Problem Solving & Search

Informatics Engineering Study Program
School of Electrical Engineering and Informatics

Institute of Technology Bandung

Contents

- Review
- Problem Solving
- Example of Problem
- Formal Definition
- Search

Review

- What is AI → 4 approaches
  - For now we use 4th approach (acting rationally)
  - Rationality ≠ omniscience ≠ success
  - Limited rationality
- Intelligent Agent
  - PEAS
  - Task Environment :
    - Accessible (vs. Inaccessible) / Fully (vs partially) observable
    - Deterministic (vs. Non-Deterministic/ Stochastic)
    - Static (vs. Dynamic)
    - Discrete (vs. Continuous)
    - Episodic vs Sequential (non-episodic),
    - Single vs Multi agent
  - Agent Level

Problem Solving Agent

- Agent design:
  - formulate problem → search solution → execute
  - Task Environment: Remember PEAS
- Problem: satisfy goal (goal state)
  - Agent task: find out which sequence of actions will get it to a goal state
  - 5 components of **a problem formulation**:
    - initial states, intermediate states (state spaces),
    - goal state,
    - actions,
    - transition model (new_state = Result(ols_state,action)),
    - Action cost function
- Searching: process of looking for sequence of action
- Solution: sequence of action to goal state

Problem Solving

- Agent knows world dynamics
  - World states, actions
  - [when agent doesn't know → learning]
- World state is finite, small enough to enumerate
  - [when state is infinite →logic]
- World is deterministic
  - [when non-deterministic →uncertainty]
- Agent knows current state
  - [when agent doesn't know → logic, uncertainty]
- Utility for a sequence of states is a sum over path

Few real problems are like this, but this may be a useful abstraction of a real problem → solving problems by searching

Problem: Formal Definition

Problem components:

1. Initial State, State spaces: kota
   - State space forms graph (node: state, arc: action)
2. Goal State
3. Actions
4. Transition Model → S' = Result(S,A) [deterministic]
5. Action Cost Function: (S,A)* ⟶ real
   - Sum of costs: Σ c(S,A)

- Solution: graph path
- Criteria for algorithms:
  - Computation time/space
  - Solution quality

Route Planning

Example: Route Planning in a Map

A map is a graph where nodes are cities and links are roads. This is an abstraction of the real world.

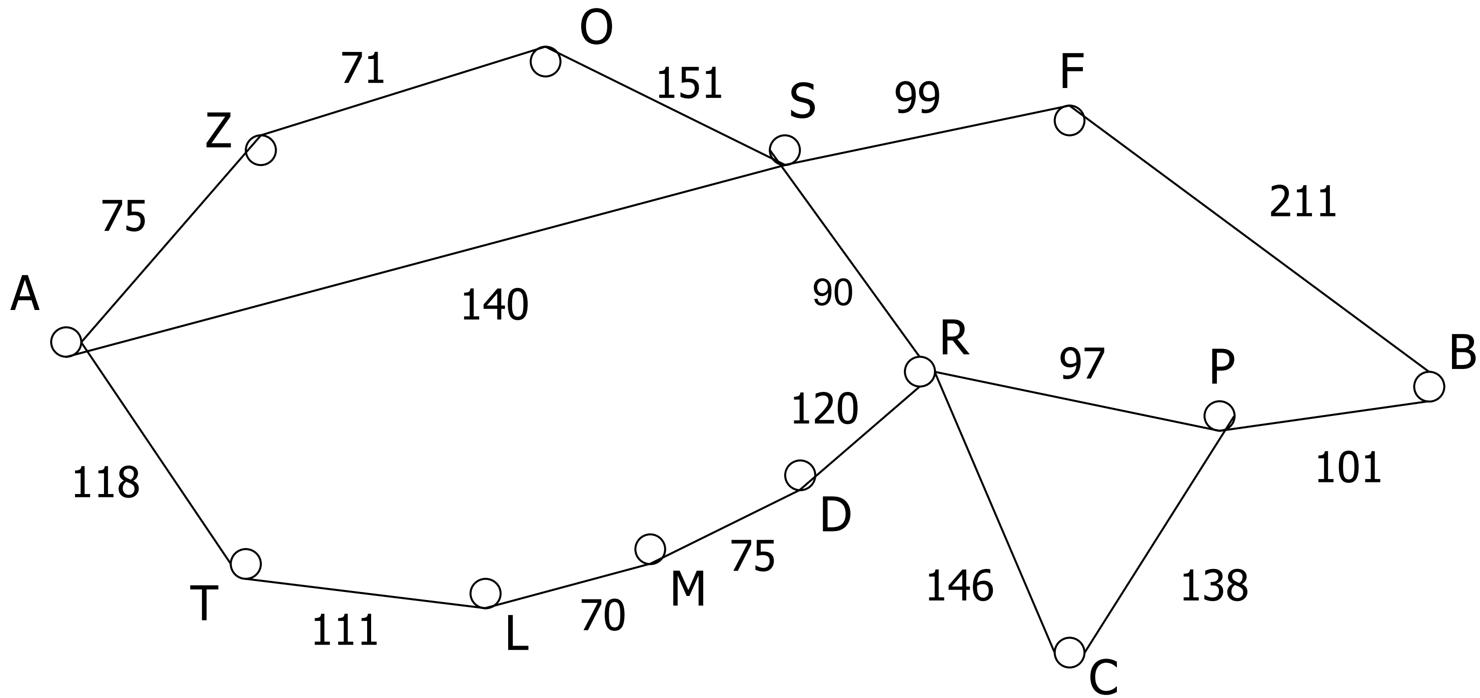
- Map gives world dynamics: starting at city X on the map and taking some road gets you to city Y.

Environment assumptions:

- Static: no change when solving problem
- Discrete: World (set of cities) is finite and enumerable.
- Deterministic: taking a given road from a given city leads to only one possible destination.
- Observable: information is complete
  - We assume current state is known.
- Utility for a sequence of states is usually either total distance traveled on the path or total time for the path.

Search

Source: Russell’s book



S: set of cities
i.s: A (Arad)
g.s: B (Bucharest)
Goal test: s = B ?
Path cost: time ~ distance

Search

- UnInformed/Blind Search
  - Look around, don't know where to find the right answer
  - No additional information beyond that provided in problem definitional
  - Example: DFS, BFS, IDS, UCS , DLS
- Informed Search
  - Heuristic Search
    - Know some information that sometimes helpful
    - Know whether one non-goal state is "more promising" than another
    - Example: Best FS, A*,
- Local Search (for Optimization Problem) → Beyond Classical Search
    - Path to goal is irrelevant
    - Use very little memory
    - Can find reasonable solutions in large or infinite state spaces for which systematic algorithms are suitable
    - Example: Hill-climbing search, simulated annealing search, GA

Search

- It's time to do searching: covering the basic methods really fast.
  - Agenda: a list of states that are waiting to be expand

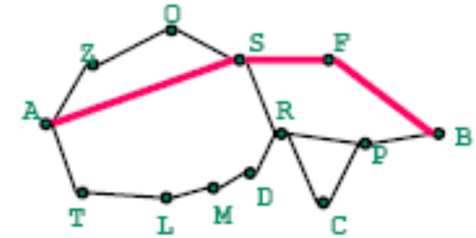
```
{Put start state (initial state) in the agenda}
AddState(Agenda, initial-state)
iterate
  GetState(Agenda, current-state)
stop: isGoal(current-state)
  if not isExpanded(current-state) then
     {put children in agenda}
     ExpandState(current-state, Agenda)
```

Search

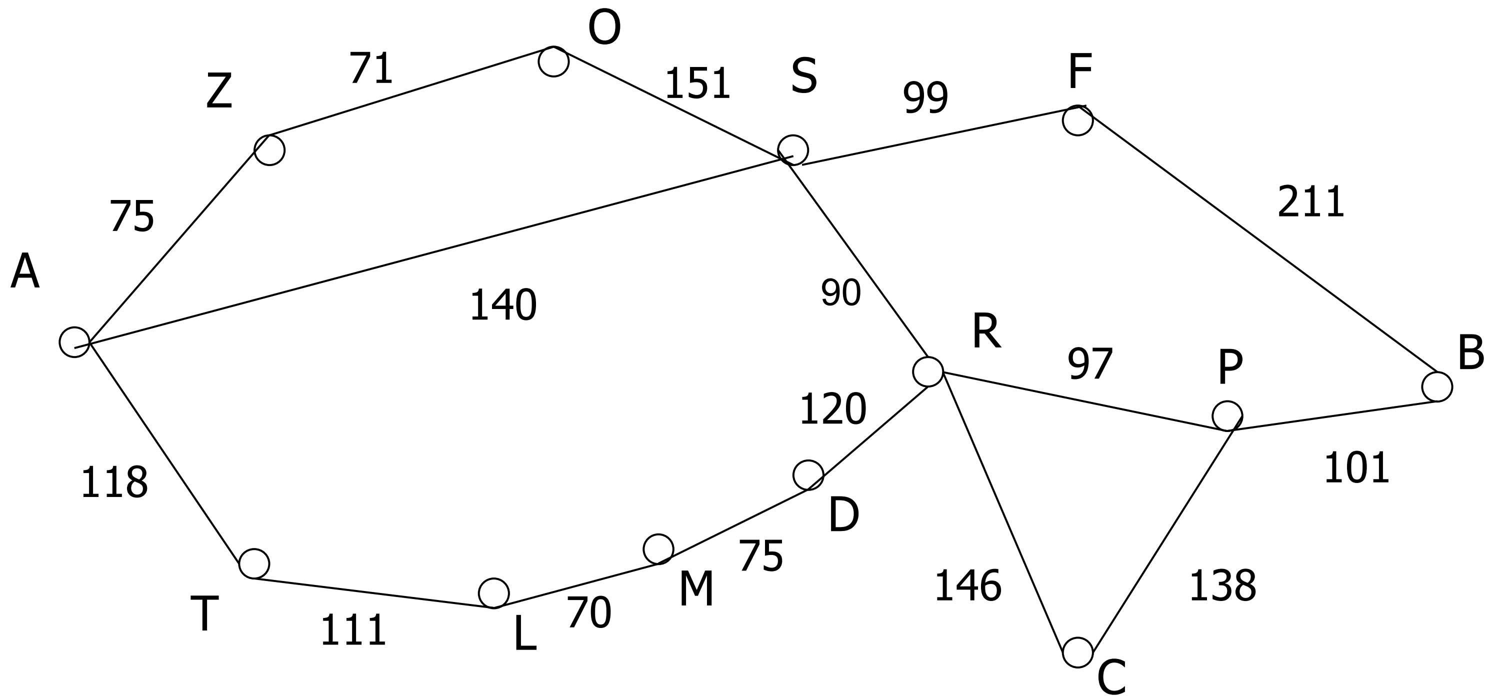
- **It's time to do searching: covering the basic methods really fast.**
  - Graph search
  - Agenda: a list of states that are waiting to be expand
  - Which state is chosen from the agenda defines the type of search & may have huge impact on effectiveness.

Uninformed Search

Breadth-First Search (BFS)



Treat agenda as a queue (FIFO)



A → Z_A, S_A, T_A → S_A, T_A, O_{AZ} → $T_A, O_{AZ}, O_{AS}, F_{AS}, R_{AS}$ → $O_{AZ}, O_{AS}, F_{AS}, R_{AS}, L_{AT}$ → $O_{AS}, F_{AS}, R_{AS}, L_{AT}$ → F_{AS}, R_{AS}, L_{AT} → R_{AS}, L_{AT}, B_{ASF} → $L_{AT}, B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}$ → $B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}, M_{ATL}$ →

Stop: B=goal, path: A → S → F → B, path-cost = 450

Breadth-First Search (BFS)

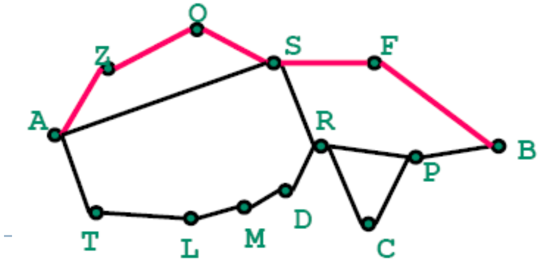
- Treat agenda as a queue (FIFO)
- Let's see what would happen if we did BFS on the graph G_1:
  - Start with initial state: A
  - Get A, expand it, add Z, S, T ⇒ Z_A, S_A, T_A
  - Get Z, expand it, add O ⇒ S_A, T_A, O_{AZ}
  - Get S, expand it, add O, F, R ⇒ $T_A, O_{AZ}, O_{AS}, F_{AS}, R_{AS}$
  - Get T, expand it, add L ⇒ $O_{AZ}, O_{AS}, F_{AS}, R_{AS}, L_{AT}$
  - Get O, expand it, nothing to add (already expanded) done twice!
  - Get F, expand it, add B ⇒ R_{AS}, L_{AT}, B_{ASF}
  - Get R, expand it, add D, C, P ⇒ $L_{AT}, B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}$
  - Get L, expand it, add M ⇒ $B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}, M_{ATL}$
  - Pop B, it is the goal state, and terminate.

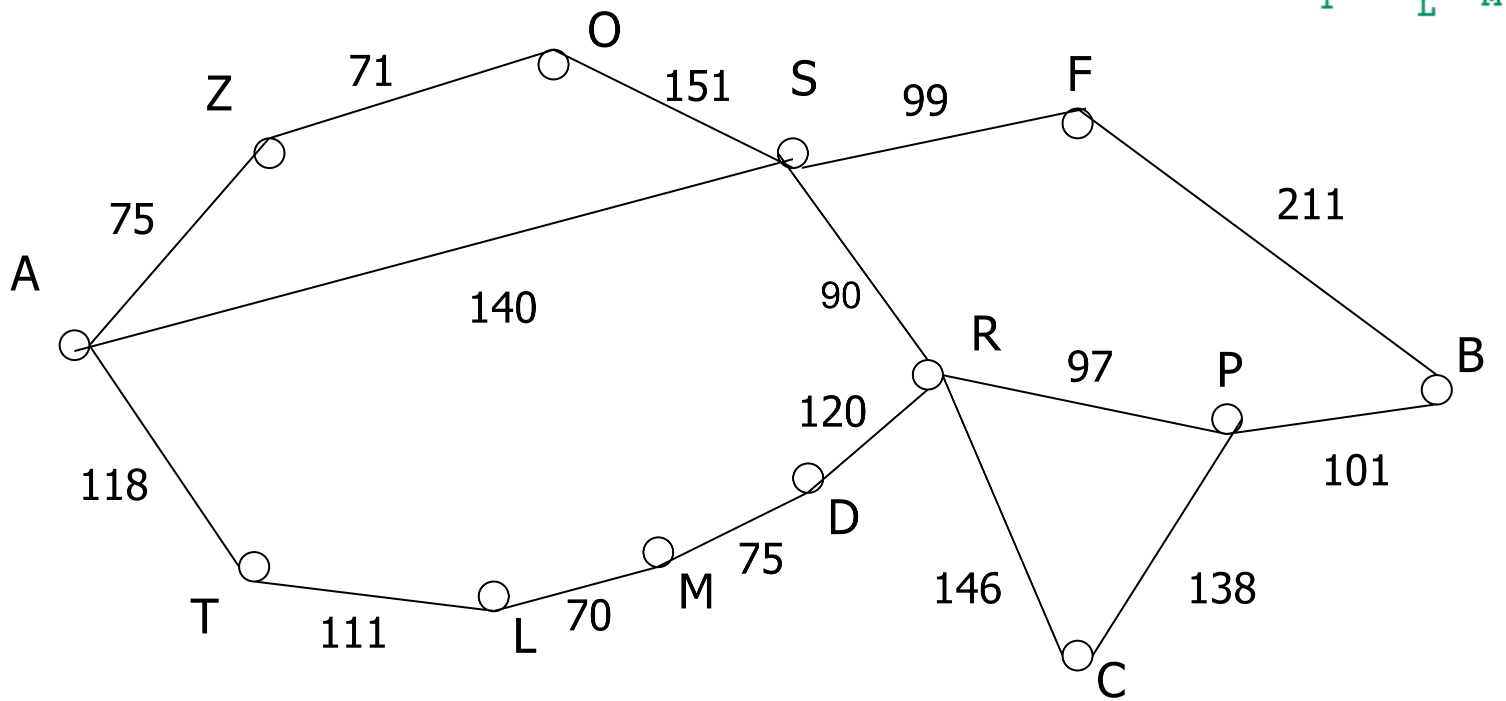
⇒ The RESULT is B_{ASF} with path: A, S, F, B

⇒ Path cost: 450

Depth-First Search (DFS)

Treat agenda as a stack (LIFO)





$A \rightarrow Z_A, S_A, T_A \rightarrow O_{AZ}, S_A, T_A \rightarrow S_{AZO}, S_A, T_A \rightarrow F_{AZOS}, R_{AZOS}, S_A, T_A \rightarrow B_{AZOSF}, R_{AZOS}, S_A, T_A \rightarrow$

Stop: B=goal, path: A → Z → O → S → F → B, path-cost = 607

Depth-Limited Search (DLS)

- BFS finds min-step path but requires exponential space
- DFS is efficient in space, but has no path-length guarantee
  - DFS: can make a wrong choice and get stuck going down a very long (or even infinite) path when a different choice would lead to a solution near root of the search tree
- Solution: DFS-limited search
  - DFS with a predetermined depth limit l
  - Nodes at depth l are treated as if they have no successors.
  - Problem: the shallowest goal is beyond the depth limit
  - Depth limit can be based on knowledge of the problem

DLS Algorithm

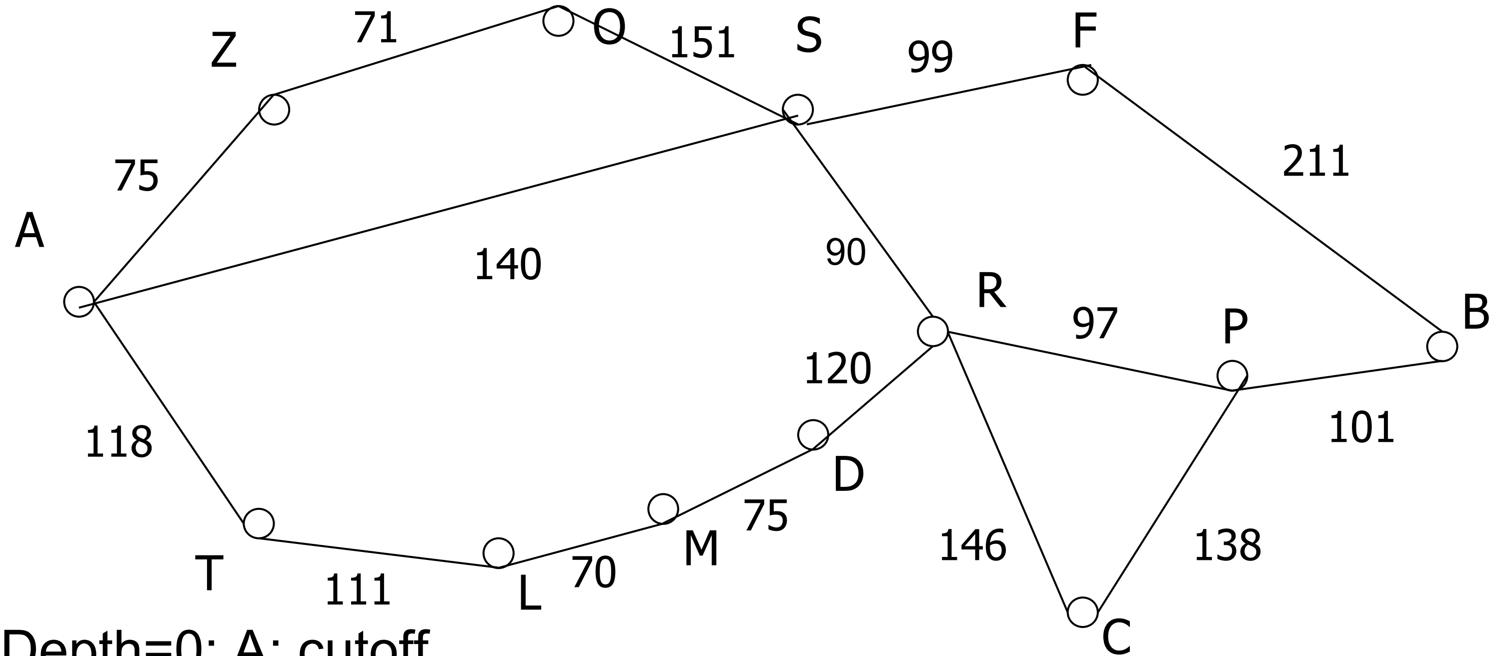
```
Function DLS (problem, limit) returns solution/ cutoff/
  failure
→ rec_DLS(make_node(init_state),problem,limit)

Function Rec_DLS (node,problem, limit) returns solution/
  cutoff/ failure
  if isGoal(node) then → solution(node)
  else if limit=0 then → cutoff
  else
      cutoff_occured ← false
      for each action in problem.Actions(node.State) do
         child ← CHILD-Node(problem, node, action)
         result ← rec_DLS(child,problem,limit-1)
         if result=cutoff then cutoff_occured ← true
         else if result≠failure then → result
      if cutoff_occured then → cutoff
      else → failure
```

Iterative Deepening Search (IDS)

- IDS: perform a sequence of DFS searches with increasing depth-cutoff until goal is found
- Assumption: most of the nodes are in the bottom level so it does not matter much that upper levels are generated multiple times.

```
Function Iterative-Deepening_Search(problem) returns
  solution/ failure
  for depth = 0 to ∞ do
       result ← DLS(problem, depth)
       if result ≠ cutoff then → result
```

IDS



Depth=0: A: cutoff

Depth=1: A → Z_A,S_A,T_A → Z_A: cutoff, S_A: cutoff, T_A: cutoff

Depth=2: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → O_{AZ} : cutoff → F_{AS}, R_{AS},T_A → F_{AS} : cutoff → R_{AS} : cutoff → L_{AT} → L_{AT} : cutoff

Depth=3: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → S_{AZO},S_A,T_A → S_{AZO}: cutoff → F_{AS}, R_{AS},T_A → B_{ASF}, R_{AS},T_A → B_{ASF}

Stop: B=goal, path: A → S → F → B, path-cost = 450

Uniform Cost Search (UCS)

- BFS & IDS find path with fewest steps
- If steps ≠ cost, this is not relevant (to optimal solution)
- How can we find the shortest path (measured by sum of distances along path)?
- UCS:
  - Nodes in agenda keep track of total path length from start to that node
  - Agenda kept in priority queue ordered by path length
  - Get shortest path in queue
- Explores paths in contours of total path length; finds optimal path

Uniform Cost Search (UCS)

- Let's see what would happen if we did UCS on the graph G_1:
  - Start with start state: A
  - Remove A, add Z with cost 75, add T with cost 118, add S with cost 140 ⇒ $Z_{A-75}, T_{A-118}, S_{A-140}$
  - Remove Z (the shortest path), add its children: O_{146} ⇒ T_{A-118}, S_{A-140}, O_{AZ-146}
  - Remove T, add L_{229} ⇒ $S_{A-140}, O_{AZ-146}, L_{AT-229}$
  - Remove S, add O_{291}, F_{239}, R_{230} ⇒ $O_{AZ-146}, L_{AT-229}, R_{AS-230}, F_{AS-239}, O_{AS-291}$
  - Remove O, add nothing (already expanded)
  - Remove L, add M_{299} ⇒ $R_{AS-230}, F_{AS-239}, O_{AS-291}, M_{ATL-299}$
  - etc ...
- It seems clear that in the process of removing nodes from the agenda, we're enumerating all the paths in the graph in order of their length from the start state.

Informed Search

Best-first search

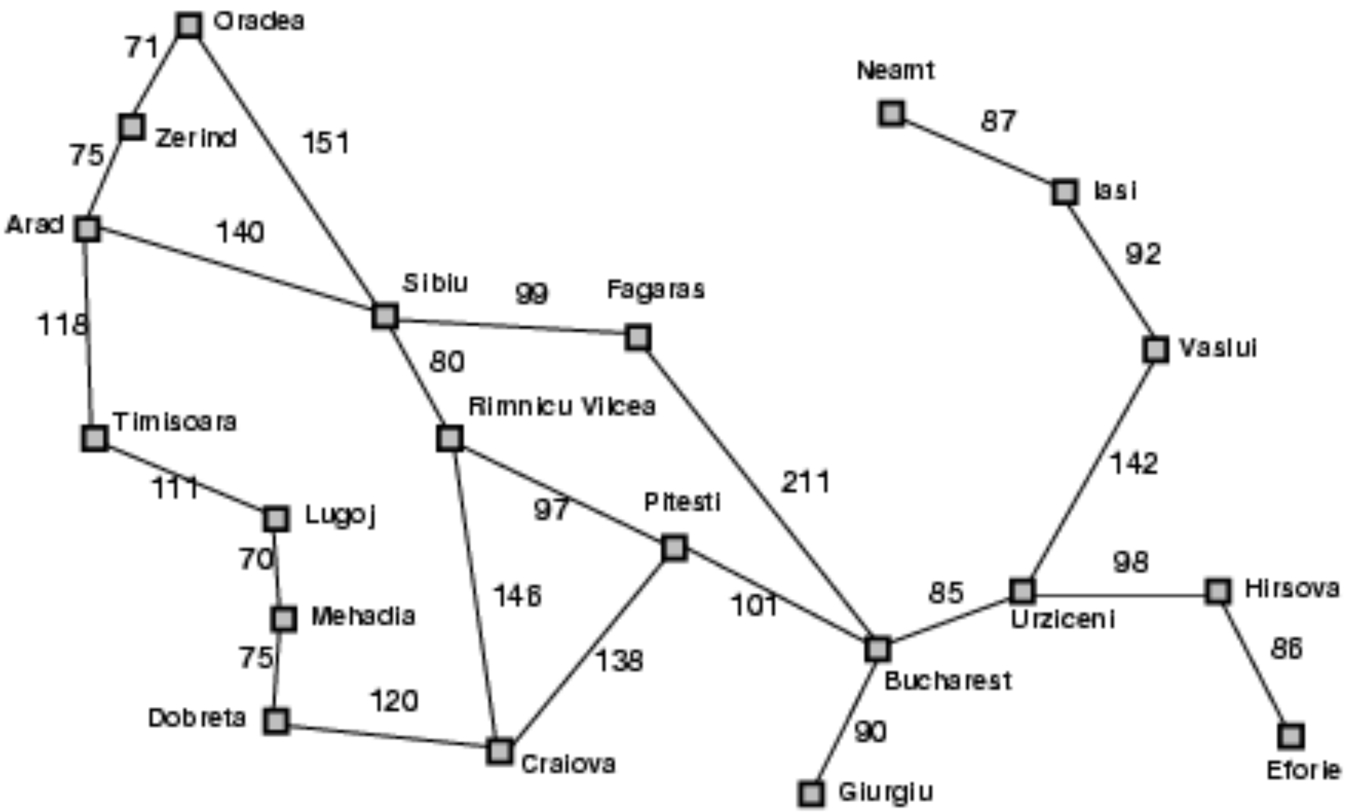
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability“

  - Expand most desirable unexpanded node

- Implementation:
- Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Romania with step costs in km



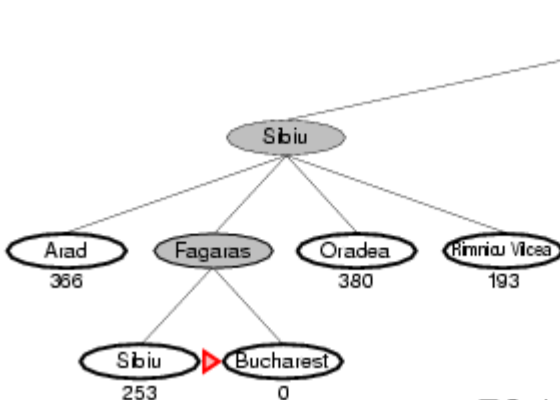
Straight–line distance to Bucharest

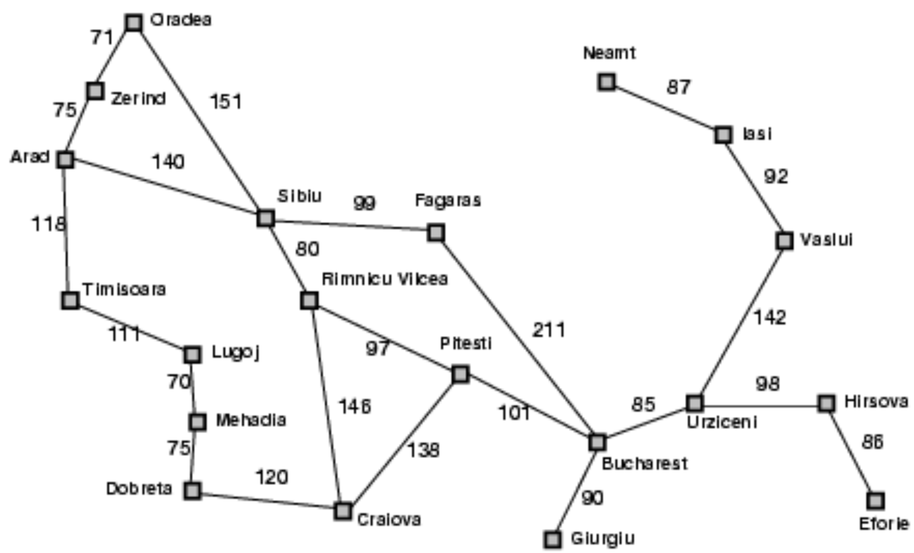
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from *n* to *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

Greedy best-first search example





Straight–line distance
to Bucharest

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

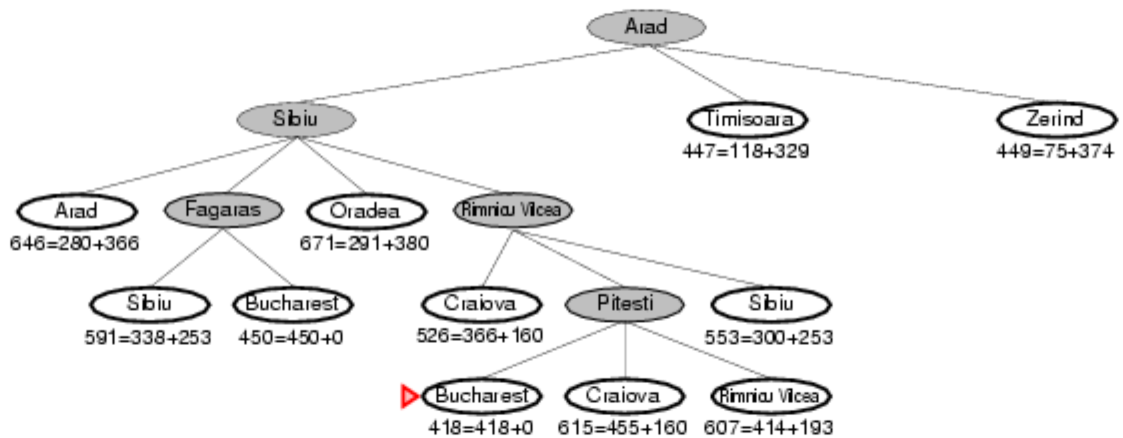
A^* search

- Idea: avoid expanding paths that are already expensive

- 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

A* search example

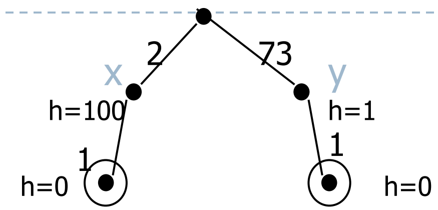


Admissible heuristics

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

Admissibility

- What must be true about h for A* to find optimal path?
- A* finds optimal path if h is admissable; h is admissible when it never overestimates.
- In this example, h is not admissible.
- In route finding problems, straight-line distance to goal is admissible heuristic.



g(X)+h(X)=2+100=102

G(Y)+h(Y)=73+1=74

Optimal path is not found!

Because we choose Y, rather than X which is in the optimal path.

THANK YOU