Dynamic programming: recursive structure, repeated subproblems → memorize intermediate results to avoid re-computing them

LIS: 1 3 2 5 4 8 1 9 4

LIS in a portion of the array (up to i

LIS[0] = 1 → base case

up to and including element 0

LIS[i] = max(LIS[j] + 1 ; j < i and A[j] < A[i])

LIS: array; fill it left → right LIS[0...n]

up to index i

LIS 1 2 2 3

0 1 2

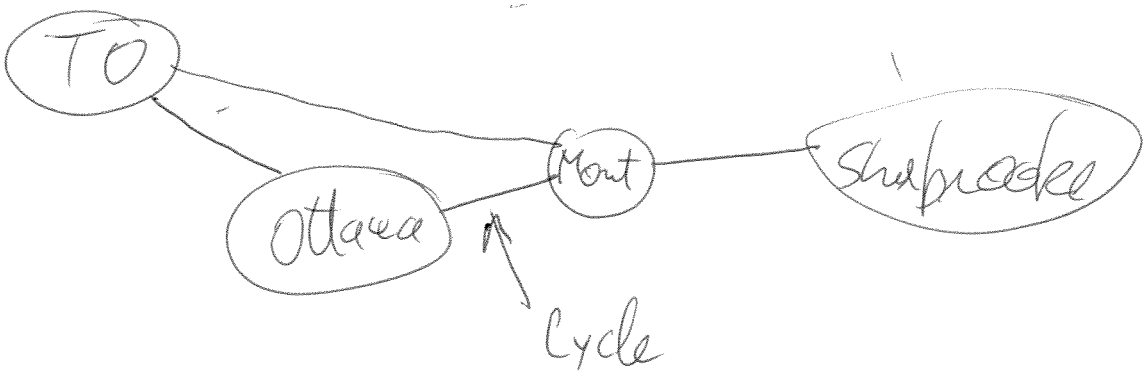
IndexLIS → indices of the corresponding seq.

```
Algorithm LongestIncreasingSubsequence (A, n)
LIS[0...n] (int)
for (int i = 0 to n)
    LIS[i] = -1 //initialize low for max
    for (int j = 0 to i-1)
        if (A[j] < A[i] and LIS[i] < LIS[j]+1)
                                  ↑ situation can be improved
            LIS[i] = LIS[j] + 1
return LIS[n];
```

=> $O(n^2)$ time ; Memory area $O(n)$

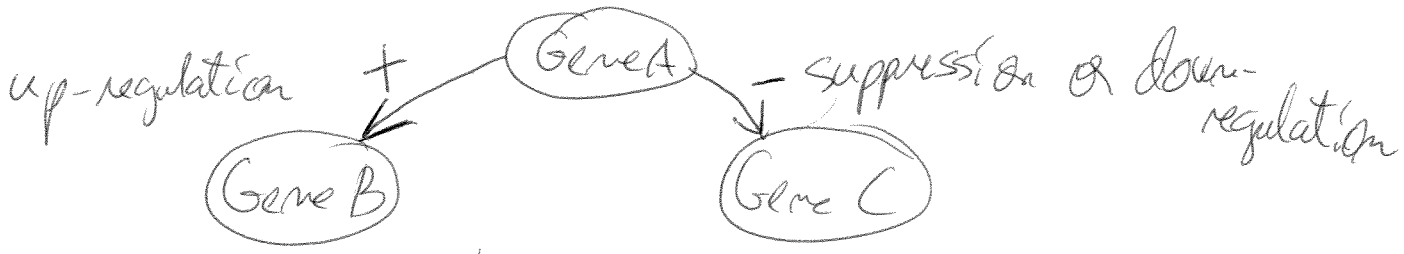
Graphs: Nodes (vertices) + Edges

E.g. Road Network

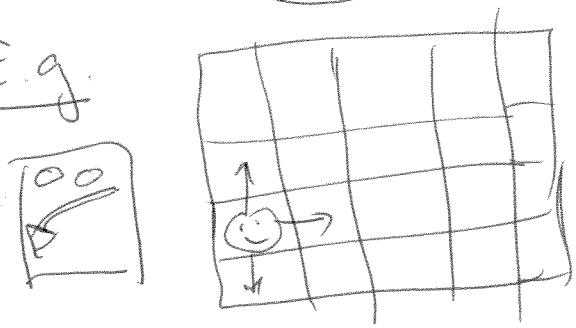


Trees are a special type of graph with no cycles.

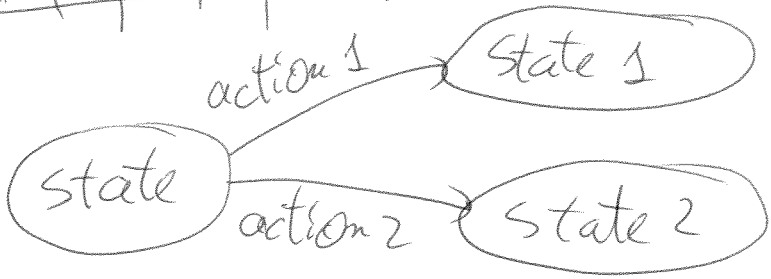
E.g. Gene regulatory network



E.g.



State of a character: position + what it owns



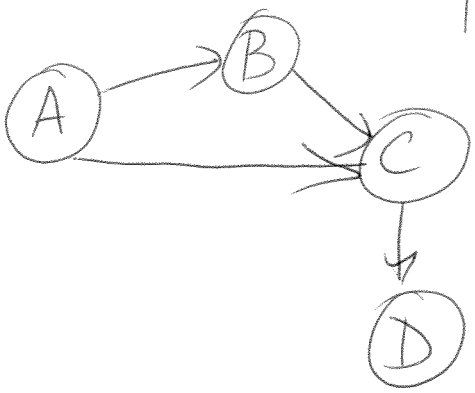
pos, own + value

Terminology

$G = (V, E)$ — E: pairs of nodes

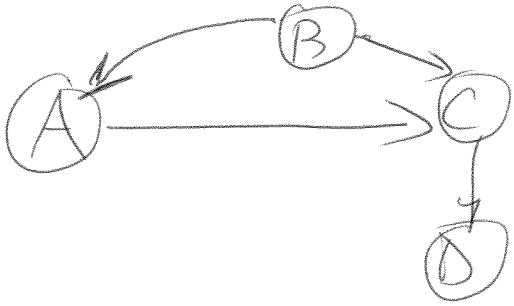
G: graph; V: vertices or nodes

Directed graph: "arrows" → edge is an ordered pair of vertices



$V = \{A, B, C, D\}$

$E = \{(A,B), (B,C), (A,C), (C,D)\}$



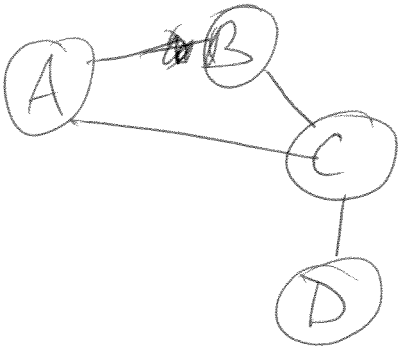
$\{(B,A), (B,C), (A,C), (C,D)\}$

Arrows restrict how one can go on edges

Undirected graph: no direction on edges

Edge is unordered pair of vertices

Can be traversed in either direction



(A,B) or (B,A) denote same edge.

Graph implementation

Vertex: Object that holds "appropiate" info (application-dependent)

Edge: Object, pair of vertices

$G = (V, E)$, V, E are <u>collections</u>

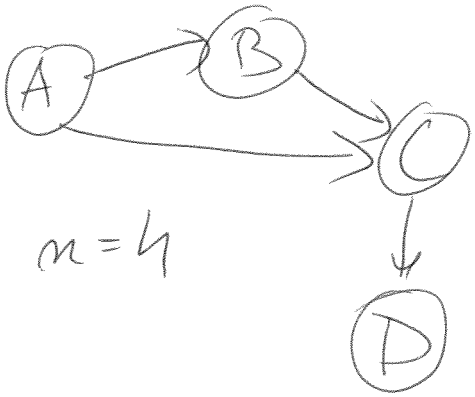
```
class Graph {
    Collection<Vertex> V;   ---> n
    Collection<Edge> E;     ---> O(n^2)
}
```

"Sparse" graph: has $\ll O(n^2)$ edges

E.g. trees have $O(n)$ edges

Finding a <u>path</u> in the graph → traversing edge collection (usually collection is linear so for large graphs this may be slow)

Alternative: Adjacency matrix



$n \times n = 4 \times 4$

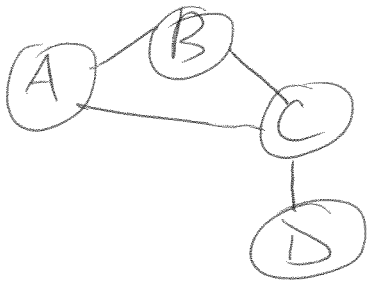
	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

0 at m_{ij} if there is no edge from $i \to j$

1 if there is an edge

n vertices => always n^2 (prev implementation could have less memory)

Prev. implementation: easy to det nr. edges out of a node but not number of edges into a node.



	A	B	C	D
A	0	1	1	0
B	1	0	1	0
C	1	1	0	1
D	0	0	1	0

Matrix is symmetric for undirected graphs!

Memory cost is constant, may be higher than in previous impl, but many interesting quantities / properties can be obtained off the graph.