SAMMENDRAG ALGDAT

Basis:

- Algoritme: En algoritme beskriver utvetydig hvordan et problem kan løses.
- Datastruktur: Algoritmen lagrer data i (og henter data fra) datastrukturer. Datastrukturer brukes altså til å organisere de dataene algoritmen skal jobbe med.

Kjøretid

$$O(\log n) \le O(\sqrt{n}) \le O(n) \le O(n \log n) \le O(n^2) \le O(n^3) \le O(2^n)$$

- Inputen til en viss algoritme består av en liste med n tall og et enkelt heltall k. Kjøretiden $\Theta(nk)$ er eksponentiell i forhold til *inputstørrelsen*. k angir ikke antall elementer i inputen, men er selv en del av inputen. Dermed må vi finne et mål på størrelsen til k. Det er rimelig å bruke antallet bits som kreves for å representere k. Dette er jo lik *floor*(lg k) + 1. La oss bruke w som betegnelse på størrelsen til k, og la oss for enkelhets skyld sette $w = \lg k$. Da er det altså w som angir k sitt bidrag til inputstørrelsen, og vi har $nk = n * 2^{\lg k} = n * 2^w$, som er eksponentielt i w og dermed eksponentielt i inputstørrelsen. Denne typen kjøretid kalles pseudopolynomisk.
- Gitt rekurrensen T(n) = T(n/3) + T(n/2) + n, T(1) = 1. Høyden til rekursjonstreet vil være dominert av T(n/2)-leddet, slik at høyden vil være $\log_2(n) + 1$ (det krever flere delinger for å komme til 0 når du bare deler på 2 enn når du deler på 3).

Master method

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad where\ \ a \ge 1, b > 1$$

Case 1

$$f(n) \in O\left(n^{\log_b a - \epsilon}\right) \qquad T(n) \in \Theta\left(n^{\log_b a}\right).$$

Case 2

$$f(n) \in \Theta\left(n^{\log_b a} log^k(n)\right) \qquad T(n) \in \Theta\left(n^{\log_b a} \log^{k+1}(n)\right).$$

Case 3

$$f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right) \qquad af\left(\frac{n}{b}\right) \le cf(n) \qquad T(n) \in \Theta(f(n)).$$

Divide and conquer

- Dersom man ikke har overlappende delproblemer.

Grådig

- Lokalt optimale valg er globalt optimale; optimal løsning avgjøres ut fra hva som virker best der og da.
- Løses som regel ovenfra og ned; man tar et valg og ender opp med et mindre delproblem. Altså: Begynner med største problem og deler seg nedover.

- Dersom løsning av ett delproblem ikke avhenger av løsningene av andre delproblemer. Motsatt innebærer at man ikke kan ta et lokalt optimalt valg => benytt DP.
- Optimal Substructure (innebærer rekursiv struktur)

Dynamisk programmering

- Optimal Substructure (innebærer rekursiv struktur)
- Overlapping Subproblems (Hvis ikke: D&C)
- Memoization (må være rekursiv)
- Løses nedenfra og opp; man løser større og større delproblemer ut i fra de mindre delløsningene. Altså: begynner på minste problem og bygger seg oppover.
- Løsning av et delproblem avhenger av andre delproblem (hvis ikke: grådig)

Sorteringsalgoritmer

Algoritme	Kjøretid					
	Best	Average	Worst	Memory	Stable	Method
Insertion-sort	O(n)	O(n^2)	O(n^2)	O(1)	Y	Insertion
Mergesort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	Y	Merging
Heapsort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	N	Selection
Quicksort	O(nlogn)	O(nlogn)	O(n^2)	O(nlogn)	N	Partitioning
Bubble-sort	O(n)	-	O(n^2)	O(1)	Y	Exchanging
	Best	Average	Worst	Memory	Stable	
Bucket-sort	O(n)	O(n)	O(n^2)	O(2^k)	Y	
Counting-sort	O(n+2^k)	O(n+2^k)	O(n+2^k)	O(n+2^k)	Y	
Radixsort	O(k*n)	O(k*n)	O(k*n)	O(n)	Y	

Counting sort

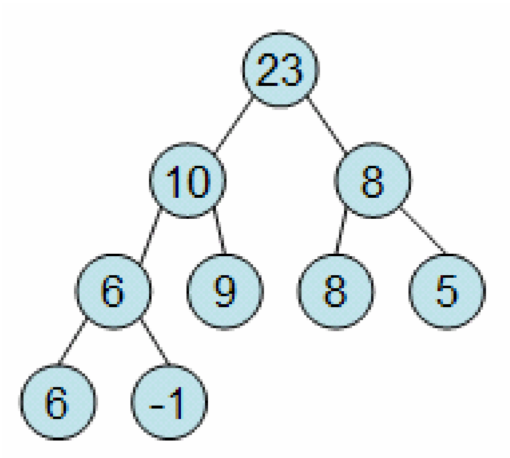
- Sorterer [0..k], $k \leq \text{floor}(\lg n)$

Mergesort

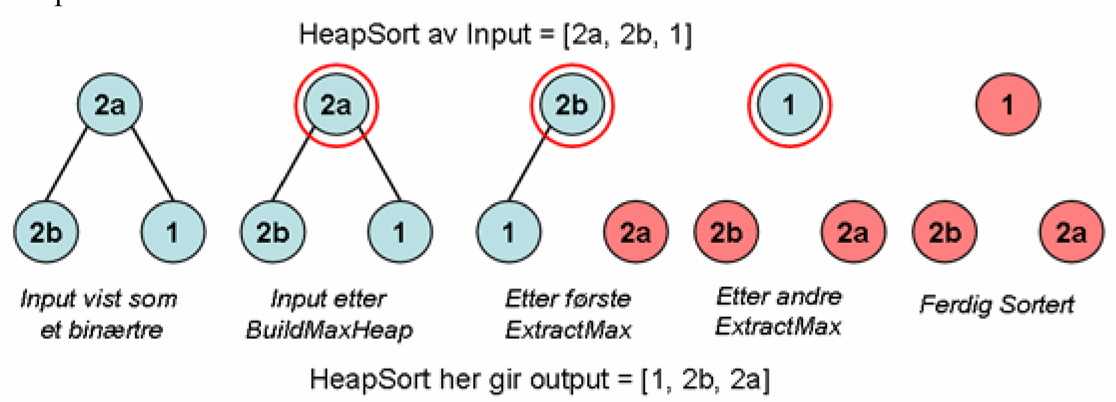
- Merging, $\Theta(n)$, merging to sorted arrays

Heapsort

- Max-Heapify, $O(\lg n)$, maintaining max-heap property
- Build-Max-Heap, $O(n)$, produce a Max-Heap for unordered array
- Heap prosedyrer, $O(\lg n)$
- Høyde i heap er alltid $\Theta(\log n)$



- Heapsort er ikke stabil:



Comparison sorts

- $\Omega(n \lg n)$ comparisons
- Can be viewed abstractly it terms of decision trees. n! permutations of n elements appear as leaves. A binary tree has no more than 2h leaves. $n! \leq 2^h \Rightarrow n = \Omega(n \lg n)$

Quicksort

- Partition, $\Theta(n)$, $A[1..k - 1] \leq pivot \leq A[k + 1..n]$

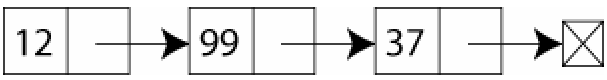
Linear sorting

- Distinct elements

Randomized-select

- n^{th} smallest element of an array
- based on quicksort, partition
- Det som står til vestre for det n-te minste tallet er mindre enn eller lik dette tallet som følge av bruk av Partition
- $\Theta(n)$
- D & C

Linked list



- List-search, $\Theta(n)$, on key
- List-insert, $\Theta(1)$
- List-delete, $\Theta(n)$

Matrix-chain multiplication

- DP
- p x q, q x r => pqr multiplications

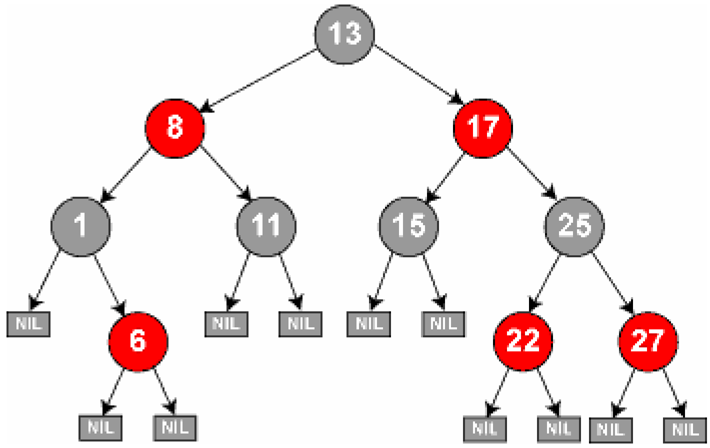
Longest common subsequence

- DP
- Gitt to seksvenser X og Y ønsker vi å finne den lengste sekvensen.

Hash tables

- When the set K of keys stored in a dictionary is much smaller than the universe U of possible keys, a hash table require much less storage
- Elements stored in slot h(k), h = hash function for key k
- Collision → chaining. Put all elements hashed to the same slot in a linked list.

Red-black trees

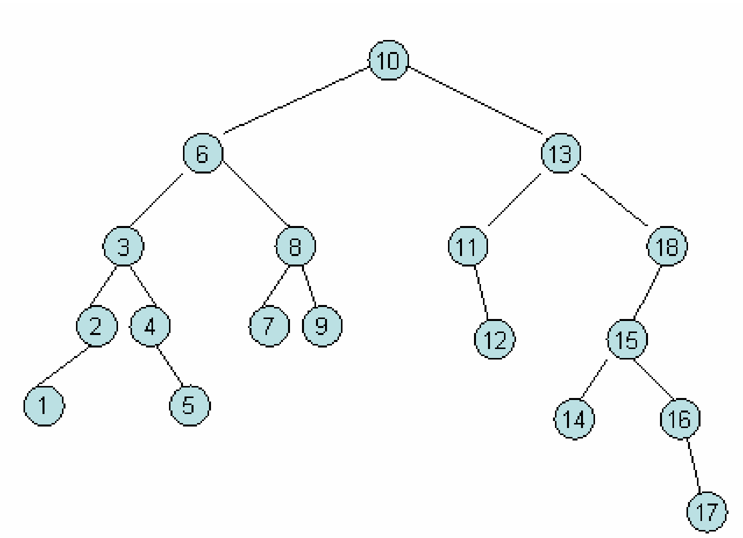


- A node is either red or black
- The root is black.
- All leaves are black
- Both children of every red node are black
- All paths from any given node to its leaf nodes contain the same number of black nodes.
- Height $\leq$ 2lg (n + 1). No path is more than twice as long as any other.
- Approximately balanced
- Er et binært søketre

Activity-selector

- Max-size subset og mutually compatible activities
- Recursive
- $\Theta(n)$

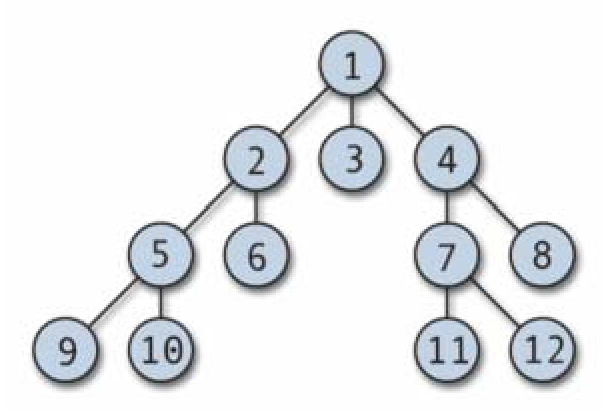
Binary Search tree



- O(log n)
- Postorder traversering: Venstre bein skrive ut verdiene på seg selv og nedover i treet. Deretter høyrebeinet og tilslutt sin egen verdi.
- Preorder traversering: Skriver ut noden som blir kalt først, så venstrebeinet og deretter høyrebeinet.
- Inorder traversering: Først venstre bein, så sin egen verdi og tilslutt høyre bein.

- Høyde = O(n)
- Antall løvnoder = antall interne noder – 1
- Kan skrive nodene ut sortert i O(n), ved bruk av inorder traversering.

Breadth-first search



- $\Theta(V + E)$
- Finding the shortest path between two nodes *u* and *v* (in an unweighted graph)
- Finding all connected components in a graph.

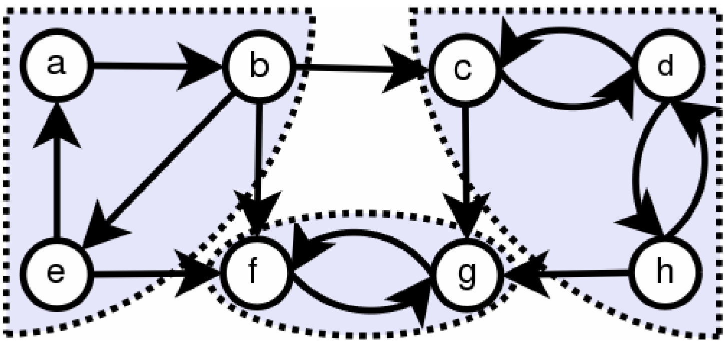
Depth-first search

- $\Theta(V + E)$
- Edges:
  - Tree edge: White
  - Back Edge: Gray
  - Forward or cross edge: Black

Topological sort

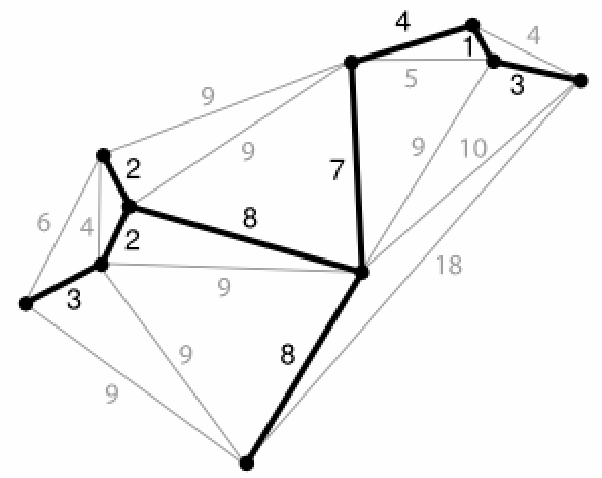
- DAG
- Først DFS, så skrive ut synkende etter slutttid
- $\Theta(V + E)$
- Lineær ordning av nodene

Strongly connected component



- For every pair of vertices u and v there is a path from u to v and a path from v to u

Minimum spanning tree



- MST er alltid unikt
- Light edge is the edge with minimum weight crossing a cut. The light edge must be a safe edge.
- For å bevise: Lag et kutt og forklar at man må velge letteste kant eller er det ikke MST.

Kruskal

- Velg alltid den lavest vektede kanten fra grafen (Grådig)
- O(E lg V)

Prim

- Velg alltid korteste vei fra en av de nodene du er eller har vært i (Grådig)
- O(E lg V)

Prim og Kruskal finner alltid spenntrær med minimal dyreste kant, fordi det ved hvert valg velges den kanten med lavest kostnad som knytter S til V - S.

Single-source shortest paths

Bellman-Ford

- Edge weights may be negative.
- O(VE)

Dijkstra

- Non-negative edge weights
- Find the shortest paths from a given start point *s* to *all other* nodes.
- $O(V^2)$, $O((E+V)\log V)$ with binary heap (sparse graph)

DAG Shortest Path

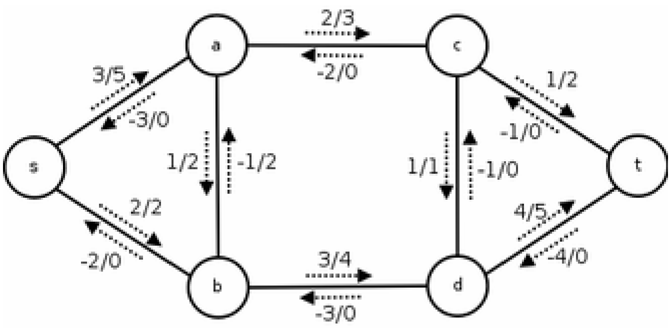
- Non-negative weight cycles
- Θ(V + E)

All pairs shortest path

Floyd-Warshall

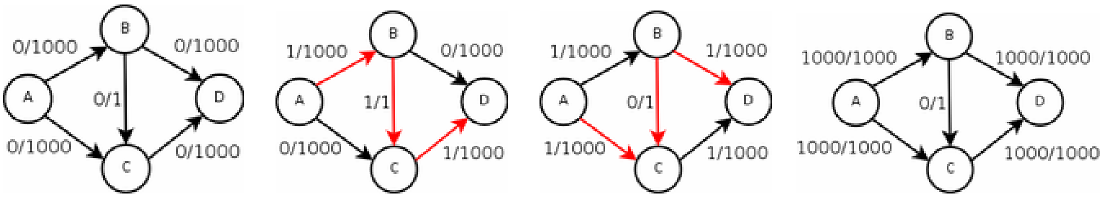
- $O(V^3)$
- No negative weight cycles
- DP
- Går først via 1, så 1 og/eller 2, så 1 og/eller 2 og/eller 3, …

Flow Network



- **Capacity constraints**:
  $$f(u,v) \le c(u,v)$$
  The flow along an edge cannot exceed its capacity.
- **Skew symmetry**:
  $$f(u,v) = -f(v,u)$$
  The net flow from *u* to *v* must be the opposite of the net flow from *v* to *u*.
- **Flow conservation**
  $$\sum_{w \in V} f(u,w) = 0$$
  The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.

Ford-Fulkerson



- Maximum flow in a flow network
- Start at 0, and increase flow by finding augmenting paths
- O(VE^2) (avhenger av implementasjonen)
- The augmenting paths can be found with a depth-first-search.
- Finner min-snitt ved å sette nodene som besøkes i siste iterasjon “til venstre” for snittet.
- The residual capacity of an edge is $c_f(u,v) = c(u,v) - f(u,v)$.
- Residualnettverket består av kanter som kan ha mer flyt.
- Ideer:
  - Residual networks
  - Augmenting paths
  - Cuts

Edmonds-Karp

- Identical to the Ford-Fulkerson, except that the search order when finding the augmenting path is defined.
- The path found must be the shortest path which has available capacity. This can be found by a BFS.
- O(VE2)

Linear Programming

- Infeasible: ingen gyldige løsninger.
- Unbounded: ikke en entydig optimal løsning.

Standardform

- A linear function to be maximized
  e.g. maximize $c_1x_1 + c_2x_2$
- Problem constraints of the following form
  $a_{11}x_1 + a_{12}x_2 \leq b_1$
  $a_{21}x_1 + a_{22}x_2 \leq b_2$
  $a_{31}x_1 + a_{32}x_2 \leq b_3$
- Non-negative variables
  $x_1 \geq 0$
  $x_2 \geq 0$

4 ting som kan gjøre at det ikke er på standardform

1. Kan ha et minimeringsproblem (endre fortegn)
2. Kan ha likheter i stedet for ulikheter (Del opp i <= og >=)
3. Kan ha større-lik ulikheter (feil vei, gang med -1)
4. Kan finnes variabler som ikke er nedre begrenset av 0 (X ikke nedre begrenset, sett X = Y-Z, X og Y > 0)

NP-komplett

- P ⊆ NP
- NPC ⊆ NP
- P: Kan løses i polynomiell tid, O(n^k)
- NP: Kan **sjekkes** i polynomiell tid.
- NPC: Tilhører klassen NP fordi det kan sjekkes i polynomisk tid, samt at de er minst like "harde" som andre problemer i NP, men ikke løselig i polynomiell tid.

Vise at et problem er NPC

- Vis at problemet er i NP. Vis at problemet er *NP-hard*, altså minst like vanskelig som alle andre problemer i NP.
- Finn et problem i NPC som kan reduseres til dette problemet .
- Hvis et problem A er reduserbar til et problem B i polynomisk tid og problem B kan løses i polynomisk tid, kan også A løses i polynomisk tid. → P(B)->P(A). Og motsatt.

Knapsack problem

- Given a set of items, each with a cost and a value, then determine the number of each item to include in a collection so that the total cost is less than some given cost and the total value is as large as possible.

Travelling salesman problem

- Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city?
- Find a Hamiltonian cycle with the least weight.

SAT

- Given a boolean expression, is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true?

Graph coloring

- Is there a coloring which uses at most *k* colors?
- On planar graph, 2-coloring and 4-colorring is in P, but 3-coloring is NPC.