ARTIFICIAL INTELLIGENCE

ARTIFICIAL INTELLIGENCE

Module I:

Introduction to AI, production system, production rules, State-space problem, Problem, Solving by Intelligent search: BFS, DFS, Iterative Deepening Search, Hill Climbing, Simulated Annealing, heuristic Search: A*, AO* , Adversary Search: MIN-MAX Algorithm, Alpha-Beta Cut-off algorithm.

ARTIFICIAL INTELLIGENCE

Module II:

Propositional Logic, Theorem Proving by Propositional Logic, Resolution principle, Predicate Logic, wff conversion to clausal form, Dealing with Imprecision and Uncertainty: Probabilistic Reasoning, Dempster-Shafer Theory for Uncertainty Management.

ARTIFICIAL INTELLIGENCE

Module III:

Machine Learning: Supervised learning, unsupervised learning, Reinforcement learning, Artificial Neural Net, perceptron model, feed-forward neural network, Back propagation.

ARTIFICIAL INTELLIGENCE

Module IV:

Fundamentals: Components, degrees of freedom, joints, reference frames, characteristics Mathematical modelling of a robot: Mapping between frames, Description of objects in space, Transformation of vectors.

ARTIFICIAL INTELLIGENCE

Module V:

Direct Kinematic model: Mechanical Structure and notations, Description of links and joints, Kinematic modelling of the manipulator, Denavit-Hartenberg, Kinematic relationship between adjacent links, Manipulator Transformation matrix, Inverse Kinematics: Manipulator workspace, Solvable of inverse kinematic model, Manipulator Jacobian, Jacobian inverse, Application of robotics : path planning of mobile robot.

Text books

1. Artificial Intelligence, Dan W Patterson, Prentice Hall of India

2. S. Russel and P. Norvig, “Artificial Intelligence – A Modern Approach”, Second Edition, Pearson Education

3. Fu, Gonzales and Lee, Robotics, McGraw Hill

4. Robotics and Control Mittal and Nagrath Tata McGraw-Hill Education

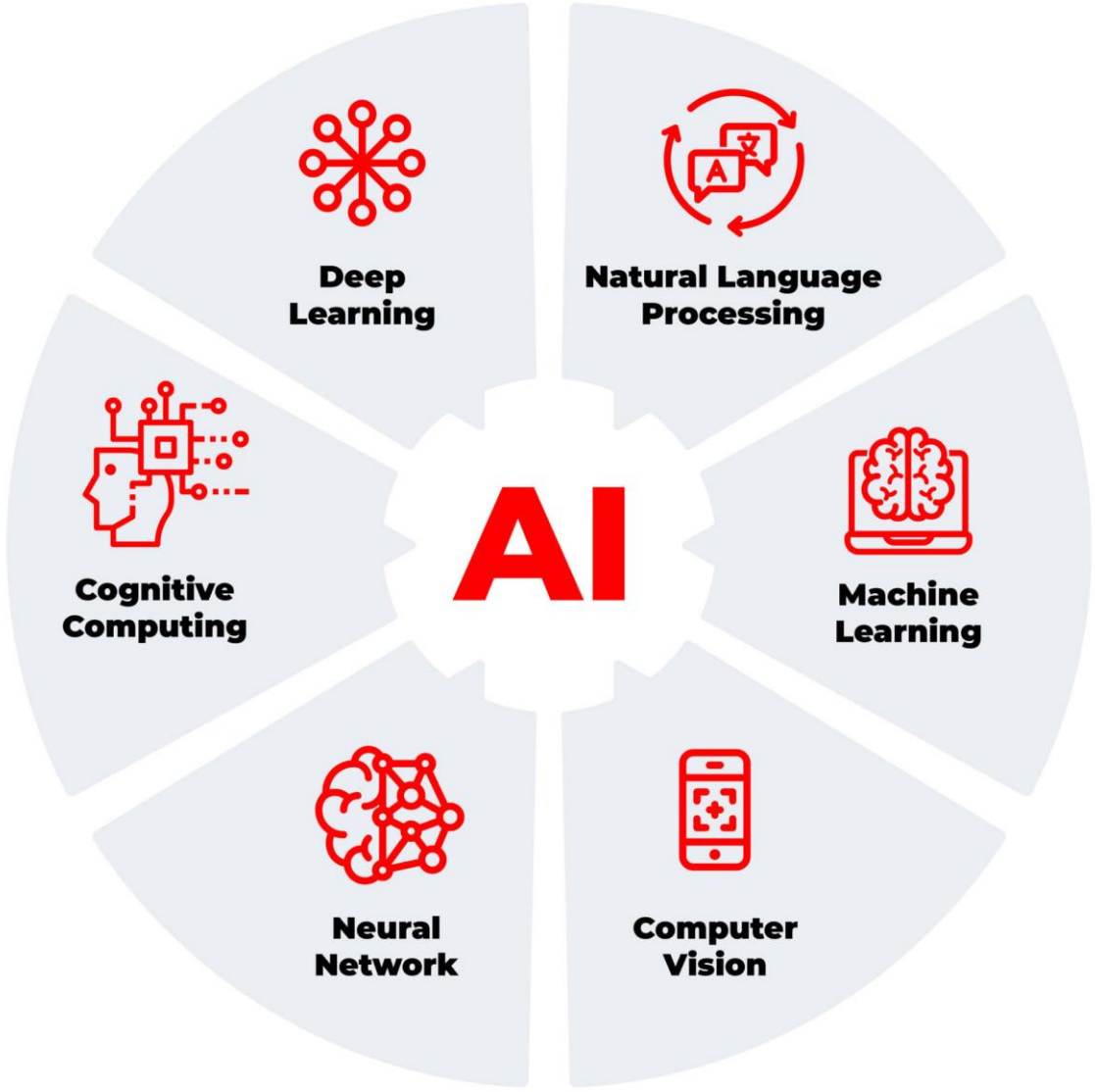
Introduction to AI

Some definitions of AI:

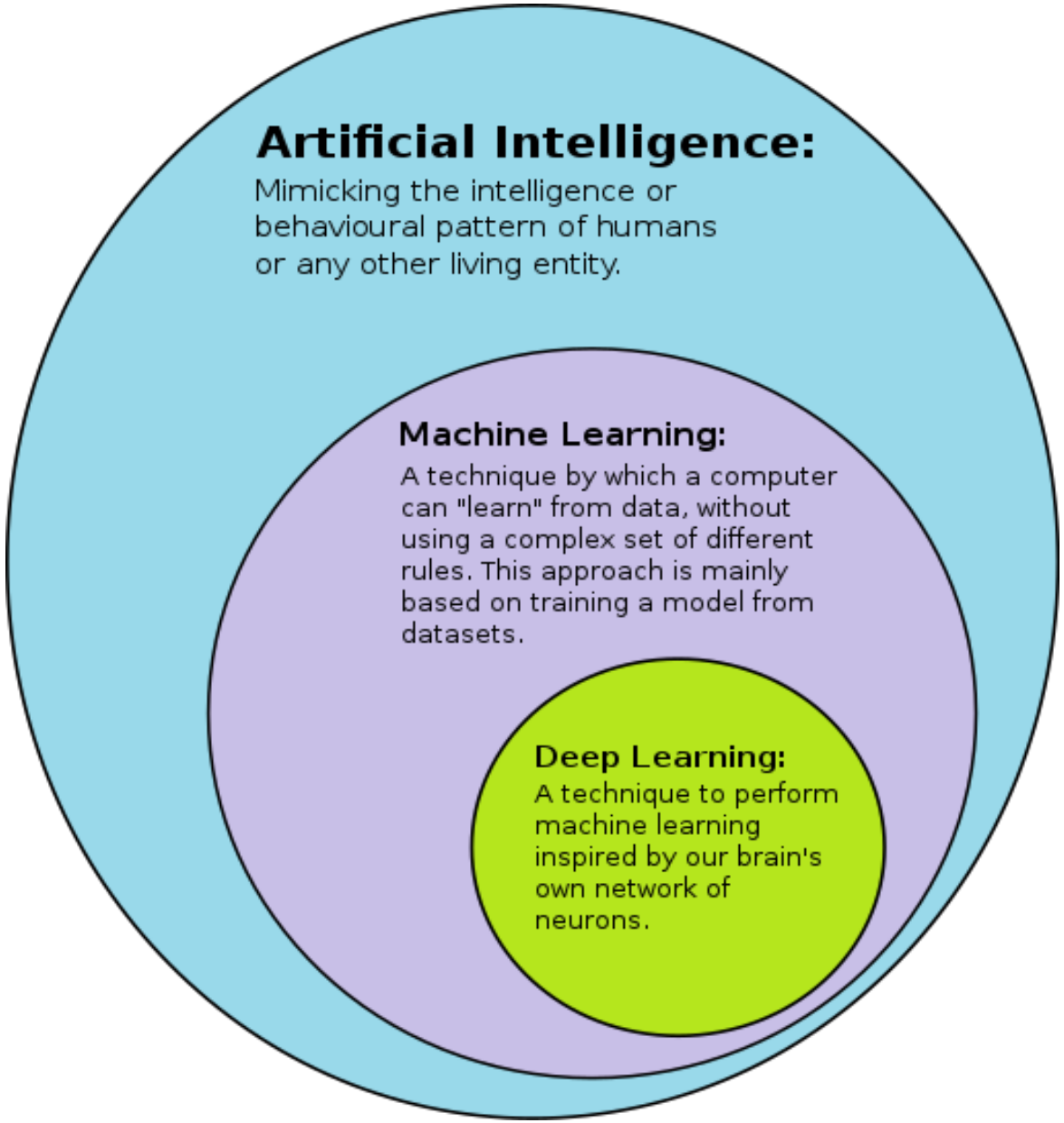
AI is a branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.

AI is refers to computers and machines to mimic the problem-solving and decision-making capabilities of the human mind.

Introduction to AI



Introduction to AI



Artificial Intelligence?

	HUMAN	RATIONAL
THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally

Artificial Intelligence?

	HUMAN	RATIONAL
THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally

"Systems that act like humans"

"The art of creating machines that perform functions that require intelligence when performed by people." **(Kurzweil)**

"The study of how to make computers do things at which, at the moment, people are better." **(Rich and Knight)**

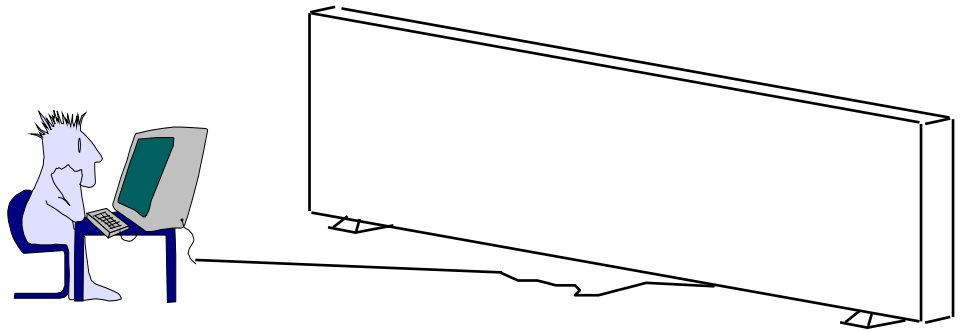
Introduction to AI

Artificial Intelligence?

THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally
	HUMAN	RATIONAL

"Systems that act like humans"

Turing Test



?

- You enter a room which has a computer terminal. You have a fixed period of time to type what you want into the terminal, and study the replies. At the other end, it is either a human being or a computer system.

- If it is a computer system, and at the end of the period you cannot surely determine whether it is a system or a human, then the system is said to be intelligent.

Introduction to AI

Artificial Intelligence?

THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally
	HUMAN	RATIONAL

"Systems that act like humans"

These cognitive tasks include:

- ***Natural language processing*** (for communication with human)
- ***Knowledge representation*** *(*to store information effectively & efficiently)
- ***Automated reasoning*** (to retrieve & answer questions using the stored information)
- ***Machine learning*** *(*to adapt to new circumstances)
- ***Computer vision*** *(to perceive objects (seeing))*
- ***Robotics*** *(to move objects (acting))*

Introduction to AI

Artificial Intelligence?

THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally
	HUMAN	RATIONAL

Artificial Intelligence?

"Systems that think like humans"

How do we know how humans think?

- Psychological experiments

Cognitive Science

- "The exciting new effort to mak computers think … machines with minds i the full and literal sense" (Haugeland)

- "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning …" (Bellman)

THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally
	HUMAN	RATIONAL

Introduction to AI

Artificial Intelligence?

	HUMAN	RATIONAL
THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally

Artificial Intelligence?

"Systems that think rationally"

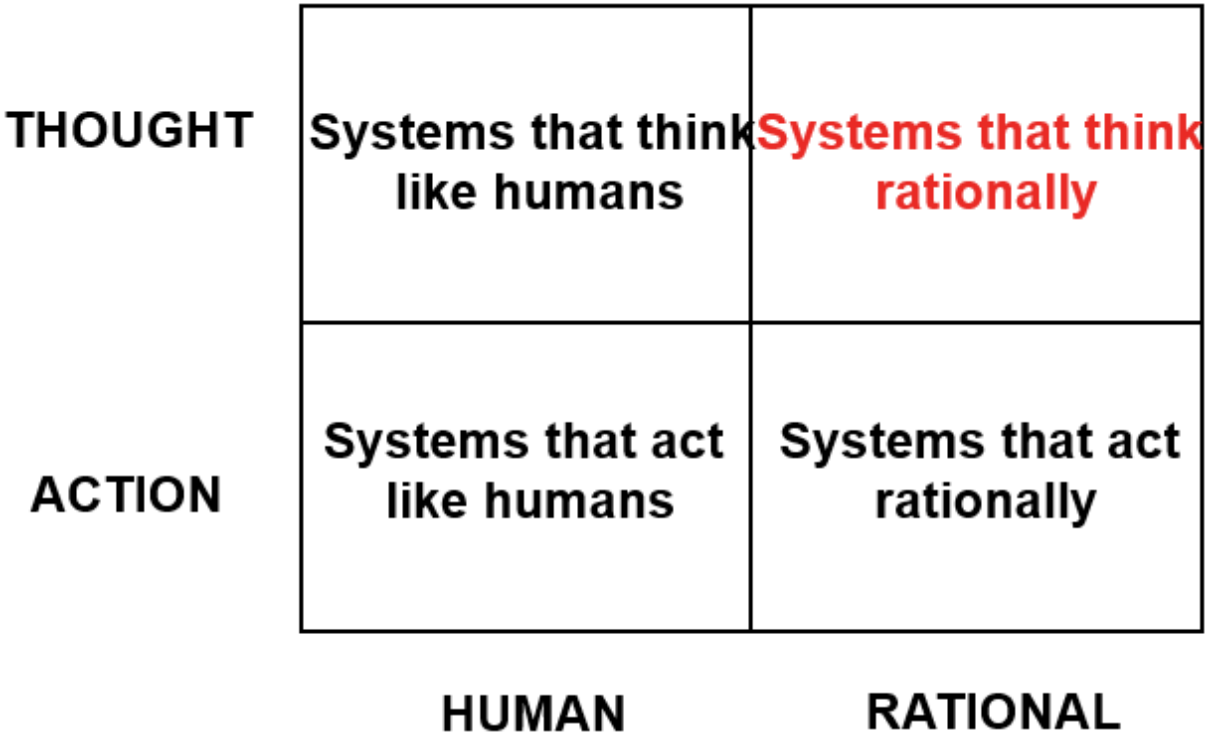
- Rational - defined in terms of logic?
- Humans are not always 'rational'!

Challenges:

- Logic can't express everything (e.g. uncertainty)
- Logical approach is often not feasible in terms of computation time (needs 'guidance')

"The study of mental facilities through the use of computational models" **(Charniak and McDermott)**

"The study of the computations that make it possible to perceive, reason, and act" **(Winston)**



Artificial Intelligence?

THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally
	HUMAN	RATIONAL

Artificial Intelligence?

"Systems that act rationally"

- Logic is the strength of rational agent.
- Humans are not always 'rational'!

Challenges:

- Sometimes logic cannot reason a correct conclusion
- At that time, some specific (in domain) human knowledge or information is used

It is more than using logic only:
i.e. AI should use LOGIC + Domain knowledge

	HUMAN	RATIONAL
THOUGHT	Systems that think like humans	Systems that think rationally
ACTION	Systems that act like humans	Systems that act rationally

Introduction to AI

History of AI

Early period - 1950's & 60's

Game playing, Theorem proving, symbol manipulation
Biological models, neural nets

Symbolic application period - 70's

Early expert systems, use of knowledge

Commercial period - 80's

boom in knowledge/ rule bases

90's and New Millenium

Real-world applications, modelling, use of theory, ...
Topics: data mining, formal models, GA's, fuzzy logic, agents, neural nets, autonomous systems
Applications: visual recognition of traffic, medical diagnosis, directory enquiries, power plant control automatic cars

Introduction to AI

History of AI

Weak AI

Weak AI, also known as narrow AI, focuses on performing a specific task, such as answering questions based on user input

Strong AI

Strong AI can perform a variety of functions, eventually teaching itself to solve for new problems

History of AI

Weak AI	Strong AI
Limited to perform specific tasks	Perform intelligent human level activities
Programmed for fixed function	Have the ability to learn, think and perform new activities like humans
It doesn't have any consciousness or awareness of its own.	It poses creativity, common sense and logic like humans.

State Space Search in AI

State Space:

It consist of **Initial state** (of a given problem), **Goal state**, **Intermediate state**, and **the actions of the agent** which is taken to go from one state to other state.

Search:

Searching in AI is the task to **explore** the **Goal state** from **Initial state** (objective is to optimize the cost).

Basics of State Space Modelling

1 STATE (CONFIGURATION):

– A set of variables which define a state or configuration

– Domains for every variable and constraints among variables to define a valid configuration

2 STATE TRANSFORMATION RULES (MOVES):

– A set of RULES which define which are the valid set of NEXT STATE of a given State

– It also indicates who can make these Moves

3 STATE SPACE (IMPLICIT GRAPH)

– The Complete Graph produced out of the State Transformation Rules.

– Typically too large to store. Could be Infinite.

4. INITIAL or START STATE(s), GOAL STATE(s)

5. SOLUTION(s), COSTS

– Depending on the problem formulation, it can be a PATH from Start to Goal or a Sub-graph

6. SEARCH ALGORITHMS

– Intelligently explore the Implicit Graph or State Space by examining only a small sub-set to find the solution

– To use Domain Knowledge or HEURISTICS to try and reach Goals faster

Basics of State Space Modelling

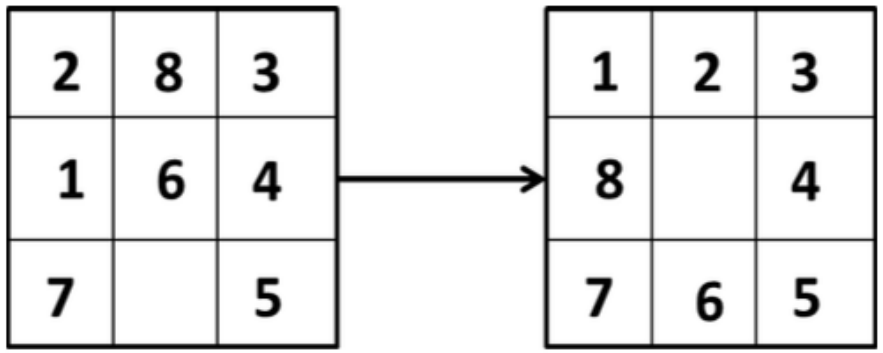
1 . STATE (CONFIGURATION):

– A set of variables which define a state or configuration

– Domains for every variable and constraints among variables to define a valid configuration

8 puzzle problem

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.



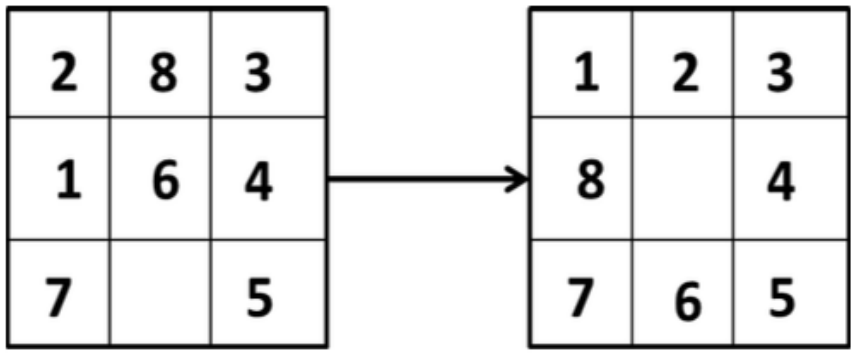
Basics of State Space Modelling

1 . STATE (CONFIGURATION):

– A set of variables which define a state or configuration

– Domains for every variable and constraints among variables to define a valid configuration

8 puzzle problem



Initial State Goal State

CONFIGURATION

X1	X2	X3
X4	X4	X6
X7	X8	X9

Variables: {X1, X2, X3, X4, X5, X6, X7, X8, X9}

Domain: {1, 2, 3, 4, 5, 6, 7, B}

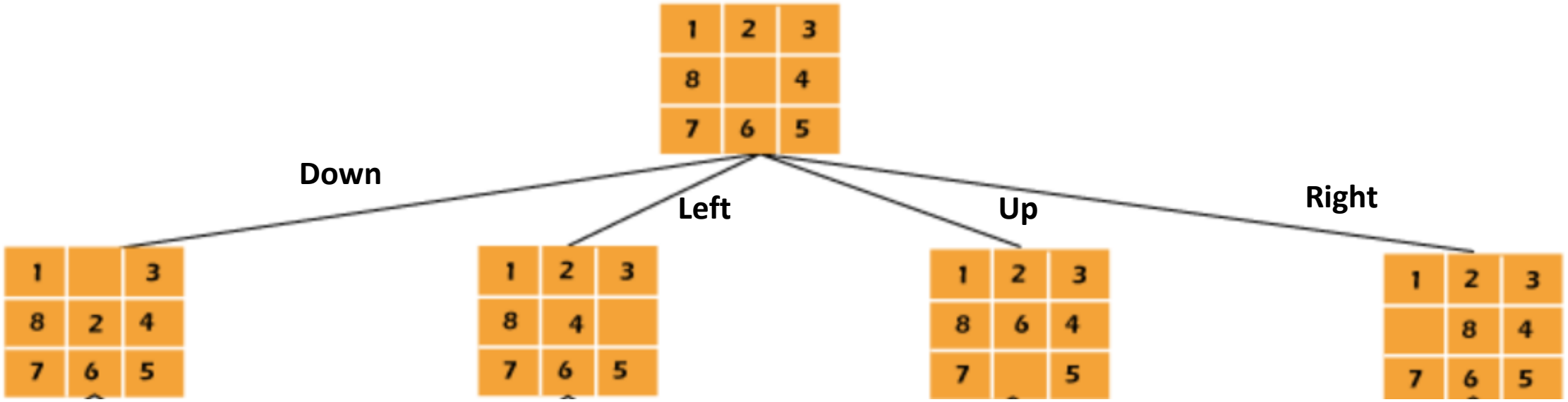
Basics of State Space Modelling

2. STATE TRANSFORMATION RULES (MOVES):

– A set of RULES which define which are the valid set of NEXT STATE of a given State

– It also indicates who can make these Moves

8 puzzle problem



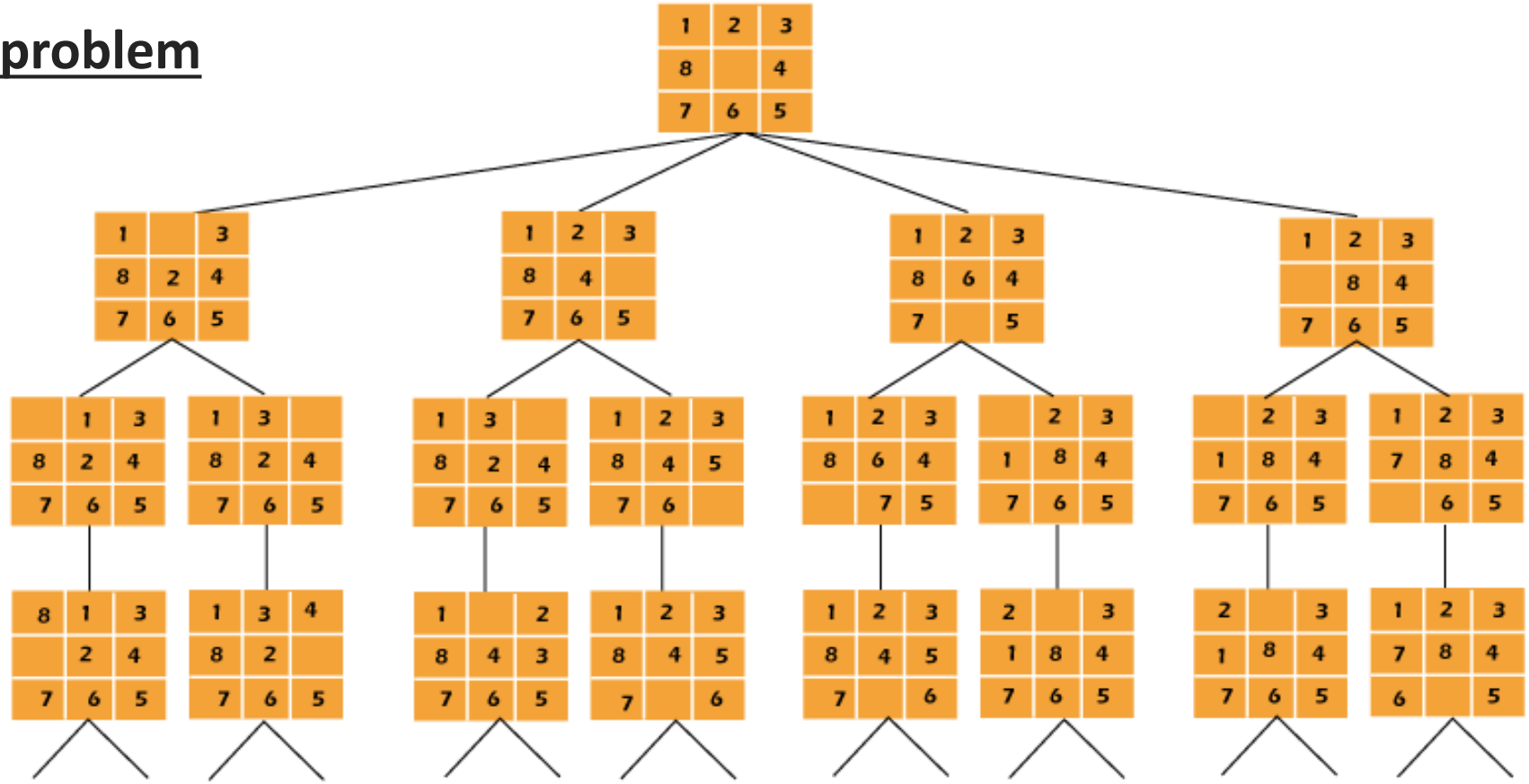
Basics of State Space Modelling

3. STATE SPACE (IMPLICIT GRAPH)

– The Complete Graph produced out of the State Transformation Rules.

– Typically too large to store. Could be Infinite.

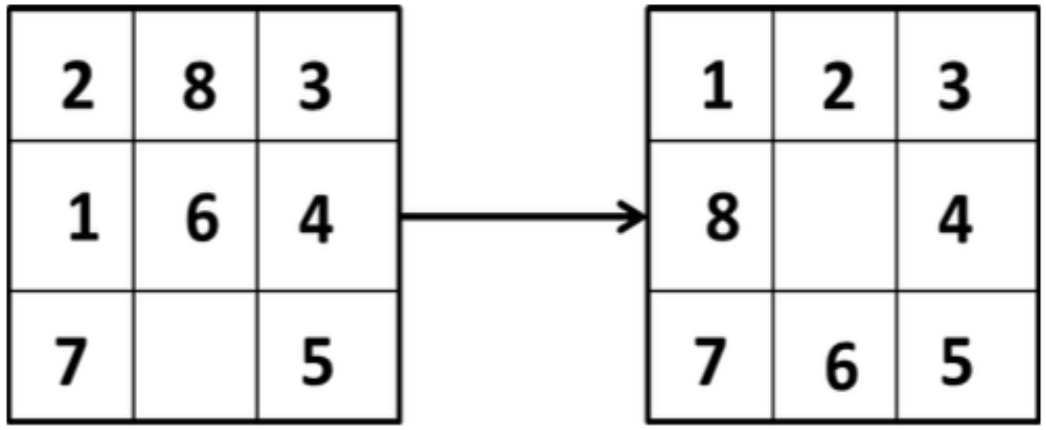
8 puzzle problem



Basics of State Space Modelling

4. INITIAL or START STATE(s), GOAL STATE(s)

8 puzzle problem

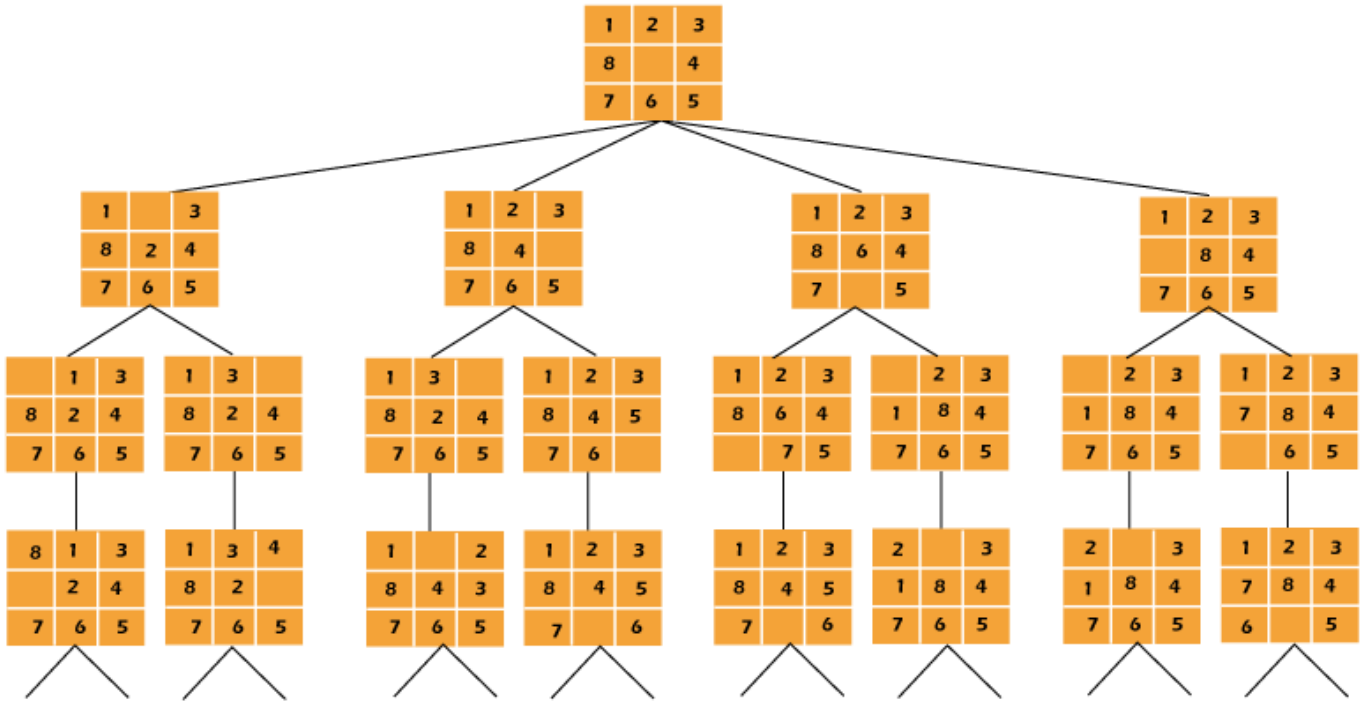


Basics of State Space Modelling

5. SOLUTION(s), COSTS

– Depending on the problem formulation, it can be a PATH from Start to Goal or a Sub-graph

8 puzzle problem



Basics of State Space Modelling

6. SEARCH ALGORITHMS

– Intelligently explore the Implicit Graph or State Space by examining only a small sub-set to find the solution

– To use Domain Knowledge or HEURISTICS to try and reach Goals faster

SEARCH ALGORITHMS: BFS, DFS, Iterative Deepening Search, Hill Climbing, Simulated Annealing, heuristic Search: A*, AO* , Adversary Search: MIN-MAX Algorithm, Alpha-Beta Cut-off algorithm

Introduction to AI

Basics of State Space Modelling (8 Puzzle Problem)

1. **STATE** (CONFIGURATION):

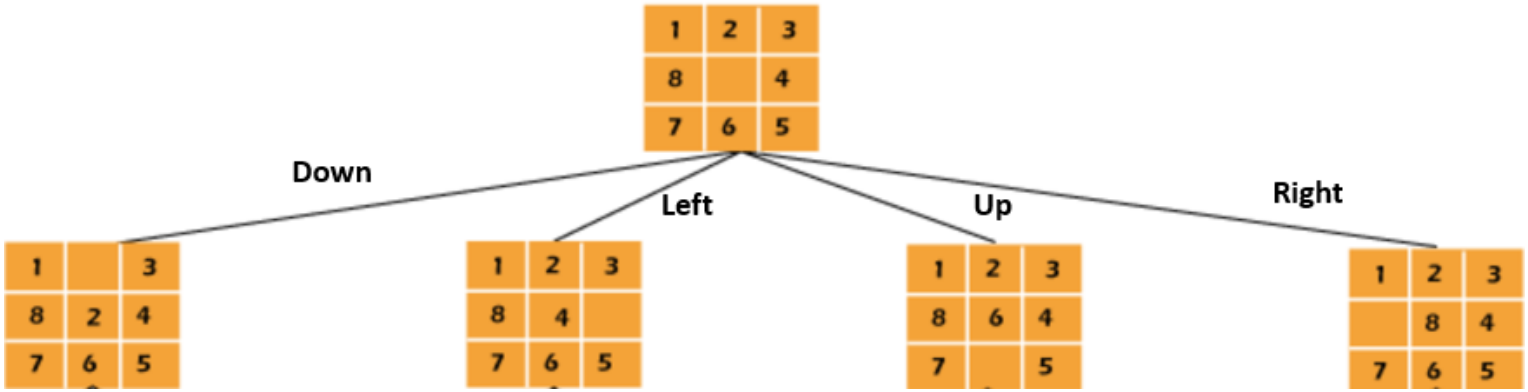
X1	X2	X3
X4	X4	X6
X7	X8	X9

Variables: {X1, X2, X3, X4, X5, X6, X7, X8, X9}

Domain: {1, 2, 3, 4, 5, 6, 7, B}

2	8	3
1	6	4
7		5

2. **STATE TRANSFORMATION RULES** (MOVES):

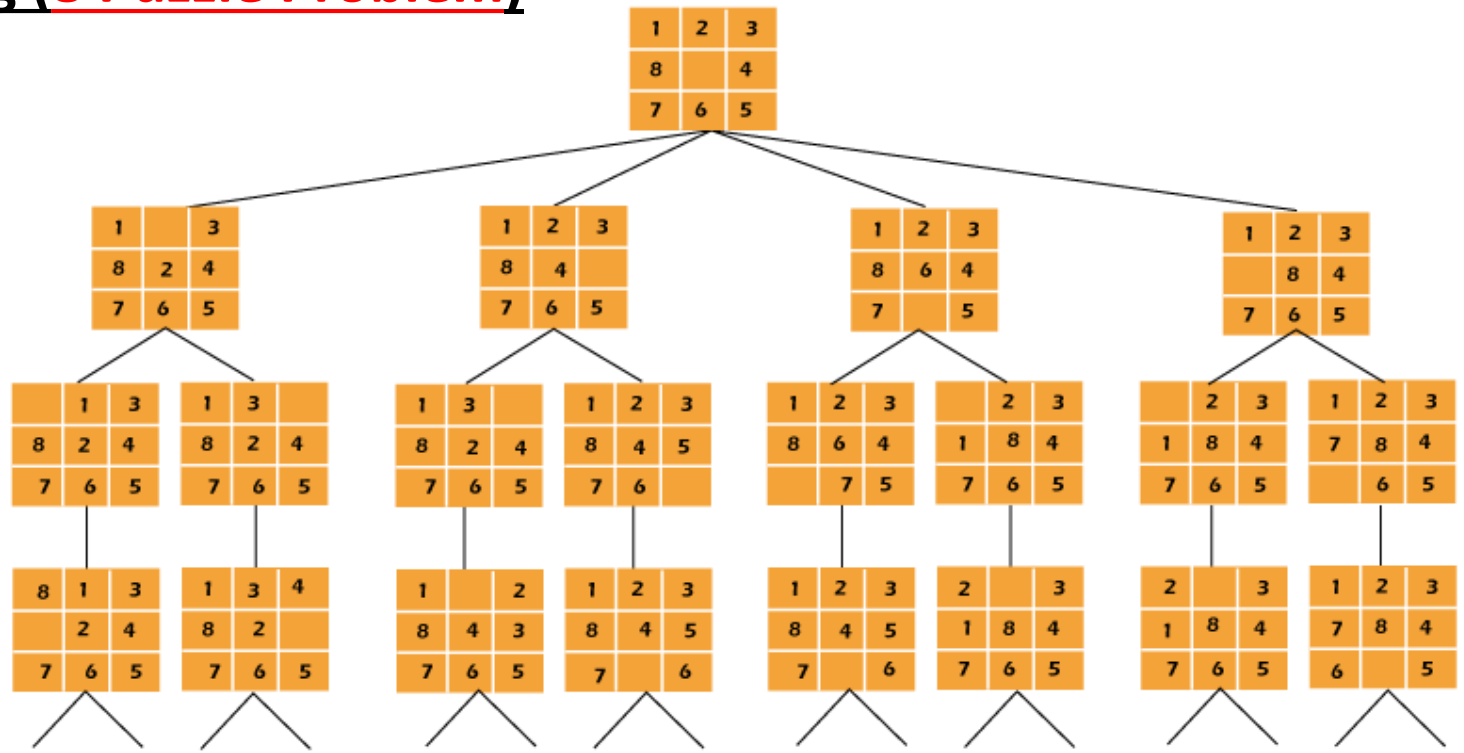


Actions = {**Up**, **Down**, **Left**, **Right**}

Transformations: It defines how the state changes through actions

Basics of State Space Modelling (8 Puzzle Problem)

3. STATE SPACE



4. INITIAL or START STATE(s), GOAL STATE(s)

2	8	3
1	6	4
7		5

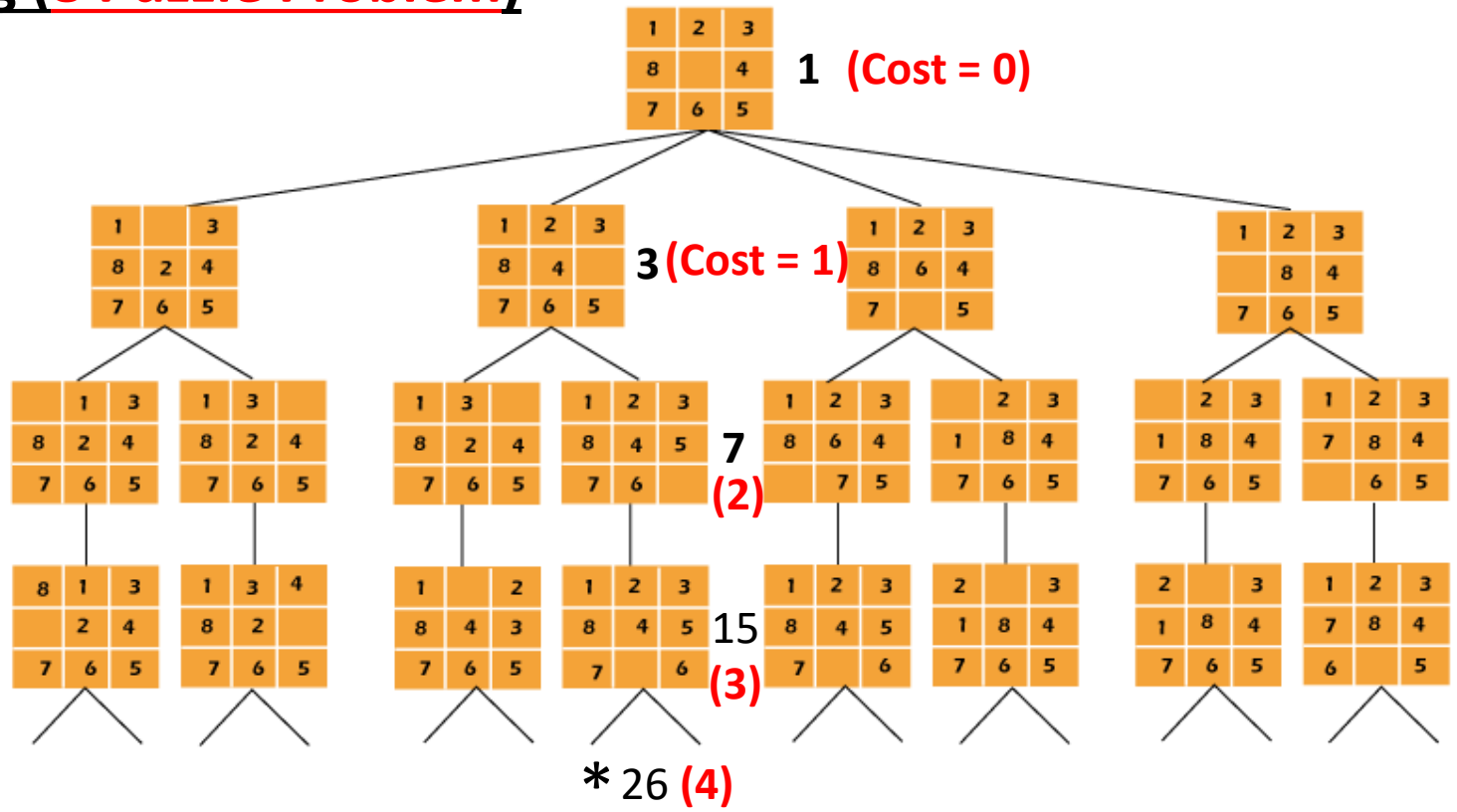
Initial/Start State

1	2	3
8		4
7	6	5

Goal State

Basics of State Space Modelling (8 Puzzle Problem)

5. SOLUTION(s), COSTS



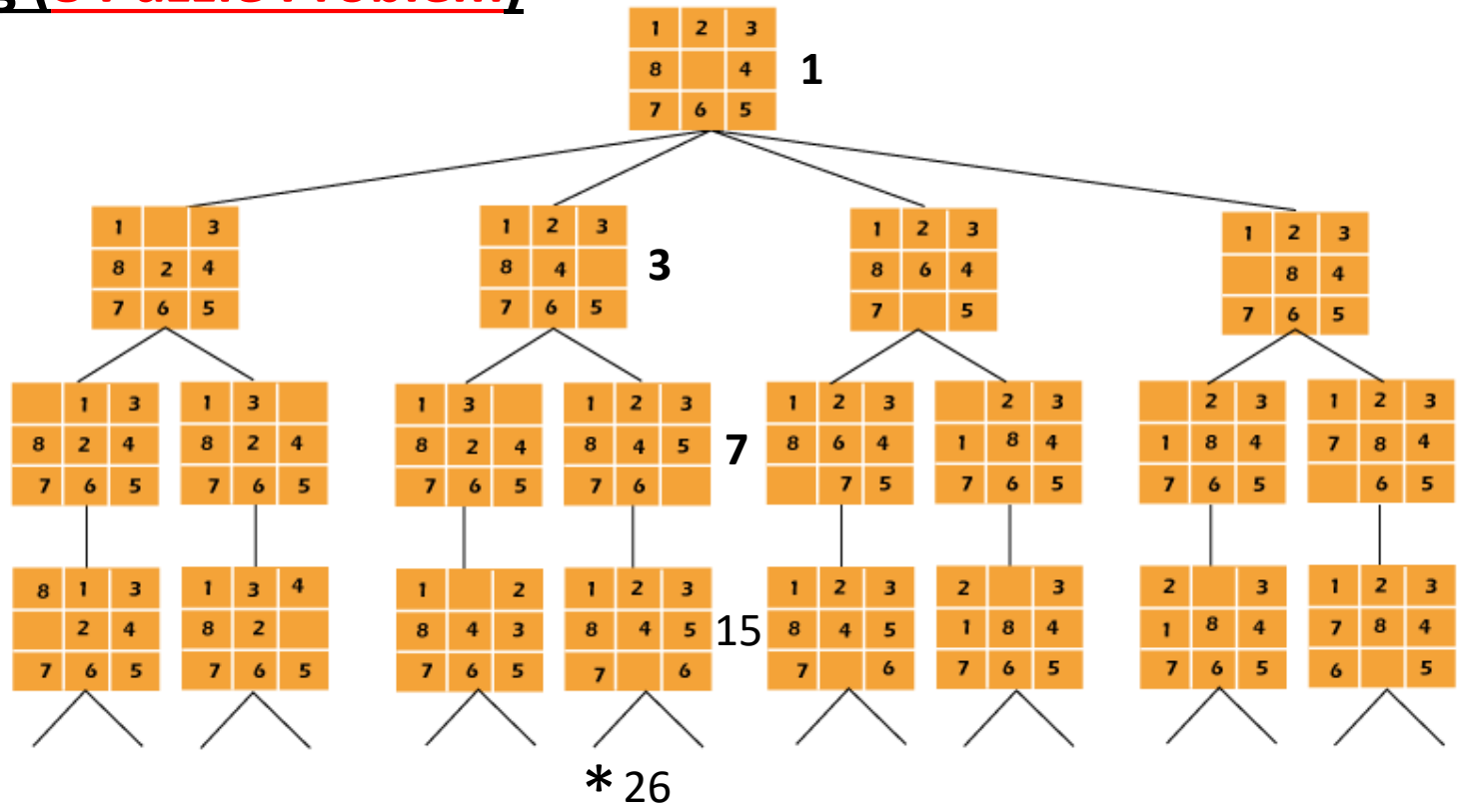
Solution: Path (1 – 3 – 7 – 15 – 26)
Cost: 4

Cost: It refers to the number of steps required to reach from one state to another desired state

Basics of State Space Modelling (8 Puzzle Problem)

6. SEARCH ALGORITHMS

Searching: It is a process of exploring Goal state from Initial state.



Solution: Path (1 – 3 – 7 – 15 – 26)
Cost: 4

State Space Modelling (bounded inc-and-square)

State: S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

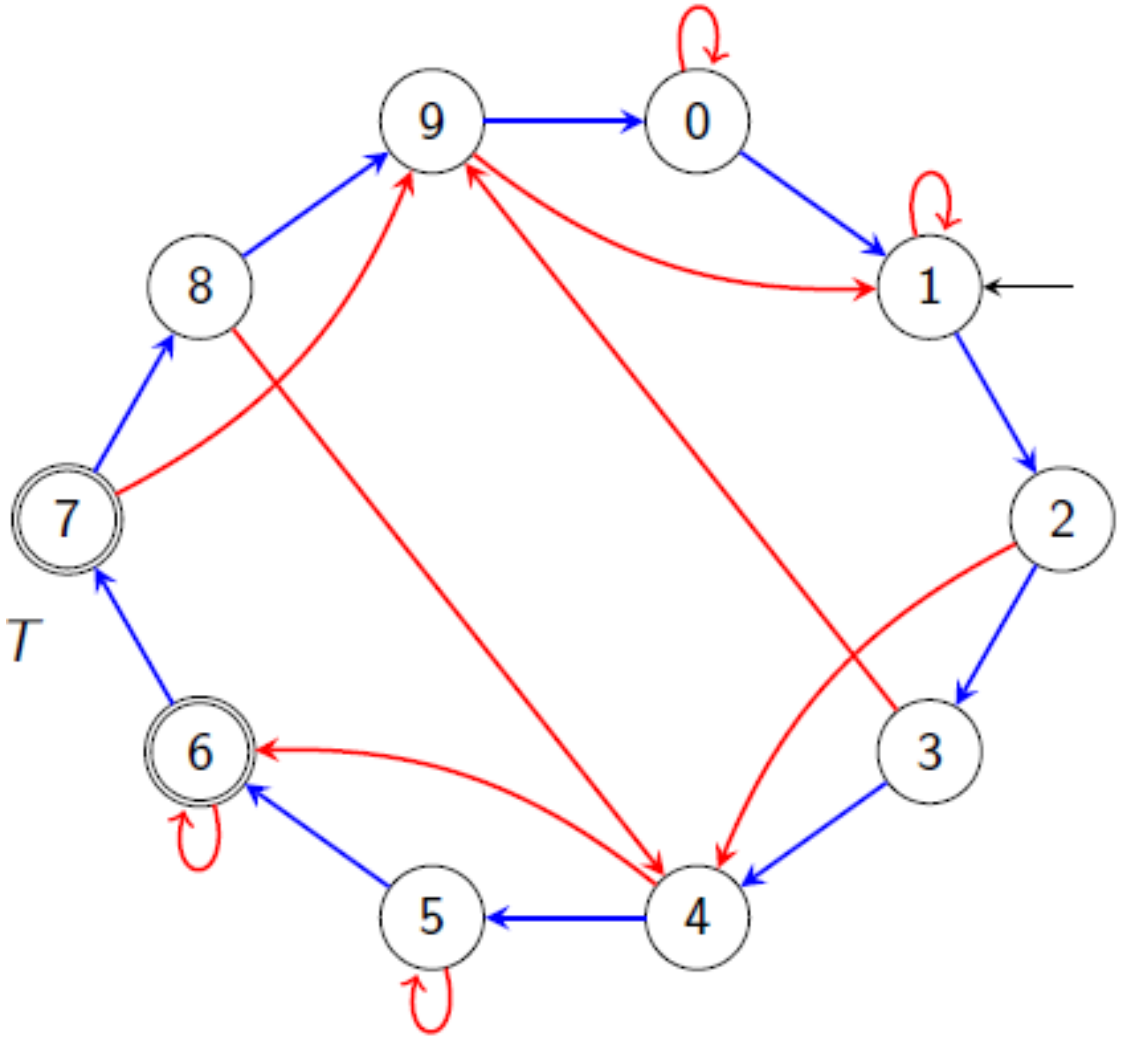
Action: Act = {Inc, Sqr}

State transformation: $\langle i, inc, (i+1) \mod 10 \rangle \in T$
$\langle i, sqr, i^2 \mod 10 \rangle \in T$

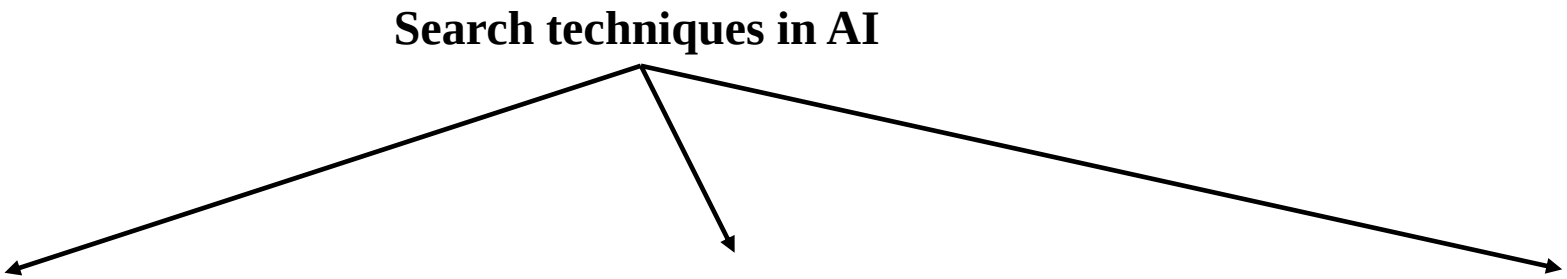
Initial State: S0 = 1

Final State: S* = {6, 7}

Cost: Cost(Inc) = Cost(Sqr) = 1



Problem solving by search:



Uninformed search (blind search) algorithms:

It is a search process that explores the state space (problem space) without any specific knowledge or information about the problem other than the initial state and the possible actions to take.

In the exploration process, no information other than basics state space modelling (i.e. states, actions, transformation, state space, cost) is provided.

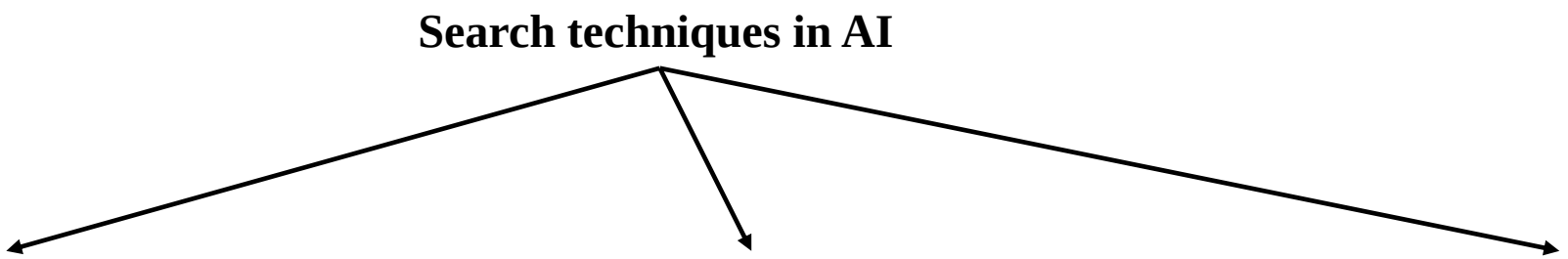
Informed search algorithms:

It uses additional information (Heuristic) to guide the search process

Adversarial search algorithms:

It is search process applied to a situation where one search agent is searching for goal while another agent prepares against other agent.

Problem solving by search:



Uninformed search (blind search) algorithms:

Breadth-first search
Uniform cost search
Depth-first search & Depth-limited search
Iterative deepening search

Informed search algorithms:

Greedy best-first search
A*
AO*
Hill Climbing,
Simulated Annealing

Adversarial search algorithms:

MIN-MAX Algorithm
Alpha-Beta Cut-off algorithm

Breadth-first search (BFS)

Steps:

- Select the “Initial state” and put it on OPEN list (Queue).
- If initial state is the goal state, then stop and return path and cost. If it is in CLOSE list, then discard it. Else, explore all its adjacent states and put them on OPEN. Remove the “initial state” from OPEN and place it on CLOSE list.
- Then, select the next state from queue in FIFO manner. If it is the goal state, then stop and return path and cost. Else, explore all its adjacent states and put them on OPEN. Remove the “initial state” from OPEN and place it on CLOSE list.
- Repeat the same steps until the goal state is obtained or OPEN list is empty.

Breadth-first search (BFS) on bounded inc-and-square problem

State: S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

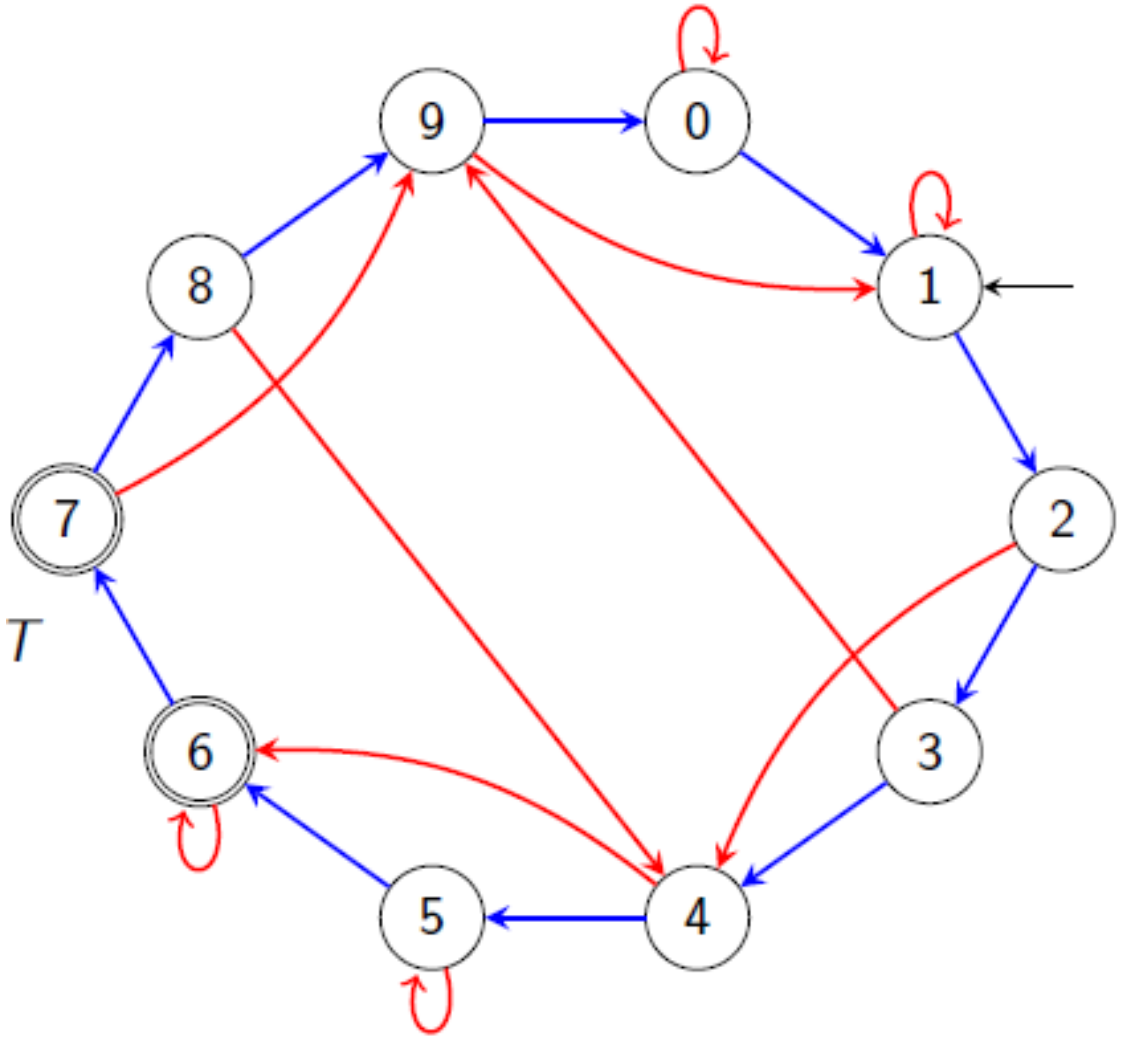
Action: Act = {Inc, Sqr}

State transformation: $\langle i, inc, (i+1) \mod 10 \rangle \in T$
$\langle i, sqr, i^2 \mod 10 \rangle \in T$

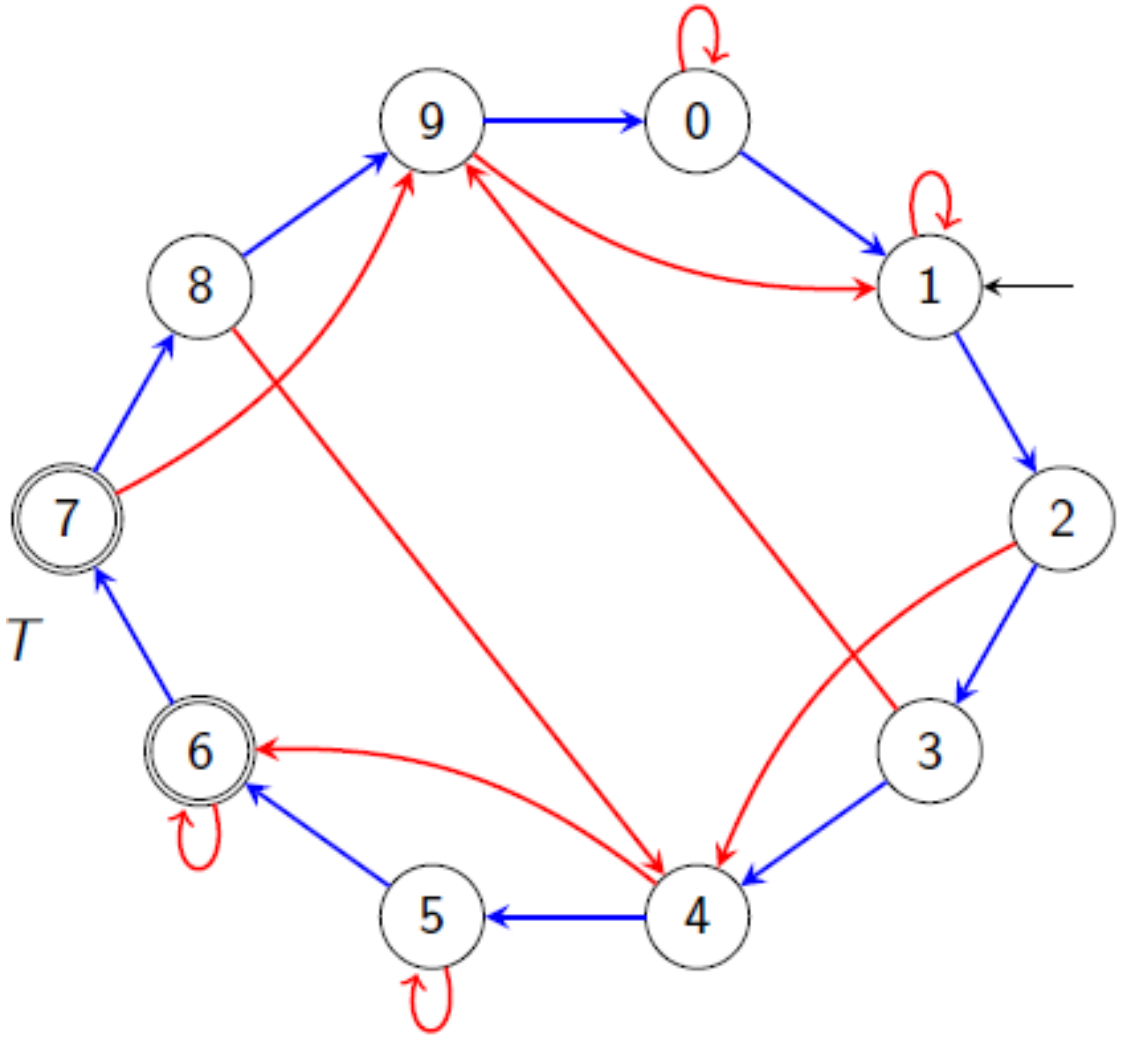
Initial State: S0 = 1

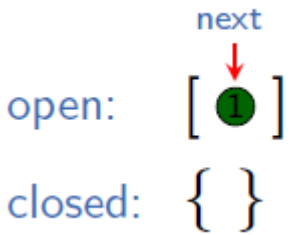
Final State: S* = {6, 7}

Cost: Cost(Inc) = Cost(Sqr) = 1

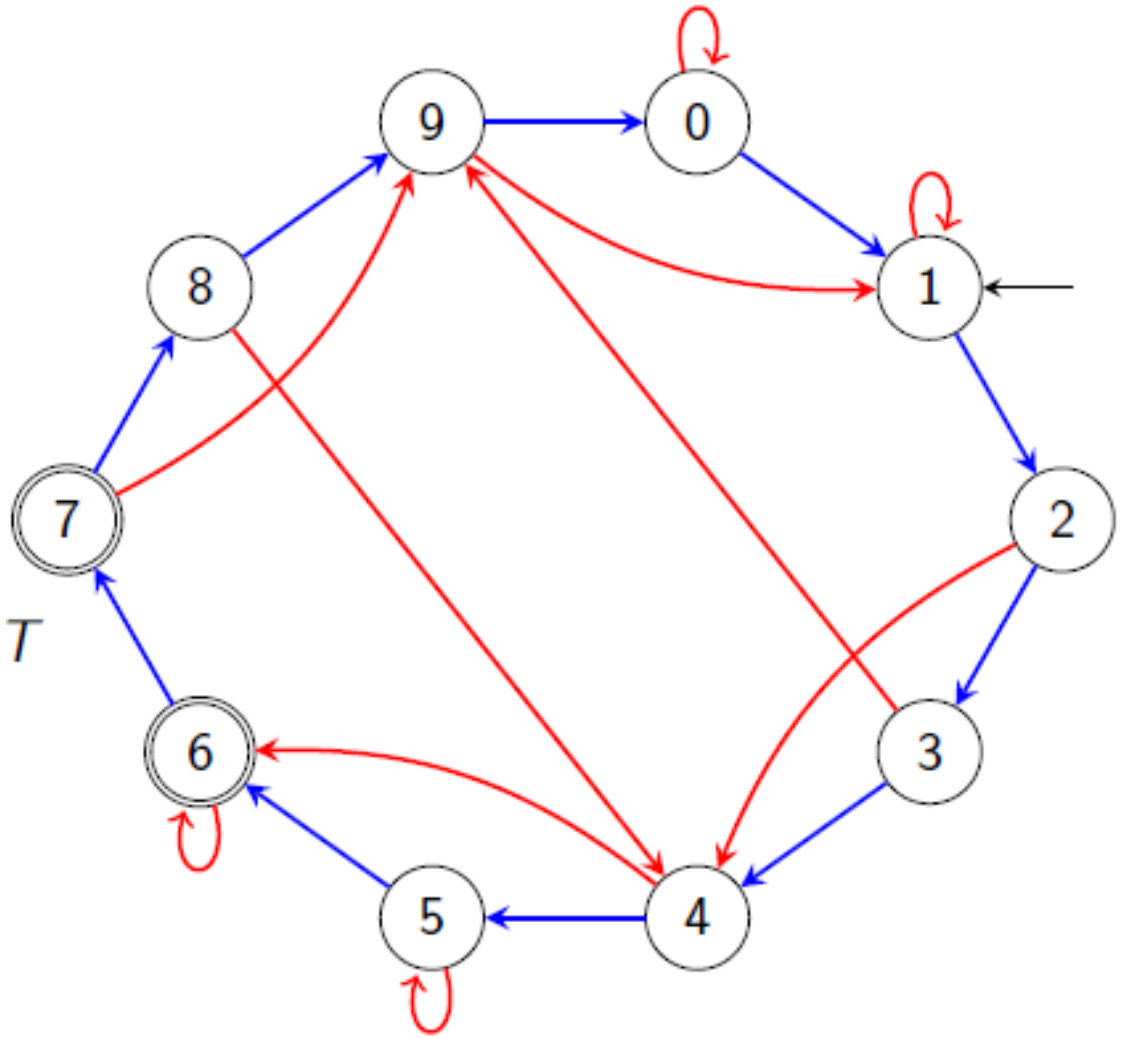


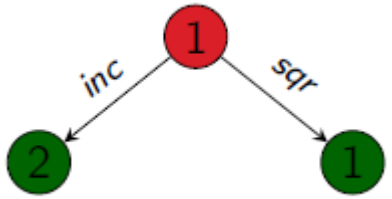
Breadth-first search (BFS)

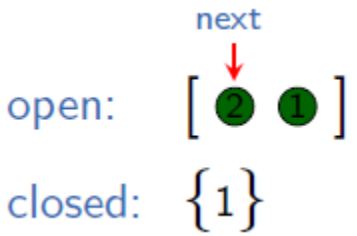




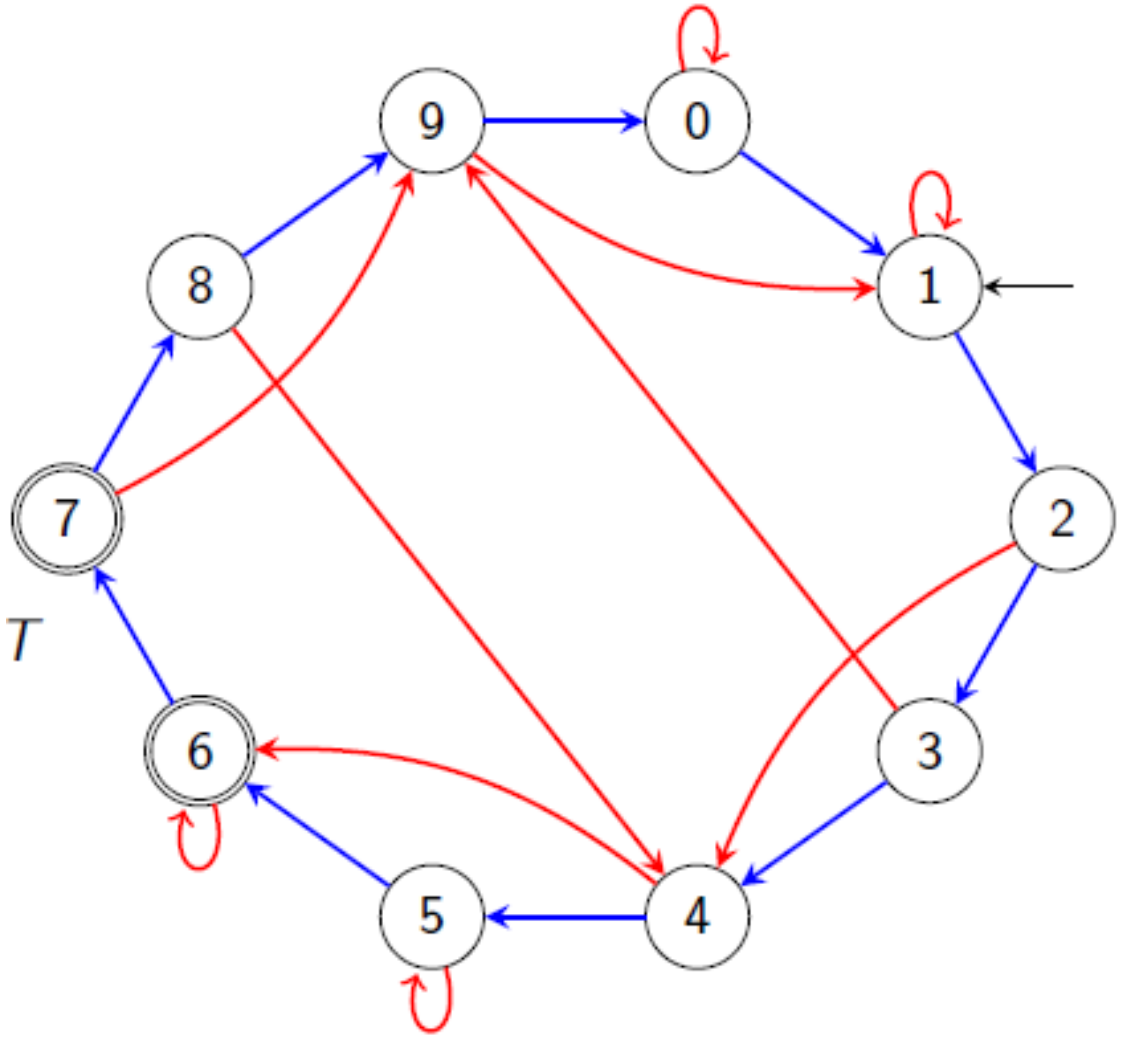
Breadth-first search (BFS)

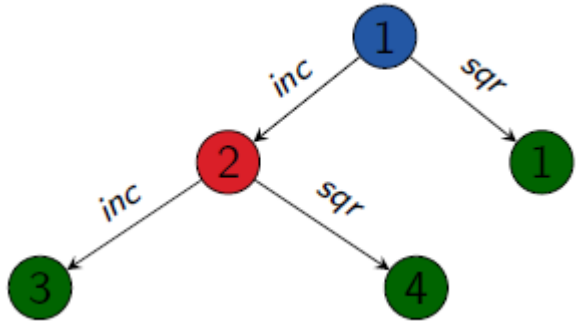


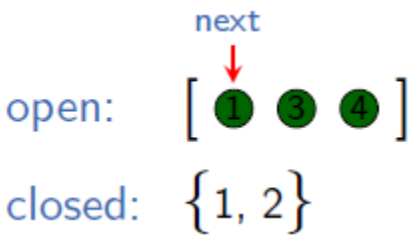




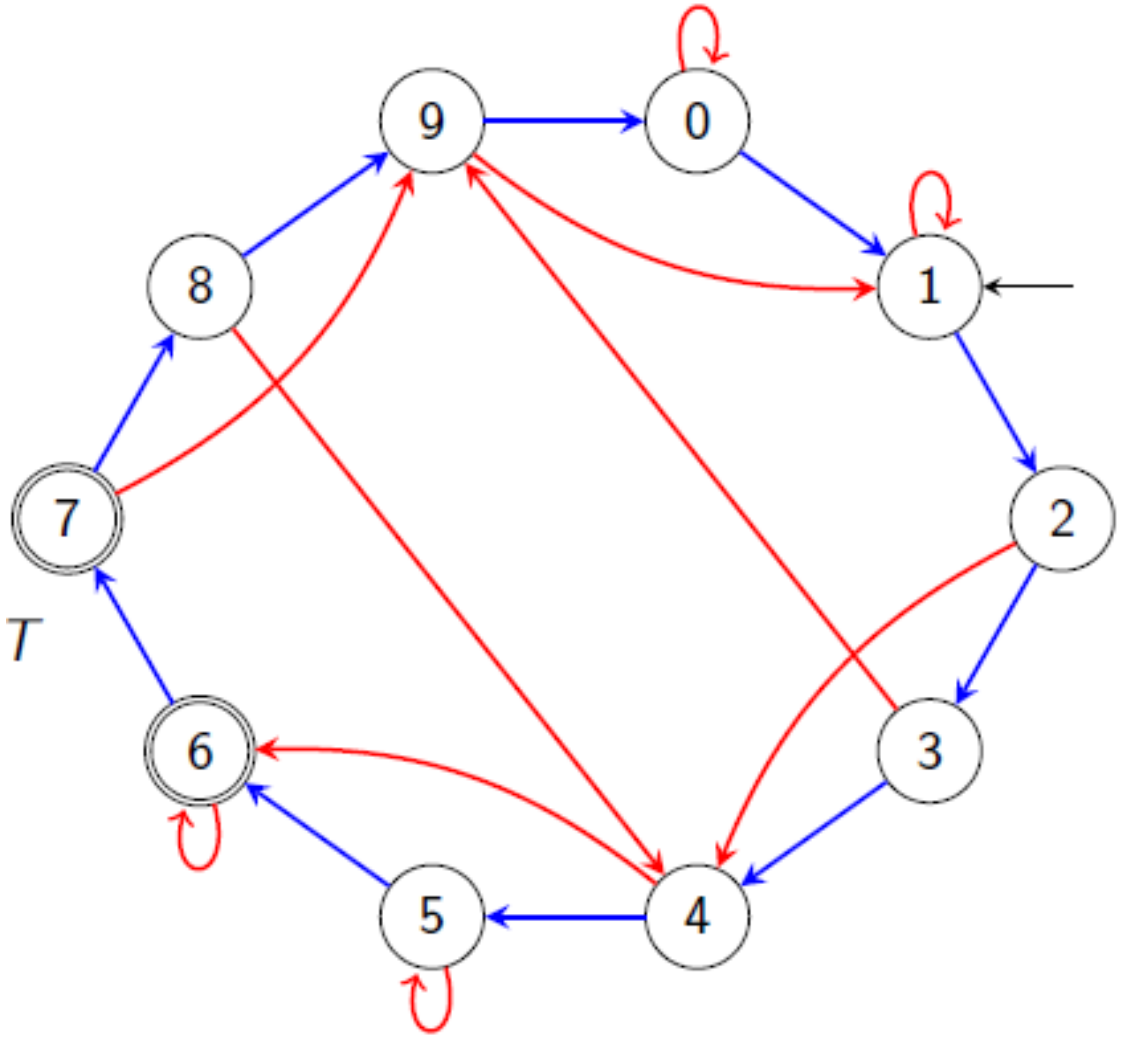
Breadth-first search (BFS)

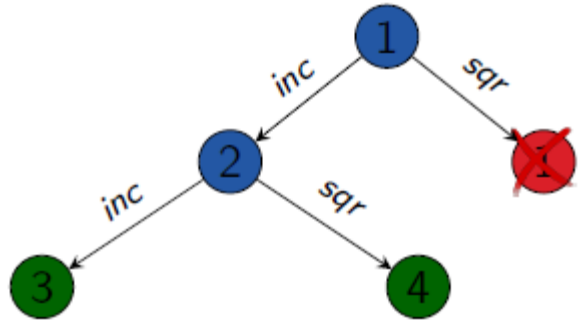


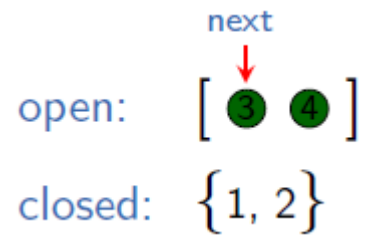




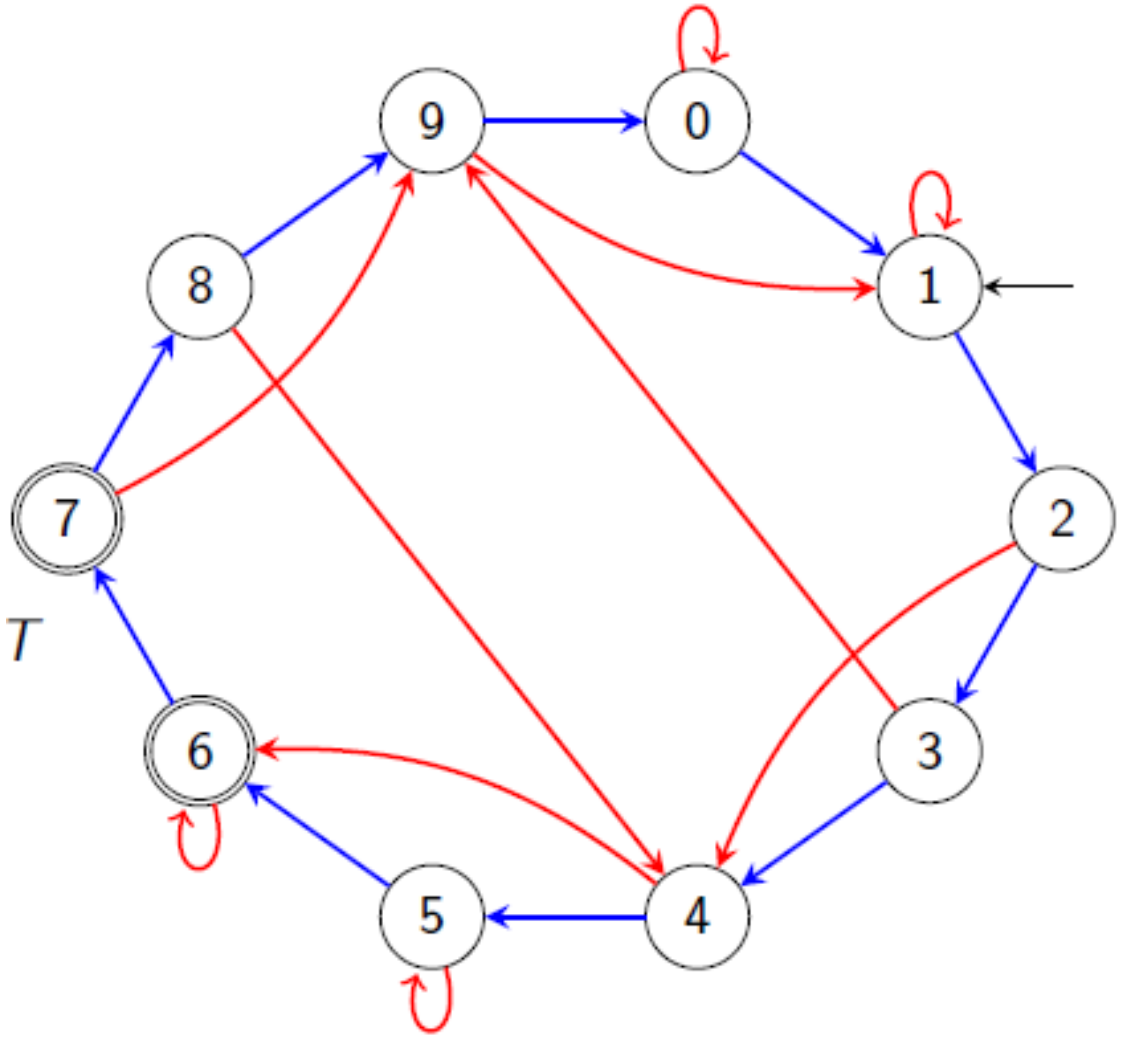
Breadth-first search (BFS)

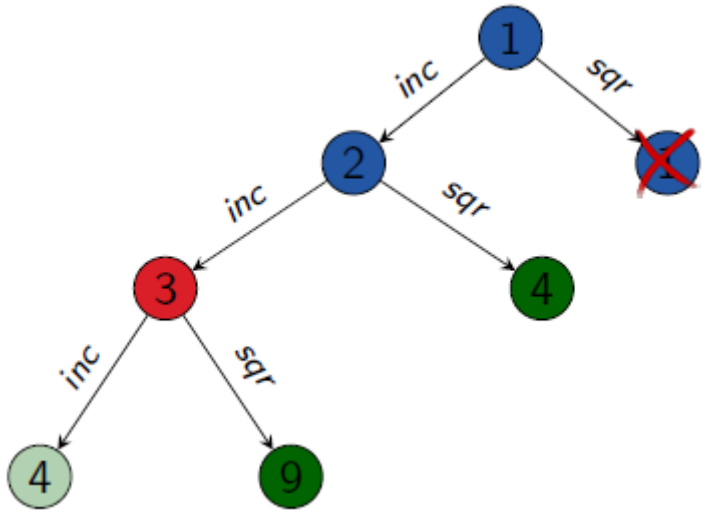


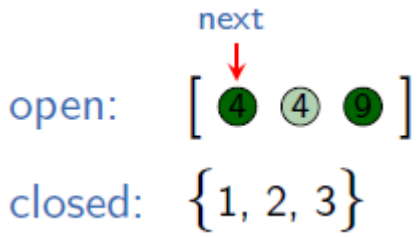




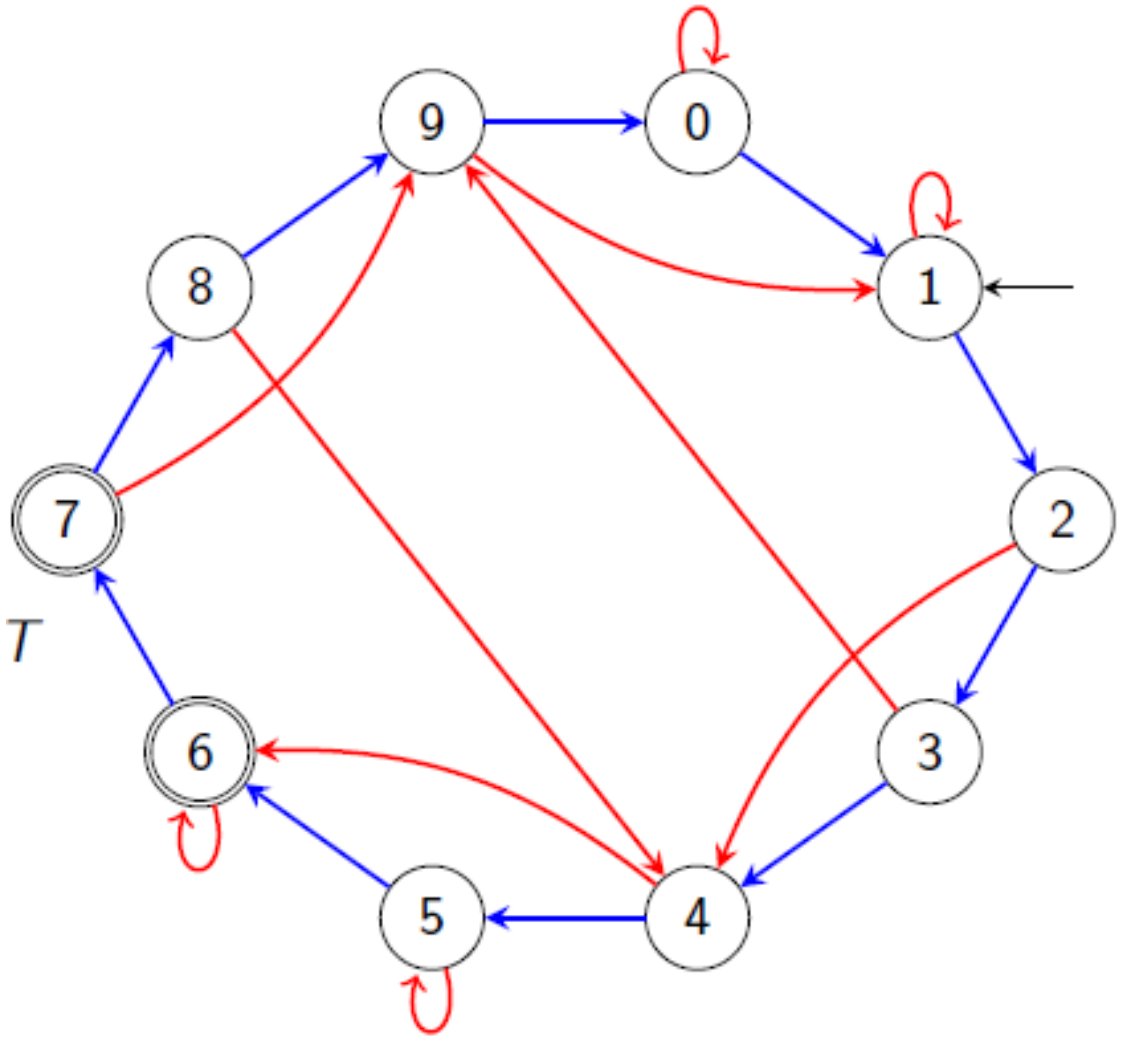
Breadth-first search (BFS)

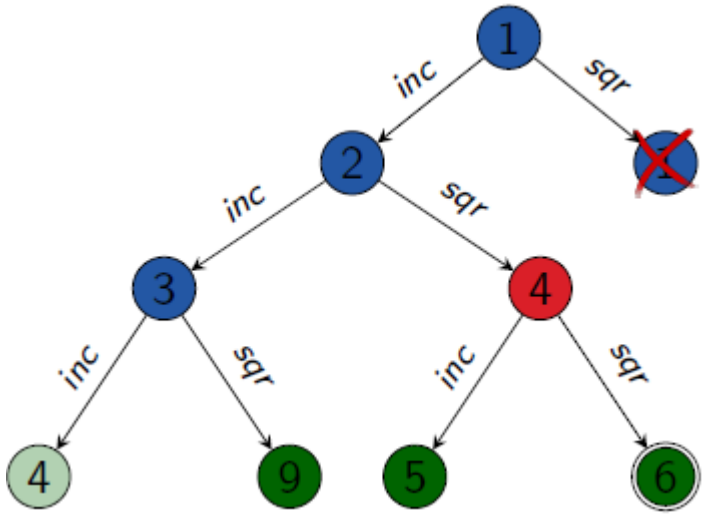


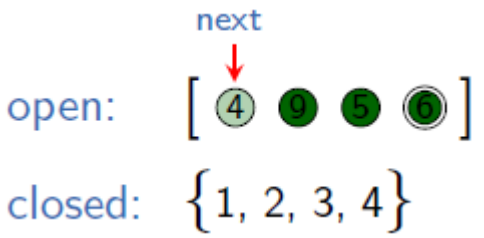




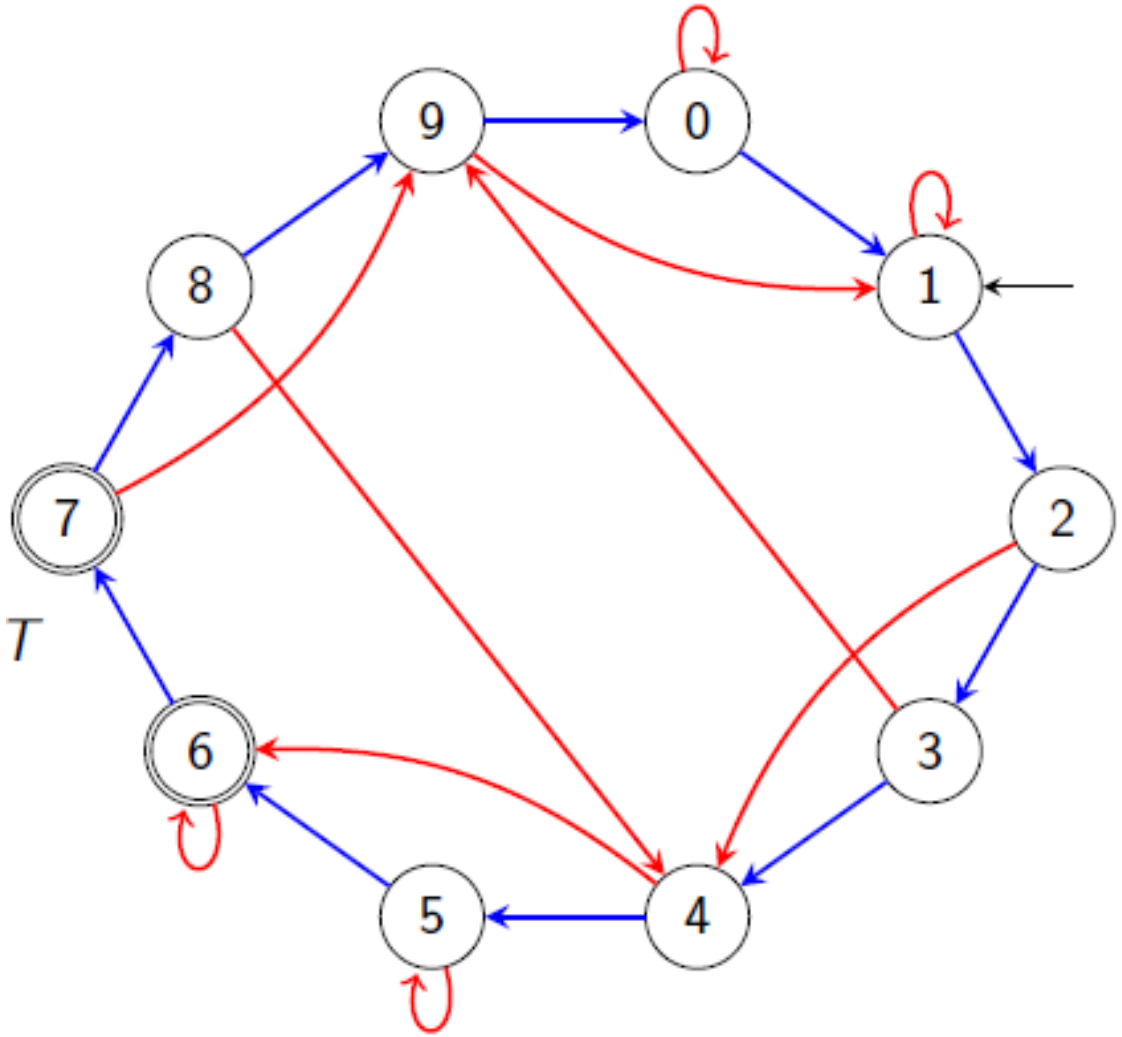
Breadth-first search (BFS)

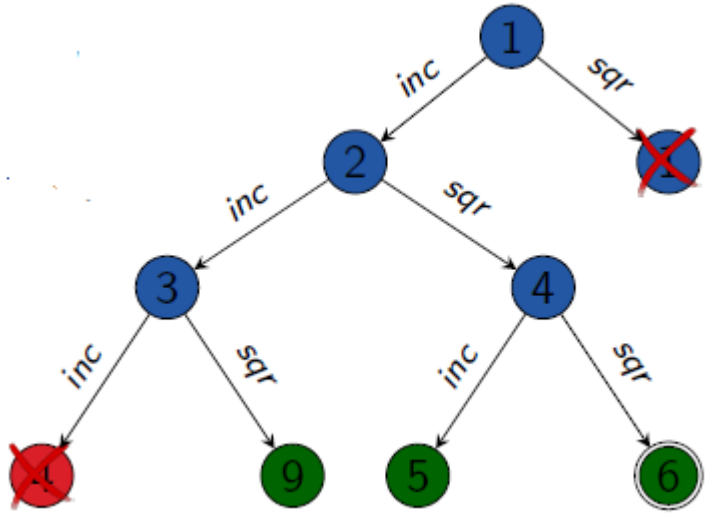


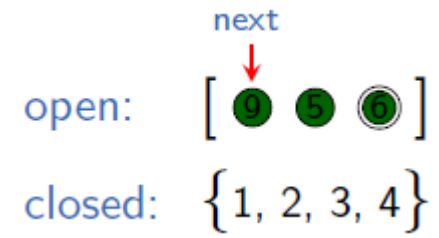




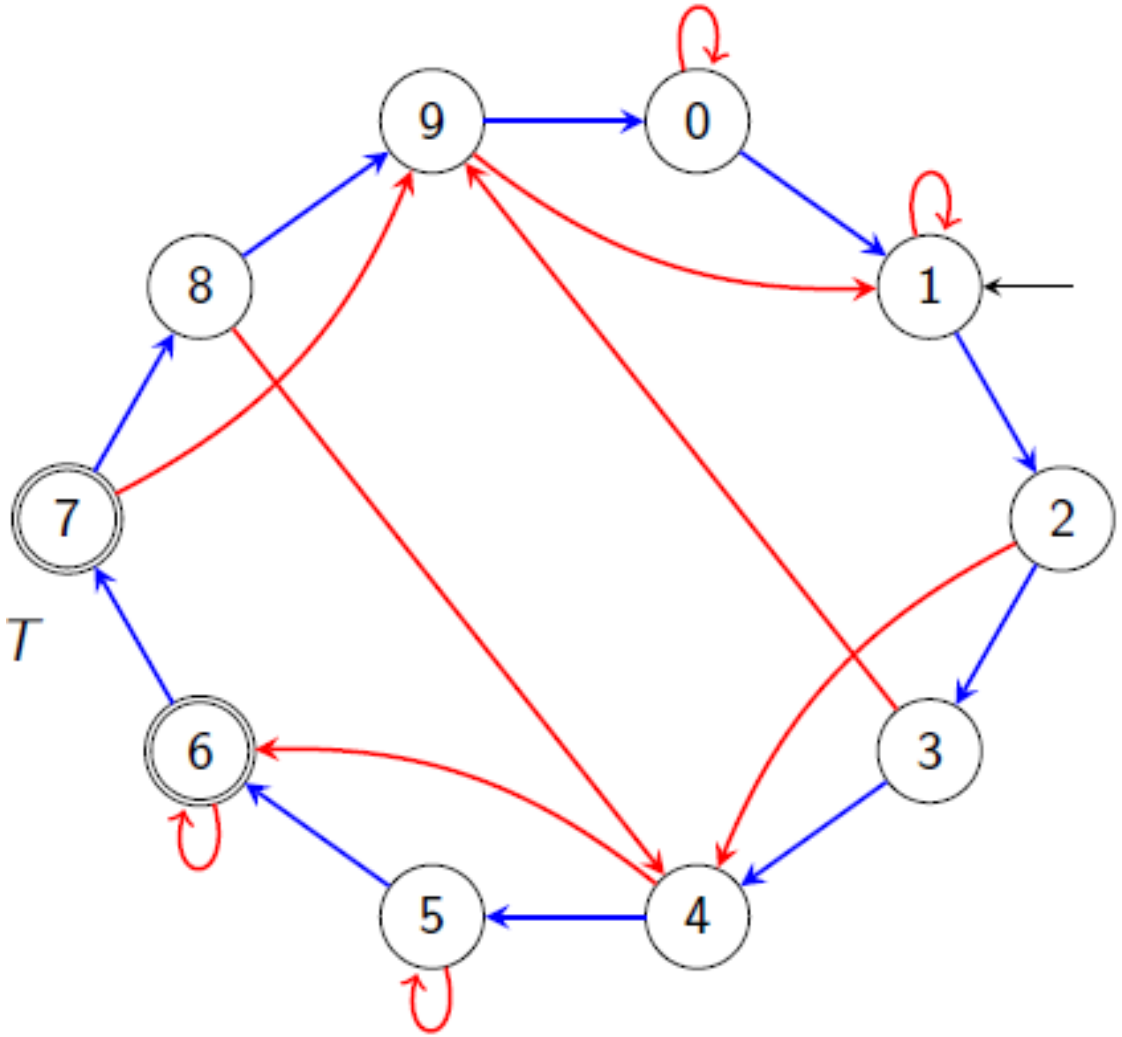
Breadth-first search (BFS)

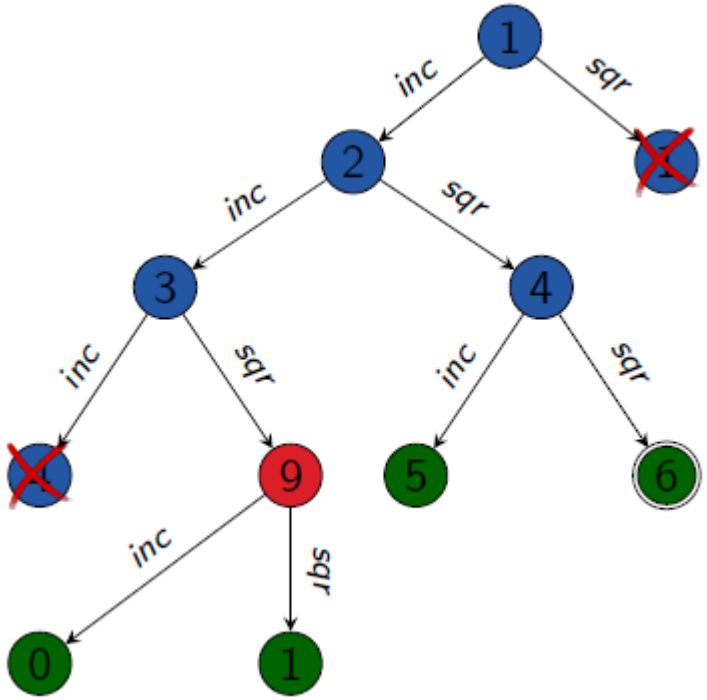






Breadth-first search (BFS)



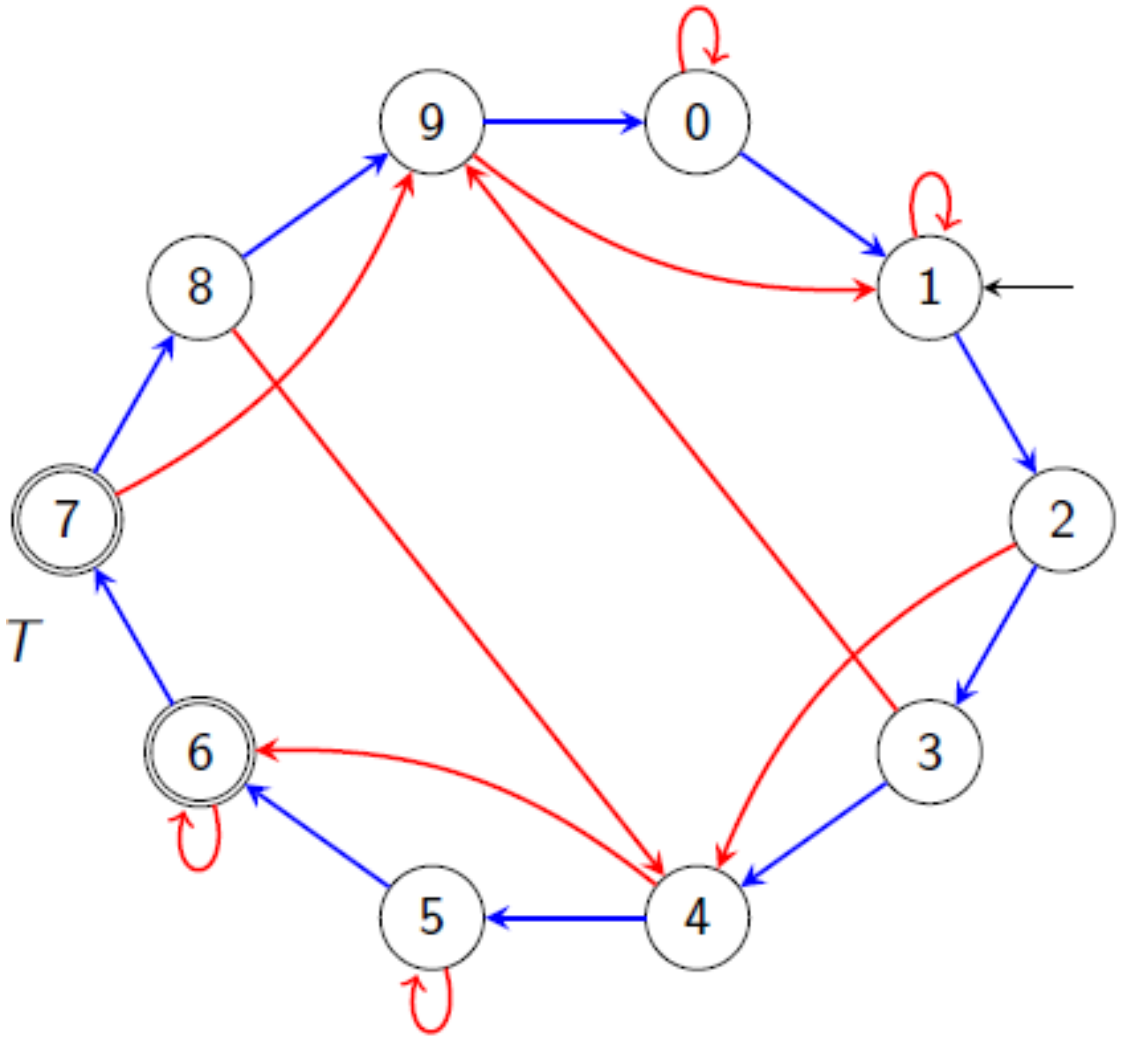


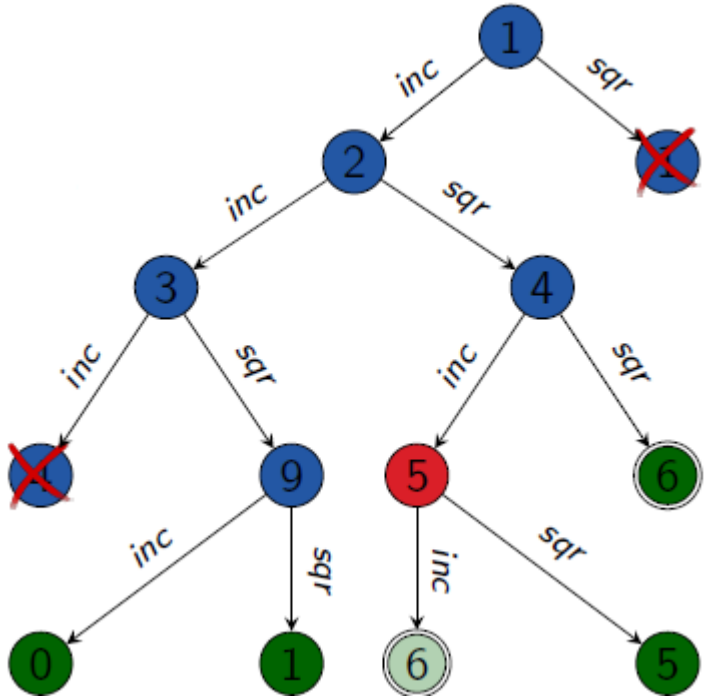
next

open: [5 6 0 1]

closed: {1, 2, 3, 4, 9}

Breadth-first search (BFS)



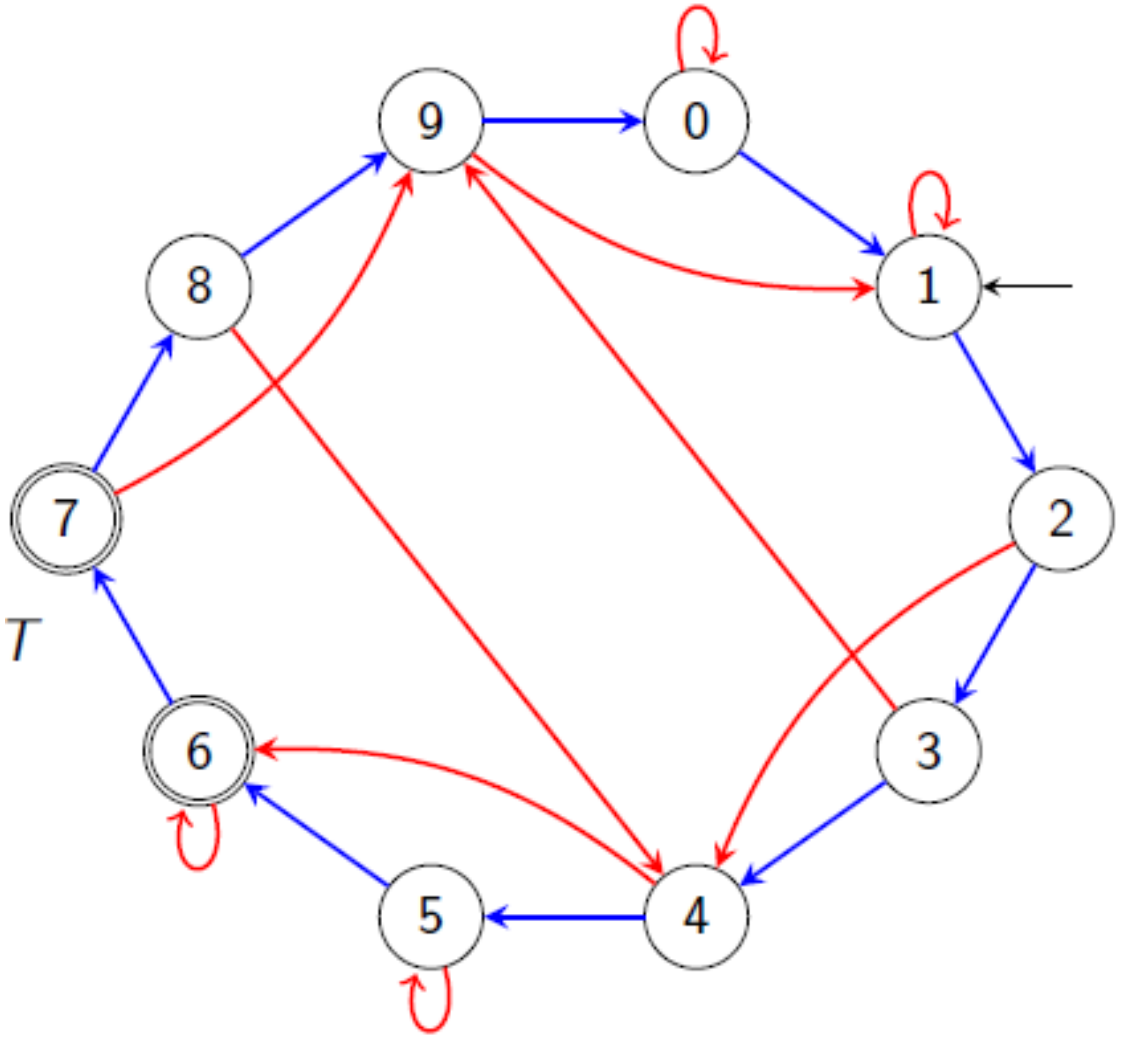


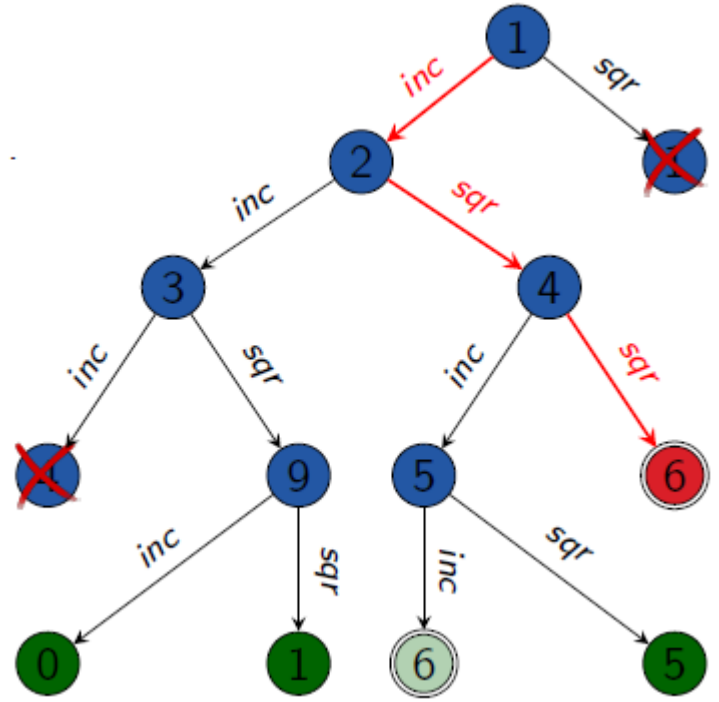
next

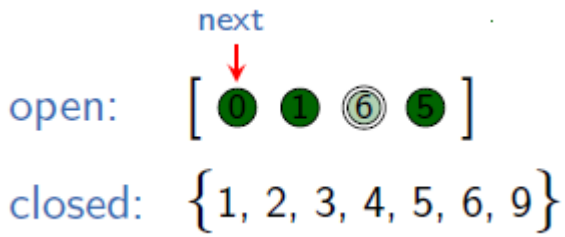
open: [6 0 1 6 5]

closed: {1, 2, 3, 4, 5, 9}

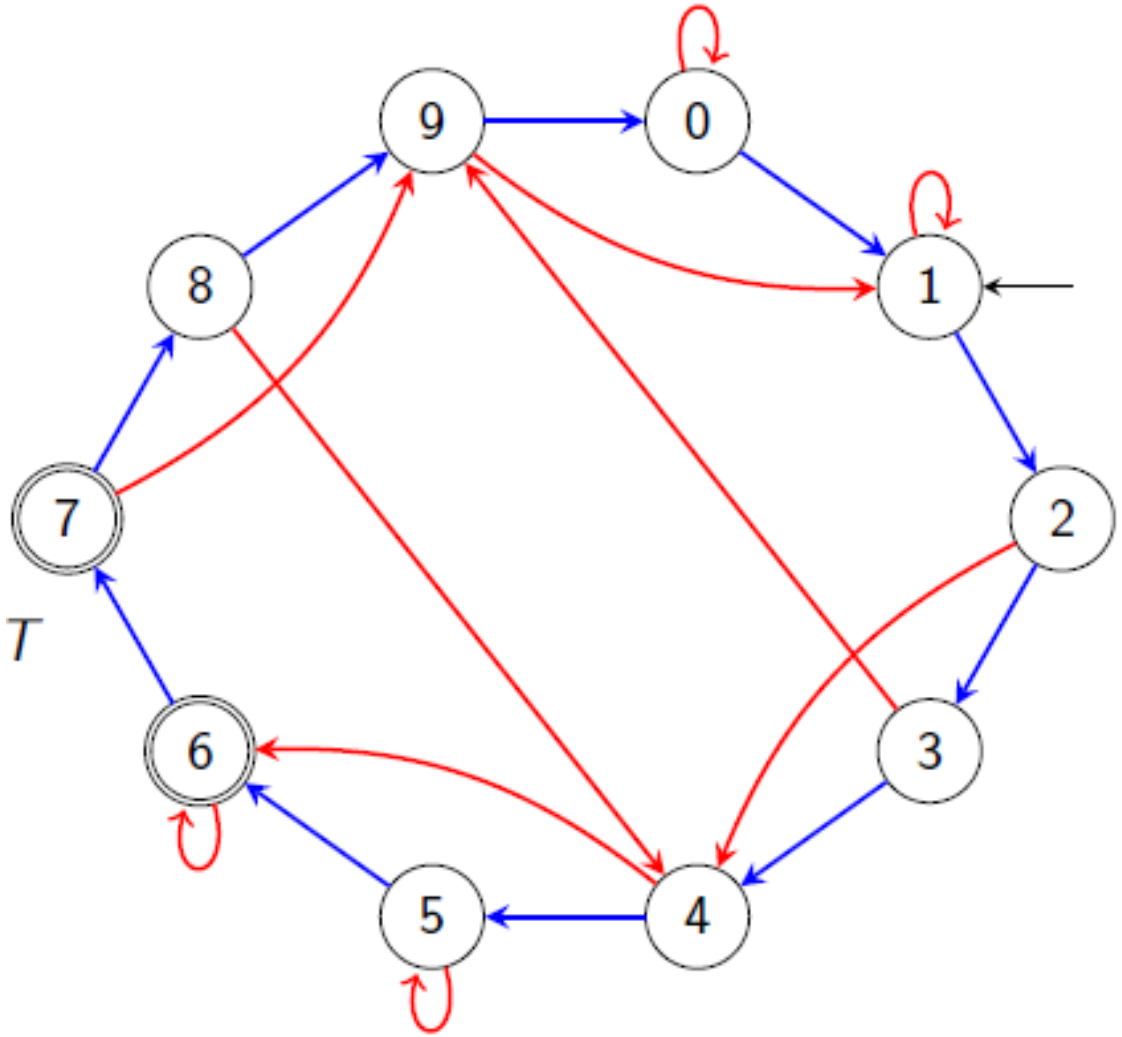
Breadth-first search (BFS)







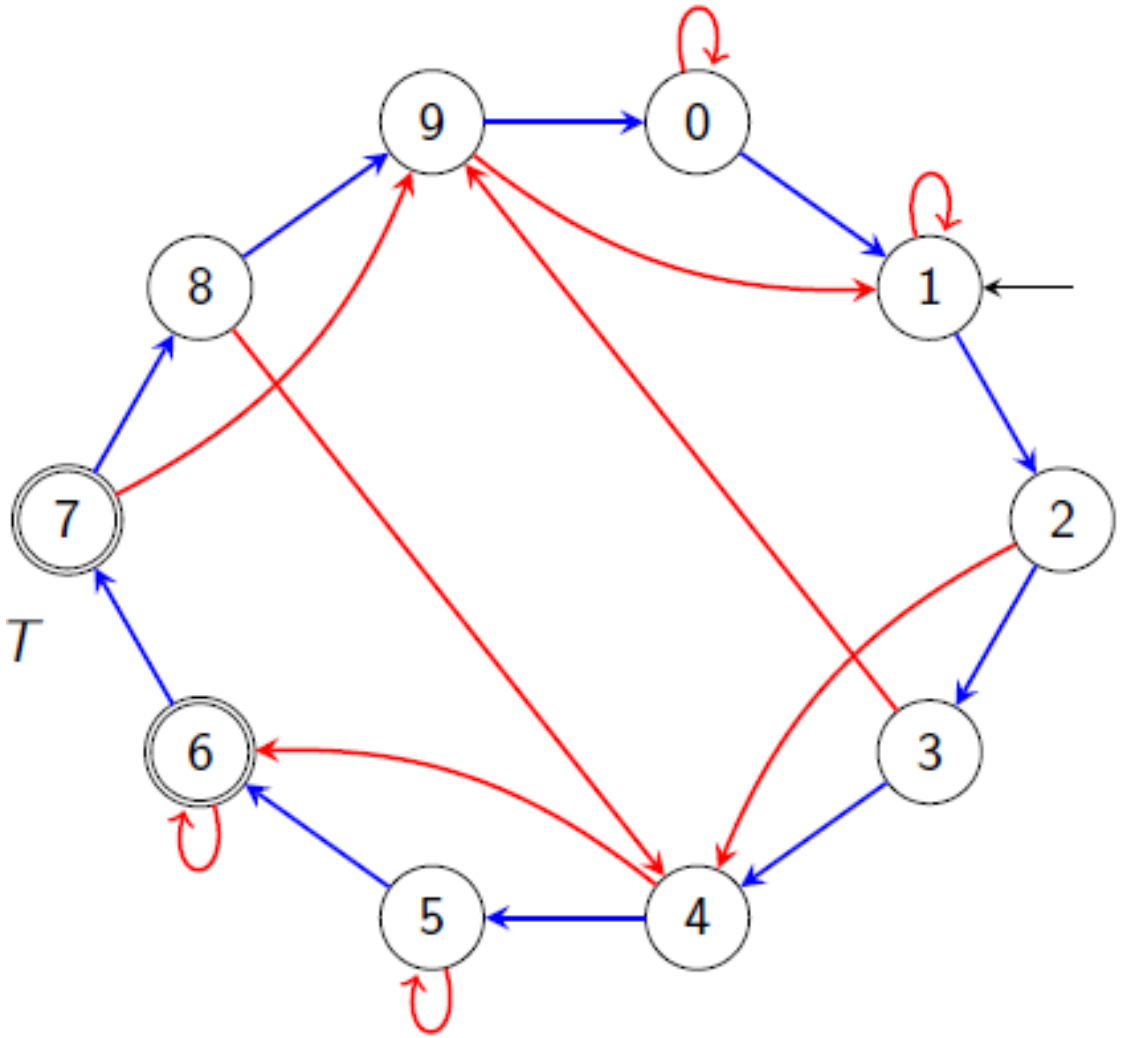
Breadth-first search (BFS)

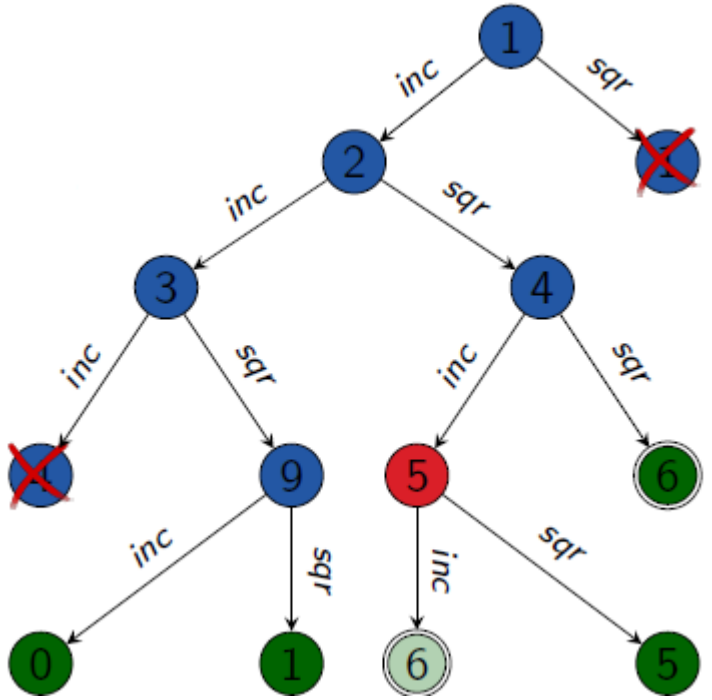


Various variants and optimizations:

- Use a closed list?
- When to update closed list?
- When to perform duplicate check?
- When to perform goal test?

Breadth-first search (BFS)



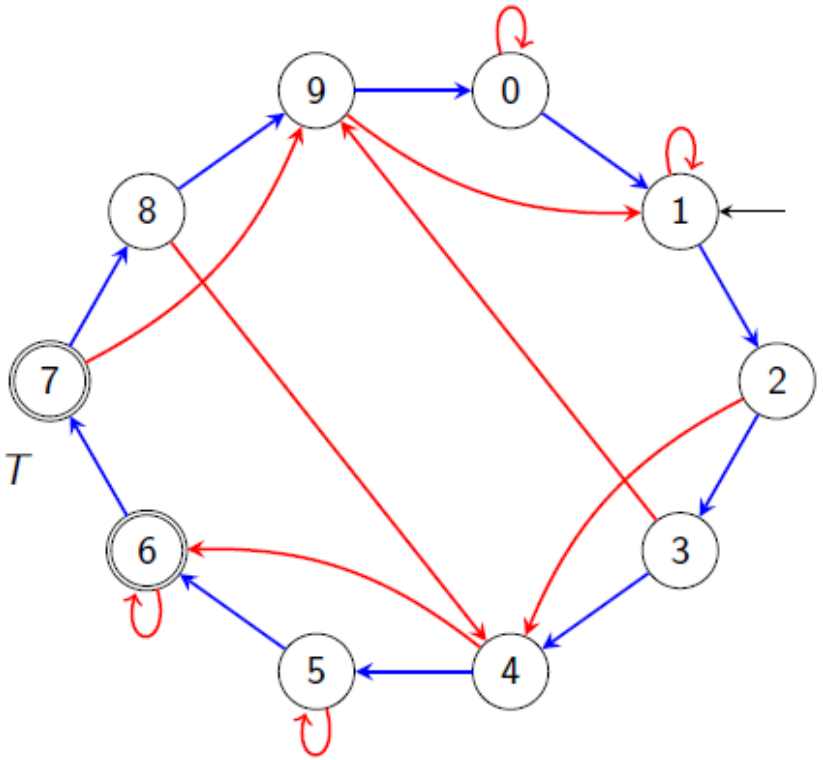


next

open: [6 0 1 6 5]

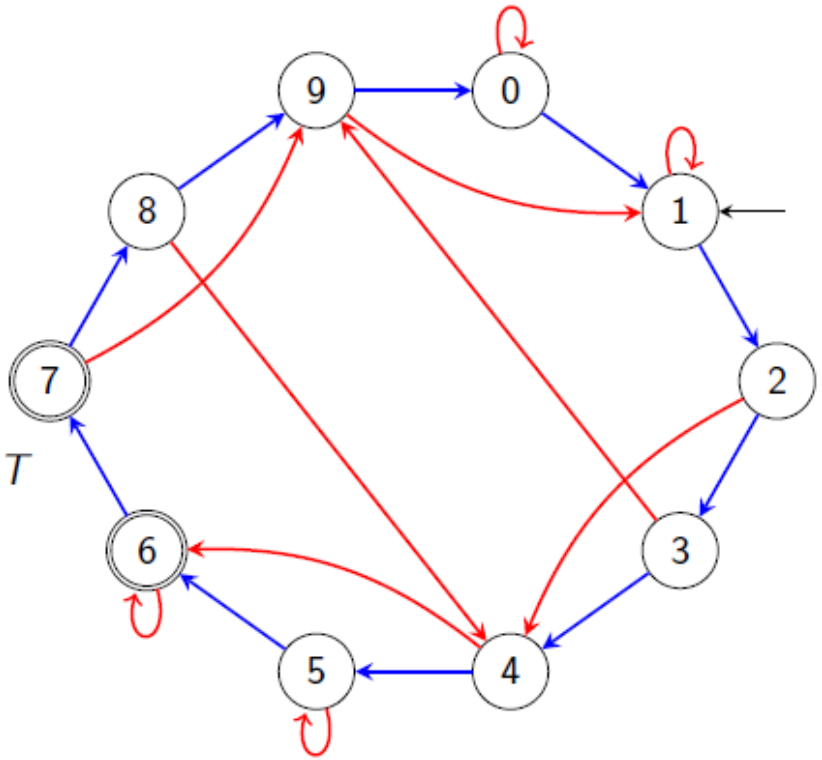
closed: {1, 2, 3, 4, 5, 9}

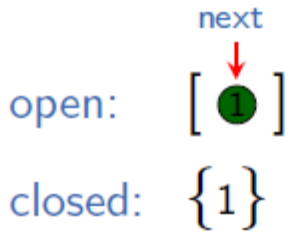
Breadth-First Search (with Duplicate Elimination)



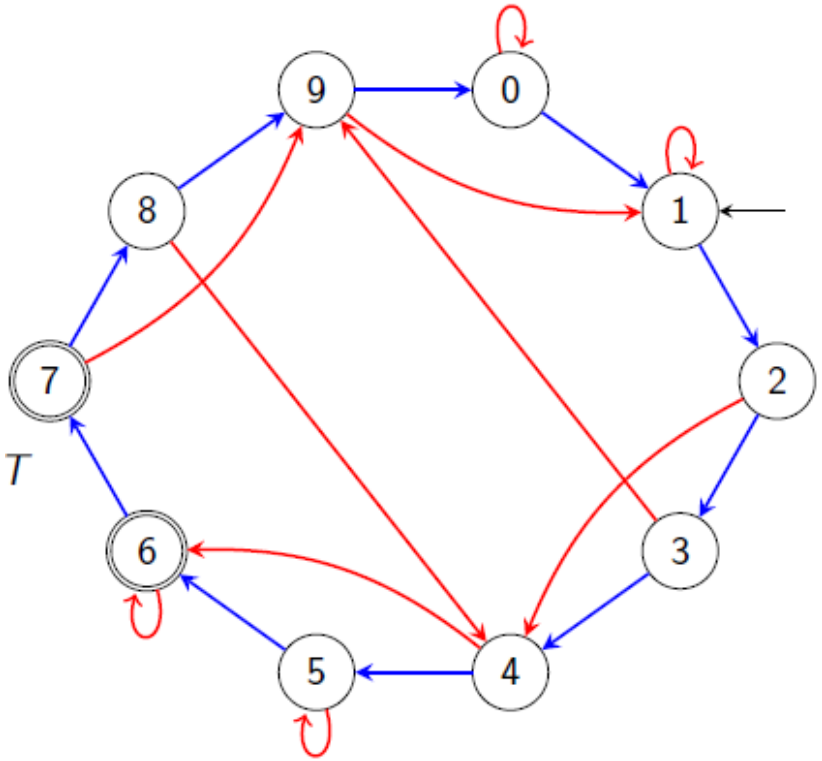
- Eliminate duplicate state
- First check (early goal tests) and then explore
- we should perform duplicate tests against the closed list and updates of the closed lists as early as possible

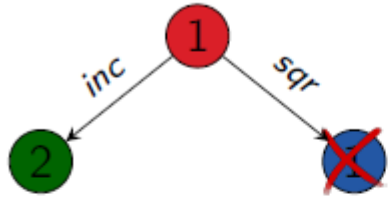
Breadth-First Search (with Duplicate Elimination)





Breadth-First Search (with Duplicate Elimination)



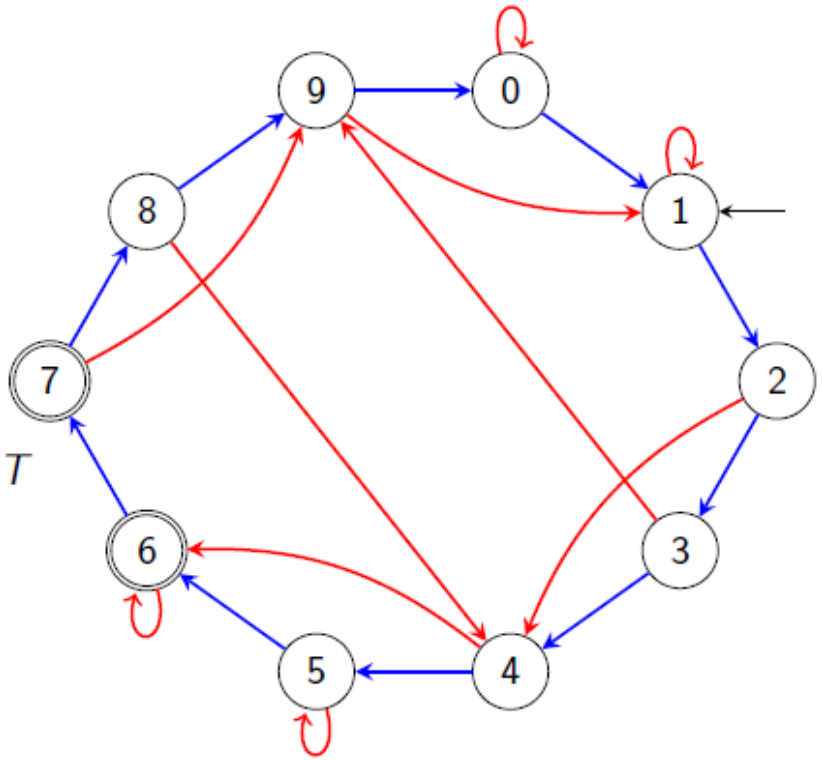


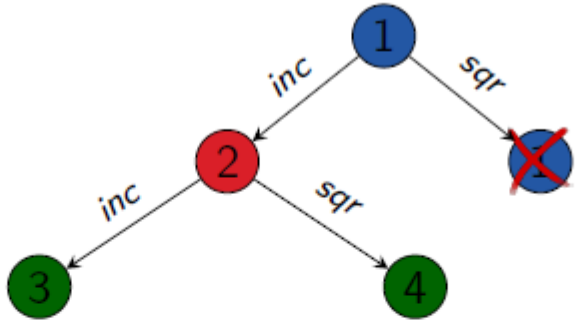
next

open: [2]

closed: {1,2}

Breadth-First Search (with Duplicate Elimination)



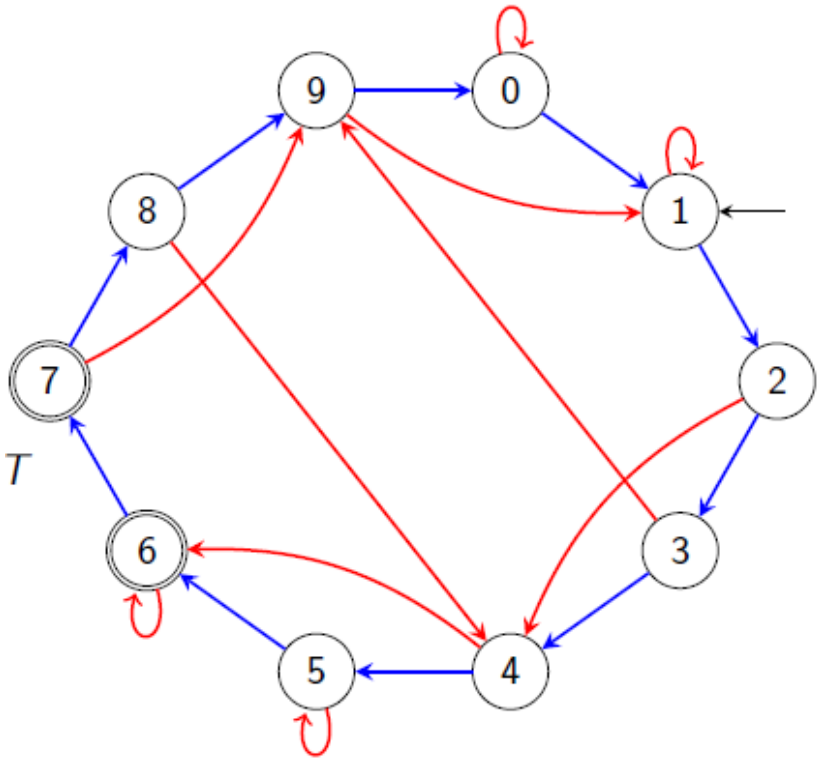


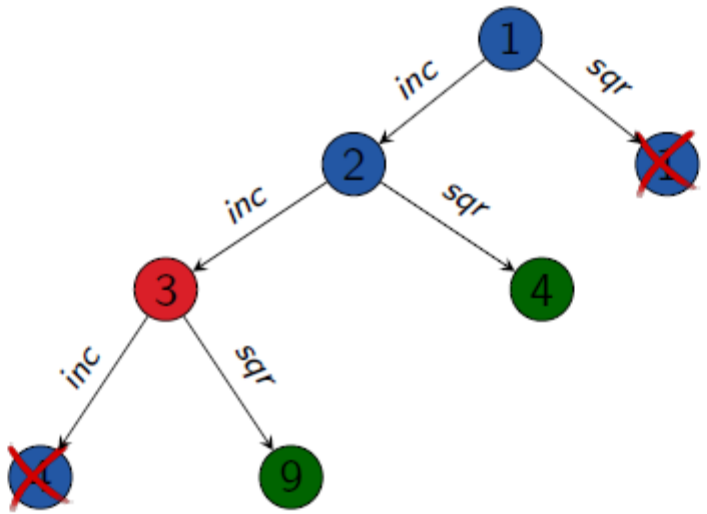
next

open: [3 4]

closed: {1, 2, 3, 4}

Breadth-First Search (with Duplicate Elimination)



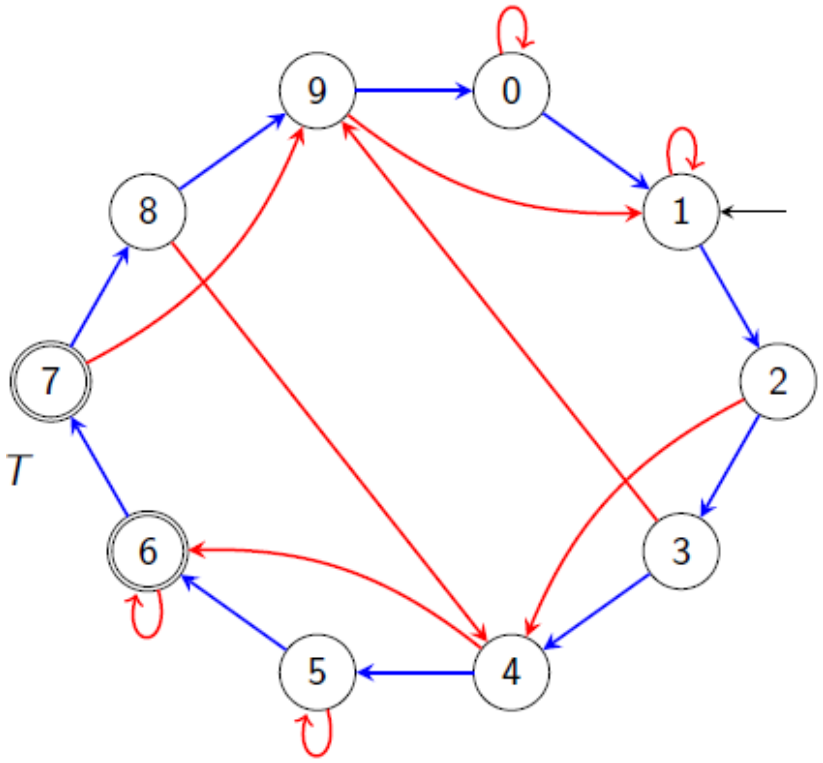


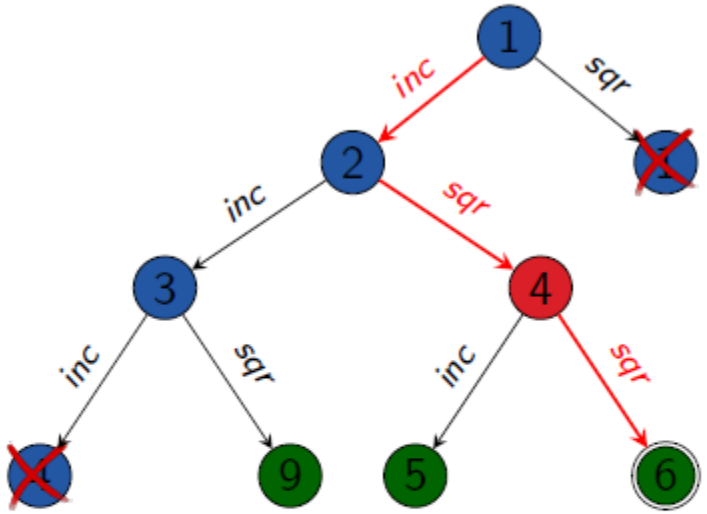
next

open: [4 9]

closed: {1, 2, 3, 4, 9}

Breadth-First Search (with Duplicate Elimination)



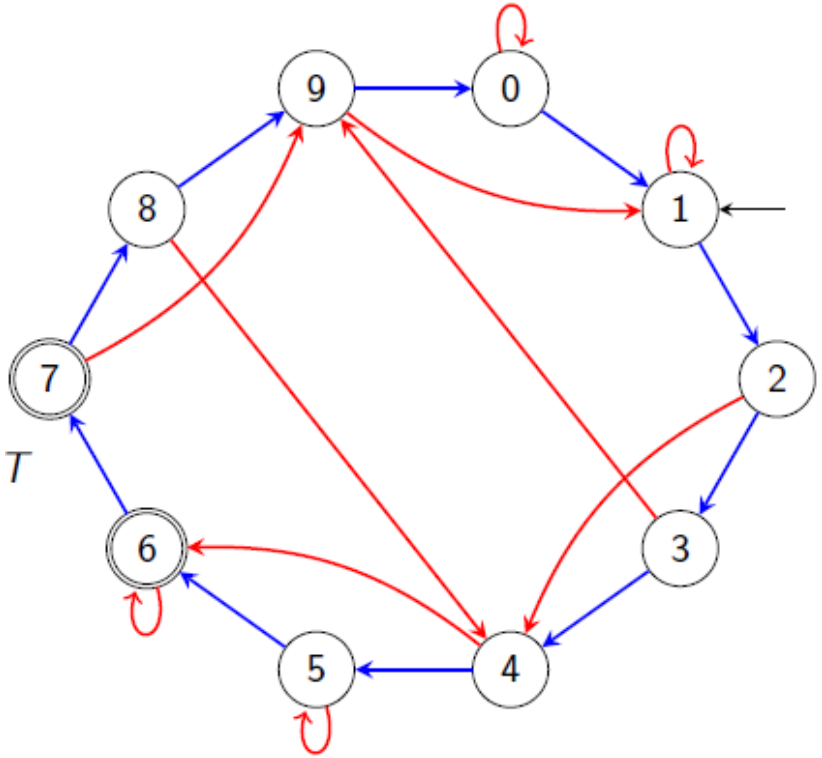


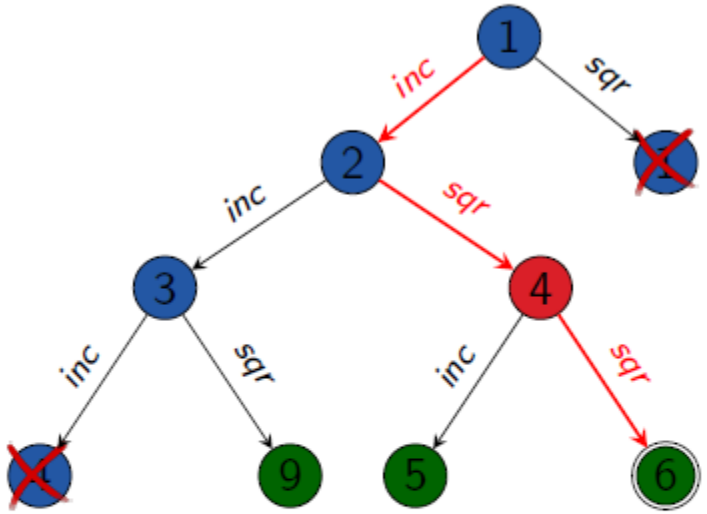
next

open: [9 5]

closed: {1, 2, 3, 4, 5, 6, 9}

Breadth-First Search (with Duplicate Elimination)



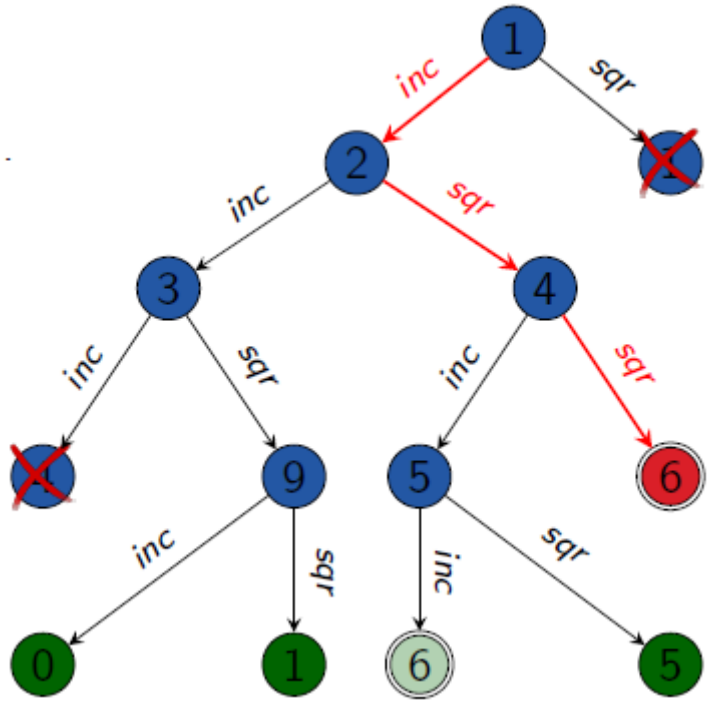


next

open: [9 5]

closed: {1, 2, 3, 4, 5, 6, 9}

BFS (without duplicate elimination)



next

open: [0 1 6 5]

closed: {1, 2, 3, 4, 5, 6, 9}

Breadth-First Search

Properties of Breadth-first Search:

- BFS is complete.
- BFS (both variants) is optimal (if all actions have the same cost, but not in general)

Breadth-First Search

Properties of Breadth-first Search:

Advantages of BFS:

- Complete
- Much (!) more efficient if there are many duplicates
- Simpler
- Less overhead (time/space) if there are few duplicates

Breadth-First Search

Complexity:

The following result applies to both BFS variants:

Let b be the branching factor and d be the minimal solution length of the given state space. Let $b \geq 2$.

Then the time complexity of breadth-first search is

$$1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

It follows that the space complexity of both BFS variants also is $O(b^d)$ (if $b \geq 2$). (Why?)

Breadth-First Search

Complexity:

Rubik's cube:

- branching factor: ≈ 13
- typical solution length: 18

Complexity = 13^{18}

Uniform Cost Search

Breadth-first search optimal if all action costs equal, Otherwise no optimality guarantee

Consider bounded inc-and-square problem with cost(inc) = 1 and cost(sqr) = 3:

Solution 1 with length 3 ⟨inc, sqr, sqr⟩ (cost: 7) is not better than

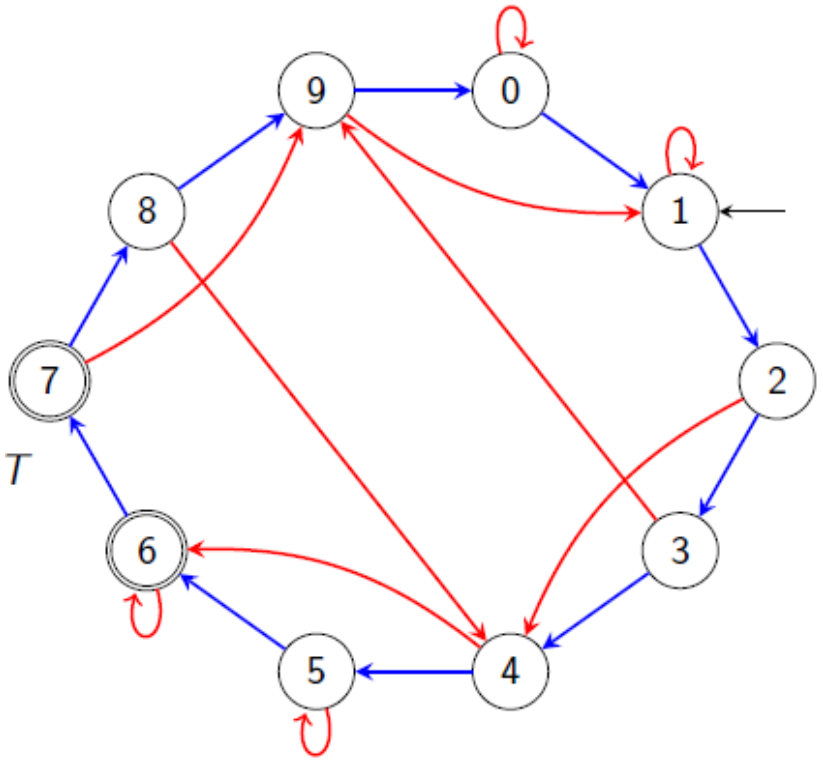
Solution 2 with length 5 ⟨inc, inc, inc, inc, inc⟩ (cost: 5) (is cheaper!)

Solution: uniform cost search always expand a node with minimal path cost (implementation: **priority queue** (min-heap) for open list)

Uniform Cost Search

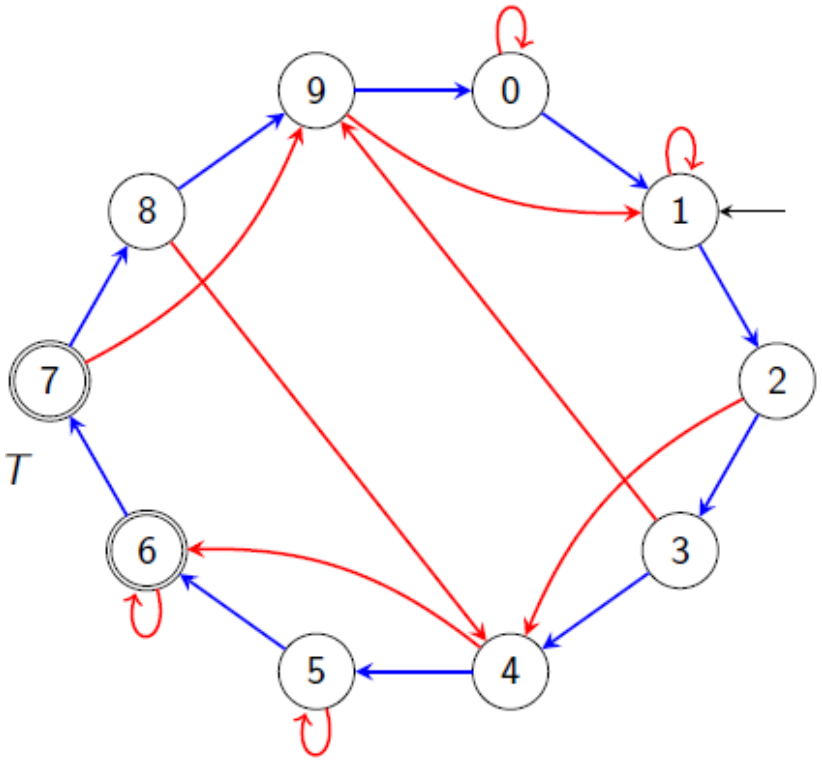
- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

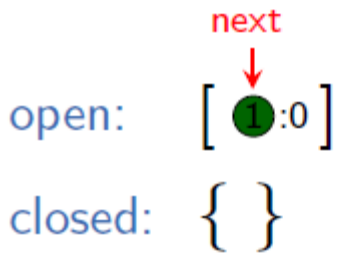
Uniform Cost Search on Bounded Inc-and-Sqr Problem



- Uniform cost search is optimal because at every state the path with the least cost is chosen.
- It is identical to Dijkstra's algorithm for shortest paths.
- Uniform cost search uses a priority queue.

Uniform Cost Search on Bounded Inc-and-Sqr Problem

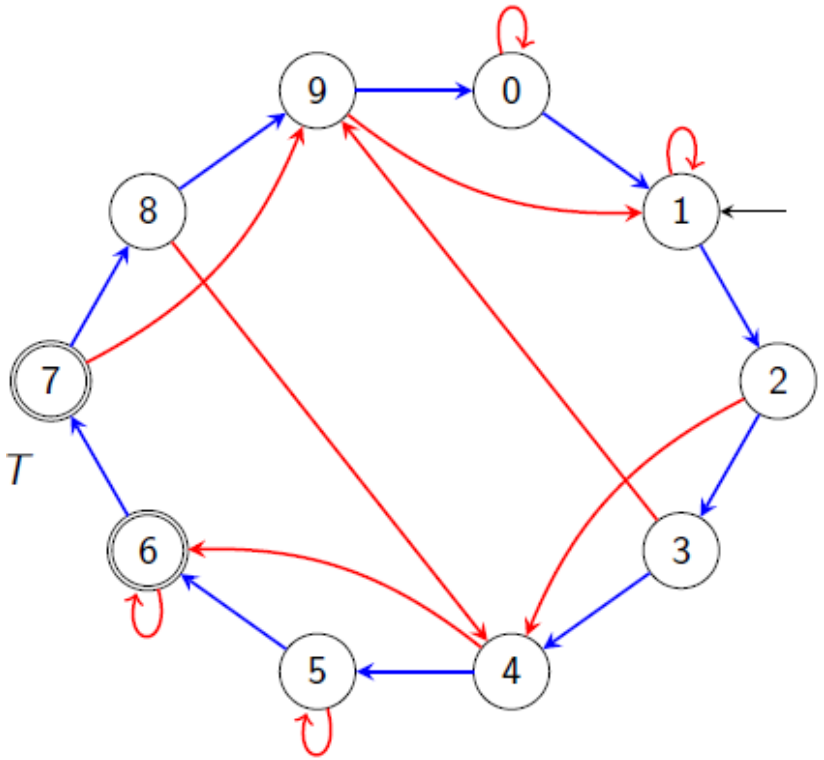


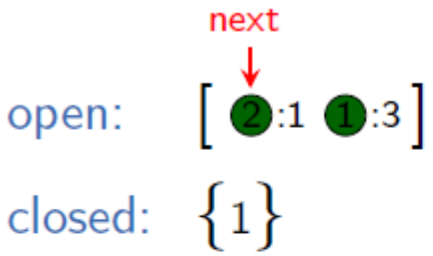


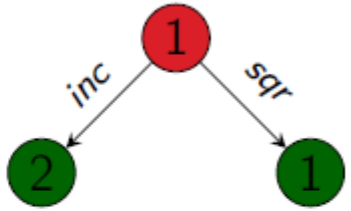
Cost (Inc) = 1

Cost (Sqr) = 3

Uniform Cost Search on Bounded Inc-and-Sqr Problem



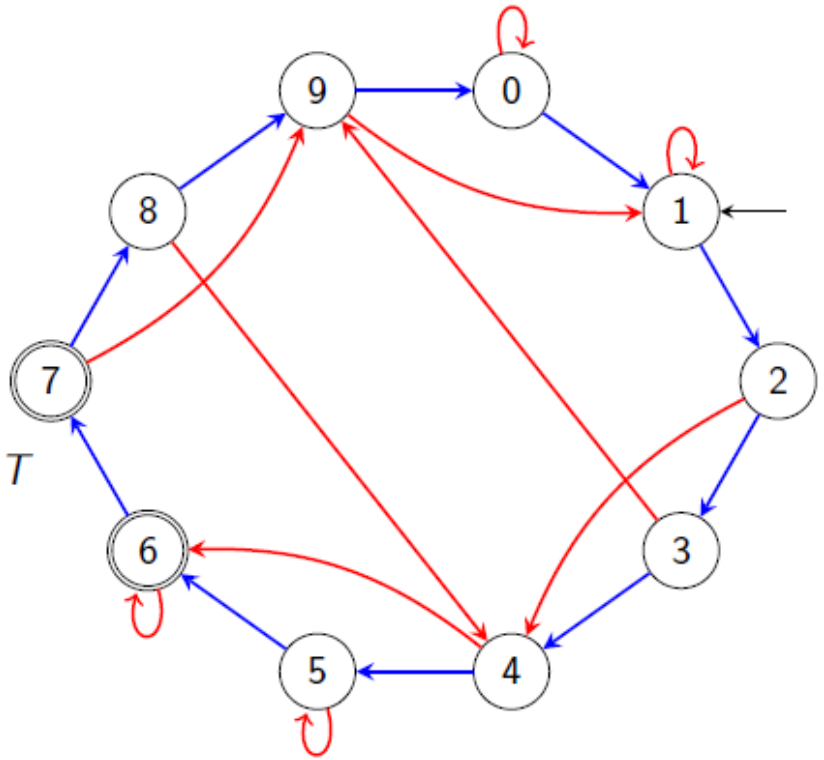


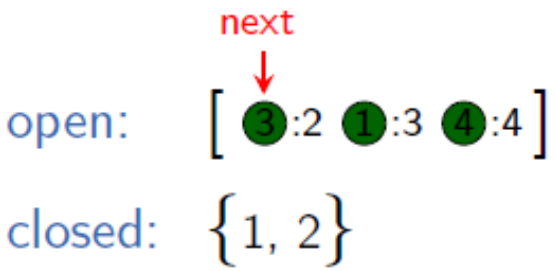


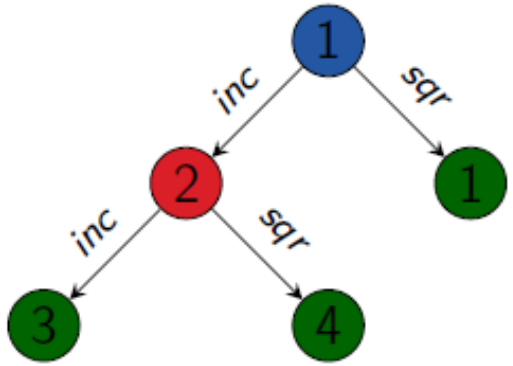
Cost (Inc) = 1

Cost (Sqr) = 3

Uniform Cost Search on Bounded Inc-and-Sqr Problem



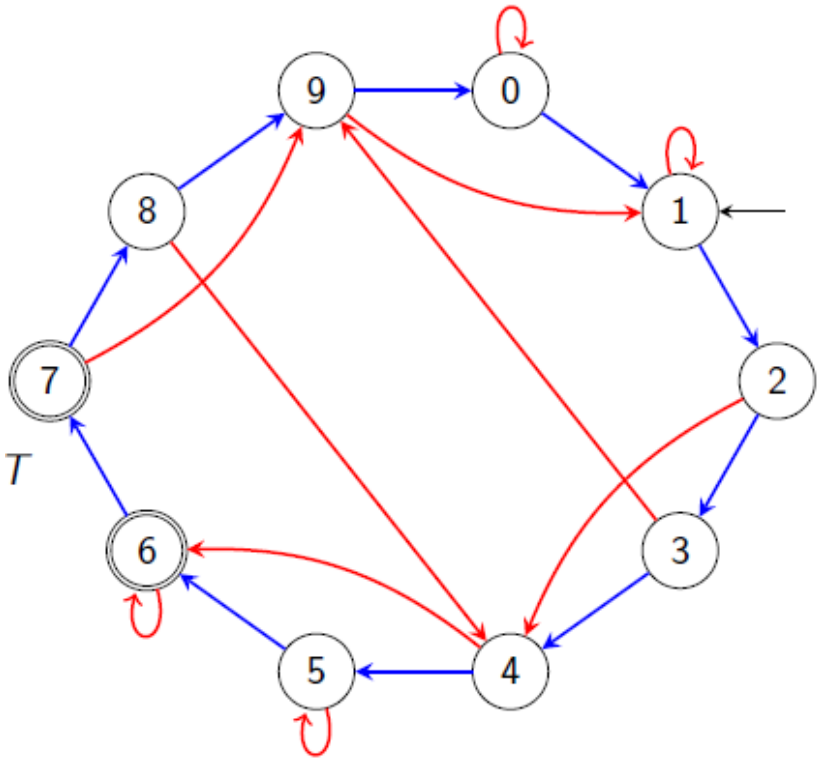


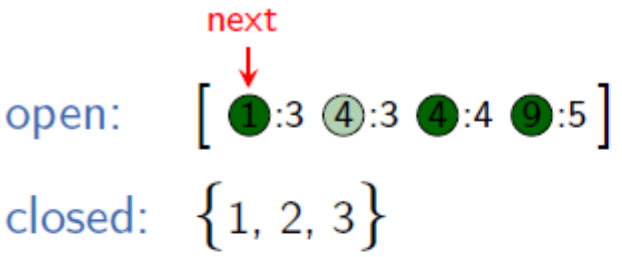


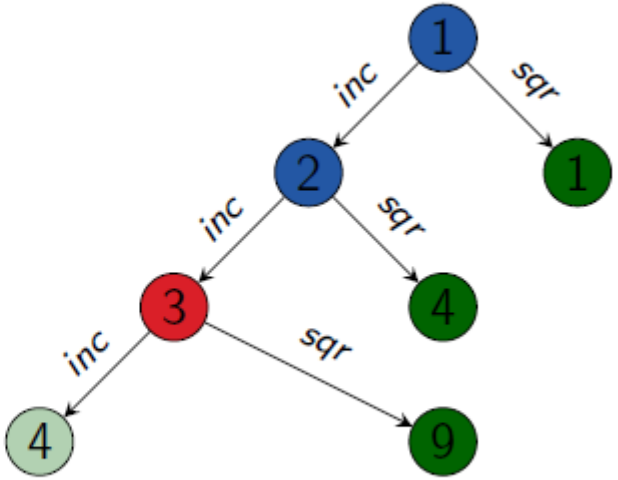
Cost (Inc) = 1

Cost (Sqr) = 3

Uniform Cost Search on Bounded Inc-and-Sqr Problem



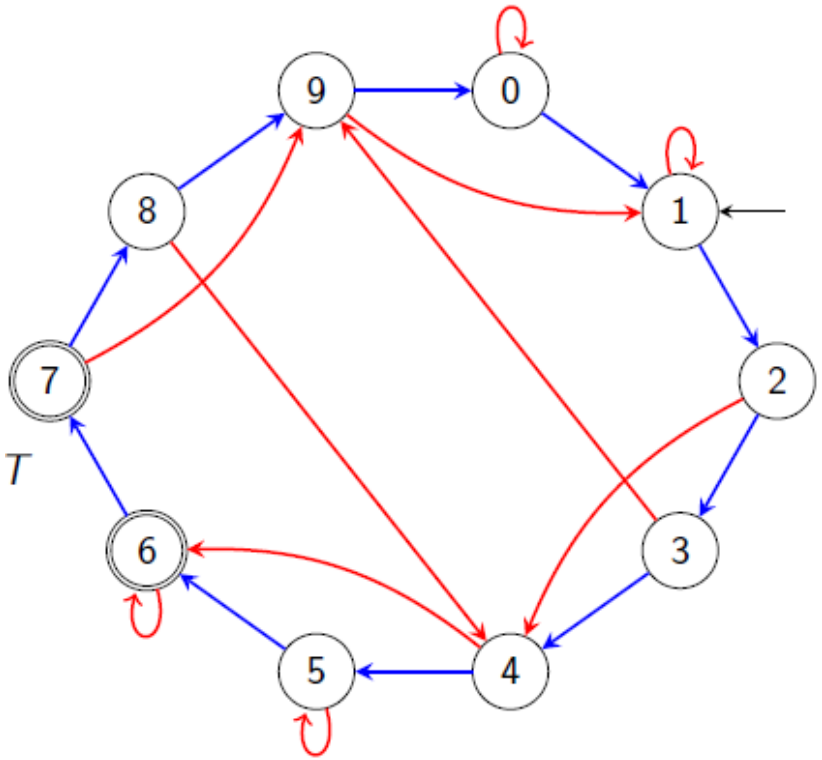




Cost (Inc) = 1

Cost (Sqr) = 3

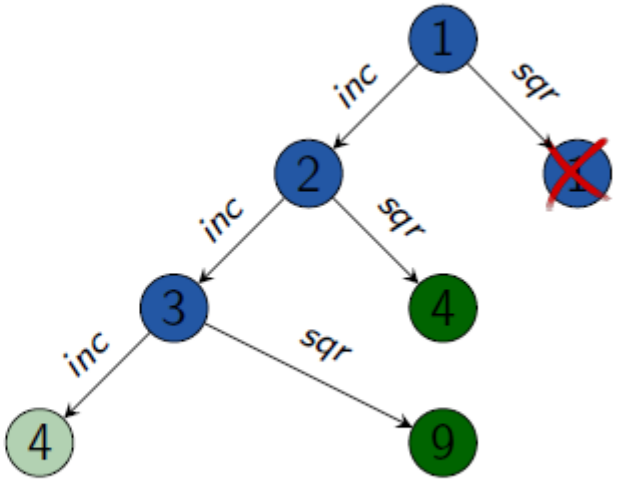
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [4:3 4:4 9:5]

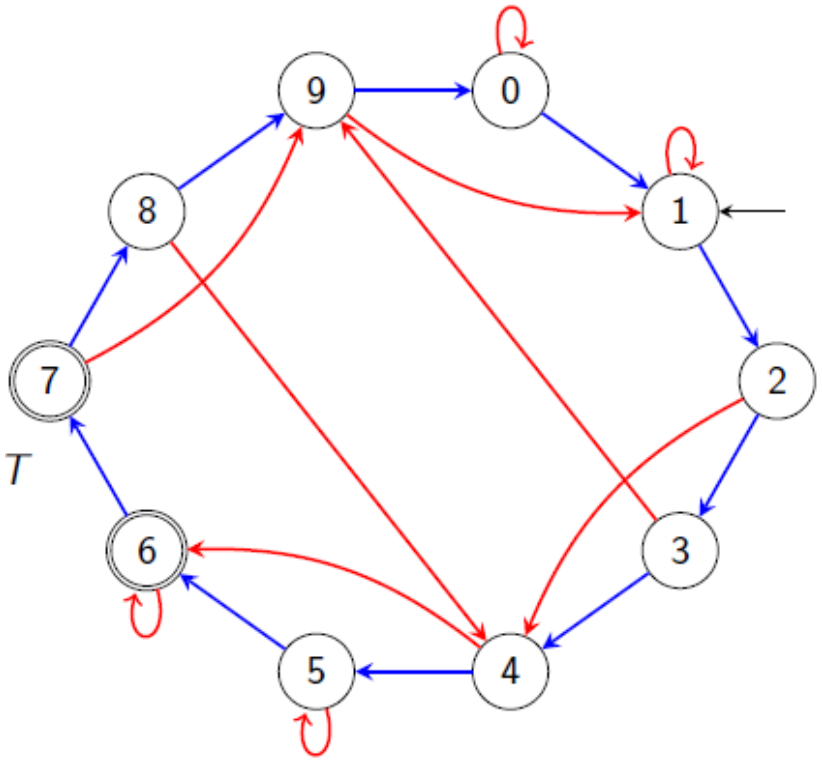
closed: {1, 2, 3}



Cost (Inc) = 1

Cost (Sqr) = 3

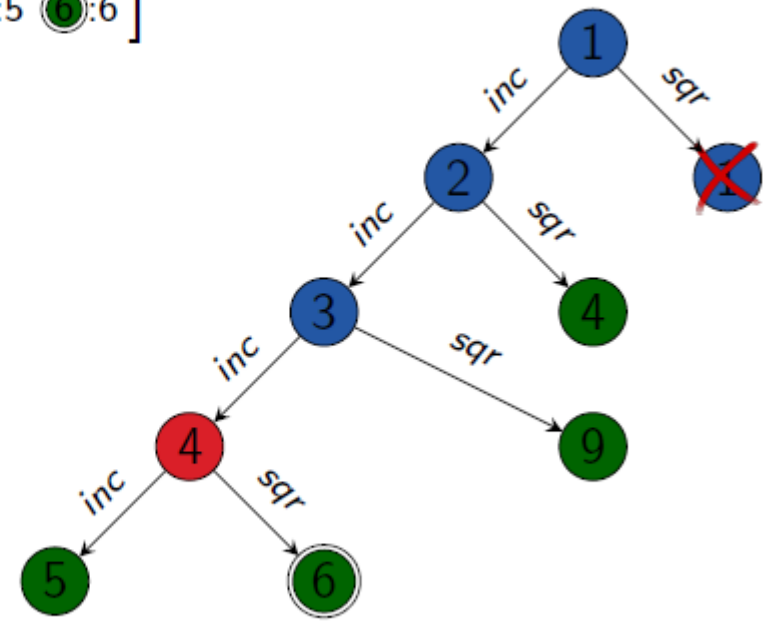
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [4:4 5:4 9:5 6:6]

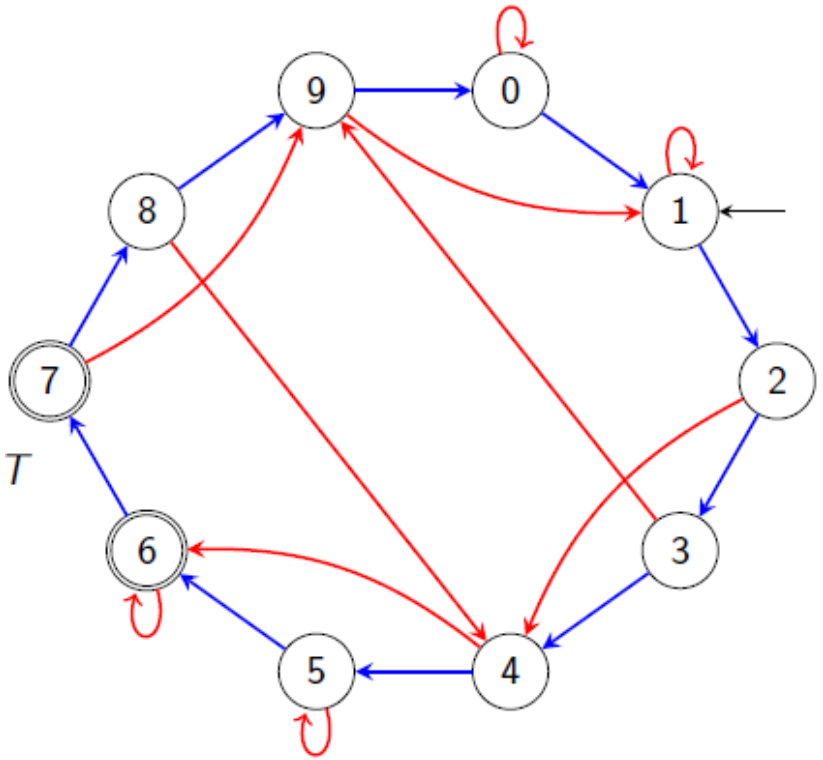
closed: {1, 2, 3, 4}



Cost (Inc) = 1

Cost (Sqr) = 3

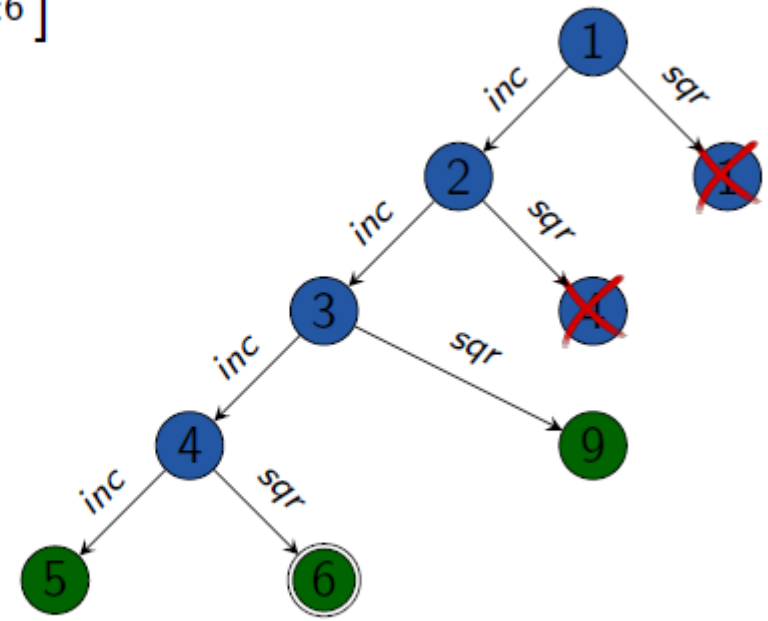
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [5 :4 9 :5 6 :6]

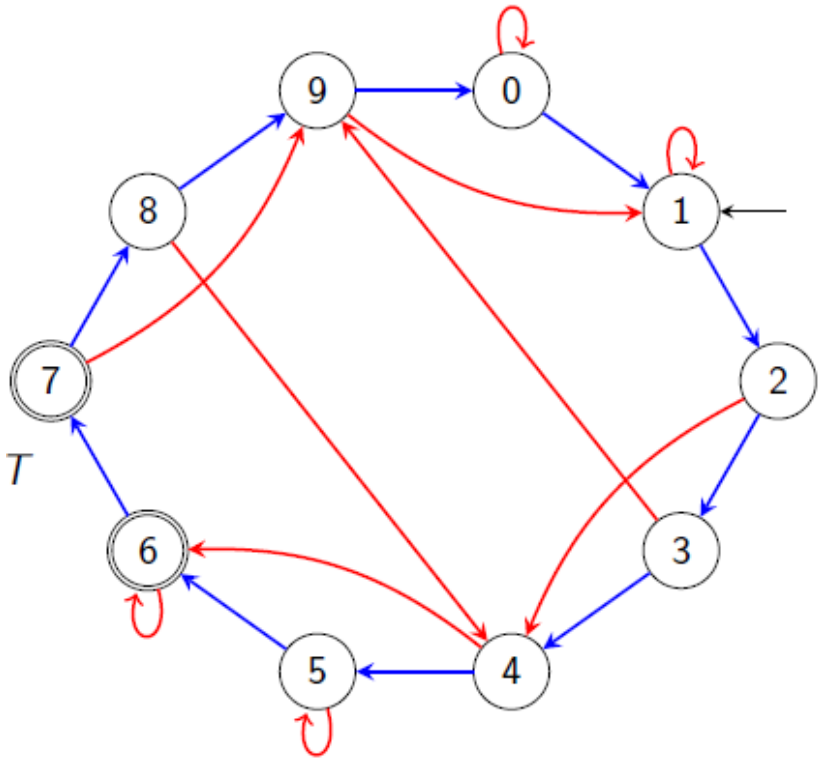
closed: {1, 2, 3, 4}



Cost (Inc) = 1

Cost (Sqr) = 3

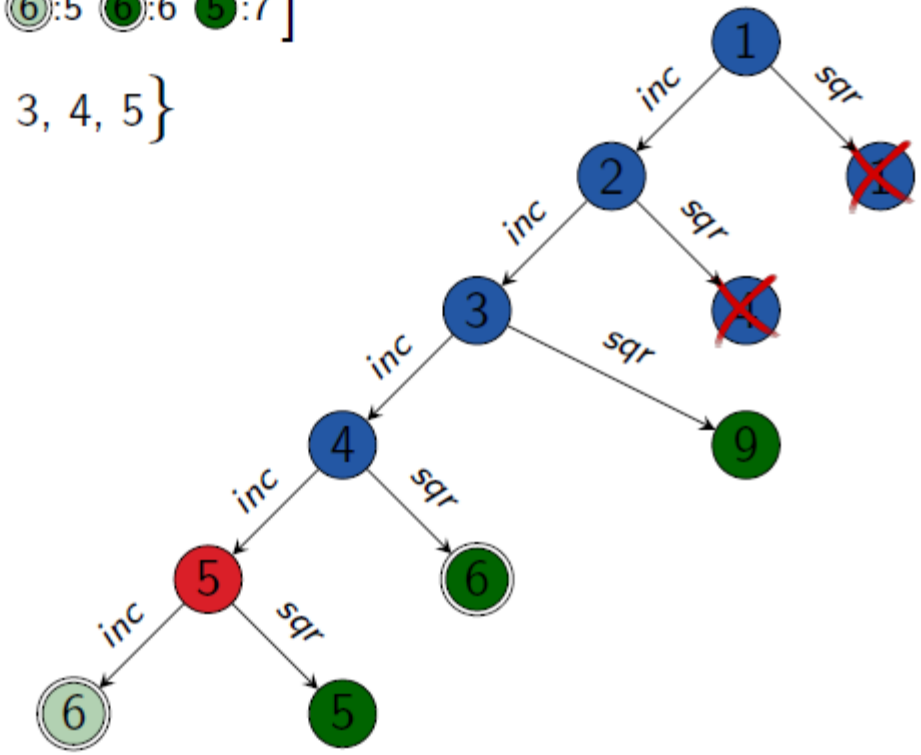
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [9:5 6:5 6:6 5:7]

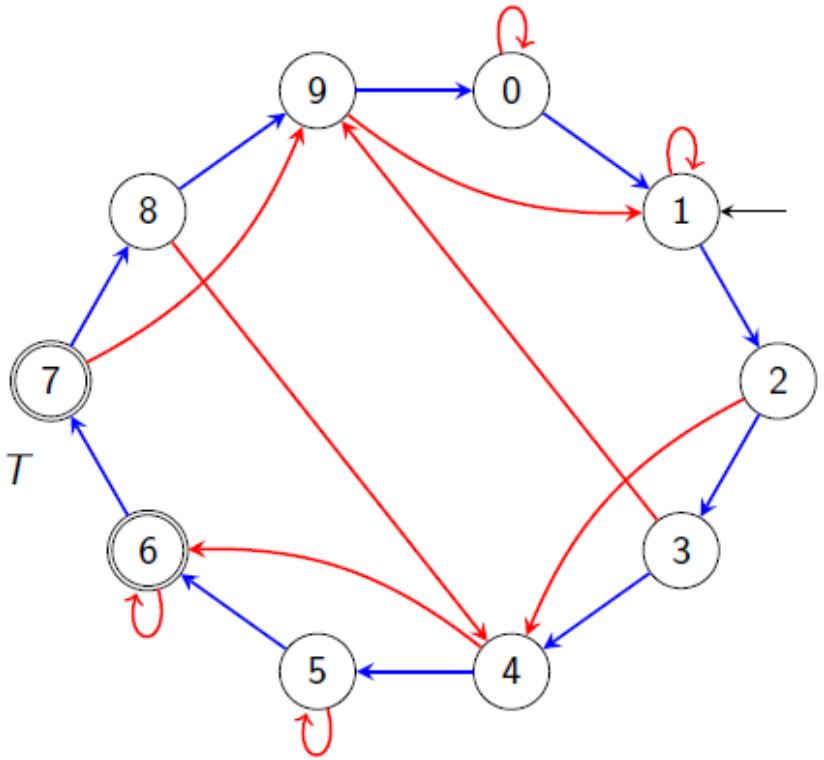
closed: {1, 2, 3, 4, 5}

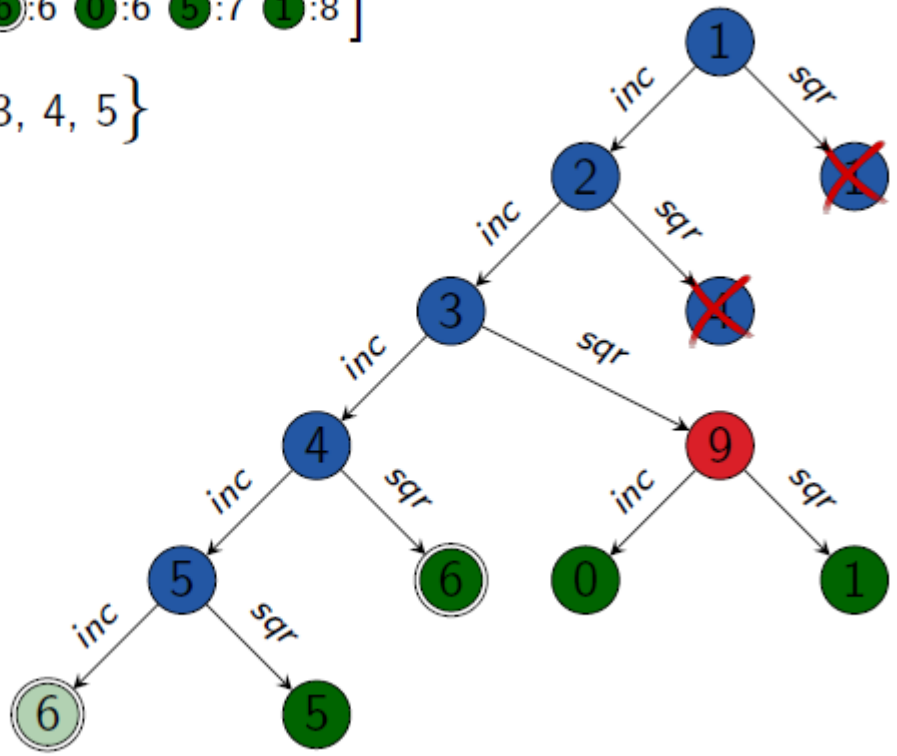


Cost (Inc) = 1

Cost (Sqr) = 3

Uniform Cost Search on Bounded Inc-and-Sqr Problem

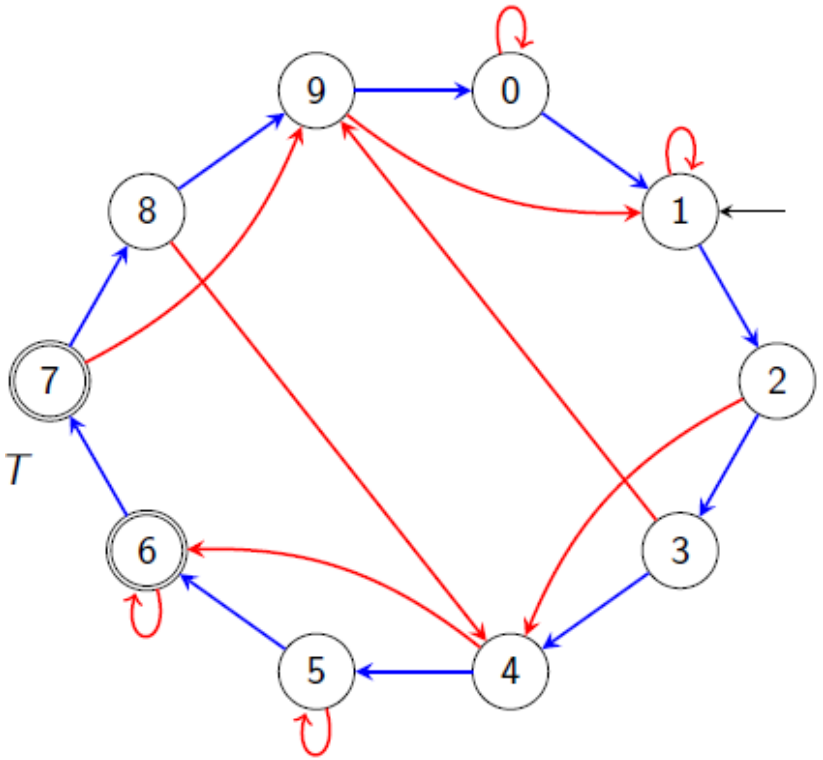




Cost (Inc) = 1

Cost (Sqr) = 3

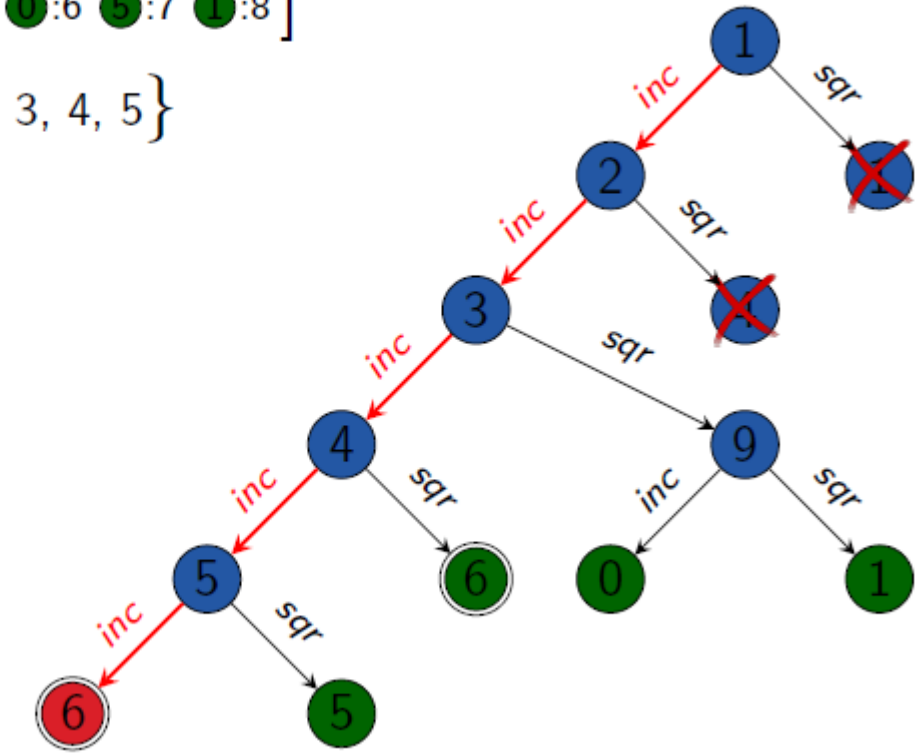
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [6:6 0:6 5:7 1:8]

closed: {1, 2, 3, 4, 5}



Cost (Inc) = 1

Cost (Sqr) = 3

Uniform Cost Search on **Bounded Inc-and-Sqr Problem**

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

- Additional early duplicate tests for generated nodes can reduce memory requirements can be beneficial or detrimental for runtime must be careful to keep shorter path to duplicate state
    - Uniform Cost Search (with elimination of duplicate state)

Uniform Cost Search on **Bounded Inc-and-Sqr Problem**

Properties of uniform cost search:

- Uniform cost search is complete (if there is a solution, Uniform cost search will find it).
- Uniform cost search is optimal.

Uniform Cost Search on Bounded Inc-and-Sqr Problem

Properties of uniform cost search:

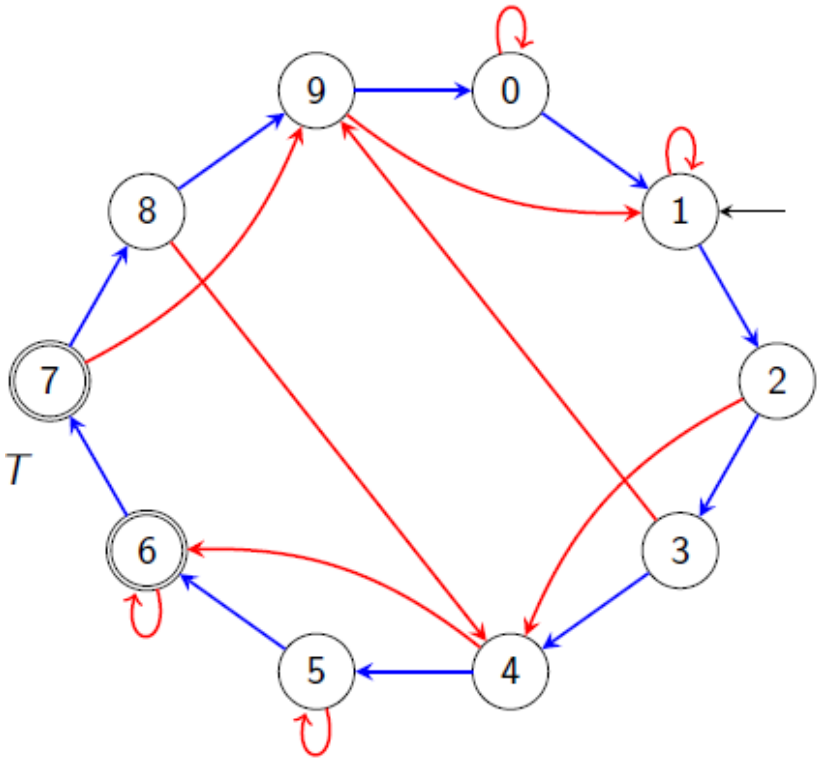
- Time complexity depends on distribution of action costs (Challenge: no simple and accurate bounds).

- Let **C*** is Cost of the optimal solution, and **ε** is each step to get closer to the goal node. Then the number of steps is = **[C*/ε]+1**. Here we have taken **+1**, as we start from state **0** and end to **C*/ε**.

  $\varepsilon := \min_{a \in A} cost(a)$ and consider the case $\varepsilon > 0$.

- Time complexity of Uniform-cost search is $O(\mathbf{b}^{1 + [C^*/\varepsilon]})$
- Space complexity = time complexity

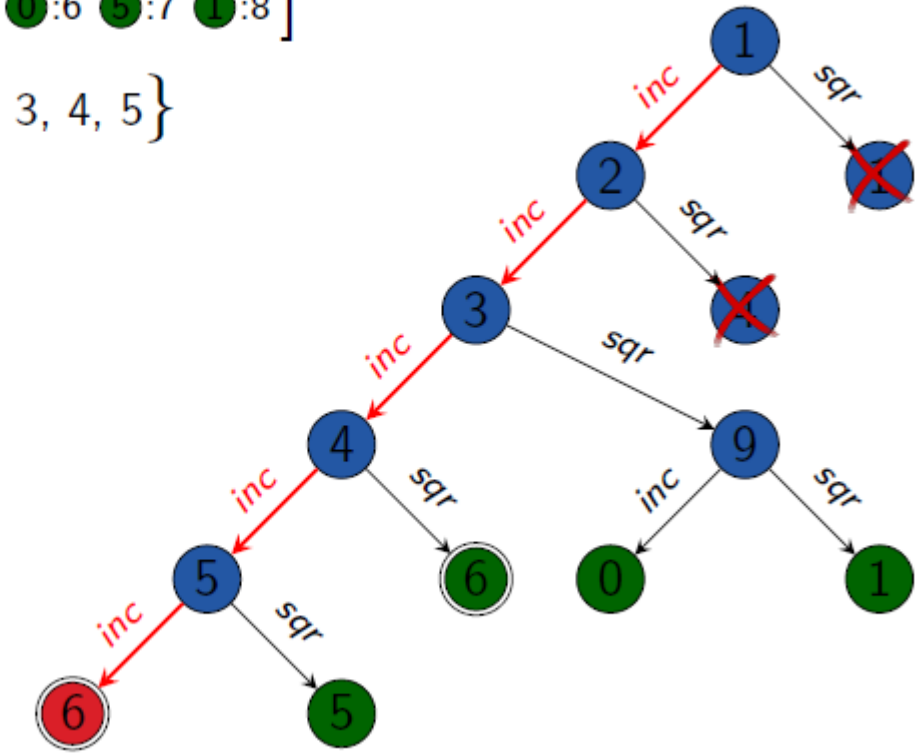
Uniform Cost Search on Bounded Inc-and-Sqr Problem



next

open: [6:6 0:6 5:7 1:8]

closed: {1, 2, 3, 4, 5}



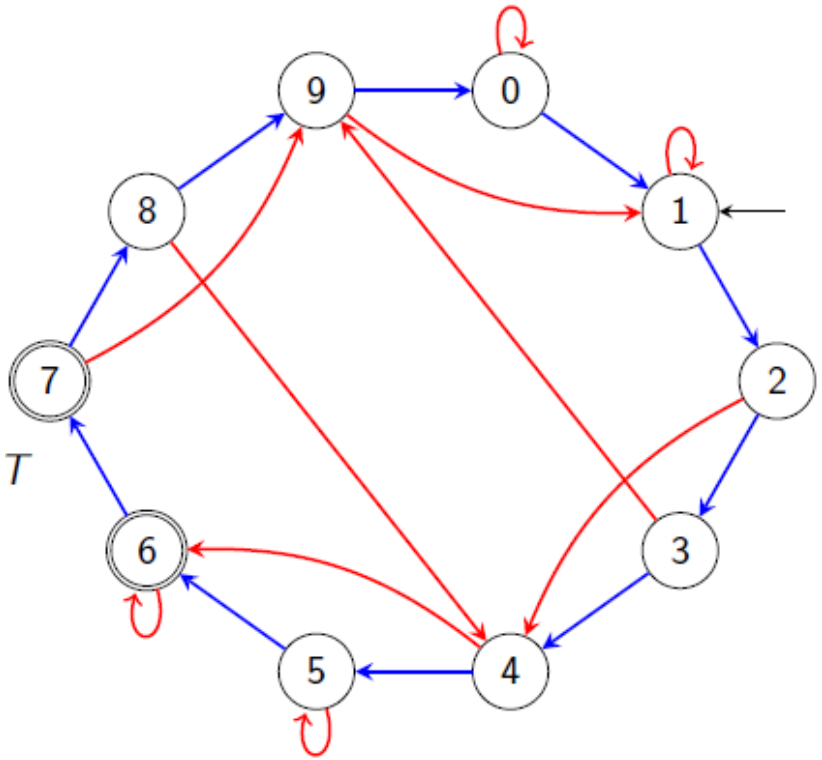
Cost (Inc) = 1

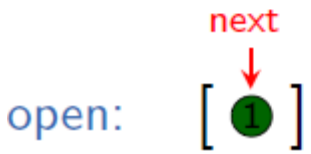
Cost (Sqr) = 3

Depth First Search

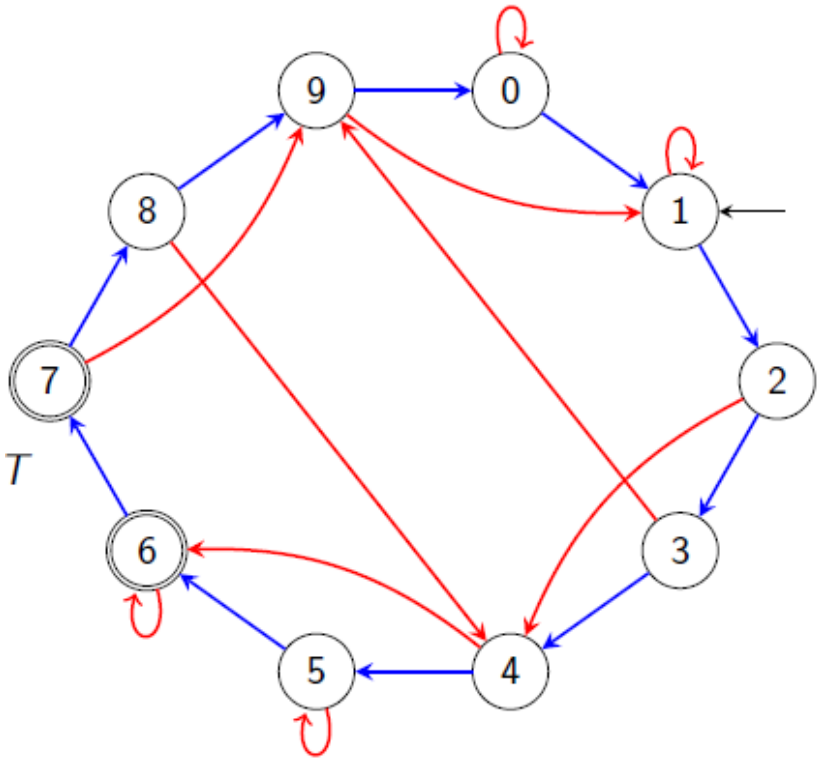
- Depth first search expands nodes in opposite order of generation (LIFO)
- Here open list implements a stack
- Here deepest node is expanded first

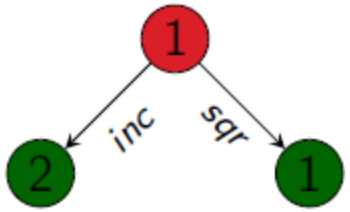
Depth First Search on Bounded Inc-and-Sqr Problem

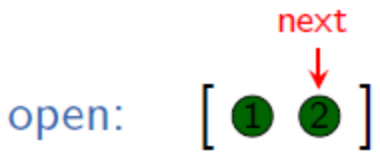




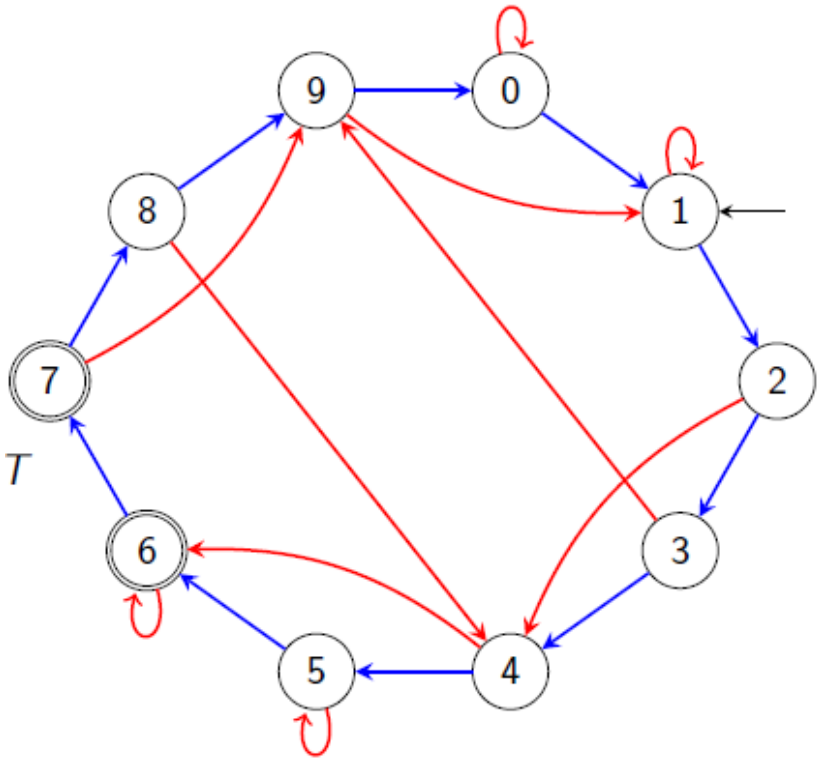
Depth First Search on Bounded Inc-and-Sqr Problem

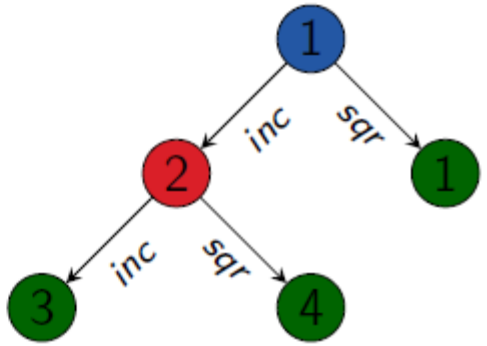


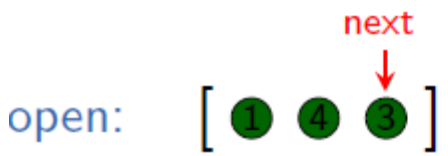




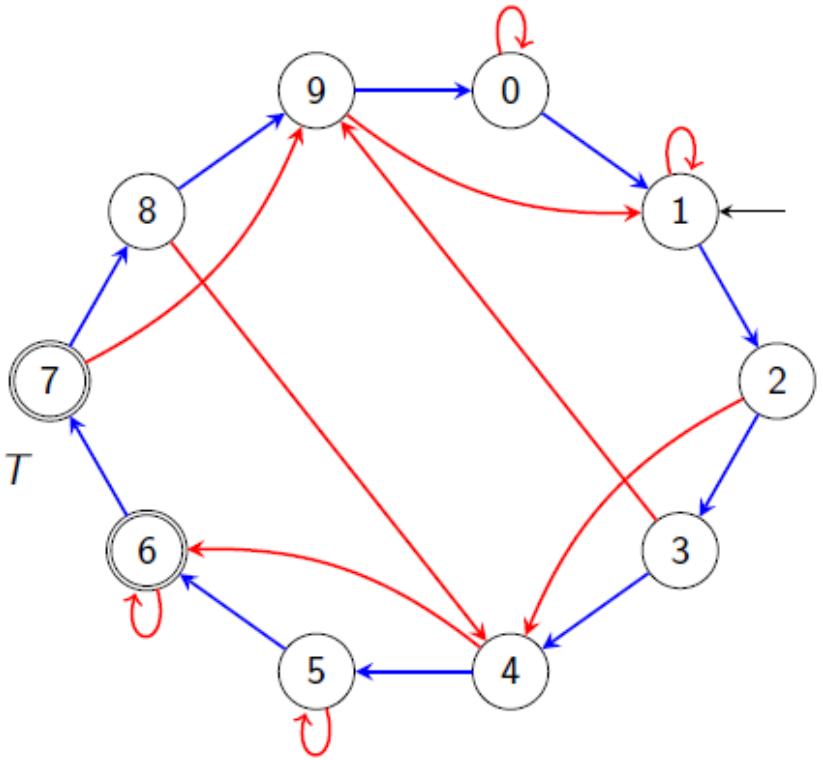
Depth First Search on Bounded Inc-and-Sqr Problem

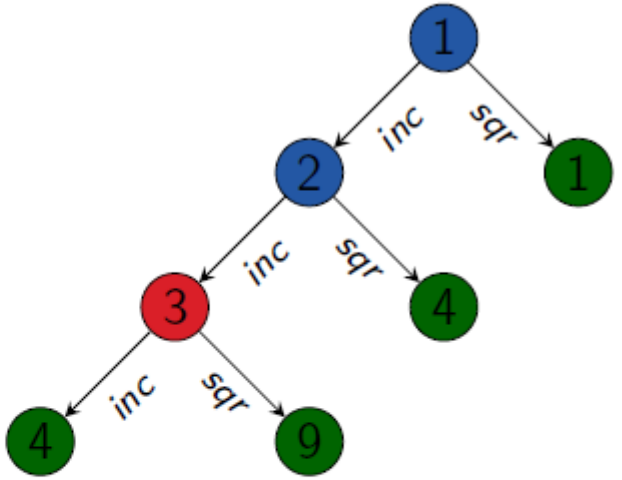


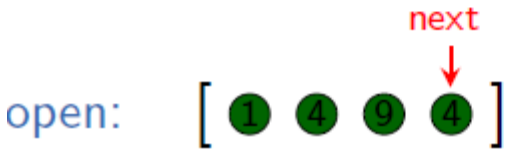




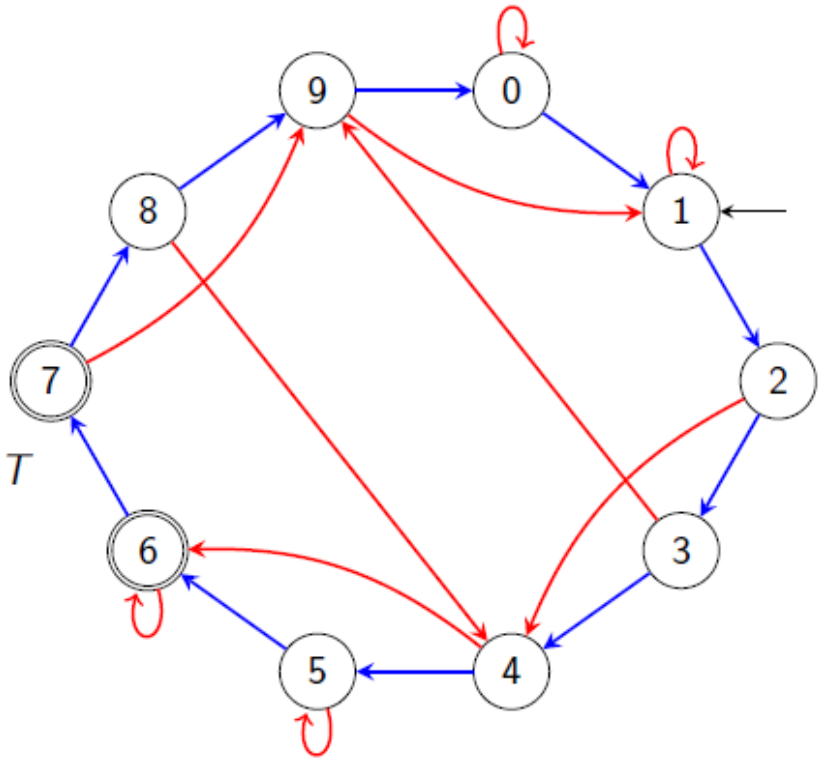
Depth First Search on Bounded Inc-and-Sqr Problem

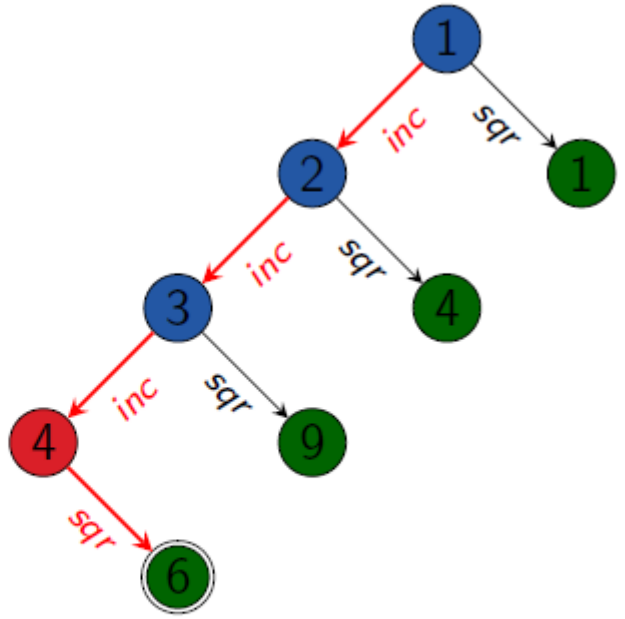


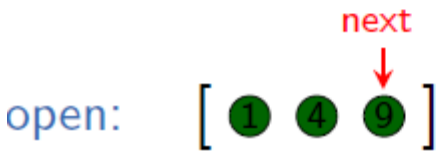




Depth First Search on Bounded Inc-and-Sqr Problem







Depth First Search

Properties of Depth First Search:

- Depth First Search is complete and not optimal
- Complete for acyclic state spaces
- However, depth-first search is simpler than other search
- Time complexity:
  - If the state space has branching factor **b** and it includes max paths of length **l**, depth-first search can generate $O(b^l)$ nodes, even if much shorter solutions (e.g., of length 1) exist.
  - On the other hand: in the best case, the solutions of length **l** can be found with O(**bl**) generated nodes.

Depth First Search

Properties of Depth First Search:

- Time complexity:
  - If the state space has branching factor **b** and it includes max paths of length **l** , depth-first search can generate $O(b^l)$ nodes, even if much shorter solutions (e.g., of length 1) exist.
  - On the other hand: in the best case, the solutions of length **l** can be found with O(**bl**) generated nodes.
- Space complexity:
  - Only need to store nodes along currently explored path ("along": nodes on path and their children)
  - Space complexity O(**bl**) if m maximal search depth reached low memory complexity main reason why depth-first search interesting despite its disadvantages

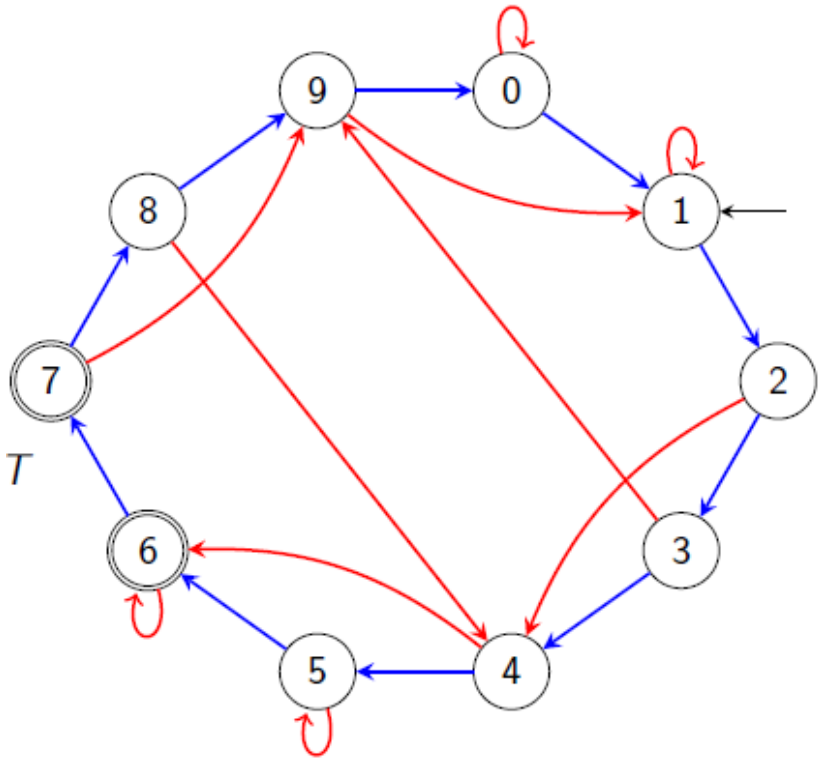
BFS VS UCS VS DFS

	BFS	UCS	DFS
Optimal	Yes	Yes	No
Completeness	Yes	Yes	No
Time Complexity	$O\left(b^{d}\right)$	$O\left(b^{(C^{*}/\varepsilon)+1}\right)$	$O\left(b^{l}\right)$
Space Complexity	$O\left(b^{d}\right)$	$O\left(b^{(C^{*}/\varepsilon)+1}\right)$	$O\left(bl\right)$

Depth limited Search (Depth Bounded Search)

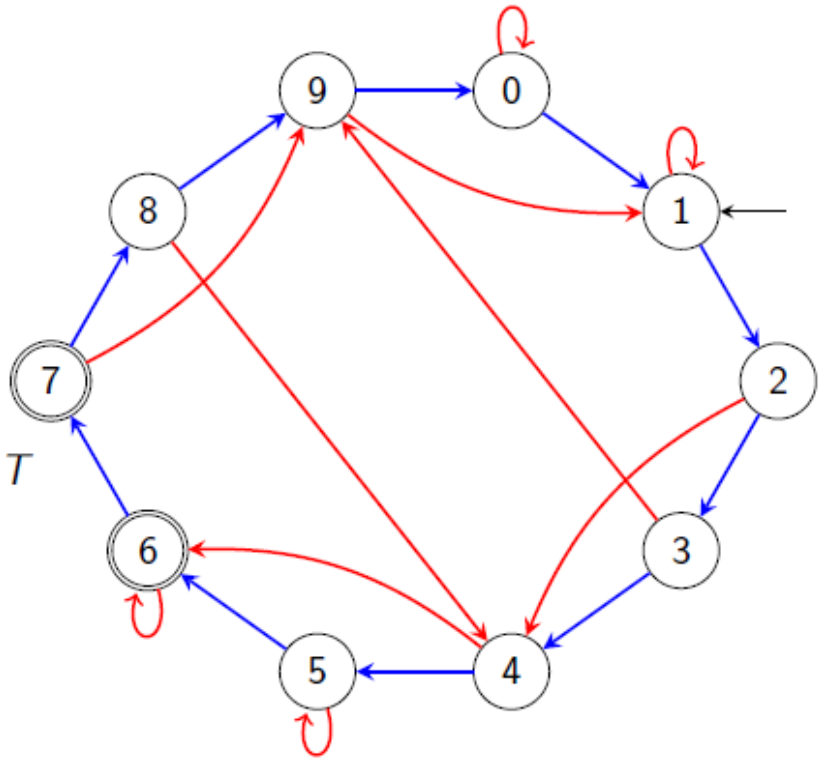
- It is a variant of DFS. It expands nodes in opposite order of generation (LIFO)
- It is parametrized with depth limit ℓ
- It prunes (does not expand) search nodes at depth $d \geq \ell$

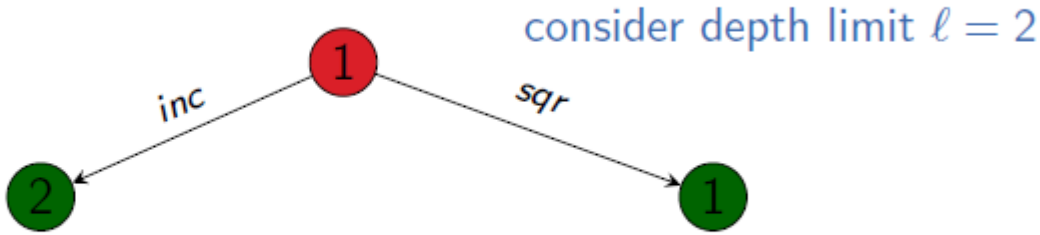
Depth limited Search on Bounded Inc-and-Sqr Problem



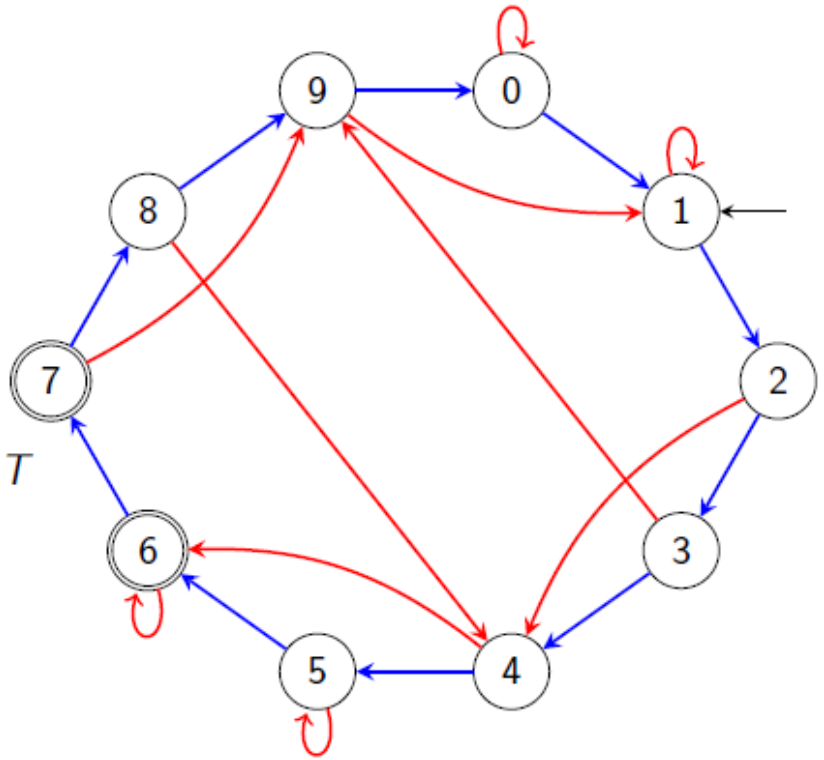
consider depth limit $\ell = 2$

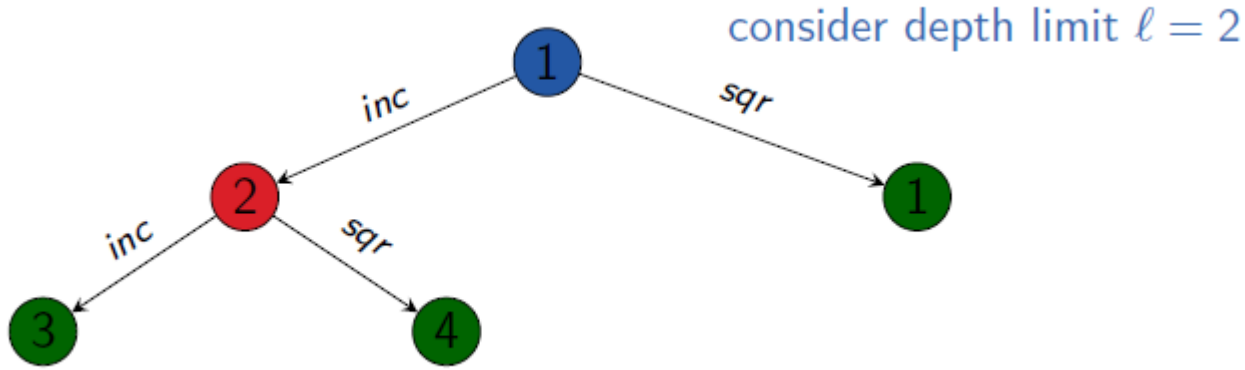
Depth limited Search on Bounded Inc-and-Sqr Problem



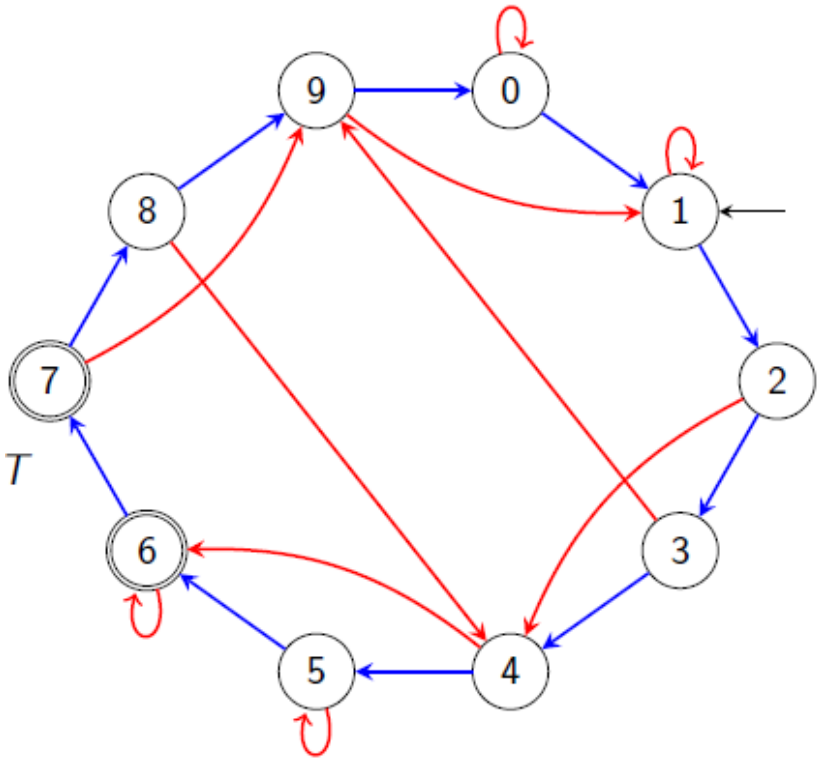


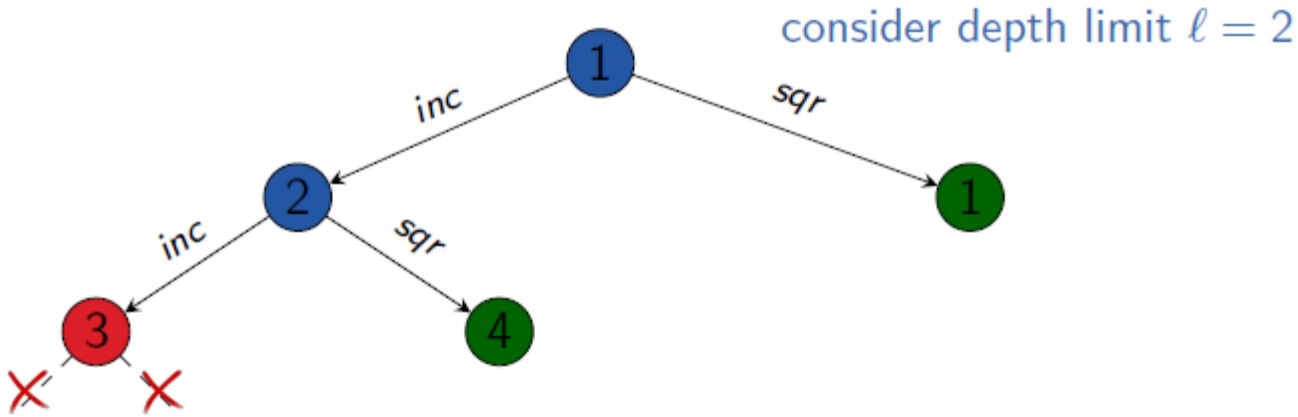
Depth limited Search on Bounded Inc-and-Sqr Problem



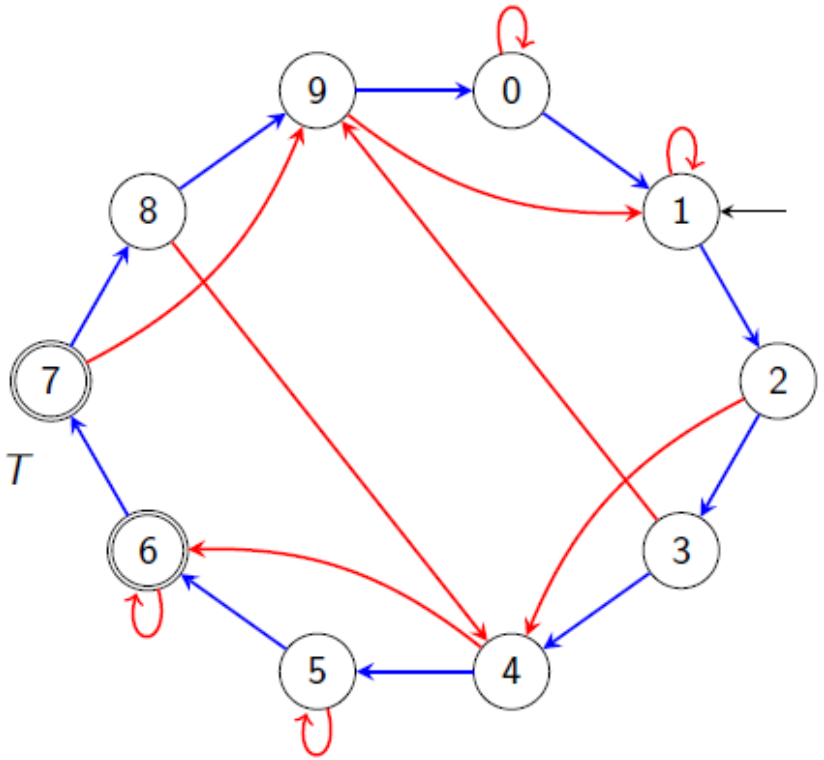


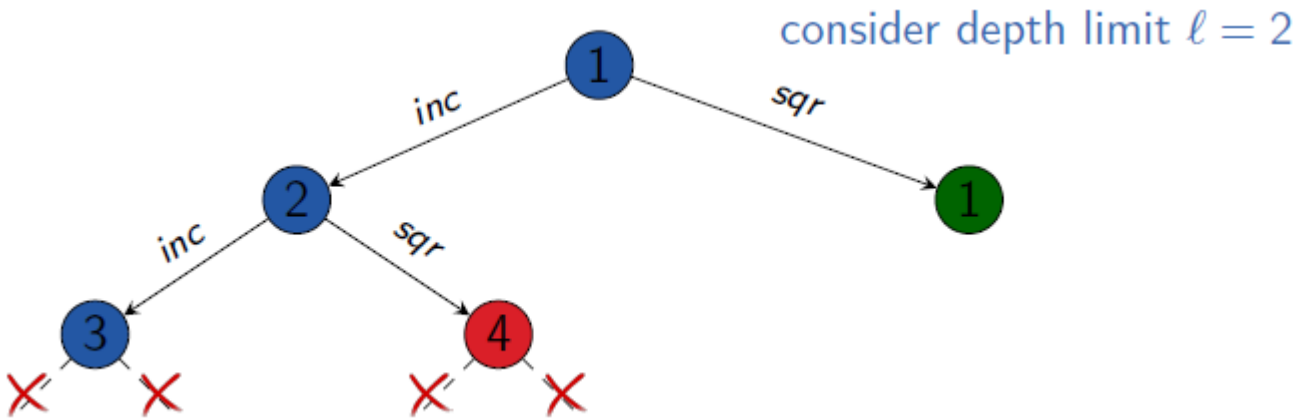
Depth limited Search on Bounded Inc-and-Sqr Problem



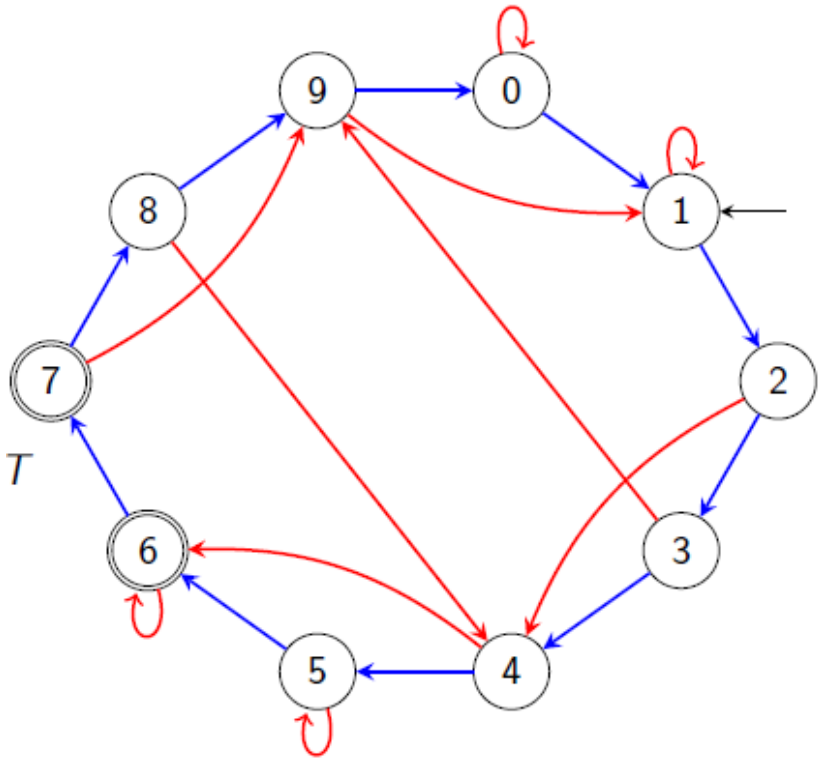


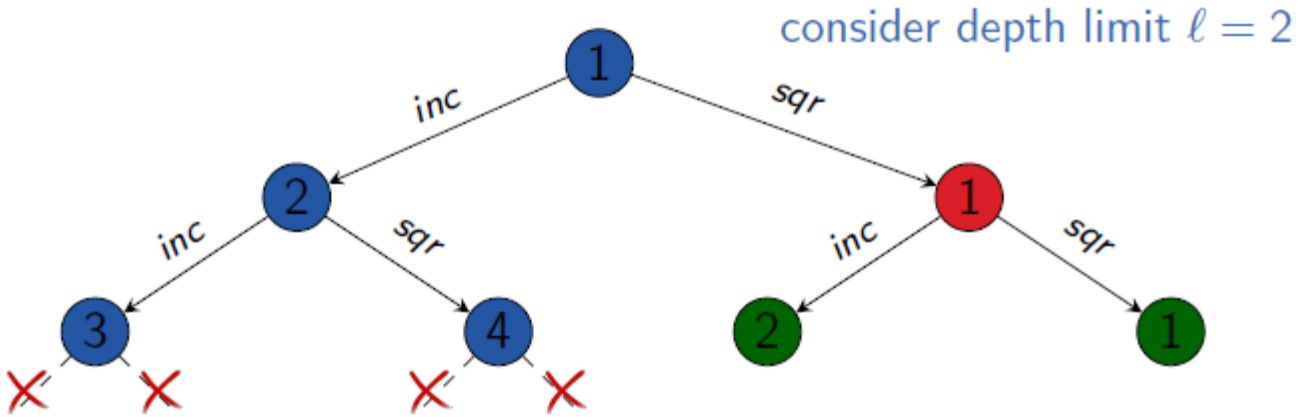
Depth limited Search on Bounded Inc-and-Sqr Problem



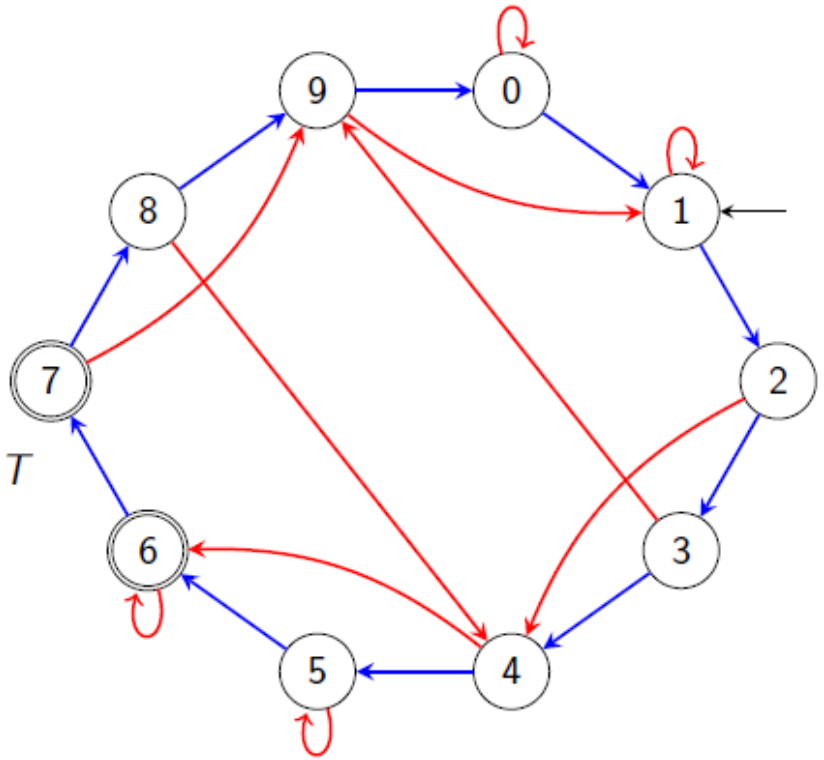


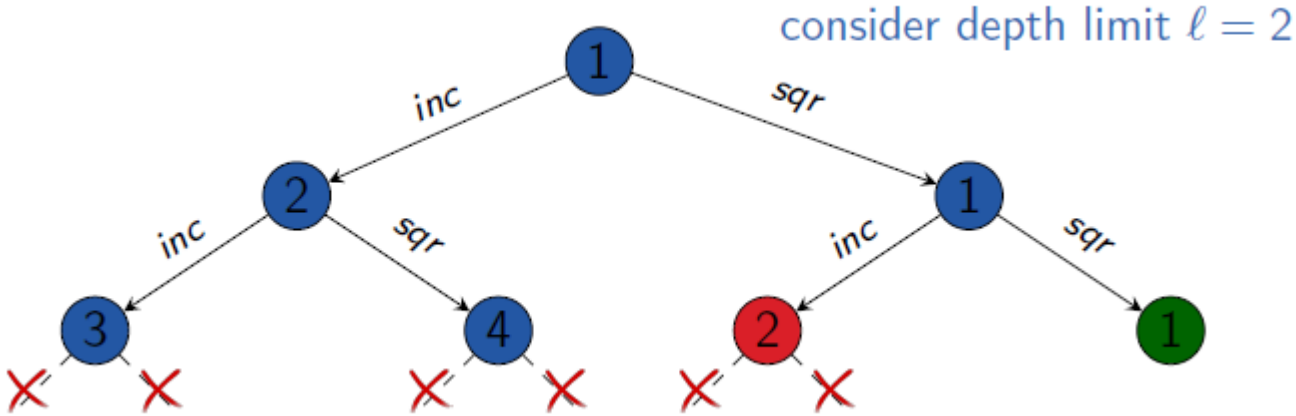
Depth limited Search on Bounded Inc-and-Sqr Problem



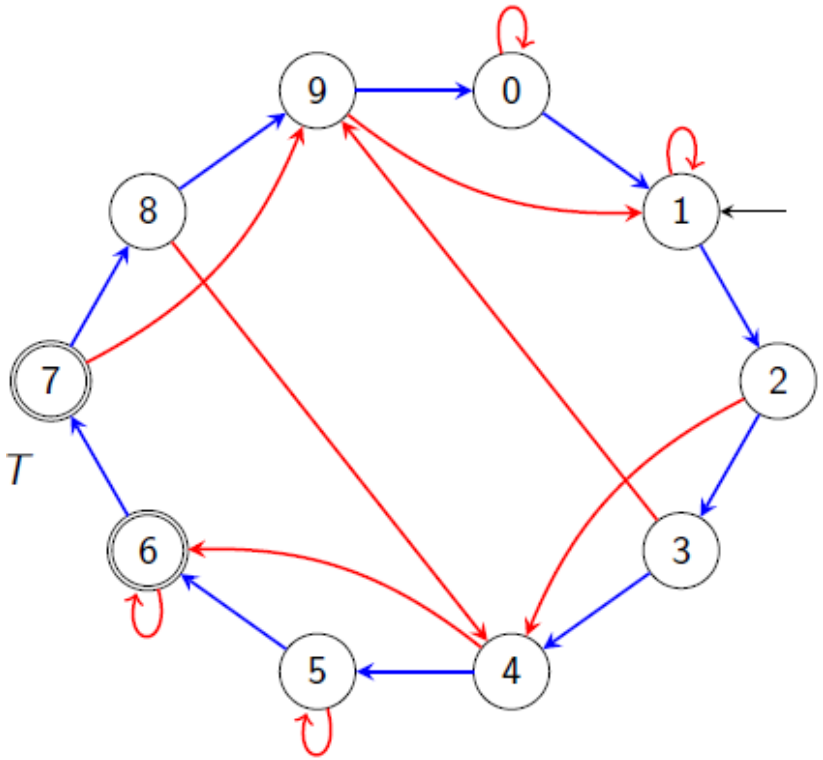


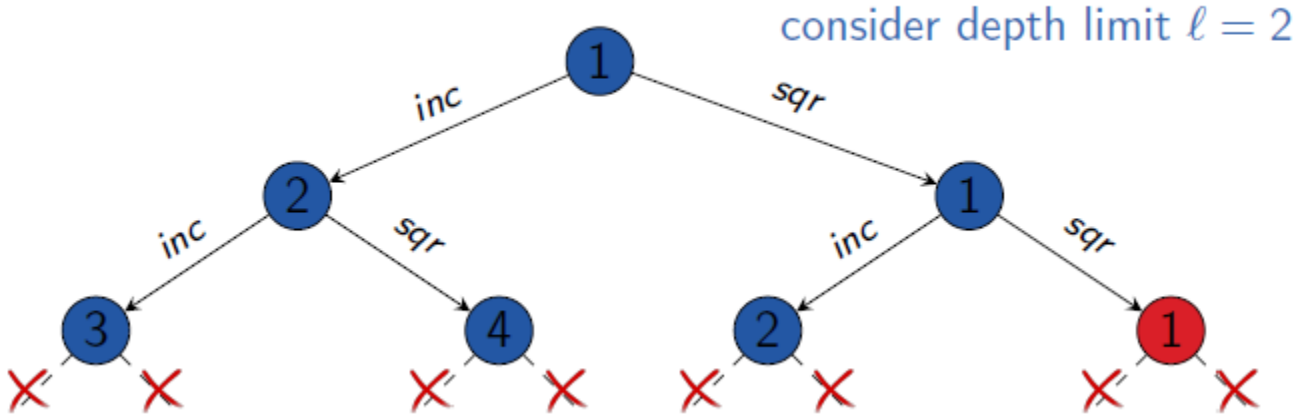
Depth limited Search on Bounded Inc-and-Sqr Problem





Depth limited Search on Bounded Inc-and-Sqr Problem





Depth Limited Search (Depth Bounded Search)

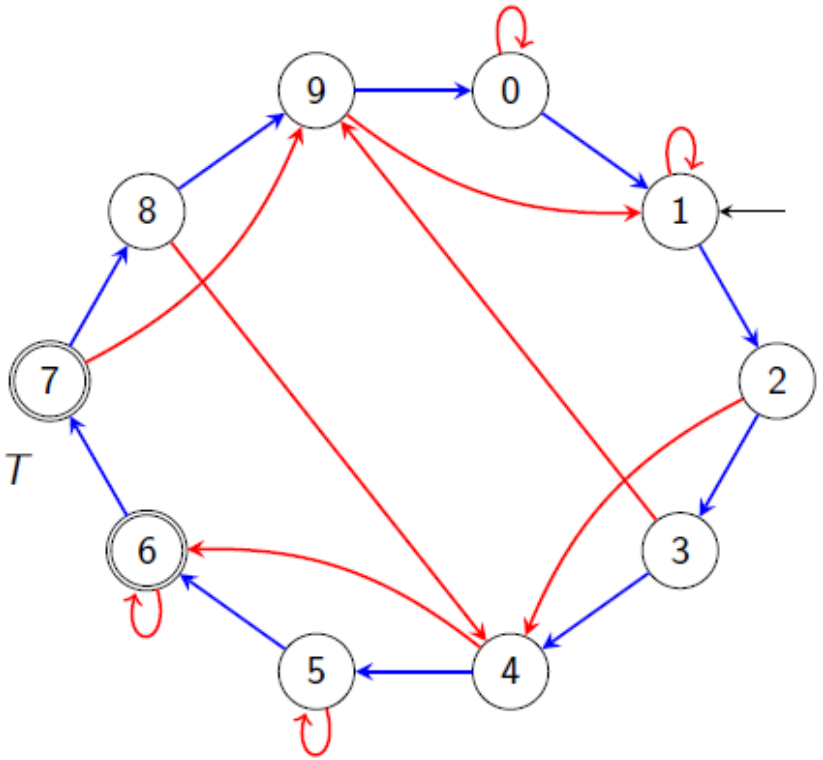
Properties of Depth Limited Search:

- It is not complete
- It is not optimal
- Time complexity: If the state space has branching factor **b** and it has length limit **l**, then time complexity is $O\left(b^l\right)$
- Space complexity: O(bl) (only need to store nodes along one path)

Iterative Deepening Depth-first Search

- It performs a sequence of depth-limited searches
- It increases depth limit ℓ in each iteration
- It sounds wasteful (each iteration repeats all the useful work of all previous iterations)
- In fact overhead acceptable (Why?)

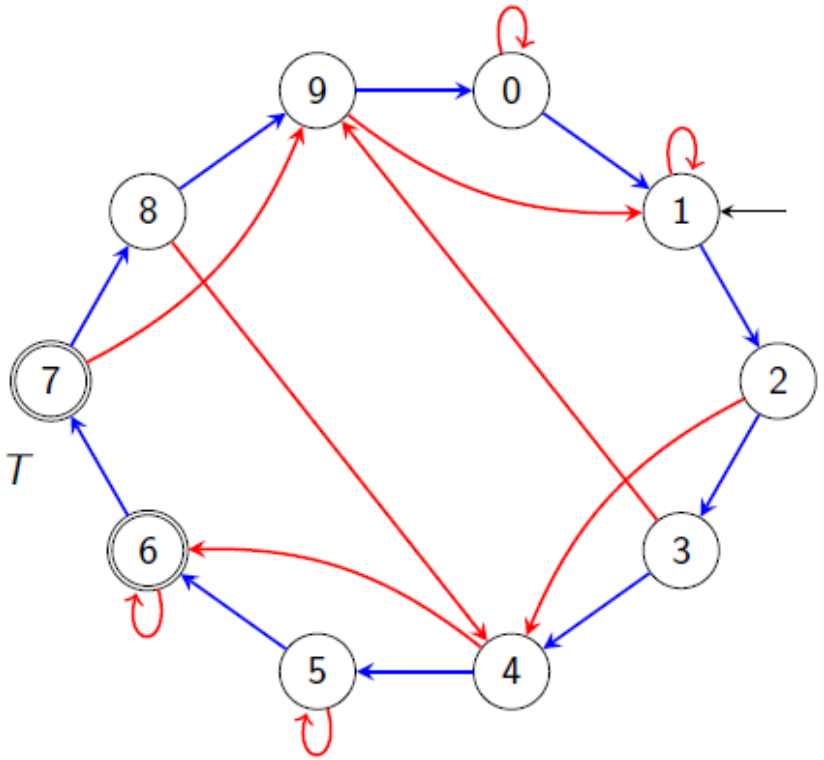
Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem



depth limit: 0

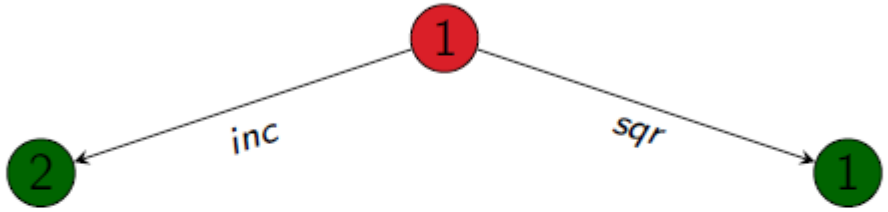
generated nodes: 1

Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

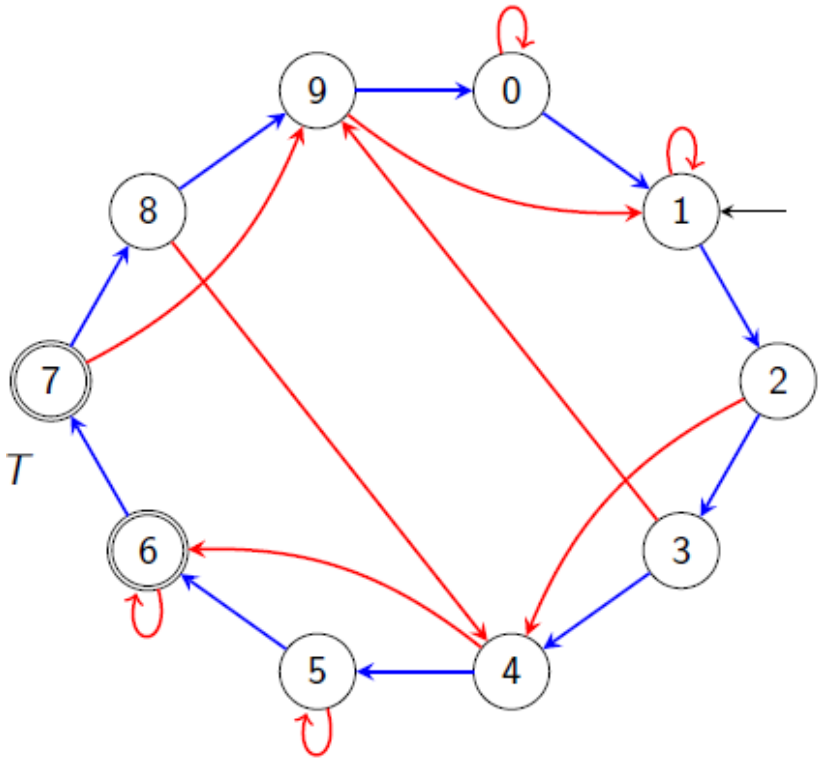


depth limit: 1

generated nodes: 1+3

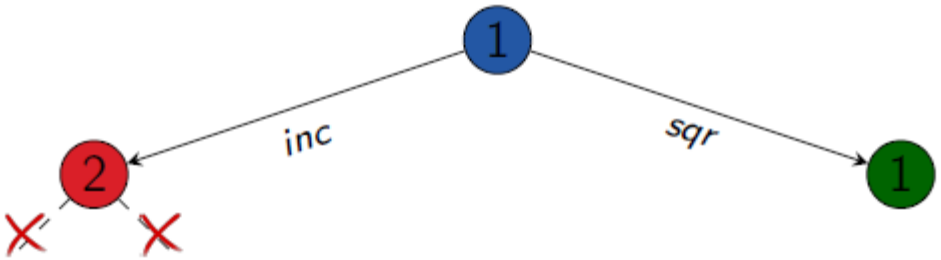


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

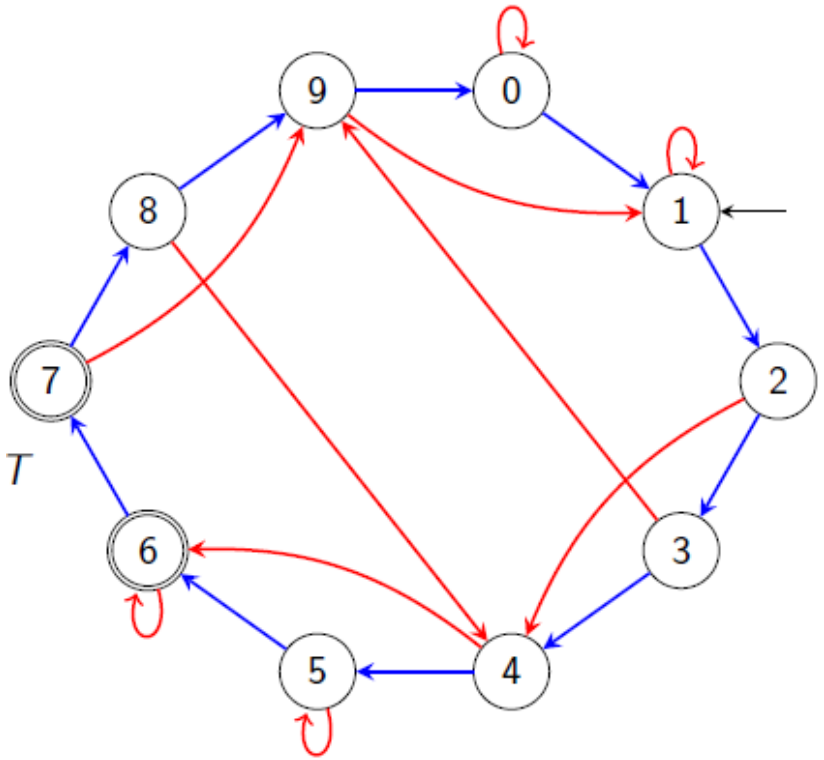


depth limit: 1

generated nodes: 1+3

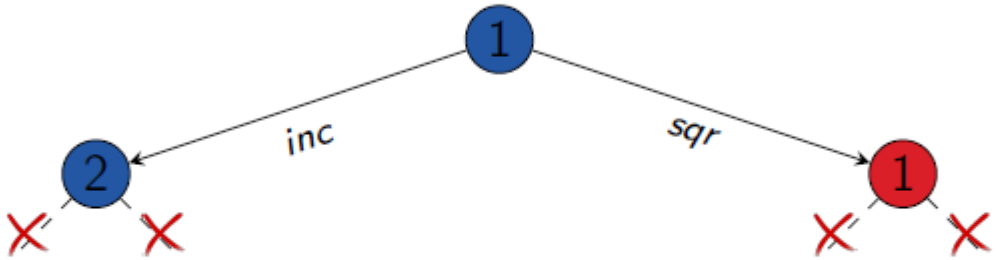


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

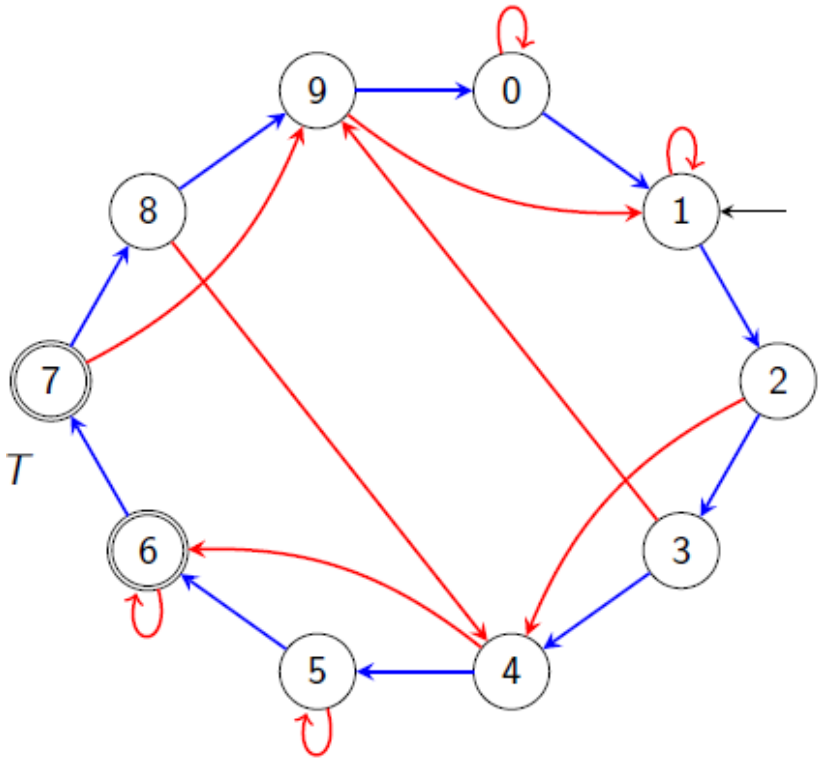


depth limit: 1

generated nodes: 1+3

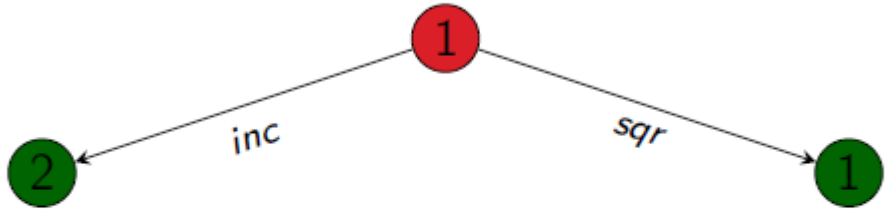


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

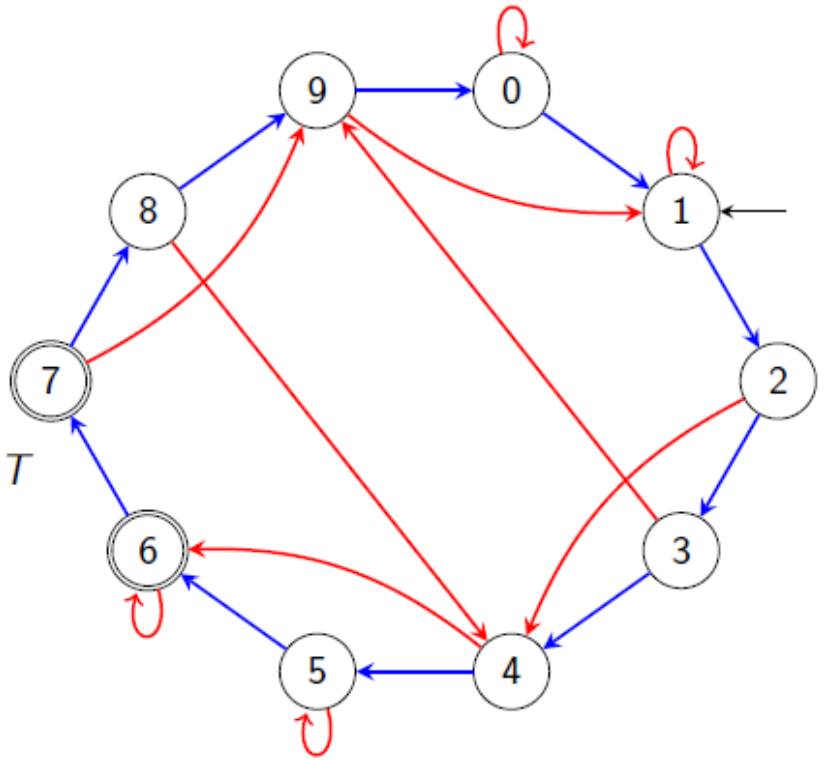


depth limit: 2

generated nodes: 1+3+3

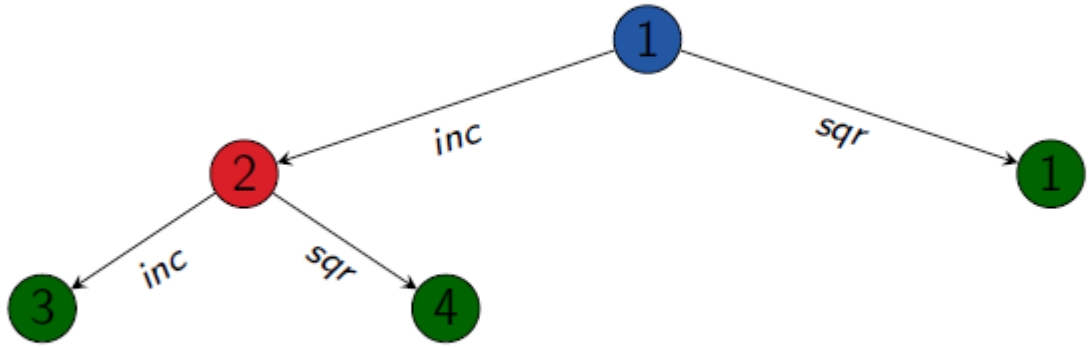


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

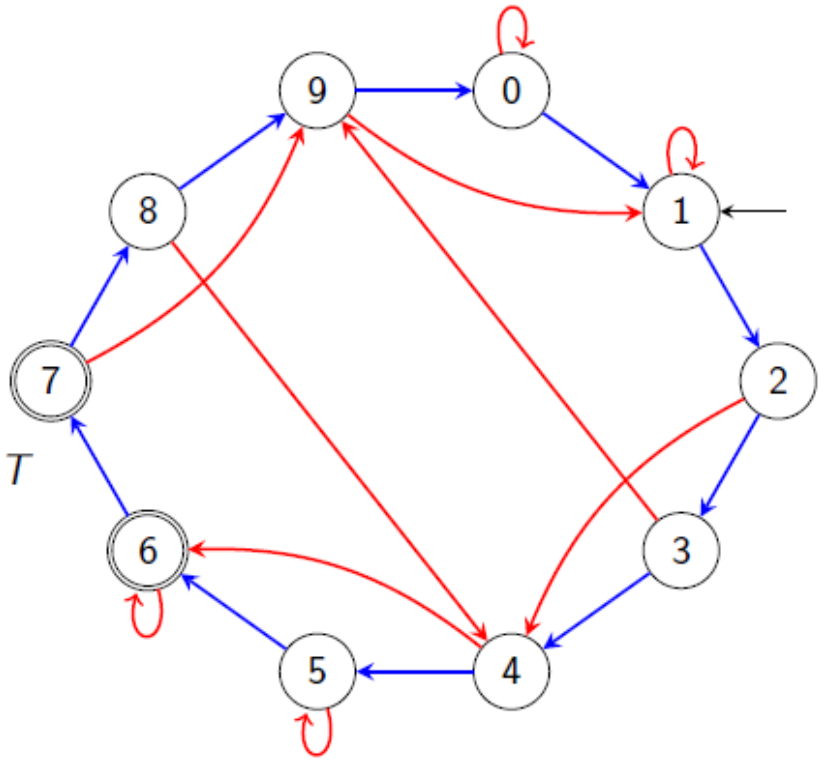


depth limit: 2

generated nodes: 1+3+5

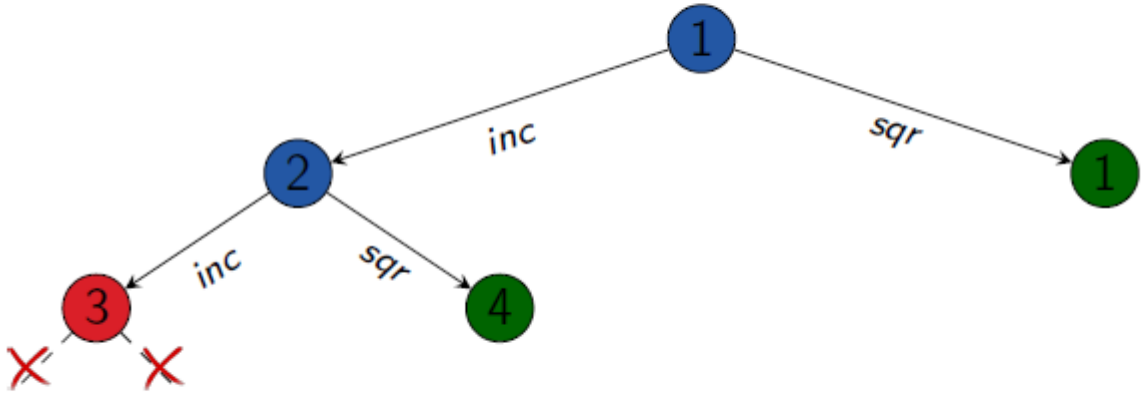


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

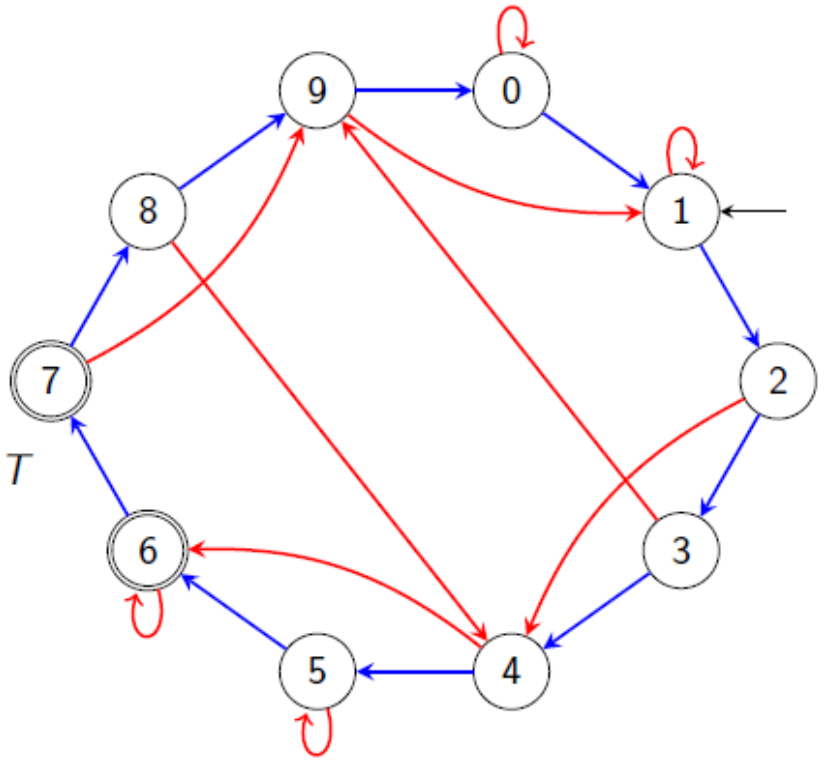


depth limit: 2

generated nodes: 1+3+5

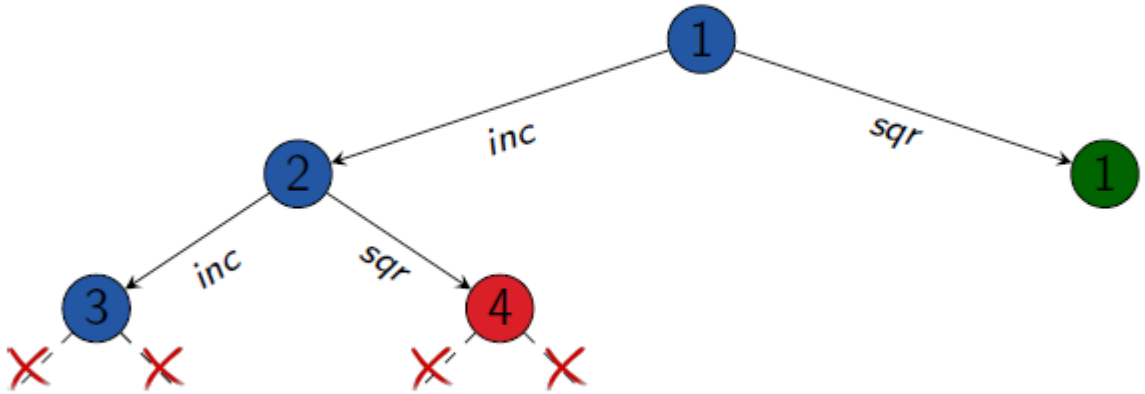


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

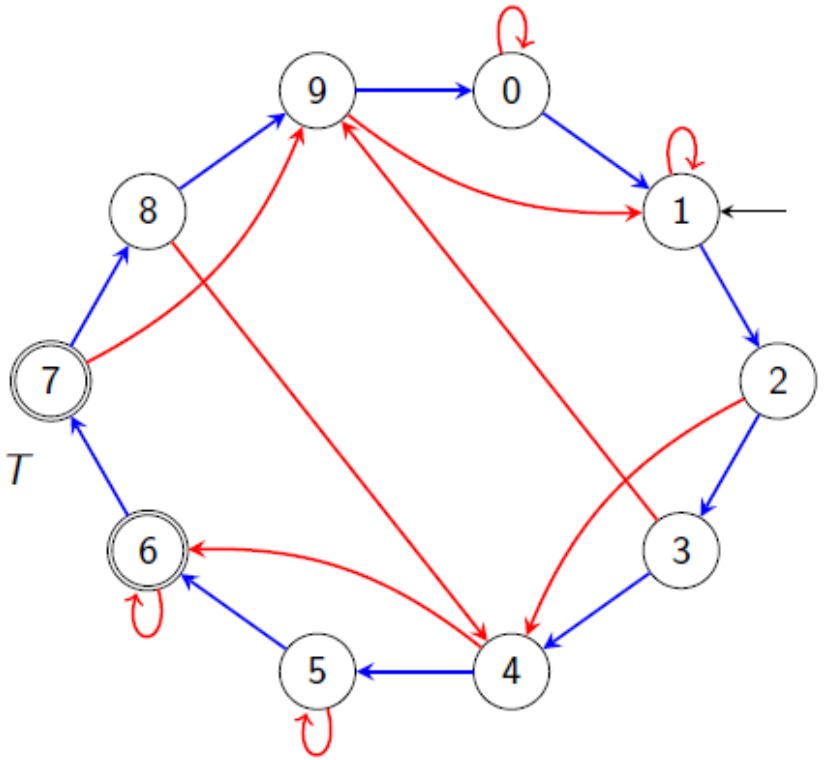


depth limit: 2

generated nodes: 1+3+5

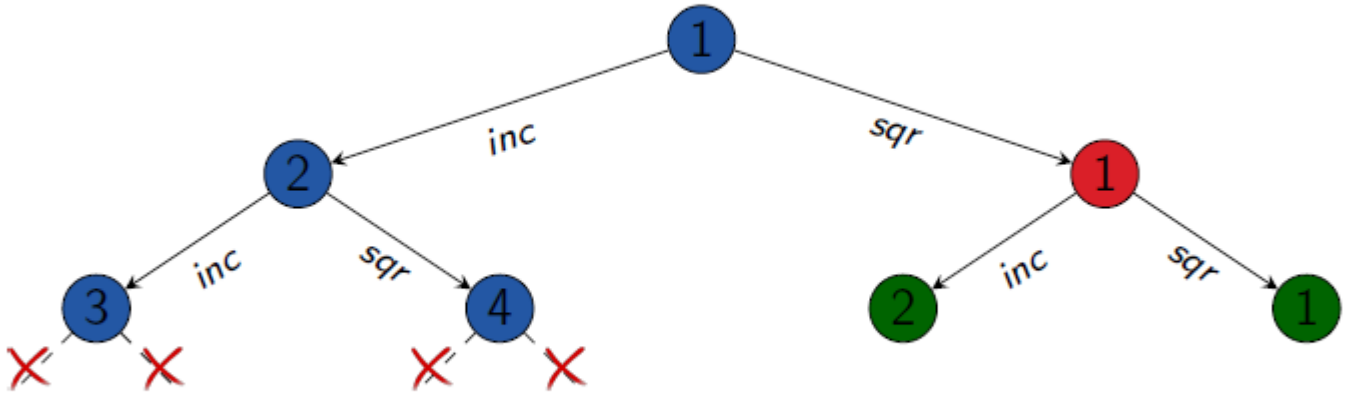


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

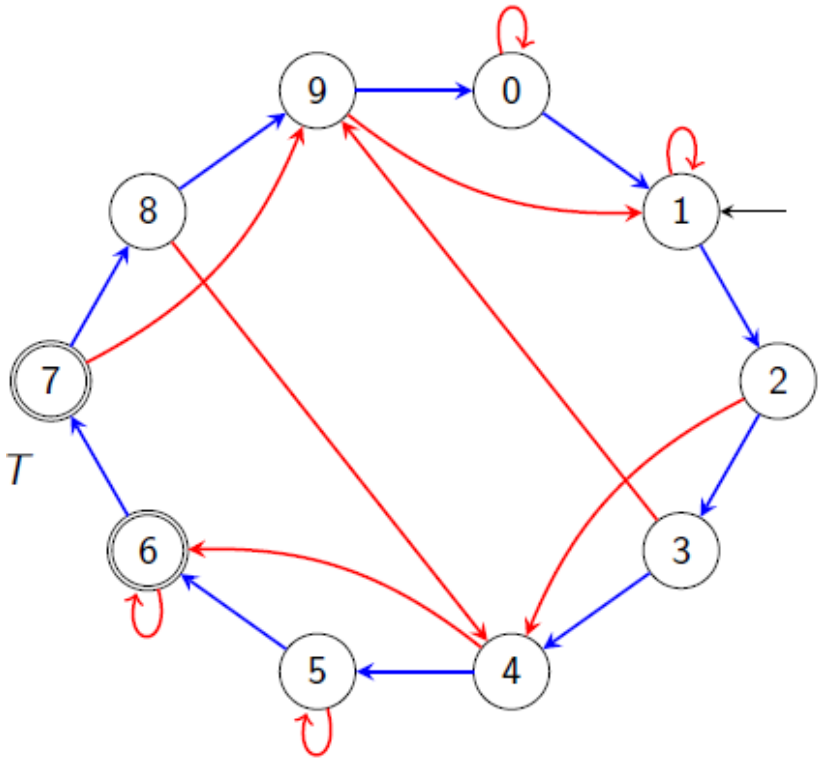


depth limit: 2

generated nodes: 1+3+7

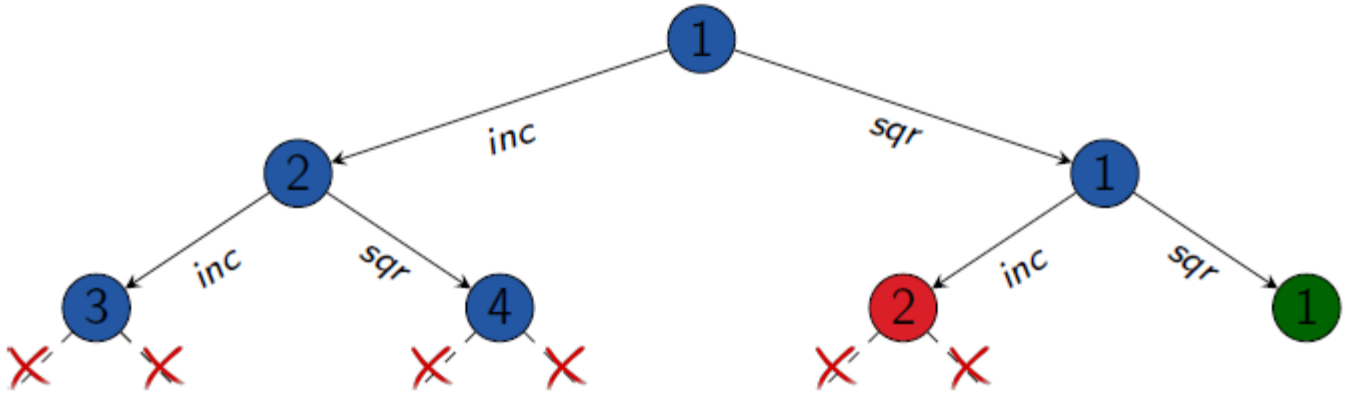


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

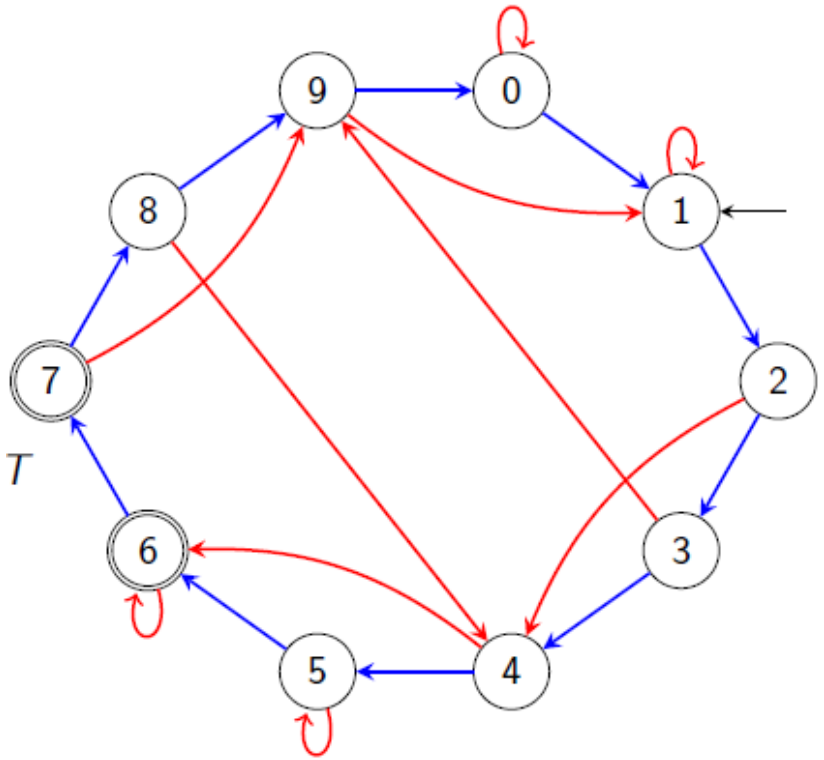


depth limit: 2

generated nodes: 1+3+7

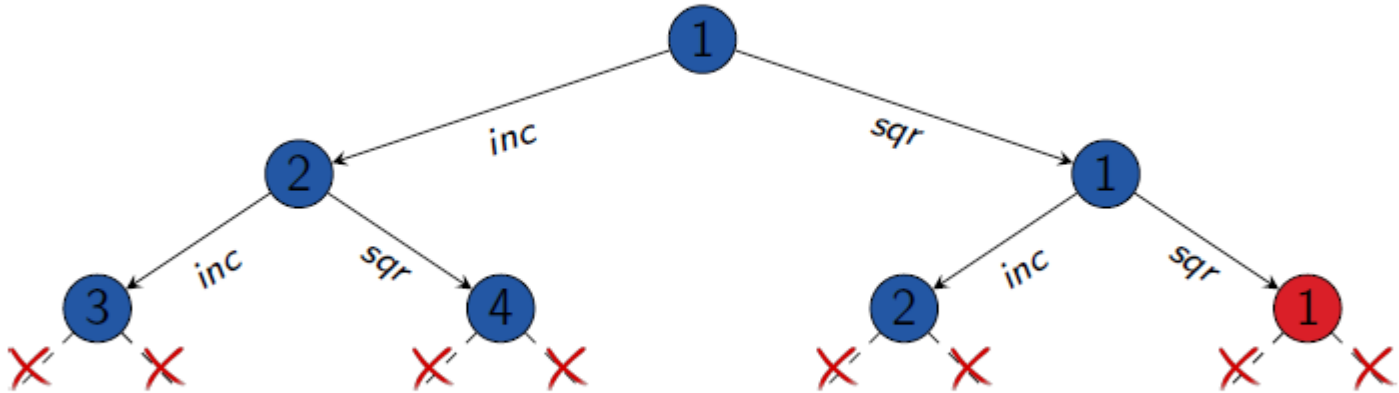


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

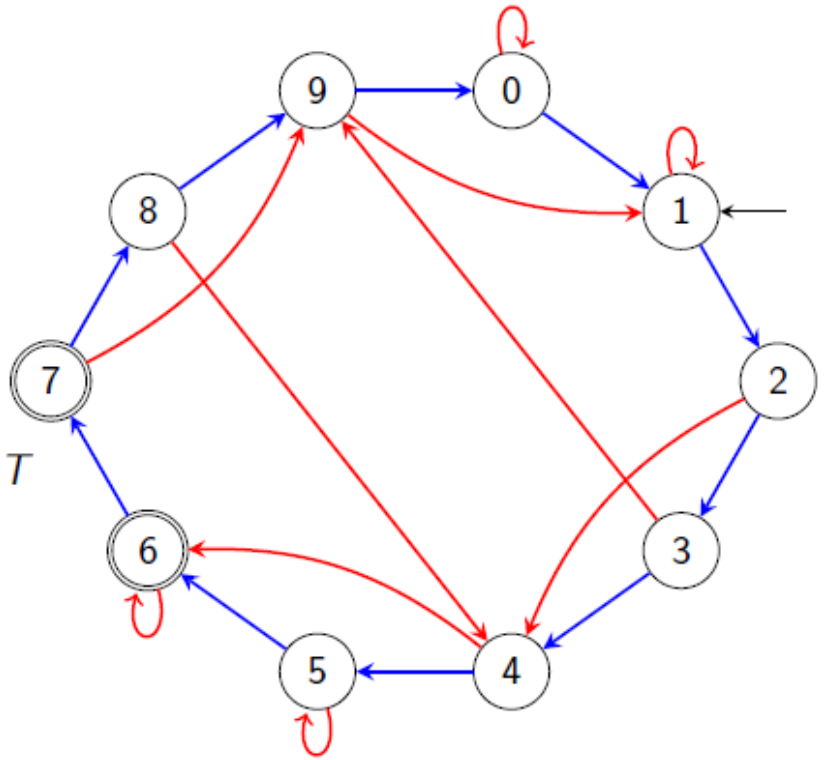


depth limit: 2

generated nodes: 1+3+7

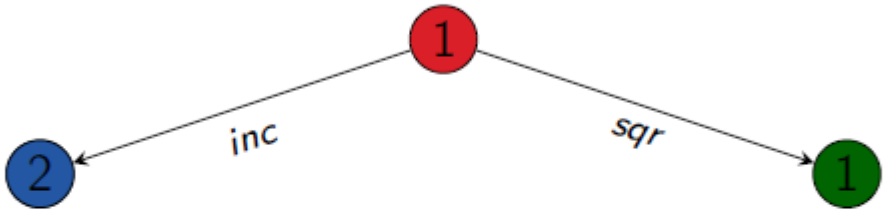


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

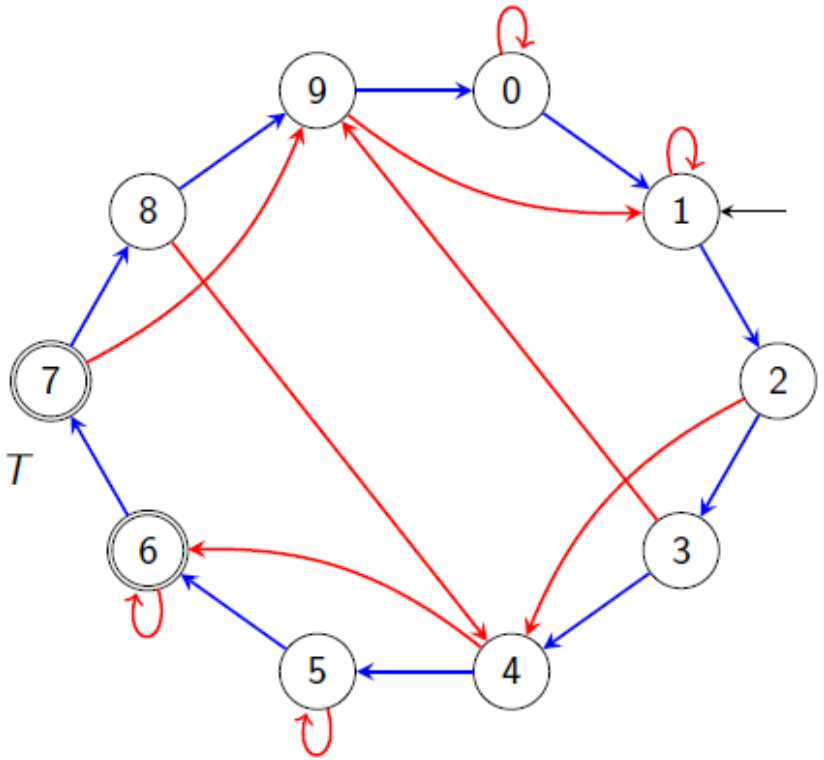


depth limit: 3

generated nodes: 1+3+7+3

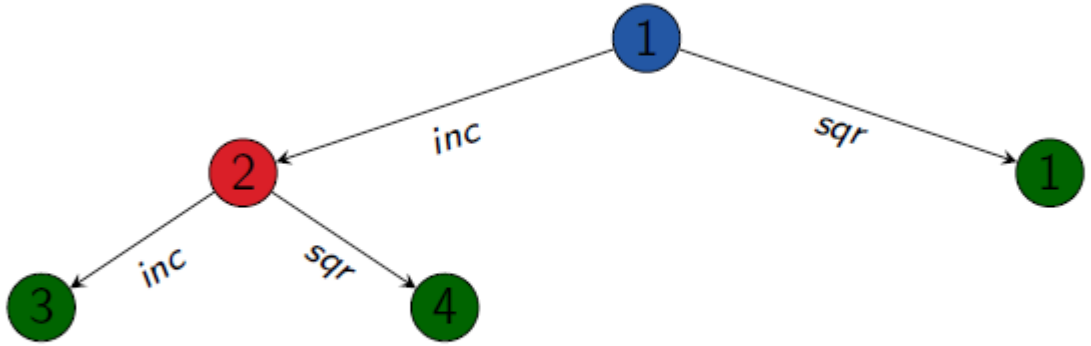


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

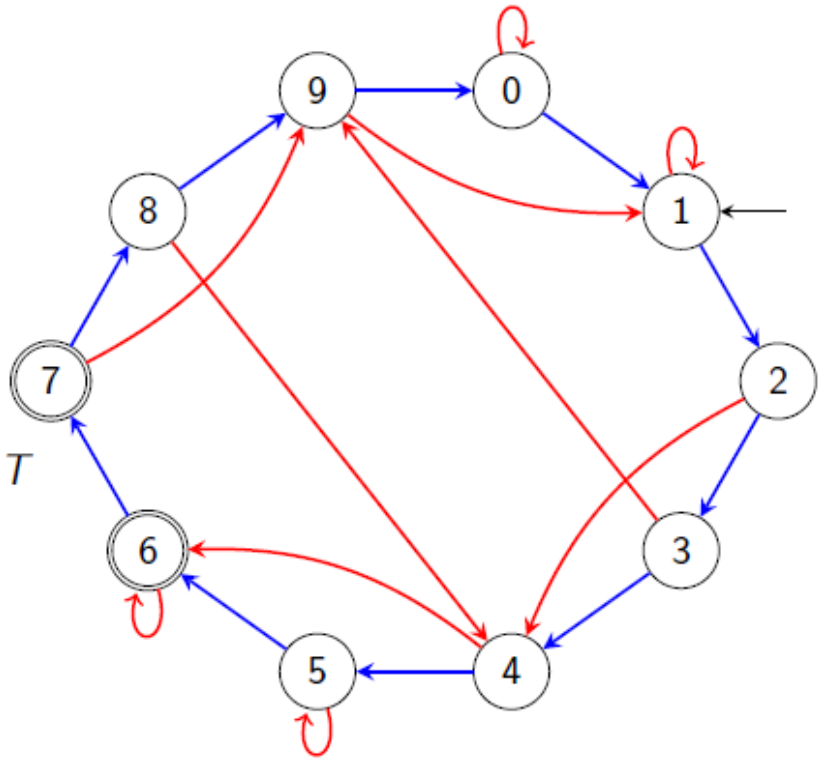


depth limit: 3

generated nodes: 1+3+7+5

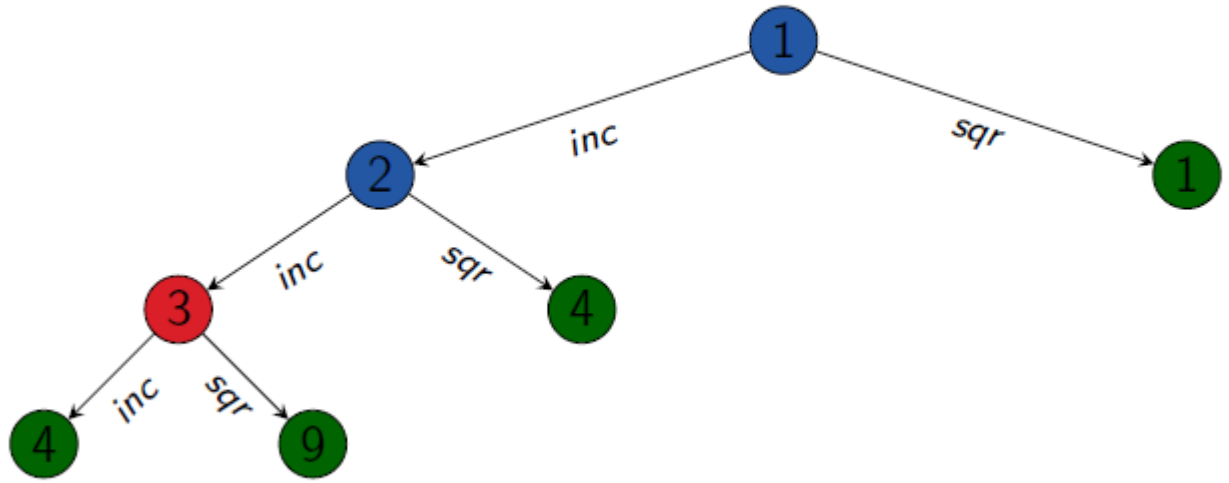


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

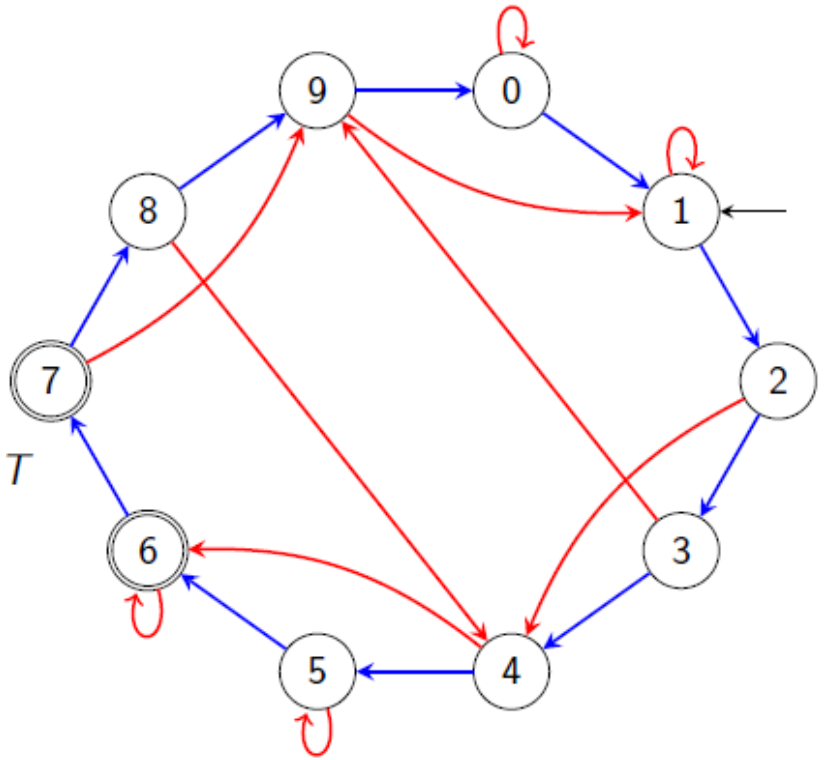


depth limit: 3

generated nodes: 1+3+7+7

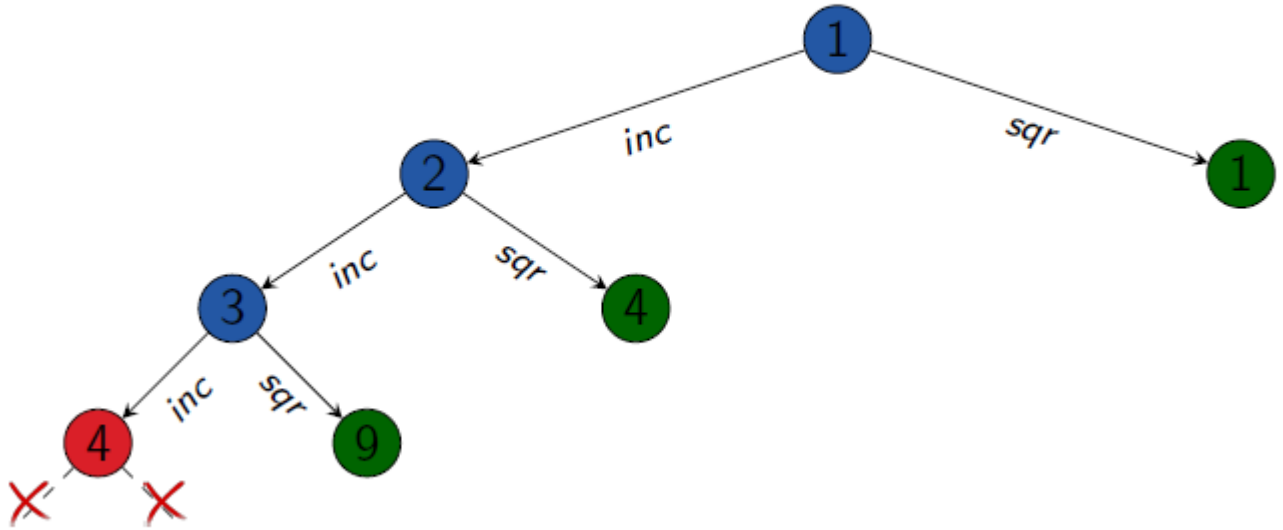


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

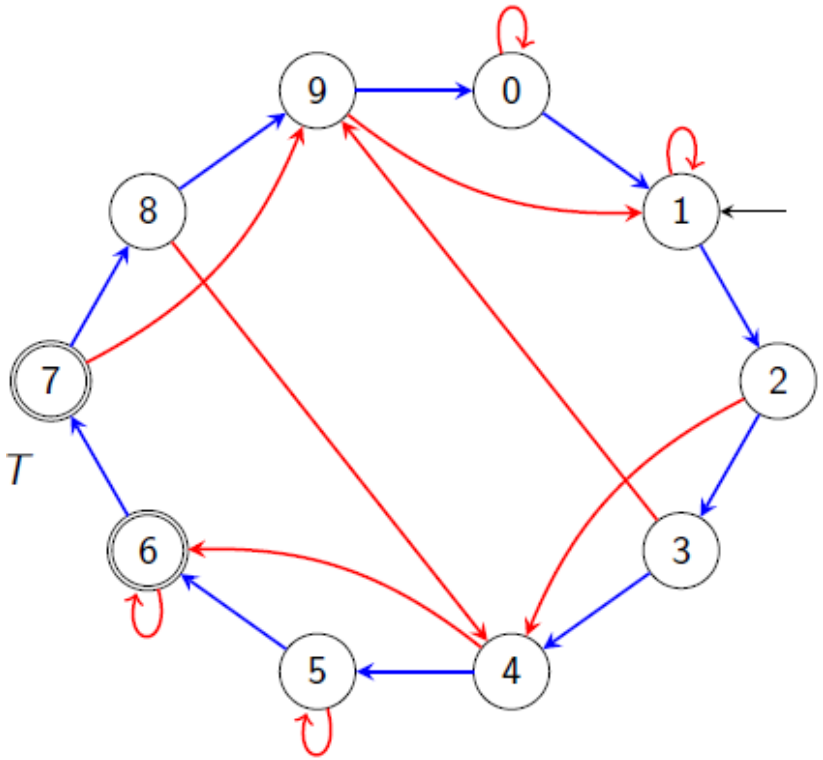


depth limit: 3

generated nodes: 1+3+7+7

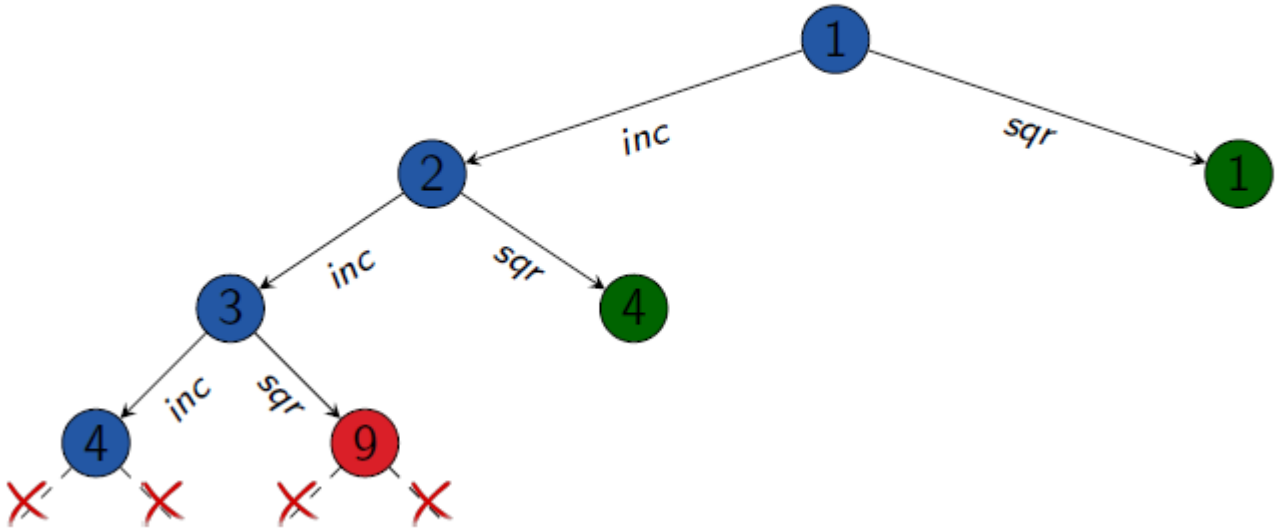


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem

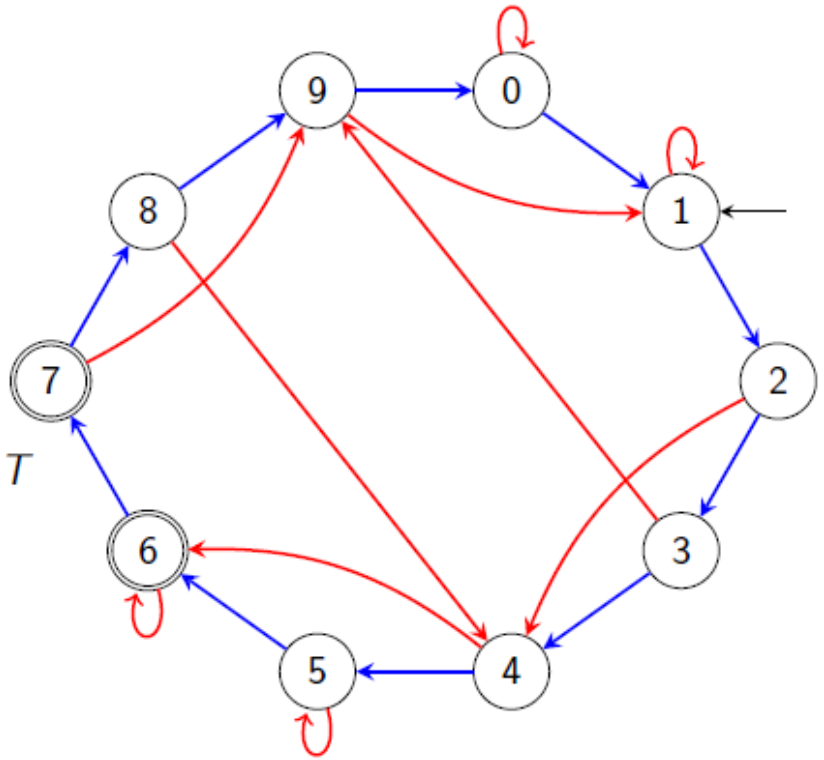


depth limit: 3

generated nodes: 1+3+7+7

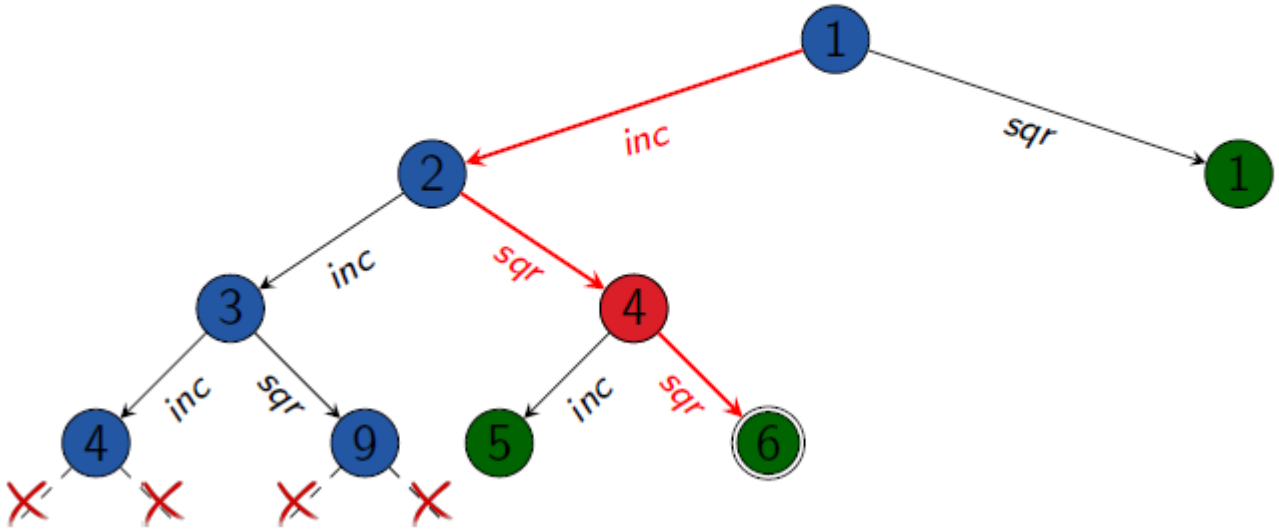


Iterative Deepening Depth-first Search on Bounded Inc-Sqr Problem



depth limit: 3

generated nodes: 1+3+7+9=20



Iterative Deepening Depth-first Search (Depth limited Search)

Properties of Iterative Deepening Depth-first Search:

- It combines advantages of breadth-first and depth-first search
- It is optimal (if all actions have same cost)
- It is semi-complete (why?)

Iterative Deepening Depth-first Search (Depth limited Search)

Properties of Iterative Deepening Depth-first Search:

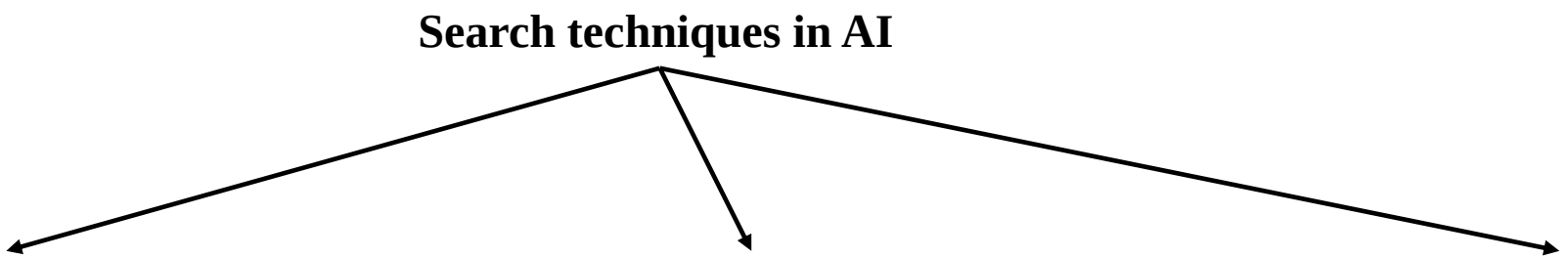
- Time Complexity: If **b ≥ 2** branching factor and **d** minimal solution depth, then time complexity is $O(b^d)$
- Space Complexity: O(bd)

Blind Search (All search technique comparison)

	search algorithm				
criterion	breadth-first	uniform cost	depth-first	depth-bounded	iterative deepening
complete?	yes*	yes	no	no	semi
optimal?	yes**	yes	no	no	yes**
time	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
space	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(bm)$	$O(b\ell)$	$O(bd)$

$b \geq 2$ branching factor
d minimal solution depth
m maximal search depth
ℓ depth bound
c^* optimal solution cost
$\varepsilon > 0$ minimal action cost

Problem solving by search:



Uninformed search (blind search) algorithms:

Breadth-first search
Uniform cost search
Depth-first search & Depth-limited search
Iterative deepening search

Informed search algorithms:

Greedy best-first search
A*
AO*
Hill Climbing,
Simulated Annealing

Adversarial search algorithms:

MIN-MAX Algorithm
Alpha-Beta Cut-off algorithm

Informed search algorithms

- Greedy best-first search
- A*
- AO*
- Hill Climbing,
- Simulated Annealing

Search algorithms considered so far: uninformed ("blind"): use no information besides formal definition to solve a problem

Scale poorly: high time and space requirements and suitable for simple problems (time complexity usually $O(b^d)$)

Informed search algorithms

Definition (heuristic)

Let $\mathcal{S}$ be a state space with states S.
A heuristic function or heuristic for $\mathcal{S}$ is a function

$$h : S \to \mathbb{R}_0^+ \cup \{\infty\},$$

mapping each state to a non-negative number (or ∞).

Informed search algorithms

Idea:

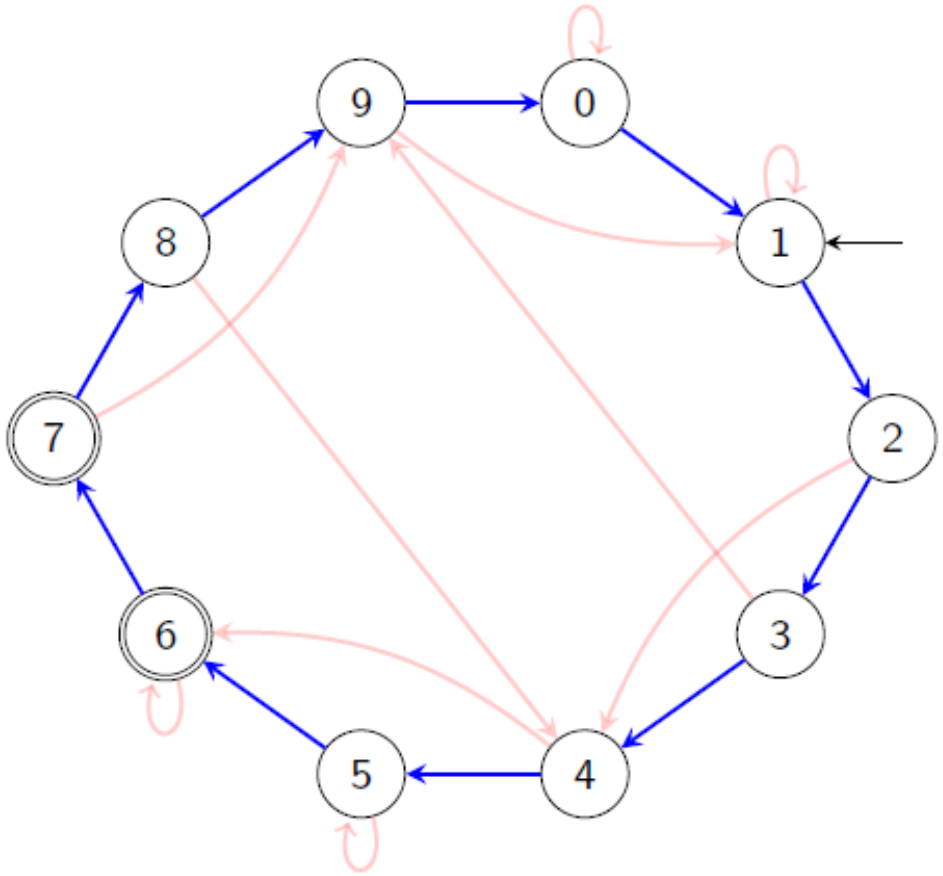
- h(s) estimates distance (= cost of cheapest path) from s to closest goal state.
- heuristics can be arbitrary functions

Intuition:

- The closer h is to true goal distance, the more efficient the search using h
- The better h separates states that are close to the goal from states that are far, the more efficient the search using h

Informed search algorithms

bounded inc-and-square:



possible heuristics:

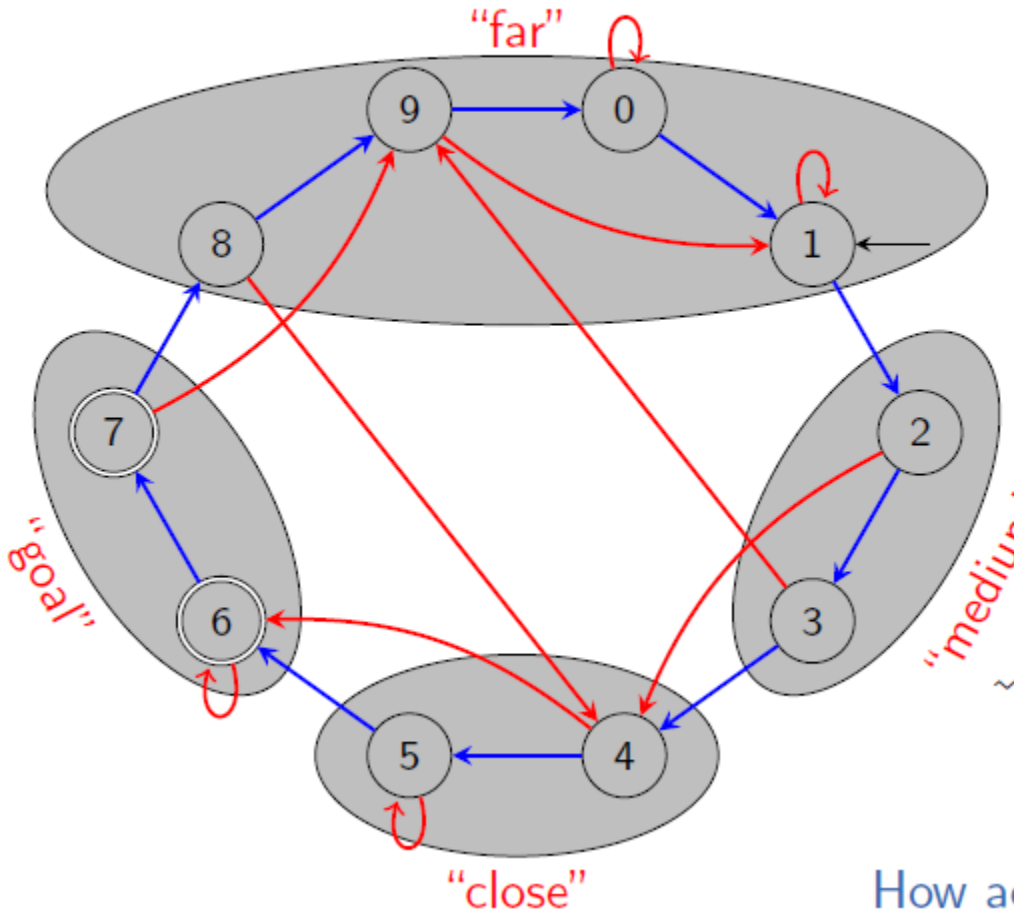
$$h_1(s) = \begin{cases} 0 & \text{if } s = 7 \\ (16 - s)\%10 & \text{otherwise} \end{cases}$$

$\rightsquigarrow$ number of *inc* actions to goal

How accurate is this heuristic?

Informed search algorithms

bounded inc-and-square:



possible heuristics:

$$h_1(s) = \begin{cases} 0 & \text{if } s = 7 \\ (16 - s)\%10 & \text{otherwise} \end{cases}$$

$\rightsquigarrow$ number of *inc* actions to goal

$$h_2(s) = \begin{cases} 0 & \text{if } s \text{ is a "goal"} \\ 1 & s \text{ is "close"} \\ 2 & s \text{ is "medium"} \\ 3 & s \text{ is "far"} \end{cases}$$

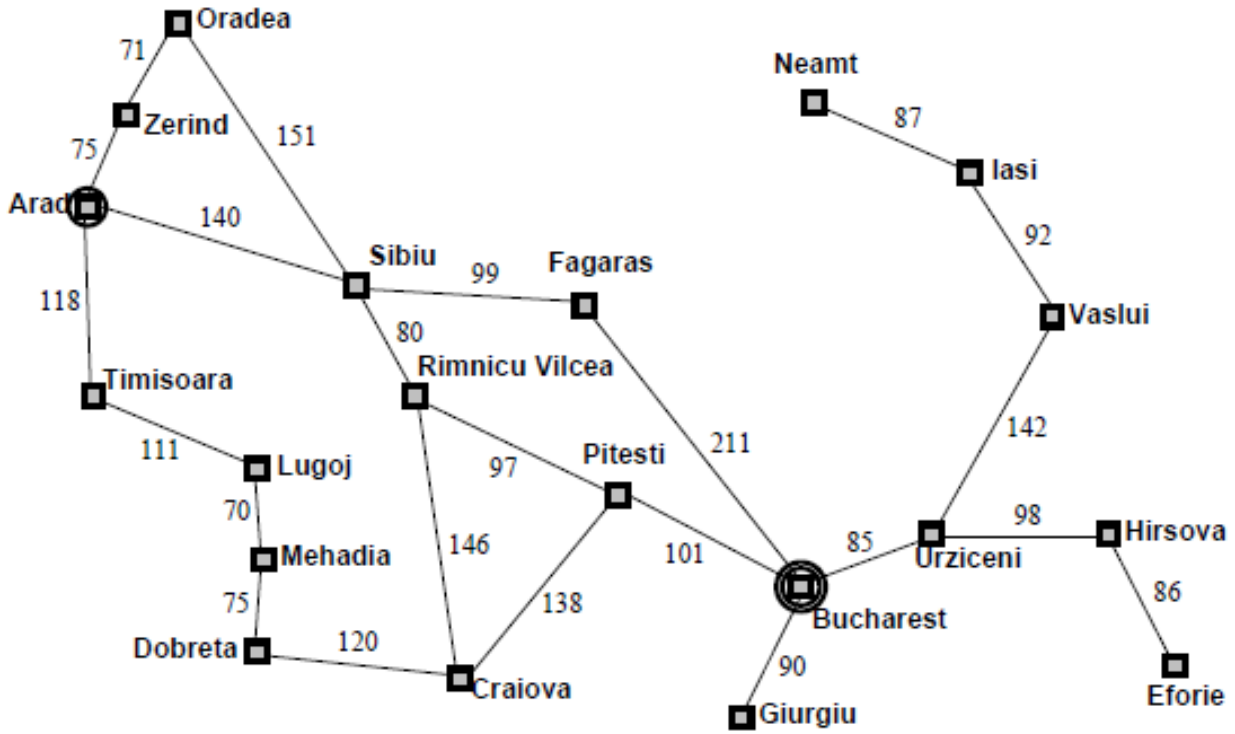
$\rightsquigarrow$ categorize states

How accurate is this heuristic?

Informed search algorithms

Example: Route Planning in Romania

Possible heuristic: straight-line distance to Bucharest



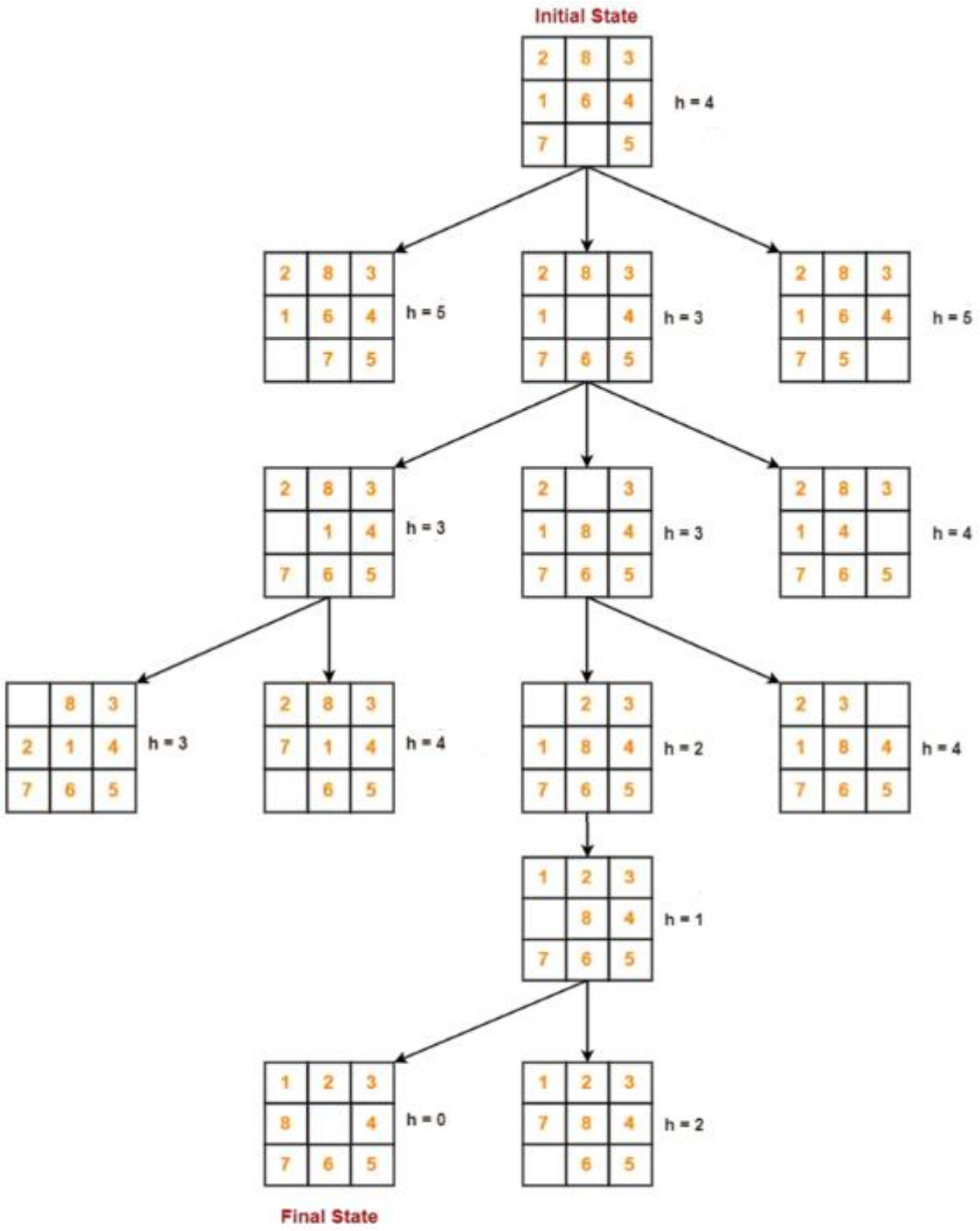
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Introduction to AI

Informed search algorithms

Example: 8 Puzzle Problem

Possible heuristic: Approximate steps required to reach the goal state



Informed search algorithms

Heuristics estimate distance of a state to the goal.

It can be used to focus search on promising states.

Search algorithms that use heuristics:

- Greedy best-first search
- A*
- AO*
- Hill Climbing,
- Simulated Annealing

Informed search algorithms

Perfect Heuristic

Definition (heuristic)

Let $\mathcal{S}$ be a state space with states S.
A heuristic function or heuristic for $\mathcal{S}$ is a function

$$h : S \to \mathbb{R}_0^+ \cup \{\infty\},$$

mapping each state to a non-negative number (or ∞).

Informed search algorithms

Perfect Heuristic

Definition (heuristic)

Let $\mathcal{S}$ be a state space with states S.
A heuristic function or heuristic for $\mathcal{S}$ is a function

$$h : S \to \mathbb{R}_0^+ \cup \{\infty\},$$

mapping each state to a non-negative number (or ∞).

Definition (perfect heuristic)

Let $\mathcal{S}$ be a state space with states S.

The perfect heuristic for $\mathcal{S}$, written h^*, maps each state $s \in S$

- to the cost of an optimal solution for s, or
- to ∞ if no solution for s exists.

.

Informed search algorithms

Properties of Heuristic

Definition (safe, goal-aware, admissible, consistent)

Let $\mathcal{S}$ be a state space with states S.

A heuristic h for $\mathcal{S}$ is called

- safe if $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$
- goal-aware if $h(s) = 0$ for all goal states s
- admissible if $h(s) \leq h^*(s)$ for all states $s \in S$
- consistent if $h(s) \leq cost(a) + h(s')$ for all transitions $s \xrightarrow{a} s'$

Informed search algorithms

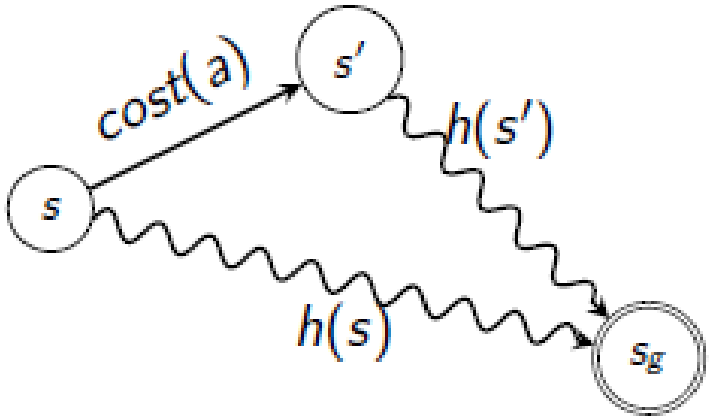
Properties of Heuristic

Definition (safe, goal-aware, admissible, consistent)

Let $\mathcal{S}$ be a state space with states S.

A heuristic h for $\mathcal{S}$ is called

- safe if $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$
- goal-aware if $h(s) = 0$ for all goal states s
- admissible if $h(s) \leq h^*(s)$ for all states $s \in S$
- consistent if $h(s) \leq cost(a) + h(s')$ for all transitions $s \xrightarrow{a} s'$



Informed search algorithms

Properties of Heuristics: Examples

Route Planning in Romania

straight-line distance:

- safe
- goal-aware
- admissible
- consistent

Why?

Informed search algorithms

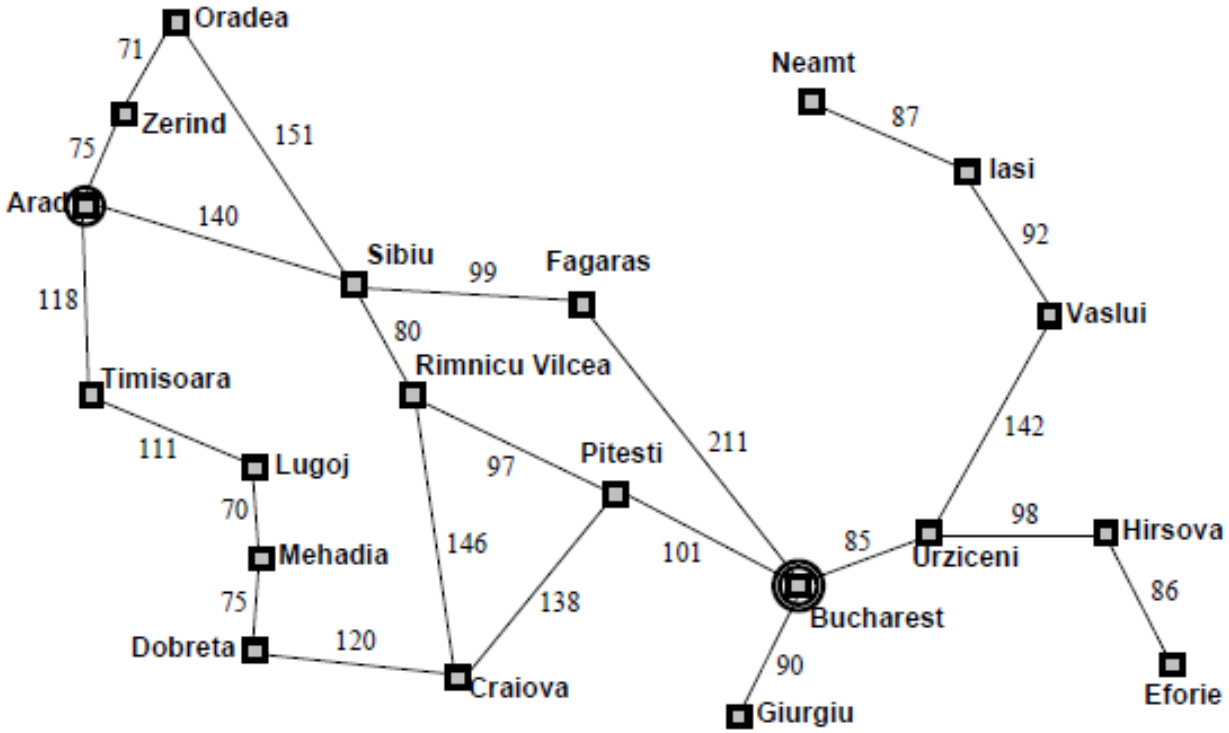
Properties of Heuristics: Examples

Route Planning in Romania

straight-line distance:

- safe
- goal-aware
- admissible
- consistent

Why?



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed search algorithms

Properties of Heuristics: Connections

Theorem (admissible $\Longrightarrow$ safe + goal-aware)

Let h be an admissible heuristic.

Then h is safe and goal-aware.

Informed search algorithms

Properties of Heuristics: Connections

Theorem (admissible $\Longrightarrow$ safe + goal-aware)

Let h be an admissible heuristic.

Then h is safe and goal-aware.

Theorem (goal-aware + consistent $\Longrightarrow$ admissible)

Let h be a goal-aware and consistent heuristic.

Then h is admissible.

Informed search algorithms

Properties of Heuristics: Summaries

- perfect heuristic h^*: true cost to the goal
- important properties: safe, goal-aware, admissible, consistent
- connections between these properties
  - admissible $\Longrightarrow$ safe and goal-aware
  - goal-aware and consistent $\Longrightarrow$ admissible

Informed search algorithms (Heuristic Search Algorithms)

- Heuristic search algorithms use **heuristic functions** to (partially or fully) determine the order of node expansion.

- **Best-first Search:** Best-first search is a class of search algorithms that expand the "**most promising**" node in each iteration using **heuristics**.

- A best-first search is a heuristic search algorithm that evaluates search nodes with an evaluation heuristic function **f** and always expands a node **n** with minimal **f (n)** value.

- Implementation essentially like uniform cost search

Informed search algorithms (Heuristic Search Algorithms)

Important Best-first Search Algorithms:

- $f(n) = h(n.\text{state})$: greedy best-first search
  $\rightsquigarrow$ only the heuristic counts
- $f(n) = g(n) + h(n.\text{state})$: A*
  $\rightsquigarrow$ combination of path cost and heuristic
- $f(n) = g(n) + w \cdot h(n.\text{state})$: weighted A*
  $w \in \mathbb{R}_0^+$ is a parameter
  $\rightsquigarrow$ interpolates between greedy best-first search and A*

Informed search algorithms (Heuristic Search Algorithms)

Important Best-first Search Algorithms:

- $f(n) = h(n.\text{state})$: greedy best-first search
  $\rightsquigarrow$ only the heuristic counts
- $f(n) = g(n) + h(n.\text{state})$: A*
  $\rightsquigarrow$ combination of path cost and heuristic
- $f(n) = g(n) + w \cdot h(n.\text{state})$: weighted A*
  $w \in \mathbb{R}_0^+$ is a parameter
  $\rightsquigarrow$ interpolates between greedy best-first search and A*

Informed search algorithms

Greedy Best First Search

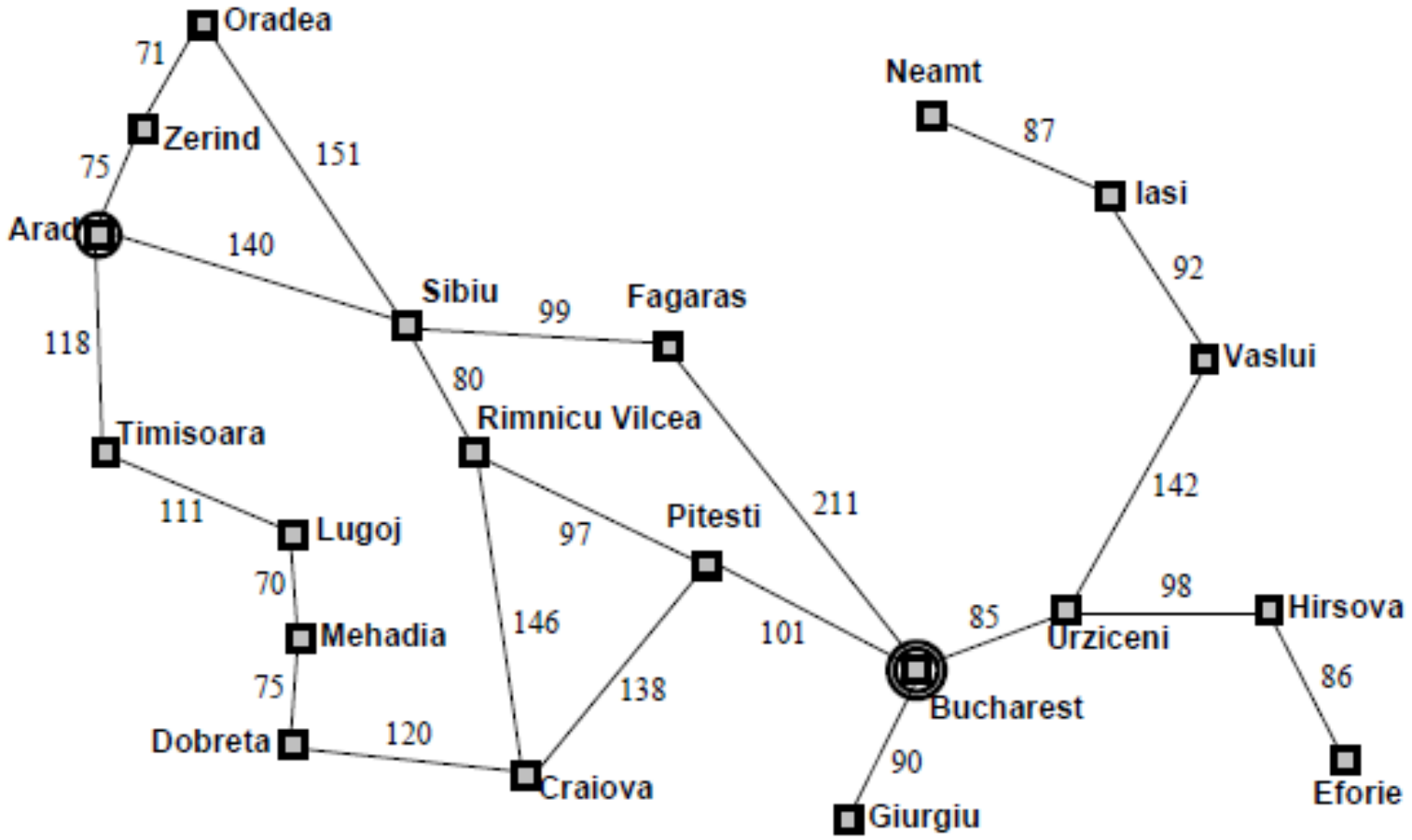
Greedy Best-first Search

only consider the heuristic: $f(n) = h(n.\text{state})$

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning



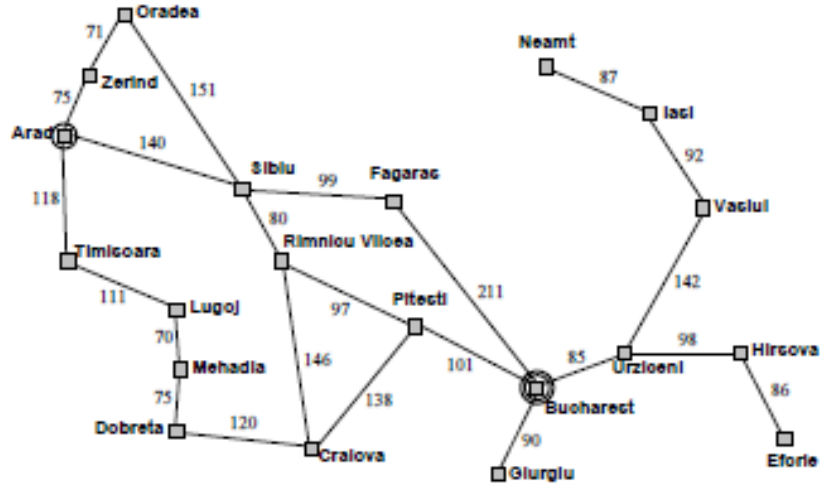
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Introduction to AI

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning



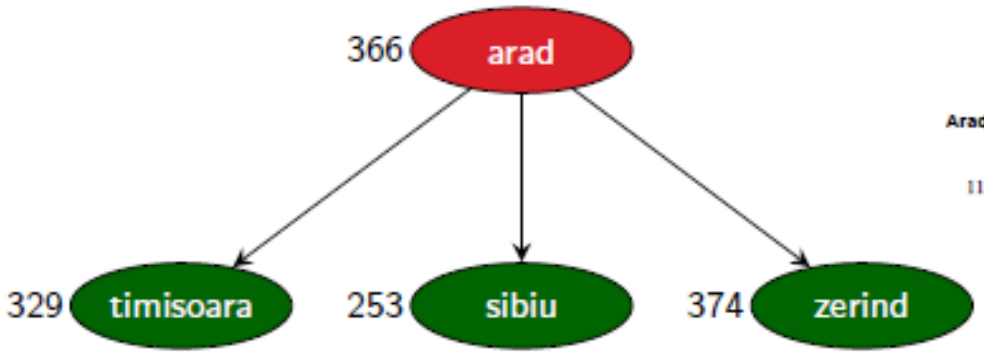
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

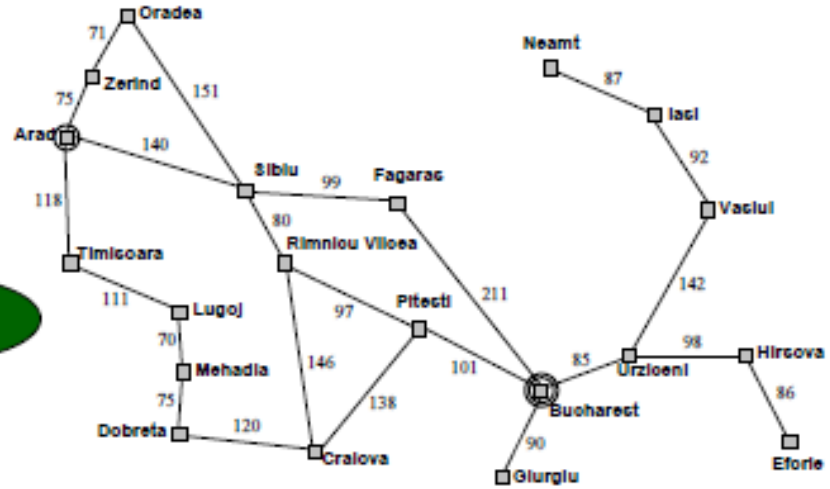
Introduction to AI

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning





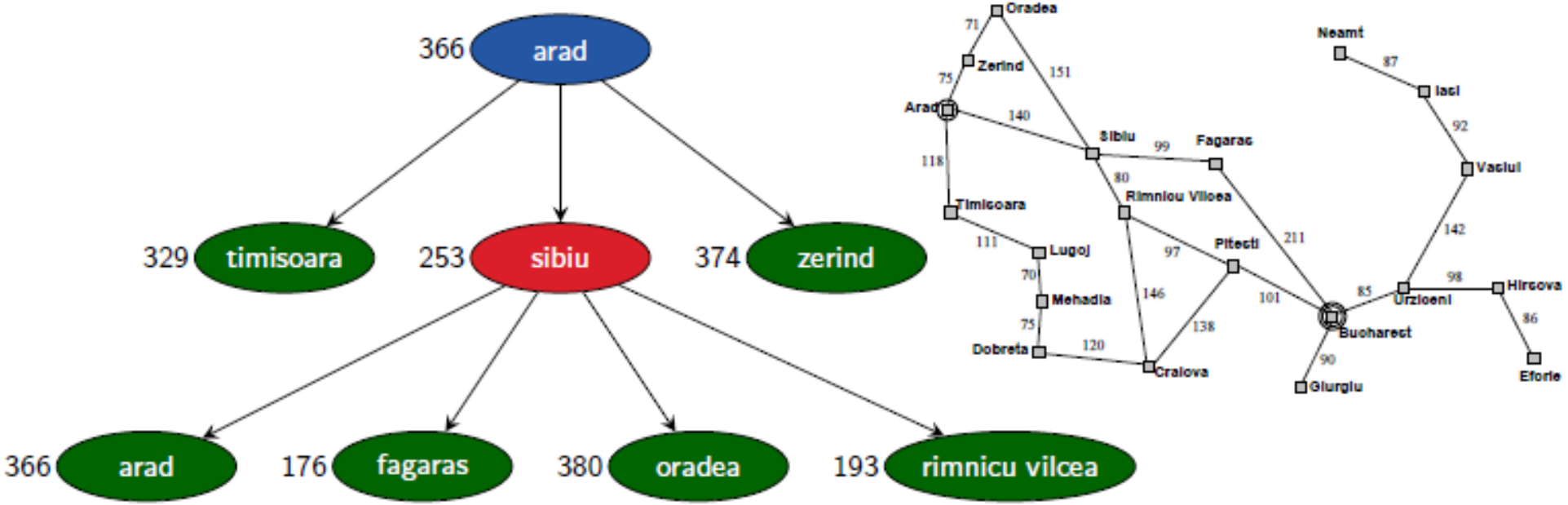
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning



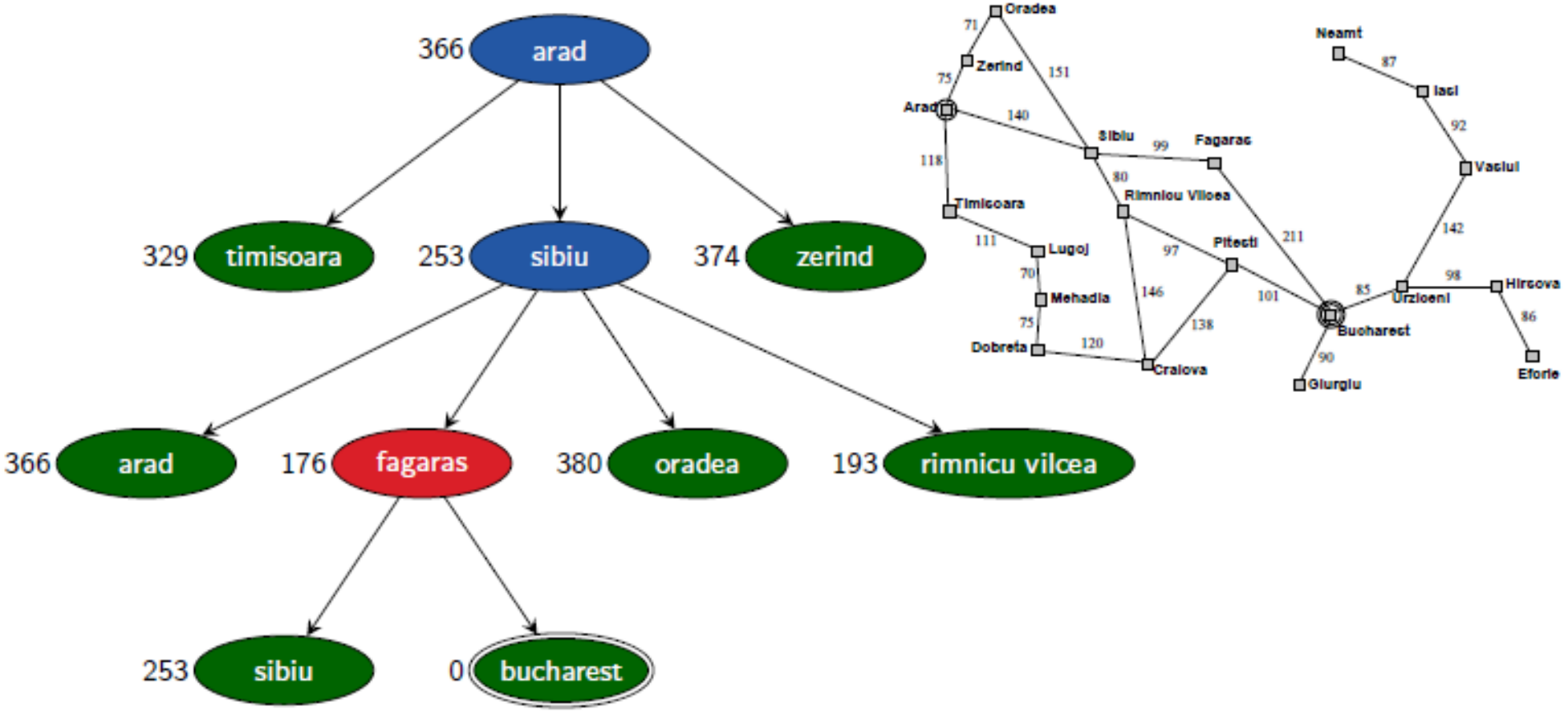
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning



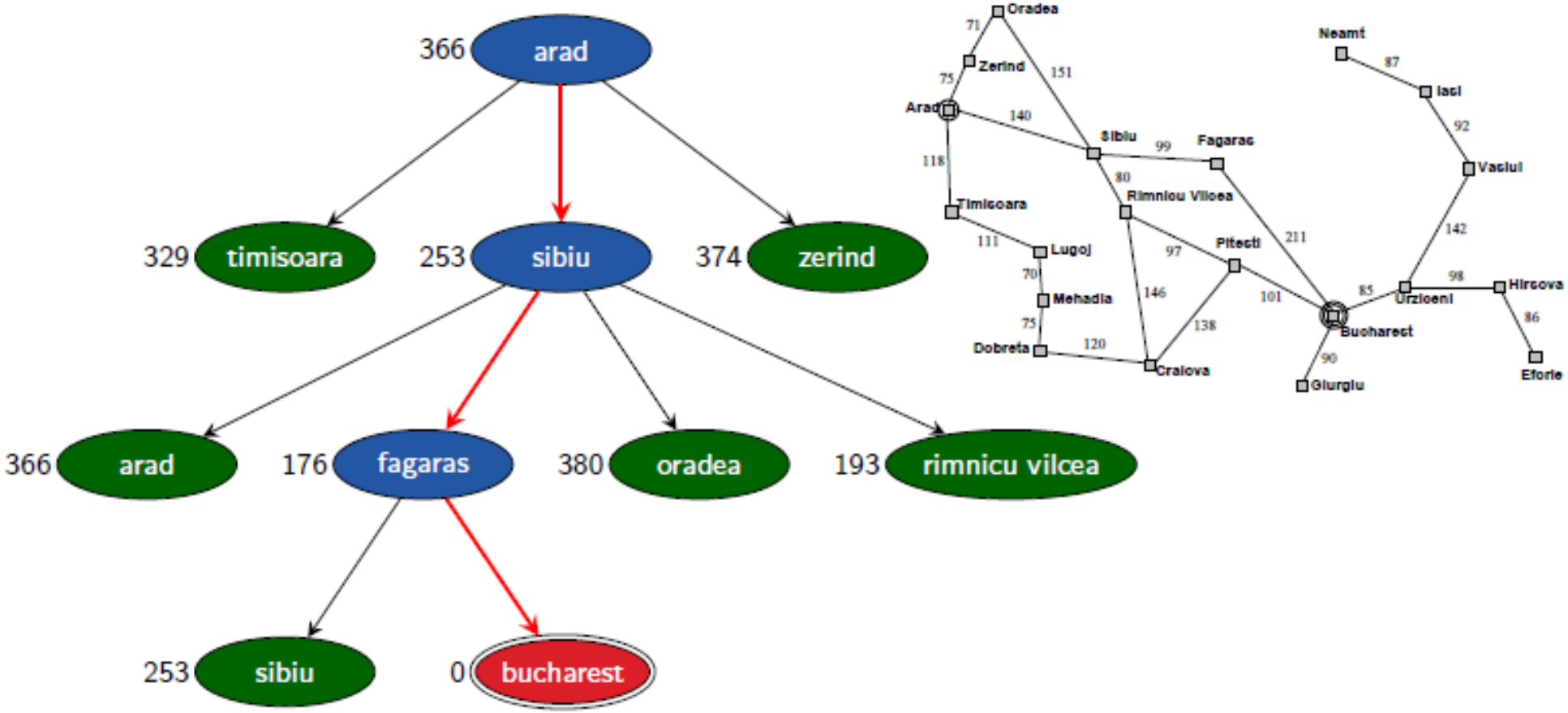
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Greedy Best First Search

Example: Greedy Best-first Search for Route Planning



Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Informed search algorithms

Greedy Best First Search

- Completeness: Complete with safe heuristics
- Optimality: suboptimal: solutions can be arbitrarily bad
- Often fast: one of the fastest search algorithms in practice
- Monotonic transformations of h (e.g. scaling, additive constants) do not affect behaviour

Informed search algorithms

Best First Search (A* Search)

- $f(n) = h(n.\text{state})$: greedy best-first search
  $\rightsquigarrow$ only the heuristic counts
- $f(n) = g(n) + h(n.\text{state})$: A*
  $\rightsquigarrow$ combination of path cost and heuristic
- $f(n) = g(n) + w \cdot h(n.\text{state})$: weighted A*
  $w \in \mathbb{R}_0^+$ is a parameter
  $\rightsquigarrow$ interpolates between greedy best-first search and A*

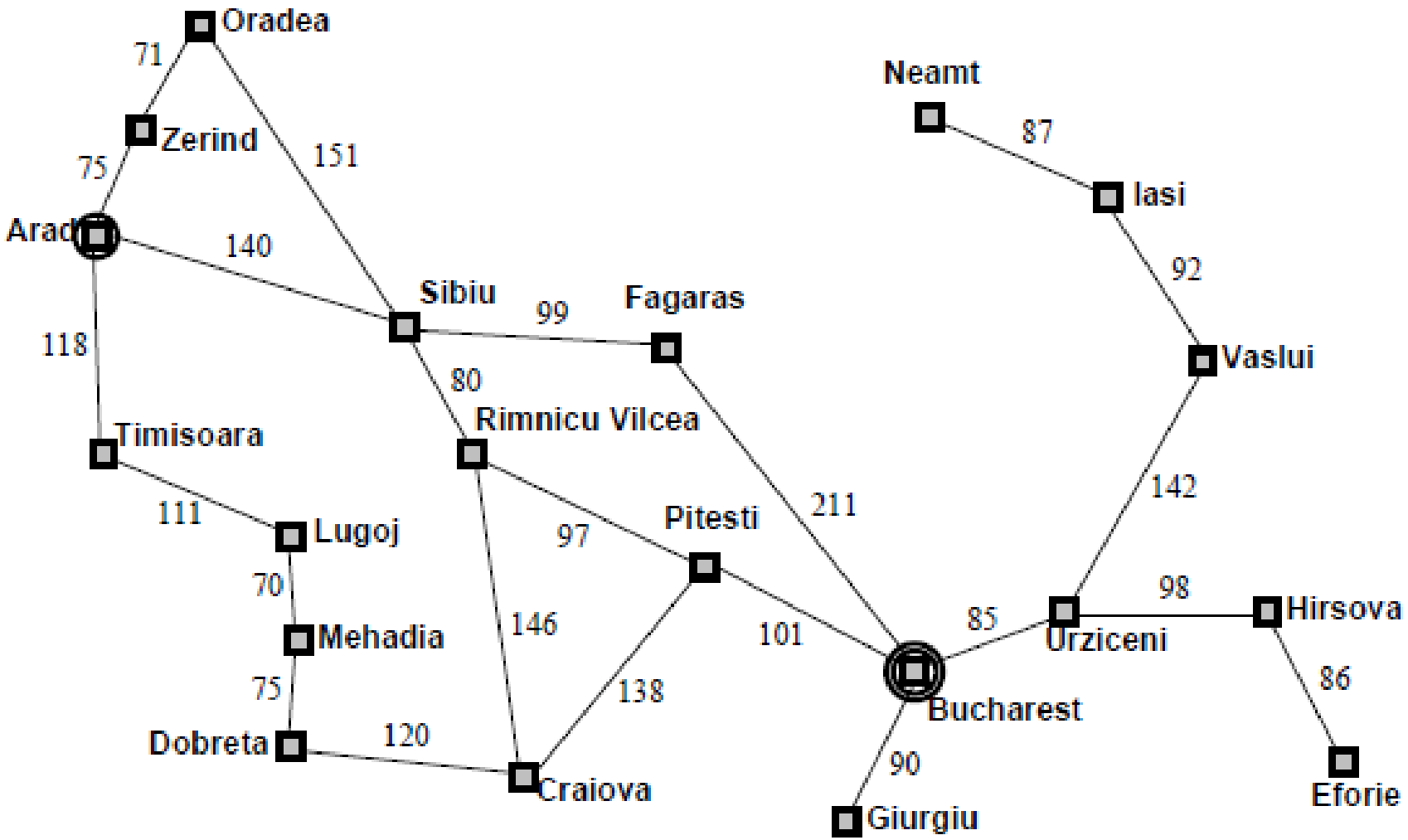
A*

combine greedy best-first search with uniform cost search:
$f(n) = g(n) + h(n.\text{state})$

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



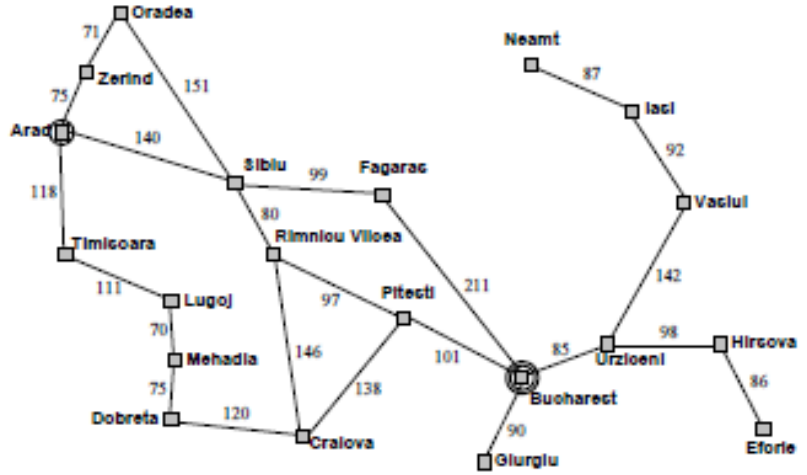
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



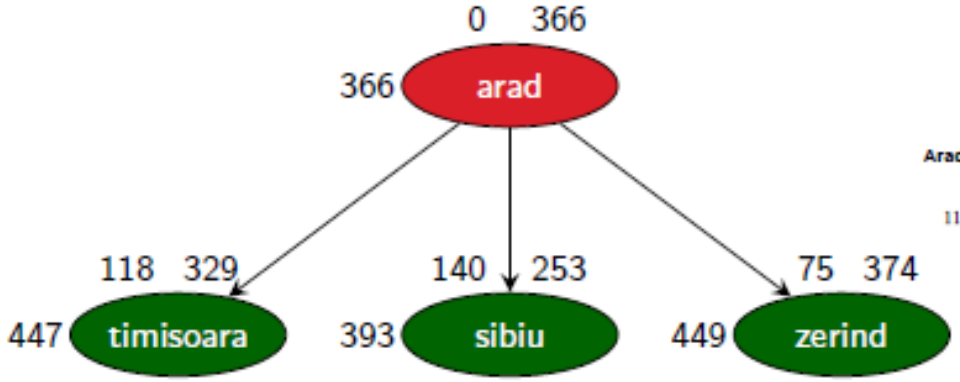
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

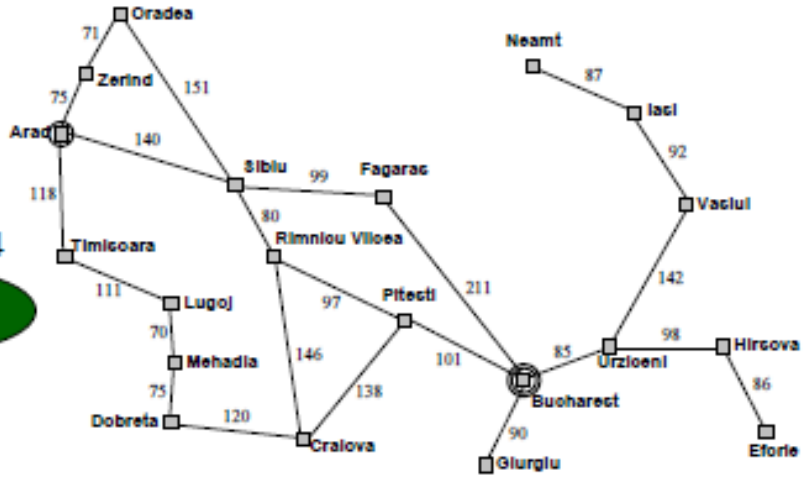
Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning





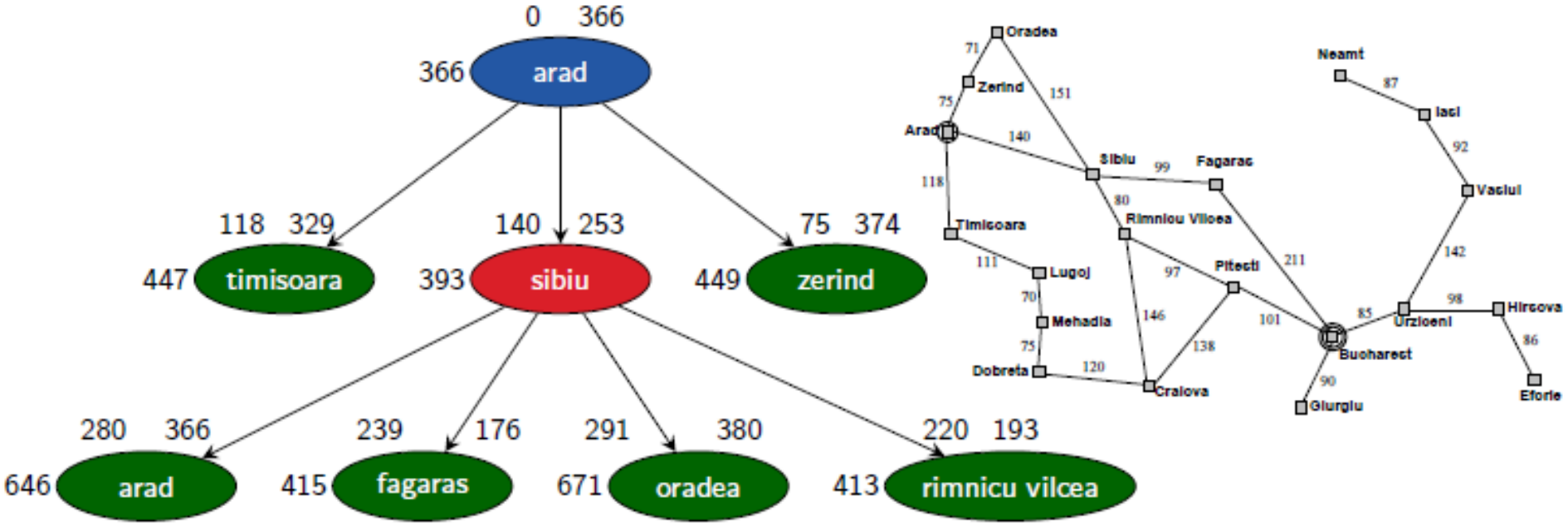
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



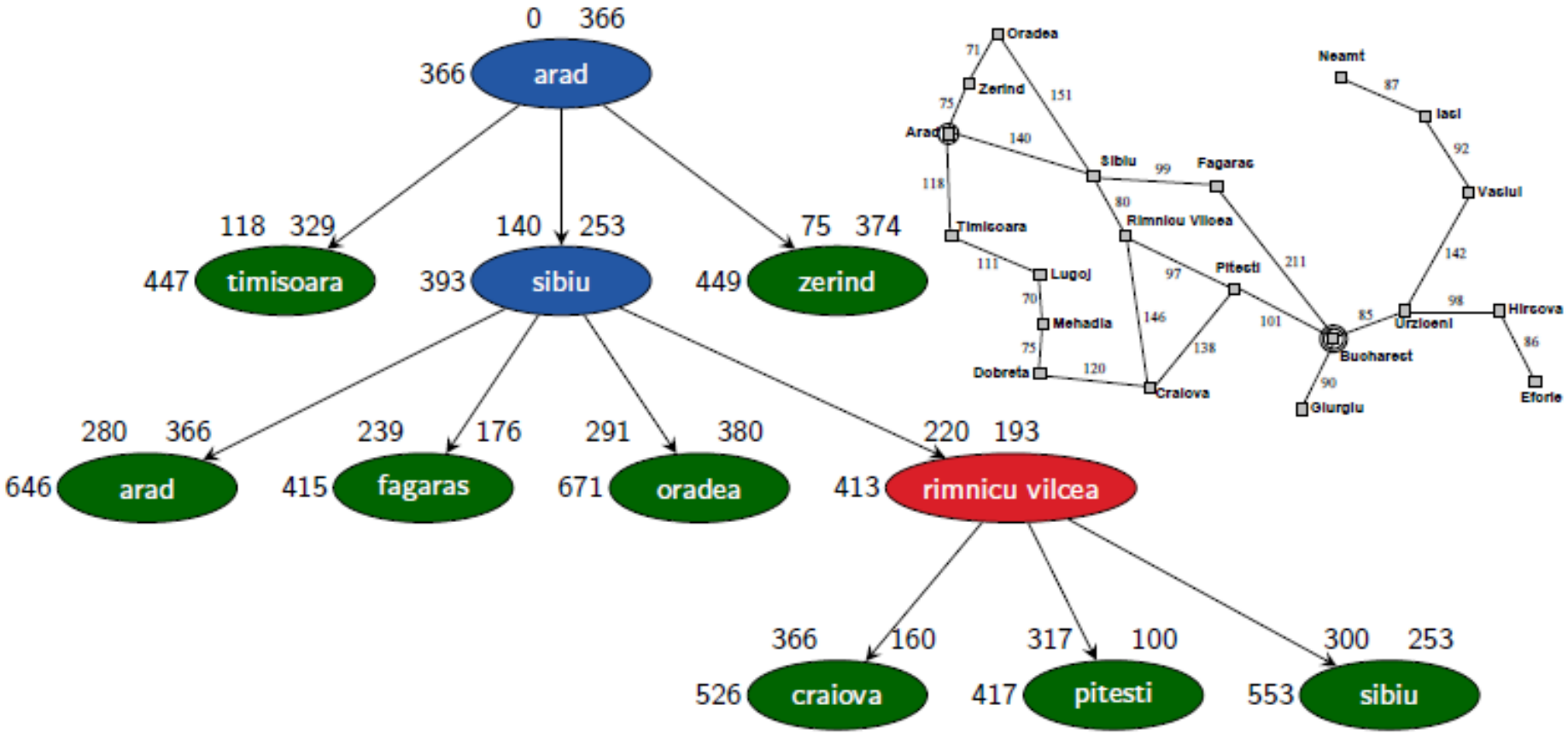
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



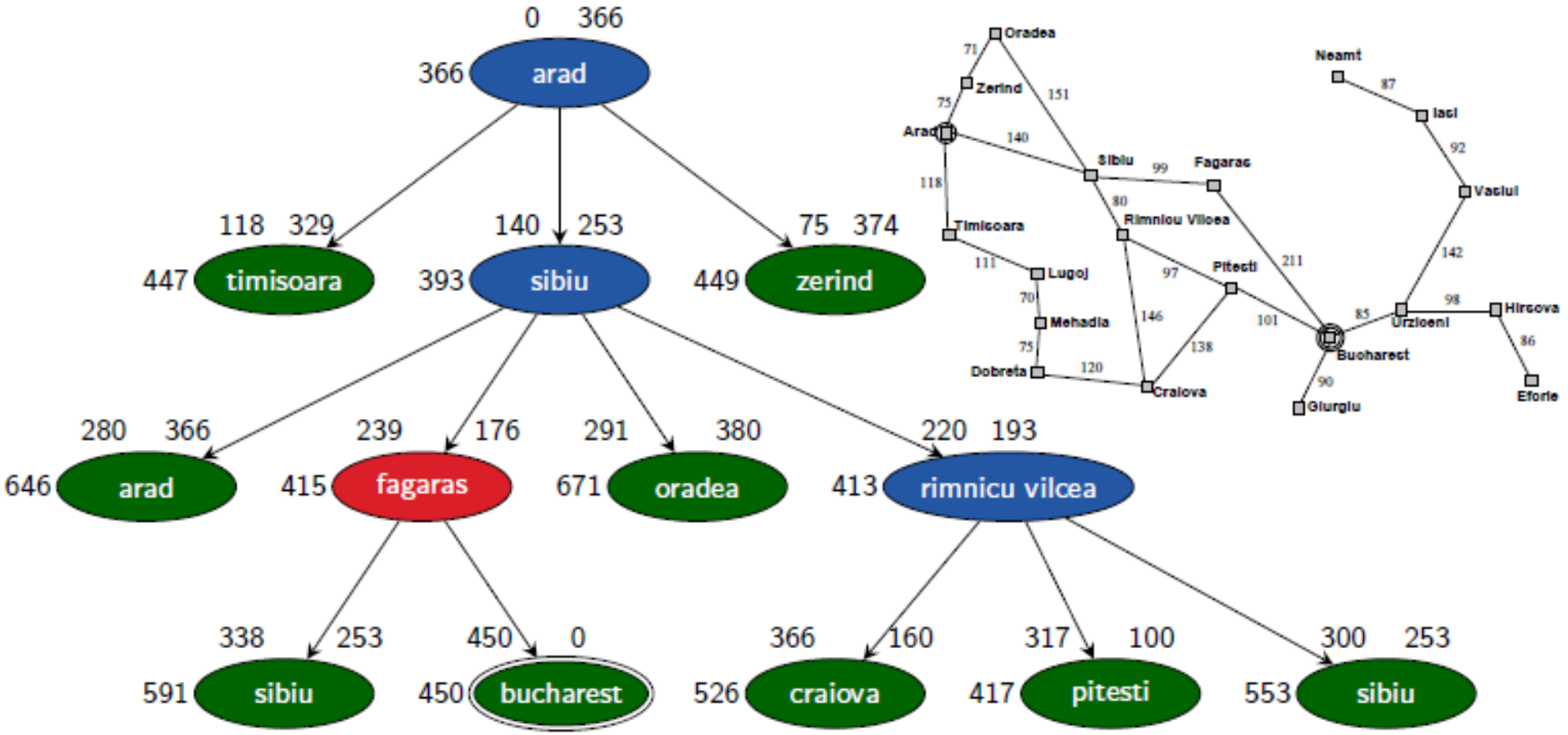
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



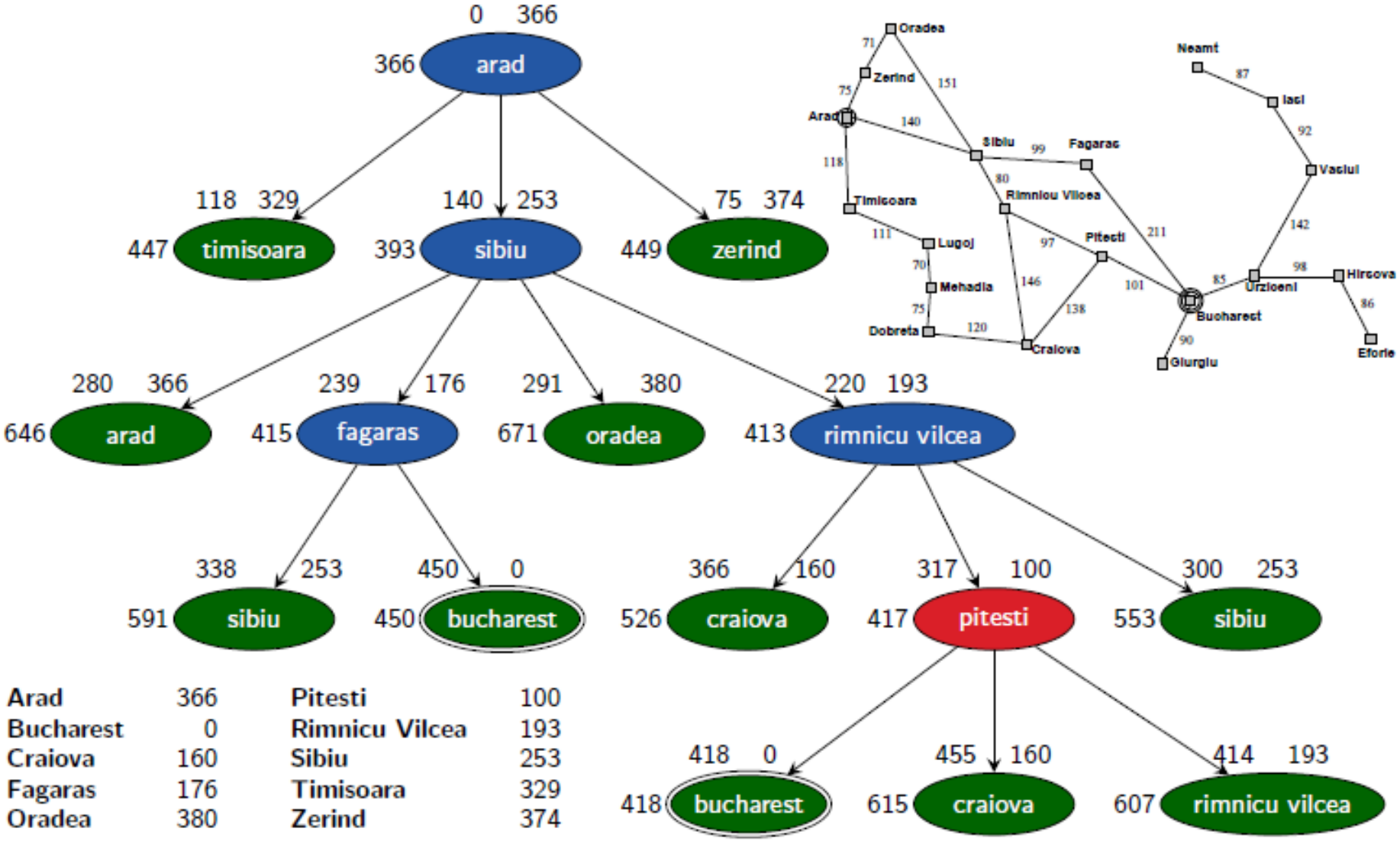
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



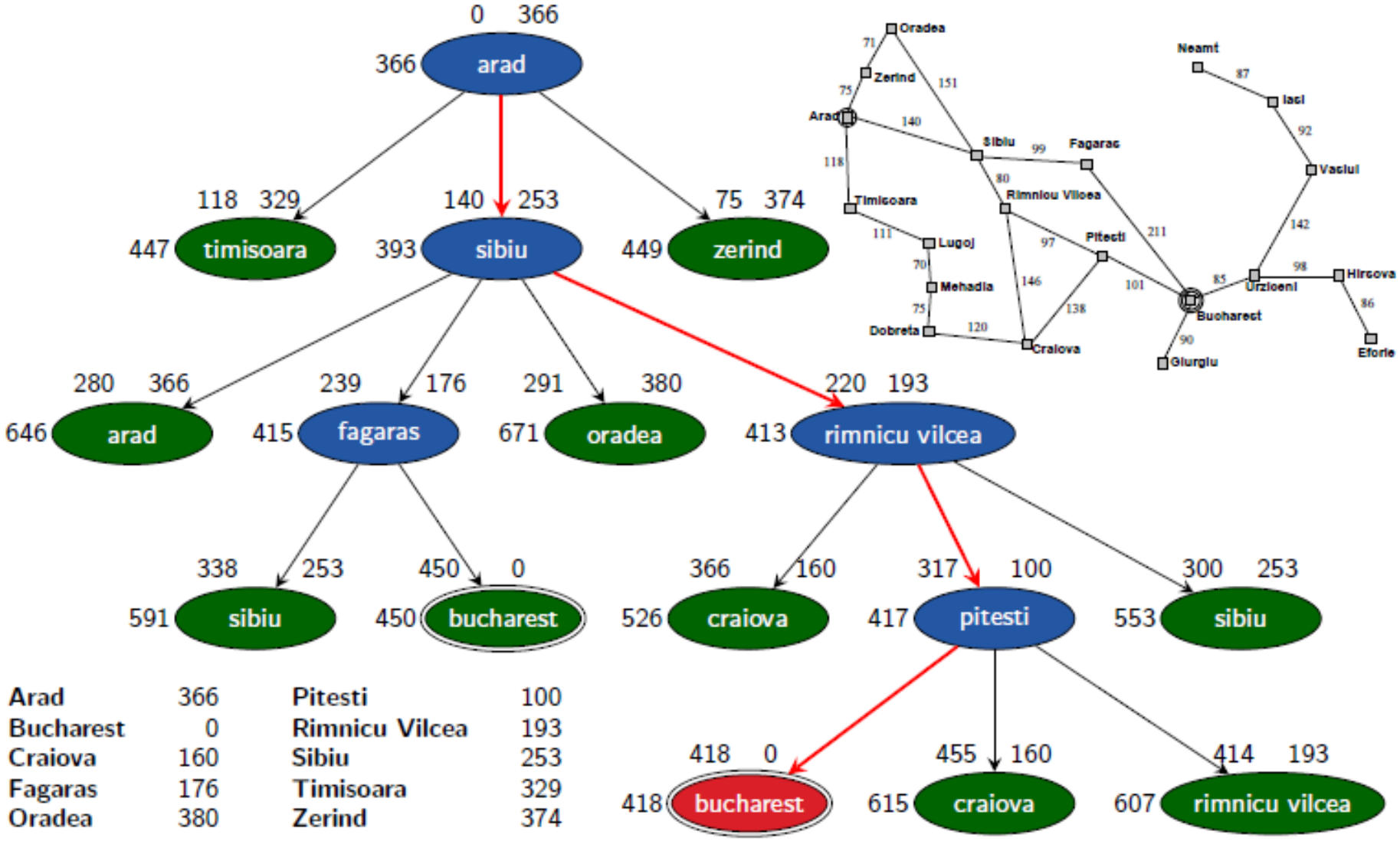
Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Introduction to AI

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning



Arad	366	Pitesti	100
Bucharest	0	Rimnicu Vilcea	193
Craiova	160	Sibiu	253
Fagaras	176	Timisoara	329
Oradea	380	Zerind	374

Informed search algorithms

Best First Search (A* Search)

Example: A* for Route Planning

- **Completeness:** complete with safe heuristics
- **Optimality:**
  with reopening: optimal with admissible heuristics
  without reopening: optimal with heuristics that are admissible and consistent
- Slower than Greedy Best First Search

Informed search algorithms

Best First Search (Weighted A* Search)

- $f(n) = h(n.\text{state})$: greedy best-first search
  $\rightsquigarrow$ only the heuristic counts
- $f(n) = g(n) + h(n.\text{state})$: A*
  $\rightsquigarrow$ combination of path cost and heuristic
- $f(n) = g(n) + w \cdot h(n.\text{state})$: weighted A*
  $w \in \mathbb{R}_0^+$ is a parameter
  $\rightsquigarrow$ interpolates between greedy best-first search and A*

Weighted A*

A* with more heavily weighted heuristic:
$f(n) = g(n) + w \cdot h(n.\text{state})$,
where weight $w \in \mathbb{R}_0^+$ with $w \geq 1$ is a freely choosable parameter

Note: $w < 1$ is conceivable, but usually not a good idea

Informed search algorithms

Weighted A* Search: Properties

weight parameter controls "greediness" of search:

- $w = 0$: like uniform cost search
- $w = 1$: like A*
- $w \to \infty$: like greedy best-first search

with $w \geq 1$ properties analogous to A*:

- h admissible:
  found solution guaranteed to be at most w times as expensive as optimum when reopening is used
- h admissible and consistent:
  found solution guaranteed to be at most w times as expensive as optimum; no reopening needed

Informed search algorithms

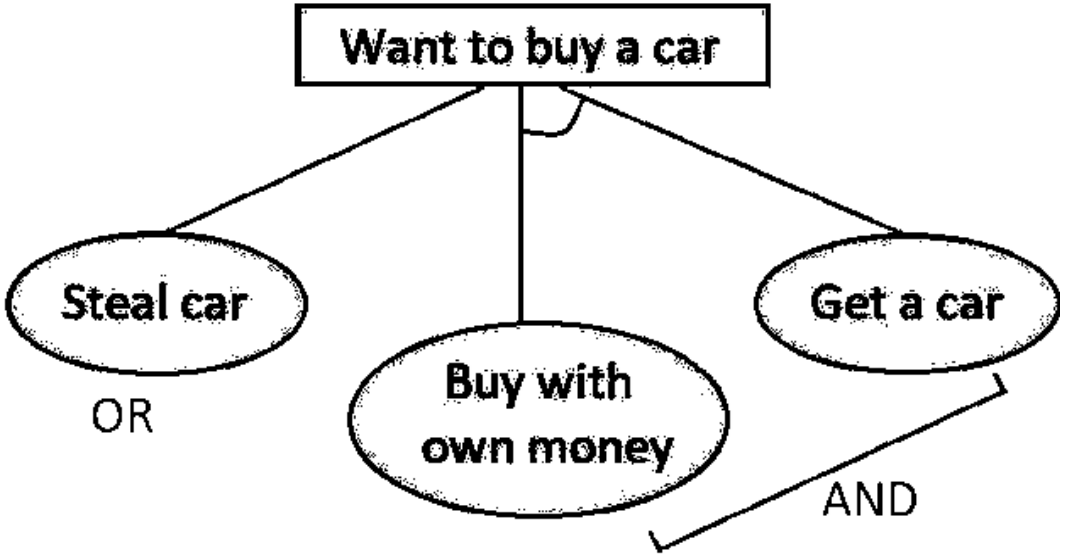
AO* Search

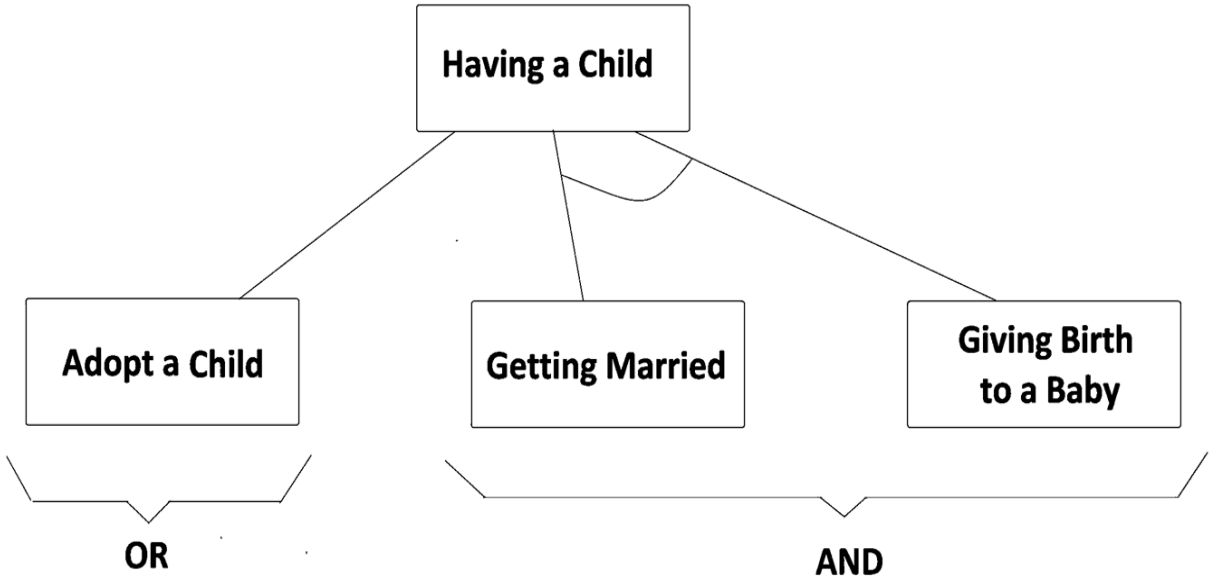
- AO* algorithm is a variant of best first search algorithm.
- AO* algorithm uses the concept of **AND-O**R graphs to decompose any complex problem given into smaller set of problems which are further solved.
- AND-OR graphs are specialized graphs that are used in problems that can be broken down into sub problems where AND side of the graph represent a set of task that need to be done to achieve the main goal , whereas the or side of the graph represent the different ways of performing task to achieve the same main goal.

Informed search algorithms

AO* Search

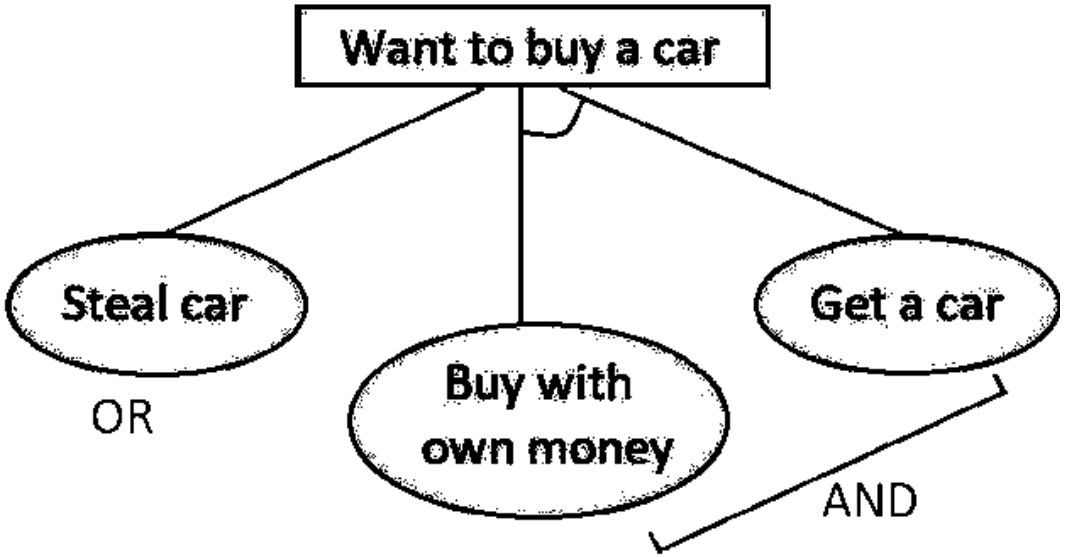
AND-OR graphs: It breaks complex problems into smaller ones and then solve them. The AND side of the graph represents those tasks that need to be done with each other to reach the goal, while the OR side stands alone for a single task. Let's see an example below:

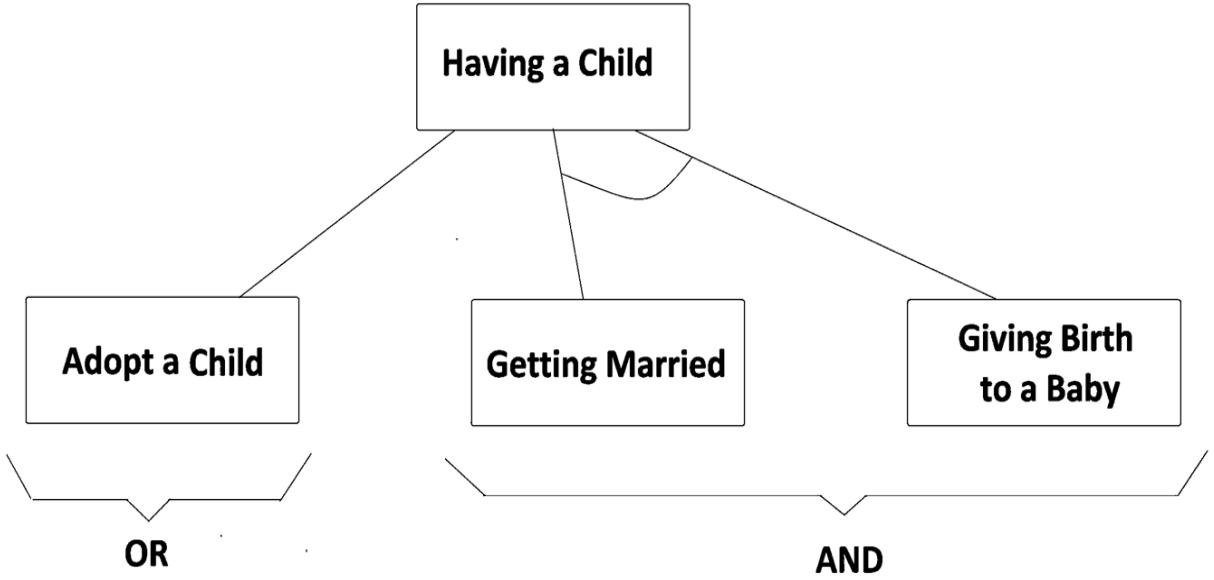




Informed search algorithms

AO* Search





- In the above examples, we can see both tasks of the AND side, which are connected by an AND-ARCS, need to be done to have a child, while the OR side lets having a child just by a single task of adoption.
- In general, in a graph for each node, there could be multiple OR and AND sides, where each AND side itself may have multiple successor nodes connected to each other by an AND-ARCS.

Informed search algorithms

AO* Search

AO* works based on this formula: **f(n) = g(n) + h(n)**

Where

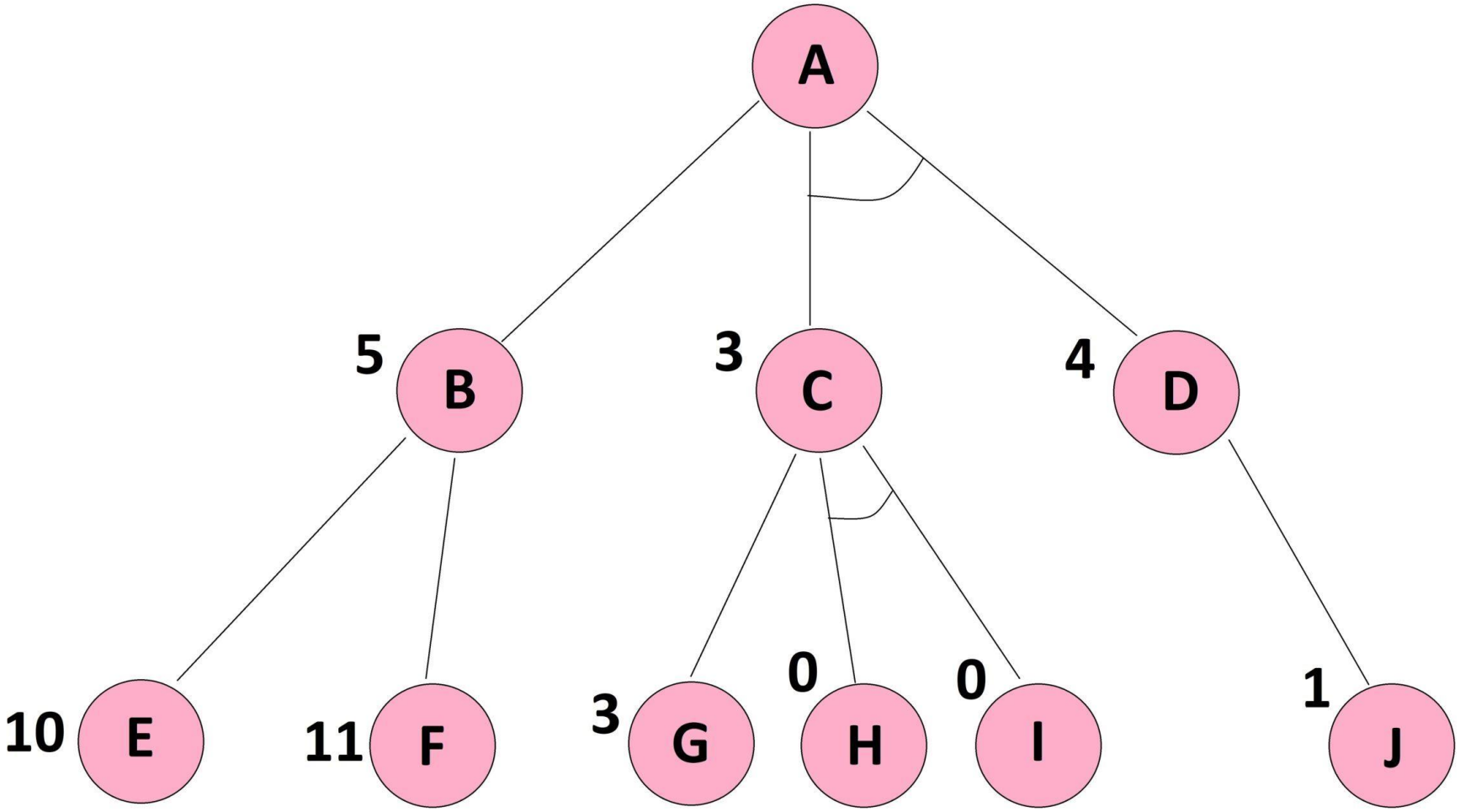
g(n) is the actual cost of going from the starting node to the current node,

h(n) is the estimated or heuristic cost of going from the current node to the goal node, and

f(n) is the actual cost of going from the starting node to the goal node.

Informed search algorithms

AO* Search



Informed search algorithms

AO* Search

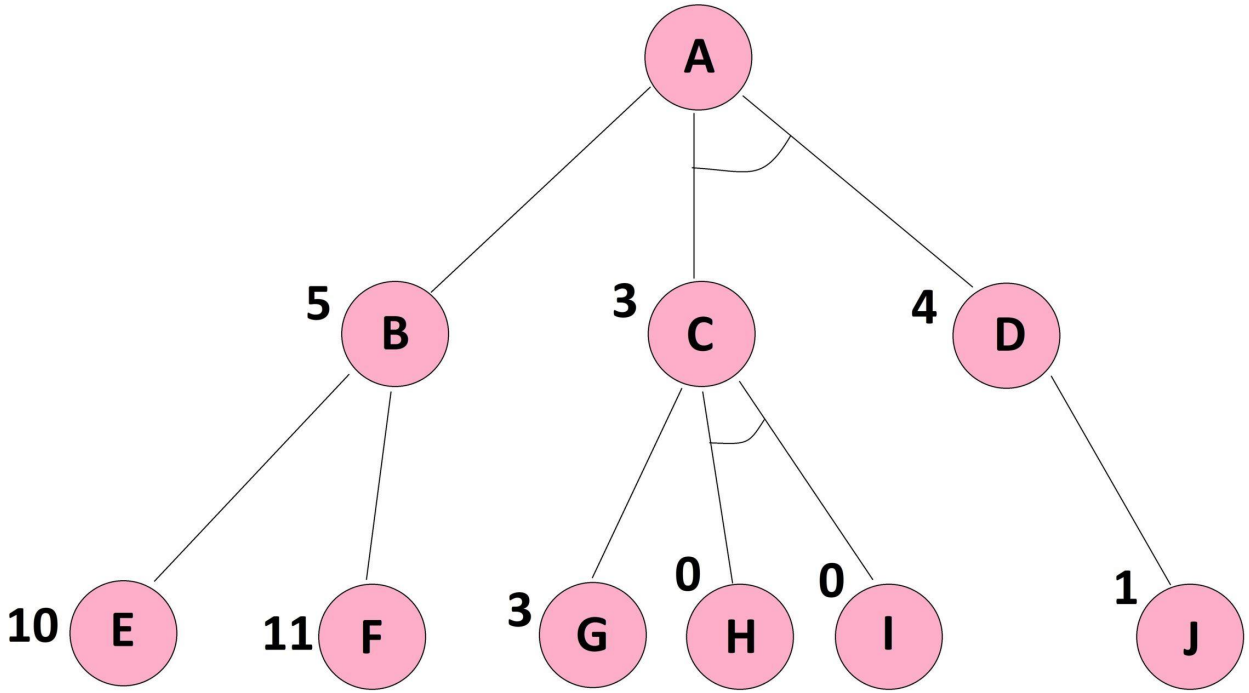
Forward Propagation

First, we begin from node A and calculate each of the OR side and AND side paths.

The OR side path f(A-B) = g(B) + h(B) = 1 + 5 = 6

Where 1 is the cost of the edge between A and B, and 5 is the estimated cost from B to goal node.

The AND side path f(A-C and A-D) = g(C) + f(C) + g(D) + g(D) = 1 + 3 + 1 + 4 = 9



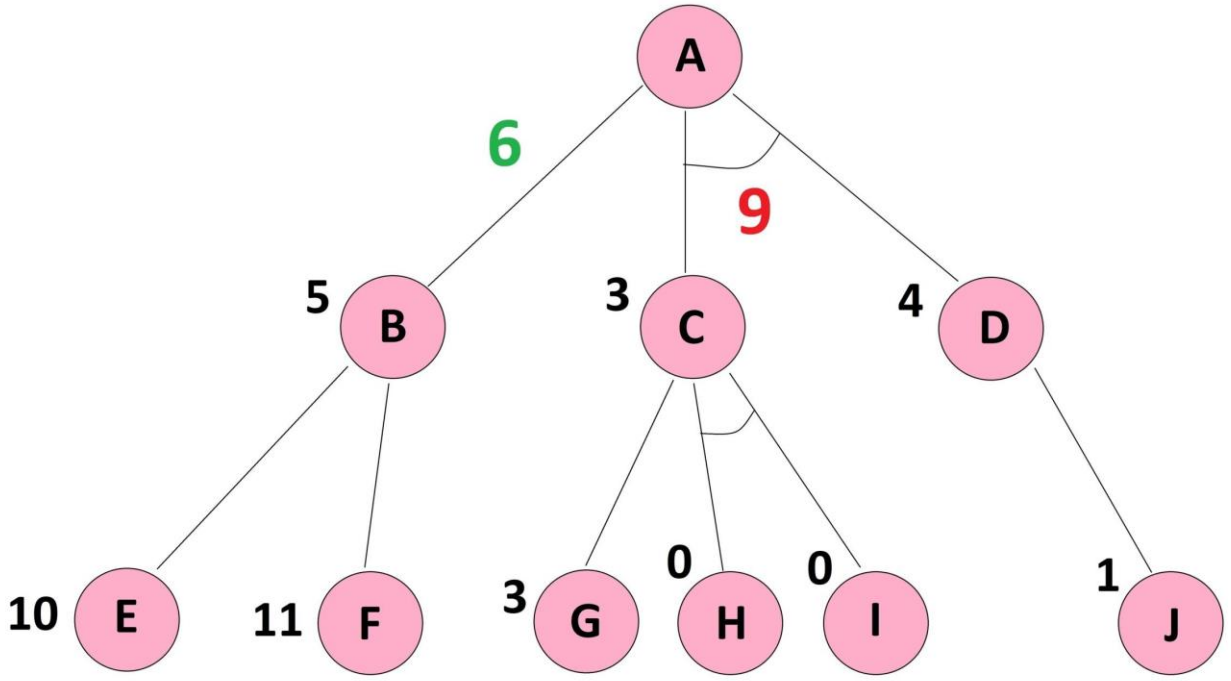
Informed search algorithms

AO* Search

Forward Propagation
First, we begin from node A and calculate each of the OR side and AND side paths.

The OR side path f(A-B) = g(B) + h(B) = 1 + 5 = 6
Where 1 is the cost of the edge between A and B, and 5 is the estimated cost from B to goal node.

The AND side path f(A-C and A-D) = g(C) + f(C) + g(D) + g(D) = 1 + 3 + 1 + 4 = 9



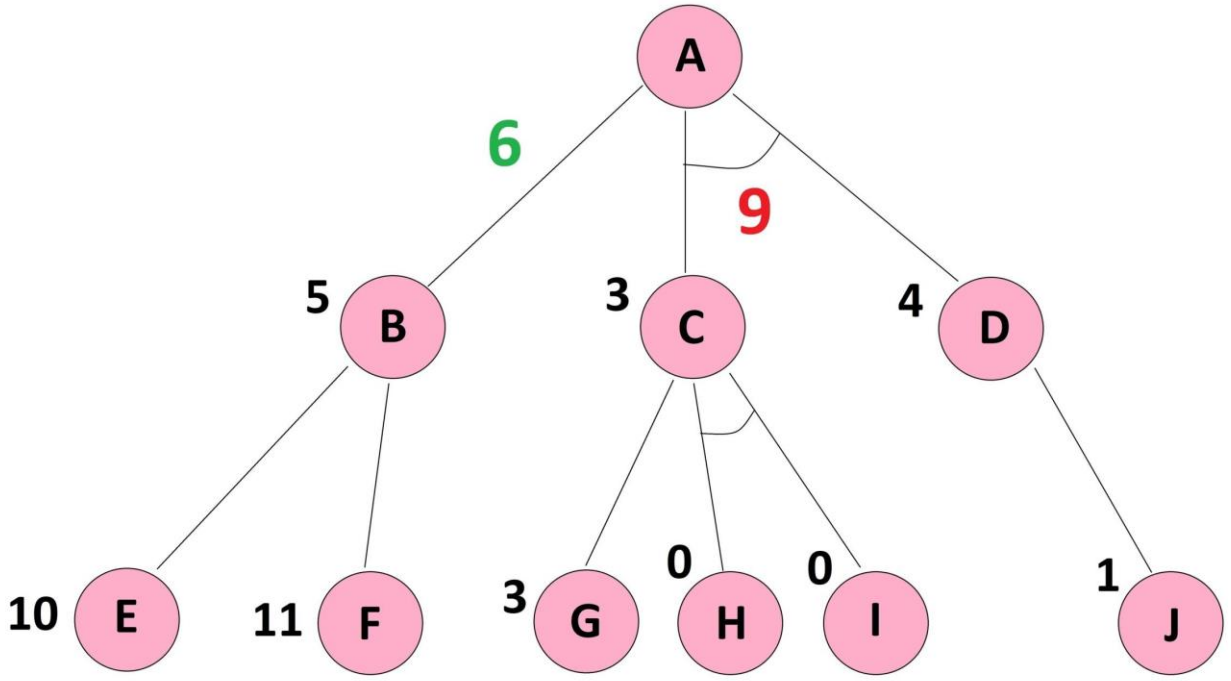
Informed search algorithms

AO* Search

Forward Propagation
First, we begin from node A and calculate each of the OR side and AND side paths.

The OR side path f(A-B) = g(B) + h(B) = 1 + 5 = 6
Where 1 is the cost of the edge between A and B, and 5 is the estimated cost from B to goal node.

The AND side path f(A-C and A-D) = g(C) + f(C) + g(D) + g(D) = 1 + 3 + 1 + 4 = 9



Since the cost of the path f(A-B) is the minimum cost path, we proceed on this path in the next step.

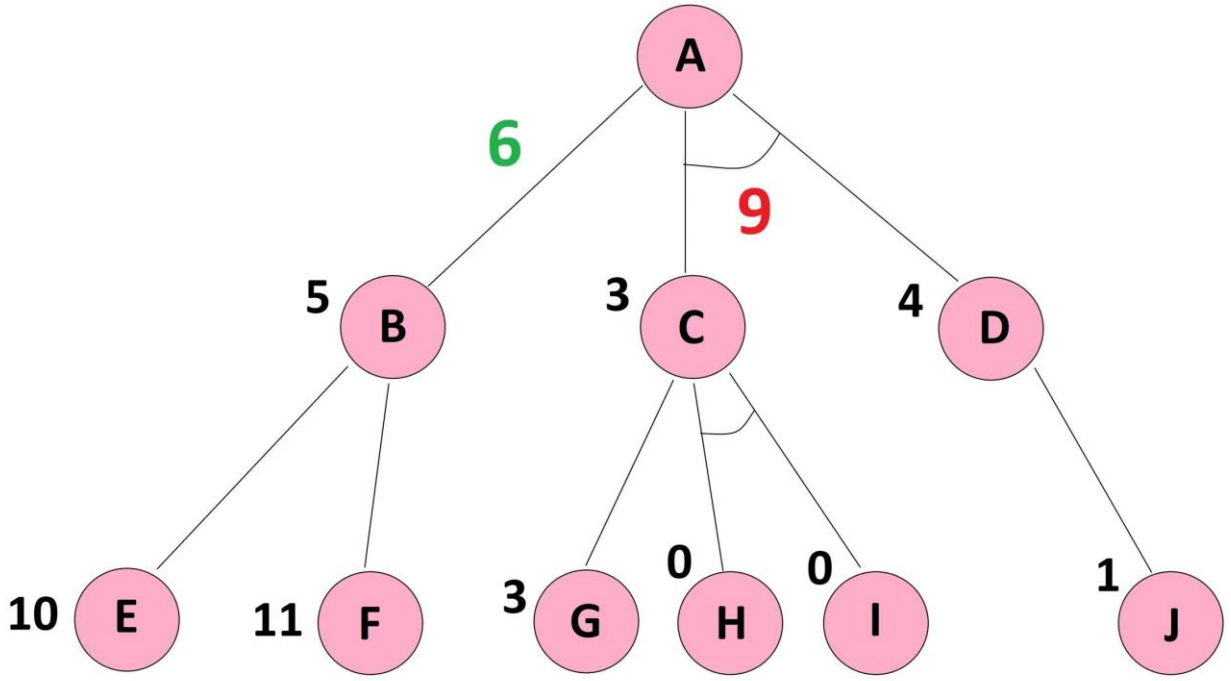
Informed search algorithms

AO* Search

Forward Propagation
First, we begin from node A and calculate each of the OR side and AND side paths.

The OR side path f(A-B) = g(B) + h(B) = 1 + 5 = 6
Where 1 is the cost of the edge between A and B, and 5 is the estimated cost from B to goal node.

The AND side path f(A-C and A-D) = g(C) + f(C) + g(D) + g(D) = 1 + 3 + 1 + 4 = 9



Since the cost of the path f(A-B) is the minimum cost path, we proceed on this path in the next step.

Note: It may not be the correct minimum cost path because we have made our calculations based on the heuristics down to only one level. However, the given graph has provided us with a deeper level whose calculations may update the achieved values.

Informed search algorithms

AO* Search

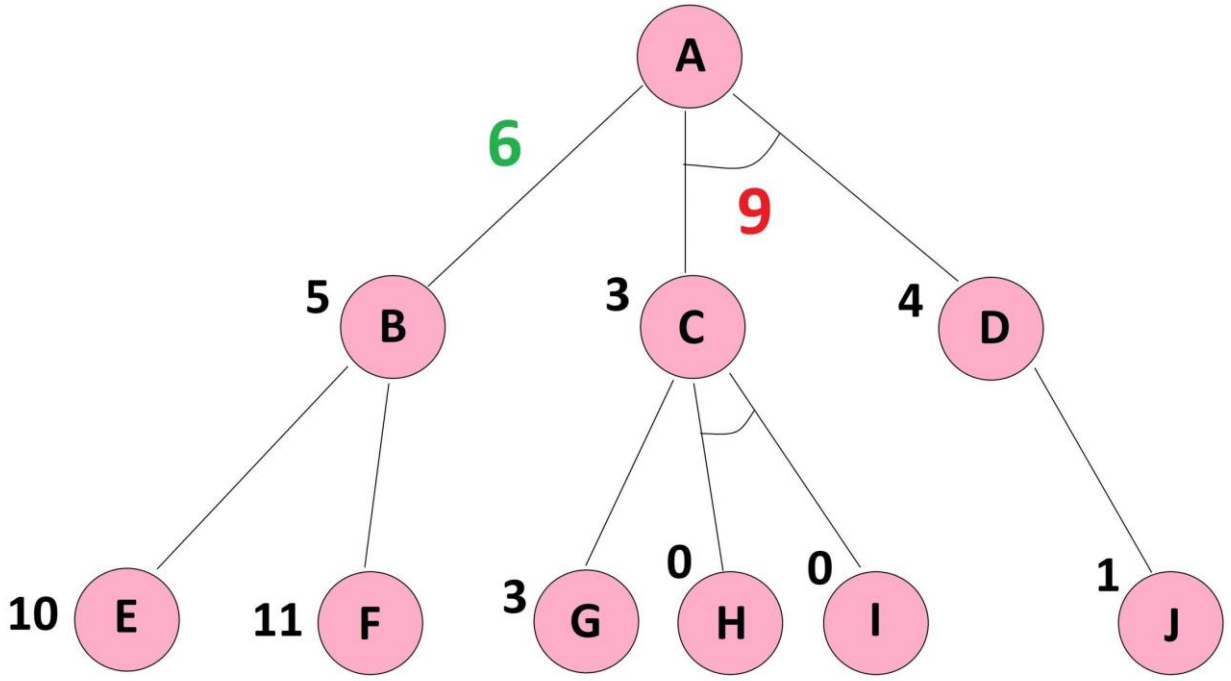
Reaching the Last Level and Back Propagation

In this step we continue on the f (A-B) and from B to to its successor nodes i.e., E and F, where

f(B-E) = g(E) + h(E) = 1 + 10 = 11

and f(B-F) = g(F) + h(F) = 1 + 11 = 12.

Here, f(B-E) has a lower cost and would be chosen. Now, we have reached the bottom of the graph where no more level is given to add to our information. Therefore, **we can do the backpropagation and correct the heuristics of upper levels**.



The updated heuristic will be h(B) = f(B-E) = 11, and as a consequence the updated heuristic h(A-B) = g(B) + update h(B) = 1 + 11 = 12

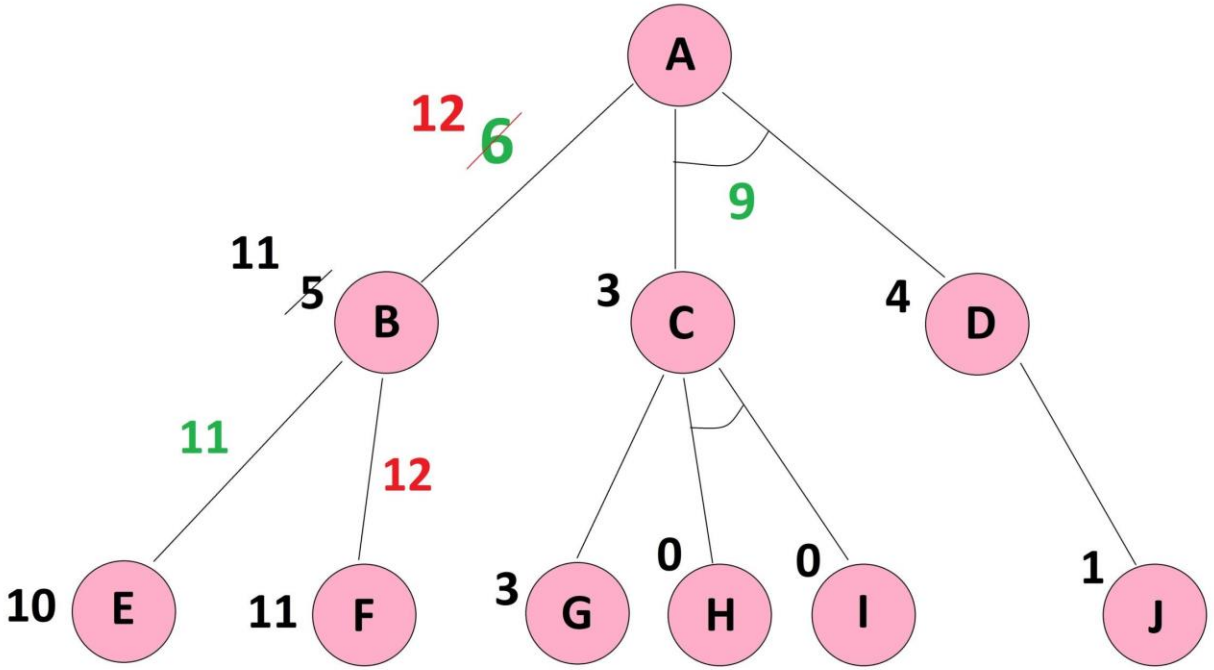
Informed search algorithms

AO* Search

Reaching the Last Level and Back Propagation

Now, we can see that f(A-C and A-D) with a cost of 9 is lower than the updated f(A-B) with a cost of 12.

Therefore, we need to proceed on this path A-C-D to find the minimum cost path from A to the goal node.



Informed search algorithms

AO* Search

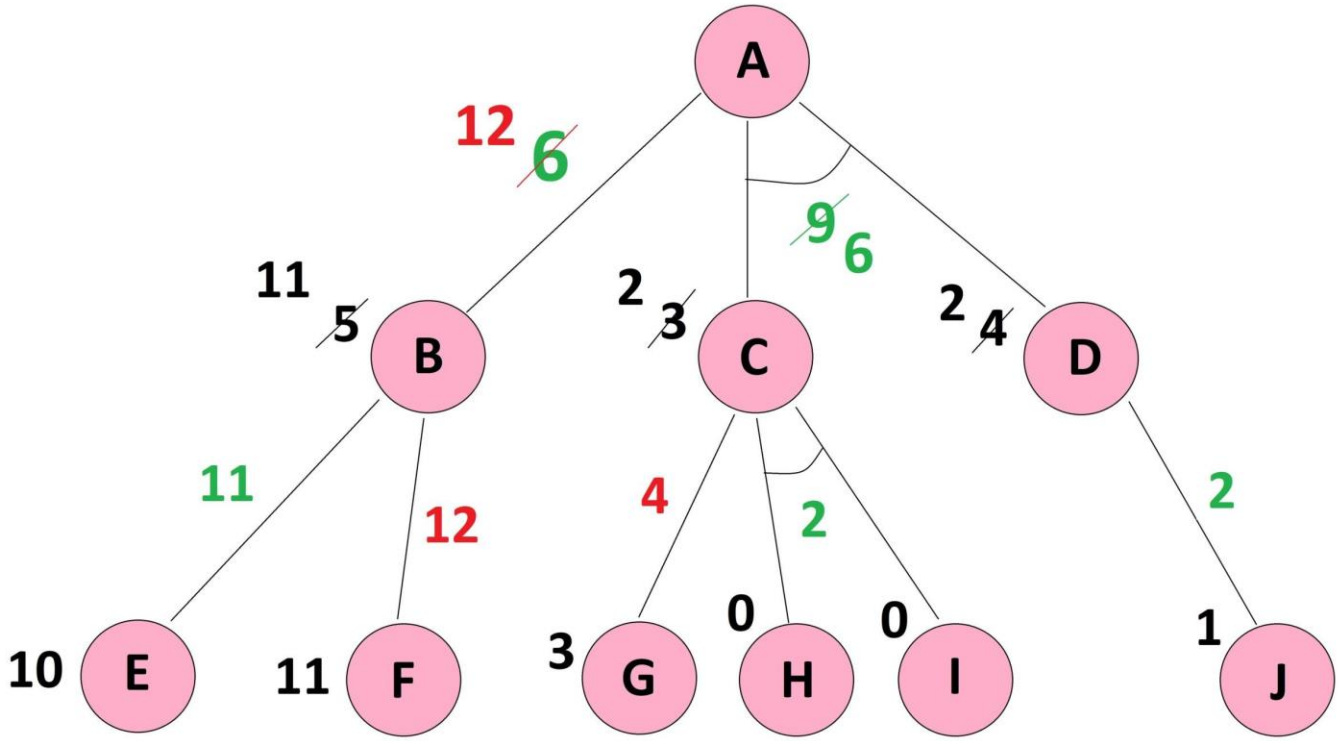
Correcting the Path from Start Node

In this step, we do the calculations for the AND side path, i.e., f(A-C and A-D) and first explore the paths attached to node C.

In this node again we have an OR side where f(C-G) = 1 + 3 = 4
and an AND side where
f(C-H and C-I) = 1 + 0 + 1 + 0 = 2

Therefore as a consequence the updated h(C) = 2
Also, the updated h(D) = 2

By these updated values for h(C) and h(D), the updated h(A-C and A-D) = 1 + 2 + 1 + 2 = 6



Informed search algorithms

AO* Search

Difference between A* and AO*

- A* algorithm is an OR Graph Algorithm while AO* is an AND-OR Graph Algorithm.
- A* algorithm cost function include f = g + h while AO* algorithm cost function is simply f’ = g + h’
- Unlike A*, AO* algorithm does not always give minimum cost.

Informed search algorithms

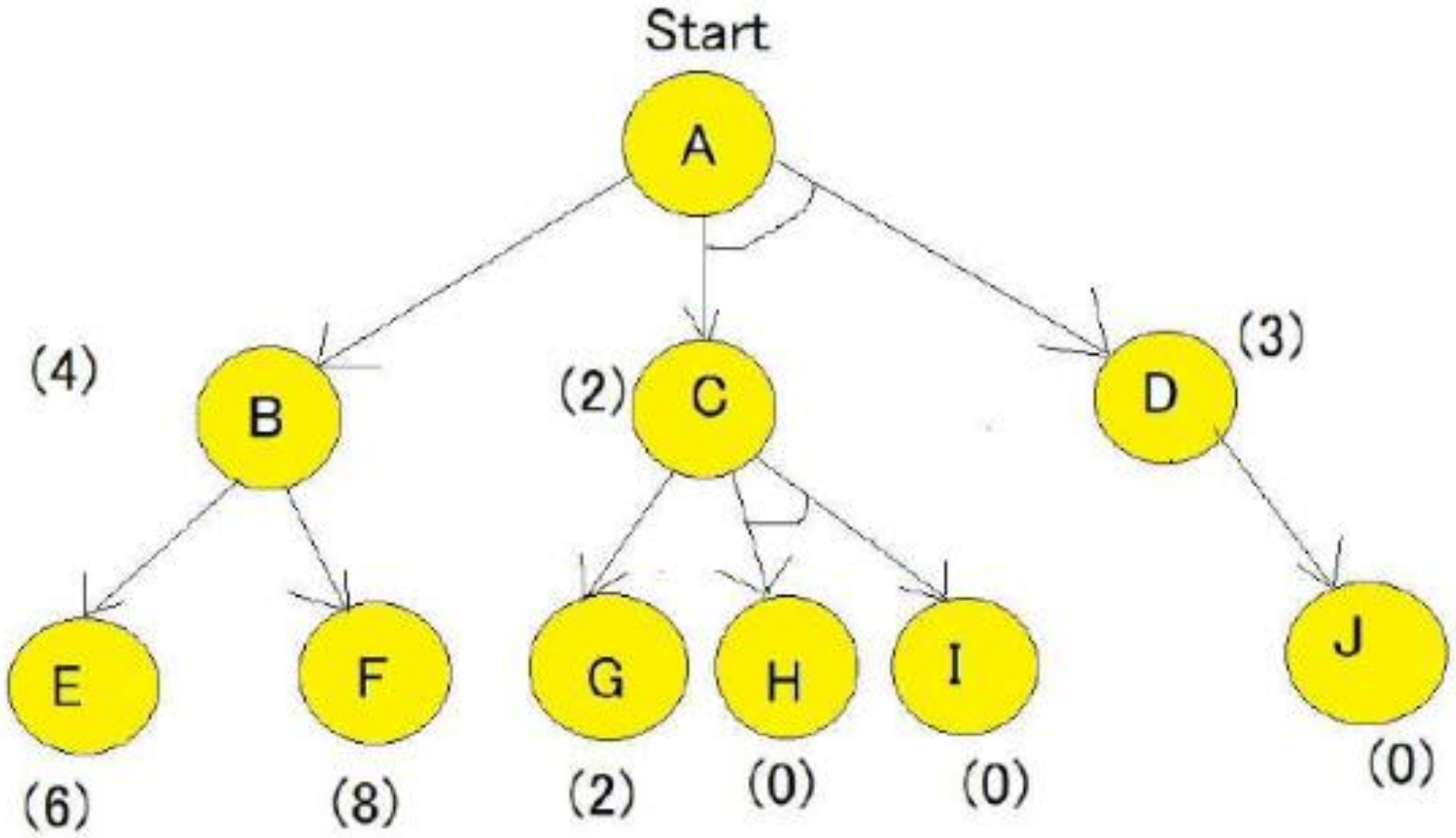
AO* Search

- **Completeness:** complete with safe heuristics
- **Optimality:**
  Optimal with admissible heuristics, otherwise semi-optimal
- Slower than Greedy Best First Search

Informed search algorithms

AO* Search

Assignment 1

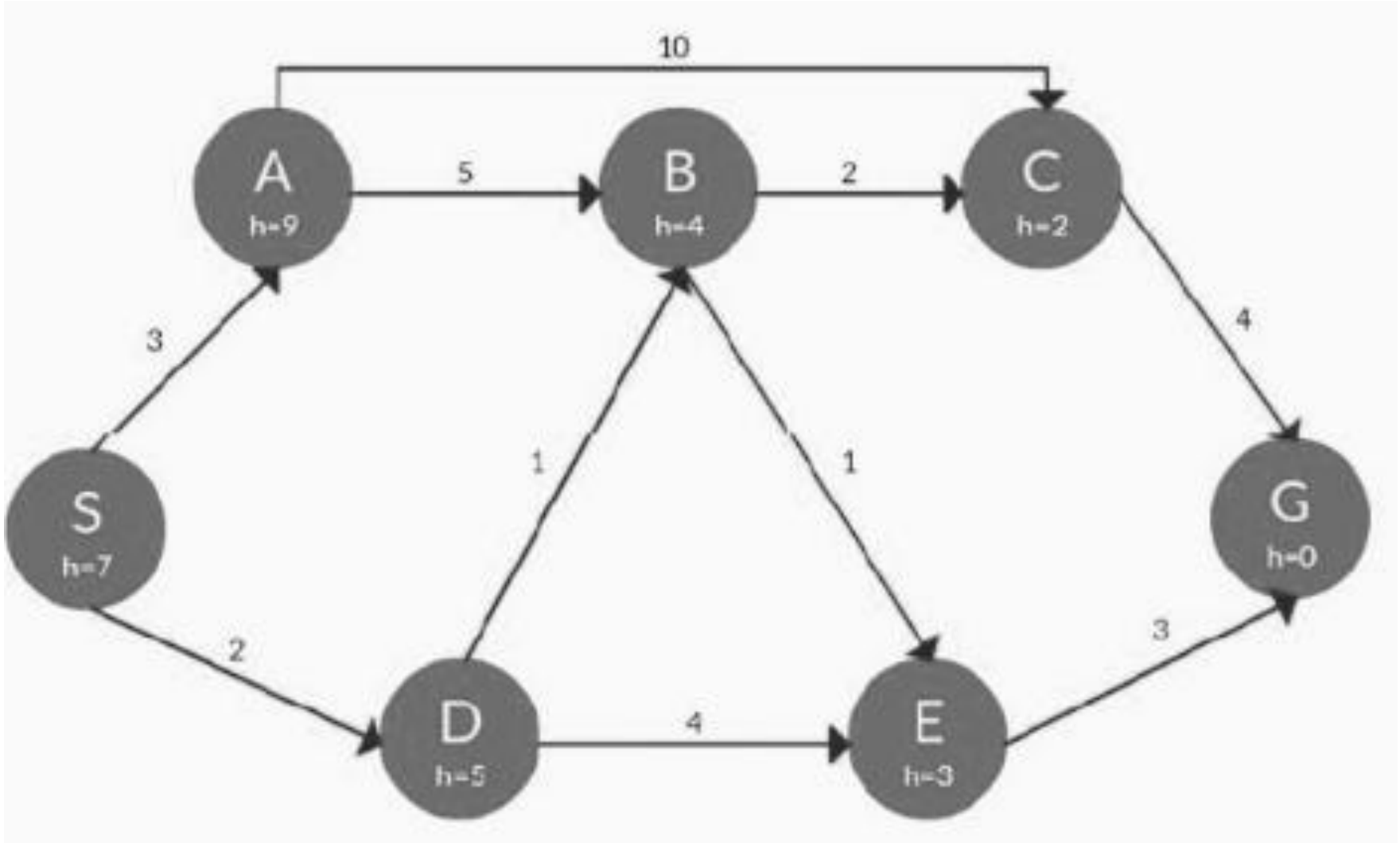


Informed search algorithms

AO* Search

Assignment 2

Use AO* technique to find paths and cost from S to G in the following graph.



Informed search algorithms

Hill Climbing Search

- Local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

- Greedy local search as it only looks to its good immediate neighbor state and not beyond that.

- A node of hill climbing algorithm has two components which are **state** and **heuristic** value.

- Hill Climbing is mostly used when a good heuristic is available.

Informed search algorithms

Hill Climbing Search

A hill-climbing algorithm has four main features:

- **Greedy approach:** This means that it moves in a direction in which the cost function is optimized.
- **No Backtracking:** A hill-climbing algorithm only works on the current state and succeeding states (future). It does not look at the previous states.
- **Feedback mechanism:** The algorithm has a feedback mechanism that helps it decide on the direction of movement (direction on the hill). The feedback mechanism is enhanced through the generate-and-test technique.
- **Incremental change:** The algorithm improves the current solution by incremental changes.

Informed search algorithms

Hill Climbing Search

State-space diagram analysis:

- A state-space diagram provides a graphical representation of states and the optimization function. If the objective function is the y-axis, we aim to establish the local maximum and global maximum.

- If the cost function represents this axis, we aim to establish the local optimal solution (maximum/minimum) and global optimal solution (maximum/minimum).

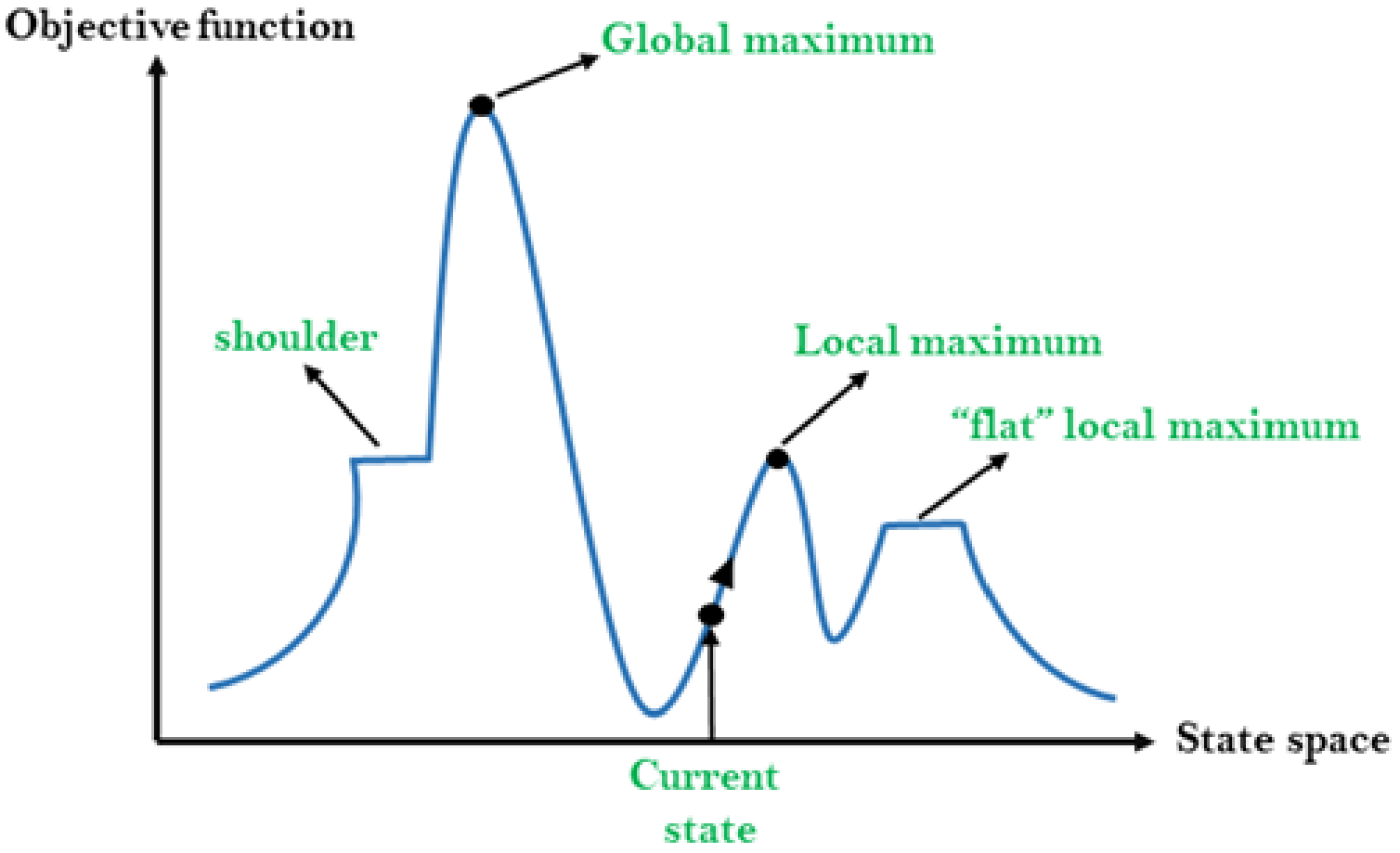
- The following diagram shows a simple state-space diagram. The objective function has been shown on the y-axis, while the state-space represents the x-axis.

Introduction to AI

Informed search algorithms

Hill Climbing Search

State-space diagram analysis:



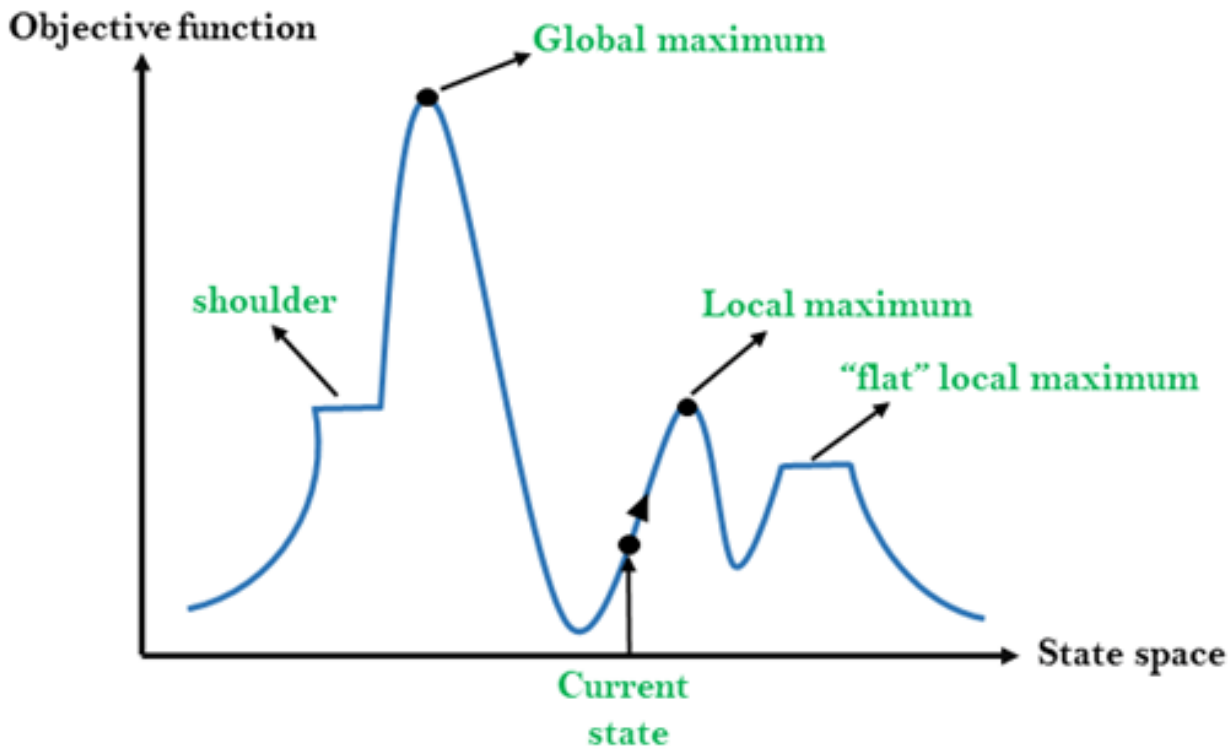
Informed search algorithms

Hill Climbing Search

State-space diagram analysis:

A state-space diagram consists of various regions that can be explained as follows:

- **Local maximum:** A local maximum is a solution that surpasses other neighboring solutions or states but is not the best possible solution.
- **Global maximum:** This is the best possible solution achieved by the algorithm.
- **Current state:** This is the existing or present state.
- **Flat local maximum:** This is a flat region where the neighboring solutions attain the same value.
- **Shoulder:** This is a plateau whose edge is stretching upwards.



Informed search algorithms

Hill Climbing Search

Problems with hill climbing:

There are three regions in which a hill-climbing algorithm cannot attain a global maximum or the optimal solution:

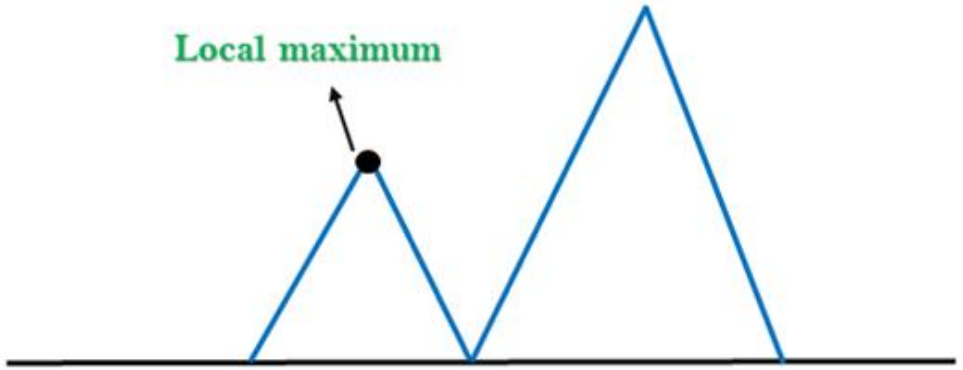
- **Local Optimal Solution**
- **Ridge**
- **Plateau**

Informed search algorithms

Hill Climbing Search

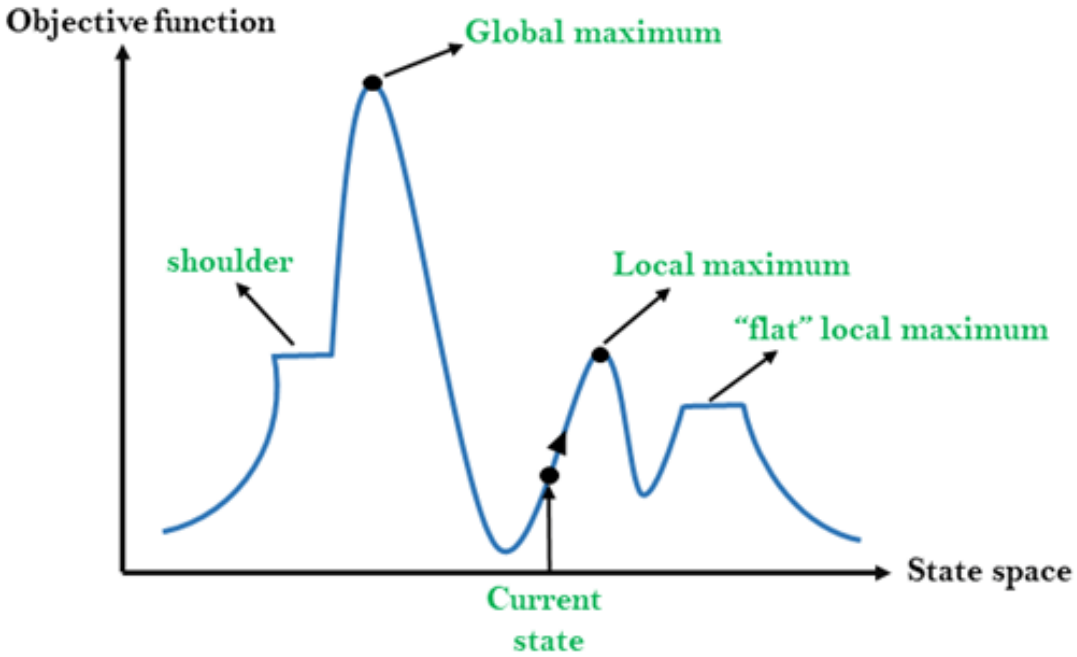
Problems with hill climbing:

Local Optimal Solution



At this point, the neighboring states have lower values than the current state. The greedy approach feature will not allow the algorithm to see other region. This will lead to the hill-climbing process's termination, even though this is not the best possible solution.

This problem can be solved using momentum. This technique adds a certain proportion (m) of the initial weight to the current one. m is a value between 0 and 1. Momentum enables the hill-climbing algorithm to take huge steps that will make it move past the local maximum.



Informed search algorithms

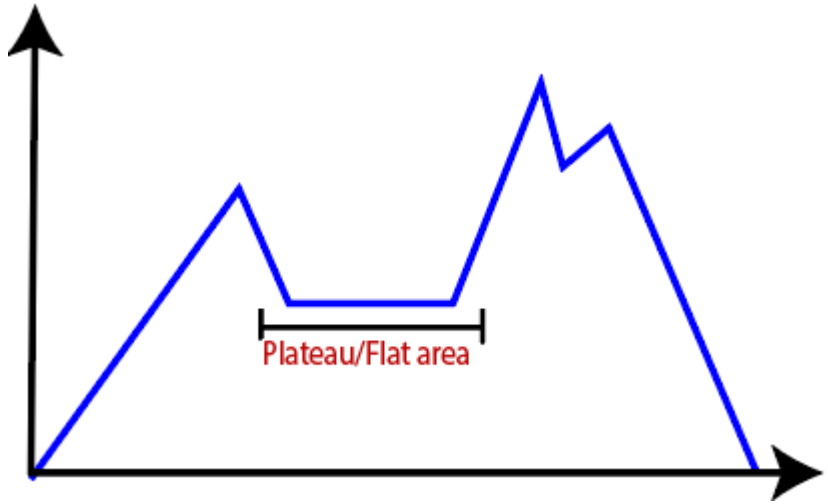
Hill Climbing Search

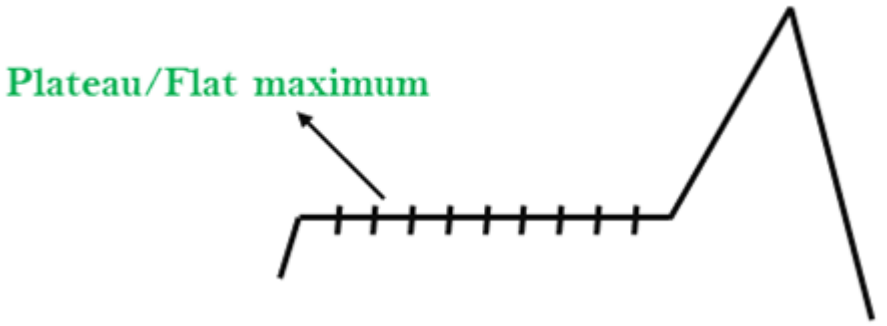
Problems with hill climbing:

Plateau

In this region, the values attained by the neighboring states are the same. This makes it difficult for the algorithm to choose the best direction.

This challenge can be overcome by taking a huge jump that will lead you to a non-plateau space.





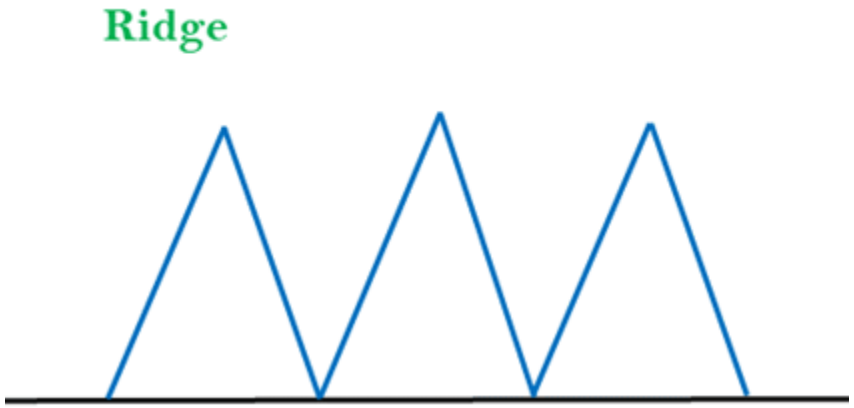
Informed search algorithms

Hill Climbing Search

Problems with hill climbing:

Ridge

A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.



This is because the peak of the ridge is followed by downward movement rather than upward movement.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

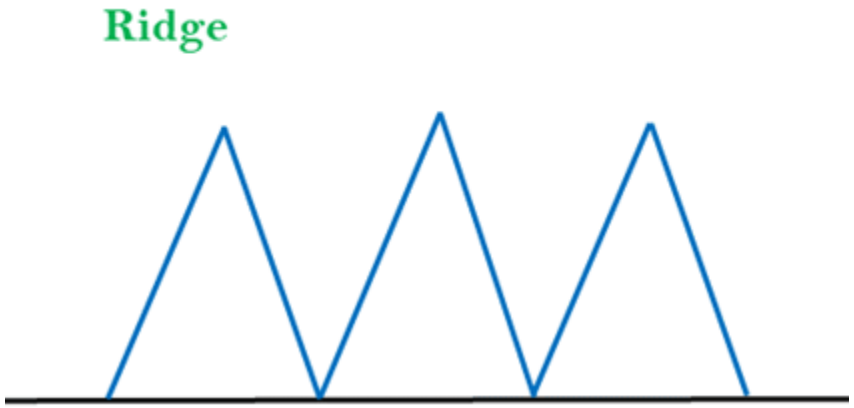
Informed search algorithms

Hill Climbing Search

Problems with hill climbing:

Ridge

A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.



This is because the peak of the ridge is followed by downward movement rather than upward movement.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Informed search algorithms

Types of Hill Climbing Search

- Simple hill climbing
- Steepest-Ascent hill climbing
- Stochastic hill climbing

Informed search algorithms

Types of Hill Climbing Search

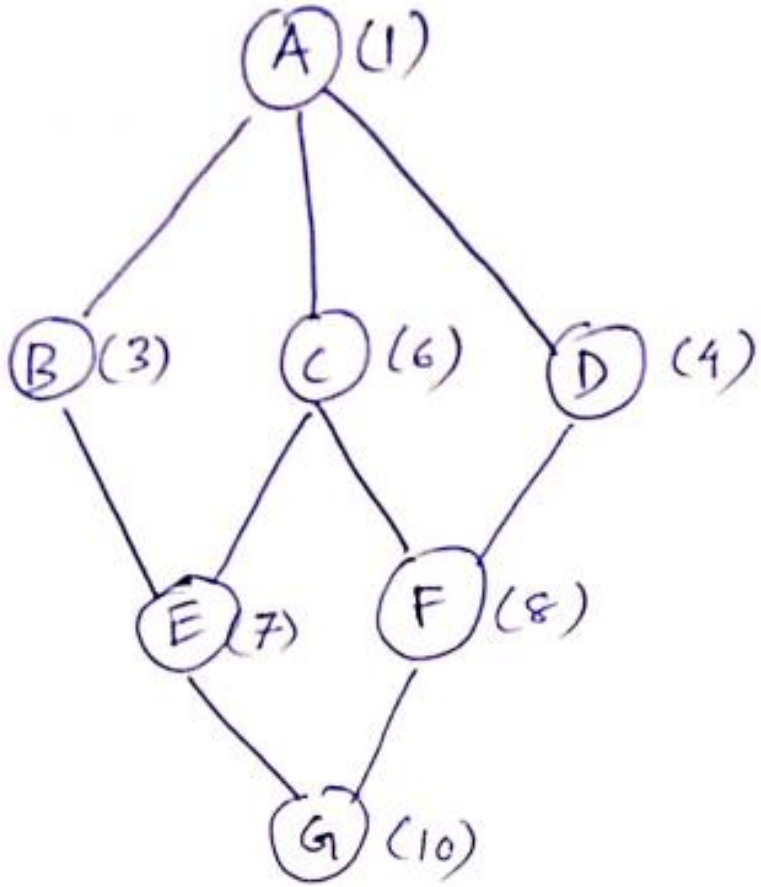
1. Simple hill climbing

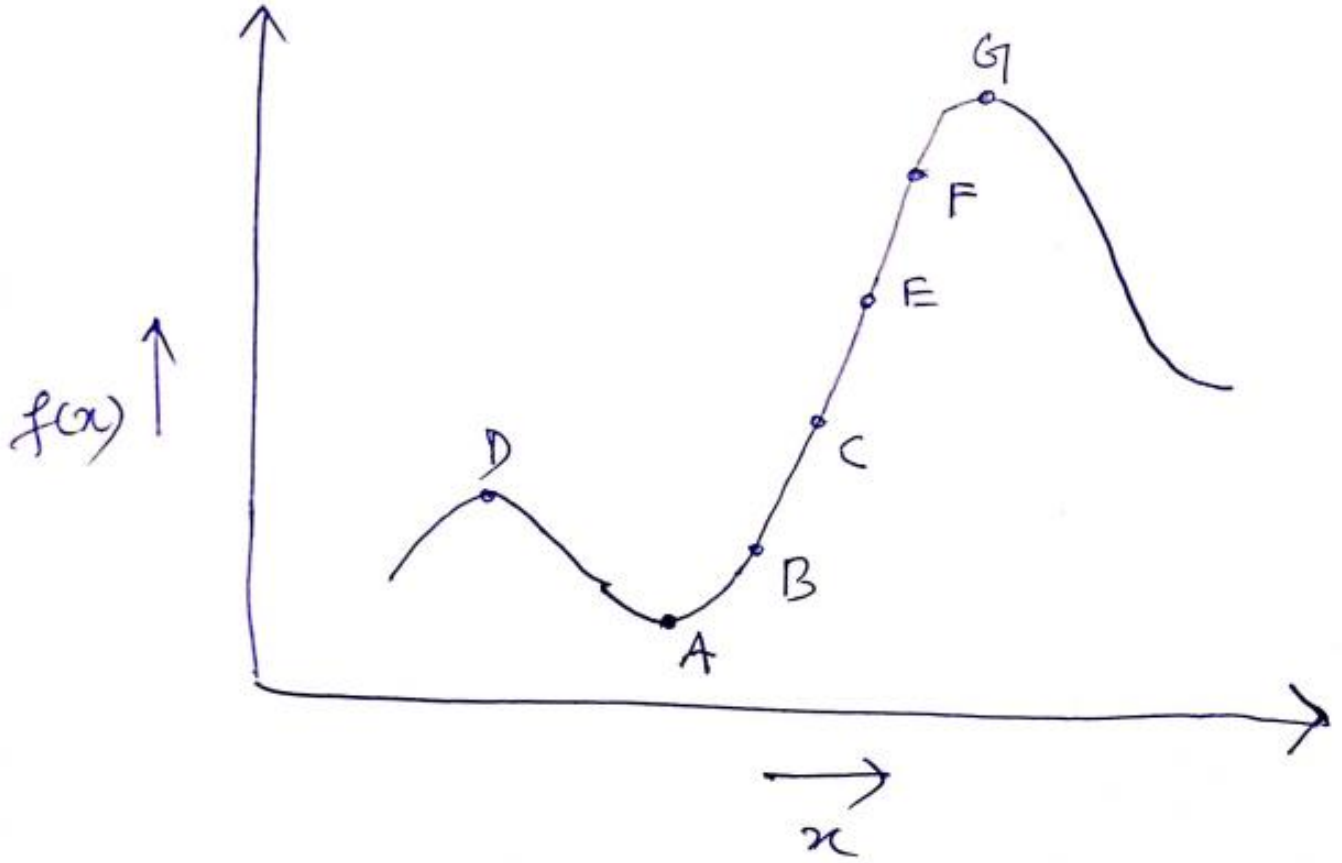
Step 1. Conduct an assessment of the current state. Stop the process and indicate success if it is a goal state.
Step 2. Perform looping on the current state if the assessment in step 1 did not establish a goal state.
Step 3. Continue looping to attain a new solution.
Step 4. Evalutae the new solution. If the new state has a higher value than the current state in steps 1 and 2, then mark it as a current state.
Step 5. Continue steps 1 to 4 until a goal state is attained. If this is the case, then exit the process.

Informed search algorithms

Types of Hill Climbing Search

1. Simple hill climbing





Informed search algorithms

Types of Hill Climbing Search

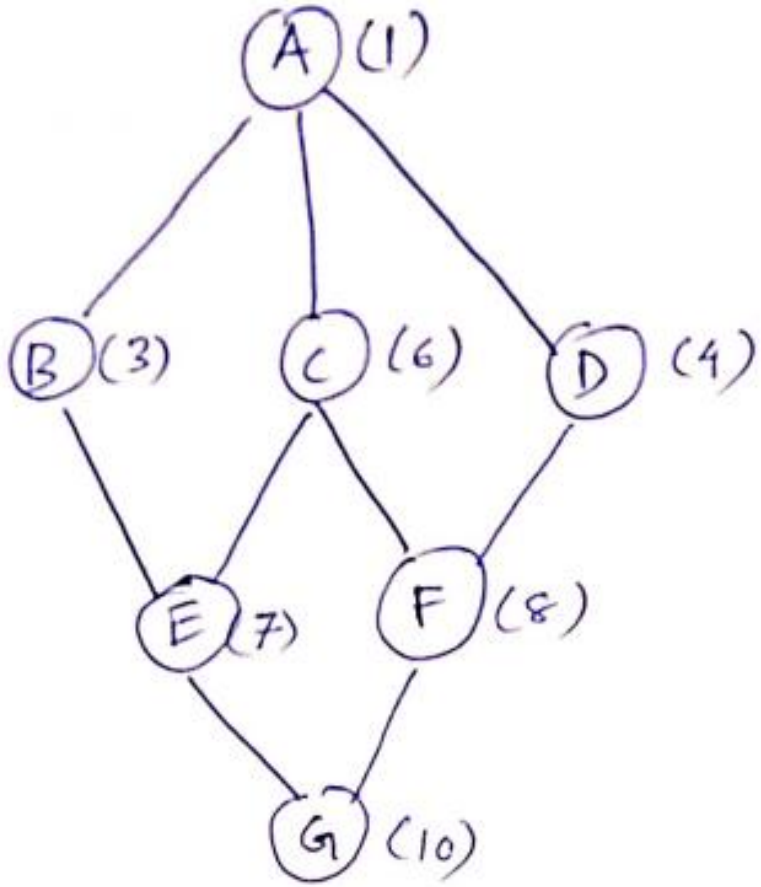
2. Steepest-Ascent hill climbing

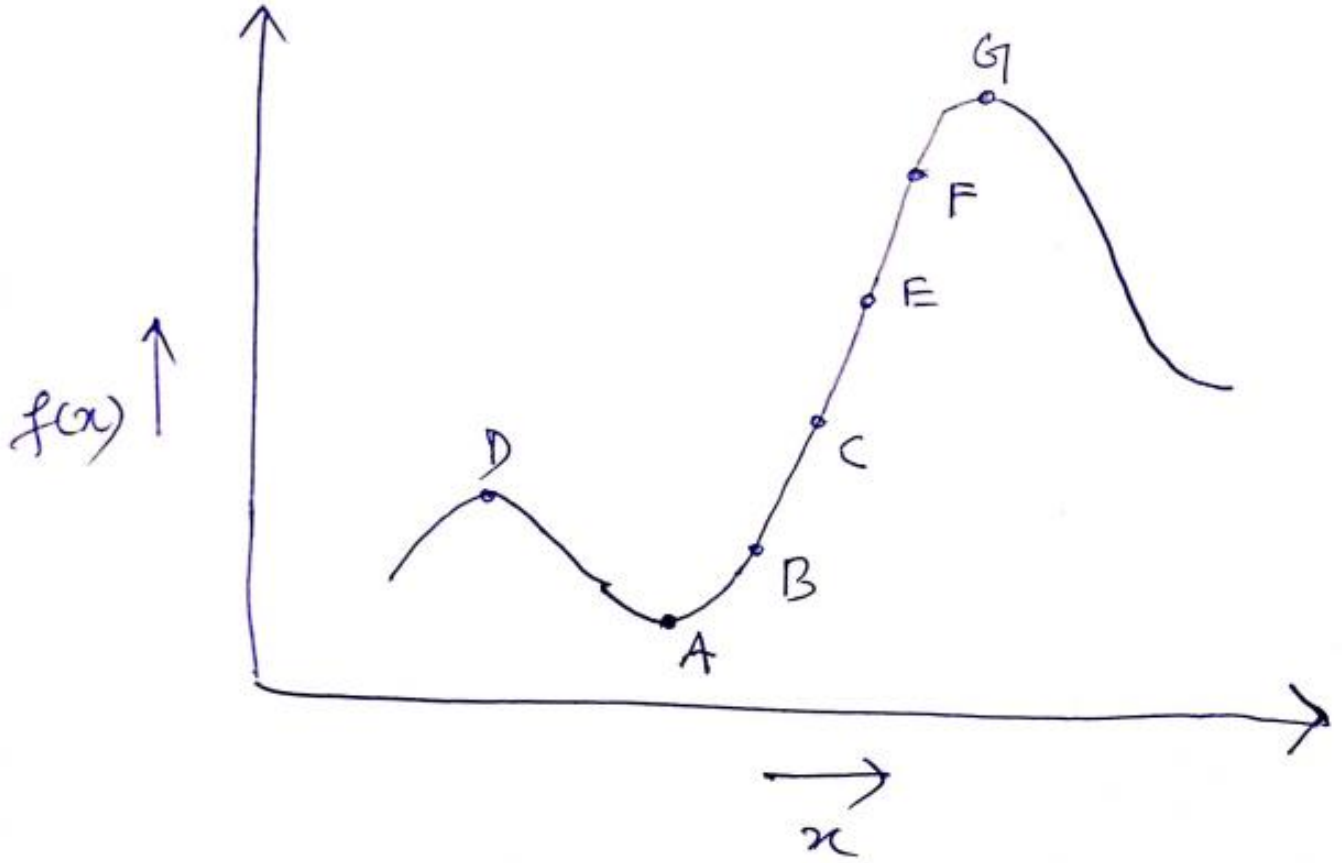
Step 1. Conduct an assessment of the current state. Stop the process and indicate success if it is a goal state.
Step 2. Perform looping on the current state if the assessment in step 1 did not establish a goal state.
Step 3. Continue looping to attain a new solution.
Step 4. Establish or set a state (X) such that current state successors have higher values than it.
Step 5. Run the new operator and produce a new solution.
Step 6. Evaluate this solution to check whether it is a goal state. If this is the goal state, then exit the program. Otherwise, compare it with the state (X).
Step 7. If the new state has a higher value than X, then update X = new state.
If X has a higher value than the current state then set current state = X.

Informed search algorithms

Types of Hill Climbing Search

2. Steepest-Ascent hill climbing





Informed search algorithms

Types of Hill Climbing Search

3. Stochastic hill climbing

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Informed search algorithms

Simulated Annealing

- Simulated Annealing (SA) is motivated by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis etc al in 1953 [Metropolis, 1953).

- The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing. If you heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling.

- If the liquid is cooled slowly enough, large crystals will be formed. However, if the liquid is cooled quickly (quenched) the crystals will contain imperfections. Metropolis's algorithm simulated the material as a system of particles.

- The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, frozen state

Informed search algorithms

Simulated Annealing

- In 1982, Kirkpatrick et al (Kirkpatrick, 1983) took the idea of the Metropolis algorithm and applied it to optimisation problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution

Informed search algorithms

Simulated Annealing

Simulated Annealing versus **Hill Climbing**

As we have seen in previous lectures, hill climbing suffers from problems in getting stuck at local minima (or maxima). We could try to overcome these problems by trying various techniques.

- We could try a hill climbing algorithm using different starting points.
- We could increase the size of the neighbourhood so that we consider more of the search space at each move.

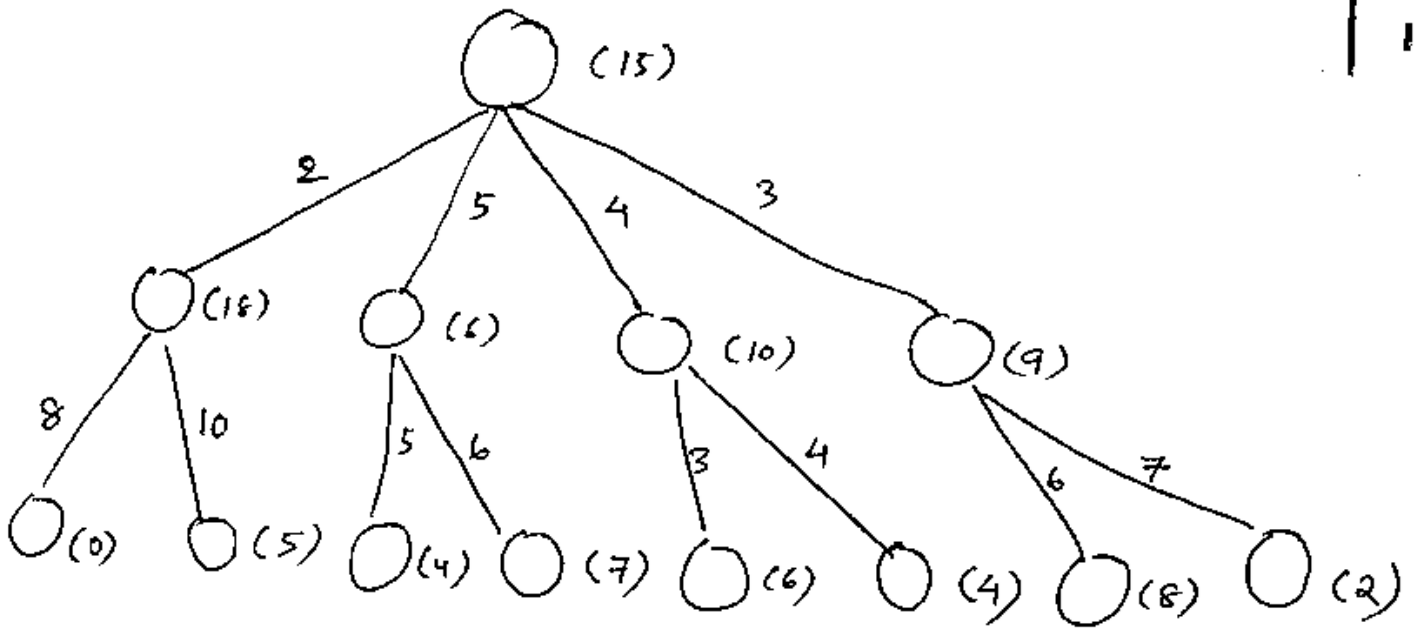
For example, we could try 3-opt, rather than a 2-opt move when implementing the TSP. Unfortunately, neither of these have proved satisfactory in practice when using a simple hill climbing algorithm. Simulated annealing solves this problem by allowing worse moves (lesser quality) to be taken some of the time. That is, it allows some uphill steps so that it can escape from local minima.

Informed search algorithms

Simulated Annealing

Simulated Annealing versus **Hill Climbing**

Unlike hill climbing, simulated annealing chooses a random move from the neighbourhood (recall that hill climbing chooses the best move from all those available – at least when using steepest descent (or ascent)). If the move is better than its current position then simulated annealing will always take it. If the move is worse (i.e. lesser quality) then it will be accepted based on some probability.

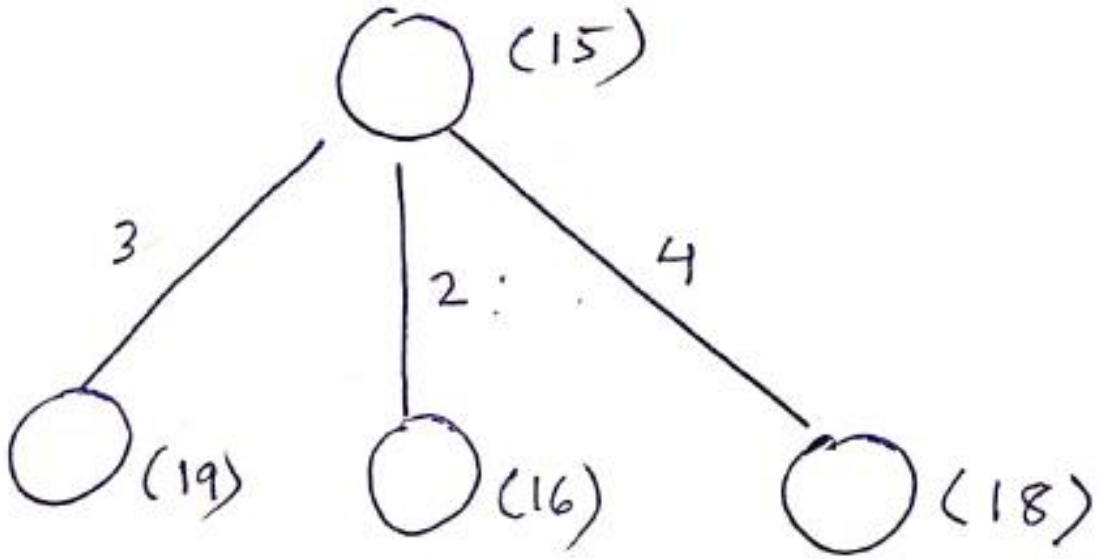


Informed search algorithms

Simulated Annealing

Simulated Annealing versus Hill Climbing

Unlike hill climbing, simulated annealing chooses a random move from the neighbourhood (recall that hill climbing chooses the best move from all those available – at least when using steepest descent (or ascent)). If the move is better than its current position then simulated annealing will always take it. If the move is worse (i.e. lesser quality) then it will be accepted based on some probability.



Informed search algorithms

Simulated Annealing

Acceptance Criteria

The law of thermodynamics state that at temperature, t, the probability of an increase in energy of magnitude, δE, is given by

$$P(\delta E) = \exp(-\delta E / kt)$$

Where k is a constant known as Boltzmann's constant and t is the present temperature

The simulation in the Metropolis algorithm calculates the new energy of the system. If the energy has decreased then the system moves to this state. If the energy has increased then the new state is accepted using the probability returned by the above formula. A certain number of iterations are carried out at each temperature and then the temperature is decreased. This is repeated until the system freezes into a steady state.

Informed search algorithms

Simulated Annealing

Acceptance Criteria

The law of thermodynamics state that at temperature, t, the probability of an increase in energy of magnitude, δE, is given by

$$P(\delta E) = \exp(-\delta E / kt)$$

Where k is a constant known as Boltzmann's constant and t is the present temperature

This equation is directly used in **simulated annealing**, although it is usual to drop the Boltzmann constant as this was only introduced into the equation to cope with different materials. Therefore, the probability of accepting a worse state is given by the equation

$$P = \exp(-c/t) > r$$

Where c = the change in the evaluation function, t = the current temperature, r = a random number between 0 and 1

Informed search algorithms

Simulated Annealing

Acceptance Criteria

In **simulated annealing**, the probability of accepting a worse state is given by the equation

$$P = \exp(-c/t) > r$$

Where c = the change in the evaluation function, t = the current temperature, r = a random number between 0 and 1

The probability of accepting a worse move is a function of both the temperature of the system and of the change in the cost function.

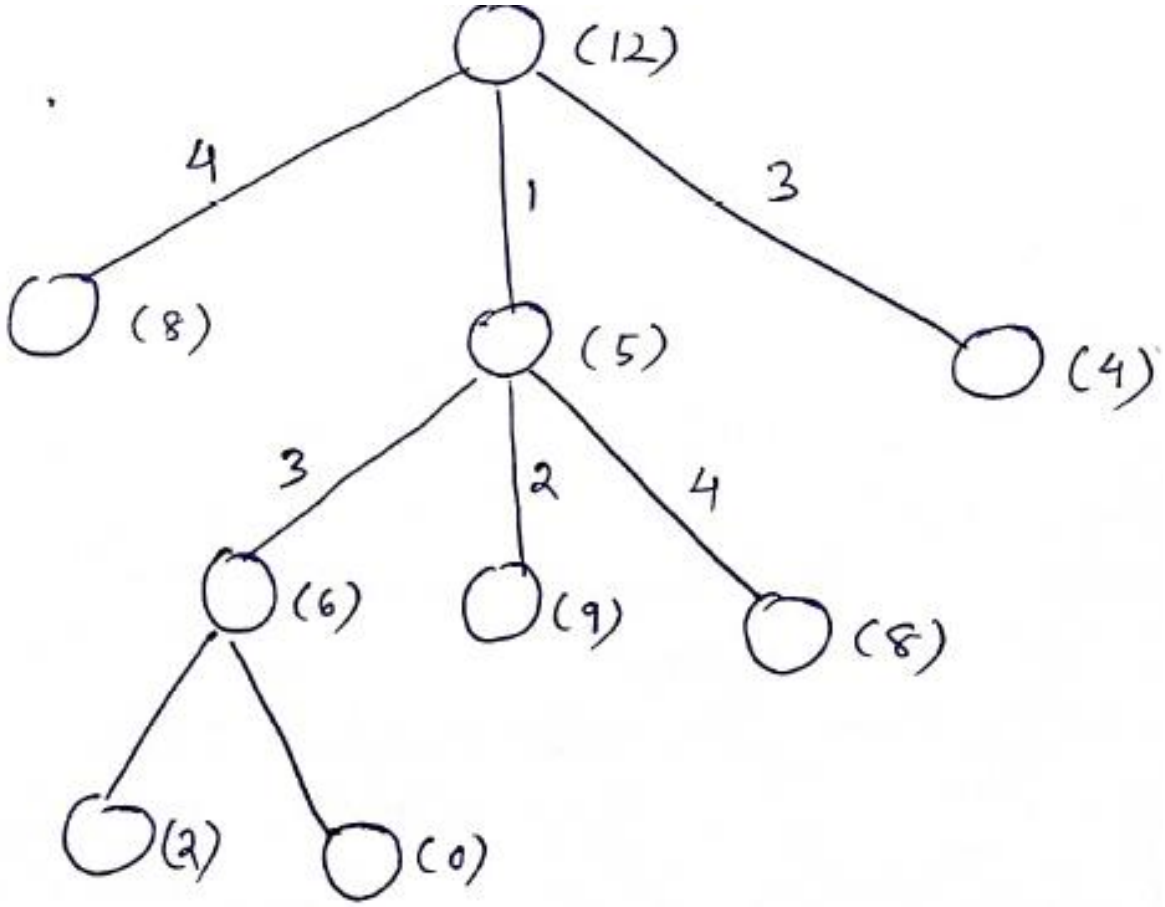
Informed search algorithms

Simulated Annealing

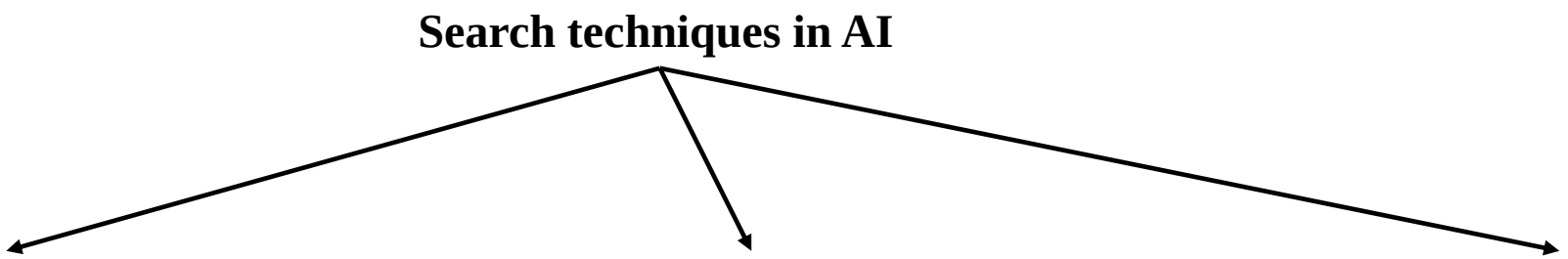
Acceptance Criteria

$$P = \exp(-c/t) > r$$

Where
c = the change in the evaluation function,
t = the current temperature,
r = a random number between 0 and 1



Problem solving by search:



Uninformed search (blind search) algorithms:

Breadth-first search
Uniform cost search
Depth-first search & Depth-limited search
Iterative deepening search

Informed search algorithms:

Greedy best-first search
A*
AO*
Hill Climbing,
Simulated Annealing

Adversarial search algorithms:

MIN-MAX Algorithm
Alpha-Beta Cut-off algorithm

Game Playing and Adversarial search algorithm (Informed search)

- MiniMax Search
- Heuristic Evaluation
- Iterative (Progressive) Deepening Vs Selective Deepening in Game tree
- Search Cut-off (Alpha-Beta pruning)

Game Playing and Adversarial search algorithm (Informed search)

In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.

But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.

The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.

So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.

Types of Game

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Perfect information: A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are tic-tac-toe, Chess, Checkers, Monopoly, Ludo, etc.

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Imperfect information: If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as Battleship, Pokers, etc.

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Deterministic games: Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are tic-tac-toe, Chess, Checkers, Battleship, etc.

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Non-deterministic games (Chance based): Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Monopoly, Ludo, Poker, etc.

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Typical assumptions

- Two agents whose actions alternate
- Utility values for each agent are the opposite of the other. This creates the adversarial situation
- Fully observable environments
- In game theory terms: “Deterministic, turn-taking, zero-sum, perfect information”

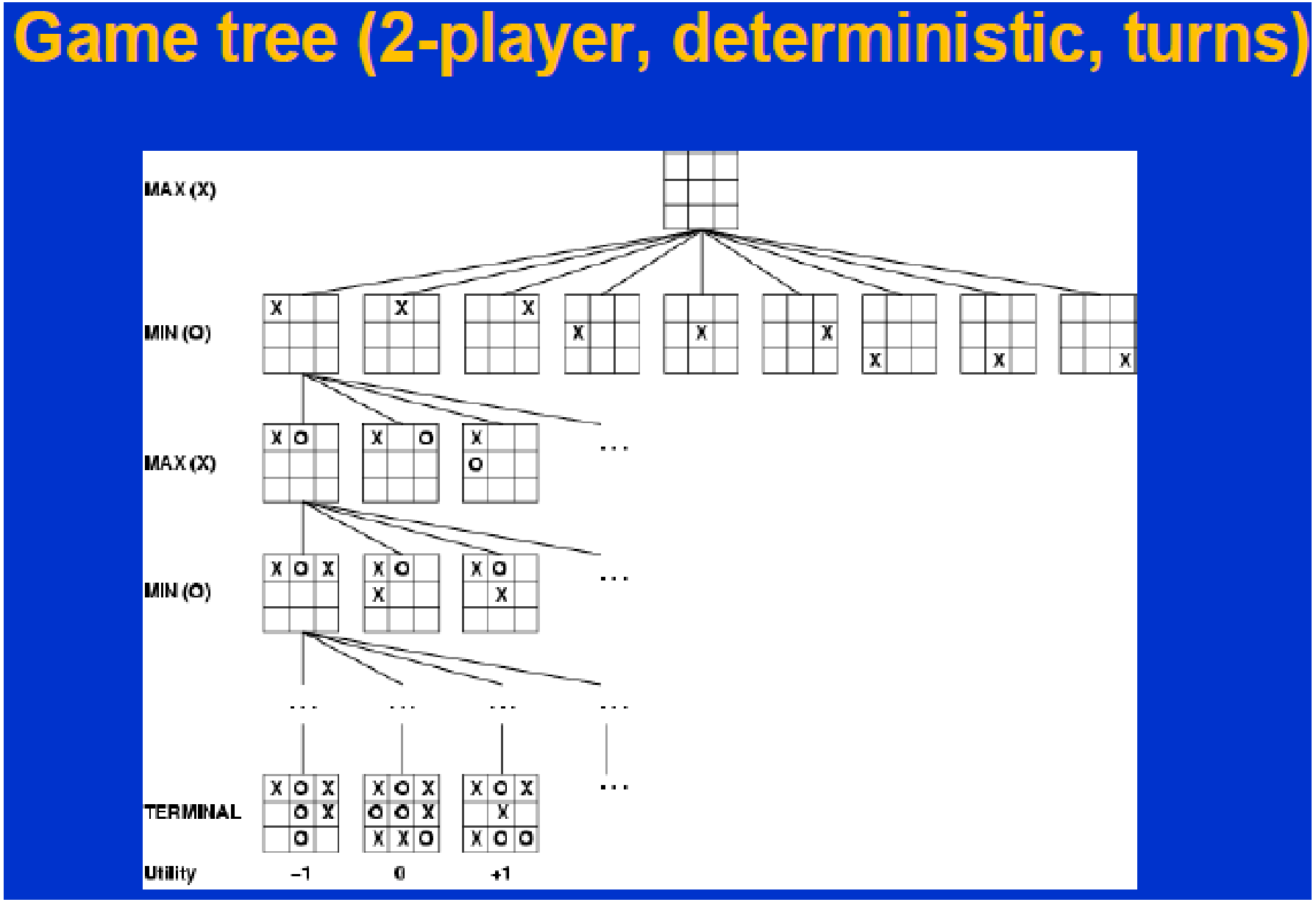
Types of Game

	Deterministic	Chance
Perfect Information	tic-tac-toe, Chess, Checkers	Monopoly, Ludo
Imperfect Information	Battleship	Pokers (Cards)

Game Playing and Adversarial search algorithm (Informed search)

Game tree:

A **game tree** is a type of recursive search function that examines all possible moves of a strategy game, and their results, in an attempt to explore the optimal move.



Game Playing and Adversarial search algorithm (Informed search)

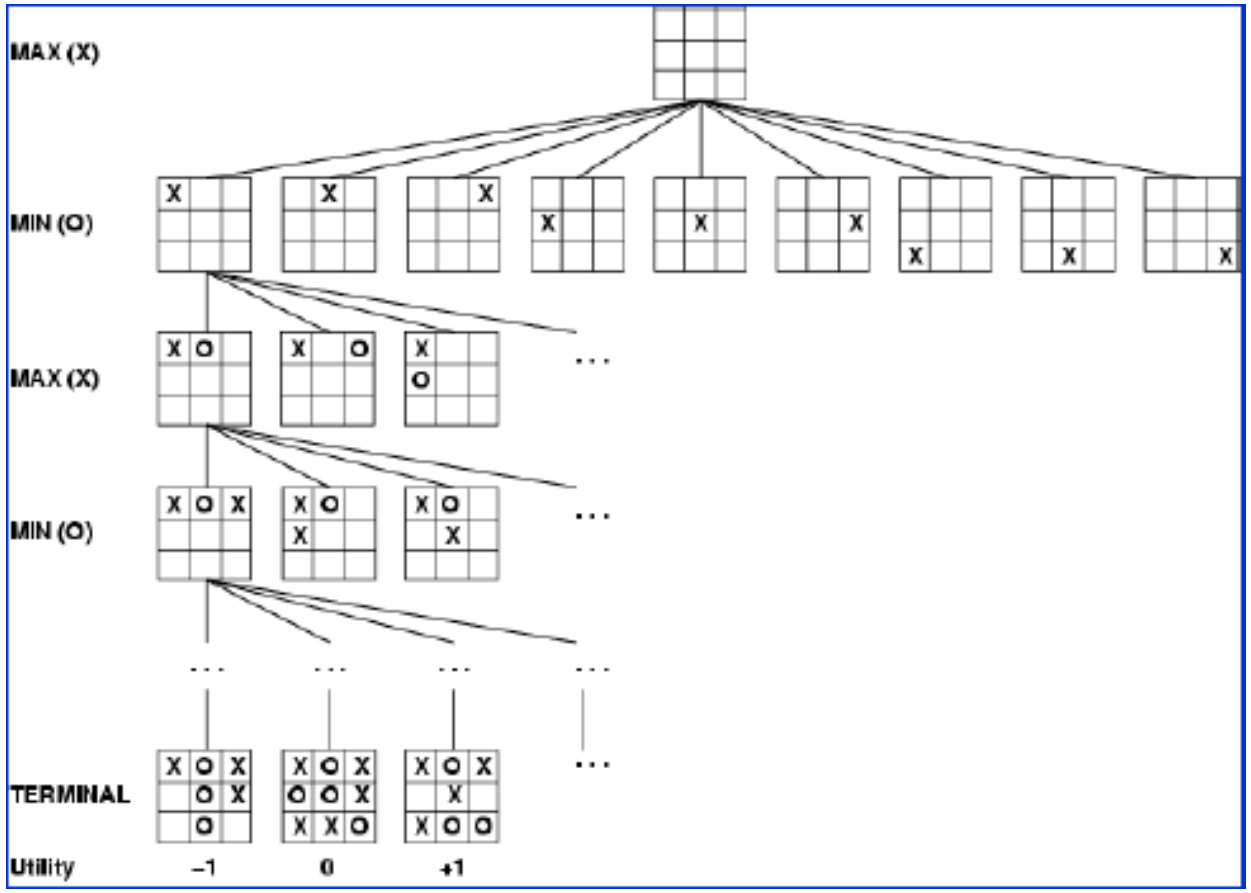
Search Vs Game

Search	Game
Solution is (heuristic) path for finding goal	Solution is strategy (strategy specifies move for every possible opponent reply)
Heuristics and CSP techniques can find optimal solution	An approximate strategy solution (with a time limit)
Evaluation function: estimate of cost from start to goal through given node	Evaluation function: evaluate "goodness" of game position
Examples: path planning, scheduling activities	Examples: chess, checkers, tic-tac-toe

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Two players: MAX and MIN



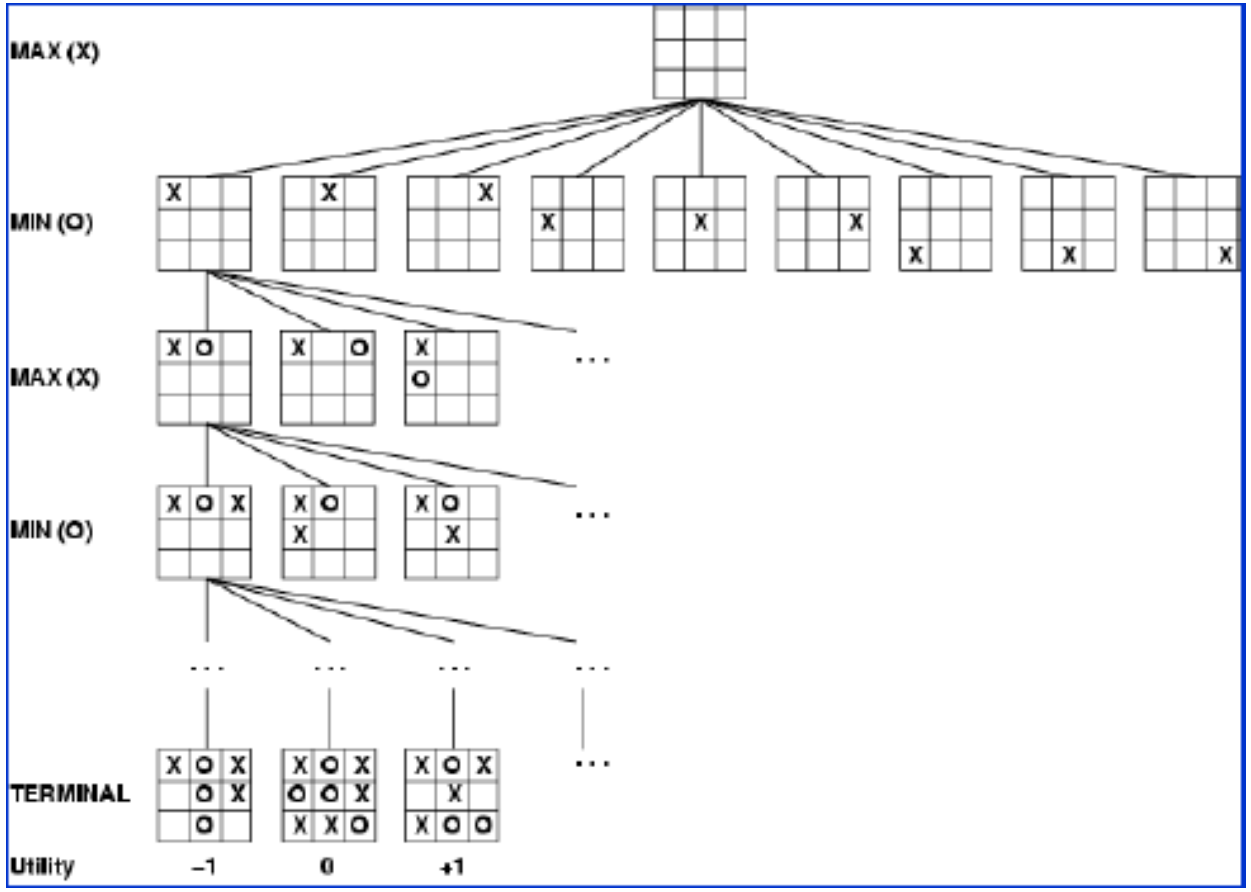
Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Two players: MAX and MIN

MAX moves first and they take turns until the game is over

- Winner gets reward, loser gets penalty.
- "Zero sum" means the sum of the reward and the penalty is a constant.



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

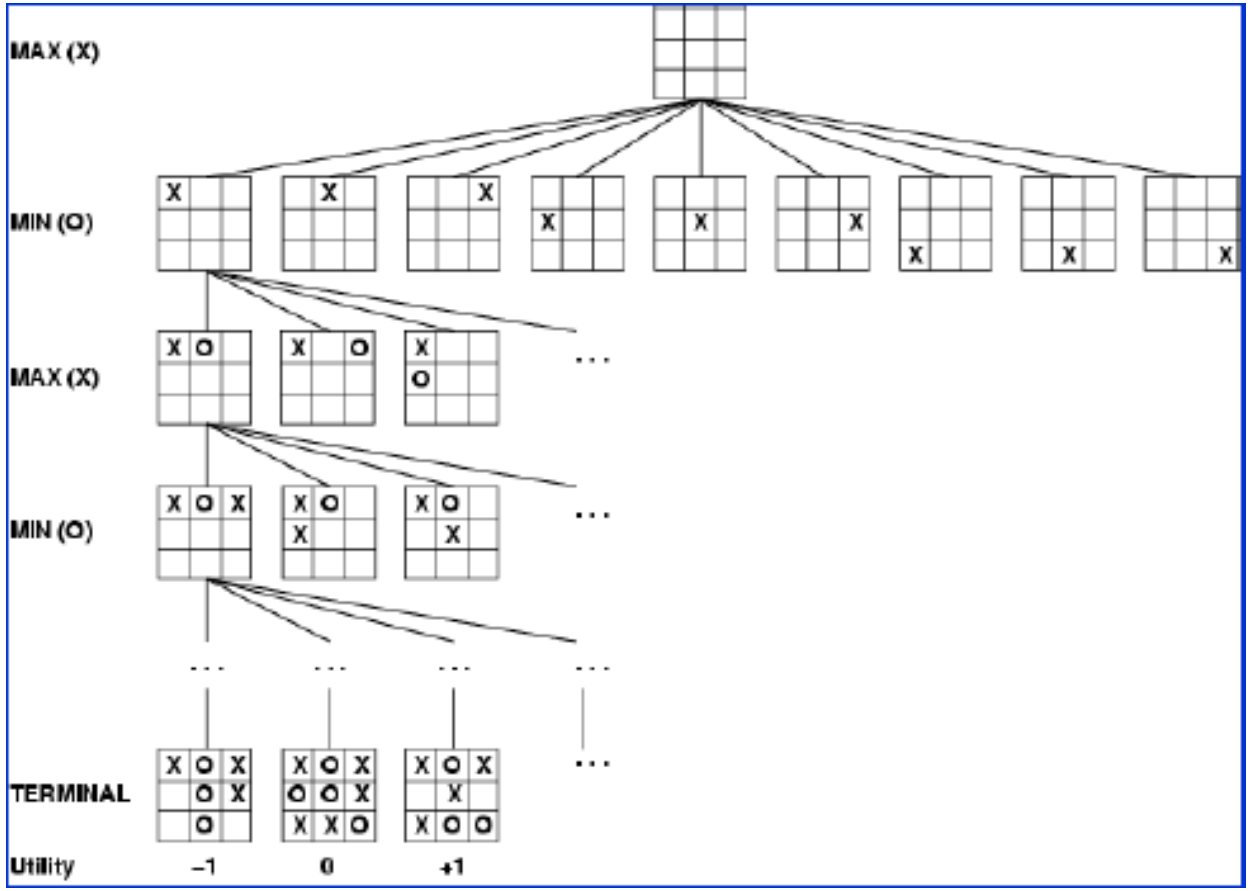
Two players: MAX and MIN

MAX moves first and they take turns until the game is over

- Winner gets reward, loser gets penalty.
- "Zero sum" means the sum of the reward and the penalty is a constant.

Formal definition as a search problem:

- **Initial state:** Set-up specified by the rules, e.g., initial board set-up of chess.
- **Player(s):** Defines which player has the move in a state.
- **Actions(s):** Returns the set of legal moves in a state.
- **Result(s,a):** Transition model defines the result of a move.
- **Terminal-Test(s):** Is the game finished? True if finished, false otherwise.
- **Utility function(s,p):** Gives numerical value of terminal state s for player p.
  - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
  - E.g., win (+1), lose (0), and draw (1/2) in chess.



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

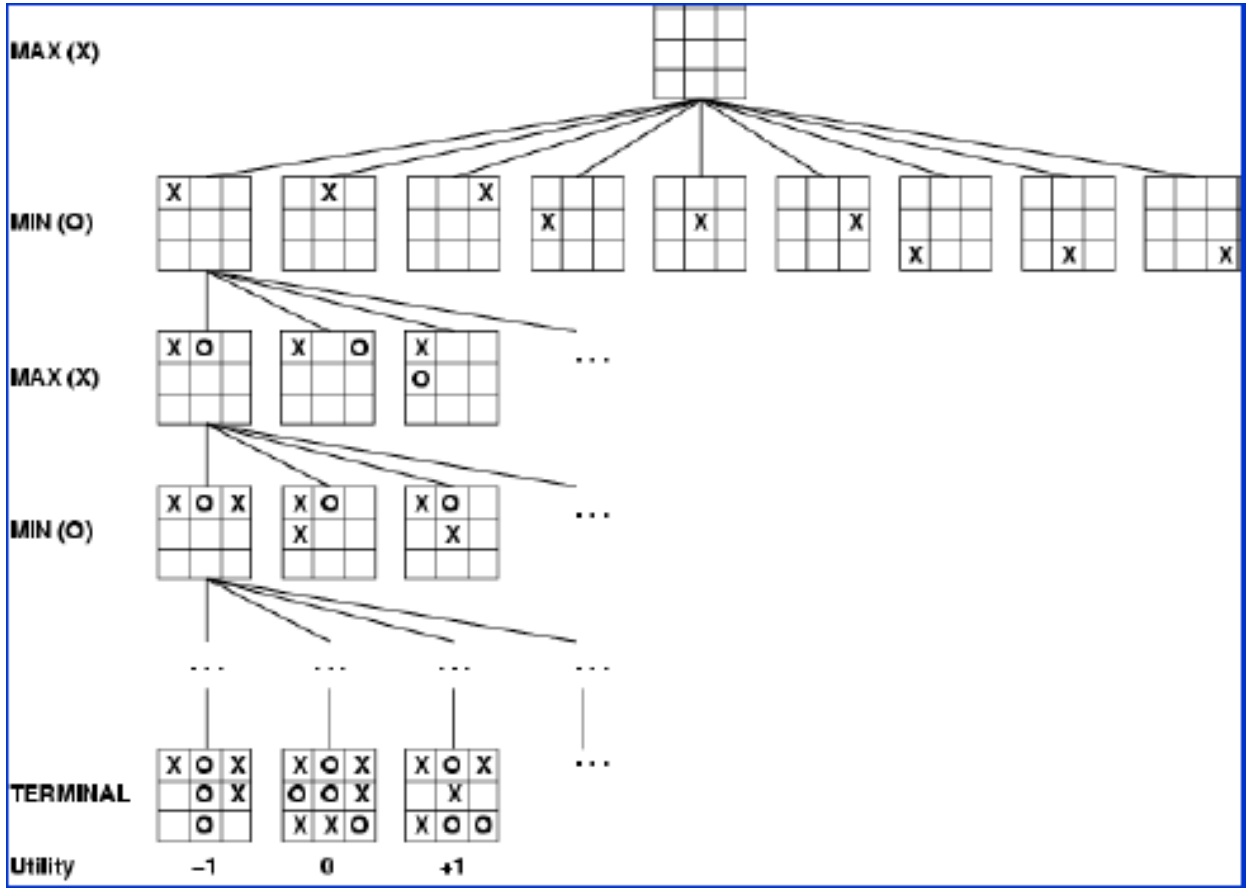
Two players: MAX and MIN

MAX moves first and they take turns until the game is over

- Winner gets reward, loser gets penalty.
- "Zero sum" means the sum of the reward and the penalty is a constant.

Formal definition as a search problem:

- **Initial state:** Set-up specified by the rules, e.g., initial board set-up of chess.
- **Player(s):** Defines which player has the move in a state.
- **Actions(s):** Returns the set of legal moves in a state.
- **Result(s,a):** Transition model defines the result of a move.
- **Terminal-Test(s):** Is the game finished? True if finished, false otherwise.
- **Utility function(s,p):** Gives numerical value of terminal state s for player p.
  - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
  - E.g., win (+1), lose (0), and draw (1/2) in chess.



MAX uses search tree to determine "best" next move.

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

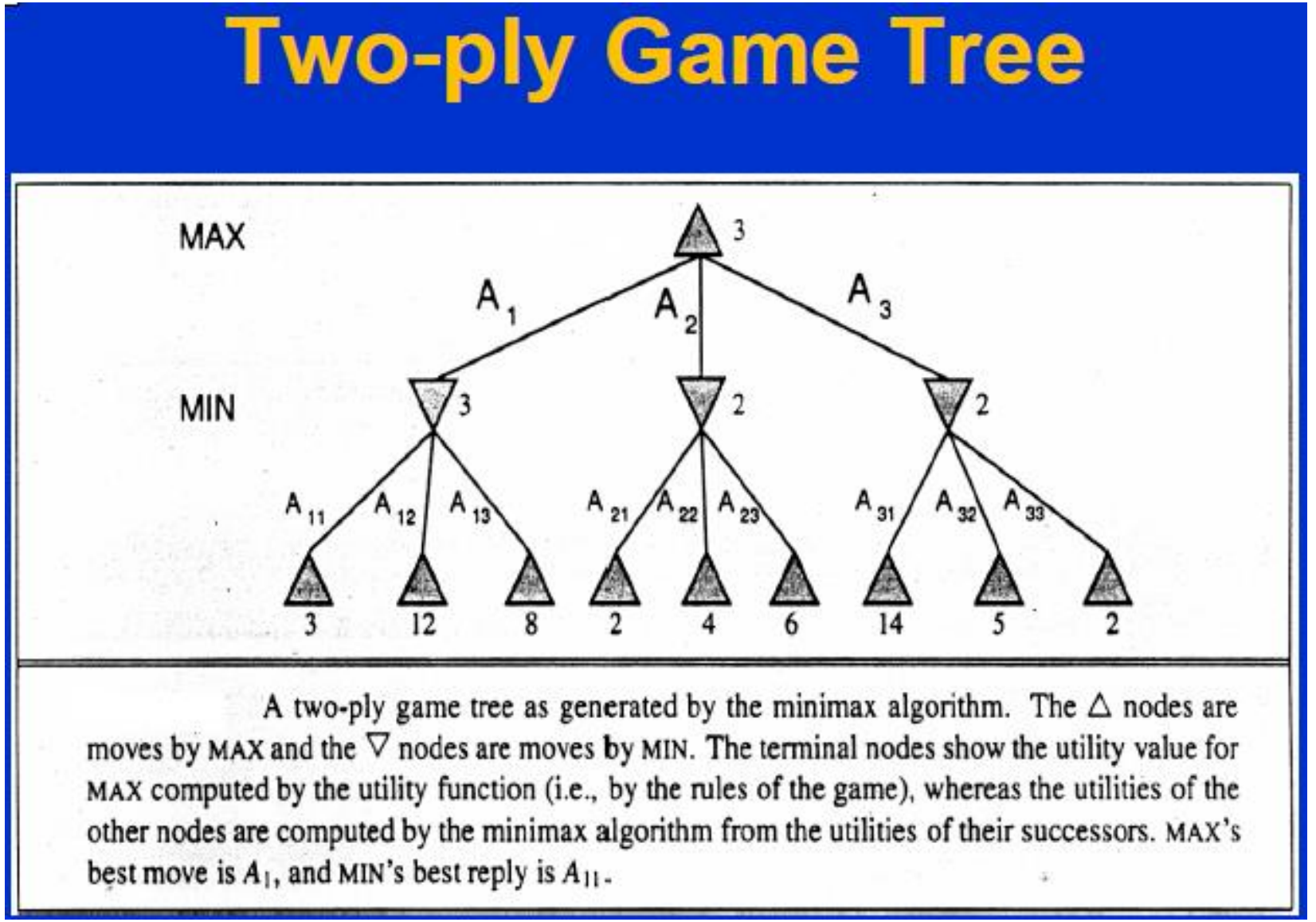
It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.



A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX's best move is A_1, and MIN's best reply is A_{11}.

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

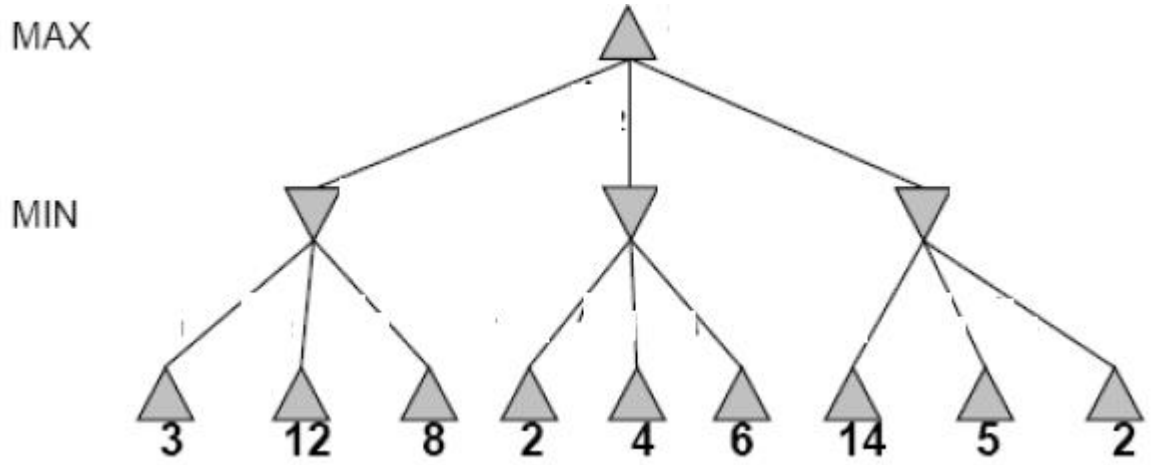
It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

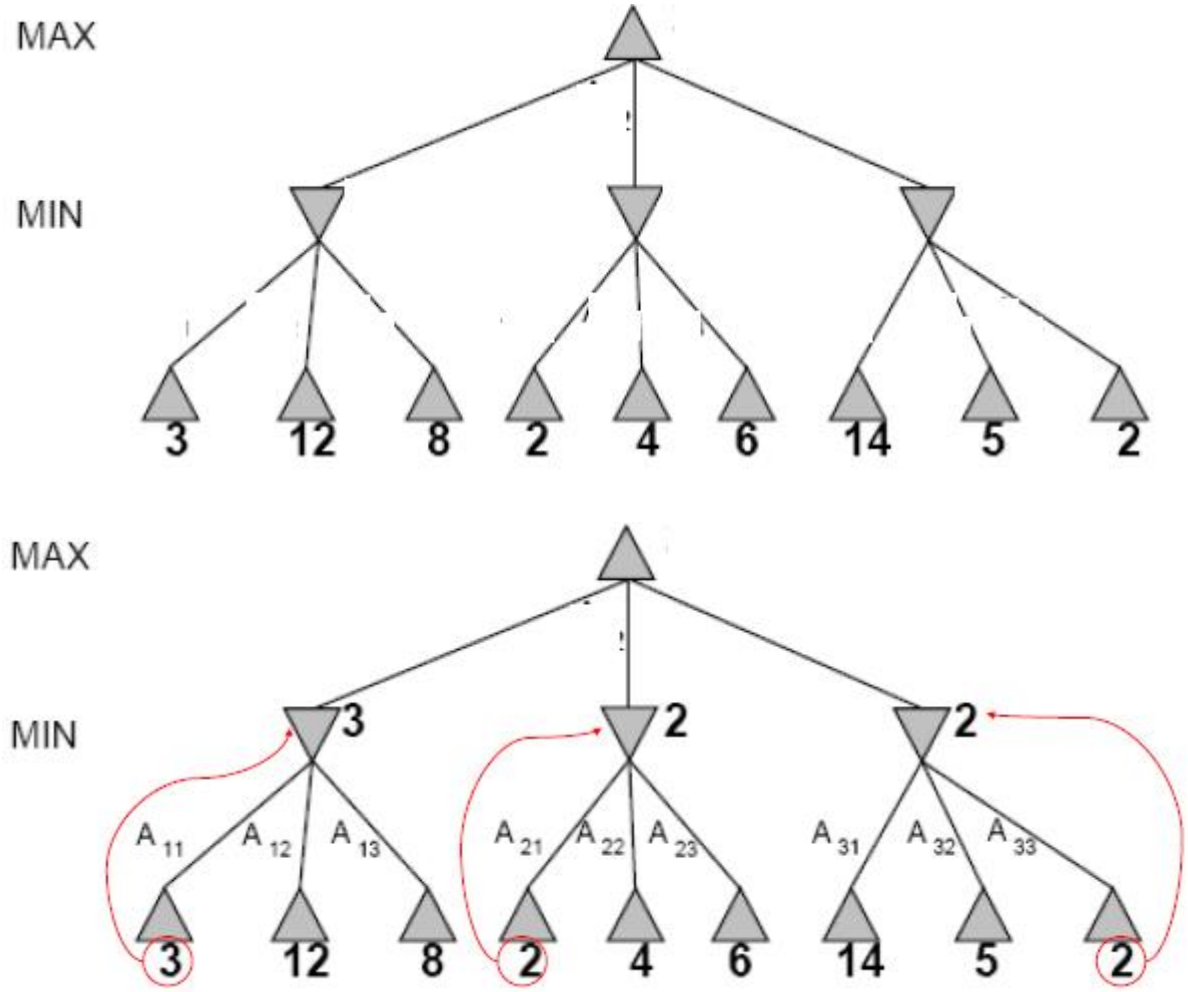
It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

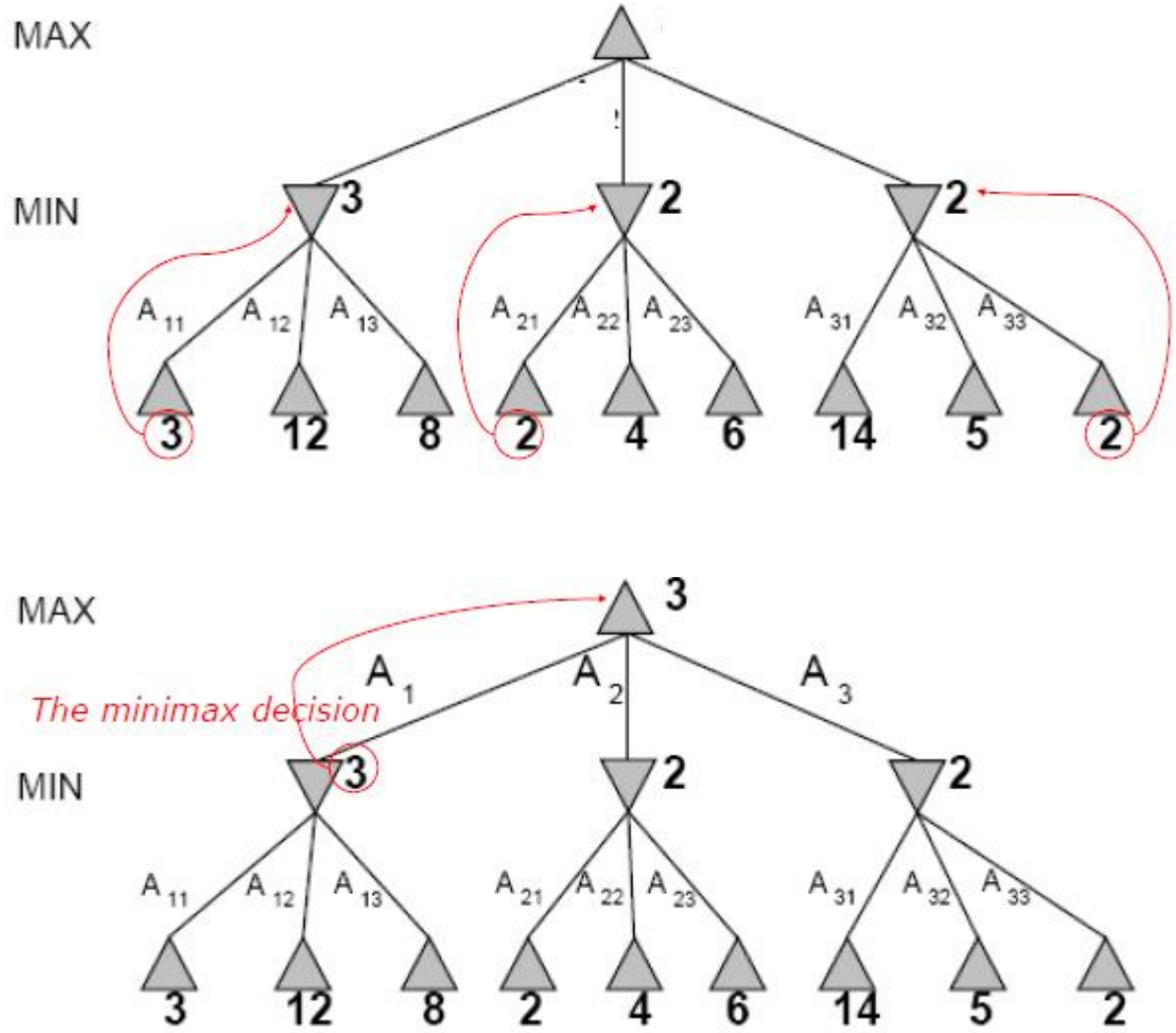
It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

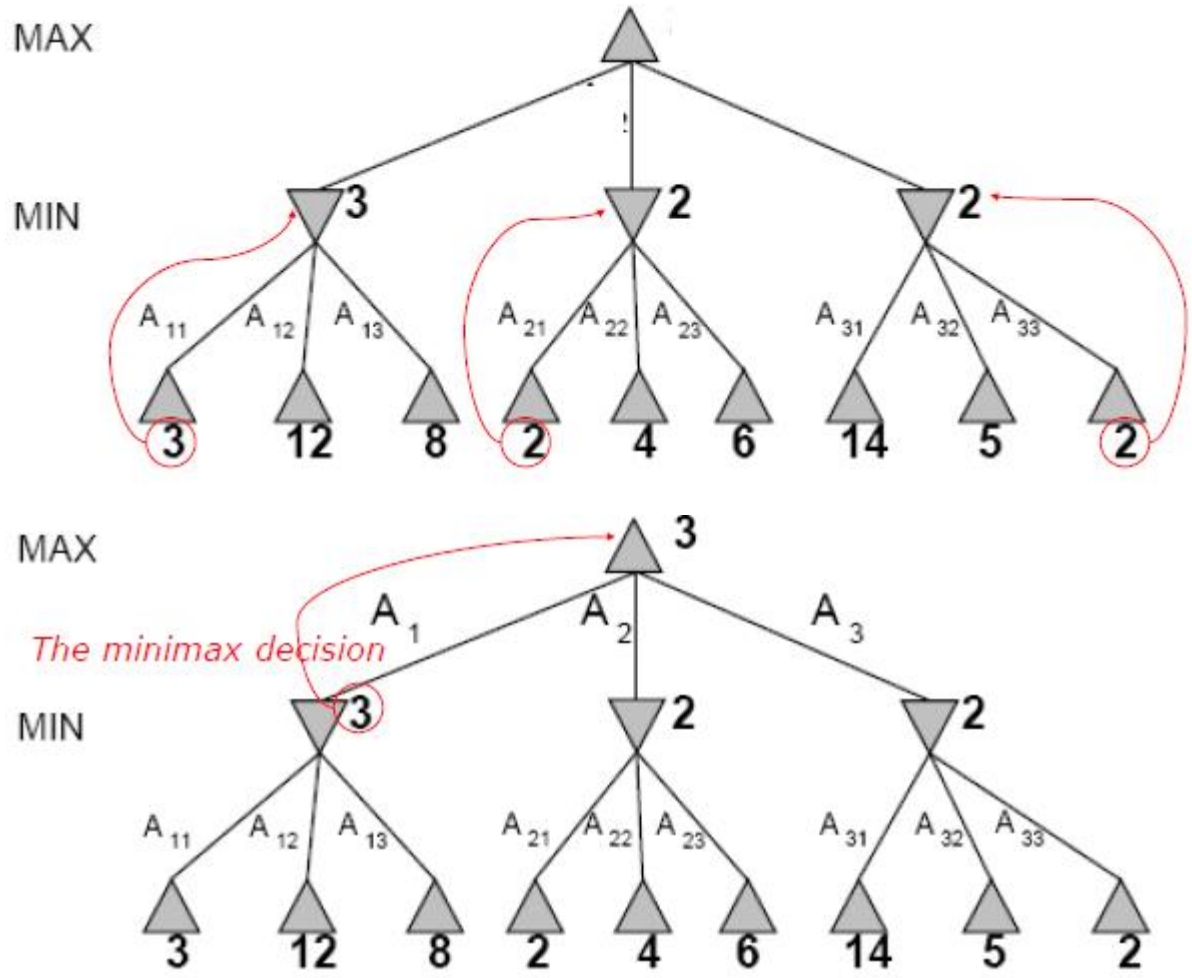
It is a method to find the optimal strategy and best next move for Max:

1. Generate the whole game tree, down to the leaves.

2. Apply utility (payoff) function to each leaf.

3. Back-up values from leaves through branch nodes:
-a Max node computes the Max of its child values
–a Min node computes the Min of its child values

4. At root: Choose move leading to the child of highest value.



Minimax maximizes the utility of the worst-case outcome for Max

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

Game Tree Size

- **Tic-Tac-Toe**
  - b ≈ 5 legal actions per state on average, total of 9 plies in game.
    - "ply" = one action by one player, "move" = two plies.
  - 5^9 = 1,953,125
  - 9! = 362,880 (Computer goes first)
  - 8! = 40,320 (Computer goes second)
  - → **exact solution quite reasonable**

- **Chess**
  - b ≈ 35 (approximate average branching factor)
  - d ≈ 100 (depth of game tree for "typical" game)
  - $b^d \approx 35^{100} \approx 10^{154}$ nodes!!
  - → **exact solution completely infeasible**

- **It is usually impossible to develop the whole search tree.**

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

Heuristic Evaluation Function

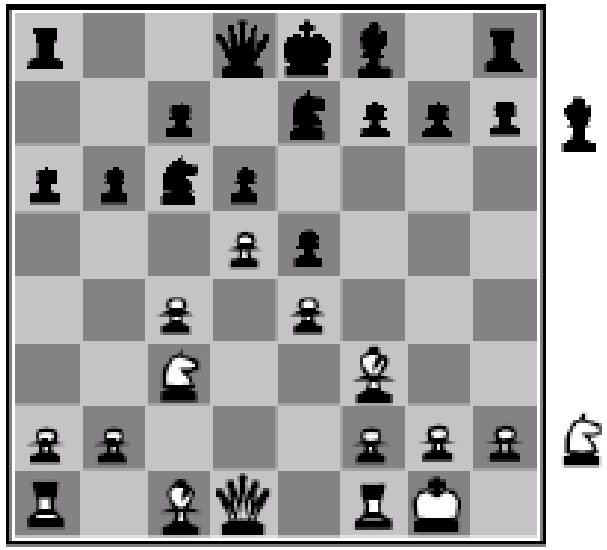
- **An Evaluation Function:**
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent's score from the player's.
  - Often called "static" because it is called on a static board position.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces
- **Typical values :**
  - -infinity (loss) to +infinity (win) or [-1, +1] or [0, +1].
- **Board evaluation X for player is -X for opponent**
  - "Zero-sum game" (i.e., scores sum to a constant)

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

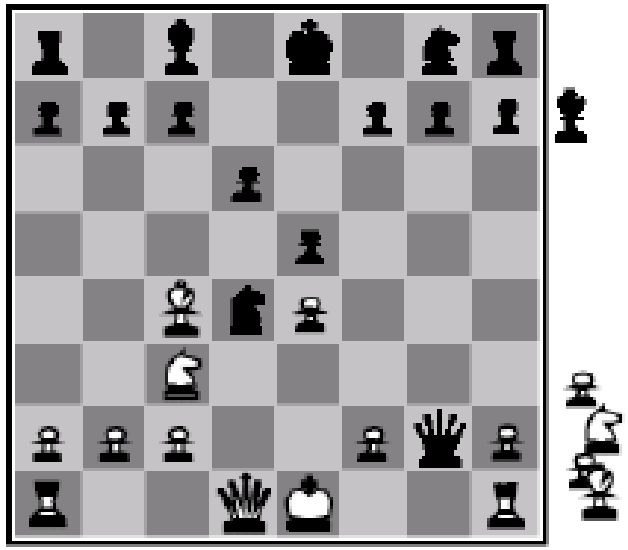
Min-Max Search:

Heuristic Evaluation Function



Black to move

White slightly better



White to move

Black winning

For chess, typically *linear* weighted sum of features

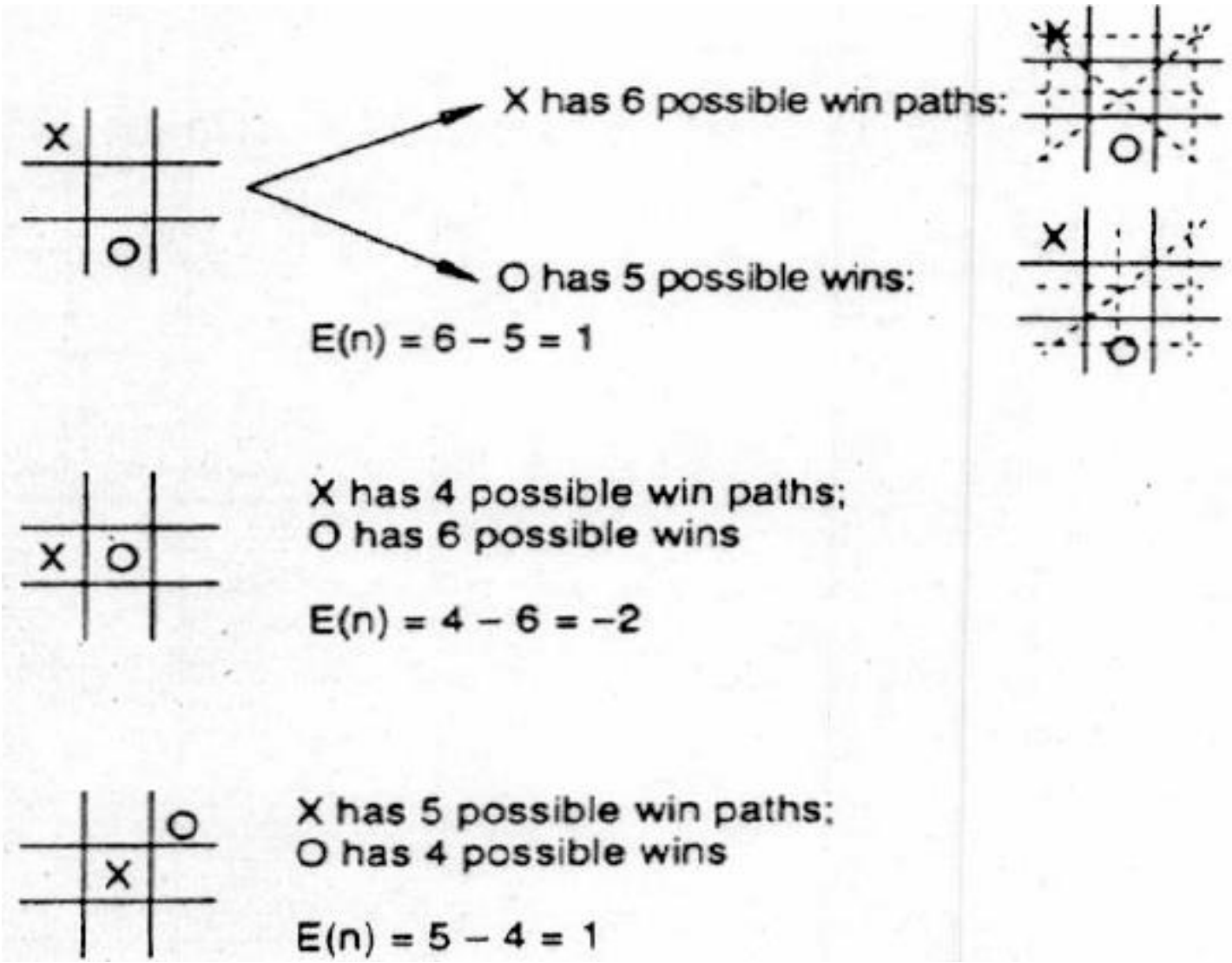
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:



Heuristic Evaluation Function

Heuristic is E(n) = M(n) – O(n)

where M(n) is the total of My possible winning lines

O(n) is total of Opponent's possible winning lines

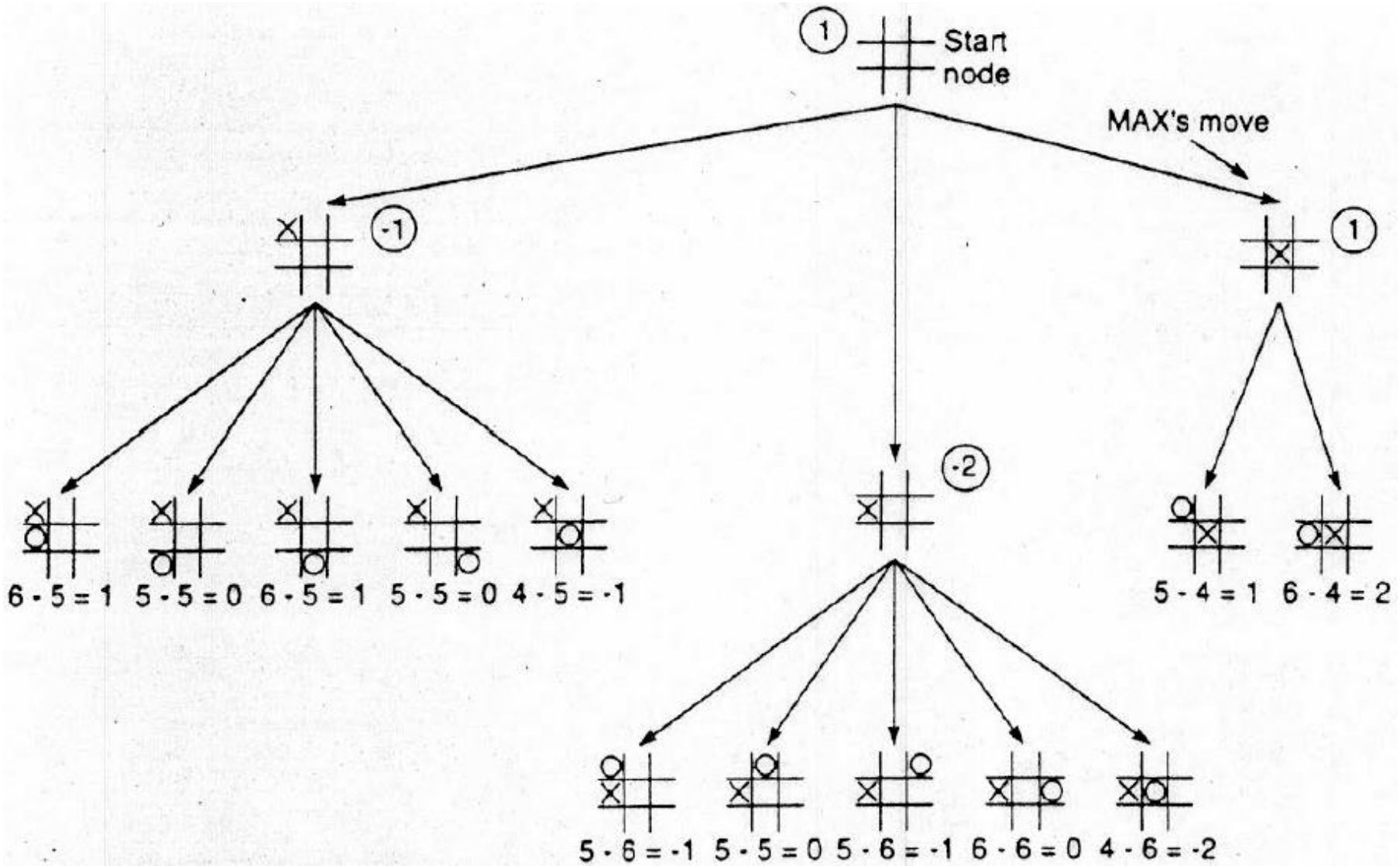
E(n) is the total Evaluation for state n

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

Heuristic Evaluation Function

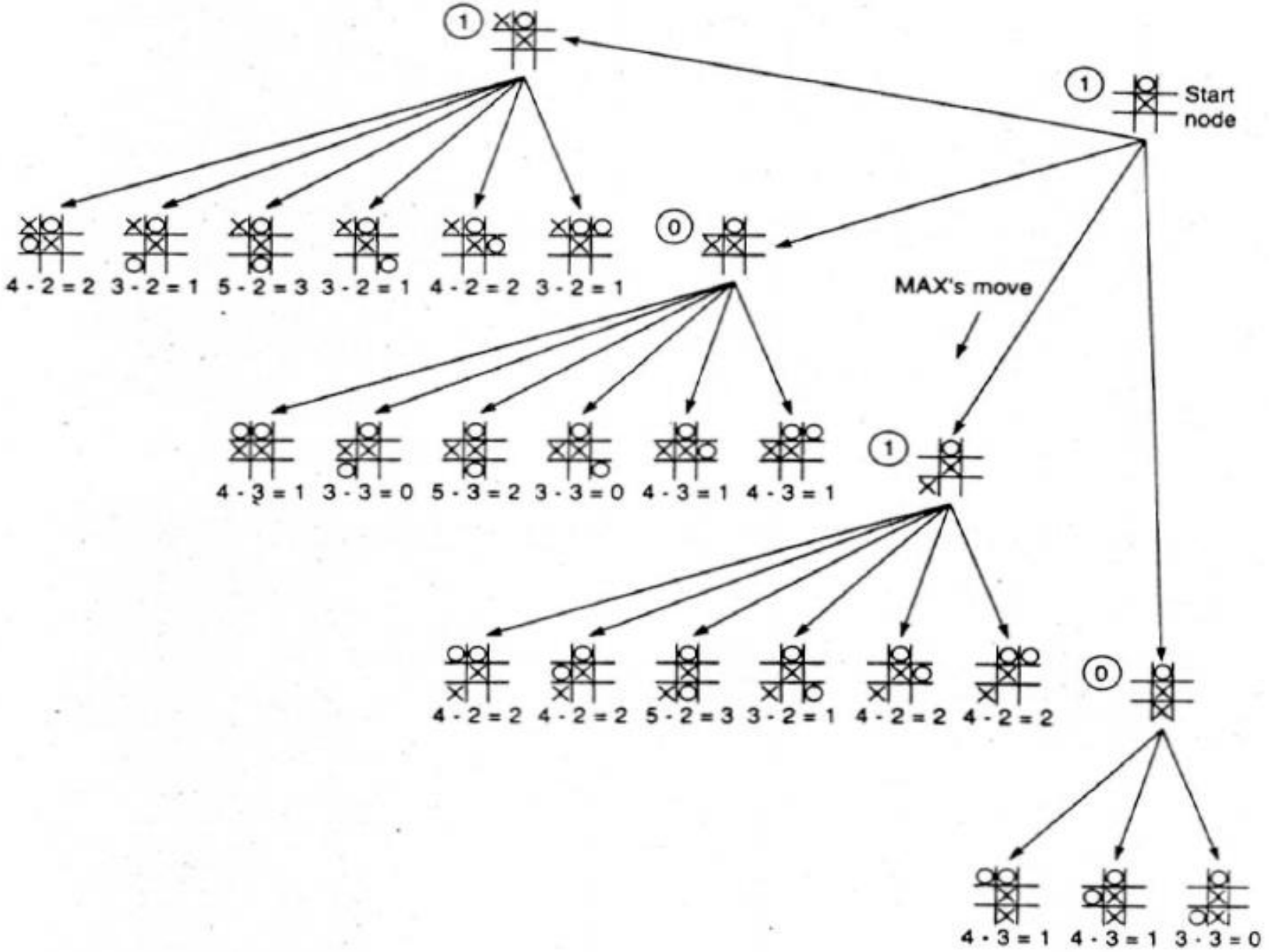


Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:

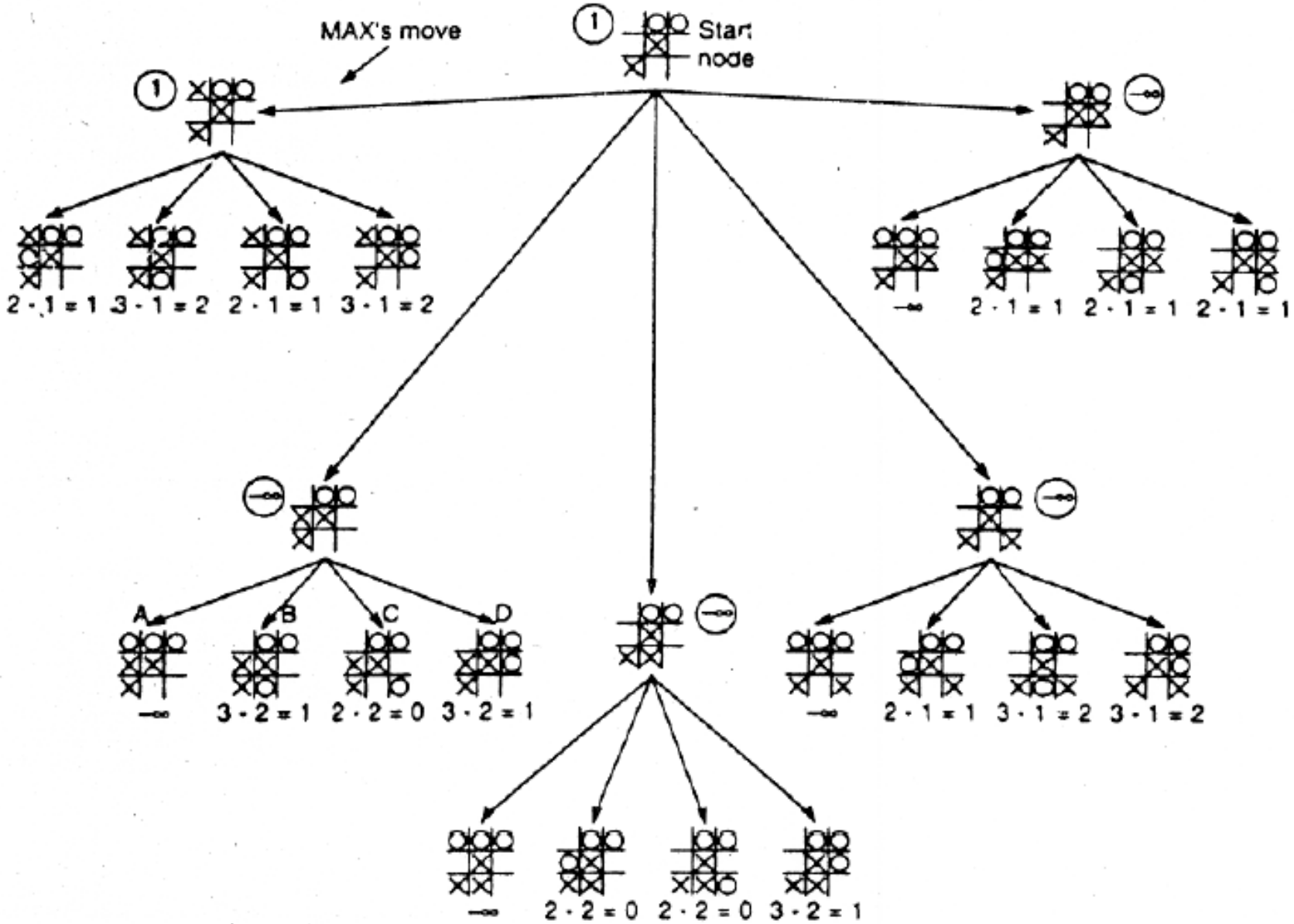
Heuristic Evaluation Function



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search:



Heuristic Evaluation Function

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search (with Iterative Deepening)

In real games, there is usually a time limit T to make a move

How do we take this into account?

Using MiniMax we cannot use "partial" results with any confidence unless the full tree has been searched So, we could be conservative and set a conservative depth-limit which guarantees that we will find a move in time < T Disadvantage: we may finish early, could do more search

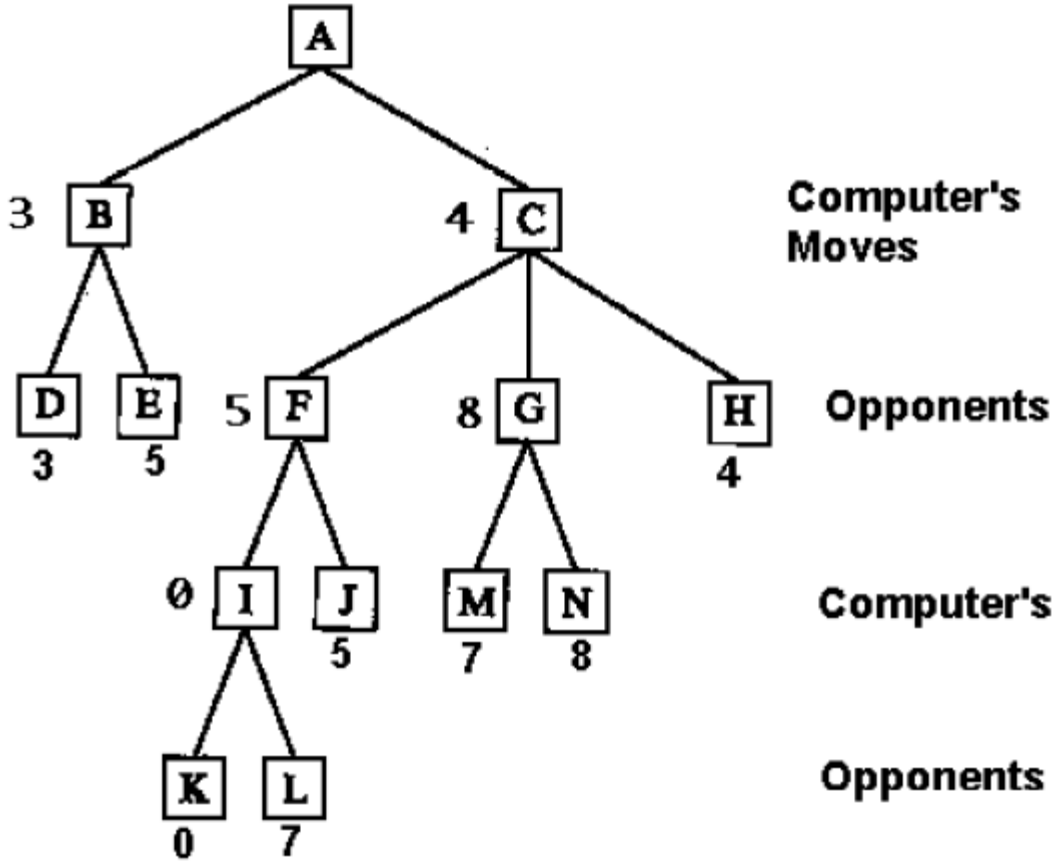
In practice, Iterative Deepening Search (IDS) is used
IDS runs depth-first search with an increasing depth-limit

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search (with Iterative Deepening)

Selectively Deeper Game Trees



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search (with Iterative Deepening)

Eliminate Redundant Nodes

- **On average, each board position appears in the search tree approximately 10^100 times**
- **Vastly redundant states**
- **Can’t remember all nodes (too many). Also, Can’t eliminate all redundant nodes.**

Cutting off search: In general, it is infeasible to search the entire game tree.
In practice, Cutoff-Test decides when to stop searching further.
- With alpha-beta pruning (next lecture)

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search (with Iterative Deepening)

Eliminate Redundant Nodes

- **On average, each board position appears in the search tree approximately 10^100 times**
- **Vastly redundant states**
- **Can’t remember all nodes (too many). Also, Can’t eliminate all redundant nodes.**

Cutting off search: In general, it is infeasible to search the entire game tree.
In practice, Cutoff-Test decides when to stop searching further.
- With alpha-beta pruning

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Min-Max Search (with Iterative Deepening)

Cutting off search: In general, it is infeasible to search the entire game tree.
In practice, Cutoff-Test decides when to stop searching further.
- With alpha-beta pruning (next lecture)

Alpha-beta pruning

Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Alpha-beta pruning

- Try to cut-off search space by exploring less number of nodes
- The final outcome is not hampered after the cut-off some search space
- It is a Depth First Approach

Assumptions:

1. **Two players** (**Max** and **Min**)
2. **Alpha** is used for **Max** and **Beta** is used for **Min** to decide **limit (bound)**
3. **Alpha** is initialized with **–Infinity** and **Beta** is initialized with **+Infinity**

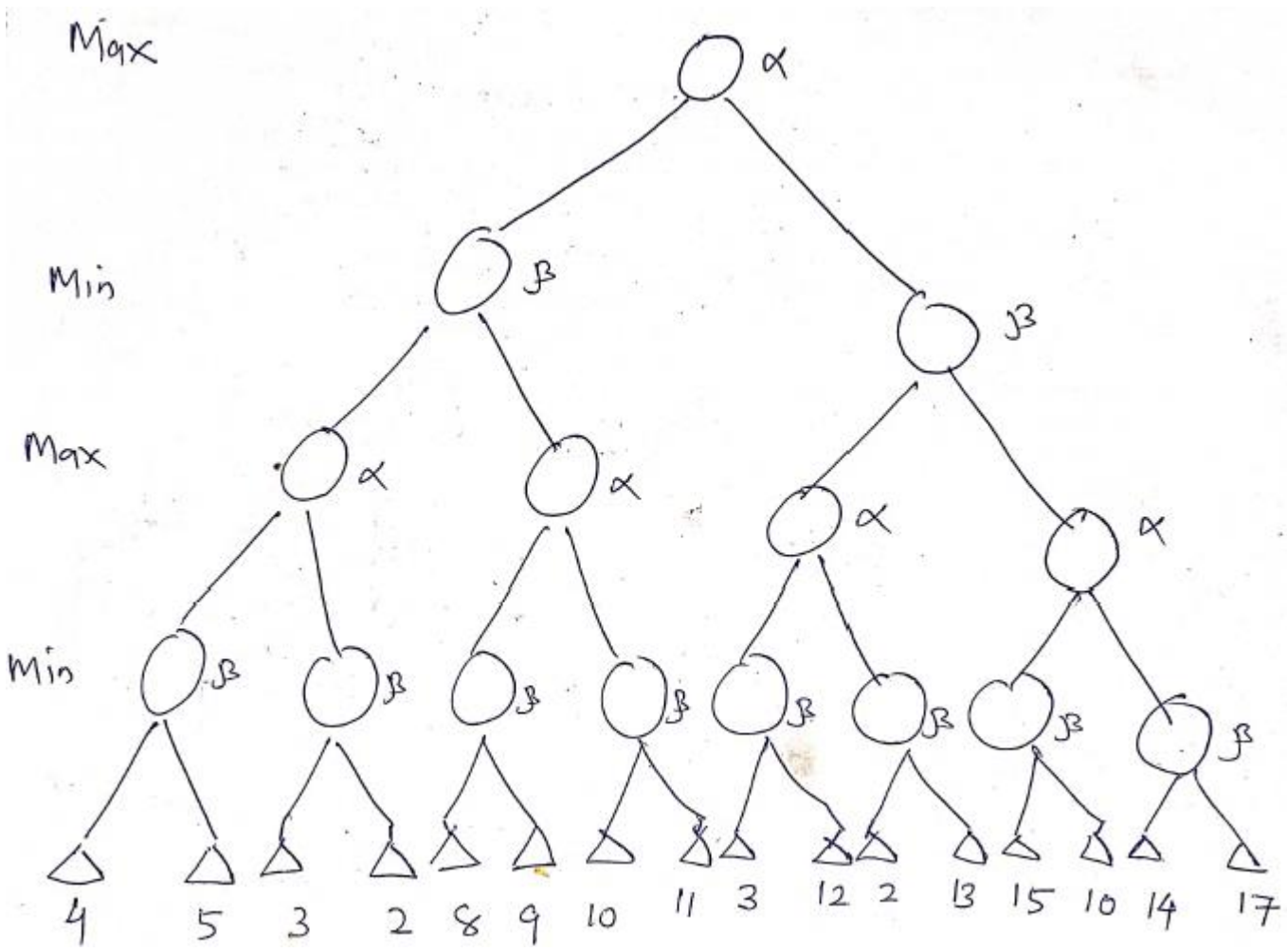
Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Alpha-beta pruning (Example)

Assumptions:

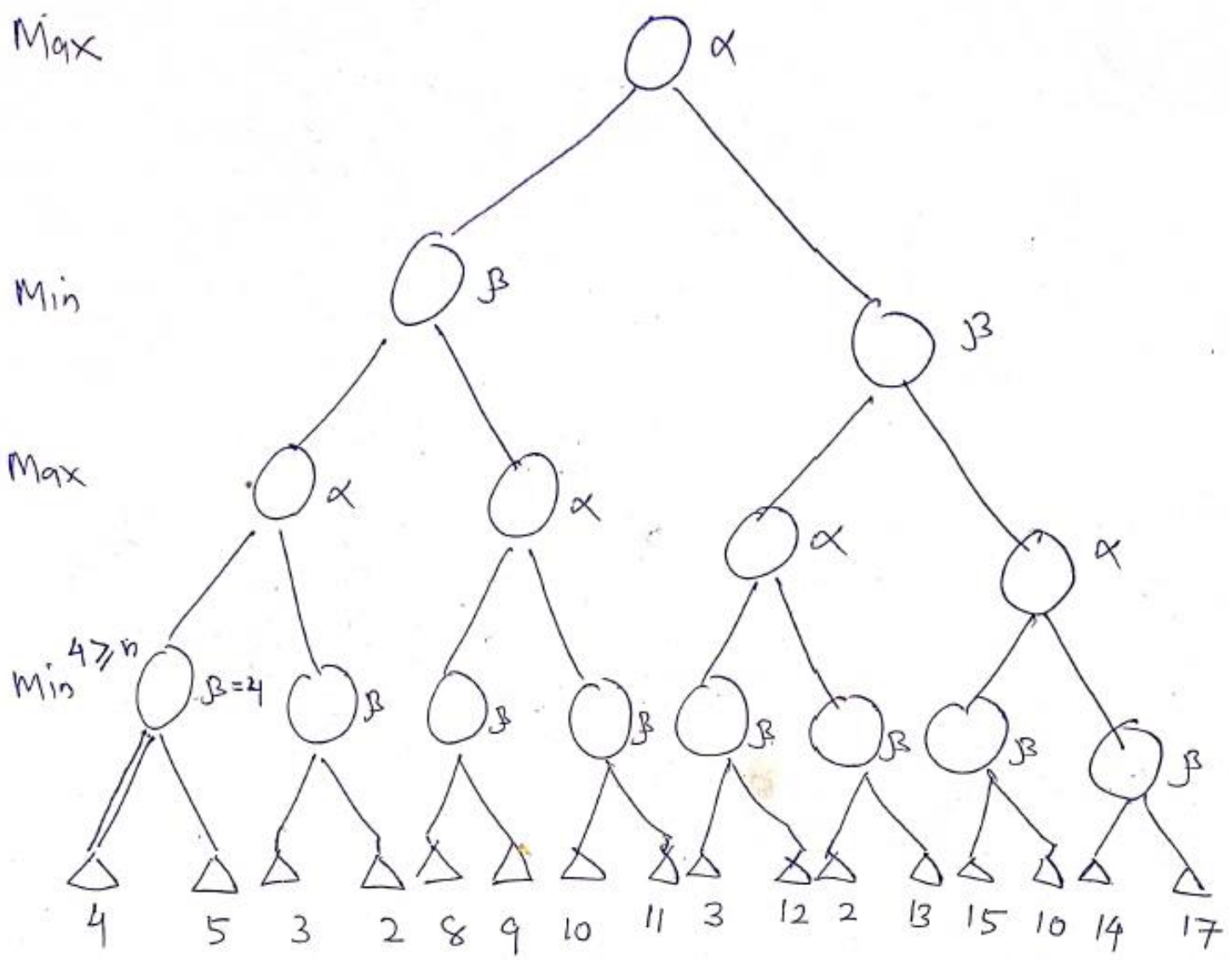
1. **Two players** (**Max** and **Min**)
2. **Alpha** is used for **Max** and **Beta** is used for **Min** to decide **limit (bound)**
3. **Alpha** is initialized with **–Infinity** and **Beta** is initialized with **+Infinity**



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

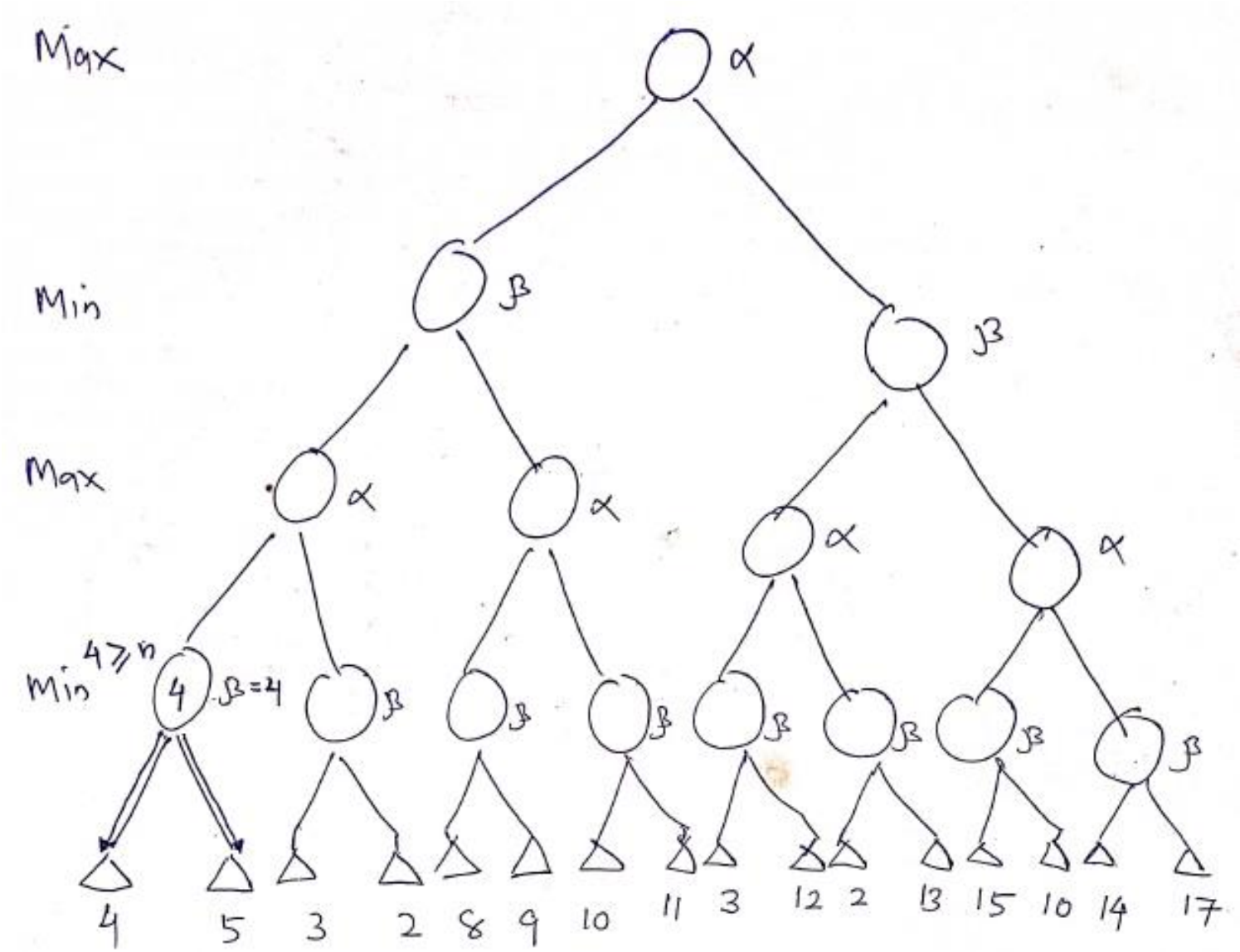
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

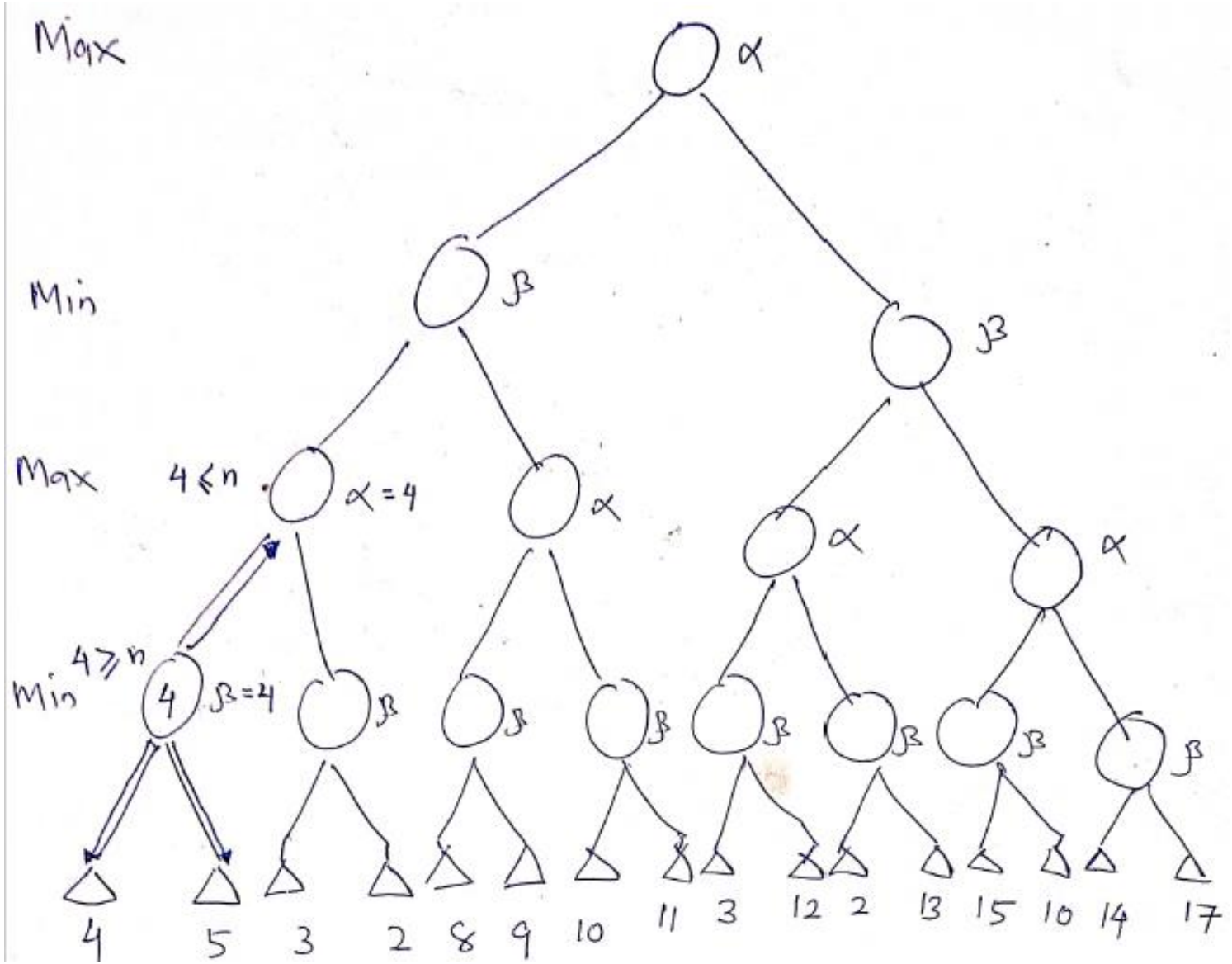
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

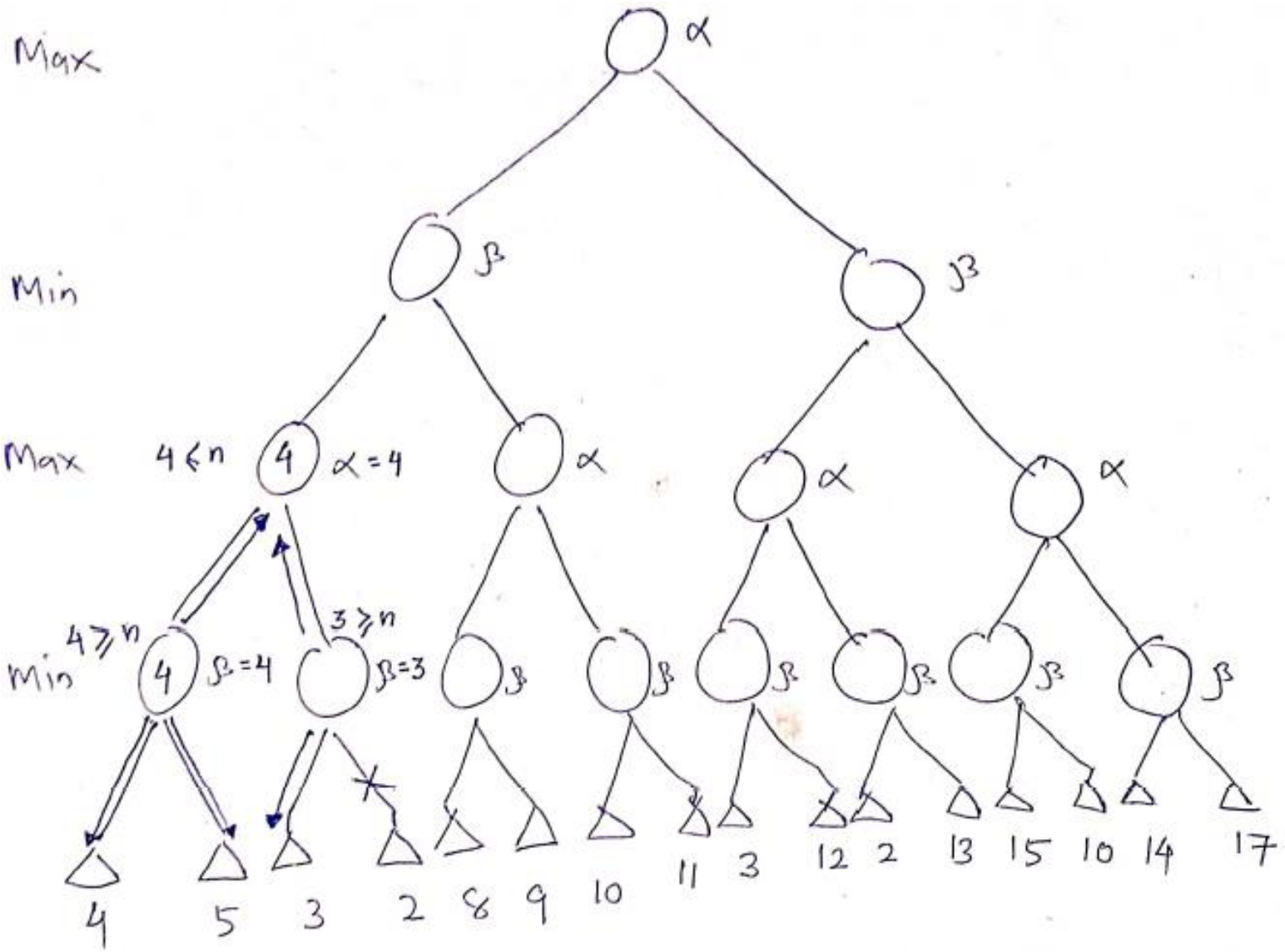
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

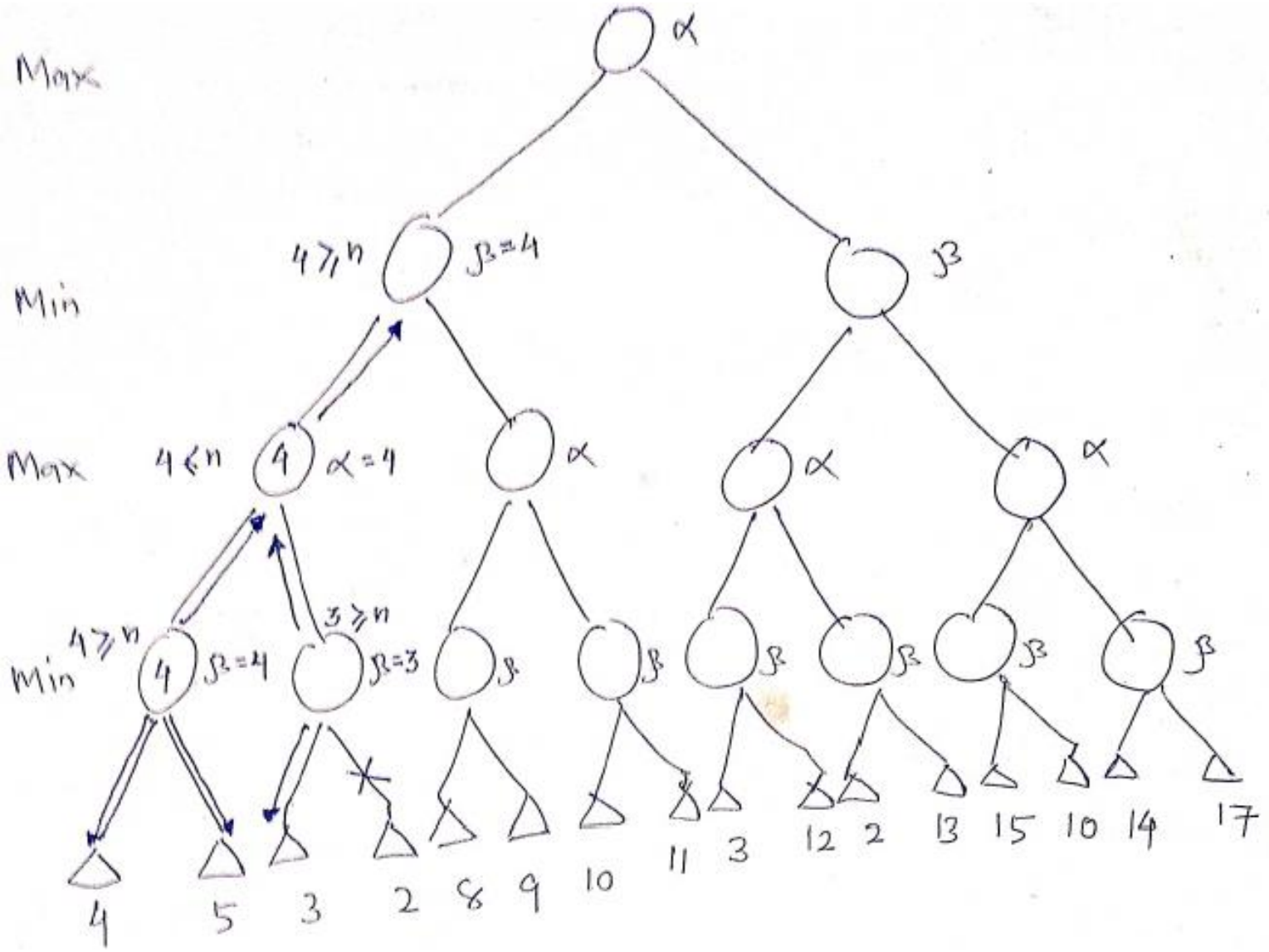
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

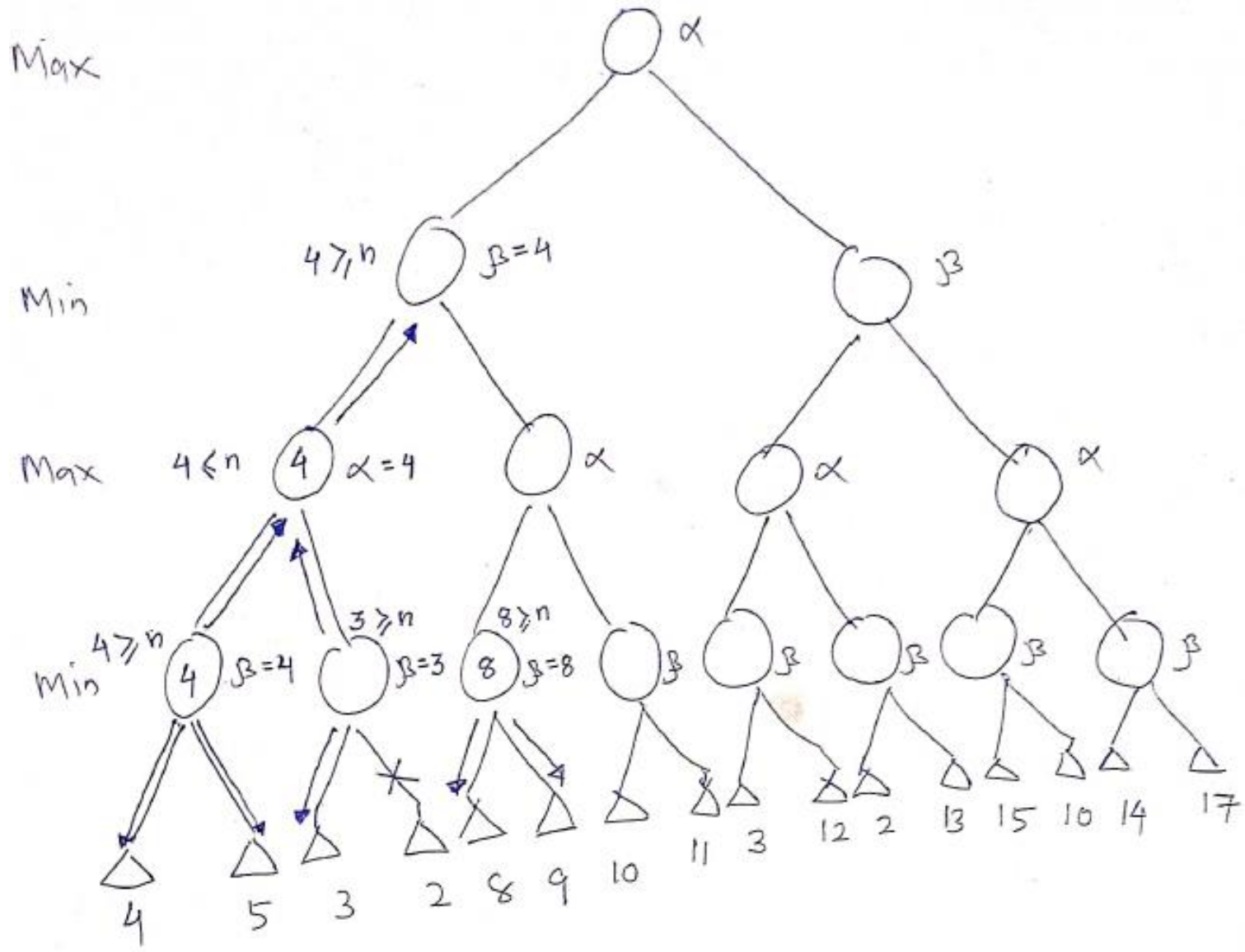
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

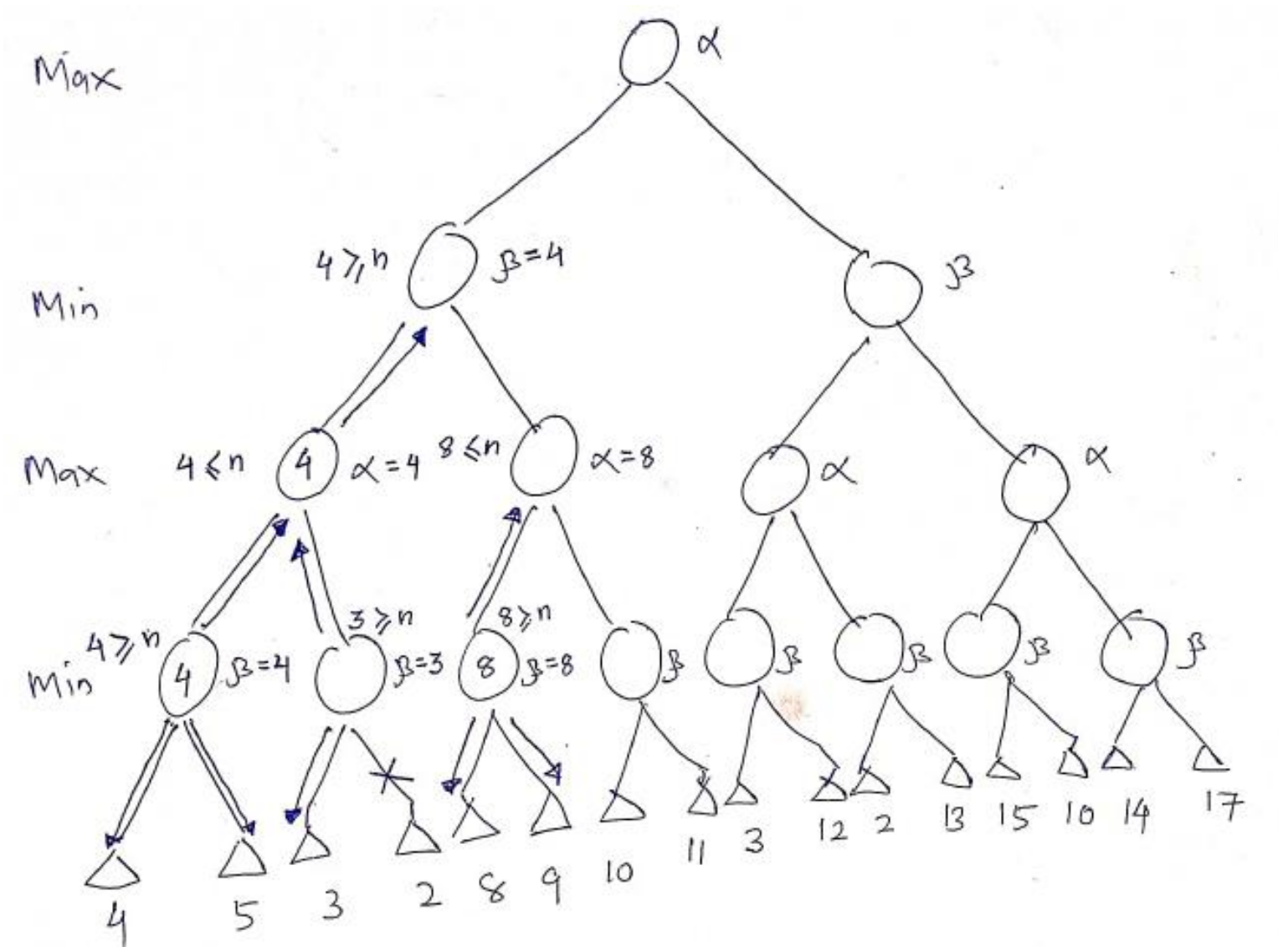
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

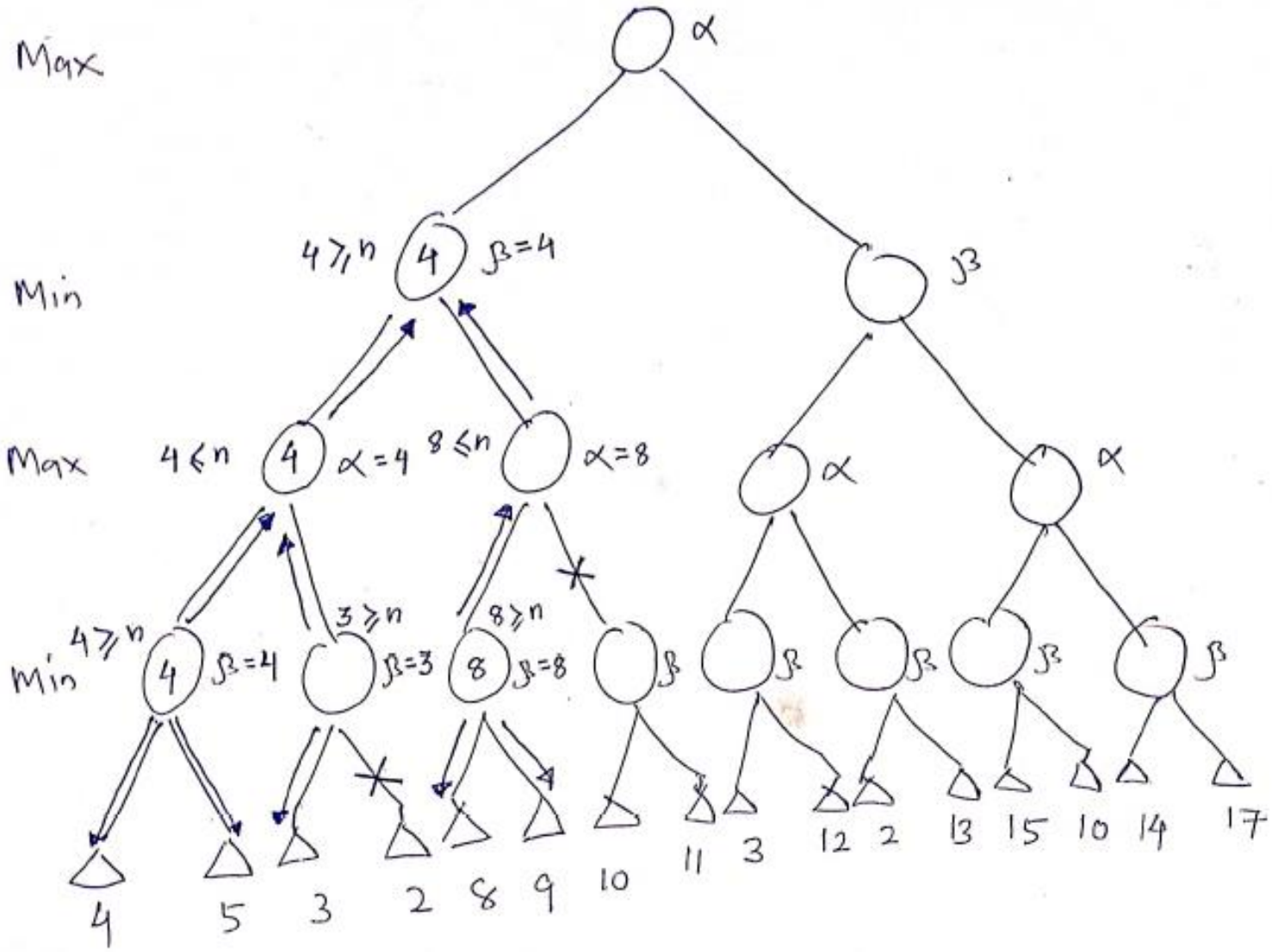
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

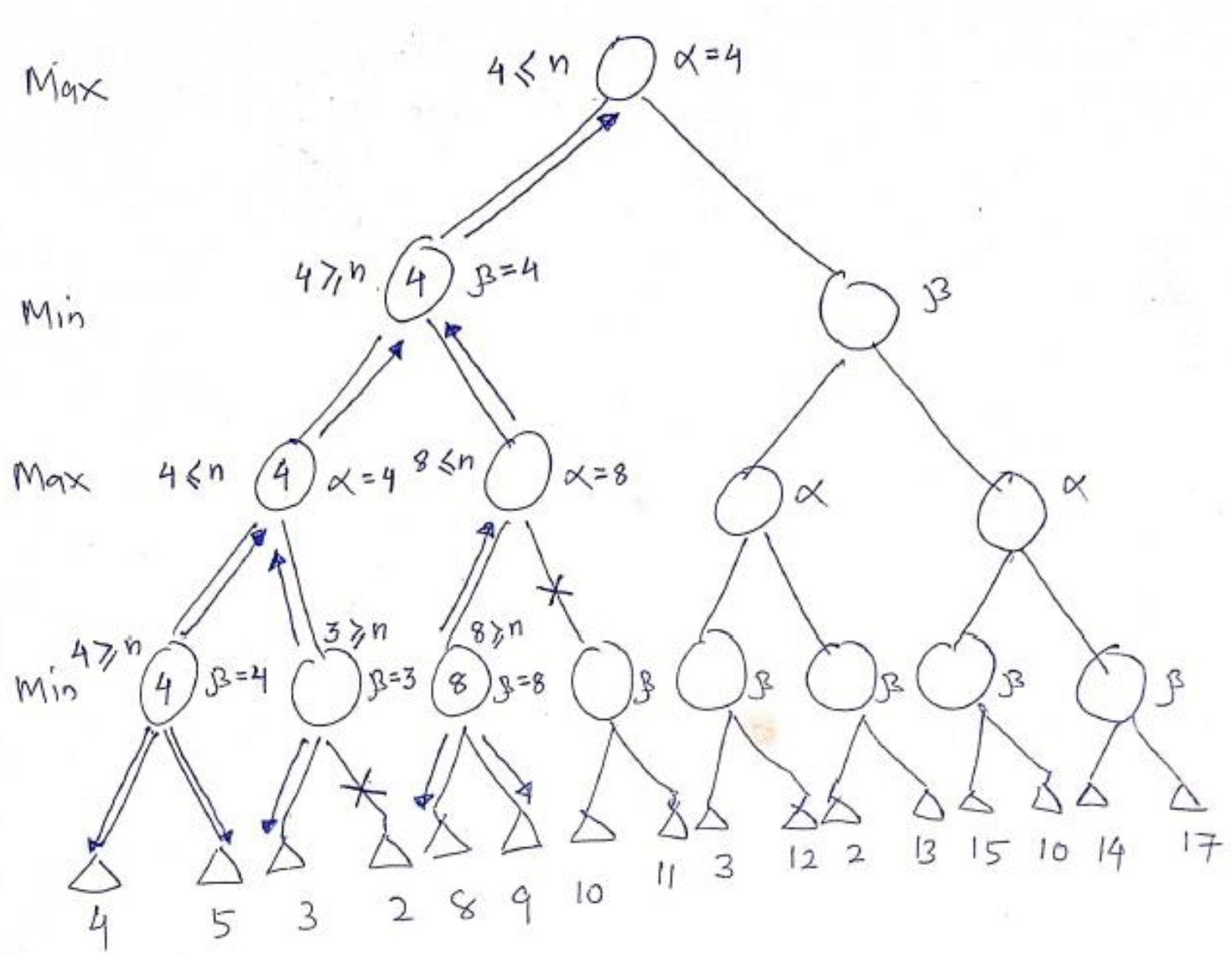
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

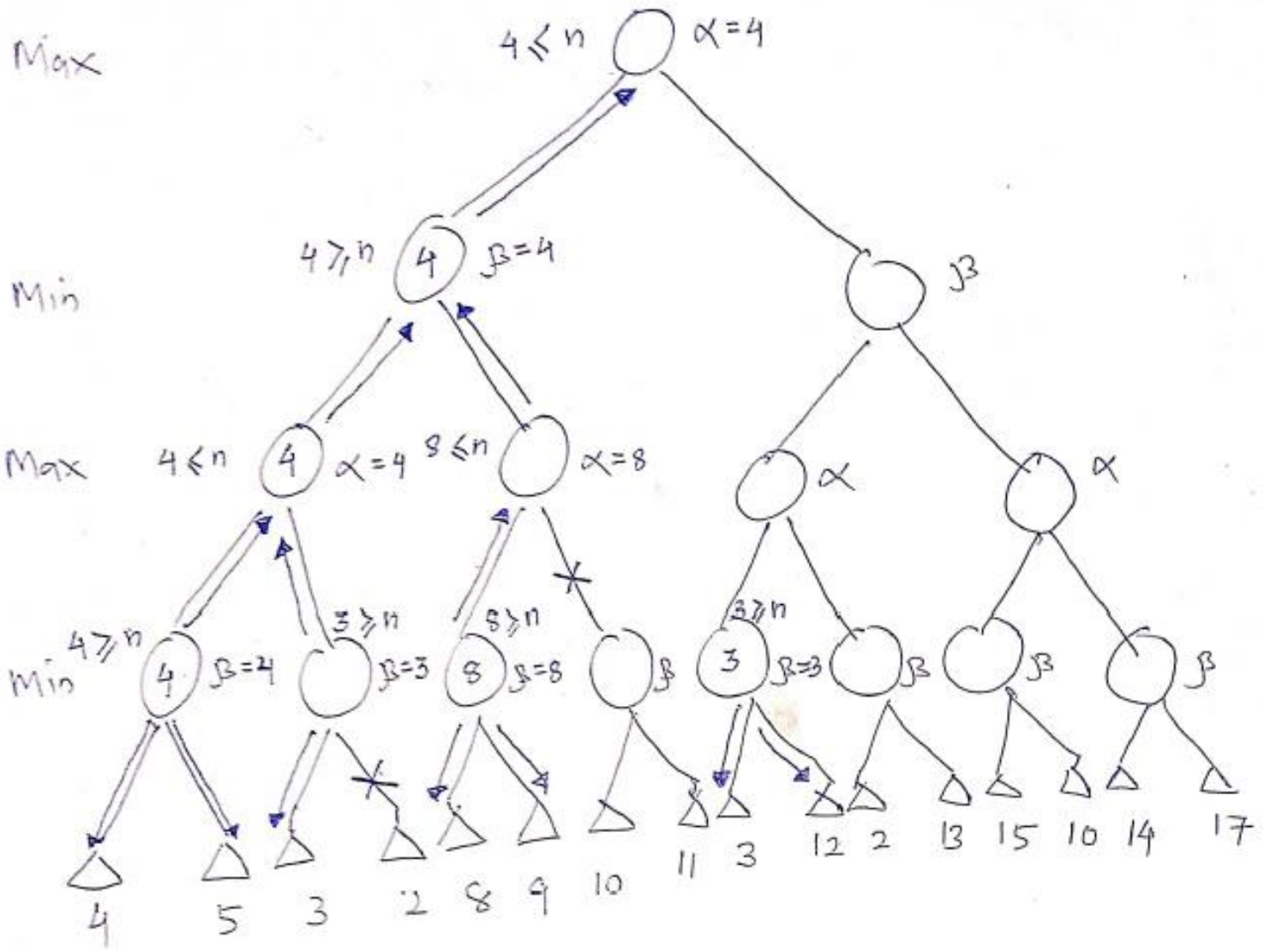
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

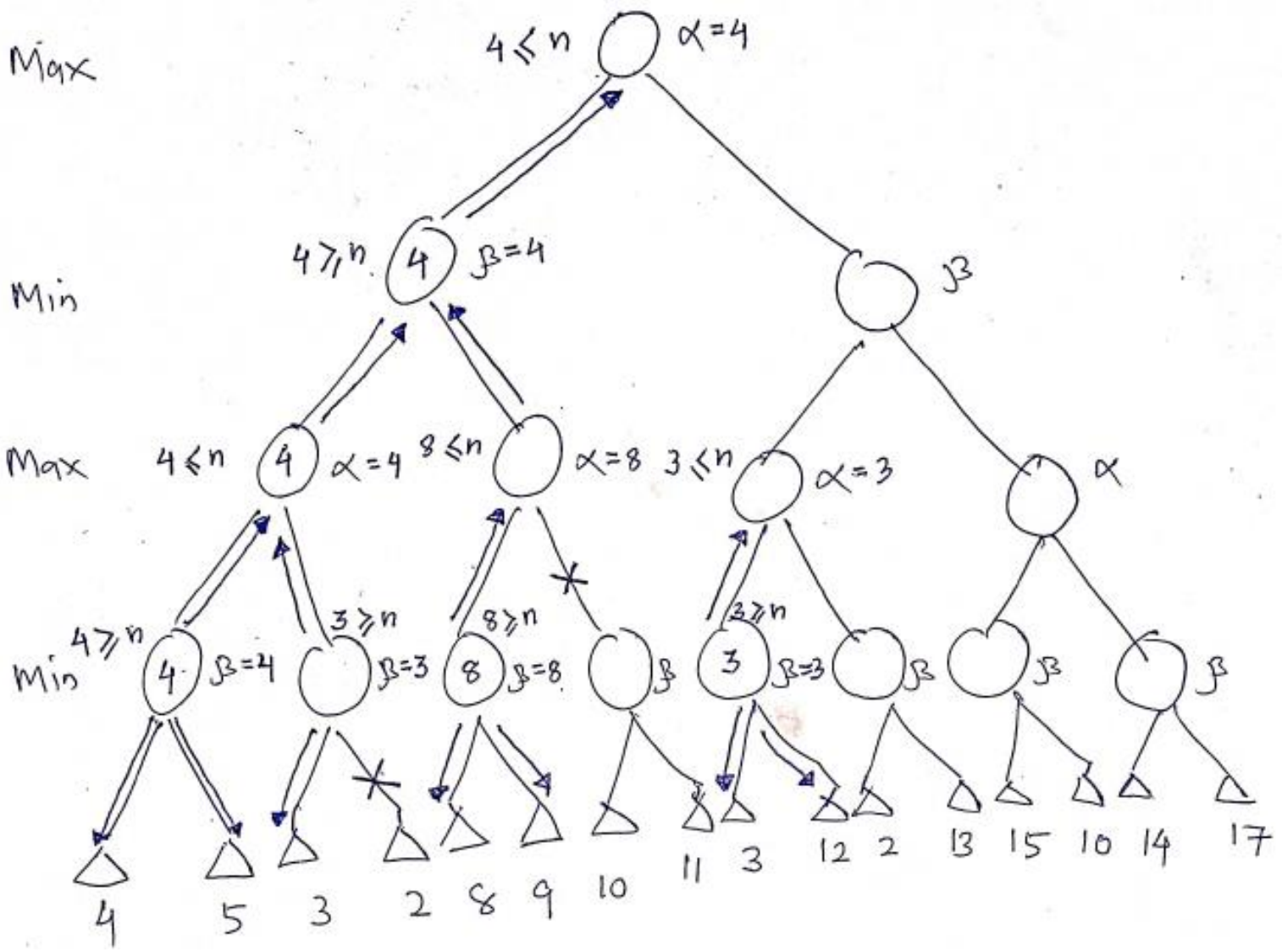
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

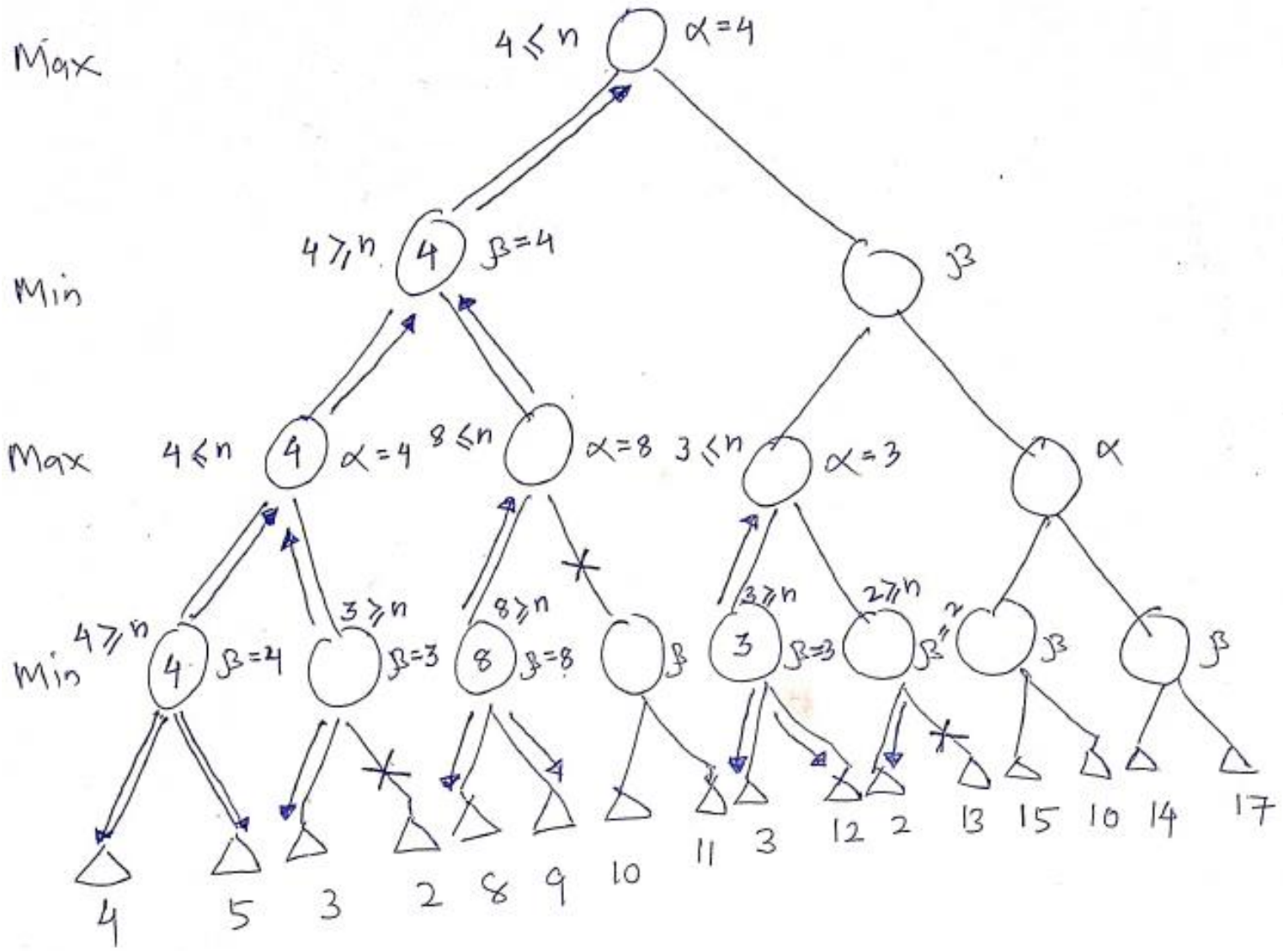
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

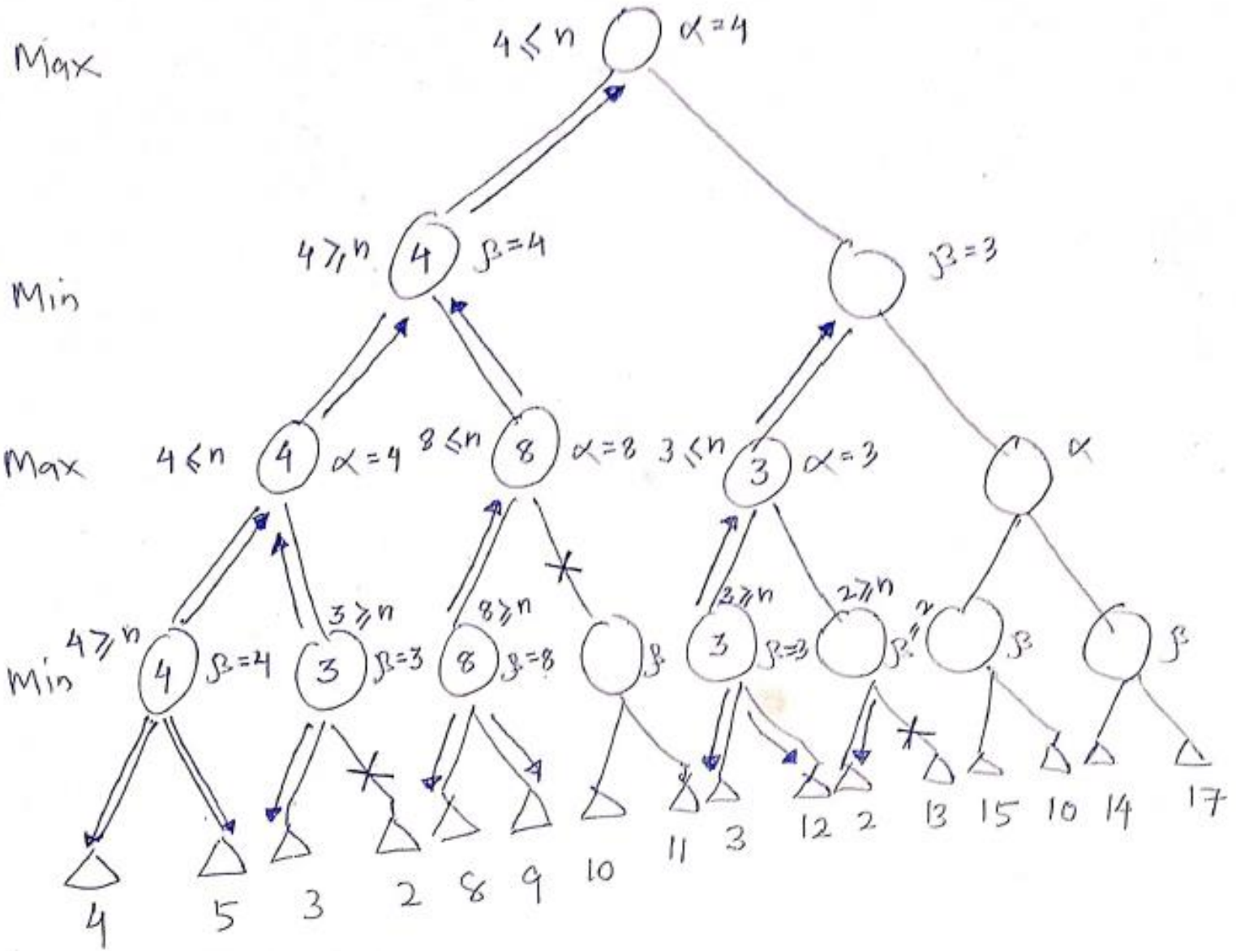
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

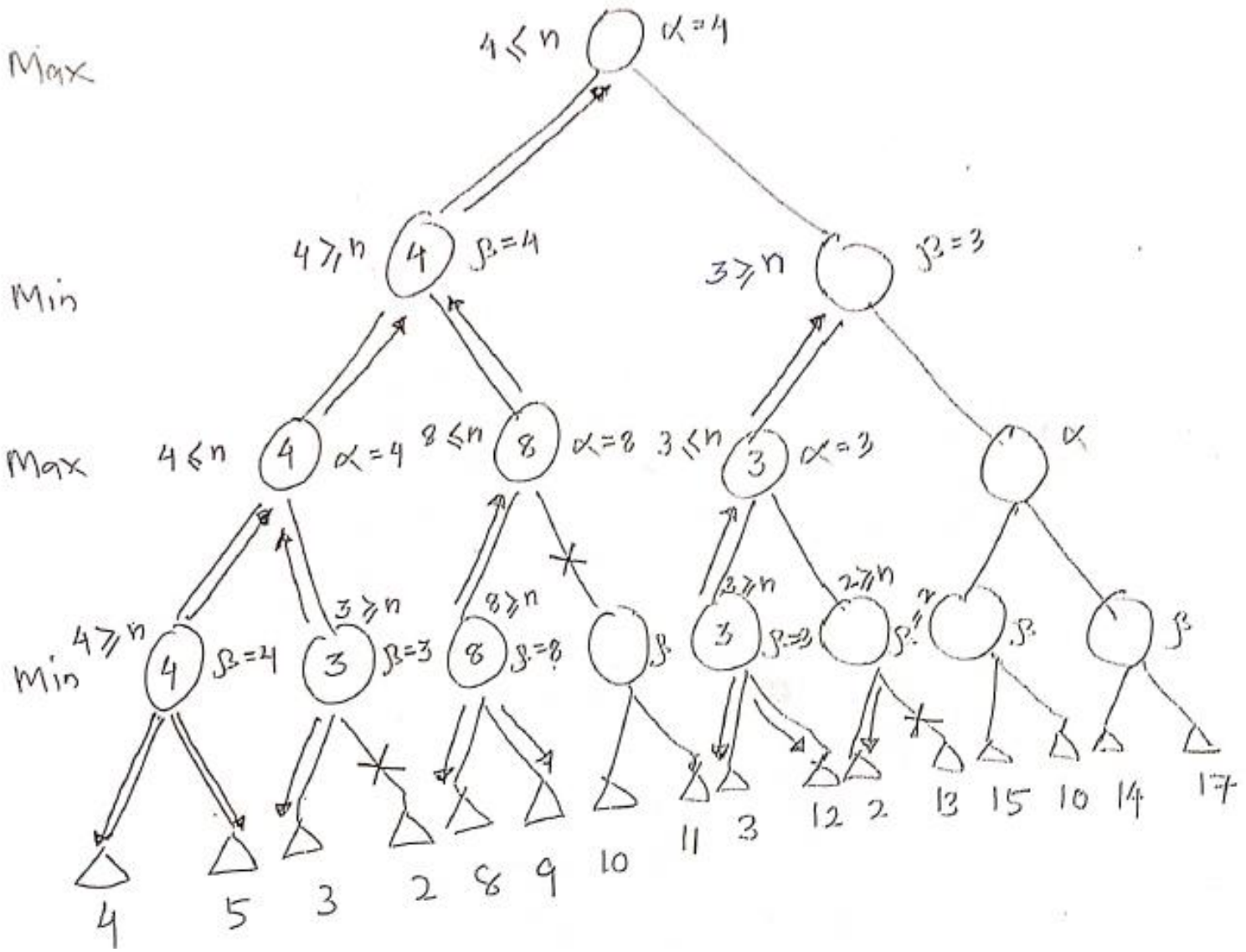
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

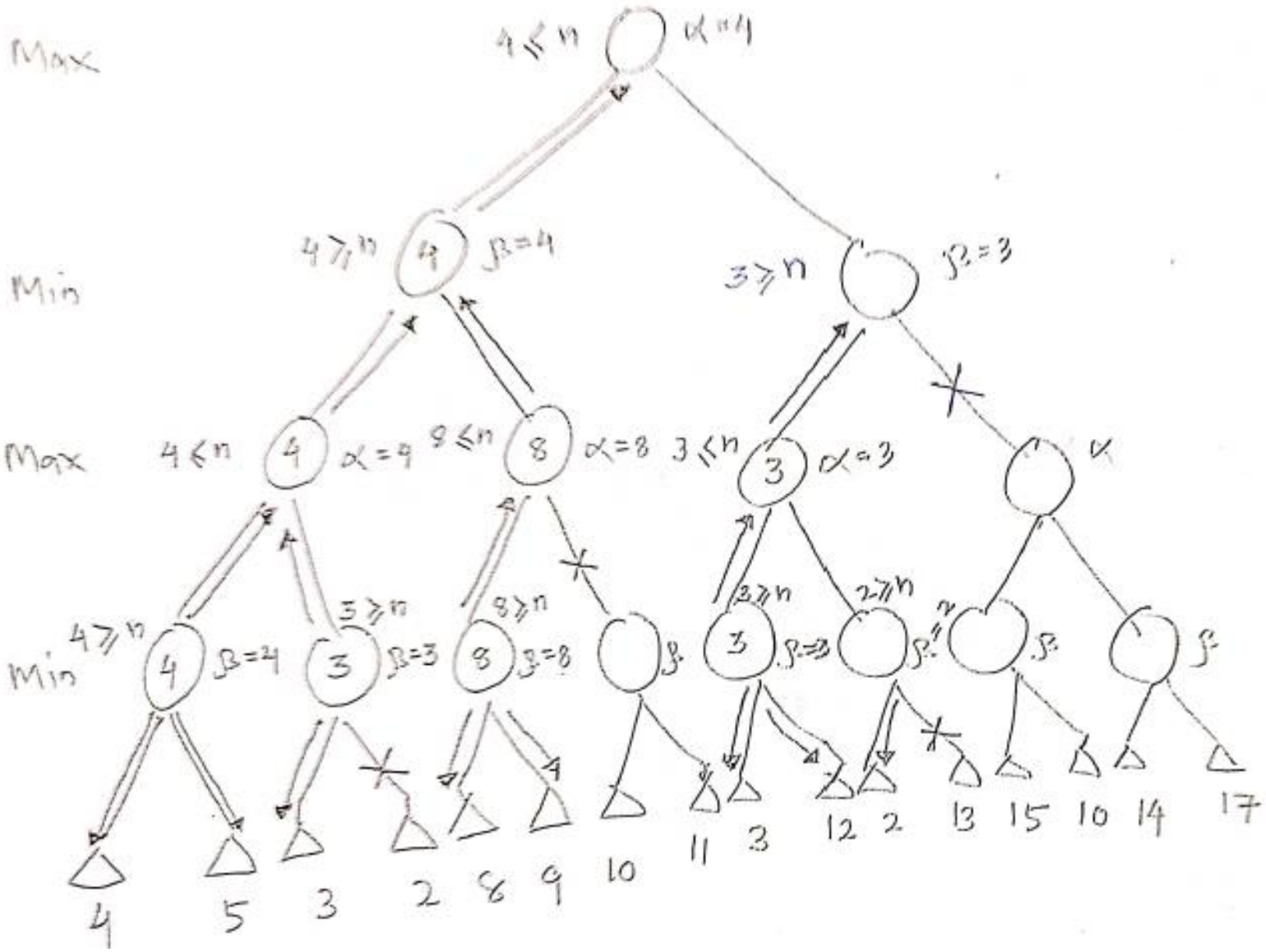
Alpha-beta pruning (Example)



Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

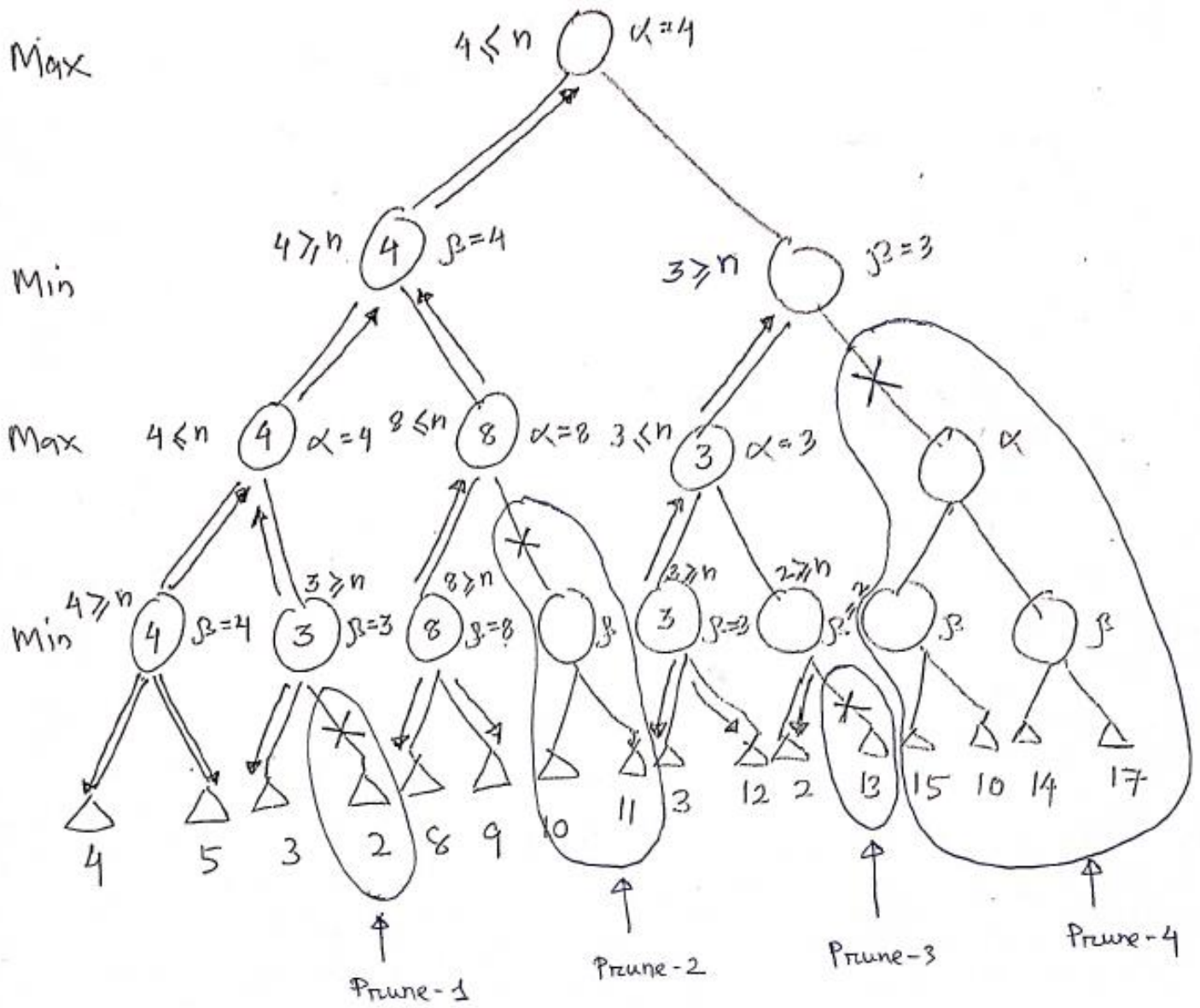
Alpha-beta pruning (Example)



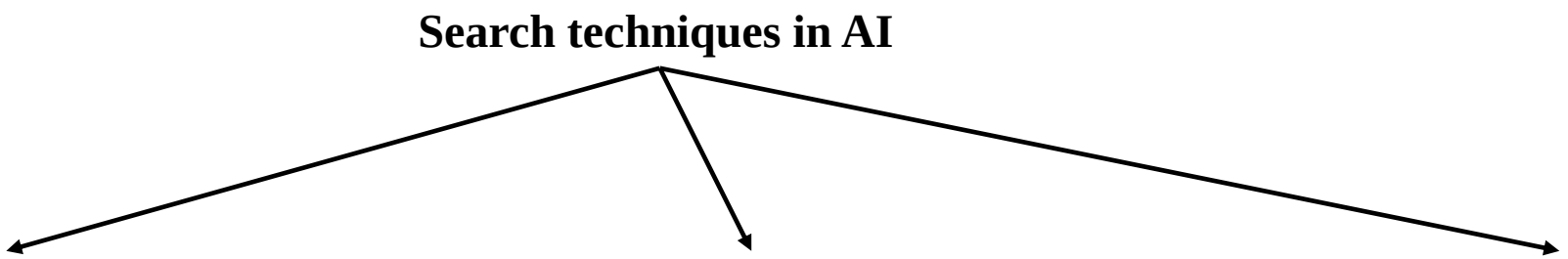
Game Playing and Adversarial search algorithm (Informed search)

Game as a Search

Alpha-beta pruning (Example)



Problem solving by search:



Uninformed search (blind search) algorithms:

Breadth-first search
Uniform cost search
Depth-first search & Depth-limited search
Iterative deepening search

Informed search algorithms:

Greedy best-first search
A*
AO*
Hill Climbing,
Simulated Annealing

Adversarial search algorithms:

MIN-MAX Algorithm
Alpha-Beta Cut-off algorithm