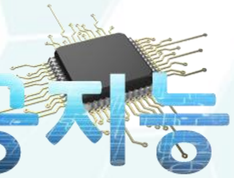


인공지능의 기초

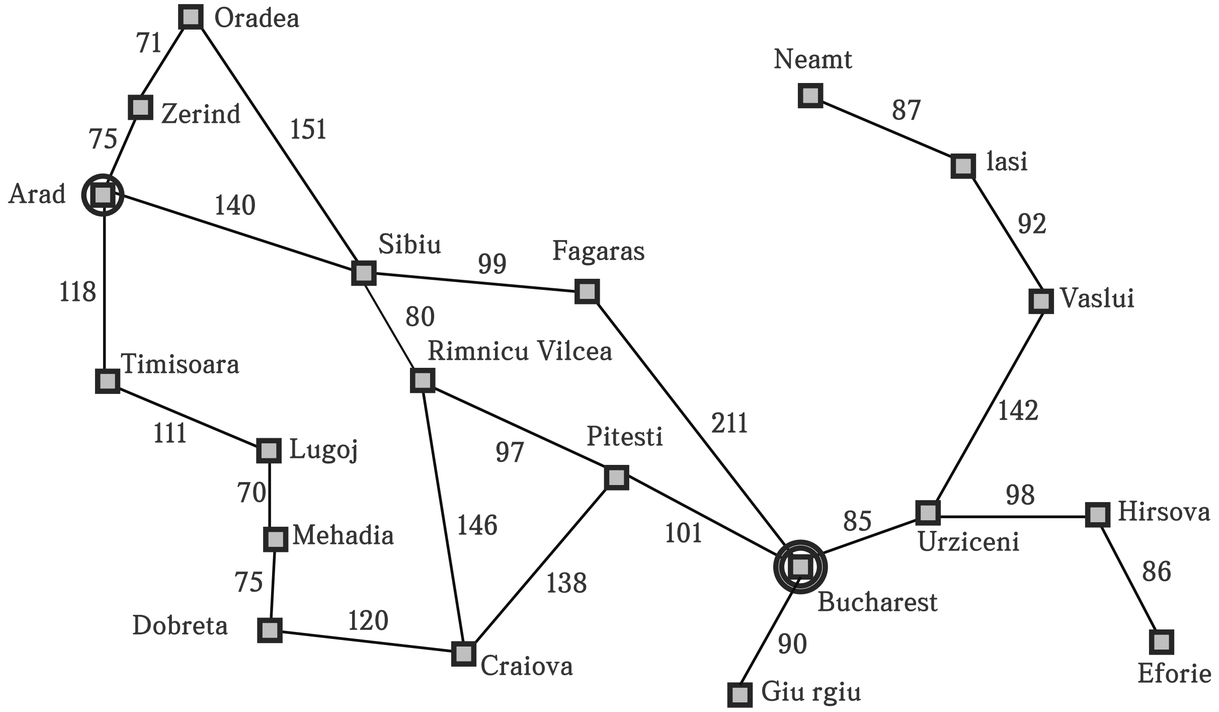
휴리스틱 탐색

Best-First Search

- **Informed search strategy: uses problem-specific knowledge beyond the definition of the problem itself**
  - Can find solutions more efficiently than an uninformed strategy
- **Idea: use an evaluation function $f(n)$ at each node**
  - Estimate of desirability
  - Expand the most desirable unexpanded node
  - Implementation: order the nodes in fringe in a decreasing order of desirability
- **Special cases:**
  - Greedy best-first search
  - A* search

MEMO

Romania with Step Costs in km



Straight-line distance to Bucharest

Arad	366	Fagaras	176	Mehadia	241	Sibiu	253
Bucharest	0	Giurgiu	77	Neamt	234	Timisoara	329
Craiova	160	Hirsova	151	Oradea	380	Urziceni	80
Dobreta	242	Iasi	226	Pitesti	10	Vaslui	199
Eforie	161	Lugoj	244	Rimnicu Vilcea	193	Zerind	374

MEMO

Greedy Best-First Search

- **Evaluation function $f(n) = h(n)$ (heuristic)**
  - $h(n)$: estimated cost of the cheapest path from node n to a goal
  - e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- **Greedy best-first search expands the node that appears to be closest to goal**

MEMO

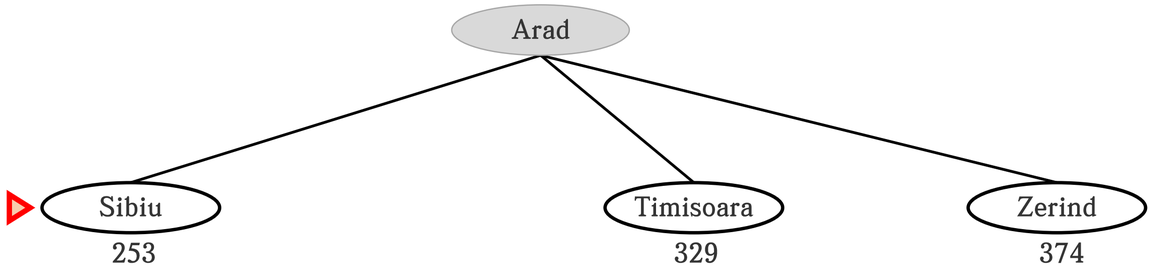
Greedy Best-First Search Example

Straight-line distance to Bucharest

Arad	366	Fagaras	176	Mehadia	241	Sibiu	253
Bucharest	0	Giurgiu	77	Neamt	234	Timisoara	329
Craiova	160	Hirsova	151	Oradea	380	Urziceni	80
Dobreta	242	Iasi	226	Pitesti	10	Vaslui	199
Eforie	161	Lugoj	244	Rimnicu Vilcea	193	Zerind	374

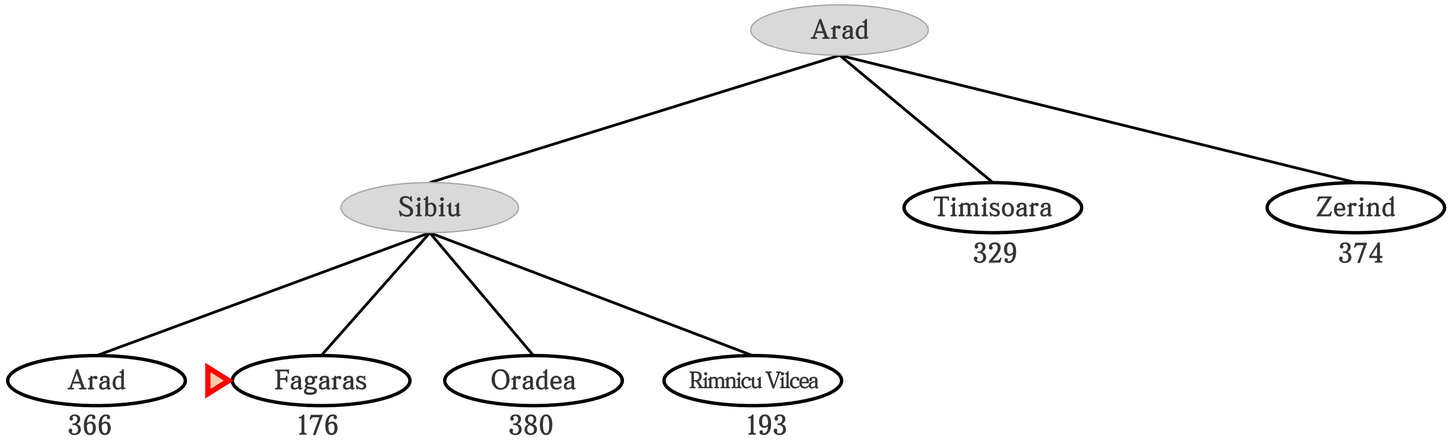
MEMO

Greedy Best-First Search Example



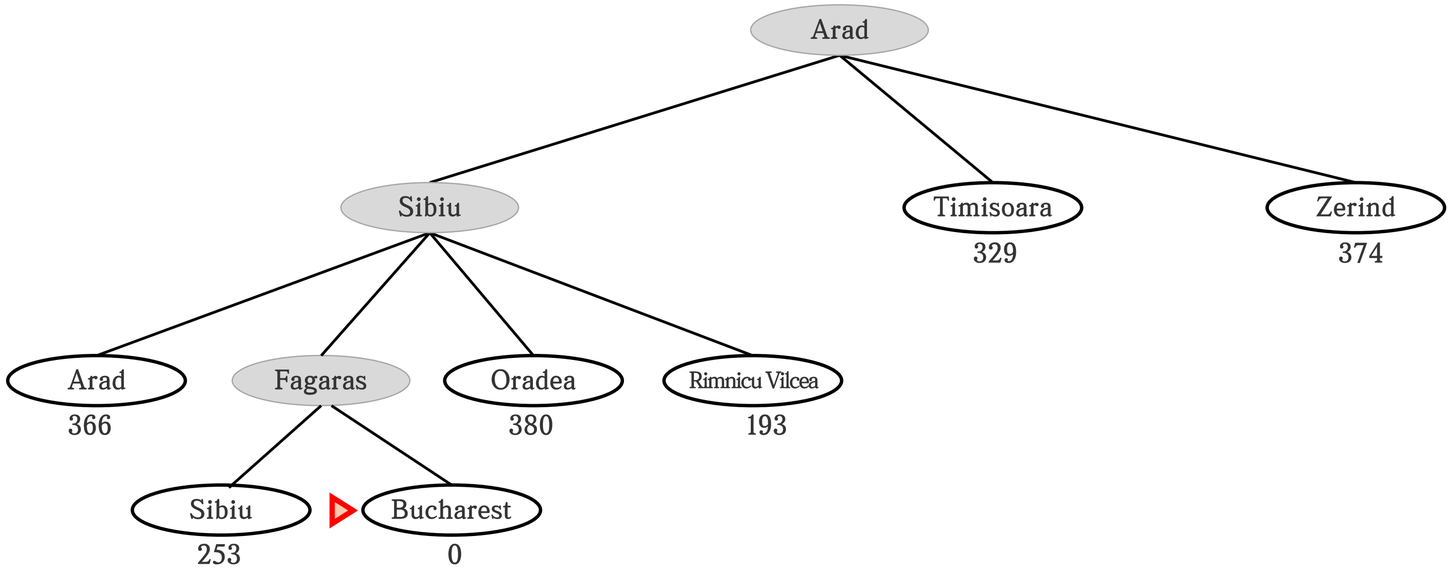
MEMO

Greedy Best-First Search Example



MEMO

Greedy Best-First Search Example



MEMO

A* Search (A-star)

- **Evaluation function $f(n) = g(n) + h(n)$**
  - $g(n)$: cost so far to reach n
  - $h(n)$: estimated cost from n to goal
  - $f(n)$: estimated total cost of path through n to goal

- **Idea: avoid expanding paths that are already expensive**

- **Heuristic function**
  - A way to inform the search about the direction to a goal
  - May be a solution to a simplified problem
  - A trade-off between (i) the **amount of work** it takes to derive a heuristic value for a node and (ii) **how accurately** the heuristic value of a node measures the actual path cost to a goal

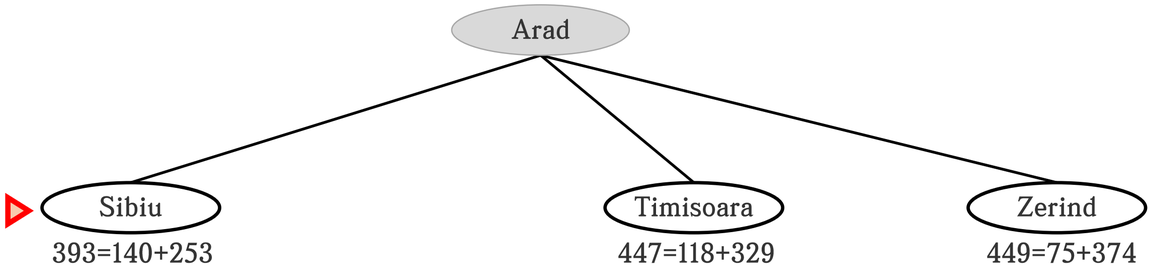
MEMO

A* Search Example

366=0+366

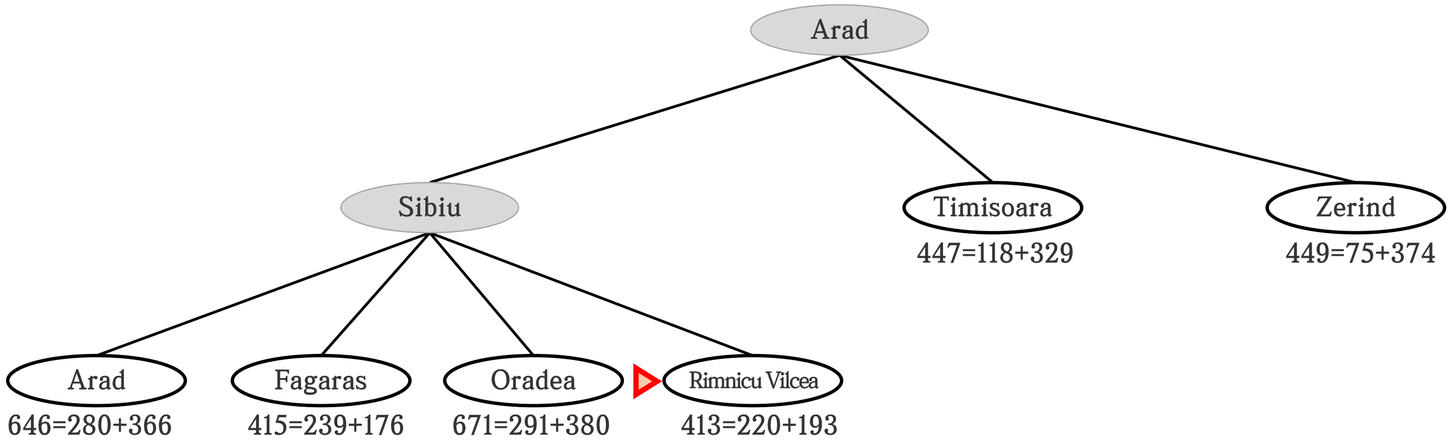
MEMO

A* Search Example



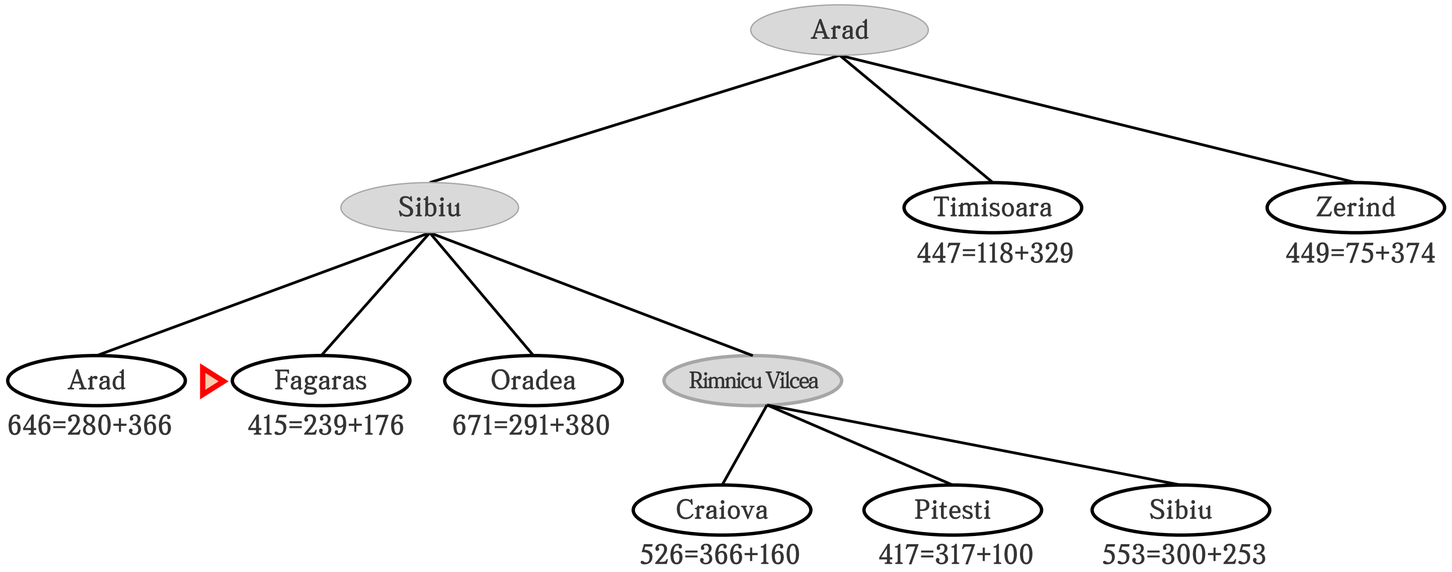
MEMO

A* Search Example



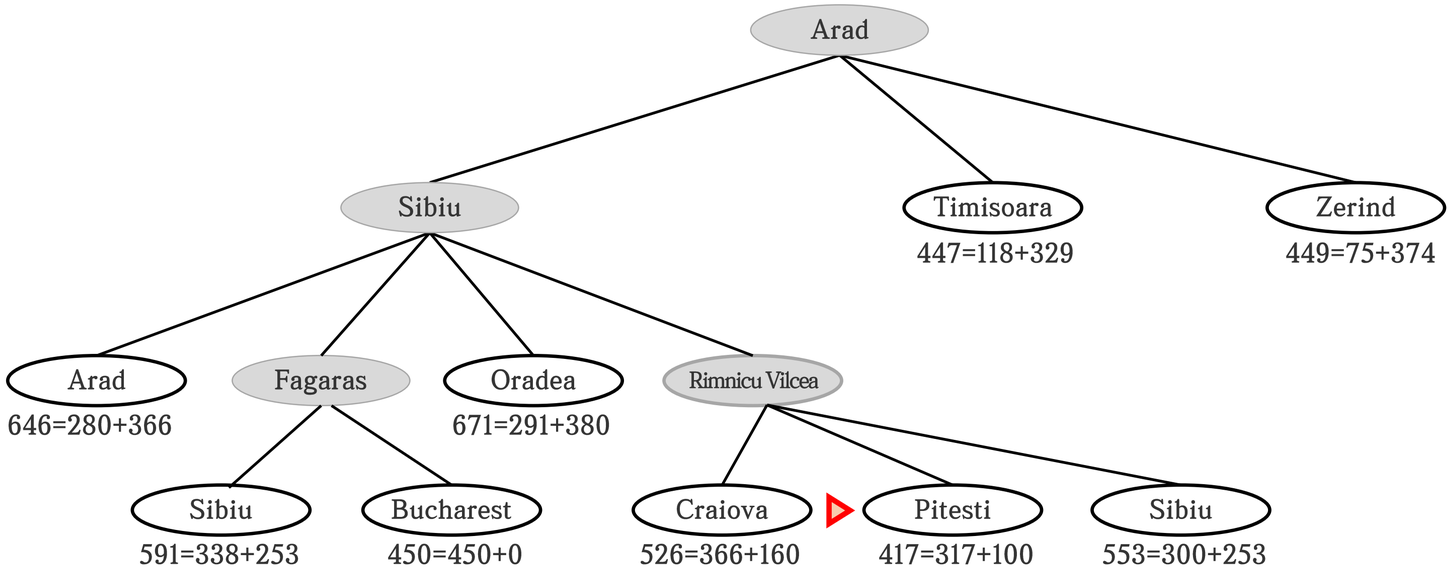
MEMO

A* Search Example



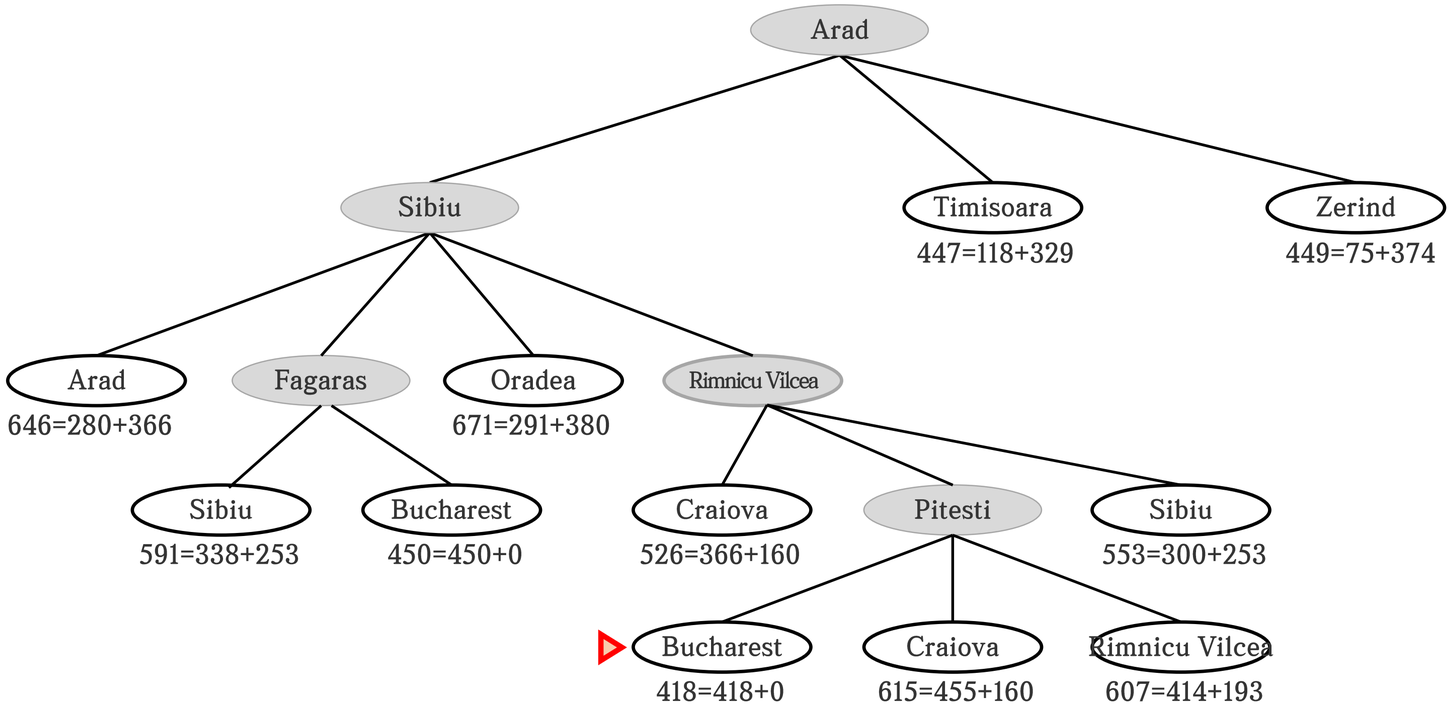
MEMO

A* Search Example



MEMO

A* Search Example



MEMO

Admissible Heuristics

- A heuristic $h(n)$ is *admissible* if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n
  - An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
  - Example: $h_{SLD}(n)$ (never overestimates the actual distance)

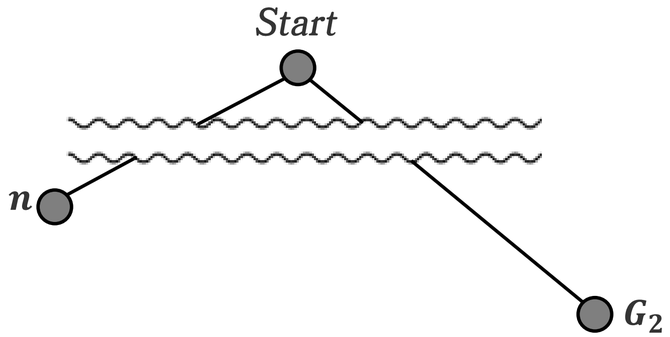
Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

MEMO

Optimality of A*

Proof

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



Then

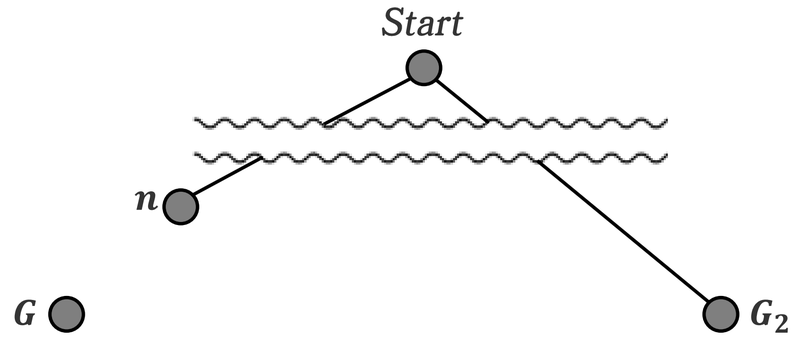
- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

MEMO

Optimality of A*

Proof

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.

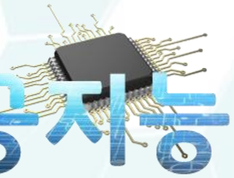


Then

- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since h is admissible
- $h(n) + g(n) \leq h^*(n) + g(n)$
- $f(n) \leq f(G)$ since f is increasing

Hence $f(G_2) > f(n)$. A* never selects G_2 for expansion

MEMO



인공지능의 기초

지역 탐색

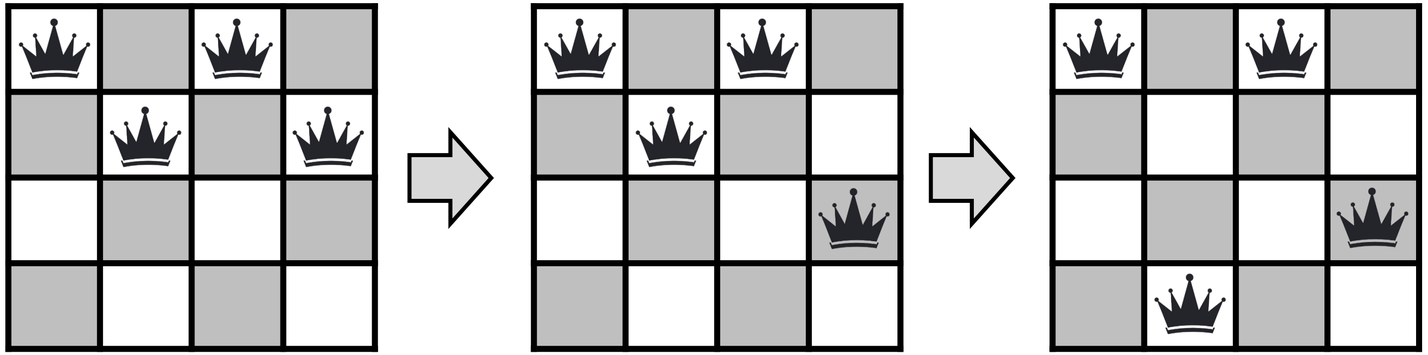
Local Search Algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
  - State space = a set of "complete" configurations
  - Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
  - Keep a single "current" state, try to improve it (by moving only to neighbors)
- Two advantages
  - Use very little memory – usually a constant amount
  - Can often find reasonable solutions in large or infinite (continuous) state spaces

MEMO

Example: n-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



MEMO

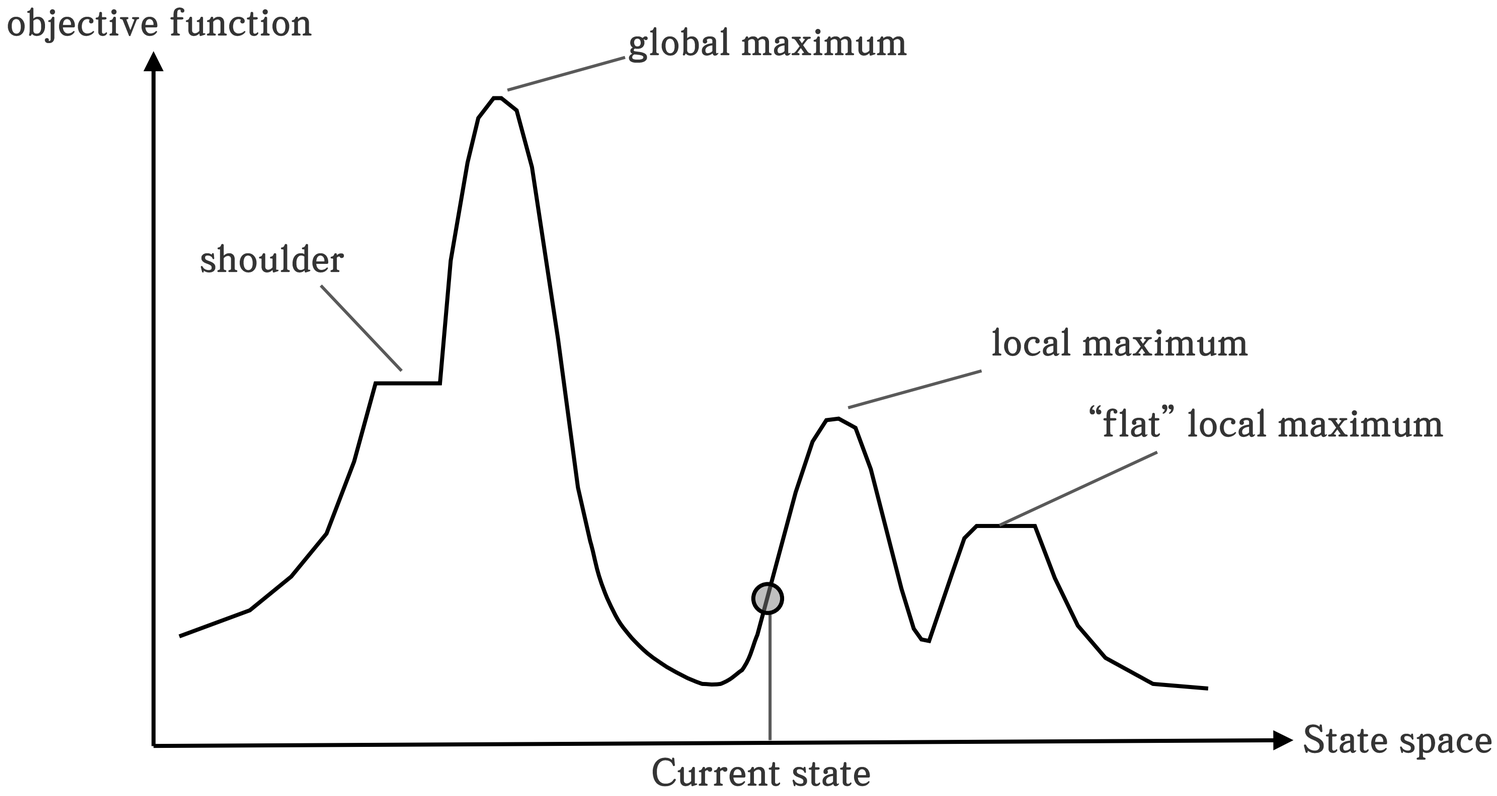
Hill-Climbing Search

- **Like climbing Everest in thick fog with amnesia**
- **The most basic local search technique**
  - Steepest ascent: at each step the current node is replaced by the best neighbor
  - Greedy local search: grabs a good neighbor state without thinking ahead about where to go next

MEMO

Hill-Climbing Search

- Problem: depending on initial state, can get stuck in local maxima



MEMO

Hill-Climbing Search: 8-queens Problem

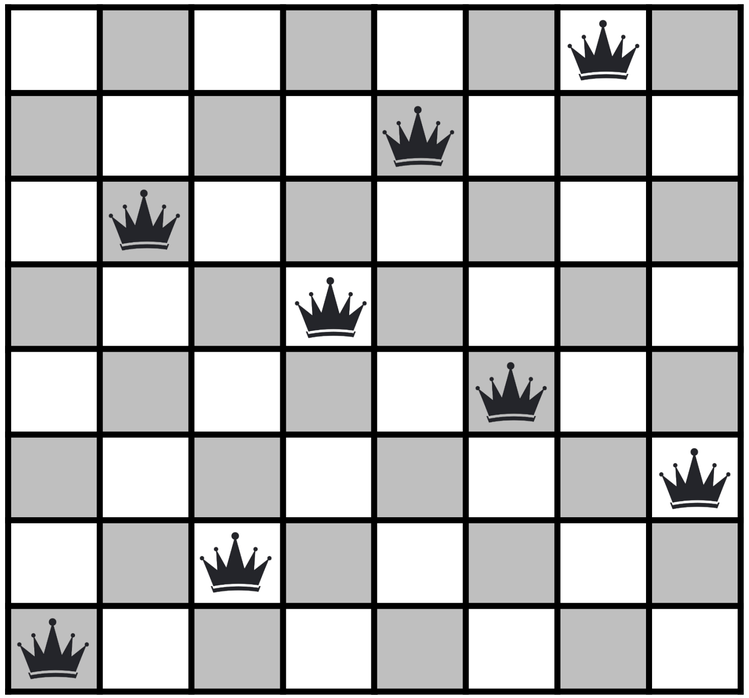
- h = number of pairs of queens that are attacking each other, either directly or indirectly
  - ▶ $h = 0$ only at the global minimum (i.e. perfect solution)
  - ▶ $h = 17$ below

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

MEMO

Hill-Climbing Search: 8-queens Problem

- h = number of pairs of queens that are attacking each other, either directly or indirectly
  - ▶ A local minimum with $h = 1$ (every successors has a higher cost)



MEMO

Simulated Annealing Search

- **Idea: escape local maxima by allowing some bad moves but gradually decrease their frequency**
  - A hill-climbing algorithm never makes downhill
  - Cannot escape from a local maximum
  - Hill-climbing + random walk (for efficiency and completeness)

- **Intuition with gradient descent**
  - Down a ball into a bumpy surface; then it comes to rest at a local minimum
  - Shake the surface to bounce the ball out
  - Start by shaking hard (at a high temperature), then gradually reduce the intensity of the shaking (lower the temperature)

MEMO

Simulated Annealing Search

Algorithm

- Instead of picking the best move, it picks a random move
- If the move improves the situation, it is always accepted
- Otherwise, less likely to be accepted (i) $|\Delta E|$ is high, (ii) temperature T is low

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```

MEMO