Elementary graph algorithms (s. 589) (ill. 590)

Adjacency lists - array med linked lists for hver knude
Adjacency matrices - godt til dense grafer

BFS (breadth-first search) (kap 22.2 s. 595) (ill. 596)

Laver et træ udfra en graf
Sætter u.d og u.π for hver vertex
Initialiser alle vertexes. Initialiser startknuden s. Tilføj s til køen Q. While Q ikke er tom, så tilføj alle naboer til køen og sæt u.d til en højere end den nuværende
Tager tid ***O(V+E)***

DFS (kap 22.3 s. 603) (ill. 605)

Algoritme s. 604
Tager tid ***O(V+E)***

Discovery time	sættes når algoritmen første gang når knuden
Finishing time	sættes når alle udgående kanter er besøgt

Tree edges	Edges in the depth first forest
Back edges	Edges to a ancestor (grey node) (cycles)
Forward edges	Nontree edges to a descendant
Cross edges	All other edges

Topological sorting (ill. 613)

DFS to compute finishing times
When a vertex is finished, insert it in the front of a linked list
return the linked list of vertices

Tager ***O(V+E)*** tid da DFS tager O(V+E) tid og det tager O(1) tid at indsætte

Strongly connected components (kap 22.5 s. 617) (ill. 616)

1. Call DFS(G) to compute finishing times u.f for each vertex u
2. compute G^T
3. call DFS(G^T)but in the main loop og DFS consider the vertices in order of decreasing u.f
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

Minimum spanning tree (kap 23.1 s. 626)

Generic algorithm - while A does not form a spanning tree: find an edge (u,v) that is safe for A and add it.

A **cut** (S, V-S) of an undirected graph is a partition of V
An edge (u,v) **crosses** the cut (S, V-S) if u is in S and v is in V-S
A cut **respects** a set A of edges if no edge in A crosses the cut
An edge is a **light edge** if it's weight is the minimum of any edge crossing the cut
A **safe edge** is an edge that can be included in a minimum spanning tree (see p. 626)

Det er her algoritmerne er forskellige

Kruskal's algorithm (kap 23.2 s. 631) (ill. 632-633)

Bruger unionfind strukturen
Algoritmen ⇒ Sorter kanter efter vægt. For hver kant: hvis denne går mellem to *forskellige* sæt, så tilføj den til træet.
Tager tid ***O(E lg V)***

Prim's algorithm (kap 23.2 s. 634) (ill. 635)

Bruger en attribut v.key for at angive den mindste vægt for en connection til træet. (se midt s. 634)
Opfører sig lidt ligesom Dejkstra's algoritme
Tager tid ***O(E lg V)*** (kan dog forbedres vha. Fibonacci heaps)

Single-Source Shortest Path (kap 24 s. 643) (ill. 648)

Finder den korteste vej til alle knuder fra en given startknude

Initialisation (s. 648) ⇒ Hver knudes *d* og *pi* værdi sættes til infi og NIL. Startknudens distance, *s.d*, sættes til 0.
Relaxation (s. 649) (ill. 649) - Hver knude har en værdi v.d, med værdien for den korteste vej til knuden fra udgangspunktet.

Bellman-Ford algorithm (kap 24.1 s. 651) (ill. 652)

Initialiser grafen. Relax alle kanter |V| (antallet af knuder) gange. Hvert gennemløb må mindst relaxe 1 knude ordentligt og efter |V| gennemløb må alle knuder være relaxet. Tjek derefter for negative cykler.

Returnerer false hvis der er negative cykler ellers true.
Relax sætter attributter d og pi ved hver knude.
Tager tid ***O(VE)***

Single-source Shortest Path in directed acyclic graphs (s. 655) (ill. 656)

1. Udfør topologisk sortering af G
2. Initialiser grafen (s. 648)
3. Relax langs udgående kanter for hver knude taget i topologisk rækkefølge

Dijkstra's algorithm (s. 658) (ill. 659)

Initialiser grafen. Tag knuden med den laveste d værd. Relax langs alle udgående kanter.
Tager tid ***O(E lg V)*** (kan forbedres med Fibonacci heaps)

All-Pairs Shortest Paths (kap 25 s. 684)

Giver en matrix med den den korteste vej mellem alle knuder

Faster-all-pairs-shortest-paths (s. 691) (ill. 960)

Induktion i antallet af brugte edges
Tager tid ***$O(n^3 \lg n)$*** (n = |V|)

Floyd-Warshall algorithm (kap 25.2 s. 695) (ill. 694,696)

Induktion i brugte knuder. Brug kun knuder markeret mindre end k og vis da k+1
Tager tid ***O(E lg V)***

Maximum Flow Networks (kap 26 s. 708)

Ford-Fulkerson (kap 26.2 s. 724) (ill. 726,727,728)

Tager tid ***O(E |f*|)*** (f* er det maksimale flow i the residual network)

Edmonds-Karp (kap 26.2 s. 727)

Brug BFS søgning til at finde en forbedrende sti i the residual network.
Tager tid ***O(V E^2)***

Computing Patterns in Strings (kopiark)

Suffix array ⇒ alle suffikser for en streng sorteret i alfabetisk rækkefølge. Disse angives i en række tal der angiver hvor suffikset starter (e.g. σ_g = 34152)

Trie ⇒ Et træ der repræsentereren mængde strenge. Matching tager tid O(|Σ| |x|)

- Every internal node of T has at most |Σ| children
- T has s internal nodes *(s is the number of strings)*
- The height of T is equal to the length of the longest string in S
- The number of nodes of T is O(n) *(n is the total length of all strings)*

Compressed trie ⇒ each node represents a string instead of a character. The size of the tree is proportional to the number of strings instead of their total length.

Suffix trees ⇒ en komprimeret trie over suffikser for en given streng.
Hver knude angives som et par af indexer i strengen (i,j) som repræsenterer X[i..j]

- Matching tager tid o(m) for et konstant størrelse alphabet hvor m er længden af den streng der matches

Rabin-Karp algorithm (kap 32.2 s. 993) (ill. 992)

Bruger noget modolu-beregning til at finde mulige matches hurtigt og tjekker derefter disse (mulige) matches på den traditionelle måde.
Preprocessing ***O(m)***
Matching ***O((n-m+1)m)***

Knuth-Morris-Pratt (kap 32.4 s. 1005) (ill. 1005)

Bruger noget præfixberegning af den matchede streng til at udelukke nogle matches og springer disse over.
Preprocessing ***O(m)*** *(det er her denne vinder over FA'er O(m |Σ|))*
Matching ***O(n)***