Introduction to Artificial Intelligence

English practicals 2: Problem solving and search

Marika Ivanová

Department of Theoretical Computer Science and Mathematical Logic (KTIML)
Faculty of Mathematics and Physics

February 22th 2022

Selected quiz questions

- Explain the difference between atomic and factored representations of states using N-queens problem.

Selected quiz questions

- Explain the difference between atomic and factored representations of states using N-queens problem.

  atomic: non-divisible, start state, transition model, goal node(s)

  factored: vector of values (positions of queens)

Selected quiz questions

- Explain the difference between atomic and factored representations of states using N-queens problem.

  atomic: non-divisible, start state, transition model, goal node(s)

  factored: vector of values (positions of queens)
- What is the difference between search node and state? Is it the same?

Selected quiz questions

- Explain the difference between atomic and factored representations of states using N-queens problem.

  atomic: non-divisible, start state, transition model, goal node(s)

  factored: vector of values (positions of queens)

- What is the difference between search node and state? Is it the same?

  State - representation of physical configuration. Search node - data structure, two different search nodes may correspond to one state

Selected quiz questions

Selected quiz questions

- Discuss various abstractions for N-queens problem. Hint: look at the size of search space.

Selected quiz questions

- Discuss various abstractions for N-queens problem. Hint: look at the size of search space.

  basic:

  1. States: coordinates of queens on board
  2. Initial state: empty board
  3. Goal state: test that all queens are on the board and do not attack each other
  4. Search space size: $(n \times n)^n$

Selected quiz questions

- Discuss various abstractions for N-queens problem. Hint: look at the size of search space.

  basic:

  1. States: coordinates of queens on board
  2. Initial state: empty board
  3. Goal state: test that all queens are on the board and do not attack each other
  4. Search space size: $(n \times n)^n$

  queens initially assigned to columns:

  1. States: row number of each queen
  2. Initial state: empty board
  3. Goal state: test that all queens are on the board and do not attack each other
  4. Search space size: n^n

Tree vs Graph Search

Tree search:

Graph search:

Tree vs Graph Search

Tree search:

- Possible duplicates (multiple nodes with identical state)

Graph search:

- Each state visited at most once (closed list)

Tree vs Graph Search

Tree search:

- Possible duplicates (multiple nodes with identical state)
- Search nodes correspond 1:1 to paths from initial state

Graph search:

- Each state visited at most once (closed list)
- Search nodes correspond 1:1 to reachable states

Tree vs Graph Search

Tree search:

- Possible duplicates (multiple nodes with identical state)
- Search nodes correspond 1:1 to paths from initial state
- Search tree can be infinite even for finite state space

Graph search:

- Each state visited at most once (closed list)
- Search nodes correspond 1:1 to reachable states
- Search tree finite for finite state space

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:
  - **Completeness:**
    1. If a solution exists, is it guaranteed to be found?
    2. If no solution exists, will the algorithm terminate?

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:
  - **Completeness:**
    1. If a solution exists, is it guaranteed to be found?
    2. If no solution exists, will the algorithm terminate?
  - **Optimality:** Are the returned solutions always optimal

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:
  - **Completeness:**
    1. If a solution exists, is it guaranteed to be found?
    2. If no solution exists, will the algorithm terminate?
  - **Optimality:** Are the returned solutions always optimal
  - **Time complexity:**
    1. How much time is needed until termination?
    2. Worst case analysis
    3. Typically measured in the number of generated nodes

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:
  - **Completeness:**
    1. If a solution exists, is it guaranteed to be found?
    2. If no solution exists, will the algorithm terminate?
  - **Optimality:** Are the returned solutions always optimal
  - **Time complexity:**
    1. How much time is needed until termination?
    2. Worst case analysis
    3. Typically measured in the number of generated nodes
  - **Space complexity:**
    1. How much memory does the algorithm use?
    2. Worst case analysis
    3. Typically measured in the number of nodes stored at the same time

Criteria for evaluating search algorithms

- Strategies vary in the order in which nodes are selected for expansion
- Evaluating criteria:
  - **Completeness:**
    1. If a solution exists, is it guaranteed to be found?
    2. If no solution exists, will the algorithm terminate?
  - **Optimality:** Are the returned solutions always optimal
  - **Time complexity:**
    1. How much time is needed until termination?
    2. Worst case analysis
    3. Typically measured in the number of generated nodes
  - **Space complexity:**
    1. How much memory does the algorithm use?
    2. Worst case analysis
    3. Typically measured in the number of nodes stored at the same time
- Measures of time and space complexity:
  - b - maximum branching factor
  - d - shallowest depth of a solution
  - m - maximum depth of the search space

BFS vs DFS

BFS

DFS

BFS vs DFS

BFS

- Expand shallowest node in the frontier

DFS

- Expand deepest node in the frontier

BFS vs DFS

BFS

- Expand shallowest node in the frontier
- FIFO (queue)

DFS

- Expand deepest node in the frontier
- LIFO (stack)

BFS vs DFS

BFS

- Expand shallowest node in the frontier
- FIFO (queue)
- Complete for finite b (search space can be infinite)

DFS

- Expand deepest node in the frontier
- LIFO (stack)
- Complete only for finite search spaces

BFS vs DFS

BFS

- Expand shallowest node in the frontier
- FIFO (queue)
- Complete for finite b (search space can be infinite)
- Optimal for uniform cost edges (example?)

DFS

- Expand deepest node in the frontier
- LIFO (stack)
- Complete only for finite search spaces
- Not optimal (example?)

  Not optimal

BFS vs DFS

BFS

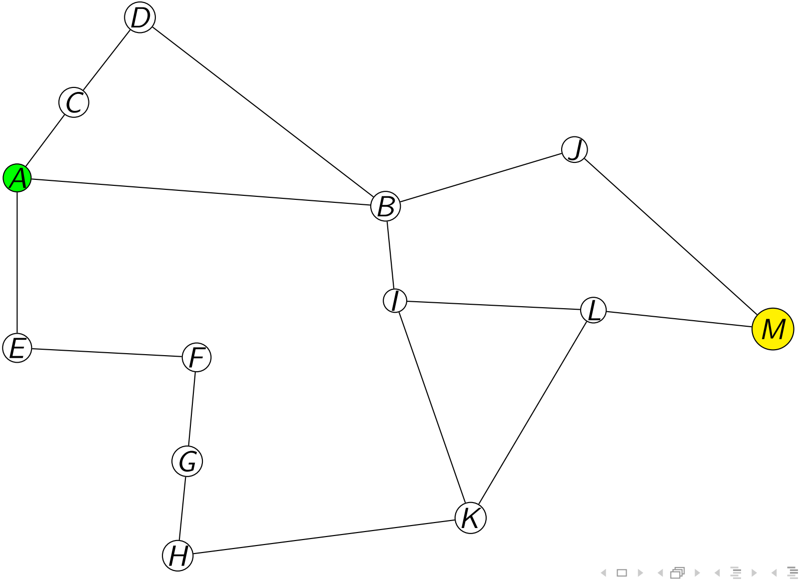
- Expand shallowest node in the frontier
- FIFO (queue)
- Complete for finite b (search space can be infinite)
- Optimal for uniform cost edges (example?)
- Time: $\mathcal{O}(b^d)$
- Space: $\mathcal{O}(b^d)$

DFS

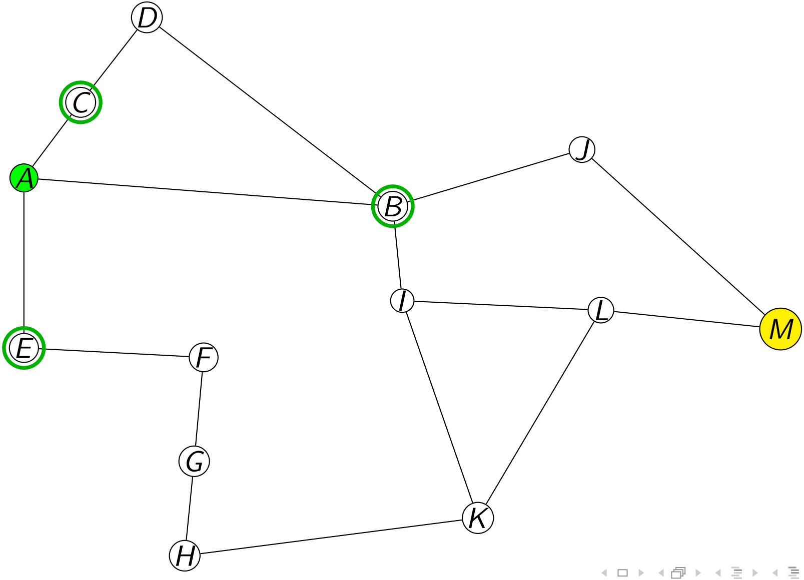
- Expand deepest node in the frontier
- LIFO (stack)
- Complete only for finite search spaces
- Not optimal (example?)

  Not optimal
- Time: $\mathcal{O}(b^d)$
- Space: $\mathcal{O}(bm)$

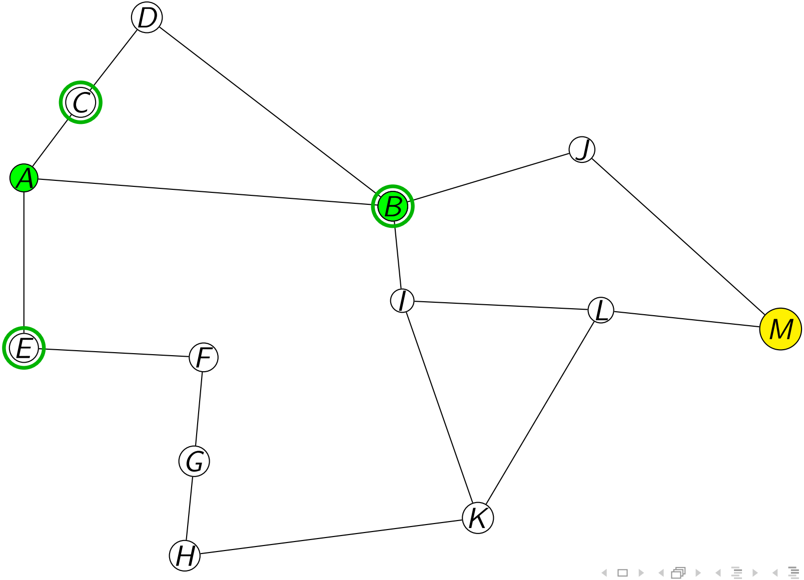
BFS



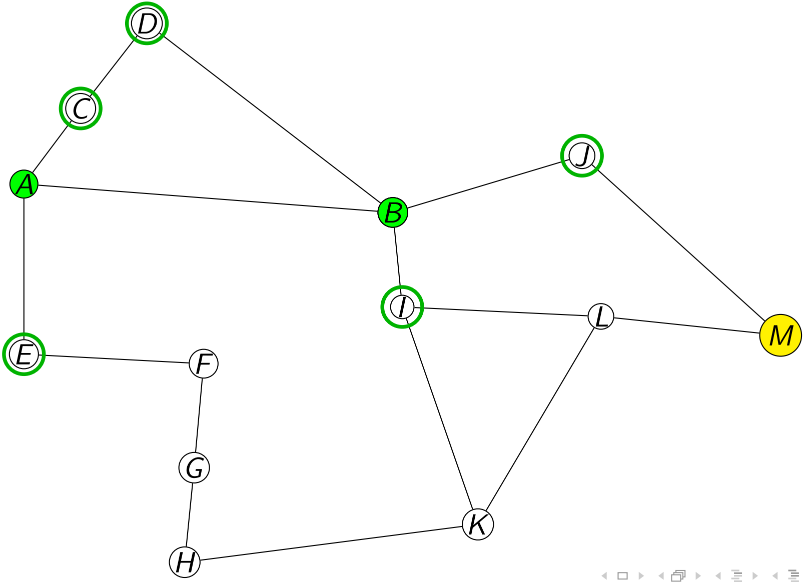
BFS



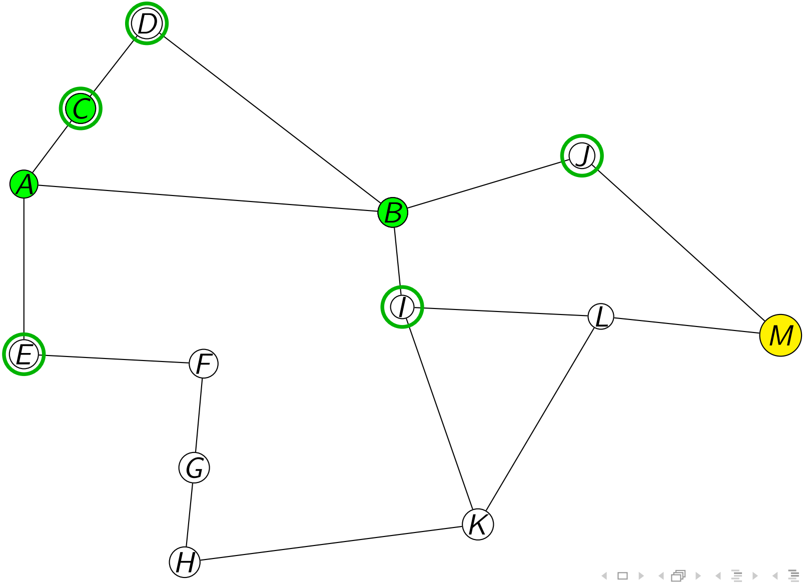
BFS



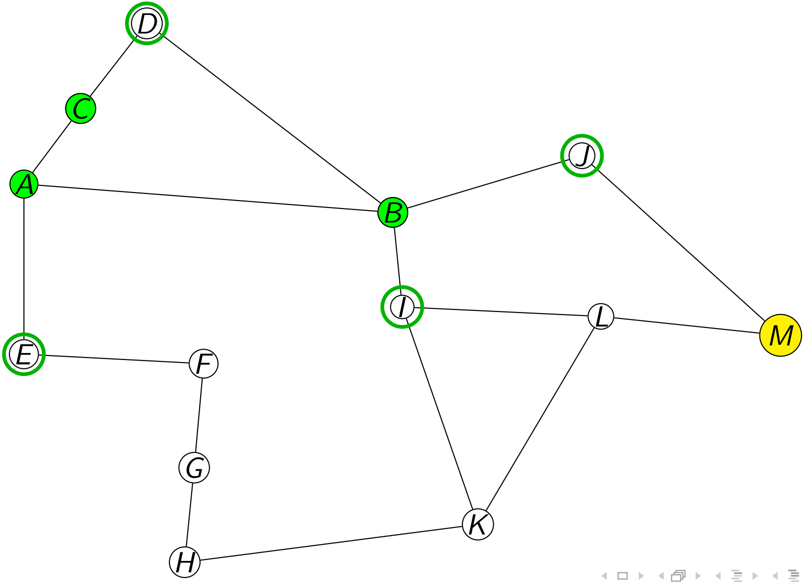
BFS



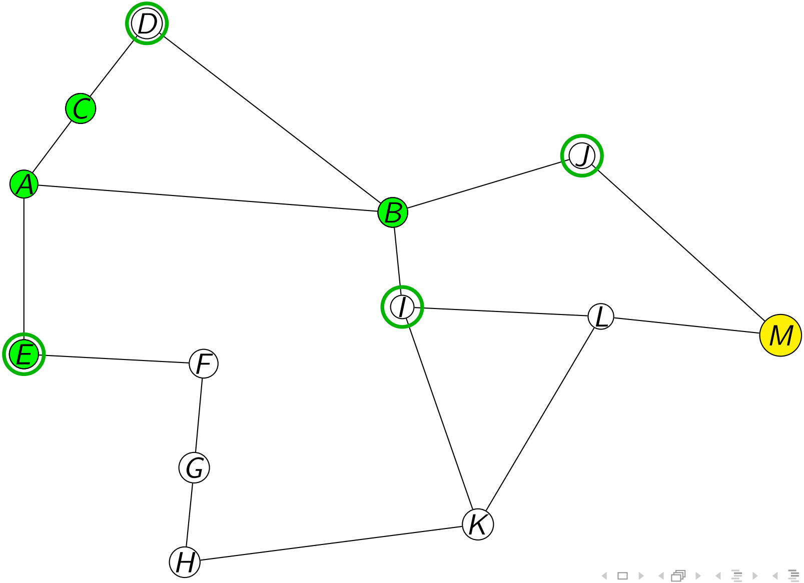
BFS



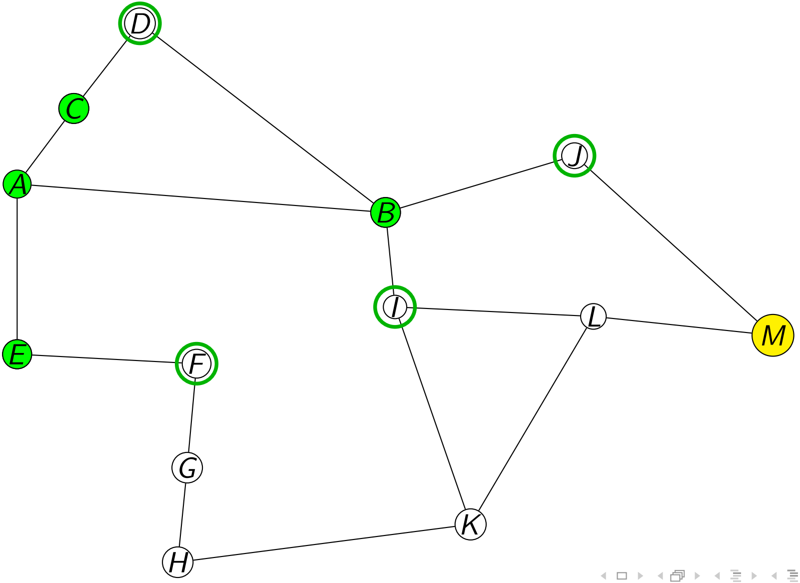
BFS



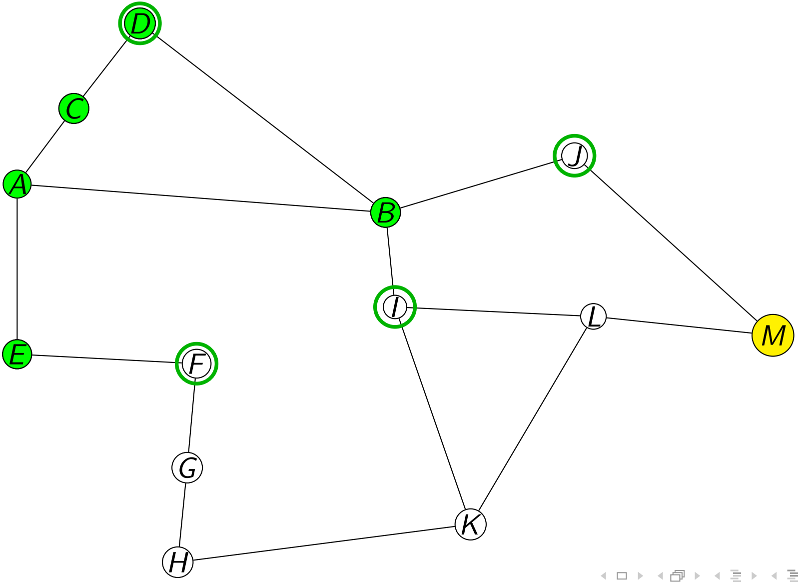
BFS



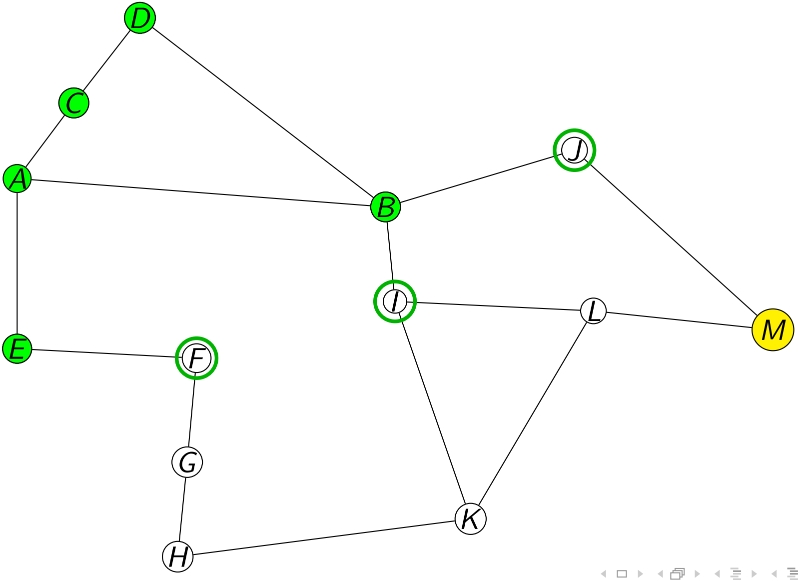
BFS



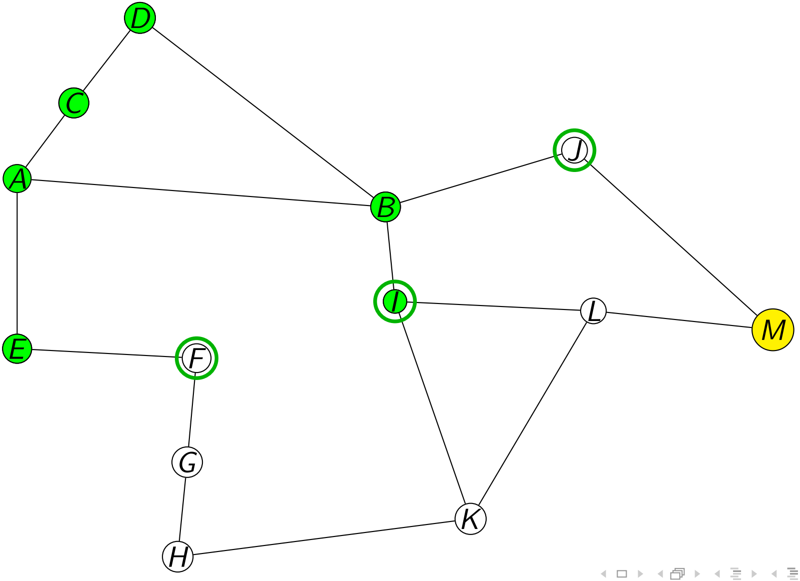
BFS



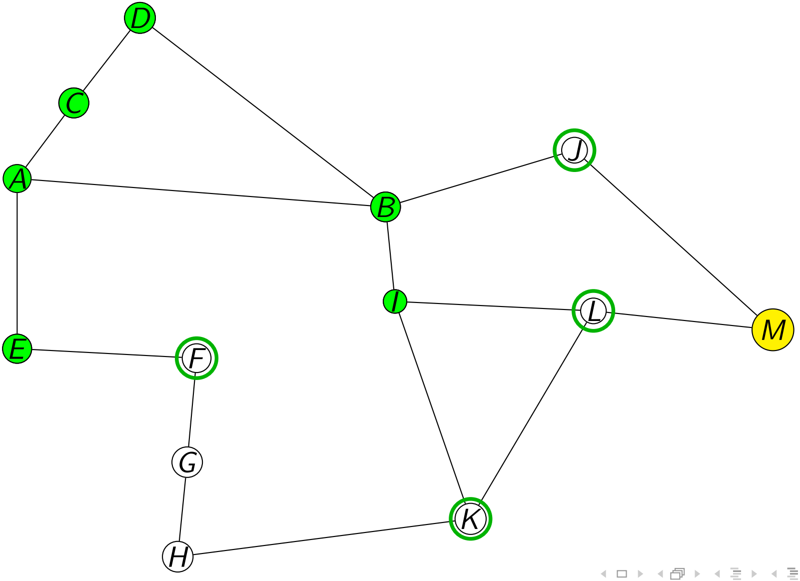
BFS



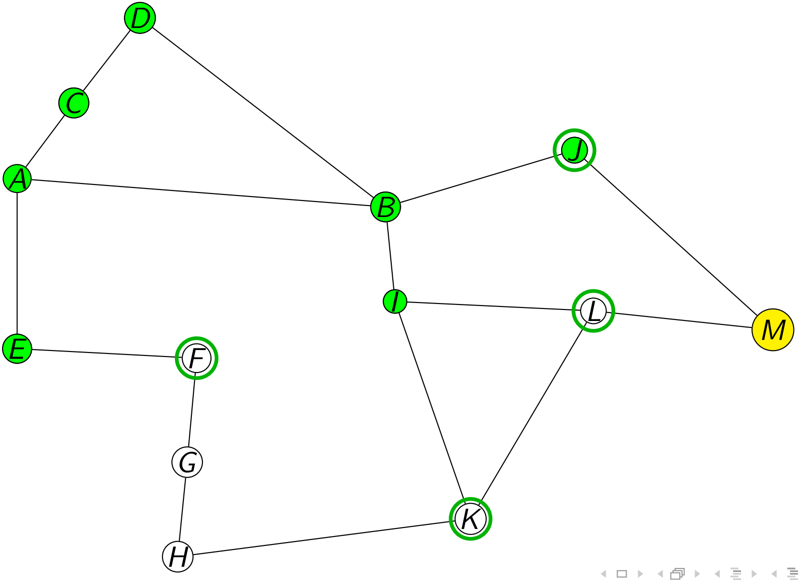
BFS



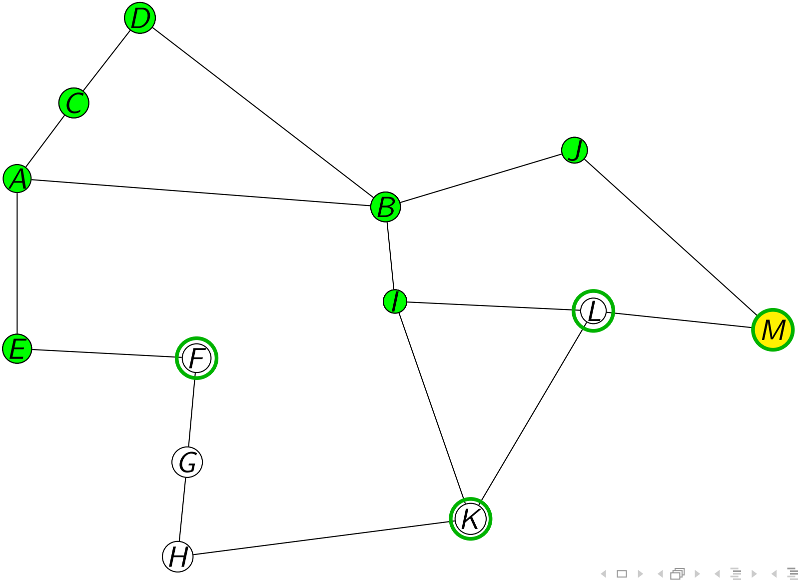
BFS



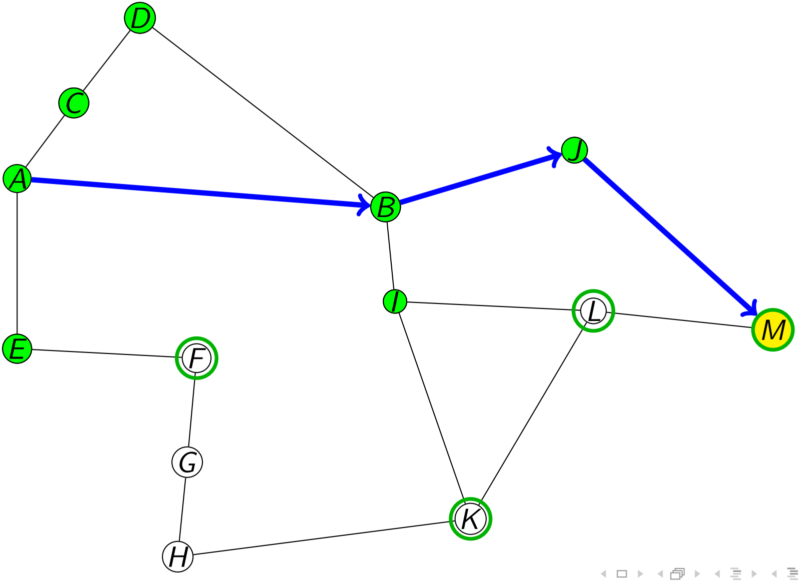
BFS



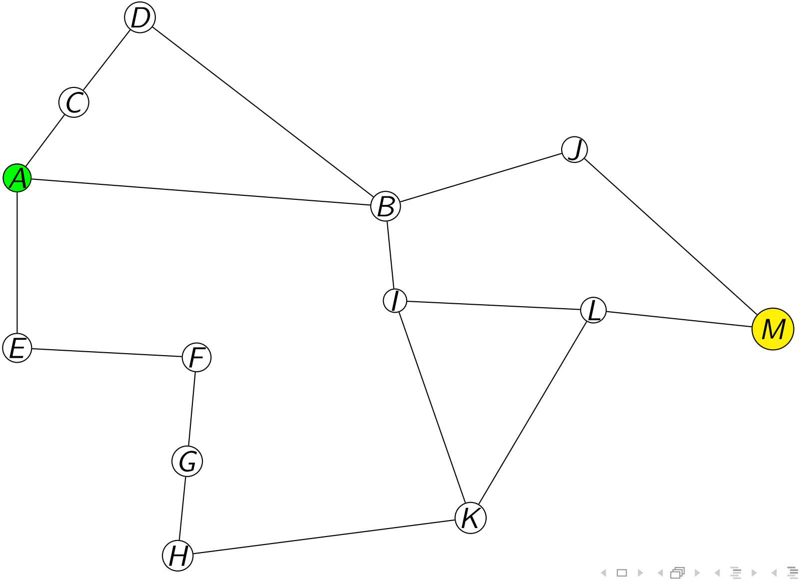
BFS



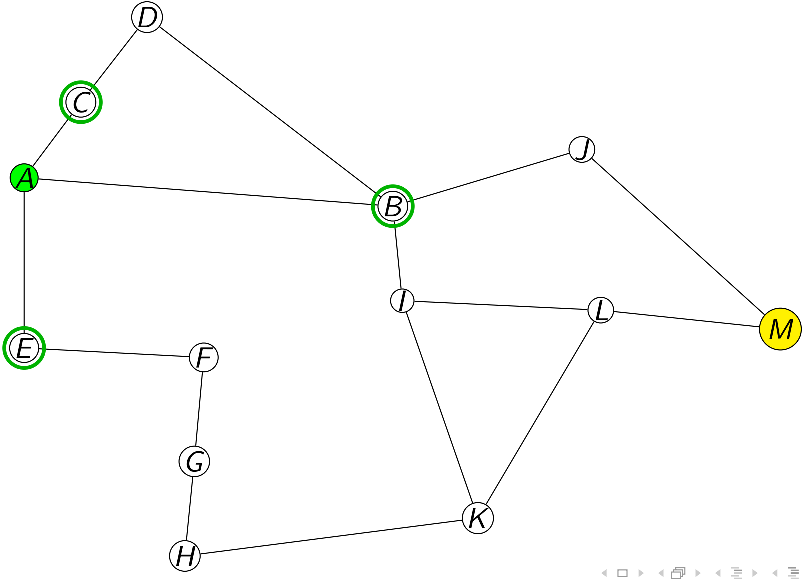
BFS



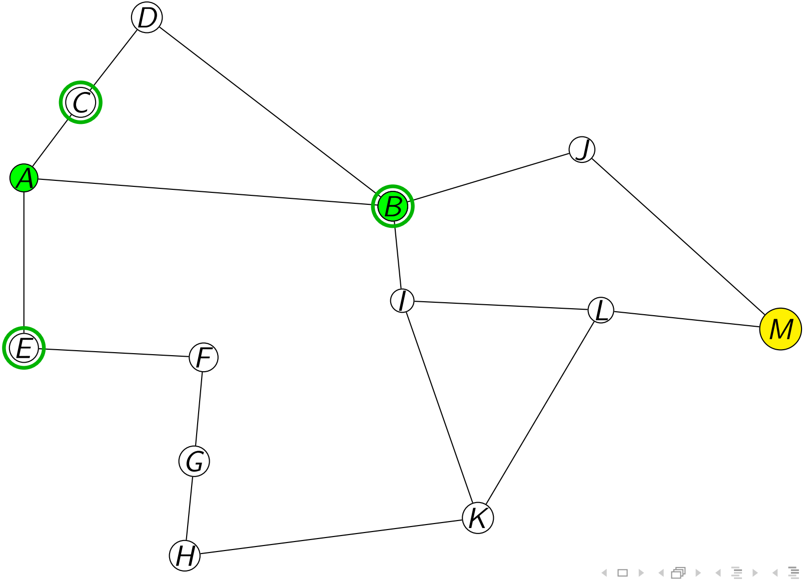
DFS



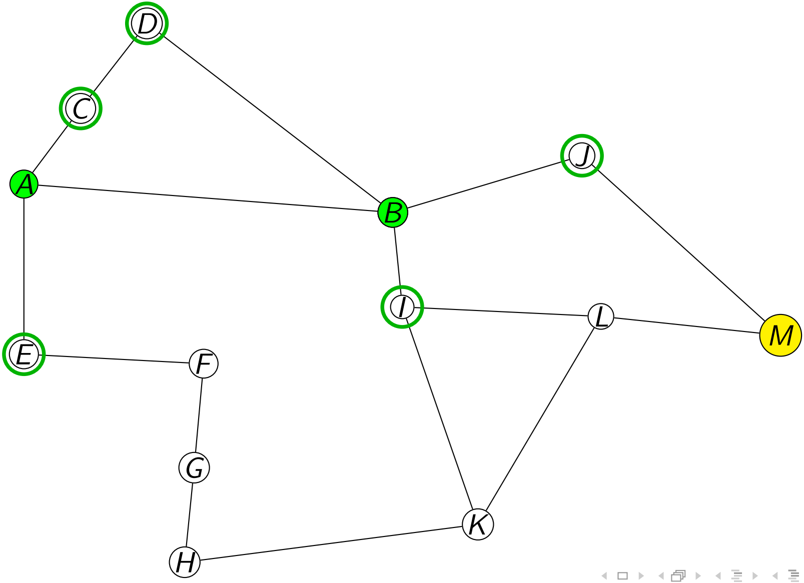
DFS



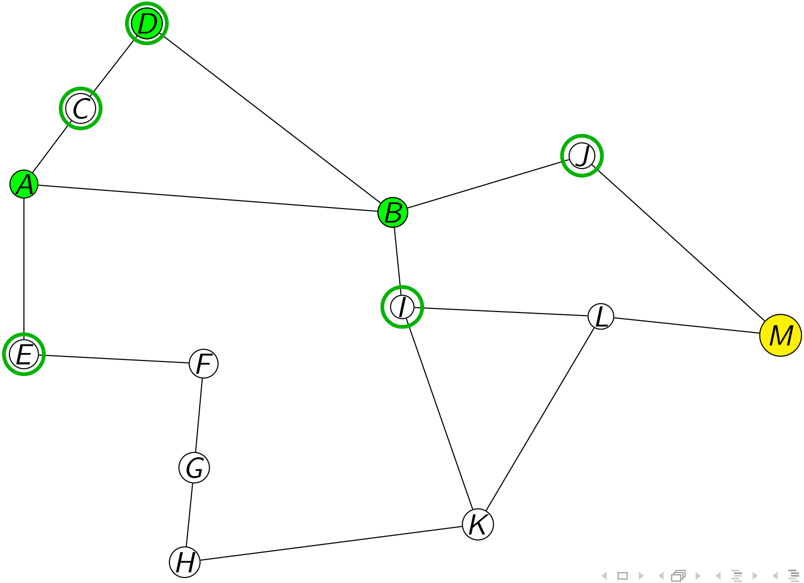
DFS



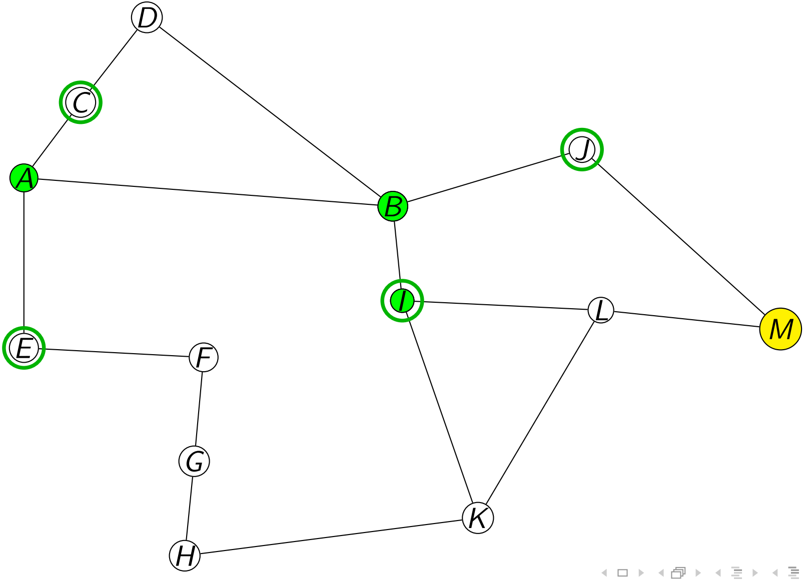
DFS



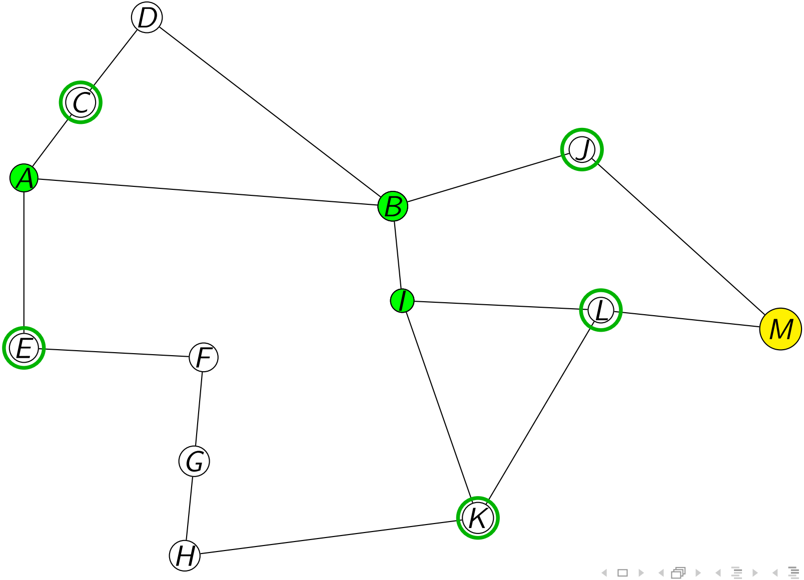
DFS



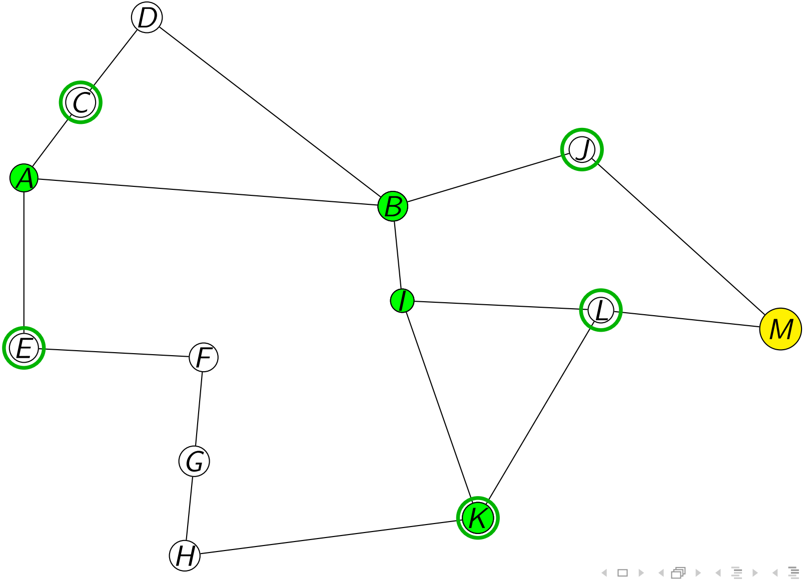
DFS



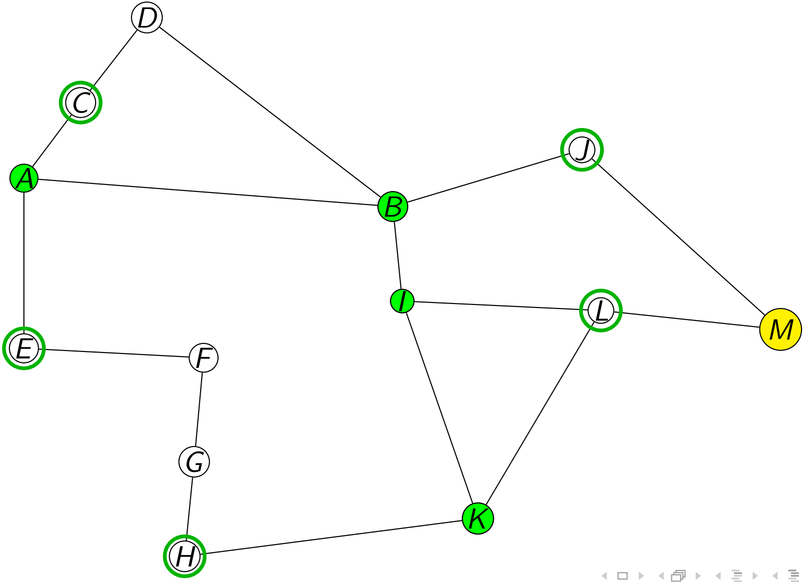
DFS



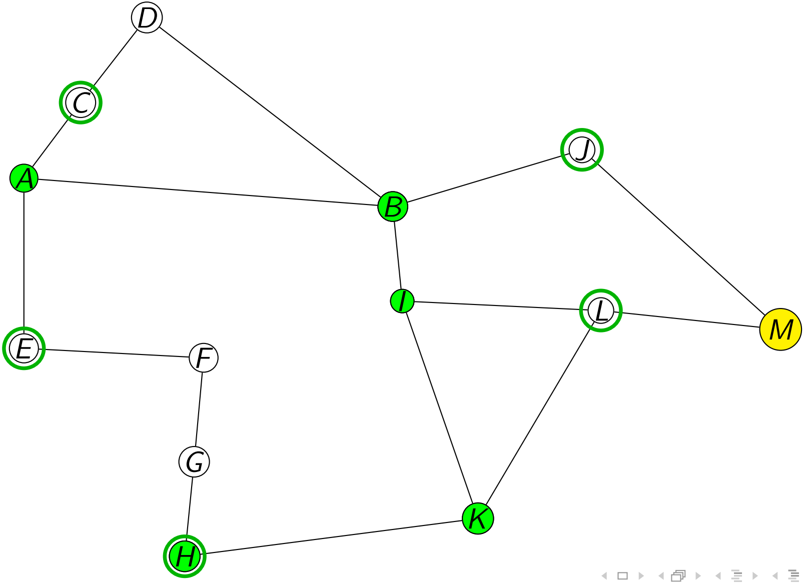
DFS



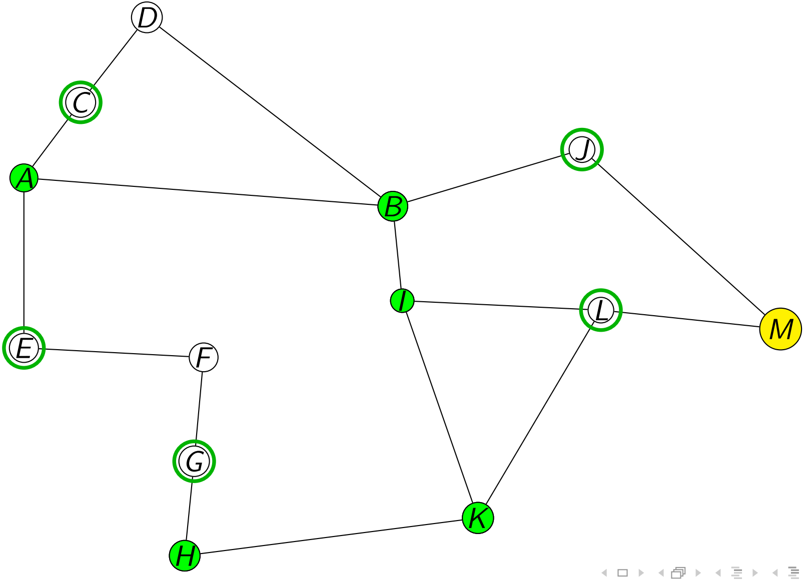
DFS



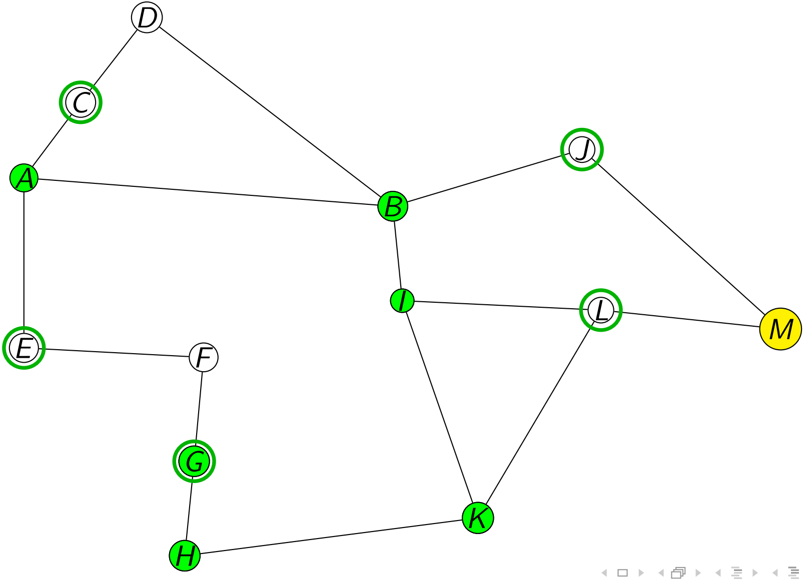
DFS



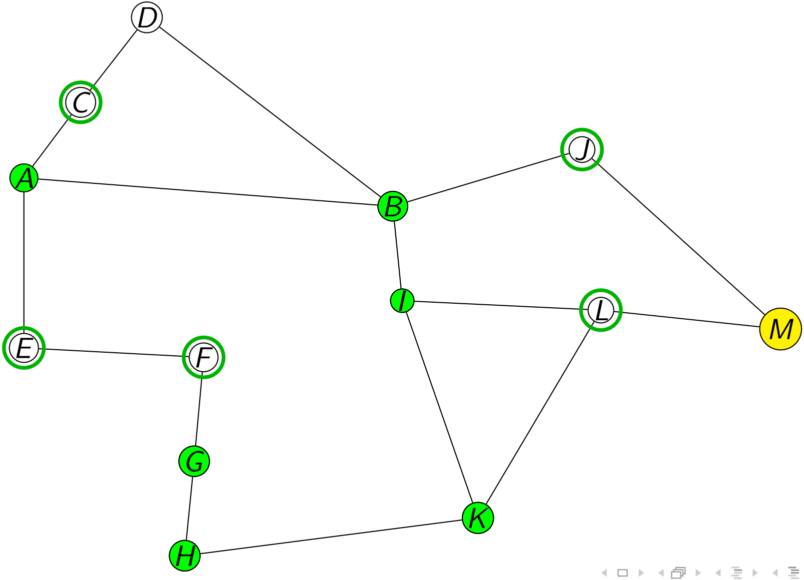
DFS



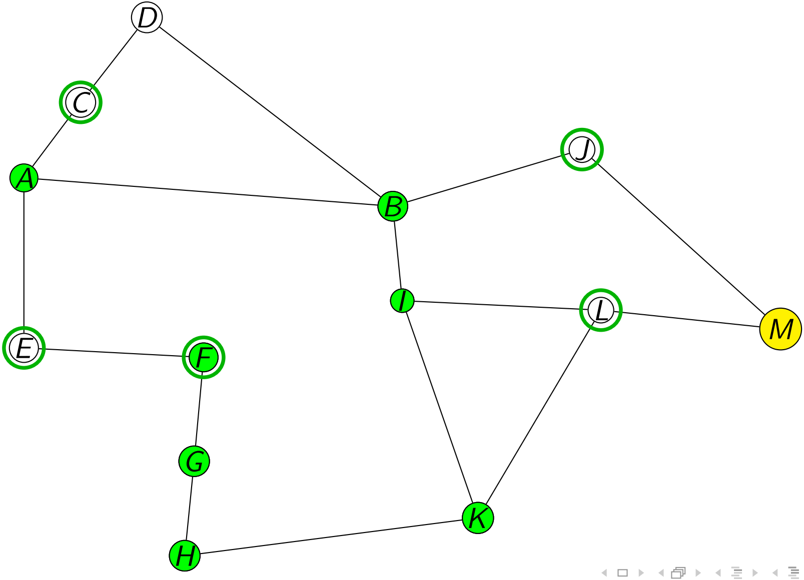
DFS



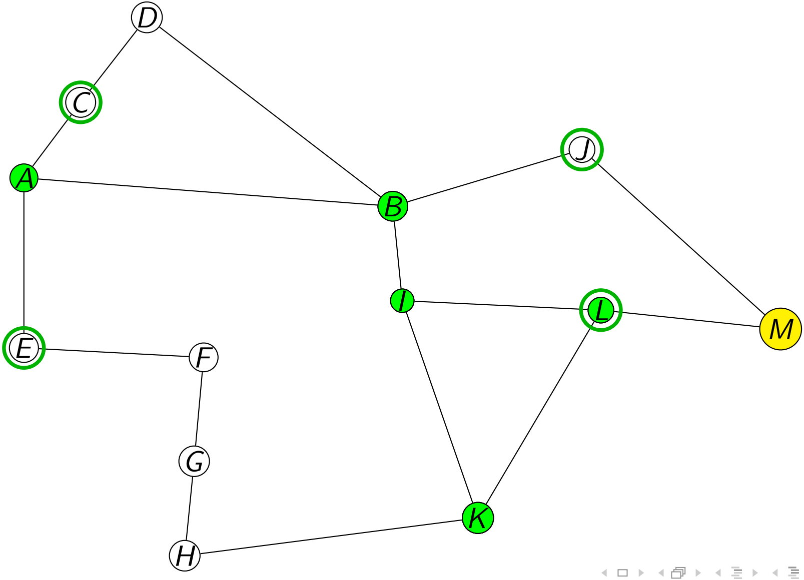
DFS



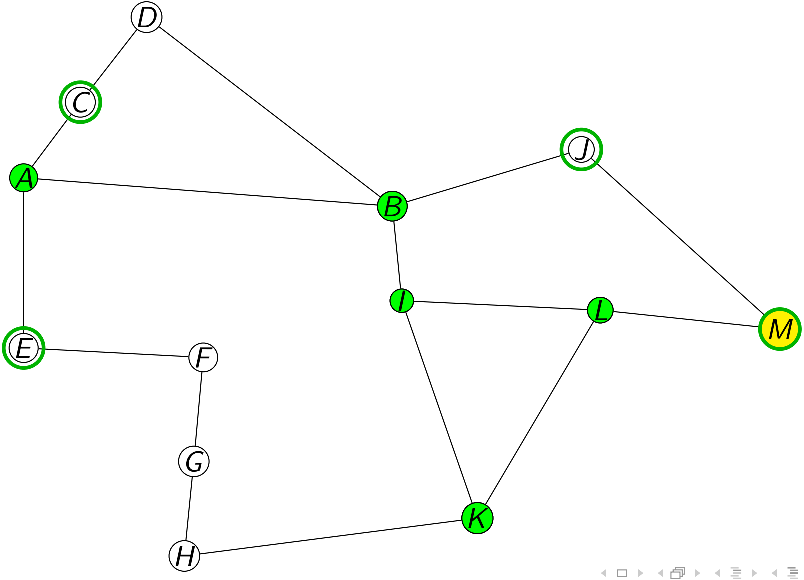
DFS



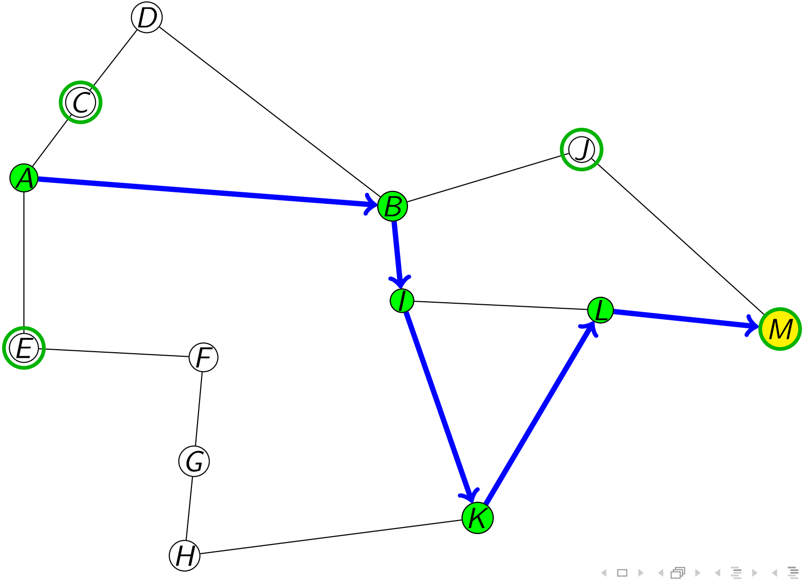
DFS



DFS



DFS

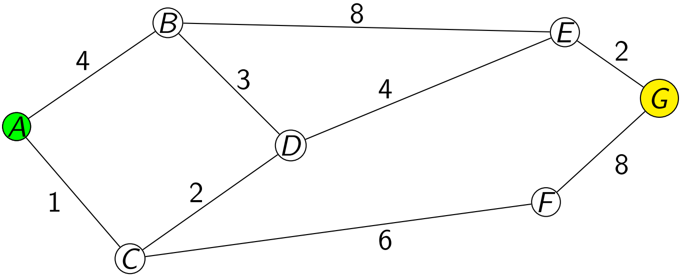


Uniform-cost search (Dijkstra)

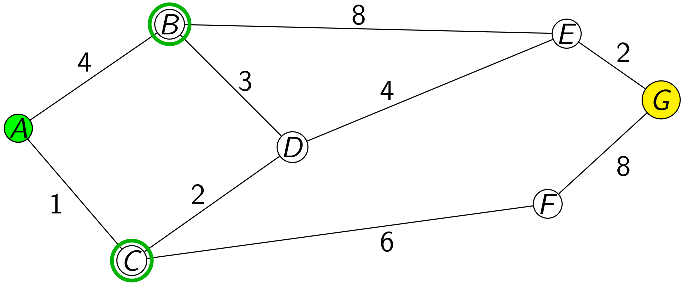
Like BFS, but:

- $g(n)$ - lowest path cost from root to n
- Frontier implemented as priority queue g
- Goal test performed when a node is **selected**
- Test if a better path is find to a node in the frontier

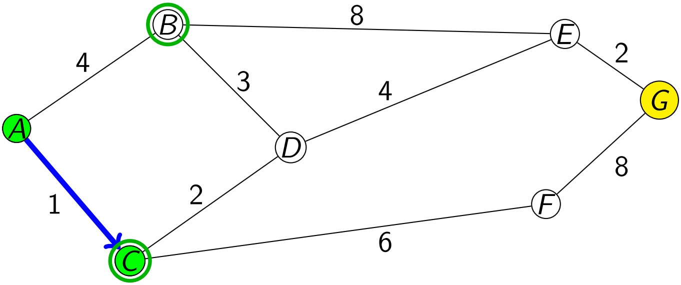
Uniform-cost search (Dijkstra)



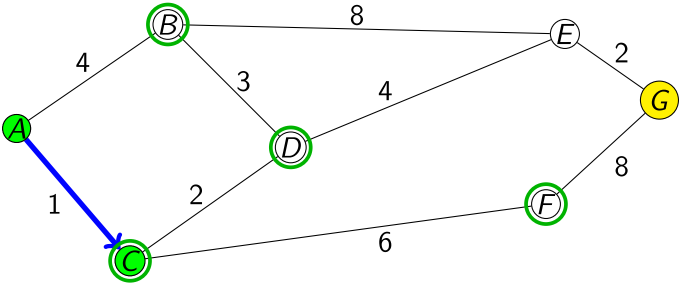
Uniform-cost search (Dijkstra)



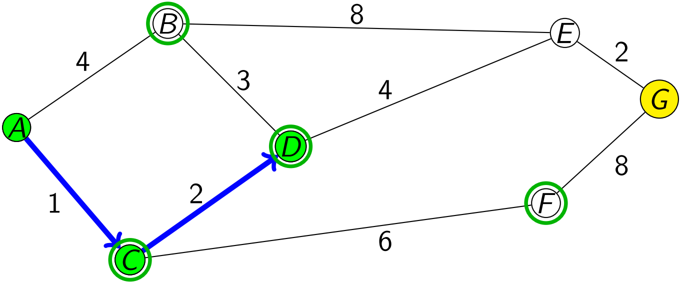
Uniform-cost search (Dijkstra)



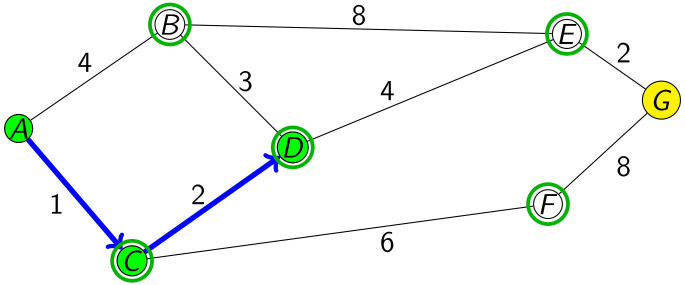
Uniform-cost search (Dijkstra)



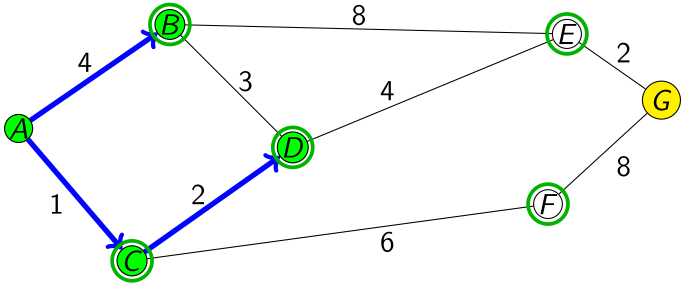
Uniform-cost search (Dijkstra)



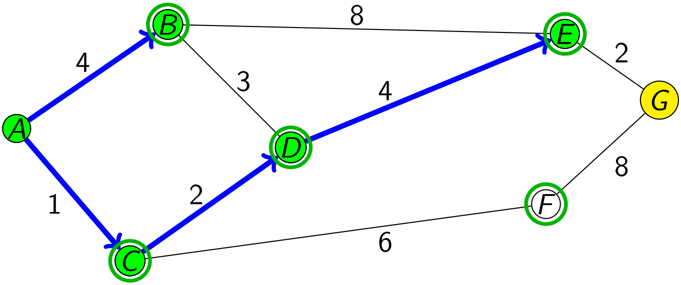
Uniform-cost search (Dijkstra)



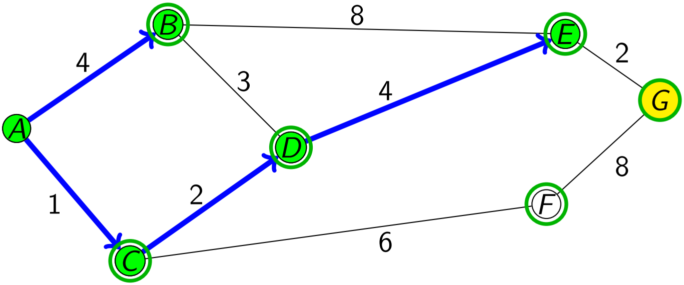
Uniform-cost search (Dijkstra)



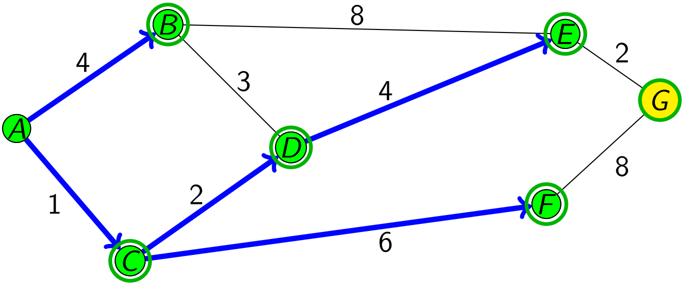
Uniform-cost search (Dijkstra)



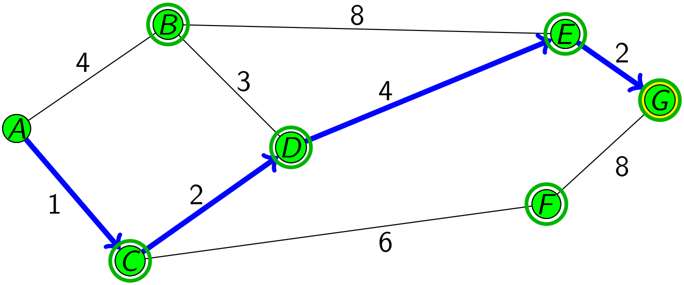
Uniform-cost search (Dijkstra)



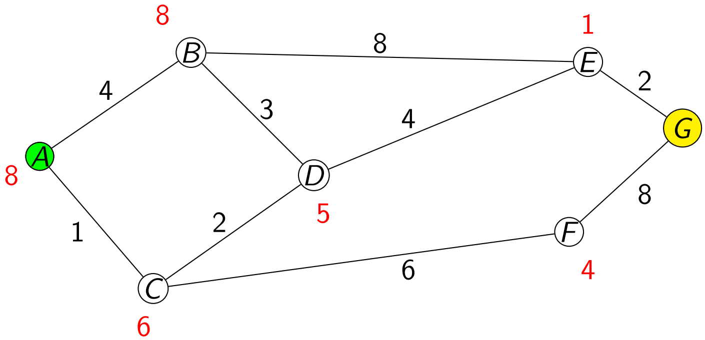
Uniform-cost search (Dijkstra)



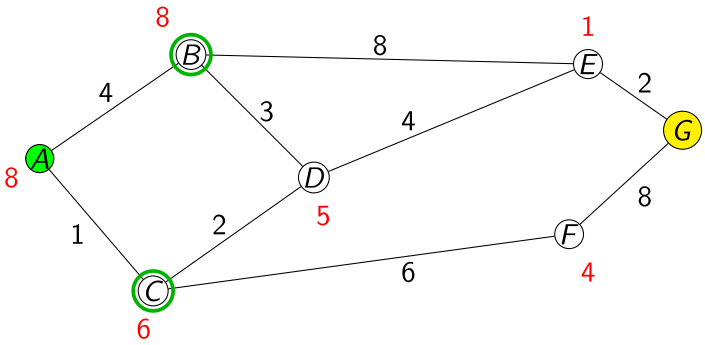
Uniform-cost search (Dijkstra)



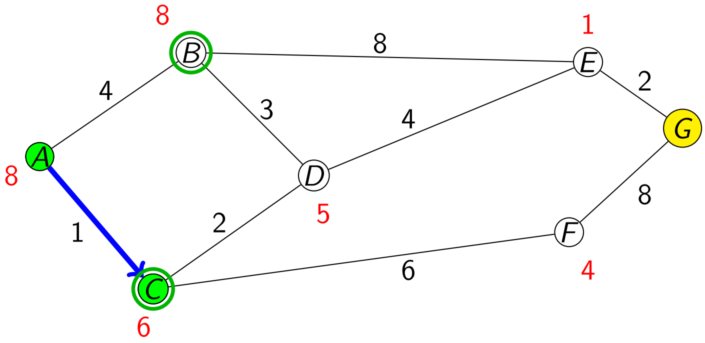
Greedy best first search



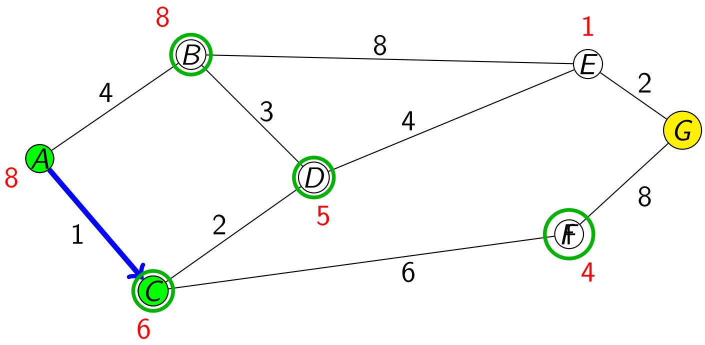
Greedy best first search



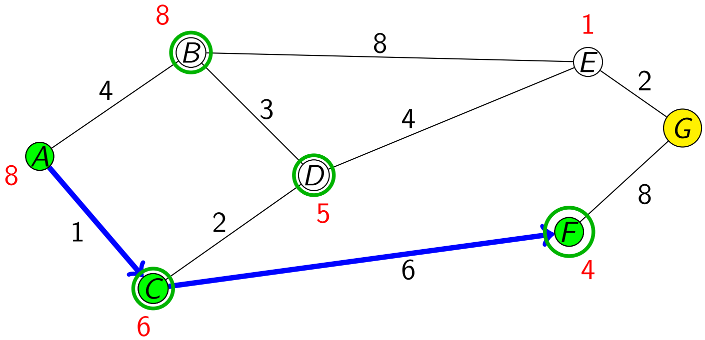
Greedy best first search



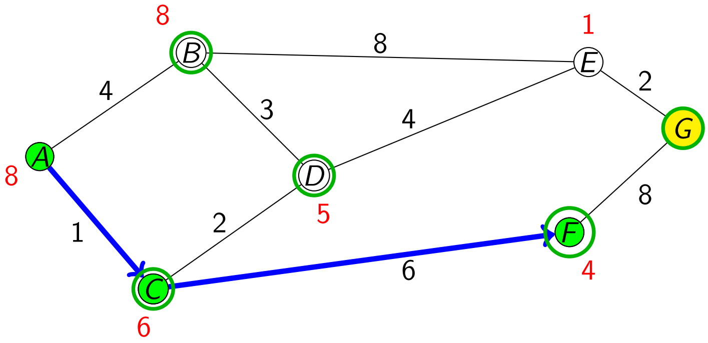
Greedy best first search



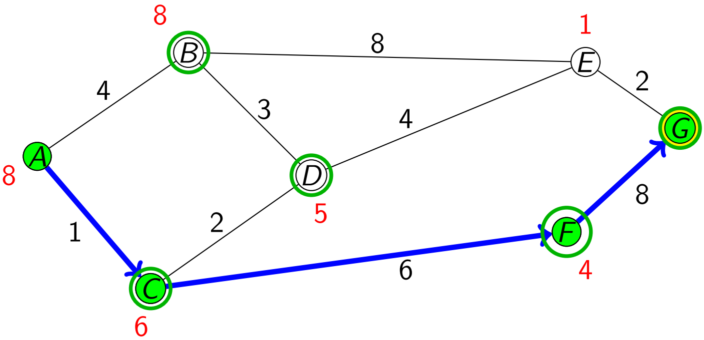
Greedy best first search



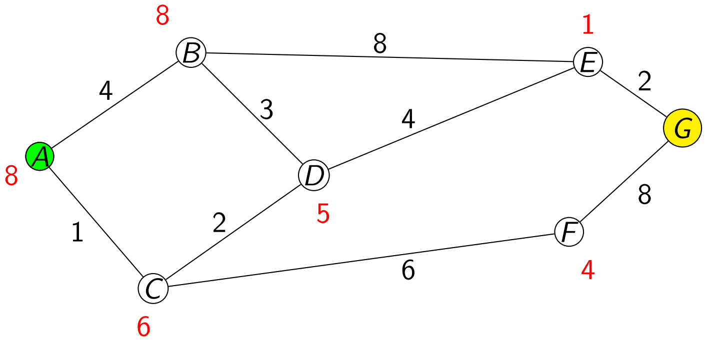
Greedy best first search



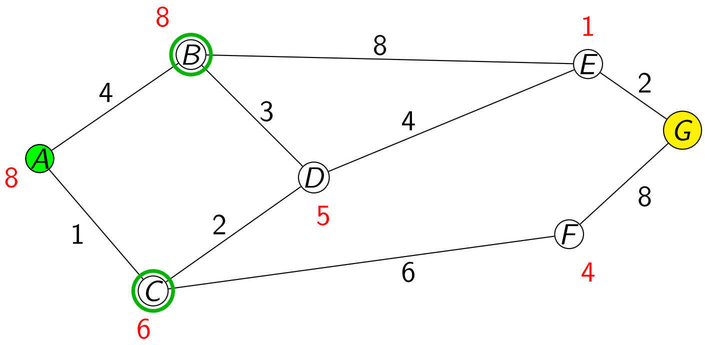
Greedy best first search



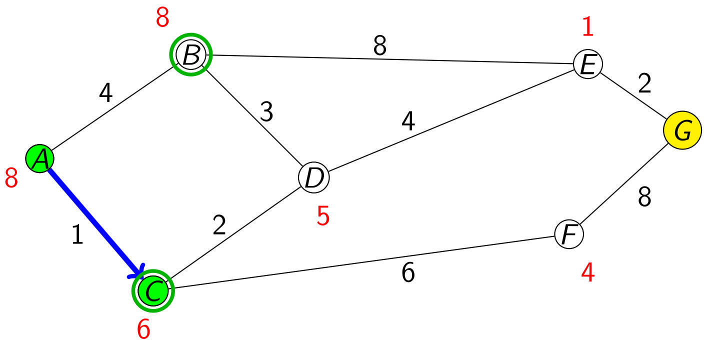
A*



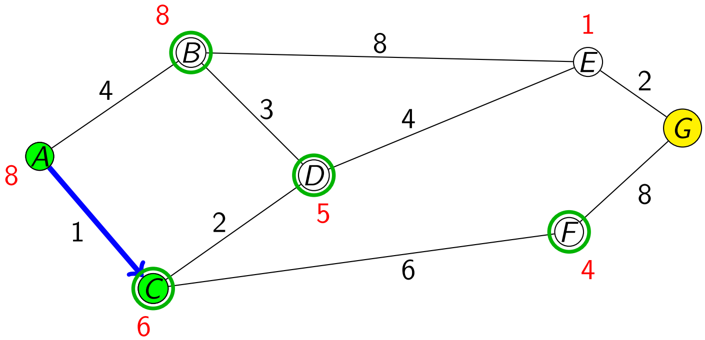
A*



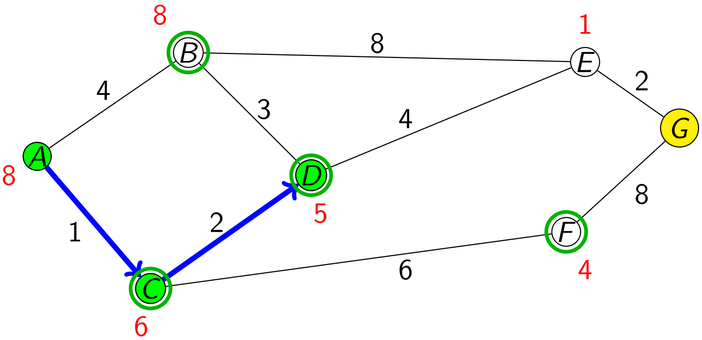
A*



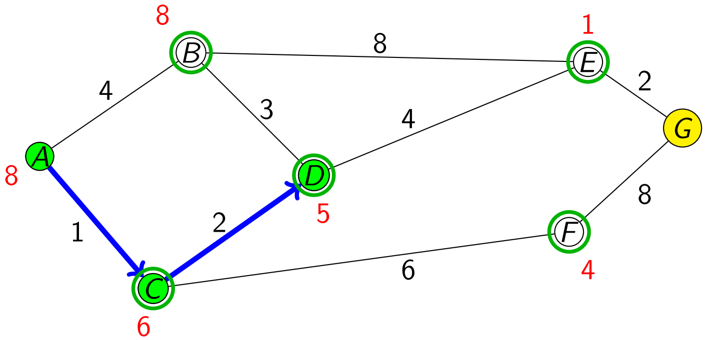
A*



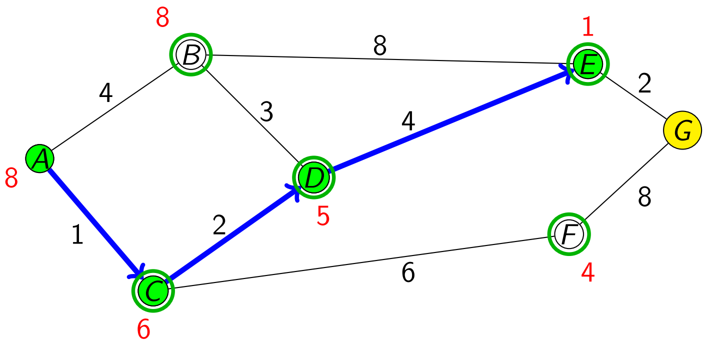
A*



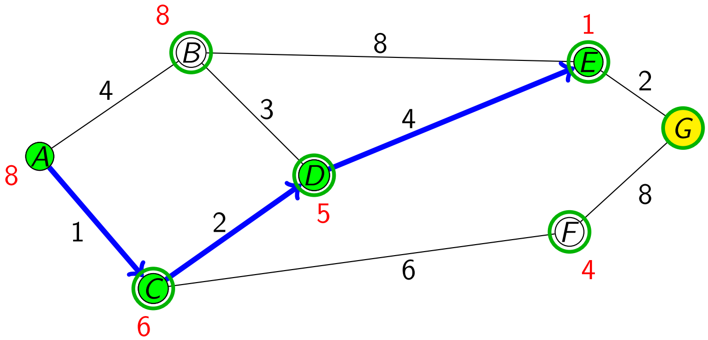
A*



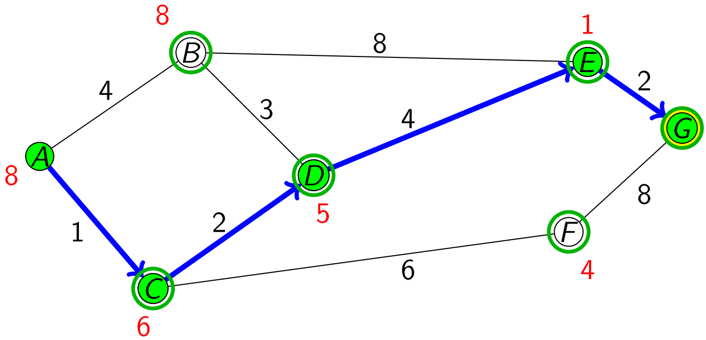
A*



A*



A*



Heuristic functions

- Heuristic function $h(n)$ is an estimate of the distance from n to goal g
- if $h(n) = 0$ for all n, we get Dijkstra
- $h(n)$ should be calculated quickly, ideally in $\mathcal{O}(1)$

Heuristic functions

Definition

$h(n)$ is admissible if for all nodes n, $0 \leq h(n) \leq p^*(n)$, where p^* is the length of a shortest path from n to g.

Definition

$h(n)$ is monotonous if $h(n) \leq h(n') + c(n, n')$, for all nodes n and edges (n, n')

Exercises

Decide whether the following heuristics are admissible and monotonous for a traffic network (cities and roads).

- Euclidean distance: $h(n) = \sqrt{(g_1 - n_1)^2 + (g_2 - n_2)^2}$
- Manhattan metric: $h(n) = |g_1 - n_1| + |g_2 - n_2|$
- Maximum metric: $h(n) = \max\{|g_1 - n_1|, |g_2 - n_2|\}$

Heuristic functions

Proposition

All monotonous heuristics are also admissible.

Exercise

Find a heuristic function that is admissible, but not monotonous.

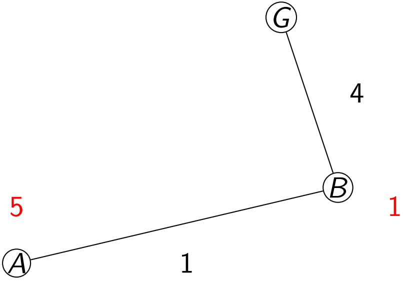
Heuristic functions

Proposition

All monotonous heuristics are also admissible.

Exercise

Find a heuristic function that is admissible, but not monotonous.



Summary

	Compl.*	Opt.	Time	Space	Node selection
BFS	Yes	Yes	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$	FIFO
DFS	No	No	$\mathcal{O}(b^m)$	$\mathcal{O}(bm)$	LIFO
Dijkstra	Yes	Yes	$\mathcal{O}(b^{1+C^*/\epsilon})$	$\mathcal{O}(b^{1+C^*/\epsilon})$	$f(n) = g(n)$
Best FS	No	No	$\mathcal{O}(b^m)$	$\mathcal{O}(b^m)$	$f(n) = h(n)$
A*	Yes	Yes**	depends	depends	$f(n) = g(n) + h(n)$

C^* is the cost of an optimal solution

ϵ smallest action (edge) cost

* depends on assumptions about the state space

** depends on heuristic