DSA from 1st principles

- ADTs — set, sequence, trees, graphs -- definitions
- Abstraction — modelling, collection, implementation
- Recursion — sequence
- Recursion — binary tree
- AVL-trees — balancing & rotations (SLR, DLR, SRR, DRR)
- Heap — heapify, add, remove
- **Dijkstra** — **SPT** (Shortest Path Tree) — **path** costs
- **Prim** — **MST** (Minimal Spanning Tree) — **edge** costs
- **Kruskal** — **MST** (Minimal Spanning Tree) — **edge** costs
- DAG — Topological Sort (**dfs** OR in-degree=0)
- Heuristic — **TSP** - Travelling Salesman Problem

Set, sequence, trees, graphs - definitions

• SET	an **unordered** collection of **unique** values
• SEQUENCE	an **ordered** collection of values
	order ➔ **position** + (possibly **sorted**)
• TREE	a **hierarchical** collection of values
• Binary Tree	Tree + max 2 children + **order** (**LC**, **RC**)
• Binary Search Tree	BT + value < node ➔ add **LC** else add **RC**
• AVL Tree	BST + balance **|height(LC) – height(RC)|<2**
• GRAPH = (V, E)	**set** of vertices $\{v_i\}$ + **set** of edges $\{(v_1,v_2)\}$
	unordered, directed / undirected edges

• **SEQUENCE**	**lists, arrays, tables**
• **TREE**	**file systems, taxonomies, parse trees**
• **GRAPH**	**networks (computer, transport)**

Recursion – sequence

- **Seq ::= Head Tail | empty; Head ::= element; Tail ::= Seq;**
- De-construction functions **head: seq ➔ el; tail: seq ➔ seq**
- (Re-)construction function **cons: head x tail ➔ seq**
- Operations – cardinality & add **(empty + non-recursive + recursive)**

```
static listref size(listref L) {
 return is_empty(L) ? 0 : 1 + size(tail(L));
}
```

model

(1) Empty case
(2) Non-empty case **head**
(3) Non-empty case **tail** **(rec)**

```
static listref add_val(listref L, valtype v) {
 return is_empty(L)              ?  create_e(v)
      :  v < get_value(head(L))  ?  cons(create_e(v), L)
      :                             cons(head(L), add_val(tail(L),v));
}
```

Recursion – BT (Binary Tree)

- **BT ::= LC N RC | empty; N ::= element; LC, RC ::= BT;**
- De-construction functions **N: BT → el; LC: BT → BT; RC: BT → BT**
- (Re-)construction function **cons: LC x N x RC → BT**
- Operations – cardinality & add **(empty + non-recursive + recursive)**

```
static treeref size(treeref T) {
 return is_empty(T) ? 0 : 1 + size(LC(T)) + size(RC(T));
}

static treeref add(treeref T, int v)
{
  return   is_empty(T)             ? create_node(v)
        : v < get_value(node(T))   ? cons(add(LC(T), v), node(T), RC(T))
        : v > get_value(node(T))   ? cons(LC(T), node(T), add(RC(T), v))
        :                            T;
}
```

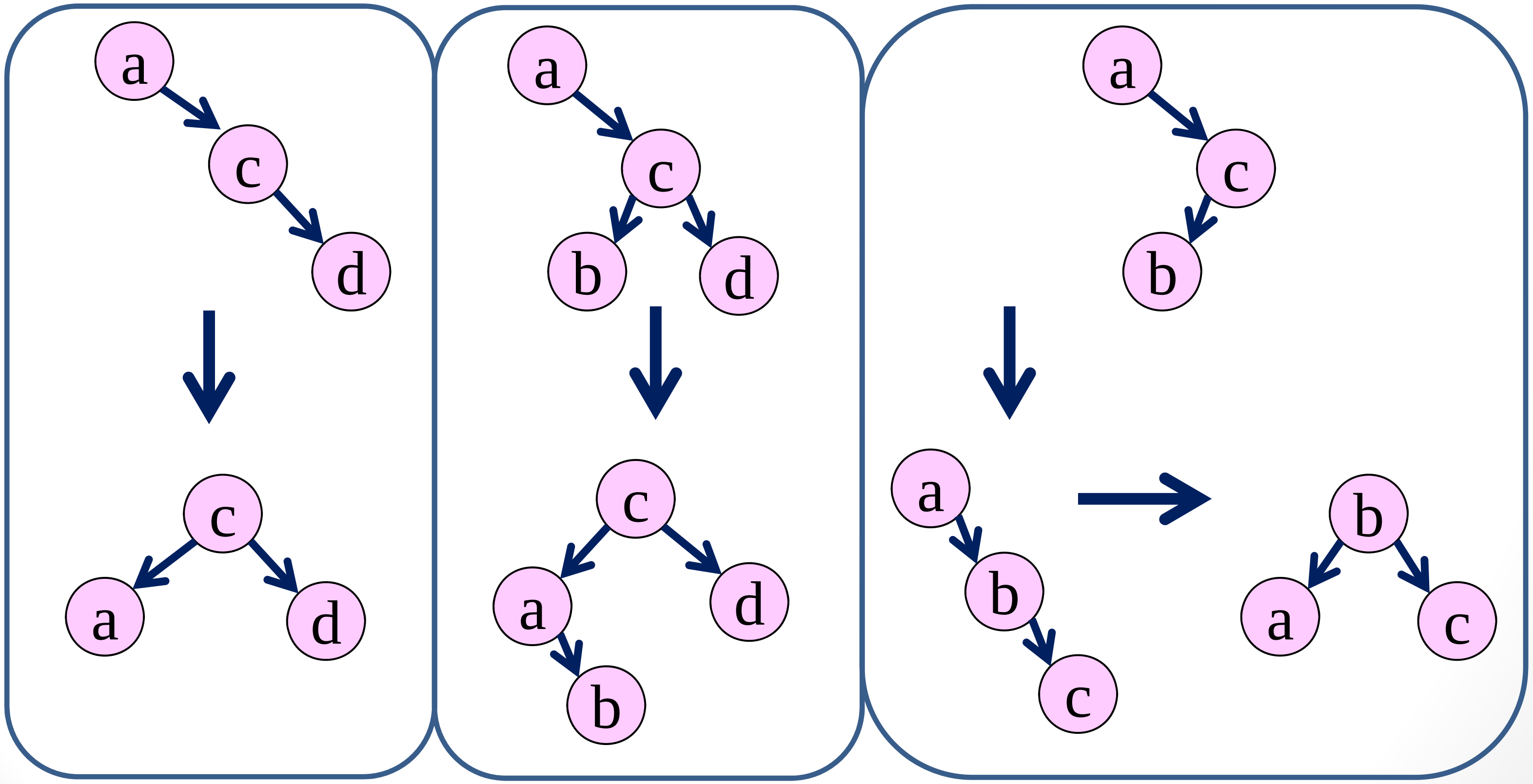
model

(1) Empty case
(2) Non-empty case **LC (rec)**
(3) Non-empty case **RC (rec)**
(4) Non-empty case **node**

AVL-trees & balancing

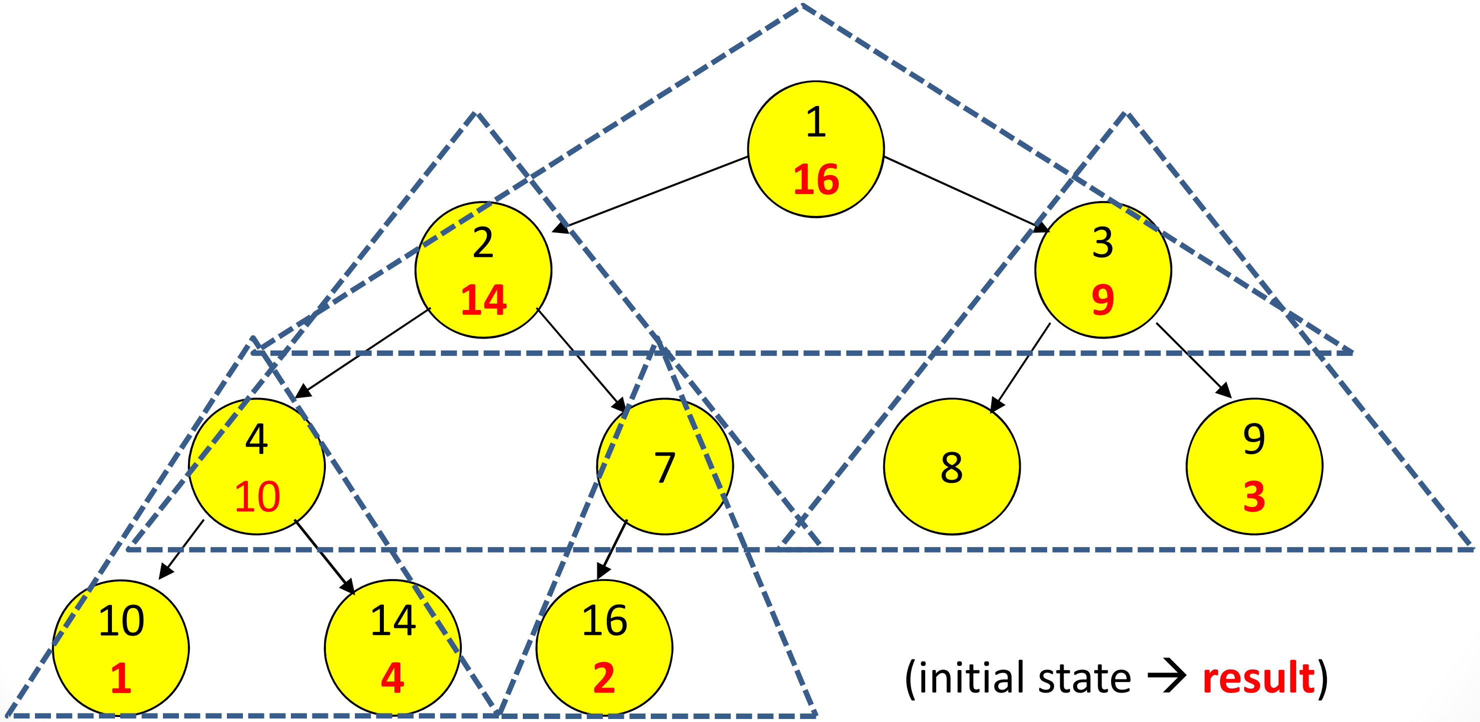
- Rotations: SLR, DLR, (mirror images - SRR, DRR)
- SLR(T) (2 cases)

DLR (SRR(RC)+SLR(T))



Heap

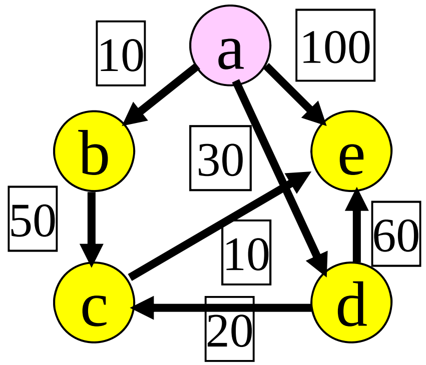
- **Heapify – parent = max_value(LC,P,RC);   start @ size(H)/2**
- **(16,7,-)→(7,16,-); (10,4,14)→(10,14,4); (8,3,9)→(8,9,3);**
- **(14,2,16)→(14,16,2);   rec (7,2,-)→(2,7,-);**
- **(16,1,9)→(1,16,9);   rec (14,1,7)→(1,14,7); (10,1,4)→(1,10,4);**

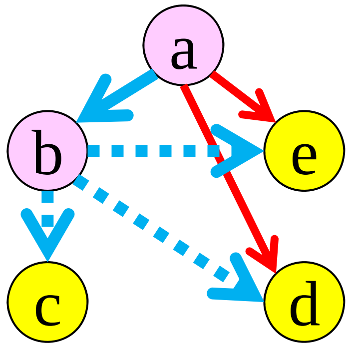


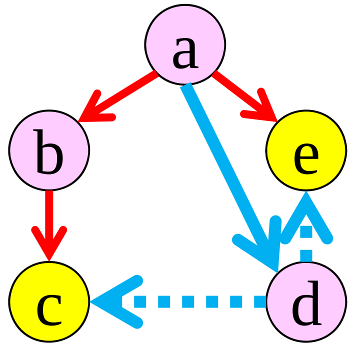
(initial state → **result**)

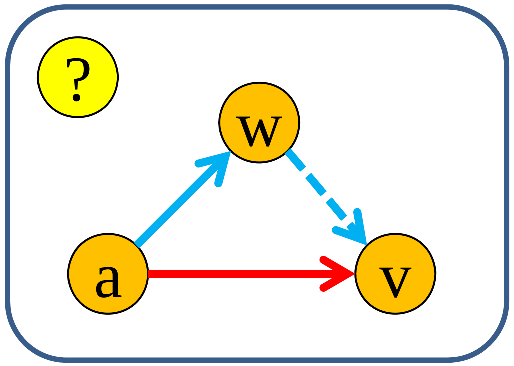
Dijkstra & SPT (Shortest Path Tree)

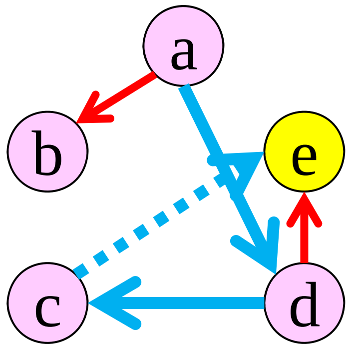
- The SPT "grows"; find shortest (local) paths

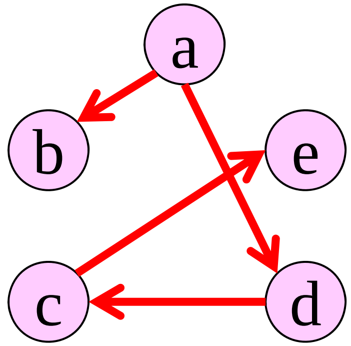






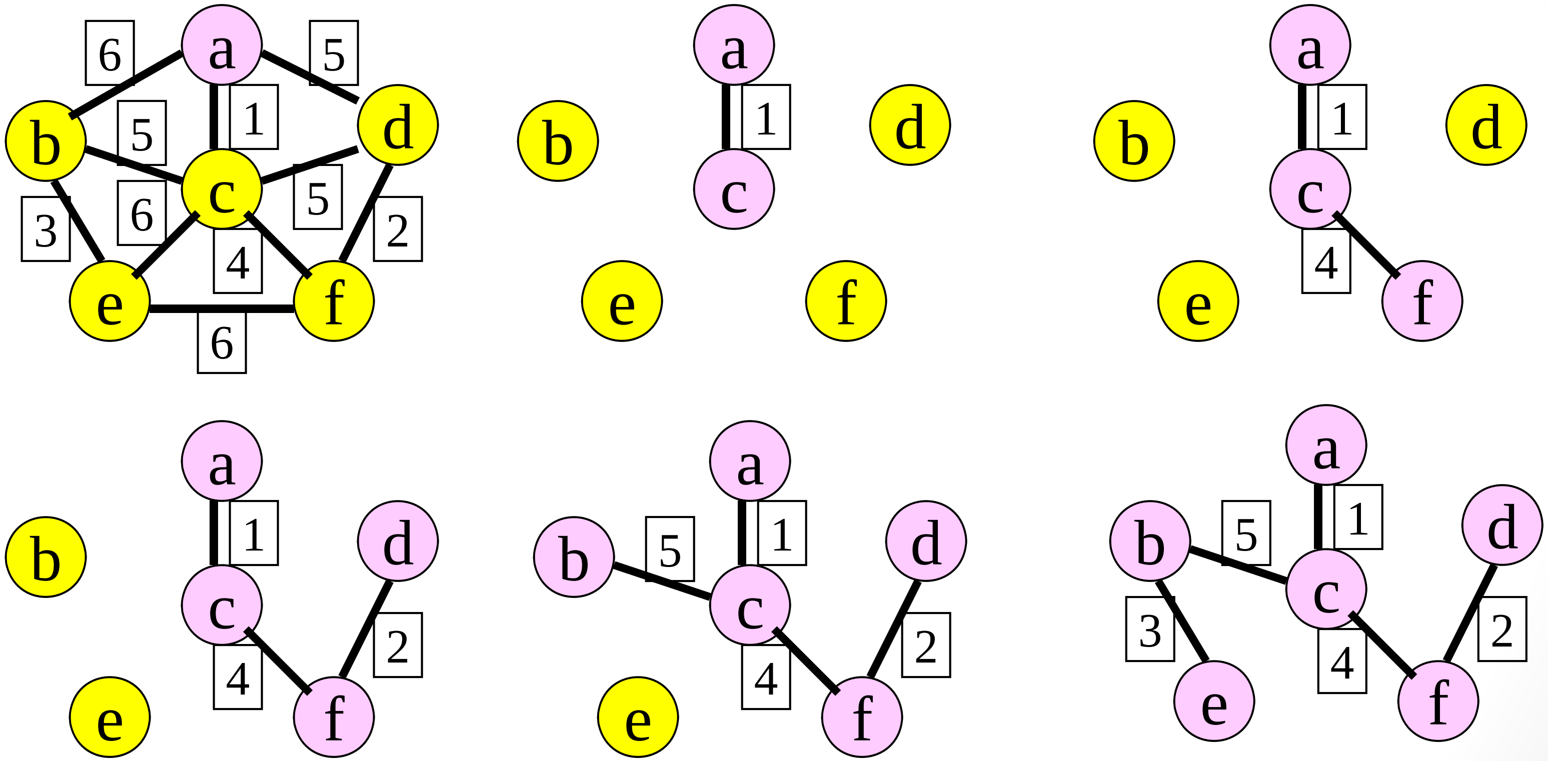






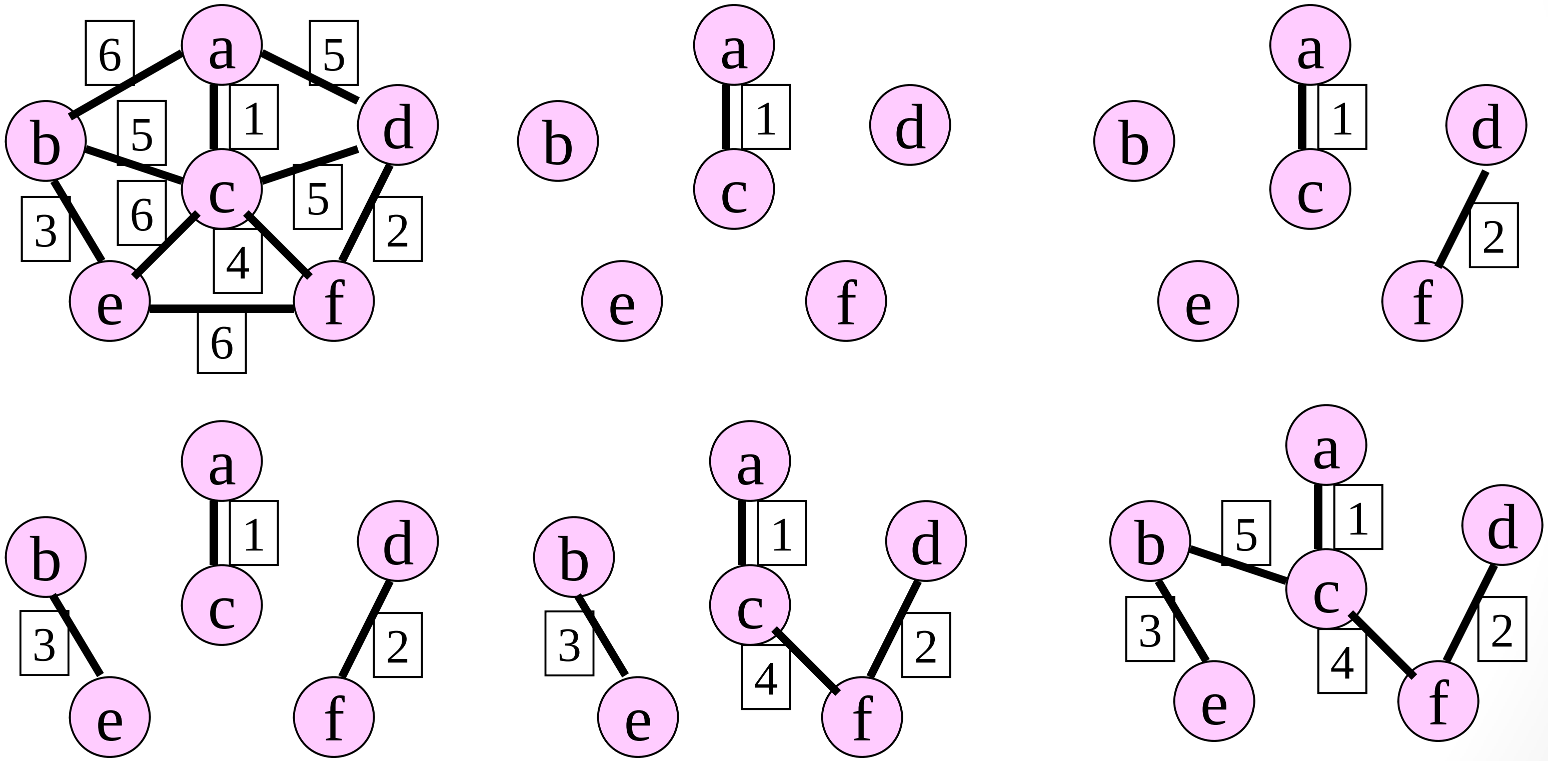
Prim & MST (minimal spanning tree)

- The MST “grows” from the start node as a single component



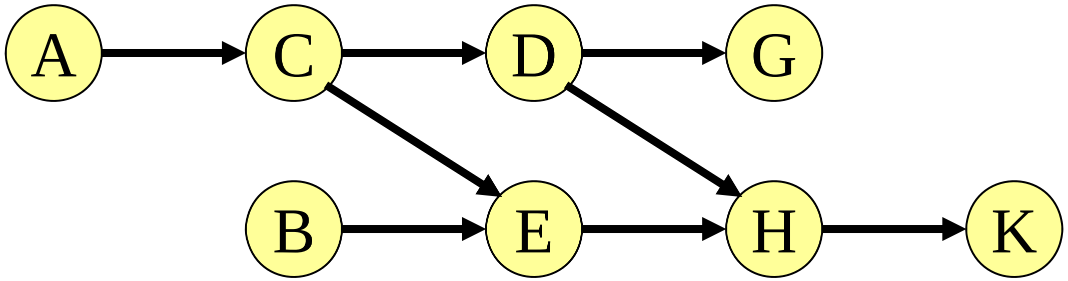
Kruskal & MST (minimal spanning tree)

- Remove edges; make PQ from edges; merge components



Topological Sort

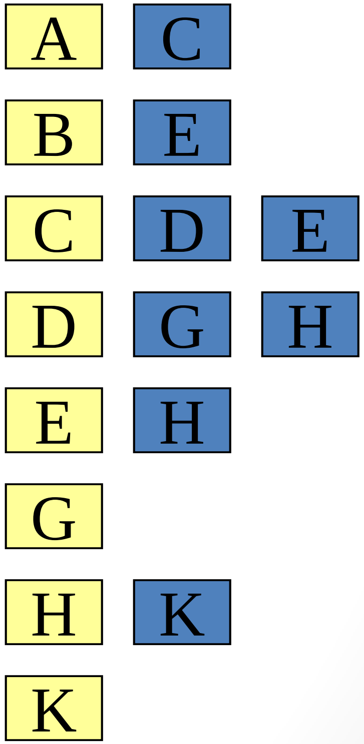
- DFS (depth first search) & reverse; **OR** in-degree = 0



start: A

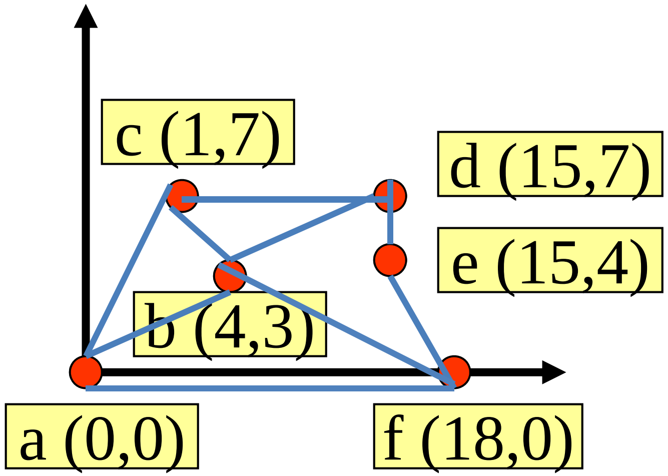
tsort(A) => G K H D E C A B

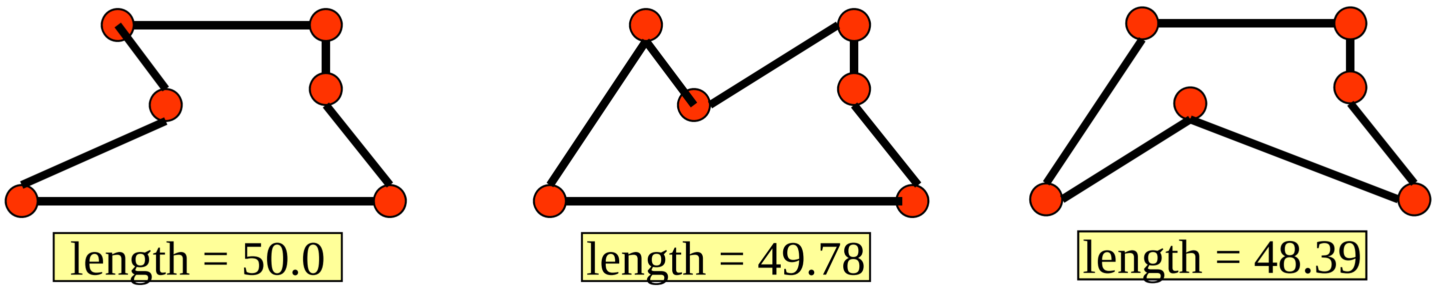
reverse => B A C E D H K G



Heuristic – TSP (Travelling Salesman Problem)

- **Hamiltonian cycle**: cycle in a graph G = (V,E) which contains each vertex in V exactly once, except for the starting and ending vertex that appears twice
- **degree(v) = 2 for all v in V**





First Principles - Summary

• Definitions	Set, Sequence, Tree, Graph = (V,E)
	non-recursive, **recursive** (sequence, BT)
• Recursion	the definition also defines the code
• Pictures	ADTs are better understood via pictures
	Dijkstra, Prim, Kruskal
• Simple cases	code: cardinality (size) & add
	AVL-tree balancing & rotations
	Heap viewed as a BT (LC, Parent, RC)
• Worked examples	problem → **method** ← solution
	creative guesswork
• Preparation	**interpret** algorithms
	reflect upon what you are doing
• **Understand!**	Simpler than memorising everything