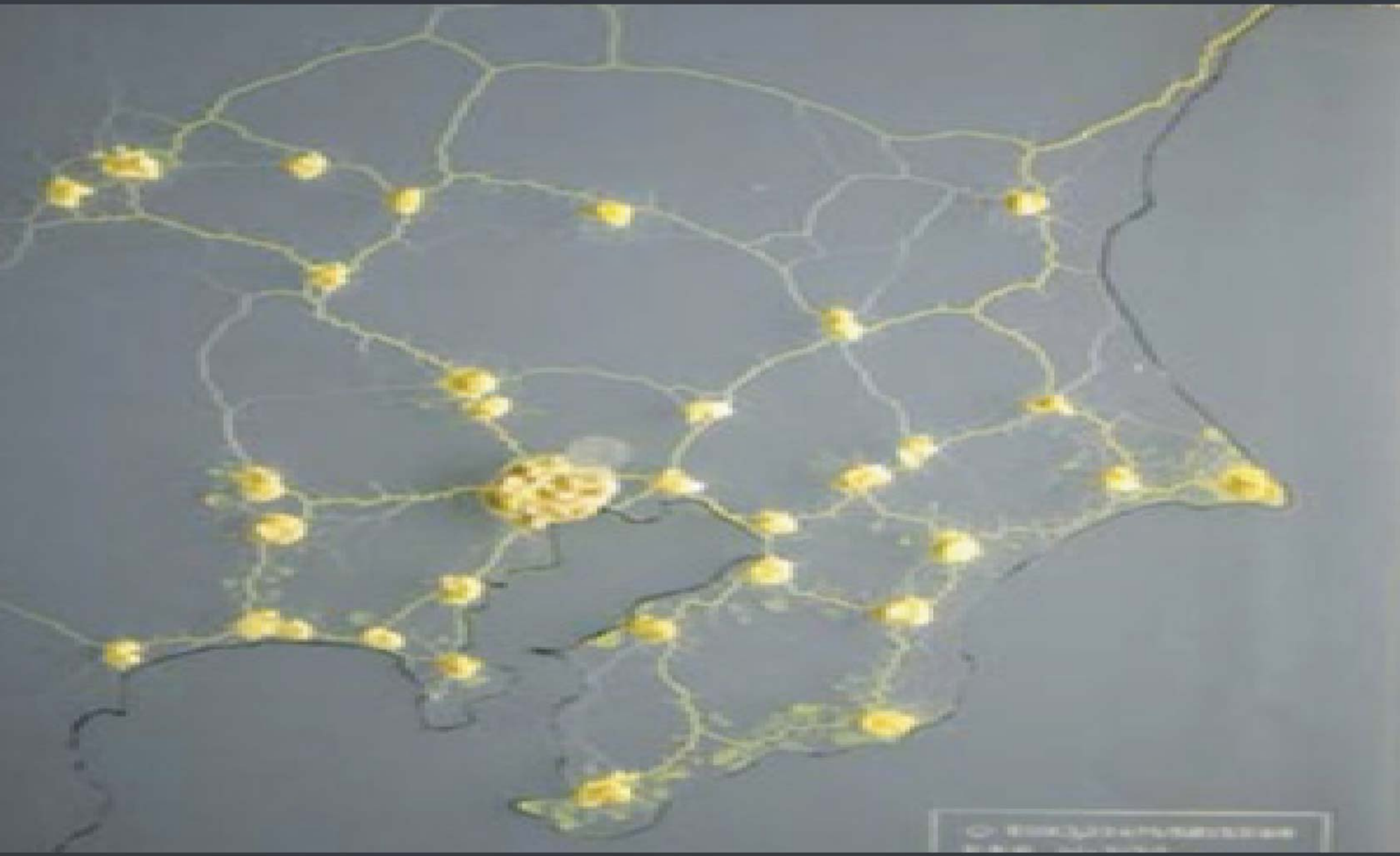


RECAP

DIVIDE and CONQUER in four phases

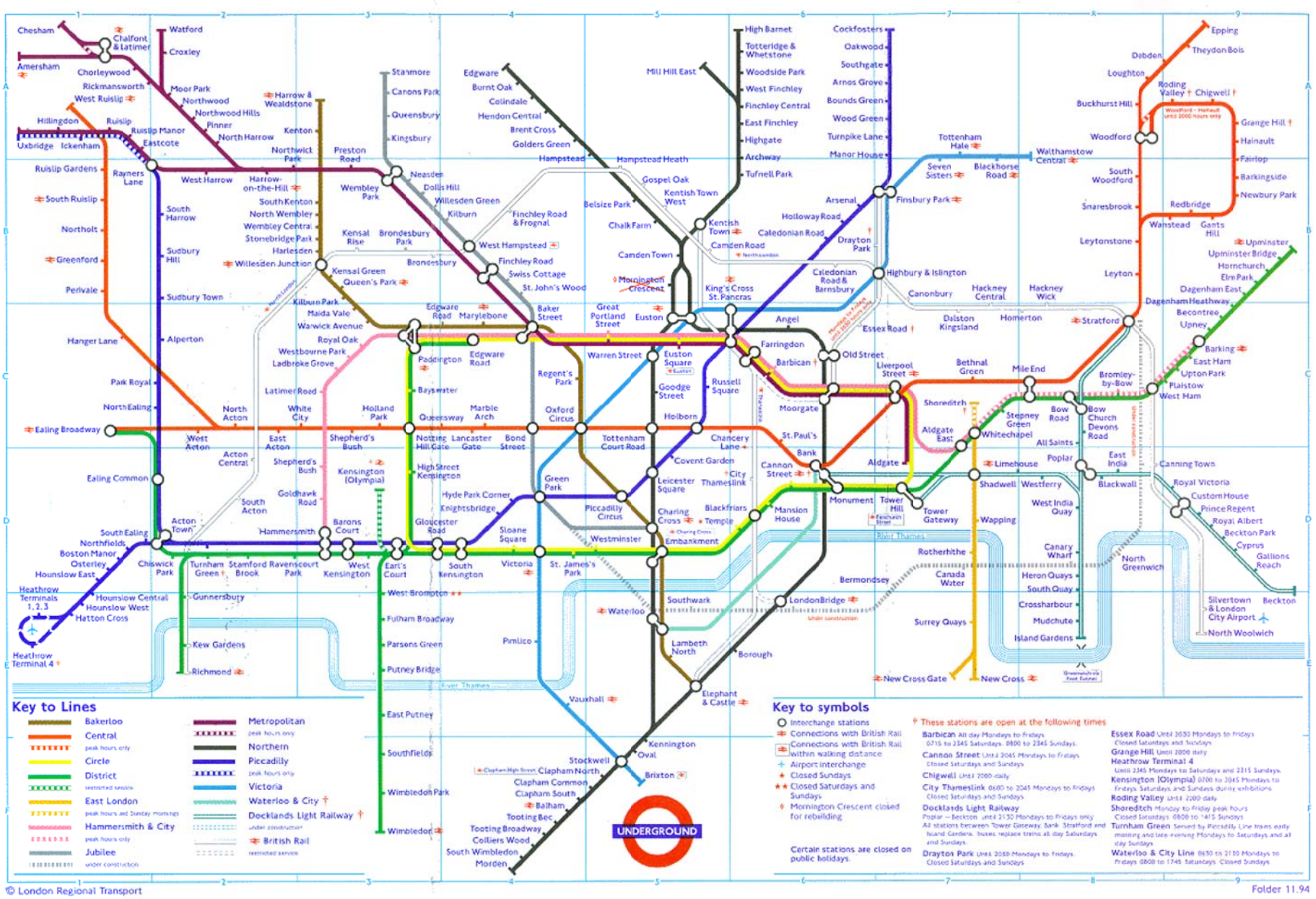
1) **PREPARE**
   Transforming the problem into a form suitable for structured subdivision
2) **DIVIDE**
   Recursively breaking the problem into sub-problems that are similar to the original problem but smaller in size,
3) **SOLVE**
   Compute solutions for the sub-problems successively and independently,
4) **CONQUER**
   Combine these solutions to create one solution to the original, larger problem.

Slime Mould forms Graph



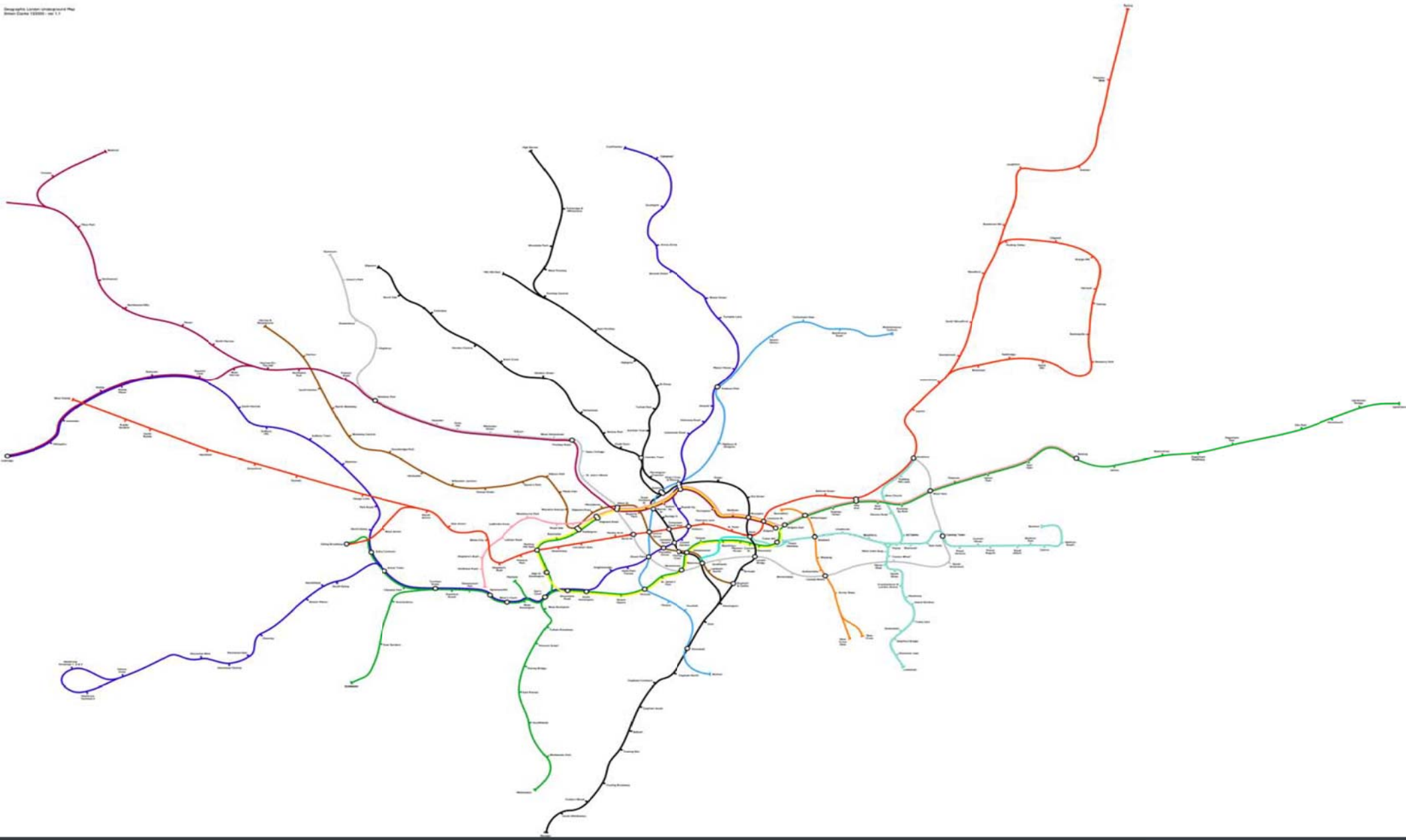
Well known London Tube Map

University of BRISTOL | Department of Computer Science



True Layout of London's Tube

University of BRISTOL | Department of Computer Science

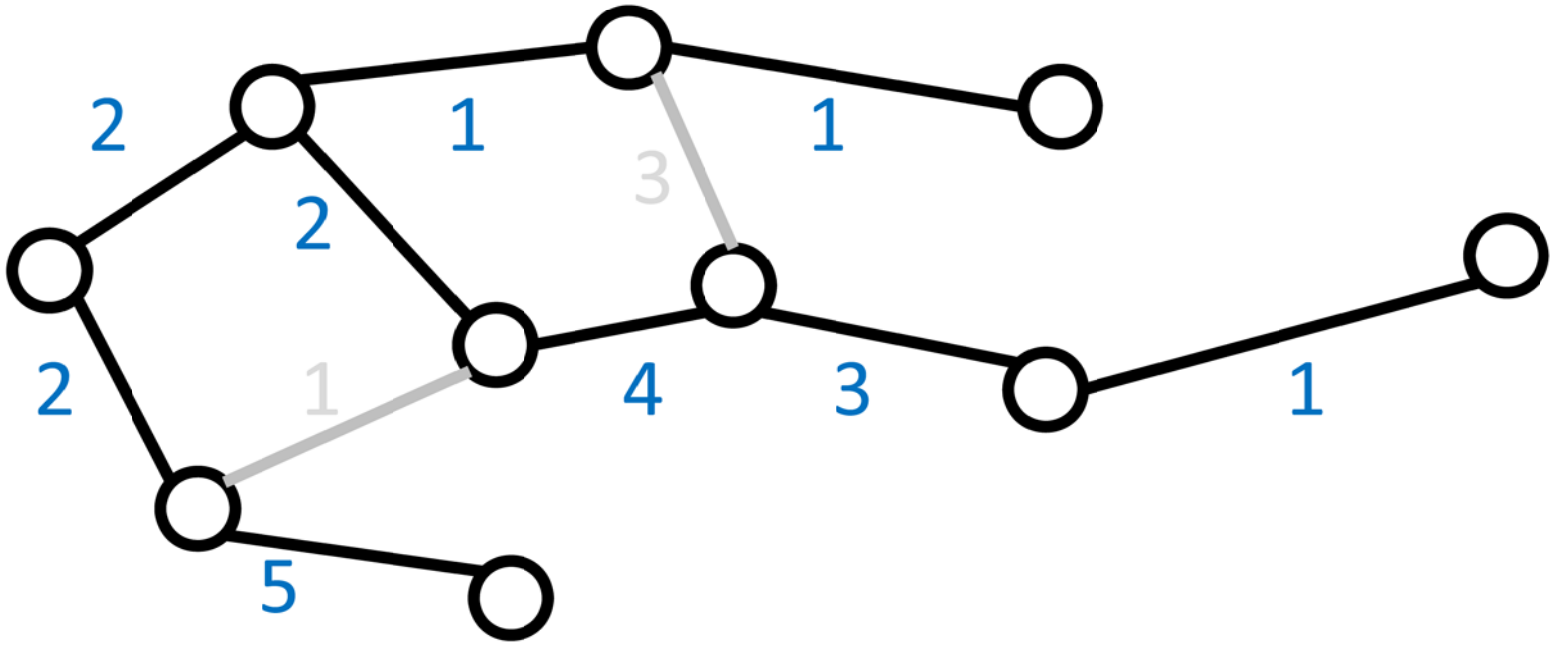


Blood Vessels Spanning an Area

Spanning Trees of a Graph

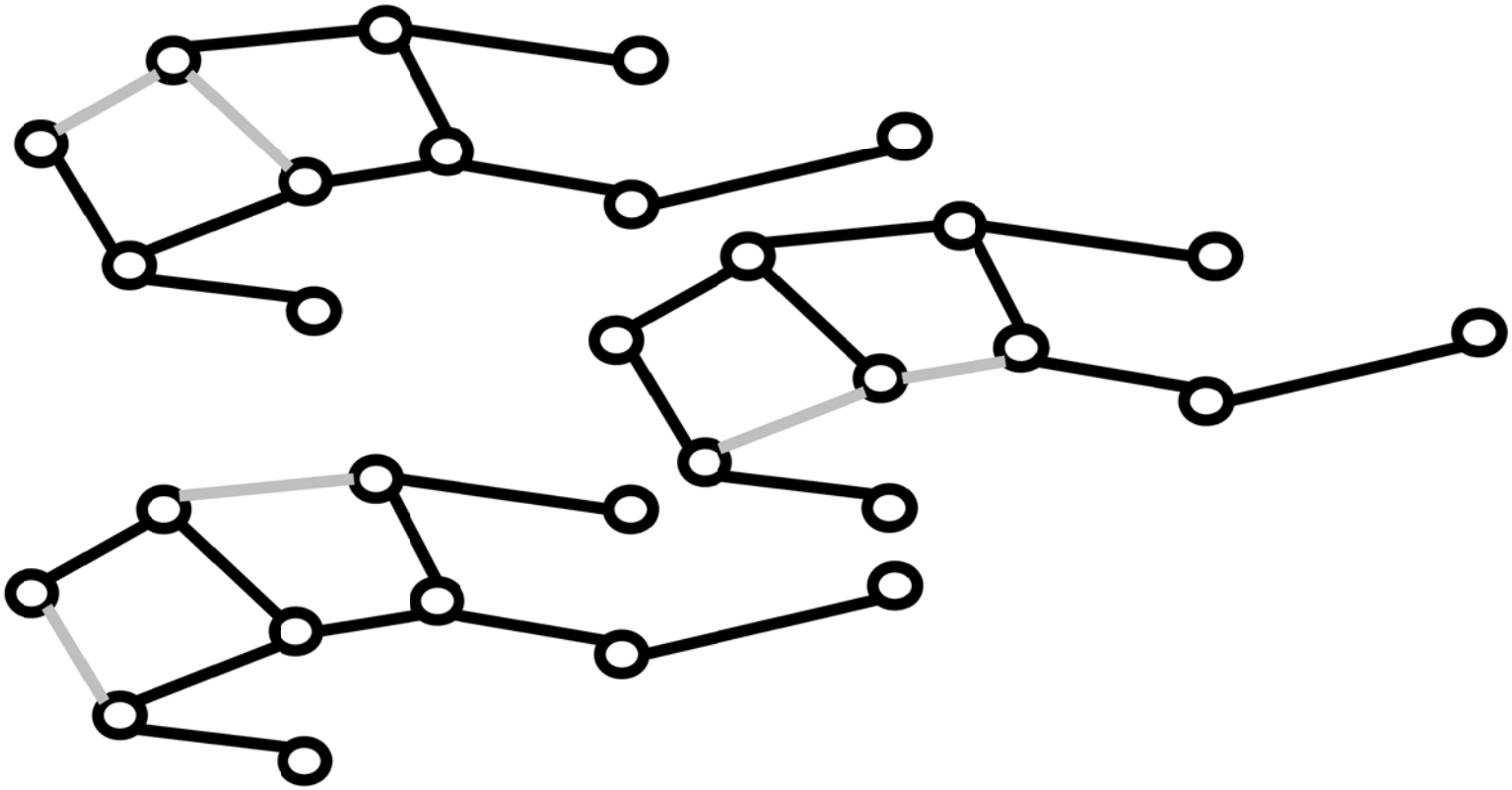
Given a connected, undirected weighted graph G, a connected subgraph T is a ***spanning tree*** if:

- T is a tree (i.e. a cycle-free graph)
- T and G have the same vertex set (i.e. T covers G)



Some Spanning Tree Properties

- spanning trees are not unique
- number of edges = number of vertices-1
- their structure may be very unbalanced

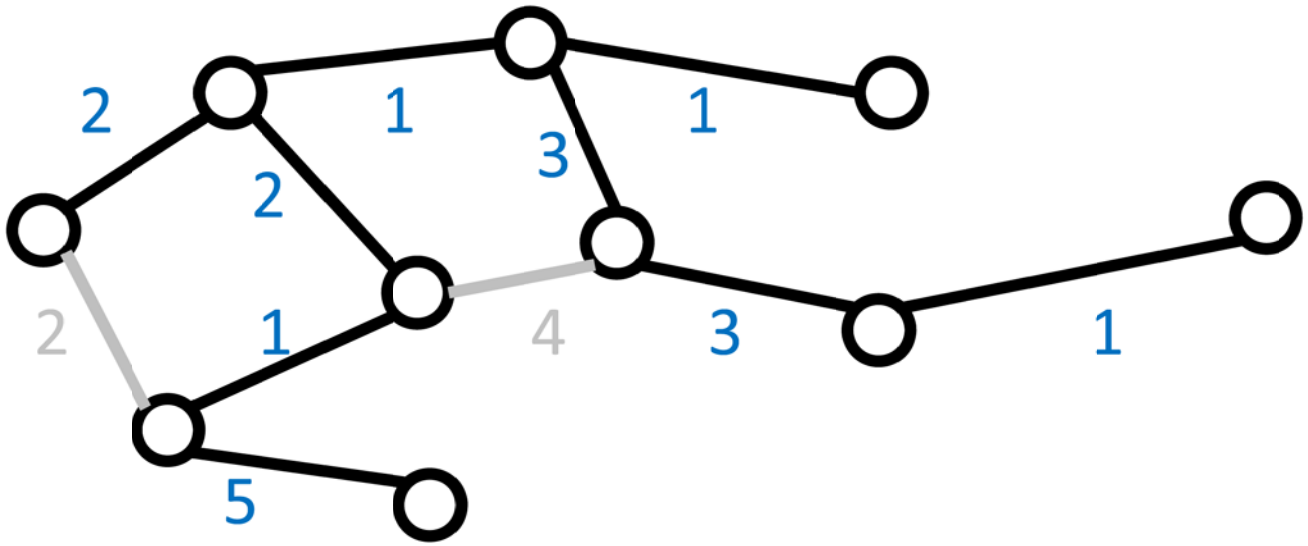


Minimal Spanning Trees

Given a connected, undirected weighted graph G, a subgraph T is a ***spanning tree*** if:

- T is a tree (i.e. a cycle-free graph)
- T and G have the same vertex set

Amongst all spanning trees of a graph G, a **minimum spanning tree (MST)** is a tree T for which the sum of the weights of its edges is minimal.



Algorithm Design Paradigm 2

GREEDY APPROACH

Concept: Solving a problem by a *sequence of locally optimal decisions* where per step only a set of immediately available options is considered.

Prim's Algorithm

$G=(V, E)$ ... graph with weights $d(v,w) \geq 0$ for all $v, w \in V$
(where $d(v,w) = \infty$ if $(v,w) \notin E$)

$W \subseteq V$ ... set of visited nodes

$F \subseteq E$ ... set of utilised edges for current tree

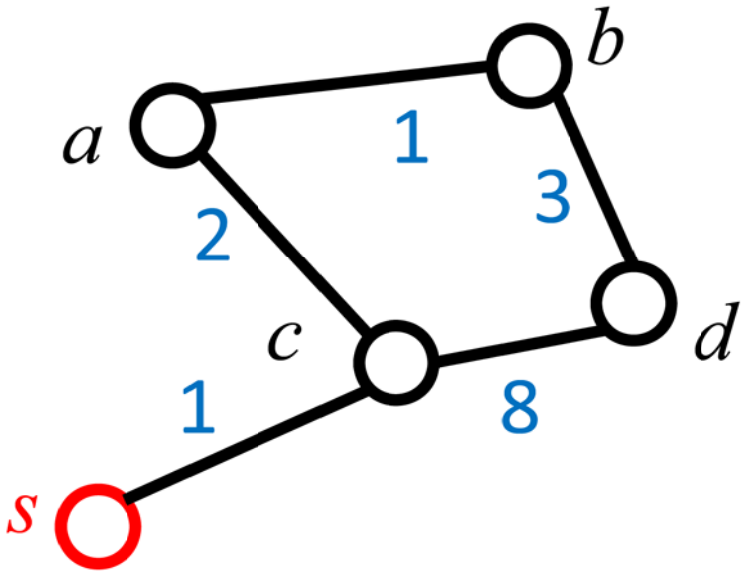
$s \in V$ ... source vertex to start tree

$D(v)$ ... immediate distance from current tree to v

---

1) Initialise: $W = \{s\}; F = \{\}; \forall_{v \in V} : D(v) = d(s,v)$
2) Select a new **current vertex *w*** in $V \backslash W$ with minimal $D(w)$
3) Add current vertex ***w*** to W, add related edge to F
4) Update distances: $\forall_{v \in V \backslash W} : D(v) = \min(D(v), d(w,v))$
5) If $V=W$ then exit, otherwise Goto 2)

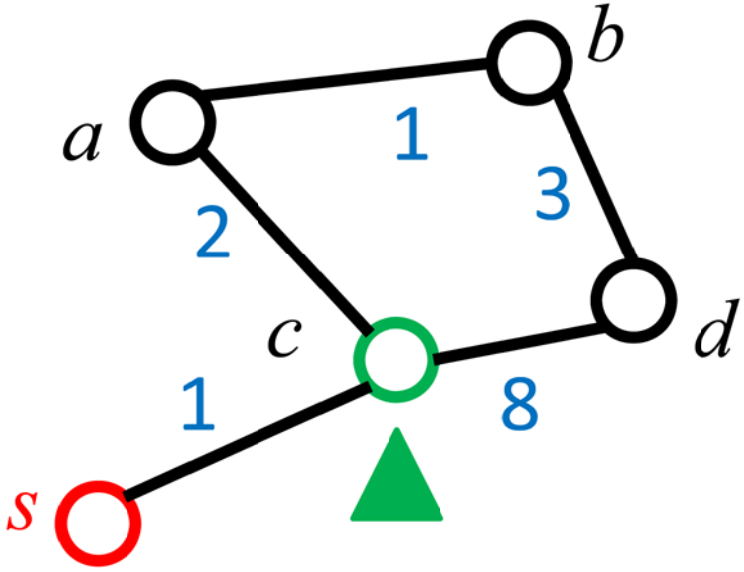
Prim's Algorithm: Example



$$F = \{\}$$
$$W = \{s\}$$
$$D(a) = \infty$$
$$D(b) = \infty$$
$$D(c) = d(s,c) = 1$$
$$D(d) = \infty$$
$$D(s) = 0$$

Initialise $W = \{s\}; F = \{\}; \forall_{v \in V} : D(v) = d(s,v)$

Prm's Algorithm: Example



$F = \{\}$

$W = \{s\}$

$D(a) = \infty$

$D(b) = \infty$

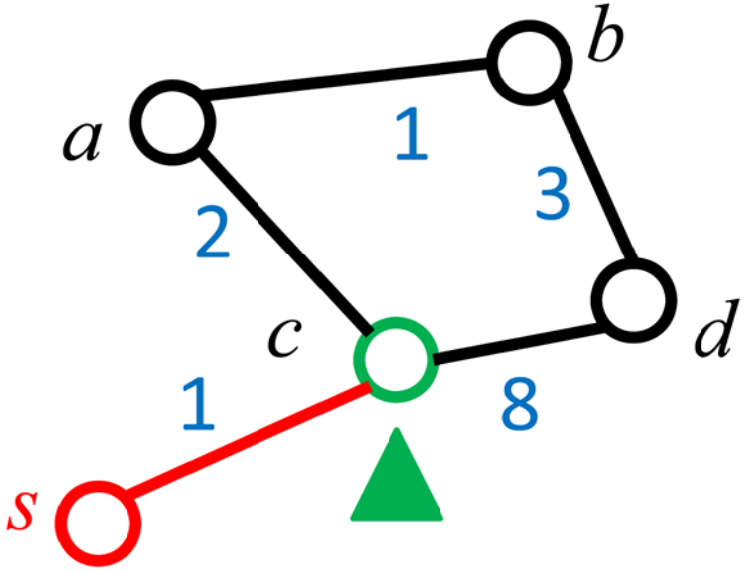
$D(c) = 1$

$D(d) = \infty$

$D(s) = 0$

Select a new **current vertex** in $V \backslash W$ with minimal $D(w)$ → c

Prim's Algorithm: Example



$F = \{(s,c)\}$

$W = \{s,c\}$

$D(a) = \infty$

$D(b) = \infty$

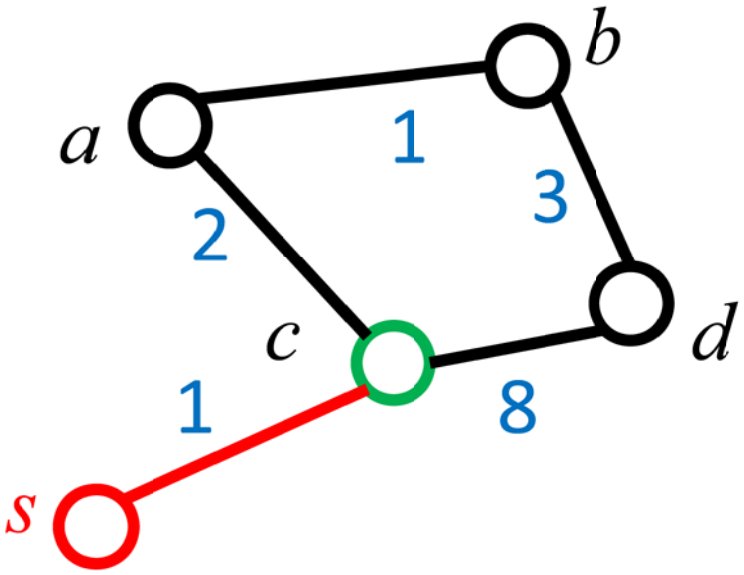
$D(c) = 1$

$D(d) = \infty$

$D(s) = 0$

Add current vertex c to W, add related edge to F

Prim's Algorithm: Example



$F = \{(s,c)\}$

$W = \{s,c\}$

$D(a) = \min(\infty, d(c,a))$

$D(b) = \min(\infty, d(c,b))$

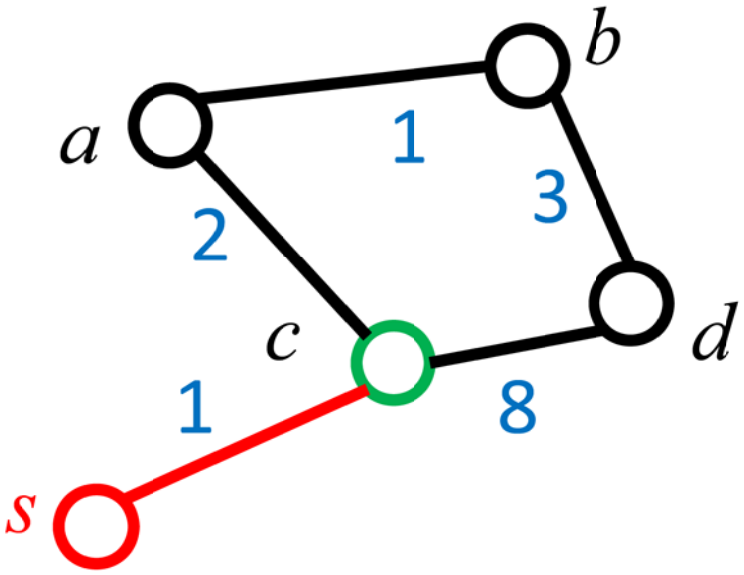
$D(c) = 1$

$D(d) = \min(\infty, d(c,d))$

$D(s) = 0$

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), d(c,v))$

Prim's Algorithm: Example



$F = \{(s,c)\}$

$W = \{s,c\}$

$D(a) = \min(\infty, 2) = 2$

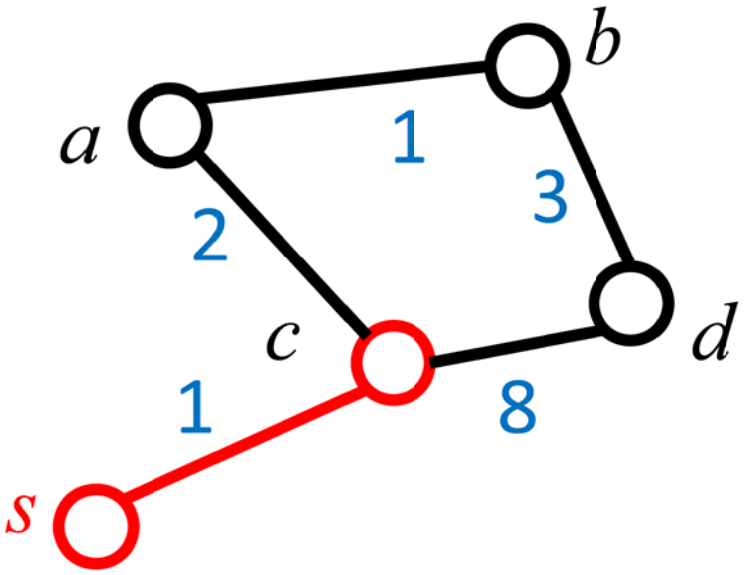
$D(b) = \min(\infty, \infty) = \infty$

$D(c) = 1$

$D(d) = \min(\infty, 8) = 8$

$D(s) = 0$

Prim's Algorithm: Example



$F = \{(s,c)\}$

$W = \{s,c\}$

$D(a) = 2$

$D(b) = \infty$

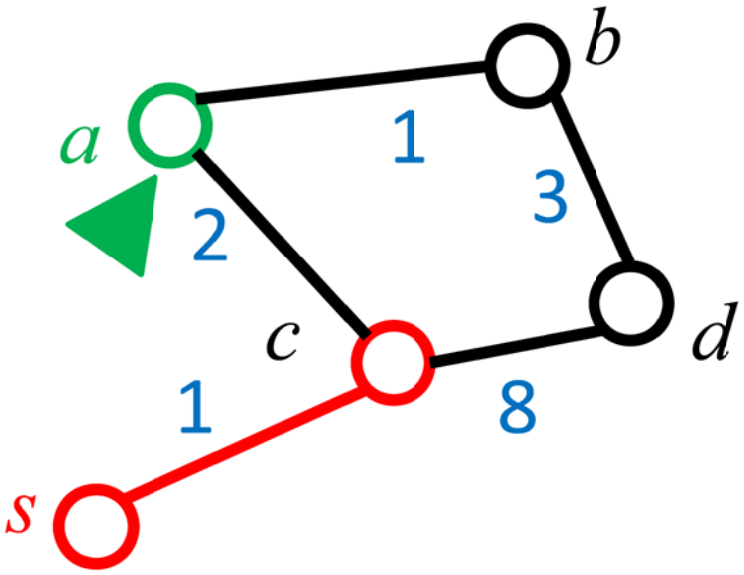
$D(c) = 1$

$D(d) = 8$

$D(s) = 0$

Exit if $V=W$? → no

Prim's Algorithm: Example



$F = \{(s,c)\}$

$W = \{s,c\}$

$D(a) = 2$

$D(b) = \infty$

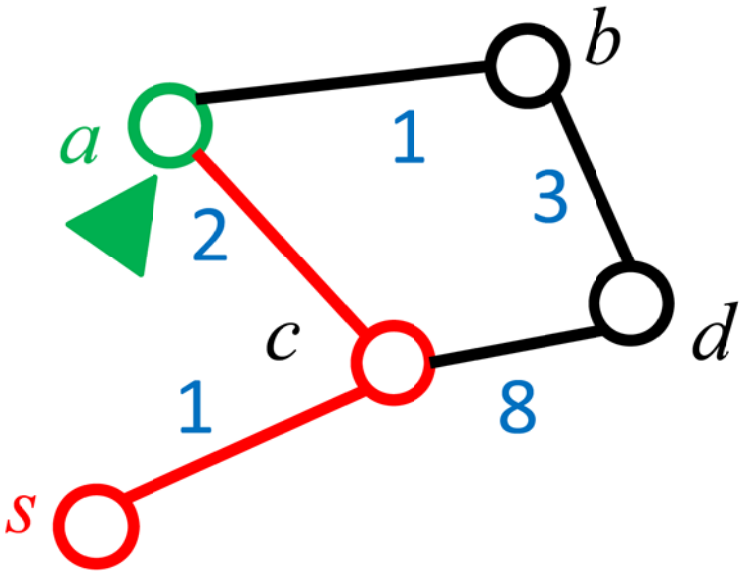
$D(c) = 1$

$D(d) = 8$

$D(s) = 0$

Select a new **current vertex** in $V \backslash W$ with minimal $D(w)$ → a

Prim's Algorithm: Example



$F = \{(s,c),(c,a)\}$

$W = \{s,c,a\}$

$D(a) = 2$

$D(b) = \infty$

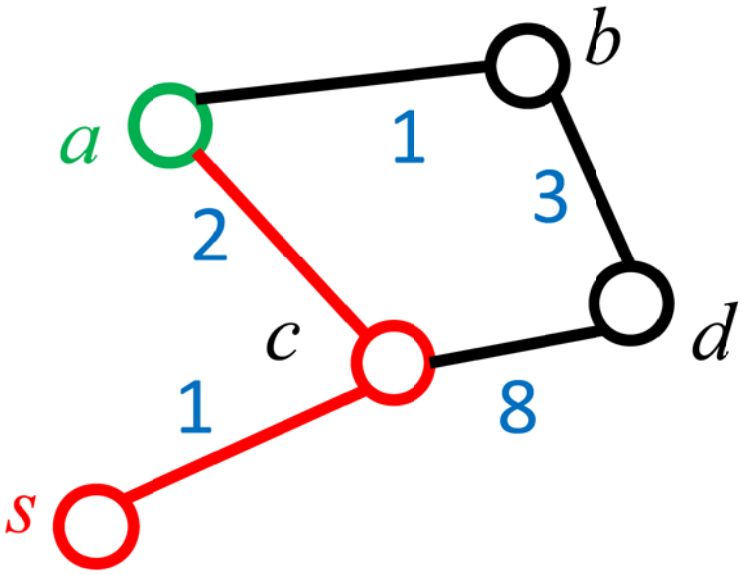
$D(c) = 1$

$D(d) = 8$

$D(s) = 0$

Add current vertex a to W, add related edge to F

Prim's Algorithm: Example



$F = \{(s,c),(c,a)\}$

$W = \{s,c,a\}$

$D(a) = 2$

$D(b) = \min(\infty, d(a,b)) = 1$

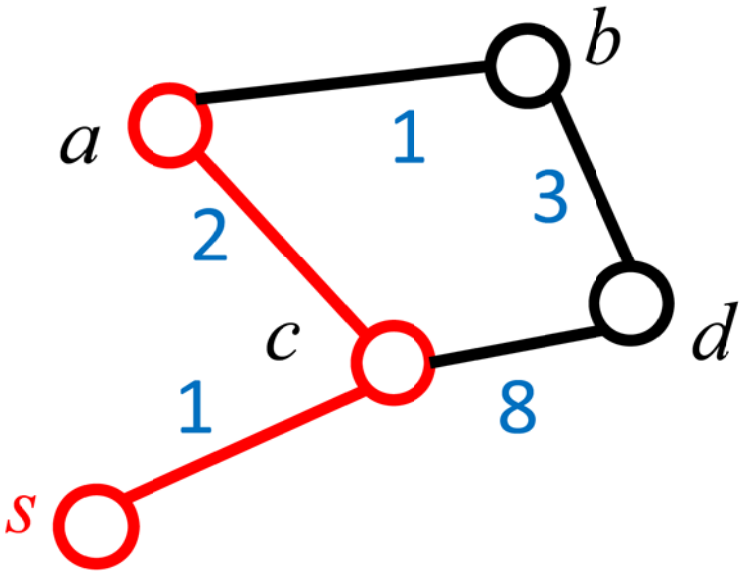
$D(c) = 1$

$D(d) = 8$

$D(s) = 0$

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), d(a,v))$

Prim's Algorithm: Example



$F = \{(s,c),(c,a)\}$

$W = \{s,c,a\}$

$D(a) = 2$

$D(b) = 1$

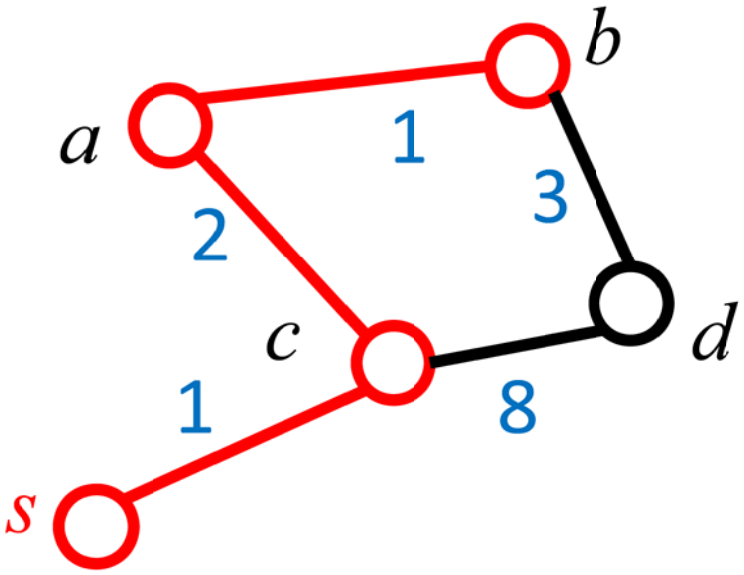
$D(c) = 1$

$D(d) = 8$

$D(s) = 0$

Exit if $V=W$? → no

Prim's Algorithm: Example



$F = \{(s,c),(c,a),(a,b)\}$

$W = \{s,c,a,b\}$

$D(a) = 2$

$D(b) = 1$

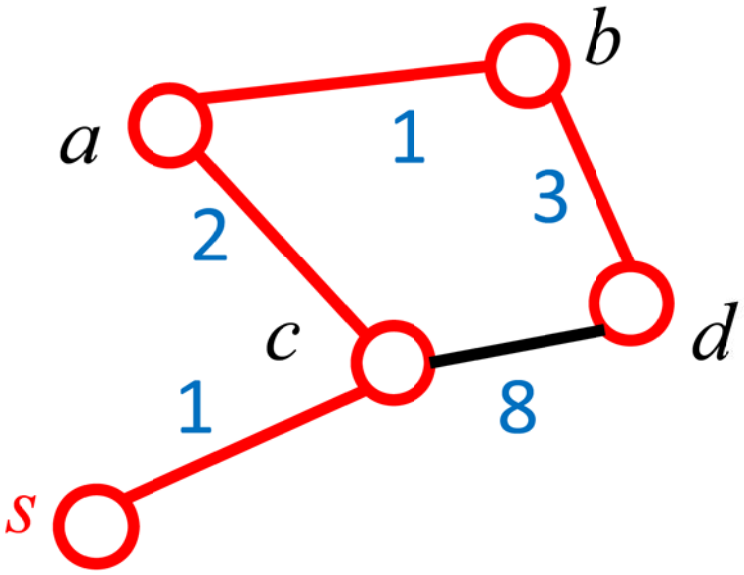
$D(c) = 1$

$D(d) = 3$

$D(s) = 0$

Exit if $V=W$? → no

Prim's Algorithm: Example



$F = \{(s,c),(c,a),(a,b),(b,d)\}$

$W = \{s,c,a,b,d\}$

$D(a) = 2$

$D(b) = 1$

$D(c) = 1$

$D(d) = 3$

$D(s) = 0$

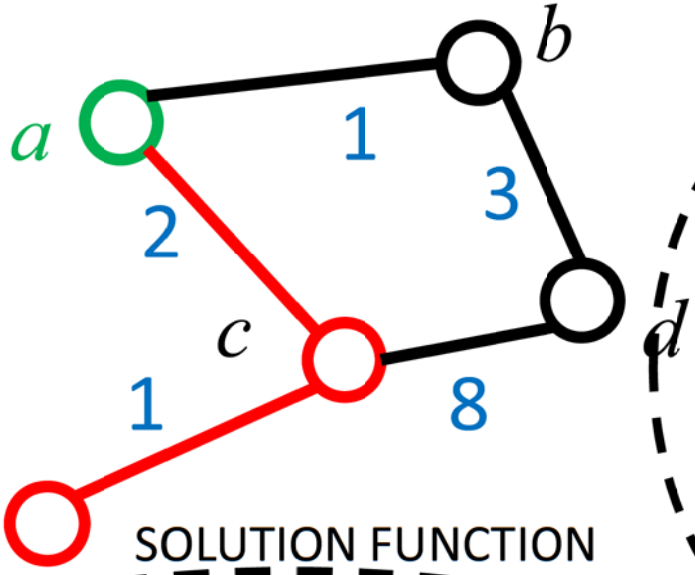
Exit if $V=W$? → YES

GREEDY ALORITHMS in five parts

1) **CANDIDATE SET**
   …from which candidates for a solution are sampled
2) **SOLUTION SET**
   …which holds current elements which contribute to a solution
3) **SELECTION FUNCTION**
   …which chooses current best candidate to potentially add to the solution
4) **FEASIBILITY FUNCTION**
   …that is used to determine if a candidate can be used to contribute to a solution
5) **OBJECTIVE/SOLUTION FUNCTION**
   …which assigns a fitness value to a current solution and determines when the solution is complete

$F = \{(s,c),(c,a)\}$ Evolving SOLUTION SET

$W = \{s,c,a\}$ CANDIDATE SET (INVERSE)



SOLUTION FUNCTION

$D(a) = 2$
$D(b) = 1$
$D(c) = 1$
$D(d) = 8$
$D(s) = 0$

Dynamic SELECTION & FEASIBILITY FUNCTION

Exit if $V = W$?

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), d(a,v))$

Dijkstra's Algorithm

$G=(V, E)$ ... graph with weights $d(v,w) \geq 0$ for all $v, w \in V$
(where $d(v,w) = \infty$ if $(v,w) \notin E$)

$W \subseteq V$ ... set of visited nodes

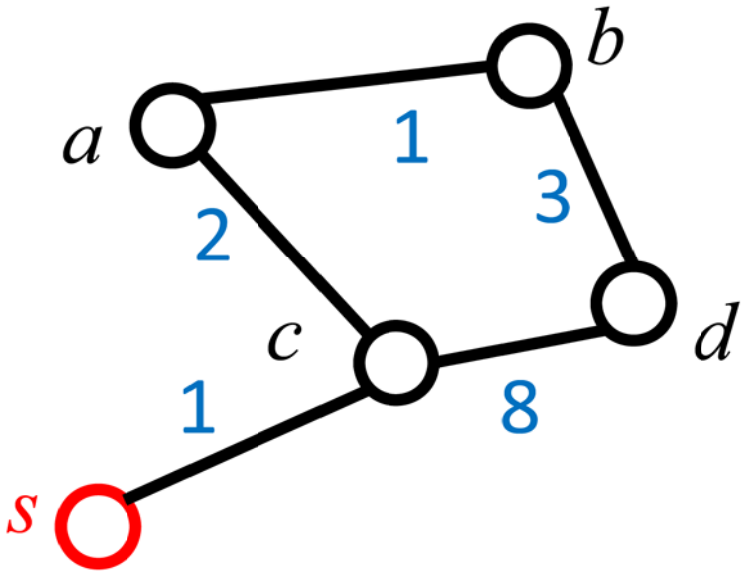
$s \in V$ ... source vertex to calculate distances to

$D(v)$ ... current shortest distance estimate from s to v

---

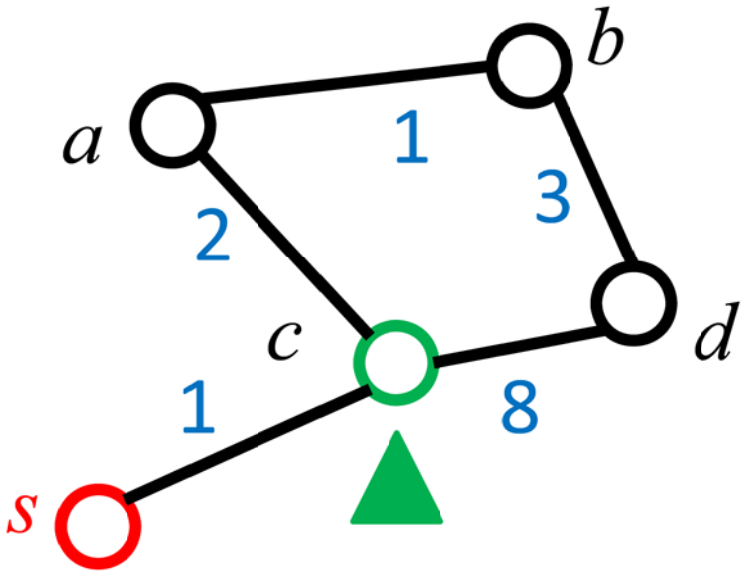
1) Initialise: $W = \{s\}; \forall_{v \in V} : D(v) = d(s,v)$
2) Select a new **current vertex *w*** in $V \backslash W$ with minimal $D(w)$
3) Add current vertex ***w*** to W
4) Update distances: $\forall_{v \in V \backslash W} : D(v) = \min(D(v), D(w) + d(w,v))$
5) If $V=W$ then exit, otherwise Goto 2)

Dijkstra's Algorithm: Example



$$W = \{s\}$$
$$D(a) = \infty$$
$$D(b) = \infty$$
$$D(c) = d(s,c) = 1$$
$$D(d) = \infty$$
$$D(s) = 0$$

Initialise $W = \{s\}; \forall_{v \in V} : D(v) = d(s,v)$



$W = \{s\}$

$D(a) = \infty$

$D(b) = \infty$

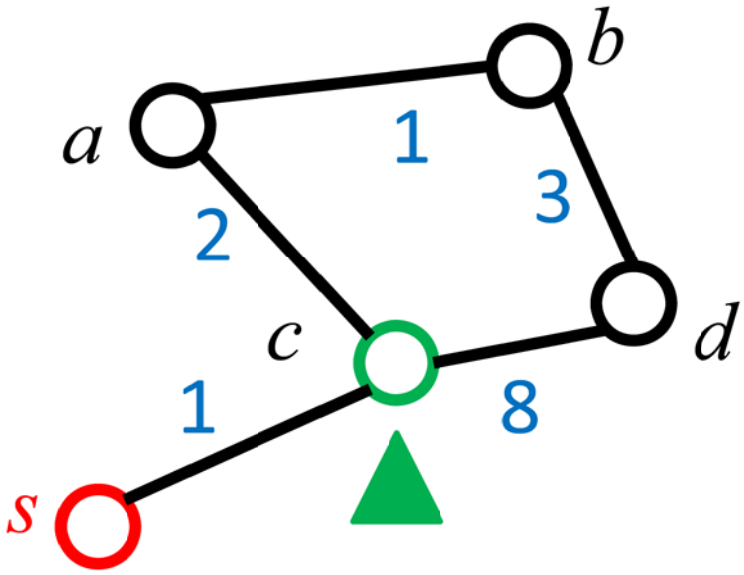
$D(c) = 1$

$D(d) = \infty$

$D(s) = 0$

Select a new **current vertex** in $V \backslash W$ with minimal $D(w)$ → c

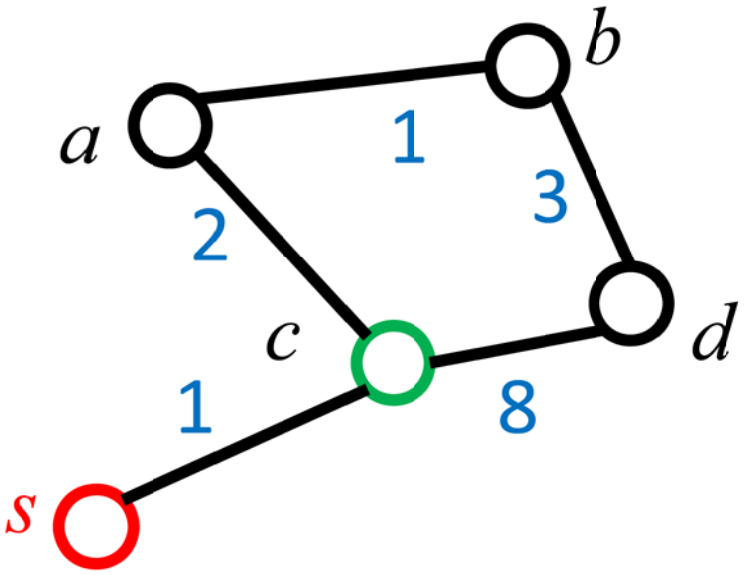
Dijkstra's Algorithm: Example



$$W = \{s, c\}$$
$$D(a) = \infty$$
$$D(b) = \infty$$
$$D(c) = 1$$
$$D(d) = \infty$$
$$D(s) = 0$$

Add current vertex c to W

Dijkstra's Algorithm: Example



$$W = \{s, c\}$$

$$D(a) = \min(\infty, D(c)+d(c,a))$$

$$D(b) = \min(\infty, D(c)+d(c,b))$$

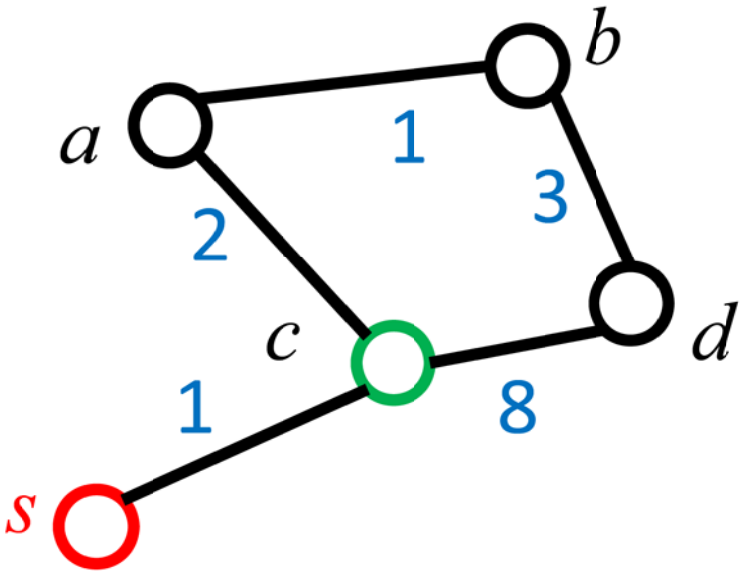
$$D(c) = 1$$

$$D(d) = \min(\infty, D(c)+d(c,d))$$

$$D(s) = 0$$

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), D(c) + d(c,v))$

Dijkstra's Algorithm: Example



$$W = \{s, c\}$$

$$D(a) = \min(\infty, 1+2)$$

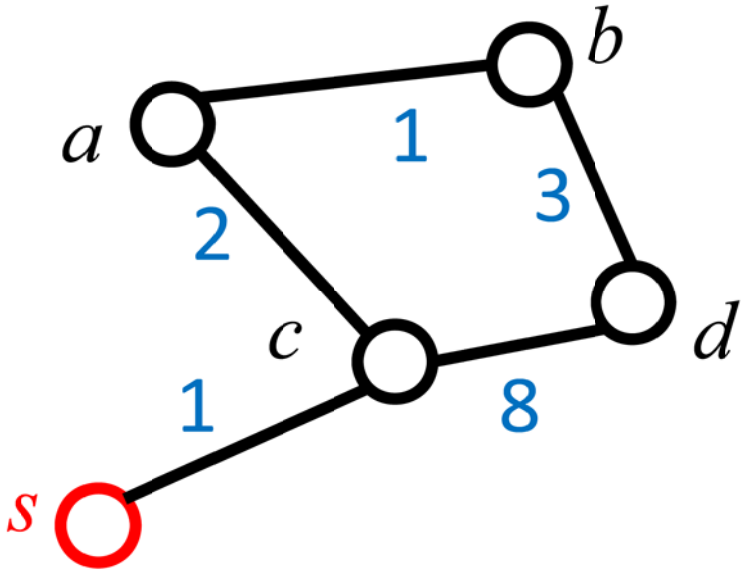
$$D(b) = \min(\infty, 1+\infty)$$

$$D(c) = 1$$

$$D(d) = \min(\infty, 1+8)$$

$$D(s) = 0$$

Dijkstra's Algorithm: Example



$W = \{s, c\}$

$D(a) = 3$

$D(b) = \infty$

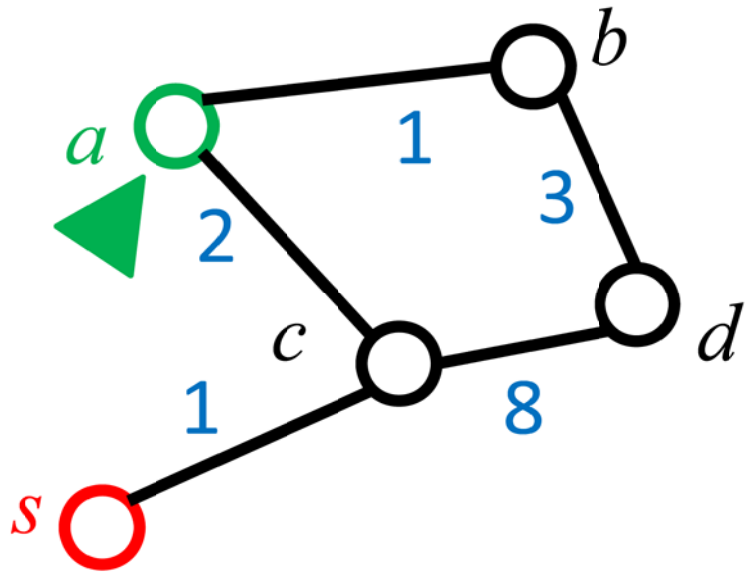
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Exit if $V = W$? → no

Dijkstra's Algorithm: Example



$W = \{s, c\}$

$D(a) = 3$

$D(b) = \infty$

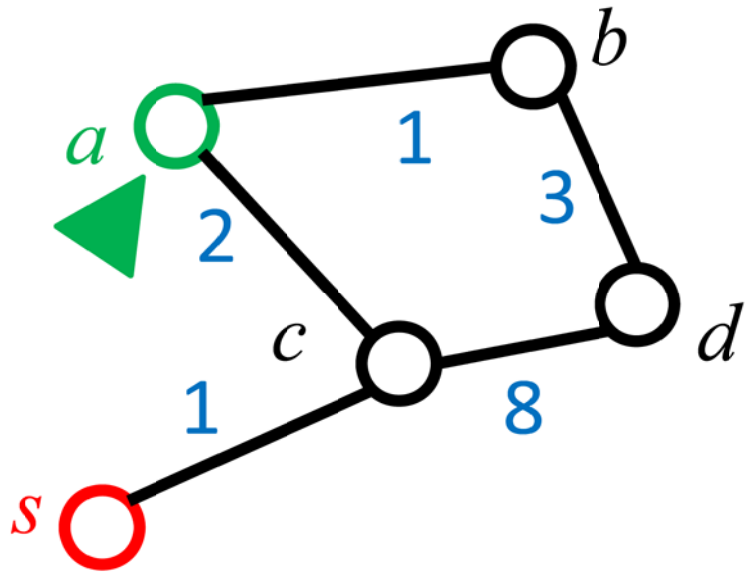
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Select a vertex in $V \backslash W$ with minimal $D(w)$ → select ***a***

Dijkstra's Algorithm: Example



$W = \{s, c, a\}$

$D(a) = 3$

$D(b) = \infty$

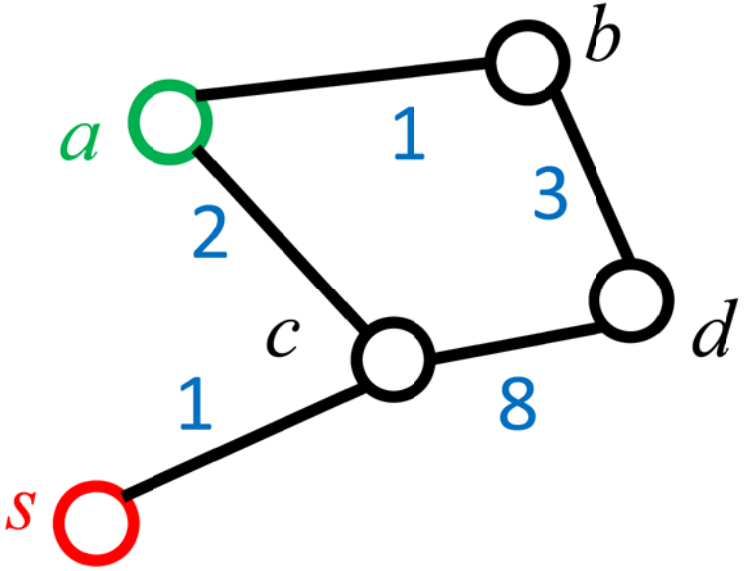
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Add current vertex a to W

Dijkstra's Algorithm: Example



$W = \{s, c, a\}$

$D(a) = 3$

$D(b) = \min(\infty, D(a)+d(a,b))$

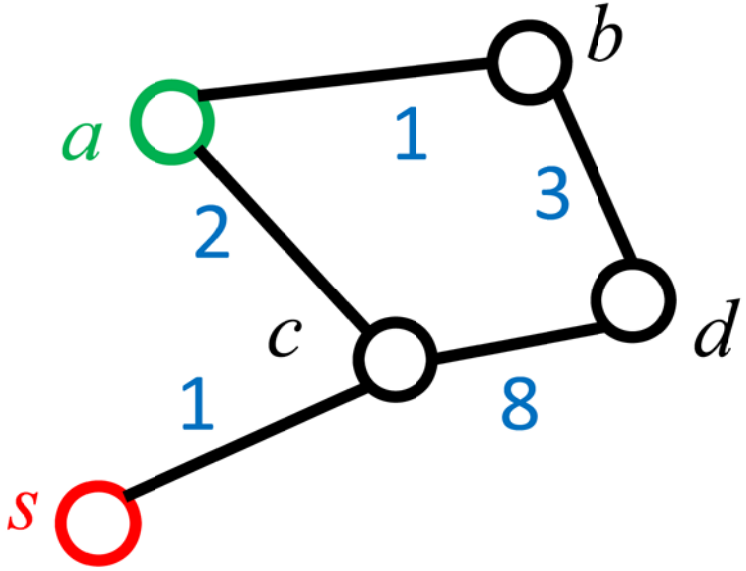
$D(c) = 1$

$D(d) = \min(9, D(a)+d(a,d))$

$D(s) = 0$

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), D(a) + d(a,v))$

Dijkstra's Algorithm: Example



$$W = \{s, c, a\}$$

$$D(a) = 3$$

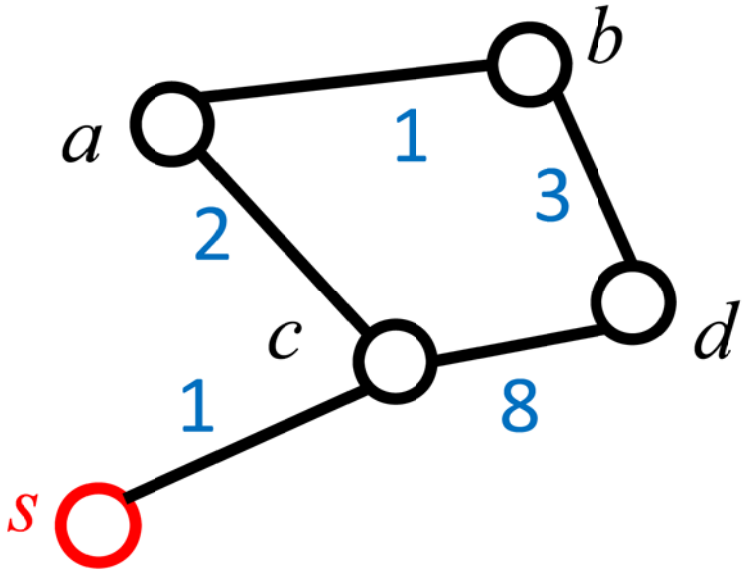
$$D(b) = \min(\infty,\ 3+1)$$

$$D(c) = 1$$

$$D(d) = \min(9,\ 3+\infty)$$

$$D(s) = 0$$

Dijkstra's Algorithm: Example



$W = \{s, c, a\}$

$D(a) = 3$

$D(b) = 4$

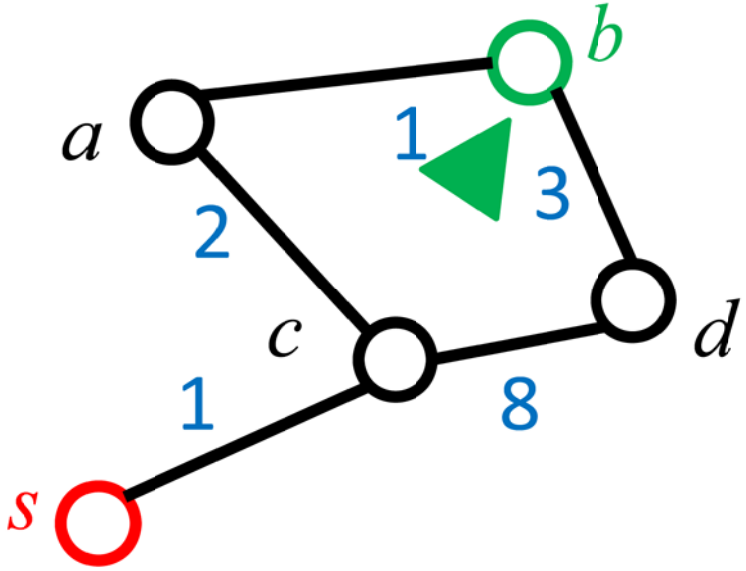
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Exit if $V = W$? → no

Dijkstra's Algorithm: Example



$W = \{s, c, a\}$

$D(a) = 3$

$D(b) = 4$

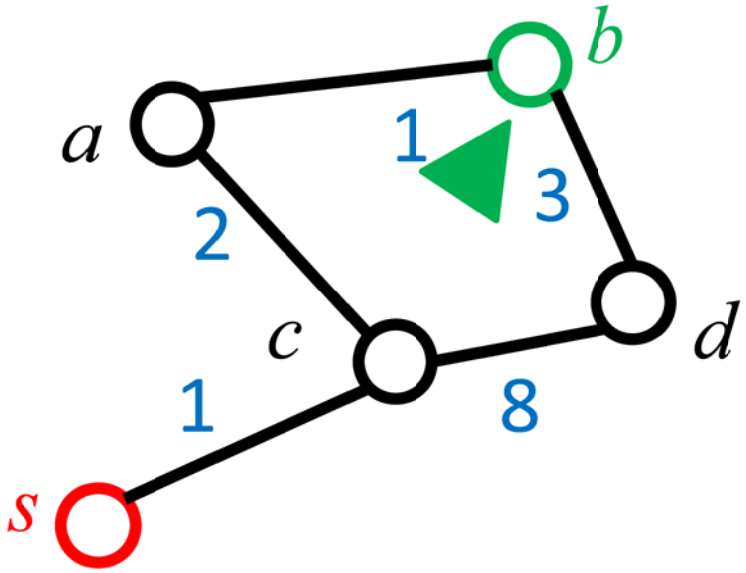
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Select a vertex in $V \backslash W$ with minimal $D(w)$ → select ***b***

Dijkstra's Algorithm: Example



$W = \{s, c, a, b\}$

$D(a) = 3$

$D(b) = 4$

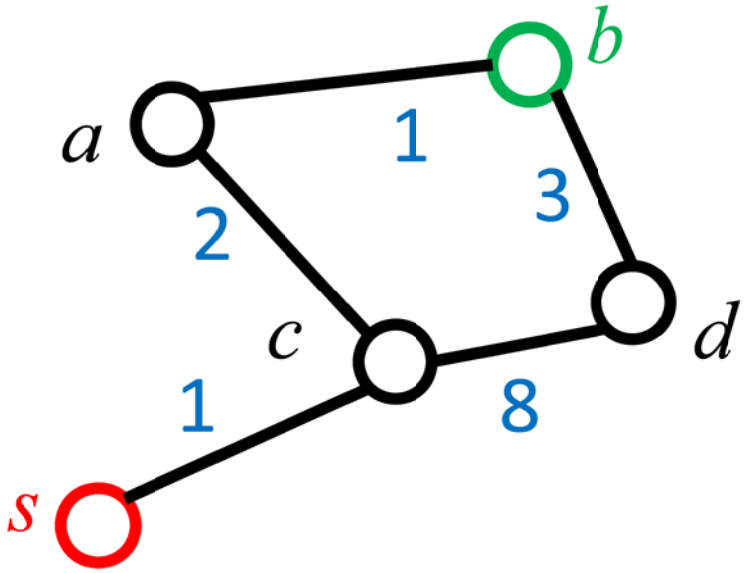
$D(c) = 1$

$D(d) = 9$

$D(s) = 0$

Add current vertex ***b*** to W

Dijkstra's Algorithm: Example



$$W = \{s, c, a, b\}$$

$$D(a) = 3$$

$$D(b) = 4$$

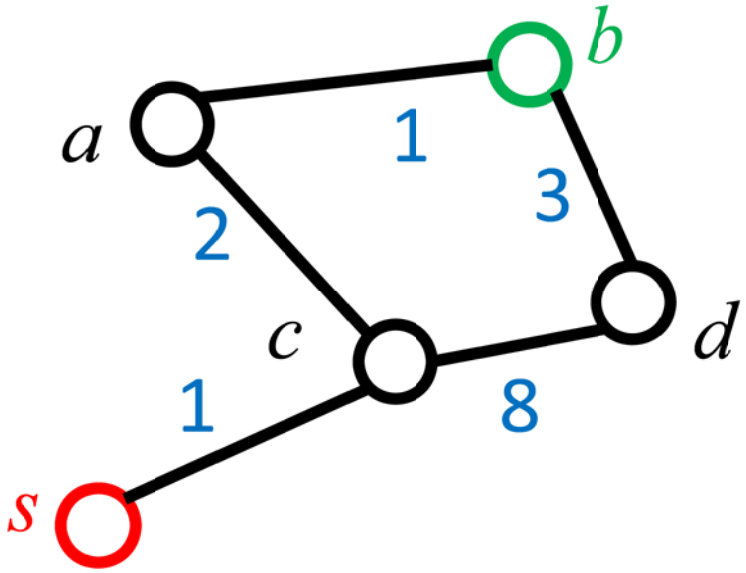
$$D(c) = 1$$

$$D(d) = \min(\,9\,,\, D(b) + d(b,d))$$

$$D(s) = 0$$

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), D(b) + d(b,v))$

Dijkstra's Algorithm: Example



$$W = \{s, c, a, b\}$$

$$D(a) = 3$$

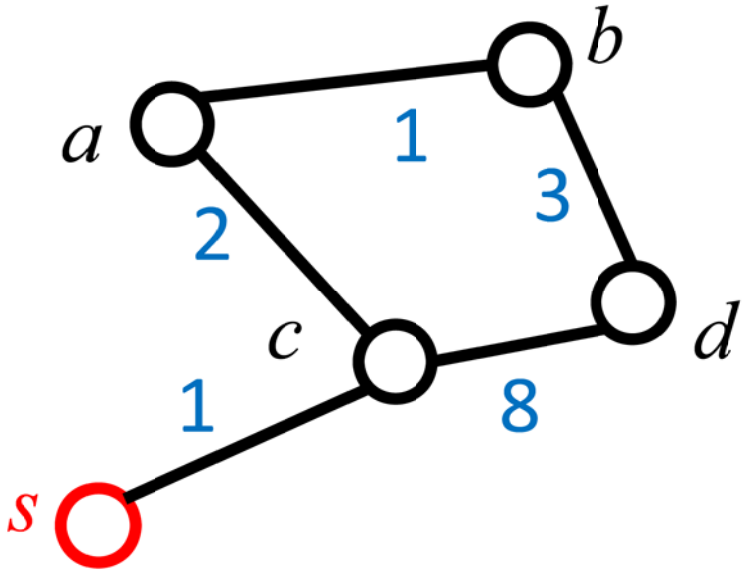
$$D(b) = 4$$

$$D(c) = 1$$

$$D(d) = \min(\, 9 \,,\, 4+3 \,)$$

$$D(s) = 0$$

Dijkstra's Algorithm: Example



$W = \{s, c, a, b\}$

$D(a) = 3$

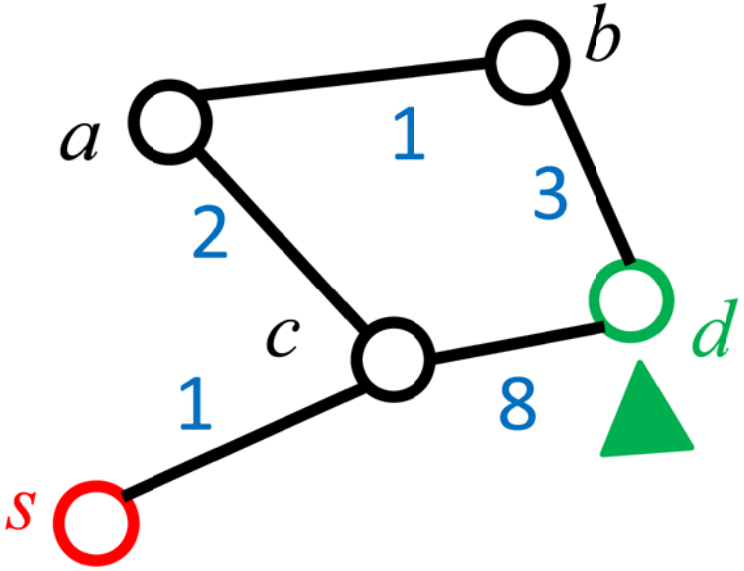
$D(b) = 4$

$D(c) = 1$

$D(d) = 7$

$D(s) = 0$

Exit if $V = W$? → no



$W = \{s, c, a, b\}$

$D(a) = 3$

$D(b) = 4$

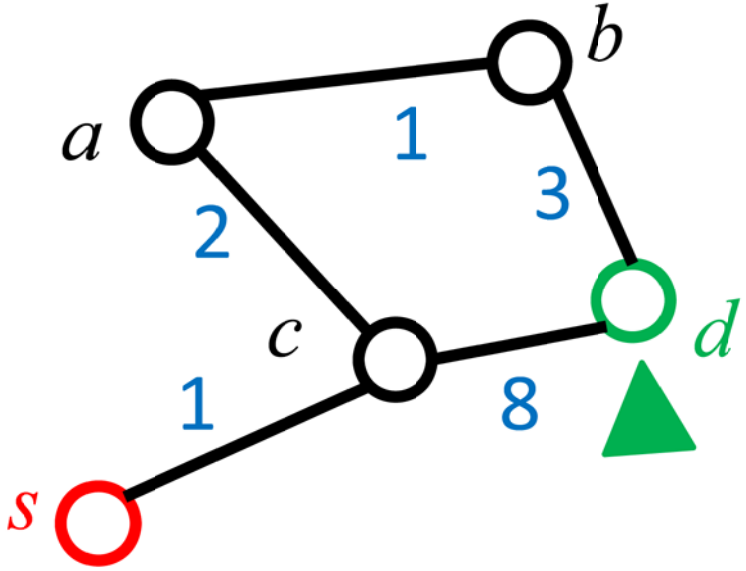
$D(c) = 1$

$D(d) = 7$

$D(s) = 0$

Select a vertex in $V \backslash W$ with minimal $D(w)$ → select ***d***

Dijkstra's Algorithm: Example



$$W = \{s, c, a, b, d\}$$

$$D(a) = 3$$

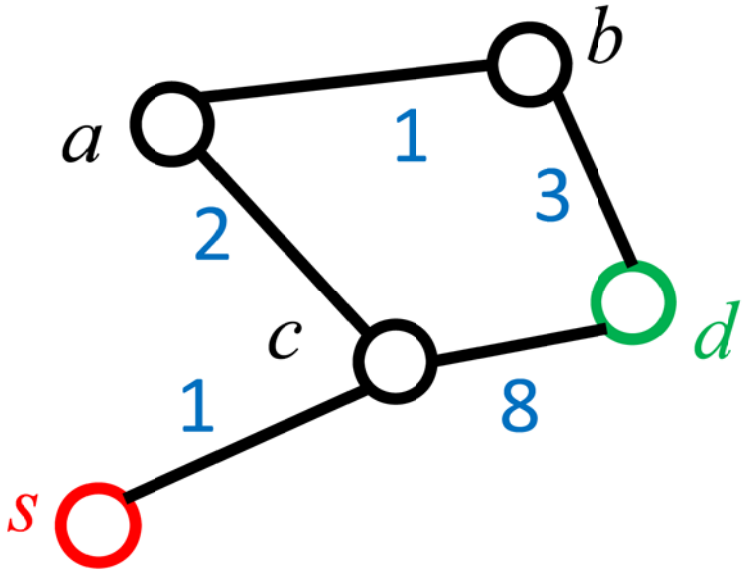
$$D(b) = 4$$

$$D(c) = 1$$

$$D(d) = 7$$

$$D(s) = 0$$

Add current vertex d to W



$$W = \{s, c, a, b, d\}$$

$$D(a) = 3$$

$$D(b) = 4$$

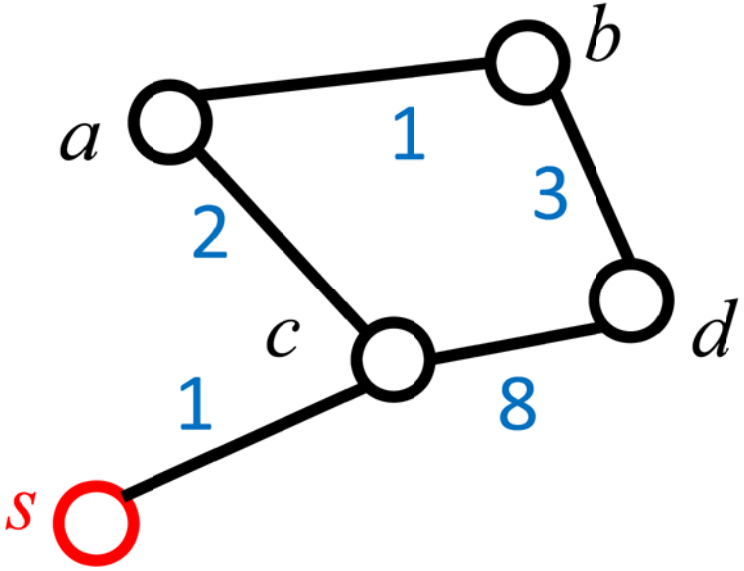
$$D(c) = 1$$

$$D(d) = 7$$

$$D(s) = 0$$

Update Distances: nothing to update since $V \backslash W$ is empty

Dijkstra's Algorithm: Example



$W = \{s, c, a, b, d\}$

$D(a) = 3$

$D(b) = 4$

$D(c) = 1$

$D(d) = 7$

$D(s) = 0$

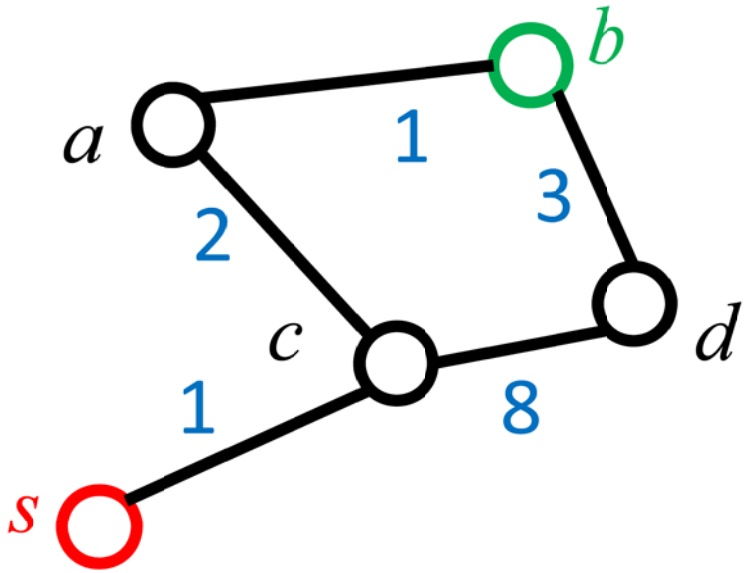
Exit if $V = W$? → **YES**

(D now contains shortest path length from s to every vertex in the graph)

GREEDY ALORITHMS in five parts

1) **CANDIDATE SET**
   …from which candidates for a solution are sampled
2) **SOLUTION SET**
   …which holds current elements which contribute to a solution
3) **SELECTION FUNCTION**
   …which chooses current best candidate to potentially add to the solution
4) **FEASIBILITY FUNCTION**
   …that is used to determine if a candidate can be used to contribute to a solution
5) **OBJECTIVE/SOLUTION FUNCTION**
   …which assigns a fitness value to a current solution and determines when the solution is complete

Dijkstra's Algorithm is `Greedy'



CANDIDATE SET (INVERSE)

$W = \{s, c, a, b\}$

$D(a) = 3$

$D(b) = 4$

$D(c) = 1$

$D(d) = 7$

$D(s) = 0$

Evolving SOLUTION SET

Dynamic SELECTION & FEASIBILITY FUNCTION

SOLUTION FUNCTION

Exit if $V = W$?

Update Distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), D(b) + d(b, v))$

Comparing Algorithms by Prim and Dijkstra

Prim's Algorithm

University of BRISTOL | Department of Computer Science

$G=(V,E)$ ... graph with weights $d(v,w) \geq 0$ for all $v,w \in V$ (where $d(v,w) = \infty$ if $(v,w) \notin E$)

$W \subseteq V$ ... set of visited nodes

$F \subseteq E$ ... set of utilised edges for current tree

$s \in V$ ... source vertex to start tree

$D(v)$ ... immediate distance from current tree to v

1) Initialise: $W = \{s\}; F = \{\}; \forall_{v \in V} : D(v) = d(s,v)$
2) Select a new **current vertex** w in $V \setminus W$ with minimal $D(w)$
3) Add current vertex w to W, add related edge to F
4) Update distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), d(w,v))$
5) If $V=W$ then exit, otherwise Goto 2)

Dijkstra's Algorithm

University of BRISTOL | Department of Computer Science

$G=(V,E)$ ... graph with weights $d(v,w) \geq 0$ for all $v,w \in V$ (where $d(v,w) = \infty$ if $(v,w) \notin E$)

$W \subseteq V$ ... set of visited nodes

$s \in V$ ... source vertex to calculate distances to

$D(v)$ ... current shortest distance estimate from s to v

1) Initialise: $W = \{s\}; \forall_{v \in V} : D(v) = d(s,v)$
2) Select a new **current vertex** w in $V \setminus W$ with minimal $D(w)$
3) Add current vertex w to W
4) Update distances: $\forall_{v \in V \setminus W} : D(v) = \min(D(v), D(w) + d(w,v))$
5) If $V=W$ then exit, otherwise Goto 2)

DYNAMIC PROGRAMMING

Concept: Iteratively *construct* a solution from solutions to *overlapping sub-problems*. Usually solutions to sub-problems are required to solve other sub-problems, and thus are stored to avoid re-computation.

If a problem can be solved optimally using dynamic programming it is said to have `*optimal substructure*'.

Greedy is often not optimal…

- for many problems greedy algorithms fail to produce optimal solutions
- in general, the greedy approach produces a `local' optimum which may be a bad case globally
  - e.g. gradient decent algorithms (tracing a function to lower and lower value down the `gradient') rarely provide the absolute global minimum
- however, if approximations are sufficient then greedy algorithms often deliver a fast way of computing results even for some of the toughest problems