Data Structures

Components, cycles and spanning trees

The notes for the lesson “Components, cycles and spanning trees”: chapters 5.5 to 5.6, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

2 chapters | Chapters 5.5 to 5.6

About these notes

These are the notes for the University free n easy video lesson **Components, cycles and spanning trees** on Data Structures: chapters 5.5 to 5.6, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

5.5 Connected components and cycle detection

Two questions come up constantly about a graph: how many separate pieces does it have, and does it contain a cycle? Both are a traversal with a little bookkeeping: the restarts of DFS count the components, and an edge that returns to a visited vertex reveals a cycle. Only a few syllabi list this as a separate topic (a common tutorial asks for components while tracing BFS and DFS); elsewhere the ideas sit inside 5.3 and 5.4.

You need: DFS from 5.3; Kahn's algorithm from 5.4.

Connected components

A connected component of an undirected graph is a maximal set of vertices in which every pair is joined by a path. Every vertex lies in exactly one component. To find them:

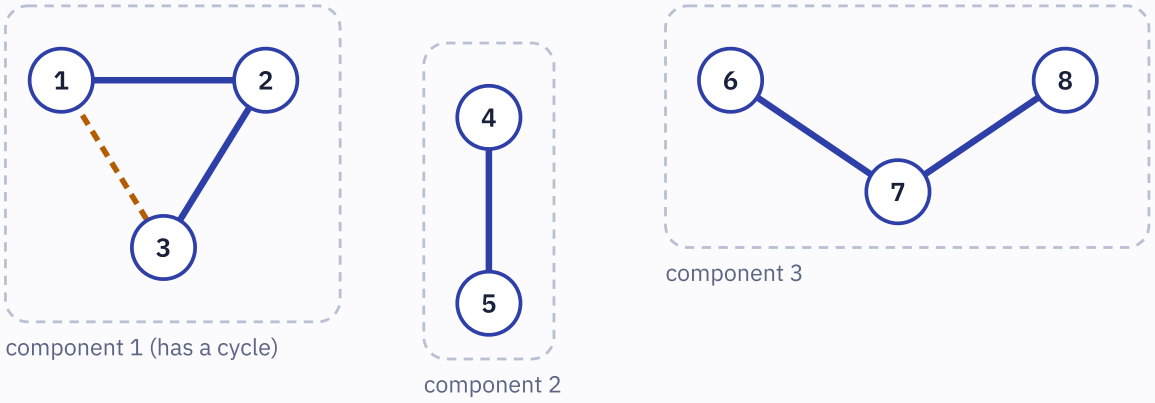
1. Set comp[v] = 0 for all v and a counter c = 0.
2. For each vertex v: if comp[v] = 0, increment c and run DFS (or BFS) from v, labelling every vertex reached with c.
3. c is the number of components; comp[u] = comp[v] tells in O(1) whether u and v are connected.

The loop is O(V + E): each vertex and edge is touched in one search. The **disjoint-set (union-find)** structure of 5.6 does the same job when edges arrive one at a time: union the endpoints of each edge, and the sets left are the components. For digraphs the matching idea is the **strongly connected component**, outside this unit.

Cycle detection

Undirected graph. Run DFS carrying the parent of each vertex. On an edge u–v: if v is unvisited, recurse (tree edge); if v is visited and v ≠ parent[u], it is a back edge and the graph has a cycle. Excluding the parent matters: u–v is the edge you just came along. A count also works: a connected graph with |E| ≥ |V| has a cycle, since a tree has only |V| − 1 edges.

Directed graph. The parent test fails, because u→v and v→u are different edges that do form a cycle. Use three colours: white (unvisited), grey (on the recursion stack), black (finished). On an edge u→v: white means recurse; grey is a back edge, hence a cycle; black is harmless. Or run Kahn's algorithm and check whether fewer than V vertices come out.



DFS from 1: tree edges 1–2, 2–3; edge 3–1 reaches a visited non-parent vertex → back edge → cycle

Figure 5.5.1 Eight vertices, six edges, three components. DFS from 1 labels 1, 2, 3; at 3 the neighbour 1 is visited and is not 3's parent, so 3–1 (dashed) is a back edge and 1–2–3–1 is a cycle. Restarts at 4 and 6 give components 2 and 3, both trees.

WORKED EXAMPLE

Question. An undirected graph has vertices 1–8 and edges 1–2, 2–3, 3–1, 4–5, 6–7, 7–8. (a) Count its components with DFS. (b) Does it have a cycle? Show the back edge. (c) For the digraph P→Q, Q→R, R→P, R→S, show the colour-based test.

Solution.

1. v = 1 unvisited: c = 1, DFS(1) visits 1, 2 (via 1–2), 3 (via 2–3). From 3, neighbour 1 is visited and parent[3] = 2 ≠ 1: back edge 3–1. Component 1 = {1, 2, 3}.
2. v = 4 unvisited: c = 2, DFS(4) visits 4, 5. Component 2 = {4, 5}.
3. v = 6 unvisited: c = 3, DFS(6) visits 6, 7, 8. Component 3 = {6, 7, 8}. Answer: **3 components**.
4. Cycle: the back edge 3–1 closes the tree path 1–2–3: cycle 1–2–3–1. Check: component 1 has 3 vertices and 3 edges, one more than a tree; the other two have $|E| = |V| - 1$.
5. Digraph: DFS(P): P grey → Q grey → R grey → edge R→P meets grey P: back edge, cycle P→Q→R→P. Kahn agrees: every in-degree is 1, the queue starts empty, $0 < 4$ vertices come out.

EXAM TRAPS

- Undirected test: ignore the edge back to the parent, or every tree edge looks like a cycle.
- Directed test: a visited vertex means a cycle only if it is grey (on the stack); a black vertex reached again is a cross edge.
- Favourite question: "Count the components and say whether the graph is a tree, a forest or has a cycle." A forest (acyclic, k components) has exactly $|E| = |V| - k$.

CHECK YOURSELF

– REMEMBER **Define a connected component.** 2 marks

A maximal set of vertices of an undirected graph in which every two vertices are joined by a path.

– REMEMBER **What do white, grey and black mean in directed cycle detection?** 2 marks

White: unvisited; grey: visited and still on the recursion stack; black: finished. An edge into a grey vertex is a back edge, so a cycle.

– UNDERSTAND **Why does the parent-exclusion test not work for directed graphs?** 3 marks

In a digraph u→v and v→u are distinct edges forming a real 2-cycle, so skipping "the parent" would miss it; in an undirected graph u–v is one edge.

– APPLY **A graph has 10 vertices, 7 edges and no cycle. How many components?** 2 marks

A forest satisfies |E| = |V| – k, so k = 10 – 7 = 3.

– ANALYZE **A connected undirected graph has V vertices and V edges. How many cycles does it have, and how do you find the cycle in O(V)?** 4 marks

Exactly one: a spanning tree uses V – 1 edges and the one extra edge closes one cycle. DFS with parents finds the single back edge u–v; the cycle is the tree path v to u plus that edge; O(V + E) = O(V).

BEYOND THE SYLLABUS

Not for the exam. Strongly connected components (Tarjan's and Kosaraju's algorithms) condense any digraph into a DAG, which is how compilers analyse call graphs. Deadlock detection in operating systems is directed cycle detection on the wait-for graph. Search: "Tarjan SCC", "wait-for graph deadlock".

5.6 Minimum spanning trees: Prim's and Kruskal's algorithms

Suppose six villages must be joined by optical fibre and the cost of laying cable between each pair is known. You want every village connected and the total cost as small as possible. The answer is a minimum spanning tree (MST): a set of |V| – 1 edges that connects all vertices with the least total weight. Two greedy algorithms find it. Kruskal's takes edges cheapest first, skipping any that would close a cycle. Prim's grows one tree from a start vertex, always adding the cheapest edge that leaves the tree. Both are exact, not approximations, because of the cut property explained below. Most syllabi examine this chapter, and some specifically want Kruskal's with the disjoint-set structure and Prim's with a priority queue.

You need: spanning tree from 5.1; cycle detection from 5.5; binary heaps from an earlier lesson.

Spanning trees and the cut property

A spanning tree of a connected undirected graph G is a subgraph that contains every vertex and is a tree, so it has exactly |V| − 1 edges and no cycle. A connected weighted graph has many spanning trees; an MST is one of minimum total weight. If all edge weights are distinct the MST is unique; with ties there may be several MSTs of the same weight.

The **cut property** is why greedy works. A cut splits V into two non-empty parts S and V − S. Take the lightest edge e crossing the cut. Some MST contains e: if an MST T did not, adding e to T makes a cycle, that cycle must cross the cut by another edge f with w(f) ≥ w(e), and swapping f for e gives a spanning tree no heavier than T. Kruskal and Prim both repeatedly pick the lightest edge across some cut, so every edge they choose is safe.

Kruskal's algorithm

1. Sort all E edges by weight, ascending: O(E log E).
2. Make each vertex its own set (disjoint-set structure).
3. For each edge (u, v) in sorted order: if find(u) ≠ find(v), the endpoints are in different trees: add the edge to the MST and union the two sets. Otherwise the edge would close a cycle: skip it.
4. Stop after |V| − 1 edges are added.

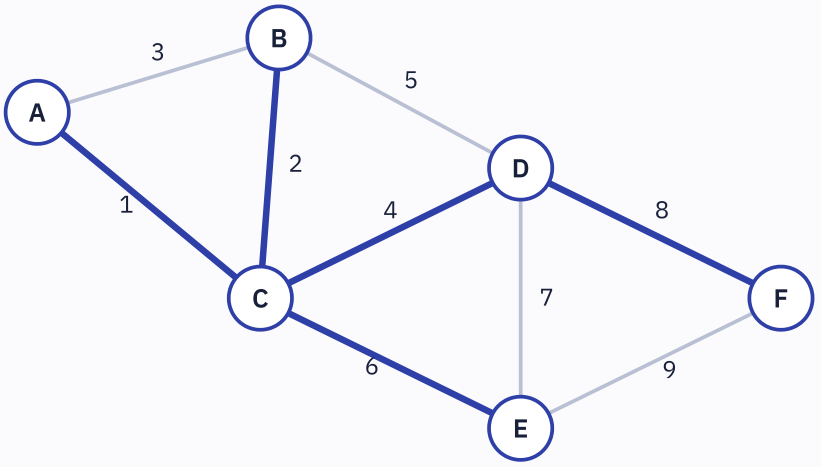
The **disjoint-set (union-find)** structure keeps a parent array: find(x) follows parents to the root, union links one root under the other. With union by rank (attach the shorter tree under the taller) and path compression (make every node on a find path point straight to the root), a sequence of operations is almost O(1) each, so Kruskal's total is dominated by the sort: O(E log E), which equals O(E log V) since $E \le V^2$.

Prim's algorithm

1. Pick a start vertex s. For every vertex keep key[v] = cheapest known edge from the tree to v (∞ at first, key[s] = 0) and parent[v].
2. Repeat V times: take the vertex u not yet in the tree with smallest key; add u (and the edge parent[u]–u) to the tree.
3. For each neighbour v of u not in the tree: if w(u, v) < key[v], set key[v] = w(u, v) and parent[v] = u.

With a plain array scan for the minimum, each of the V rounds costs O(V): $O(V^2)$, fine for dense graphs. With a binary min-heap of keys (an earlier lesson) the minimum is O(log V) and each key decrease O(log V), giving O((V + E) log V) = O(E log V) for connected graphs, better for sparse

graphs. The structure of Prim's is identical to Dijkstra's in 5.7; only the key rule differs (edge weight here, path length there).



thick edges = the minimum spanning tree (Prim from A finds the same five edges)

Kruskal picks, in weight order

AC 1 ✓
BC 2 ✓
AB 3 × cycle
CD 4 ✓
BD 5 × cycle
CE 6 ✓
DE 7 × cycle
DF 8 ✓ (5 edges, stop)

MST weight 1+2+4+6+8 = 21

Figure 5.6.1 Six vertices, nine weighted edges. Thick edges are the MST: AC, BC, CD, CE, DF, total 21. The list on the right is Kruskal's pass in weight order: AB (3) is rejected because A and B are already joined through C, and BD (5) and DE (7) likewise. Prim's from A chooses the same five edges in the order AC, CB, CD, CE, DF.

WORKED EXAMPLE 1: KRUSKAL'S ALGORITHM

Question. Find the MST of the graph in Figure 5.6.1 (edges AB 3, AC 1, BC 2, BD 5, CD 4, CE 6, DE 7, DF 8, EF 9) by Kruskal's algorithm. Show the sets after each step and the total weight.

Solution. Sorted edges: AC 1, BC 2, AB 3, CD 4, BD 5, CE 6, DE 7, DF 8, EF 9. Start with sets {A} {B} {C} {D} {E} {F}.

EDGE (W)	FIND(U), FIND(V)	ACTION	SETS AFTER	RUNNING WEIGHT
AC (1)	different	accept	{A,C} {B} {D} {E} {F}	1
BC (2)	different	accept	{A,B,C} {D} {E} {F}	3
AB (3)	same set	reject (cycle A–B–C)	unchanged	3
CD (4)	different	accept	{A,B,C,D} {E} {F}	7
BD (5)	same set	reject	unchanged	7
CE (6)	different	accept	{A,B,C,D,E} {F}	13
DE (7)	same set	reject	unchanged	13
DF (8)	different	accept (5th edge, stop)	{A,B,C,D,E,F}	21

MST = {AC, BC, CD, CE, DF}, weight **21**. EF (9) is never examined because |V| − 1 = 5 edges were already chosen.

WORKED EXAMPLE 2: PRIM'S ALGORITHM

Question. Run Prim's algorithm on the same graph from A. Show key and parent after each vertex is added.

Solution. Keys start A 0, others ∞.

ROUND	ADDED (KEY)	KEYS AFTER RELAXING ITS EDGES (B, C, D, E, F)	PARENTS CHANGED
1	A (0)	3, 1, ∞, ∞, ∞	B←A, C←A
2	C (1)	2, –, 4, 6, ∞	B←C (2 < 3), D←C, E←C
3	B (2)	–, –, 4, 6, ∞	none (BD 5 is not < 4)
4	D (4)	–, –, –, 6, 8	F←D (DE 7 is not < 6)
5	E (6)	–, –, –, –, 8	none (EF 9 is not < 8)
6	F (8)	all in tree	

Tree edges from the parents: AC 1, CB 2, CD 4, CE 6, DF 8. Total 1 + 2 + 4 + 6 + 8 = **21**, the same tree as Kruskal's, as it must be here because all weights are distinct.

EXAM TRAPS

- Kruskal's rejects an edge whose endpoints are already connected, even if it is the cheapest remaining. Students add AB (3) because "3 is small" and get a cycle.
- Prim's key is the weight of one edge, not a path length. Updating key[v] to key[u] + w(u, v) is Dijkstra's rule and gives the wrong tree.
- An MST has exactly |V| − 1 edges. If your answer has more, there is a cycle; if fewer, the graph was disconnected (then the answer is a minimum spanning forest).
- Complexities: Kruskal O(E log E) (sorting), Prim $O(V^2)$ with an array or O(E log V) with a heap. Do not write O(V log V) for Kruskal.
- Favourite question: "Apply Prim's and Kruskal's to the given graph and compare." Show the step table for each and remark that Kruskal suits sparse graphs and edge lists, Prim suits dense graphs and adjacency structures.

CHECK YOURSELF

– REMEMBER **Define a minimum spanning tree.** 2 marks

A spanning tree (|V| – 1 edges joining all vertices without a cycle) of a connected weighted undirected graph whose total edge weight is the smallest among all spanning trees.

– REMEMBER **State the time complexity of Kruskal's algorithm and what dominates it.** 2 marks

O(E log E), dominated by sorting the edges; the union-find operations are nearly constant each.

– UNDERSTAND **State the cut property and say why it justifies Prim's algorithm.** 3 marks

For any cut, the lightest crossing edge belongs to some MST. Prim's at each step takes the lightest edge leaving the current tree, that is, crossing the cut (tree, rest); so every edge it adds is safe and the result is an MST.

– UNDERSTAND **When is the MST unique?** 2 marks

It is unique if all edge weights are distinct; that condition is sufficient, not necessary. With equal weights several spanning trees can share the minimum weight, though a graph with ties can still have just one. The minimum weight itself is always unique.

– APPLY **Vertices 1–4, edges 1–2 (1), 2–3 (2), 3–4 (3), 4–1 (4), 1–3 (5). Give the MST by Kruskal's and its weight.** 4 marks

Accept 1–2 (1), 2–3 (2), 3–4 (3); 4–1 (4) rejected (1 and 4 already joined); stop at 3 edges. Weight 6.

– APPLY **Same graph, Prim's from vertex 3: list the order in which vertices join the tree.** 3 marks

3 (0); keys 2:2, 4:3, 1:5 → add 2; key 1 becomes 1 → add 1; key 4 stays 3 (edge 1–4 is 4) → add 4. Order 3, 2, 1, 4; weight 2 + 1 + 3 = 6.

– ANALYZE **If every edge weight is increased by 5, does the MST change? If every weight is squared (weights positive)?** 4 marks

Neither changes the MST: both maps preserve the order of weights, and Kruskal's decisions depend only on that order. The MST's total weight changes, though: by 5(V – 1) in the first case.

BEYOND THE SYLLABUS

Not for the exam. Borůvka's algorithm, older than both (1926), merges components in parallel rounds and is the basis of parallel and distributed MST codes; Karger, Klein and Tarjan gave a randomised expected-linear-time MST algorithm in 1995. MSTs appear in network design (BSNL fibre backbones, power grids), in single-linkage clustering in machine learning, and as the building block of approximation algorithms for the travelling salesman problem. Dynamic MST maintains the tree while edges change. Search: "Borůvka algorithm", "single-linkage clustering MST", "Steiner tree".

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
Components	number of DFS restarts; forest has $\|E\| = \|V\| - k$	5.5
Cycle test	undirected: visited non-parent neighbour; directed: edge to a grey vertex, or Kahn count < V	5.5
MST	$\|V\| - 1$ edges, minimum total weight; cut property	5.6
Kruskal	edges by weight, union-find rejects cycles; $O(E \log E)$	5.6
Prim	grow a tree, key[v] = cheapest edge to tree; $O(V^2)$ or $O(E \log V)$	5.6

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

- **4. Explain the cut property of minimum spanning trees and show how it proves the correctness of Prim's and Kruskal's algorithms. (10 marks)**

  Define cut and light edge; exchange argument (add light edge to an MST, remove the other crossing edge of the cycle); Prim picks the light edge of the cut (tree, rest); Kruskal's chosen edge is light for the cut separating its two components; hence each added edge is safe. (5.6)

Apply

- **8. Apply Kruskal's algorithm to the graph with edges AB 6, AC 1, AD 5, BC 5, BE 3, CD 5, CE 6, CF 4, DF 2, EF 6, showing the disjoint sets after each accepted edge, and state the MST weight. (12 marks)**

  Order: AC1, DF2, BE3, CF4, AD5/BC5/CD5 (ties), AB6, CE6, EF6. Accept AC, DF, BE, CF; then AD: A and D already joined (A–C–F–D), reject; BC: accept (joins {B,E} to {A,C,D,F}); 5 edges, stop. Weight 1 + 2 + 3 + 4 + 5 = 15. (5.6)

- **9. Run Prim's algorithm from vertex A on the graph of question 8, showing the key and parent arrays after each vertex is added. (10 marks)**

  Add A (keys B6 C1 D5); C (B5 D5 E6 F4); F (D2 E6); D (no change); B (5, via C; E becomes 3); E (3). Tree AC, CF, FD, CB, BE: weight 15, same as Kruskal. (5.6)

Analyze and evaluate

- **12. A city has 2×10^5 junctions and 5×10^5 road segments. Choose a representation and algorithms for (a) checking whether the map is connected, (b) finding the fewest-turn route, (c) finding the fastest route with travel times, (d) planning the cheapest set of roads to keep open after a flood. Justify each with complexity. (16 marks)**

  Adjacency list (sparse: average degree 5). (a) one DFS/BFS, O(V + E), count restarts. (b) BFS, O(V + E). (c) Dijkstra with a heap, O(E log V); not Floyd-Warshall ($V^3 \approx 8 \times 10^{15}$). (d) MST: Kruskal O(E log E) or Prim with heap. (spans 5.2, 5.3, 5.5, 5.6, 5.7)

Glossary (terms used in this lesson)

Back edge — A DFS edge to an ancestor (undirected: a visited non-parent; directed: a grey vertex); proves a cycle.

BFS	Breadth-first search: queue-based traversal by increasing distance in edges.
Connected component	Maximal set of mutually reachable vertices in an undirected graph.
Cut property	The lightest edge crossing any cut belongs to some MST.
DAG	Directed acyclic graph; the only digraphs with a topological order.
Degree	Number of edges at a vertex; in-degree and out-degree for digraphs.
DFS	Depth-first search: stack or recursion based traversal that goes deep before wide.
Kahn's algorithm	Topological sort by repeatedly removing in-degree-0 vertices.
Kruskal's algorithm	MST by taking edges in weight order and skipping those that close a cycle.
Minimum spanning tree	Spanning tree of least total weight.
Path	Walk with no repeated vertex; its length is its edge count or weight sum.
Prim's algorithm	MST by growing one tree, always adding the cheapest edge leaving it.
Spanning tree	Subgraph that is a tree and contains every vertex; $\|V\| - 1$ edges.
Strongly connected	Every vertex reaches every other by a directed path.
Weighted graph	Graph with a number w(u, v) on each edge.

Sources and further reading

Textbooks

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*, Part VI "Graph Algorithms" (elementary graph algorithms, minimum spanning trees, single-source and all-pairs shortest paths). The standard reference for the proofs in 5.6 to 5.8.
- Mark Allen Weiss, *Data Structures and Algorithm Analysis in C*, chapter "Graph Algorithms" (topological sort, shortest paths, minimum spanning tree, DFS applications). Closest to the way many syllabi phrase this unit.
- Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed, *Fundamentals of Data Structures in C*, chapter on graphs (representations, traversals, spanning trees, shortest paths).
- Aaron M. Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, *Data Structures Using C*, chapter on graphs.
- Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th edition, chapter 4 "Graphs" (undirected graphs, directed graphs, MSTs, shortest paths), with free online companion material.

Open web resources (opened while writing)

- VisuAlgo, "Graph Traversal (DFS/BFS)": animated BFS, DFS, topological sort and component finding on your own graphs, free. https://visualgo.net/en/dfsbfs
- CP-Algorithms, "Floyd-Warshall algorithm": loop order, negative-cycle handling and path reconstruction, with code. https://cp-algorithms.com/graph/all-pair-shortest-path-floyd-warshall.html
- Wikipedia, "Dijkstra's algorithm": history (conceived 1956, published 1959), the heap-based complexities quoted in 5.7, and pointers to A* and Johnson's algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Syllabi: the universal syllabus and a range of university course documents.

Built for the University free n easy YouTube channel from the universal syllabus of Data Structures (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.