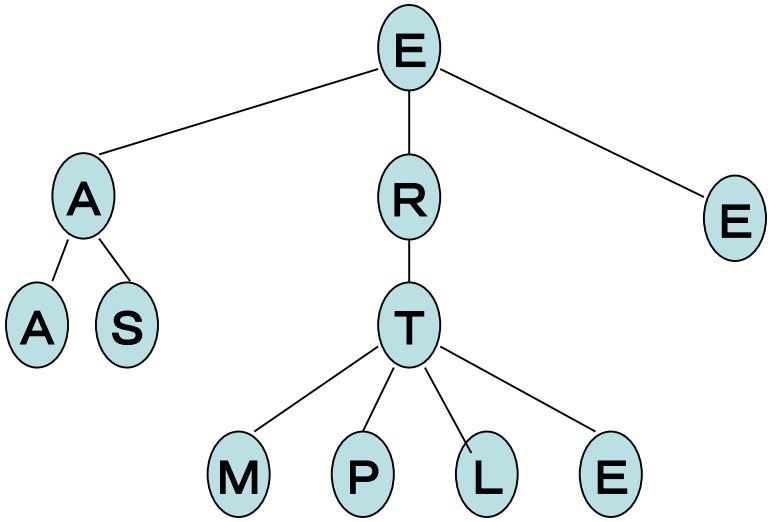
Problem formulation

Topics of this lecture

- Review of tree structure
- Review of graph structure
- Graph implementation
- State space representation
- Search graph and search tree

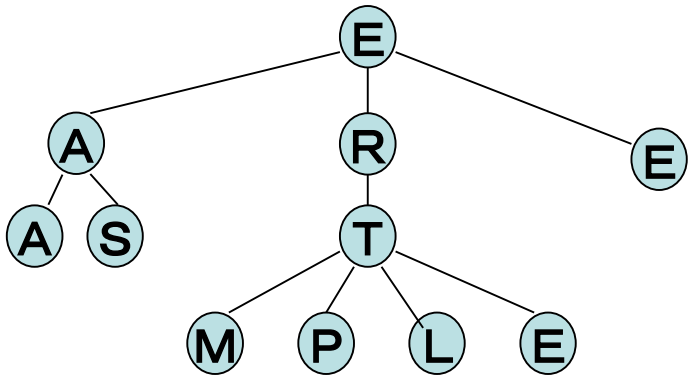
The Tree Structure

- Connected list, stack, and queue are 1-D data structures.
- Tree is a 2-D data structure (see right figure).
- Examples:
  - Family tree;
  - Tournament tree for a football game;
  - Organization tree of a company; and
  - Directory tree of a file management system.



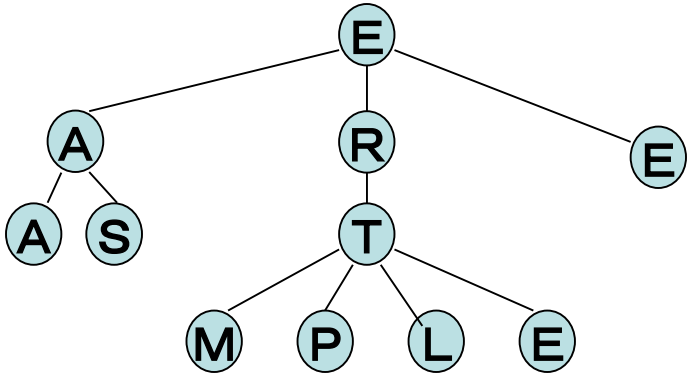
Useful terminologies

- A tree consists of a set of nodes and a set of edges.
- An edge is the connection between two nodes.
- There are different nodes:
  - Root: the first (top) node
  - Parent: a node above some other node(s) (connected)
  - Child: A node below another node
  - Internal (non-terminal) node: any parent node
  - Terminal (external or leaf) node: nodes that do not have any child



Useful terminologies

- Siblings: nodes that share the same parent
- Path: a sequence of **nodes** connected by edges
- Level of a node: number of **edges** contained in the path from the root to this node
- Height of the tree: maximum distance (number of edges) from the root to the terminal nodes



Multi-way tree and binary tree

- Multi-way tree or multi-branch tree
  - An internal node may have m (m>2) child nodes.
- Binary tree：
  - An internal node has at most 2 child nodes.
  - Binary tree is useful both for information retrieval (binary search tree) and for pattern classification (e.g. decision tree).
- Complete binary tree：
  - A binary tree in which every level, except possibly the last, is completely filled, and nodes in the last level are as far left as possible.

Tree Traversal（走查）

- Tree traversal is the process to visit all nodes of a tree, without repeating.
  - Example: print the contents of all nodes; search for all nodes that have a specified property (e.g. key), etc.
- Order of traversal
  - pre-order, in-order, post-order, and level-order
  - The first three correspond to depth-first search, and the last one to breadth-first search.

Pre-order, in-order, and post-order traversal

- Pre-order: Start from the root, visit (recursively) the current node, the left node, and the right node;
- In-order: start from the root, visit (recursively) the left node, the current node, and the right node.
- Post-order: start from the root, visit (recursively) the left node, the right node, and the current node.

```
Pre_order(struct node *t){
  visit(t);
  if(t->l != z) pre_order(t->l);
  if(t->r != z) pre_order(t->r);
}
```

Level-order traversal

- A tree can be traversed in level-order using a queue.
  - Put the root in the queue first, and then repeat the following:
  - Get a node (if any) from the queue, visit it, and put its children (if any) into the queue.

```
level_order(struct node *t) {
      enqueue(t);
      while (!queueempty()) {
         t = dequeue();
         visit(t);
         if (t->l != z) enqueue(t->l);
         if (t->r != z) enqueue(t->r);
}}
```

Graph structure

- Graph is a more general data structure.
- Formally, a graph is defined as 2-tuple

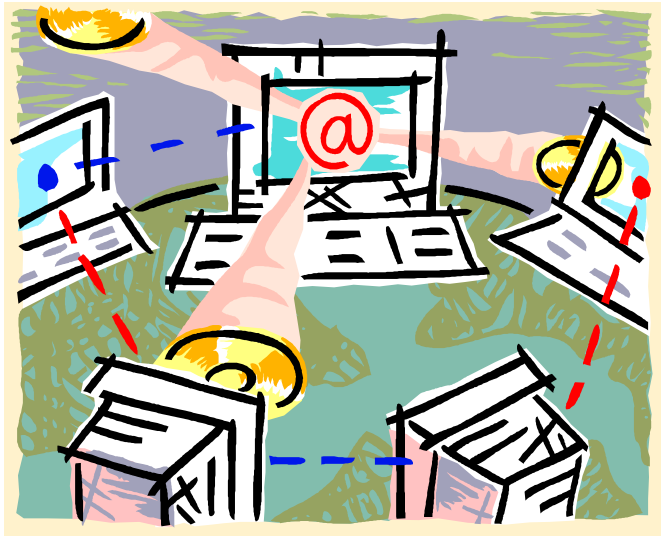
$$G = (V, E)$$

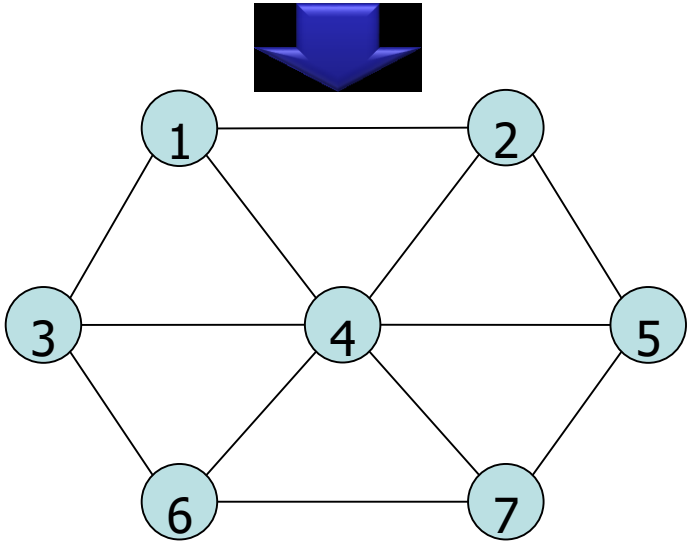
where V is a set of vertices or **nodes**; and E is a set of **edges**, arcs, or connections.

- Tree is a special graph without cycles.
  - Each node has one path from the root.
  - The path is unique.
  - All nodes are connected to the root.

Examples

- Computer networks
- Flight maps of airlines
- Highway networks
- City water / sewage networks
- Electrical circuits

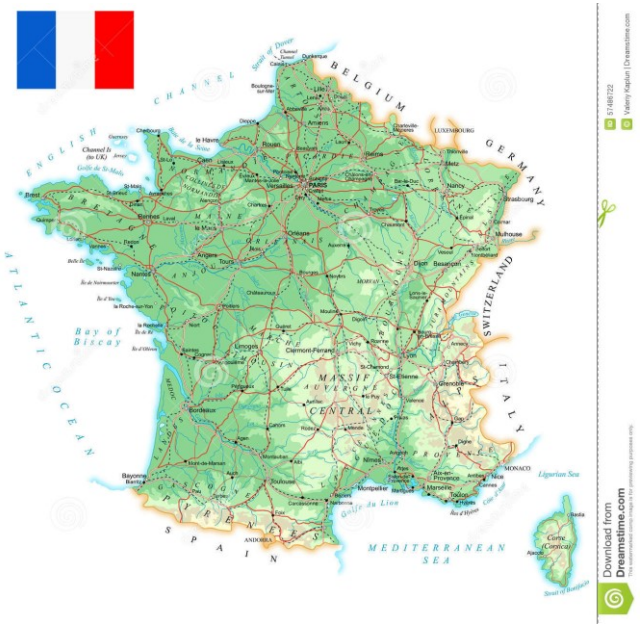




Graph is a convenient way for representing all these physical networks in digital (virtual) forms.

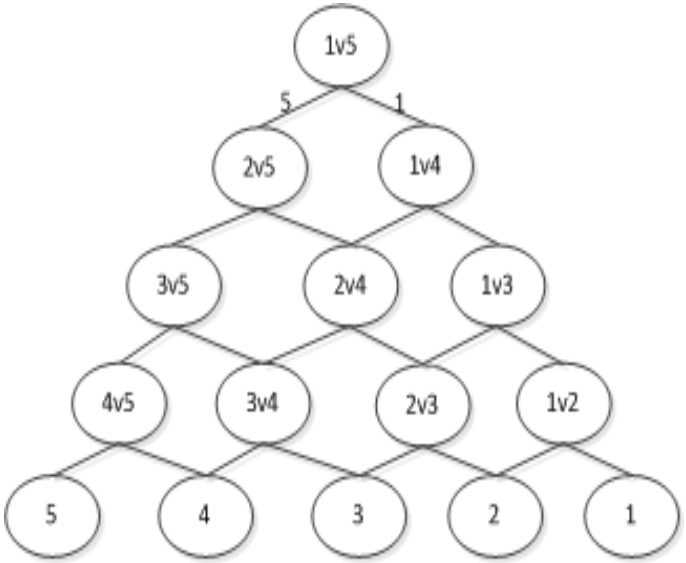
Useful terminologies

- Path: A sequence of nodes connected by edges.
- Simple path: A path in which all nodes are different.
- Cycle: A simple path with the same start and end nodes.
- Connected graph: There is a path between any two nodes.
- Connected component: A sub-graph which itself is a connected graph.
- Directed graph: The edges have directions (e.g. one way route).
- Undirected graph: The edges do not have directions (or do not care the direction).



Useful terminologies

- Weighted graph: Each edge has a weight (e.g. direct cost to move from one city to another).
- Node expansion: The process to get all child nodes of a node (useful for graph traversal or graph-based search).
- **Spanning tree: A tree that contains all nodes of a graph**.
- Directed acyclic graph (DAG): A special case of directed graph in which there is no cycles or loops.



DAG is often used to construct a larger classification system from many two-class classifiers (see the figure).

Graph Implementation - 1

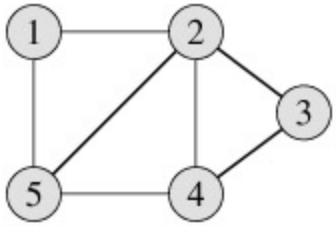
- Adjacency-list representation:
  - N=|V|: number of nodes
  - Define N lists Adj[0], Adj[1], …, Adj[N-1]
  - Adj[i] is the list for the i-th node. It contains all nodes connected to this node by an edge.
  - That is, for any node j contained in Adj[i], (i,j) belongs to the set E of edges.

Graph implementation - 2

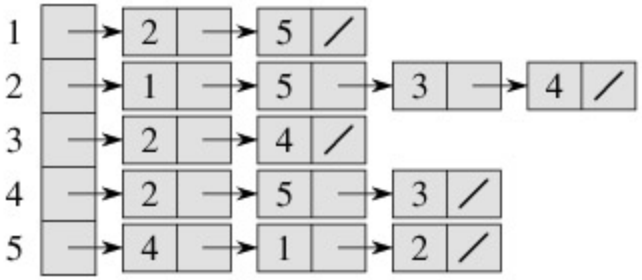
- Adjacency-matrix representation
  - N=|V|: number of nodes
  - Each node has a number (ID)
  - The adjacency-matrix A is an NxN matrix.
  - The (i,j)-th element a_{ij} is defined by

$$a_{ij} = \begin{cases} 1 & if (i,j) \in E \\ 0 & otherwise \end{cases}$$

Examples



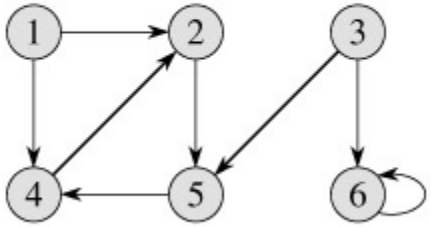
(a)



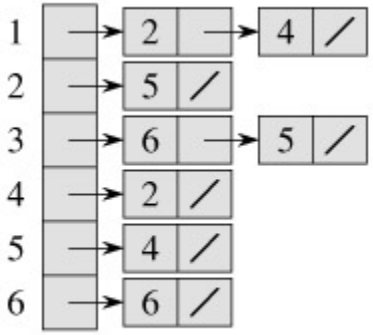
(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)



(a)



(b)

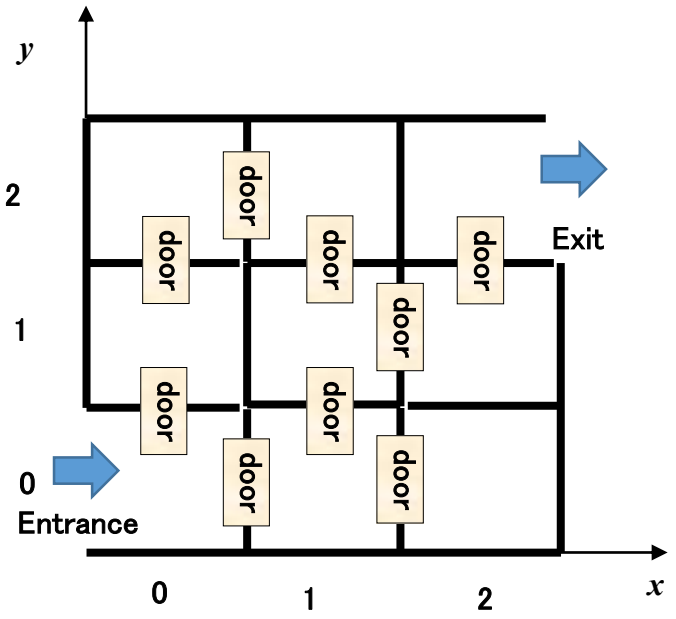
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

State space representation of AI problems

The maze problem

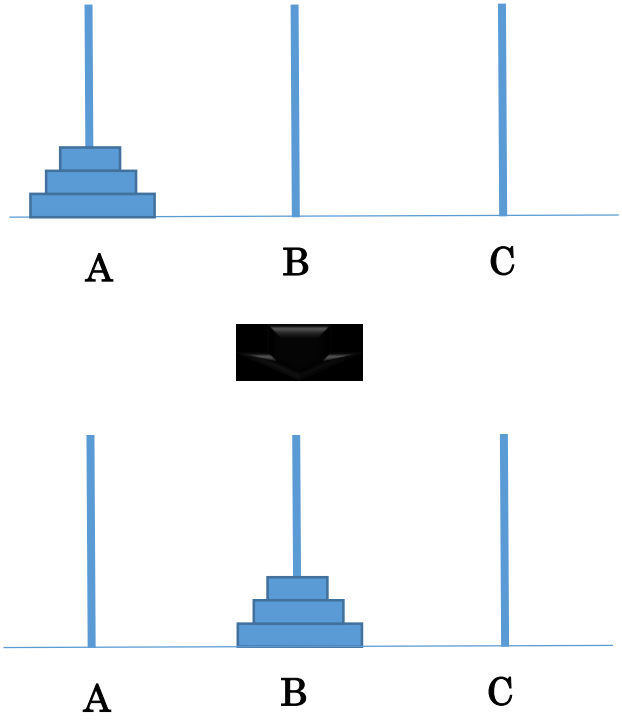
- Initial state: (0,0)
- Target state: (2,2)
- Available operations:
  - Move forward
  - Move backward
  - Move left
  - Move right
- Depends on the current state, the same operation may have different results.
- Also, an operation may not be executed for some states.



State space representation of AI problems

The Hanoi's tower

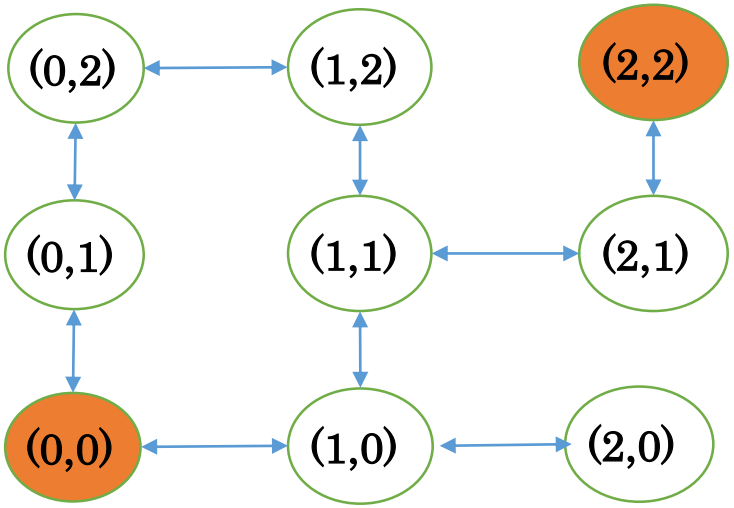
- Initial state: (123;000;000)
- Target state: (000;123;000)
- Available operations:
  - Move a disk from one place to another.
- Restriction:
  - A larger disk cannot be put on a smaller one.



Why state space representation?

- Any problem can be represented in the same form, formally.
- Any problem can be solved by finding the target state via state transition, using the available operations → Search problem!
- The results (i.e. the state transition process or the method for finding this process) can be re-used as knowledge.
- Problem: The computation cost can be large!

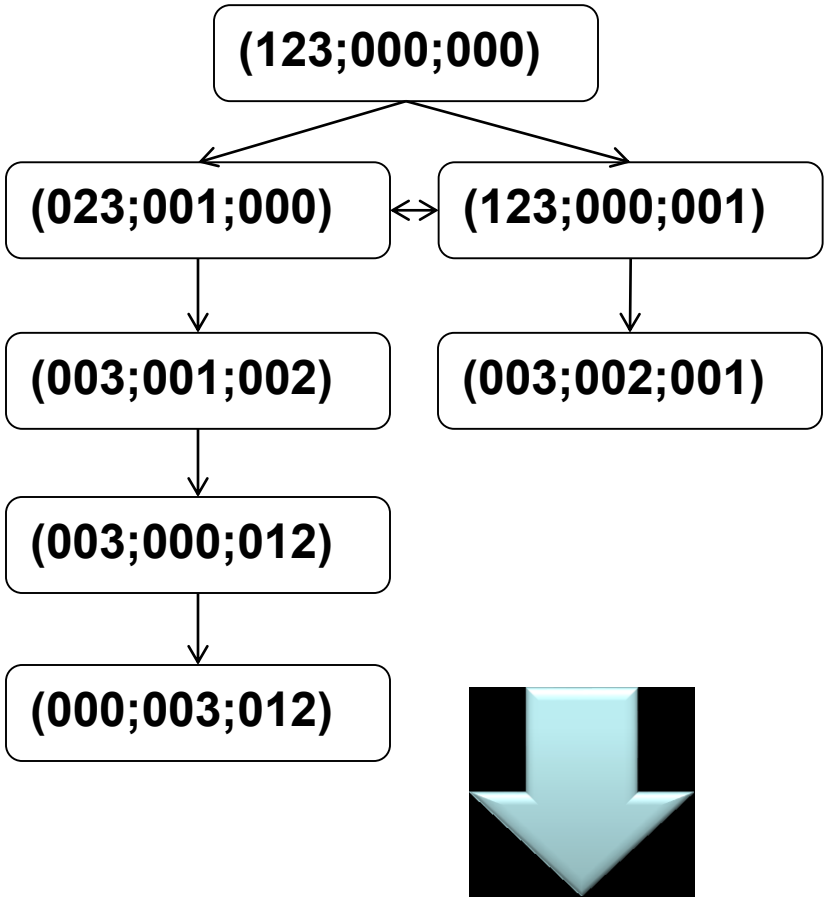
The maze problem → search graph



- To find the solution, we can just traverse the graph, starting from (0,0), and stop when we visit (2,2).
- **The result is a path from the initial node to the target node.**
- The result can be different, depends on the order of graph traversal.

Hanoi's tower problem → search tree

- Hanoi's tower can also be solved in a similar way, but more difficult if we do it manually because the number of nodes is much larger.
- Instead of using a search graph, we can use a search tree.
- That is, start from the initial node, expend the current node recursively, and stop when we find the target node.



Homework for lecture 2

- The goal of this homework is to review and understand the mechanism of depth-first search and breadth-first search. This is important for us to understand other search algorithms.
- The program to complete has the following functions:
  - Construct a graph based on nodes inputted from the keyboard or from a data file.
  - For a given initial node (state), traverse the graph, and
  - Print out the number (ID) of each visited node.
- Down-load the skeleton file, and complete it by following the instructions written in "**README.md**".
- Do not hesitate to ask the TA/SA or the teacher if you have any question.

How to down-load and use the skeleton file?

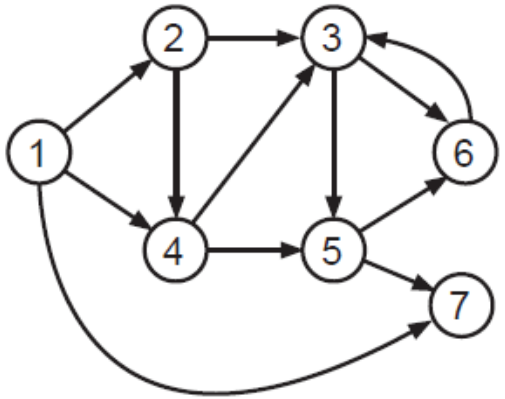
- From this lecture, each homework has a zip-file that can be down-loaded from the following link:
  - http://web-ext.u-aizu.ac.jp/~qf-zhao/TEACHING/AI/AI.html
  - The file name is ex_XX (XX = 02, 03,…)
- Un-compress the zip-file, you will get the following files:
  - README.md: Read this file first, to see what to do.
  - *.c, *.h: source files to modify and complete by you in the exercise class.
  - Makefile: “make” script (try to understand the meaning)
  - LICENSE: License information of the programs.
  - data_XX.txt: data used by the program (if any).
  - answer_XX.txt : the expected answer.
  - summary_XX.txt: Write a short summary about this homework in this file (Example: difficult points to finish this homework, and your solutions).

How to down-load and use the skeleton file?

- For example, for today's homework, down-load the file as follows:
  - cd ~/AI
  - wget http://www.u-aizu.ac.jp/~qf-zhao/TEACHING/AI/ex_02.zip
  - unzip ex_02.zip
- Do not forget to change the permission:
  - chmod -R 705 ./ex_02/
- Read the file "README.md" first, and confirm what to do.
- Compile and run the program using "make" command:
  - make test
  - > gcc -Wall -std=c99 -O2 -o ./a.out prog_02.c func.c queue.c
  - > ./a.out < data_02.txt > result_02.txt
  - > diff -y --suppress-common-lines answer_02.txt result_02.txt
- The last line finds the "difference" between the expected answer and your answer (in result_02.txt). There is no difference if your answer is correct.
- Do not forget to write a summary in the file summary_XX.txt. This file is a MUST!

Quizzes of Today

- What are the initial state and target state of the maze problem?
- What are the two sets for defining a graph?
- For the graph given on the right, answer yes (if correct) or no (if wrong) for the following sentences:
  - This is a connected graph ( )
  - This is an undirected graph ( )
  - This is a weighted graph ( )
- Find a spanning tree for this graph.



Spanning tree