# Gale-Shapley – Stable marriage (Labb 1)

Initialize each person to be free.

while (some man is free and hasn't proposed to every woman) {

Choose such a man m

w = 1st woman on m's list to whom m has not yet proposed

f (w is free)

assign m and w to be engaged

else if (w prefers m to her fiancé m')

assign m and w to be engaged, and m' to be free

else

w rejects m

}

**0(n^2)**

# Breadth-first search (BFS) (Graphs) – Word ladders (Labb 2)

* Undersöker utåt från startnoden alla möjliga vägar, lägger till noder ett lager åt gången.

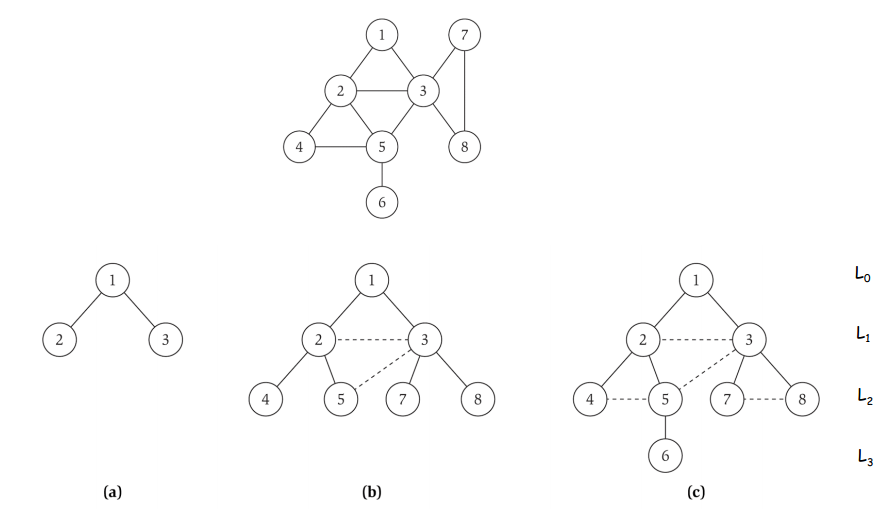
**BFS algorithm**

L0 = { s }.

L1 = all neighbors of L0

L2 = all nodes that do not belong to L0 or L1, and that have an edge to a node in L1

Li+1 = all nodes that do not belong to an earlier layer, and that have an edge to a node in Li



**O(n+m)**  (n-nodes, m-arcs) eller **O(n^2)**

# Greedy algorithms – Spanning USA (Labb 3)

* Betrakta jobben i någon viss ordning/sortering.
* Ta ut varje jobb för att jämföra det med de som redan är tagna. Beroende på vilken resultat som man vill åt sorteras jobben på olika sätt.

**Prim’s** or **Kruskal’s** algorithm (eller **Dijkstras algoritm**).

**Kruskal’s algorithm**

* För att hitta ett minimalt uppspännande träd T i den sammanhängande grafen G
* Upprepa tills T innehåller alla noder i G
  + Låt v vara den kortaste kanten i G som inte märkts som förbrukad
  + Märk v som förbrukad
  + Om v inte bildar en cykel i T
    - Lägg v till T
* T är ett minimalt uppspännande träd

**O(mlogn)** (n-nodes, m-arcs)

# Divide and conquer – Closest pair in a plane (Labb 4)

* Dela upp problemet i mindre delar
* Lös varje del rekursivt.
* Kombinera lösningarna till subproblem till en slutgiltig lösning.

Algorithm – Closest pair

* Divide: draw vertical line L so that roughly ½n points on each side.
* Conquer: find closest pair in each side recursively.
* Combine: find closest pair with one point in each side.
* Return best of 3 solutions.

**O(nlogn)**

Mergesort – O(nlogn)

# Dynamic programming - [Me: Gorilla or sea cucumber?](http://fileadmin.cs.lth.se/cs/Education/EDAF05/Labbar/cucumber.pdf) (Labb 5)

* Dela upp problemet till överlappande sub-problem
* Bygg upp svar till större och större sub-problem

**Sequence allignment**

Def. OPT(i, j) = min cost of aligning strings x1 x2 . . . xi and y1y2 . . . yj

Case 1: OPT matches xi-yj.

– pay mismatch for xi-yj+ min cost of aligning two strings x1x2. . . xi-1 and y1y2 . .yj-1

Case 2a: OPT leaves xi unmatched.

– pay gap for xi and min cost of aligning x1x2. . . xi-1 and y1y2. . . yj

Case 2b: OPT leaves yj unmatched.

– pay gap for yj and min cost of aligning x1x2 . . . xi and y1y2 . . . yj-1

**O(nm)** (n/m längden av strängarna som jämförs)

# Network flow – Railroad planning (Labb 6)

**Flow value lemma**. Let f be any flow, and let (A, B) be any s-t cut.

Then, the net flow sent across the cut is equal to the amount leaving s.

**O(m^2logC)**