Introduction to Computational Intelligence

Lecture 5

Outline

- Uninformed search strategies
  - BFS
  - Uniform cost search
  - Depth first search
  - Iterative deepening search

- Informed search strategies
  - Greedy best-first search
  - A* search

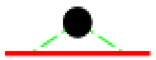
Iterative Deepening Search

- Best of both worlds: Iterative deepening combines the benefits of depth-first and breadth-first search.

- Use DFS as a subroutine with depth limit: first depth 0, then depth 1, then depth 2, and so on.
    1. Check the root
    2. Do a DFS searching for a path of length 1
    3. If there is no solution in the path of length 1, do a DFS searching for a solution in path of length 2
    4. If there is no solution in the path of length 2, do a DFS searching for a solution in path of length 3
    5. Continue the process until a solution is found

Iterative deepening search – Example

depth bound = 0

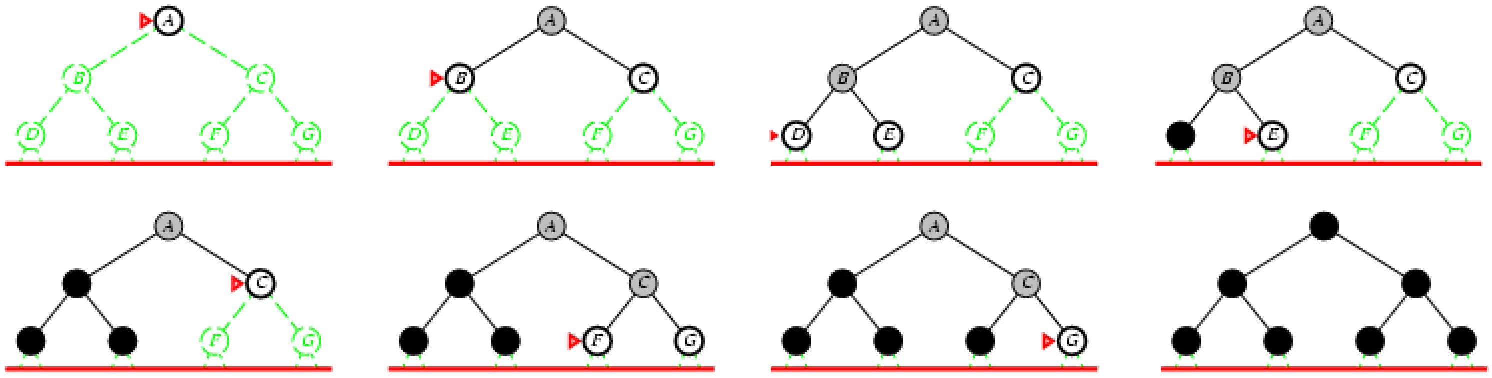


Iterative deepening search

depth bound = 1

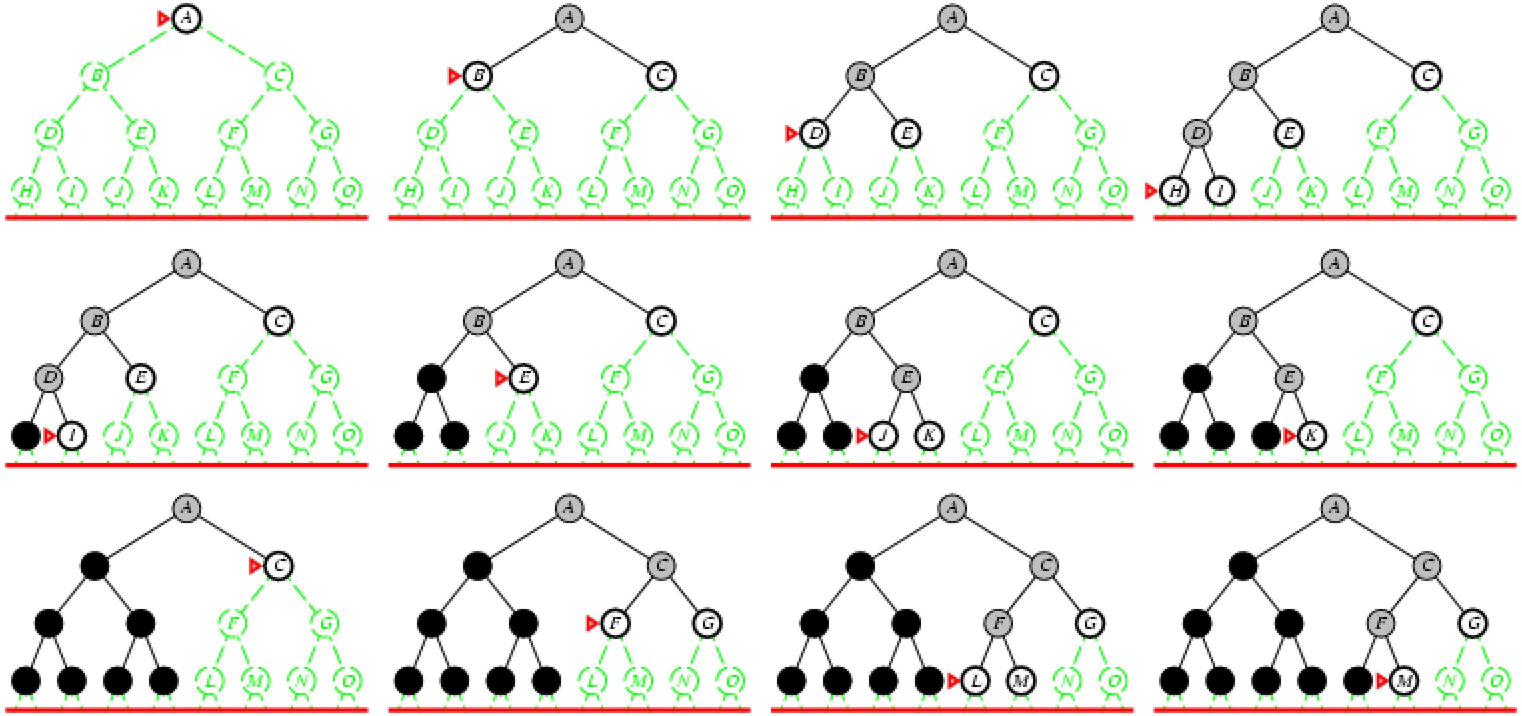
Iterative deepening search cont.

depth bound = 2



Iterative deepening search cont.

depth bound = 3



Properties of IDS

- **Question:** Isn't that wastefully redundant?
  - Generally, most work happens in the lowest level searched, so not so bad!
- **Time?**

  $(d+1)b^0 + d\,b^1 + (d\text{-}1)b^2 + \bullet\bullet\bullet + 3b^{(d-2)} + 2b^{(d-1)} + b^d \cong O(b^d)$
- **Space?**

  $O(bd)$ – linear space requirement
- **Complete?**

  Yes – guarantees to find a solution at the shallowest depth
- **Optimal?**

  Yes, if step cost = 1

Note: In general, iterative deepening is the preferred search method when there is a **large search space**, and the **depth** of the solution **is not known**

Uninformed Search Strategies Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Iterative Deepening
Complete?	Yes#	Yes#, ♦	No	Yes#
Time	$O(b^d)$	$O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right)$	$O(b^m)$	$O(b^d)$
Space	$O(b^d)$	$O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right)$	$O(bm)$	$O(bd)$
Optimal?	Yes♠	Yes	No	Yes♠

Legend:

b - branching factor
d - depth of solution
m - maximum depth of the search tree
l - depth limit
C^* - cost of the optimal solution
ε - minimal cost of an action

complete if b is finite
♦ complete if step costs $\geq \varepsilon$, where $\varepsilon > 0$.
♠ optimal if step costs are all identical

Uninformed Search Strategies Summary Cont.

- IDS is the preferred uninformed search method when there is a large search space and the depth (length) of the solution is not known

- DFS is often used because of its minimal memory requirements – compact encodings of exponential-size explored node set exists

- BFS is rarely found in practice – this does not mean that there are no applications for which this would be the search methods of choice!

- Conceptually, all search algorithms are the same, but the search strategies are distinguished by the order (priority in the fringe) in which nodes are expanded.

Informed Search

- **Idea:** give the algorithm "hints" about the desirability of different states
  - Use an evaluation function, $f(n)$ to rank nodes and select the most promising one for expansion.

- **Implementation:** Similar to that of uniform-cost search, except the next node is picked based on cost from a node n to the goal state, instead of the cost from the node n to the goal state.
  - Certainly, it must have some additional information.

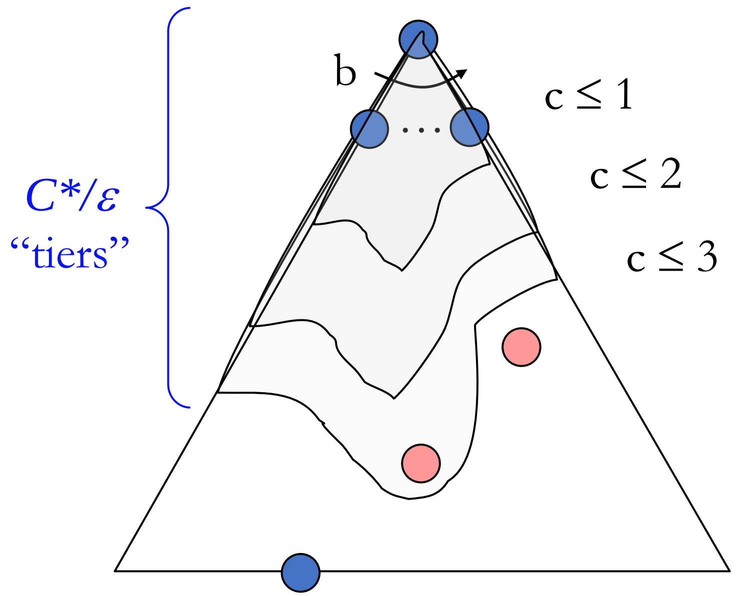
Let's Recall UCS

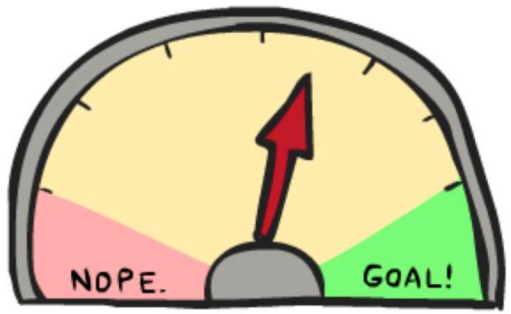
- **Strategy**: expand lowest path cost, $g(n)$.

- **+ves**: UCS is complete and optimal!

- **-ves**:
  - Explores nodes in "every direction"
  - No information about goal location

- **Solution**: Provide some sort of directional information about the goal.
  - It is called **heuristic**





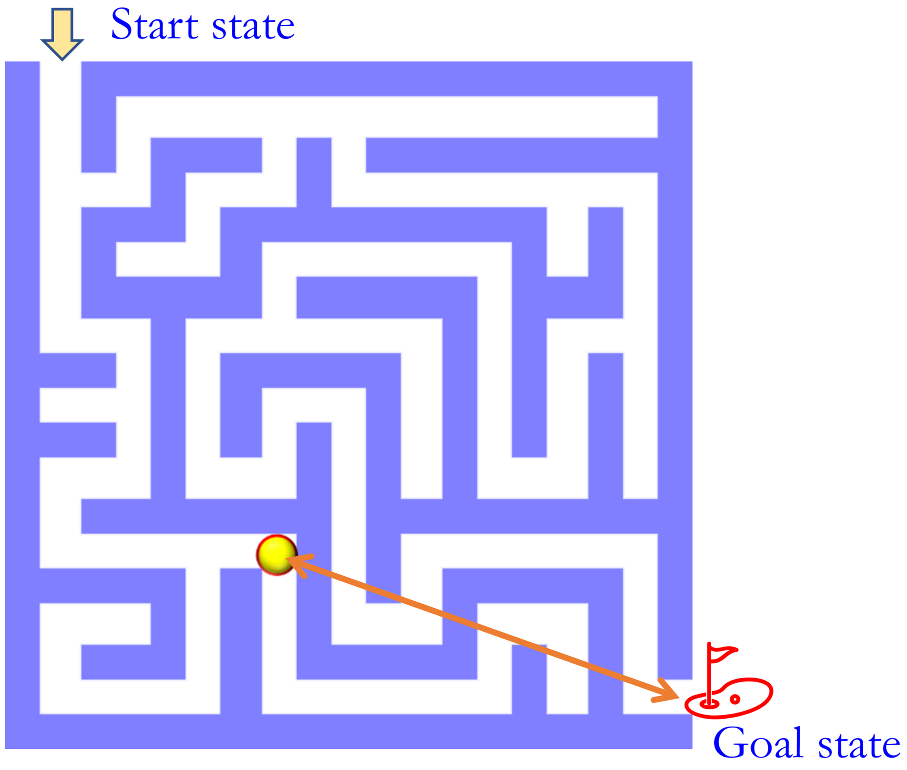
Animation Adopted from Anca Dragan, University of California, Berkeley

Heuristic Function

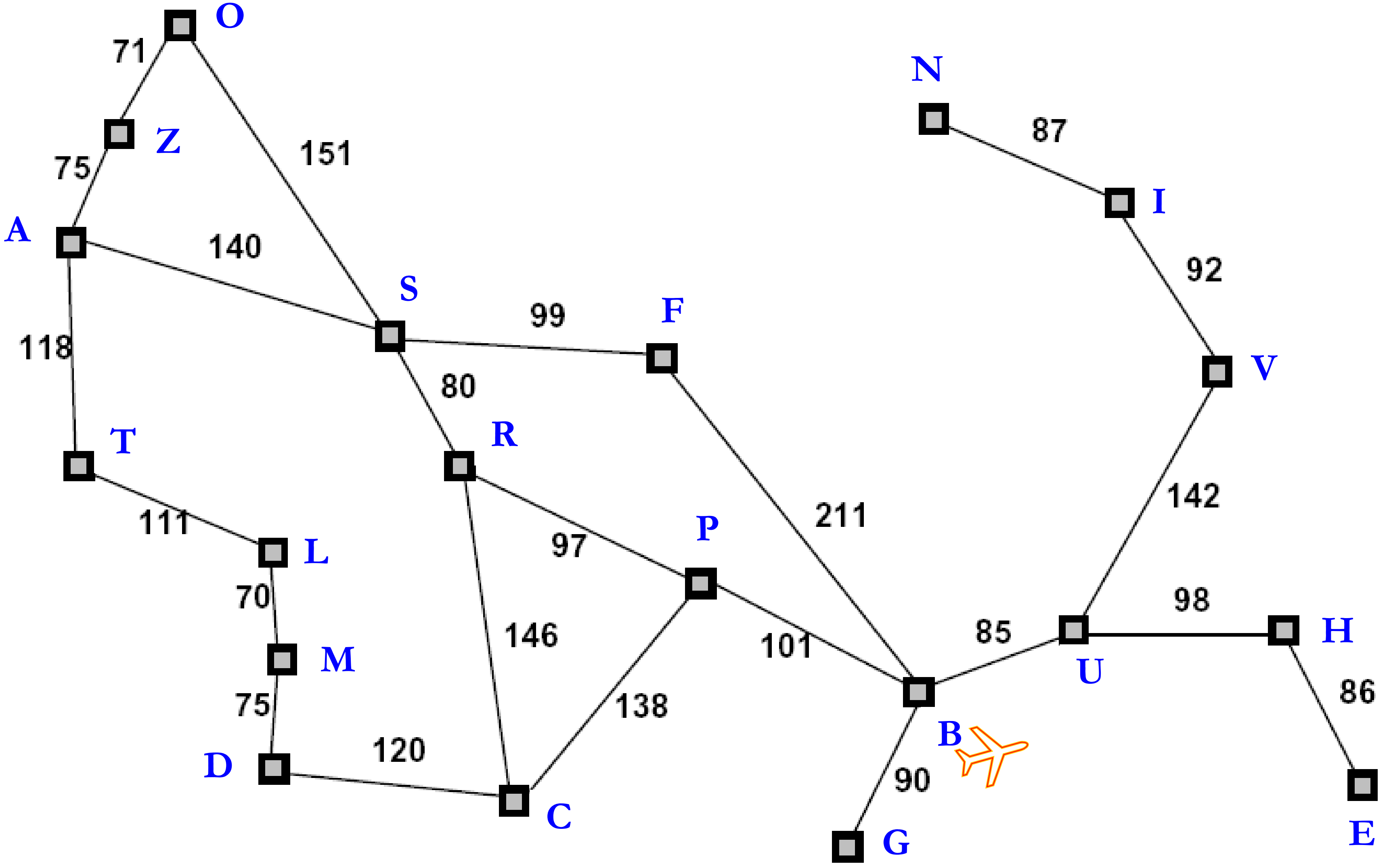
- A function, $h(n)$ that estimates how close a state, n is to a goal (i.e., estimated cost of the cheapest path from the state at node, n to a goal)

  - Designed for a particular search problem

  - E.g., Maze solving.
  - **What could be good heuristic function** that can be used to estimate the closeness to the goal state?
    - **Reasonable heuristic functions**: Manhattan distance, and Euclidean distance for pathing



Heuristic Function Example



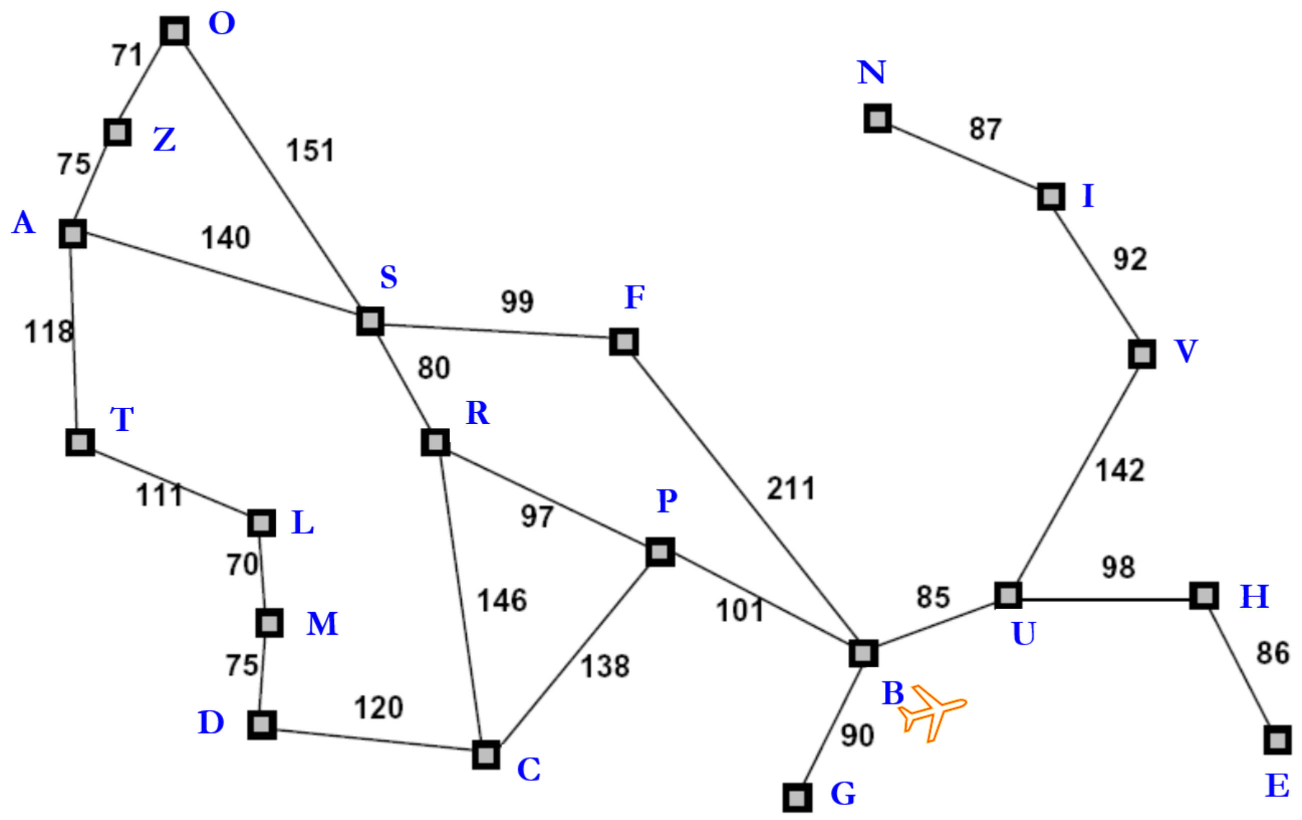
$h(n)$

A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

Let's use this heuristic in Greedy Search

Greedy Search

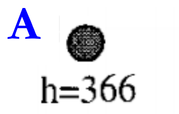
- **Strategy**: Expand the node that has the lowest value of the heuristic function $h(n)$
- Let's apply this to the route-finding problem: **Travelling from A to B**.



$h(n)$	
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

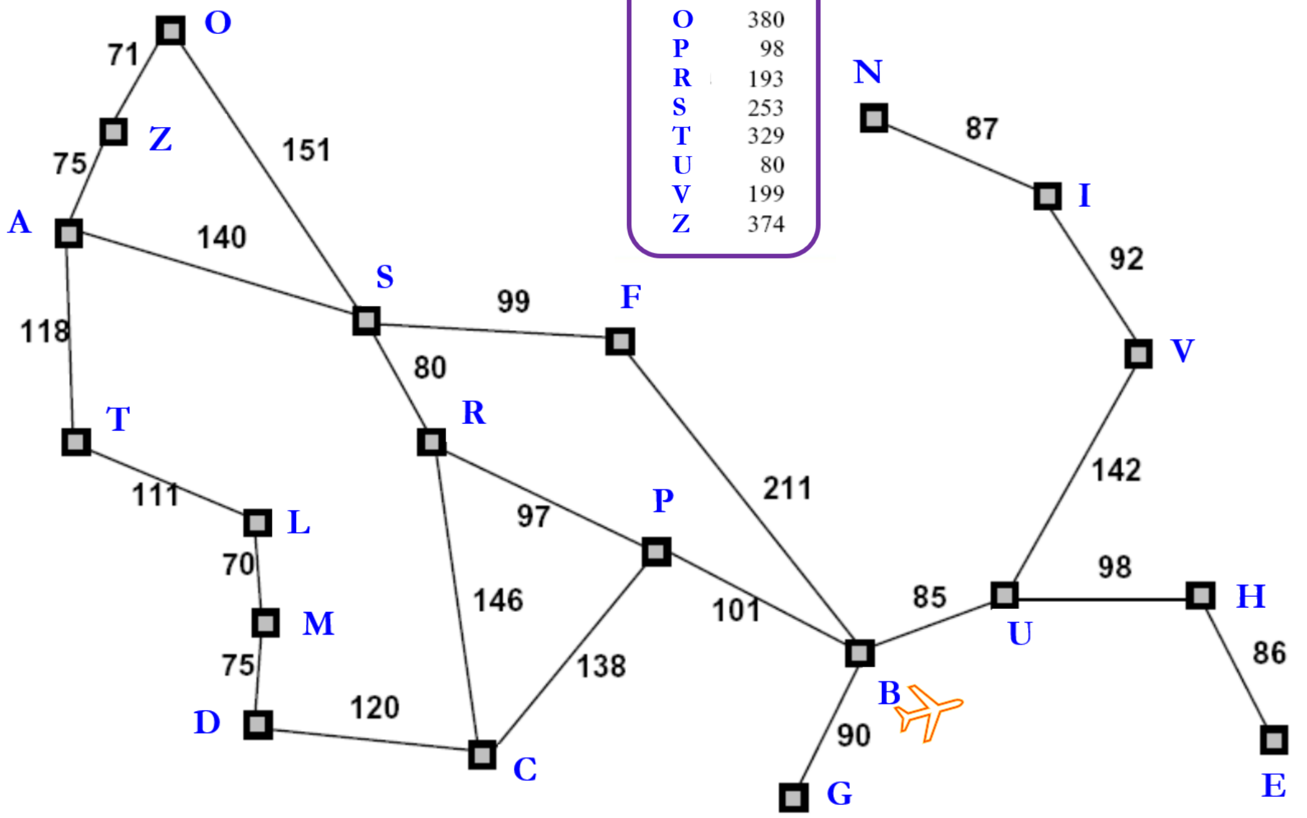
Greedy Search - Example

The initial state →



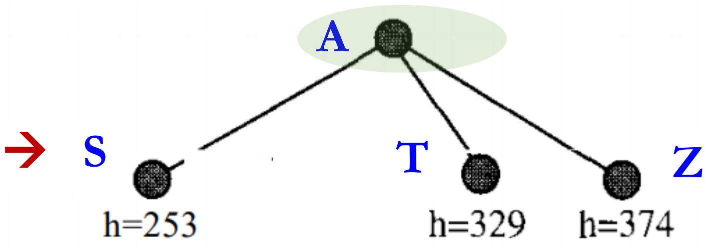
$h(n)$

A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

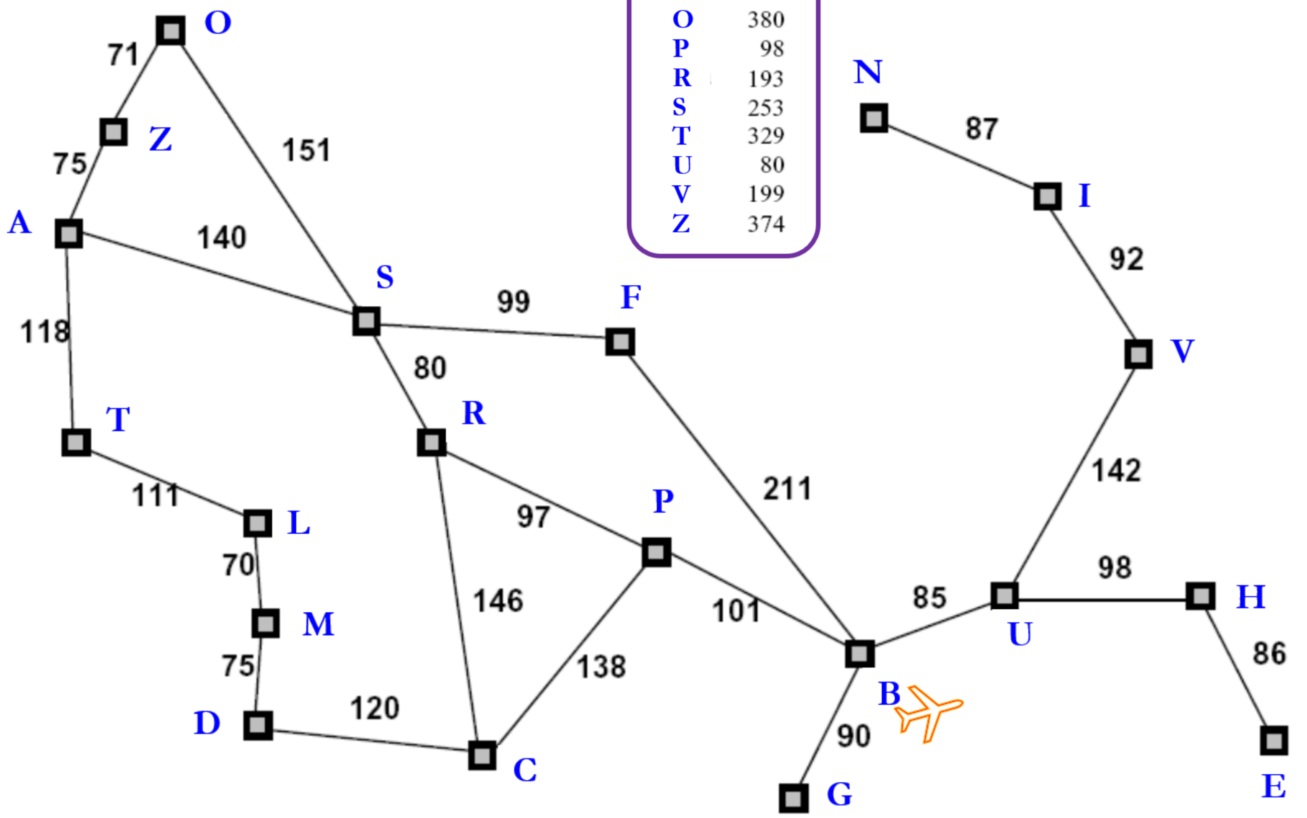


Greedy Search - Example

After expanding A

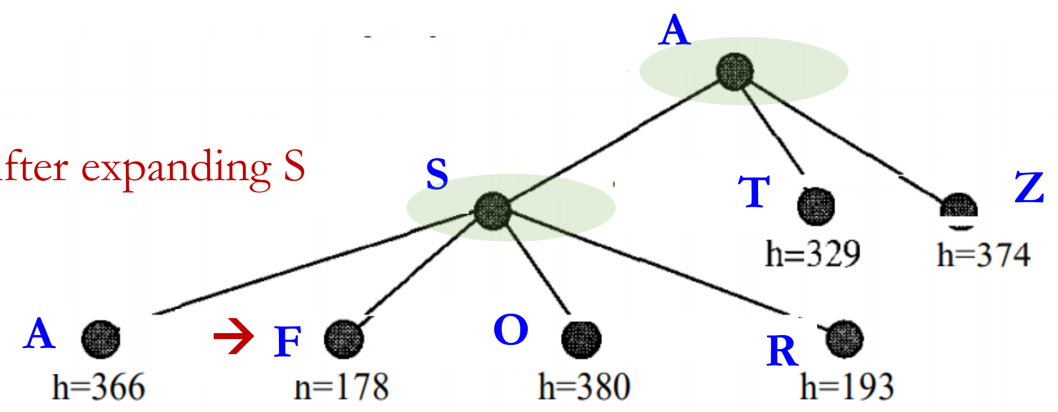


$h(n)$	
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374



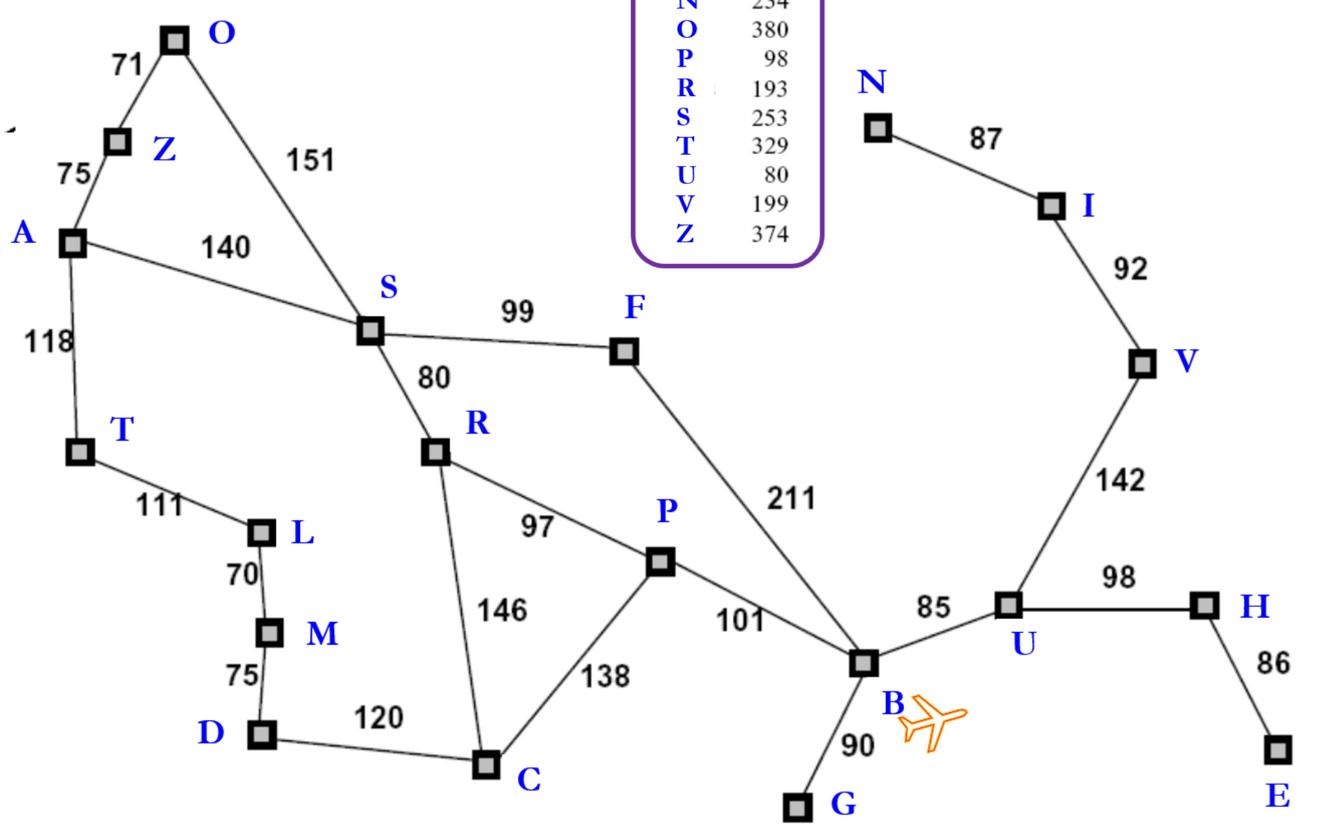
Greedy Search - Example

After expanding S

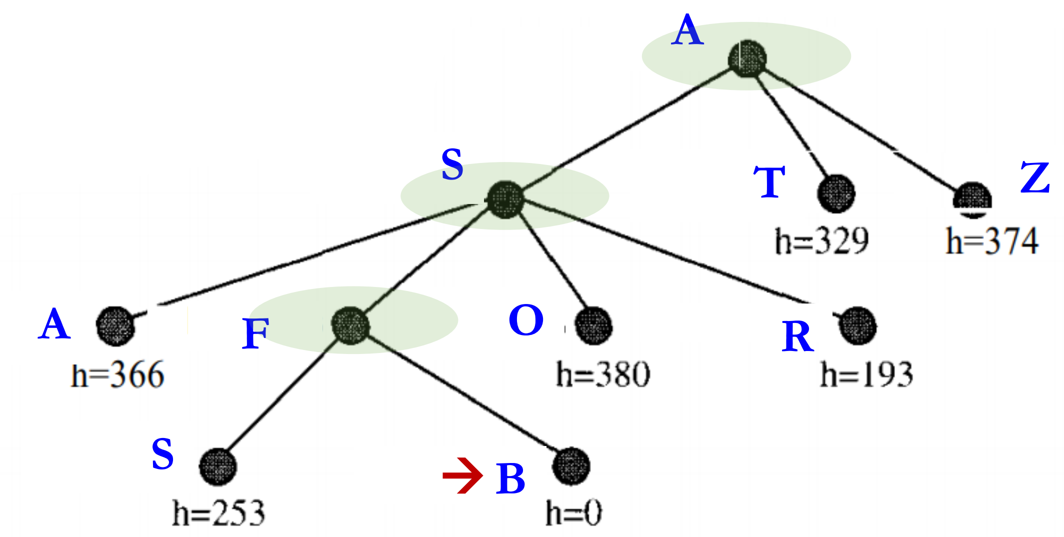


$h(n)$

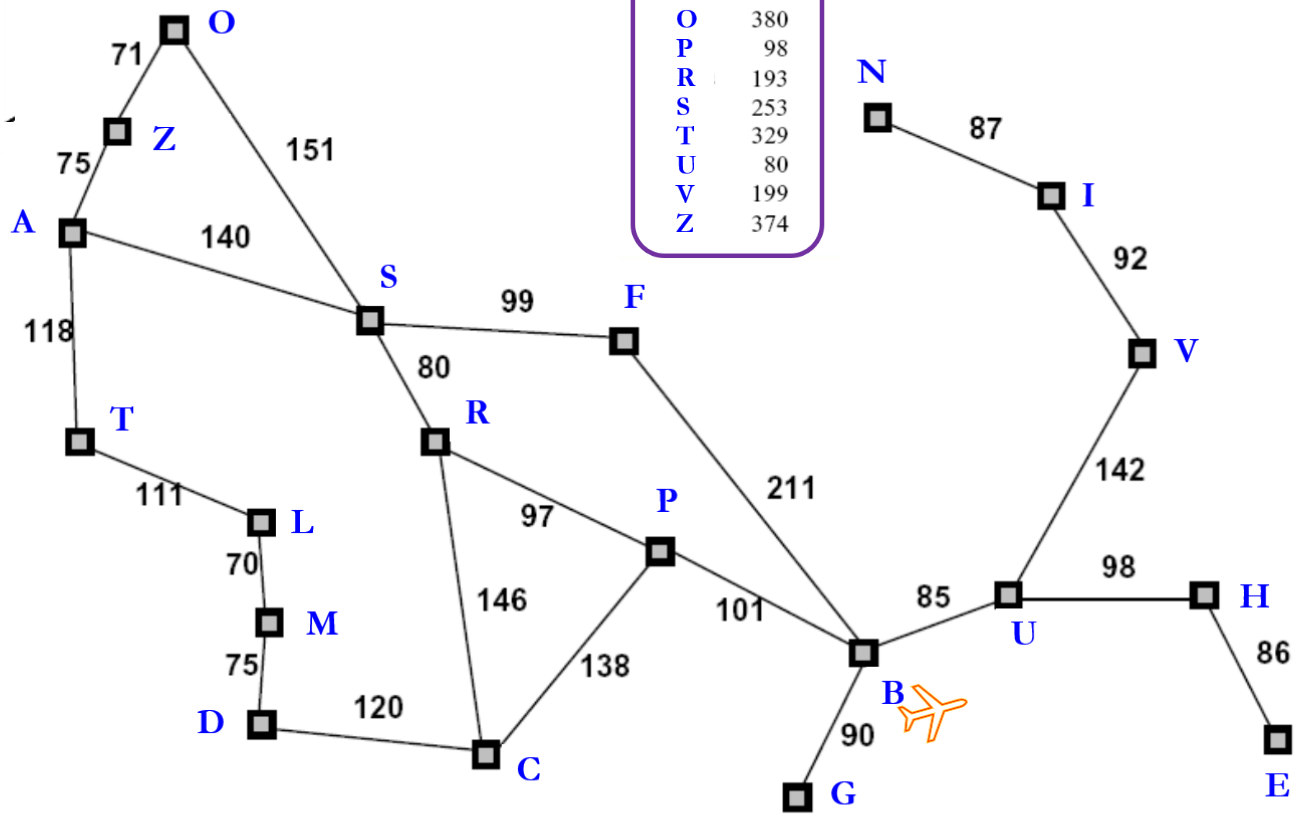
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374



Greedy Search - Example



$h(n)$	
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

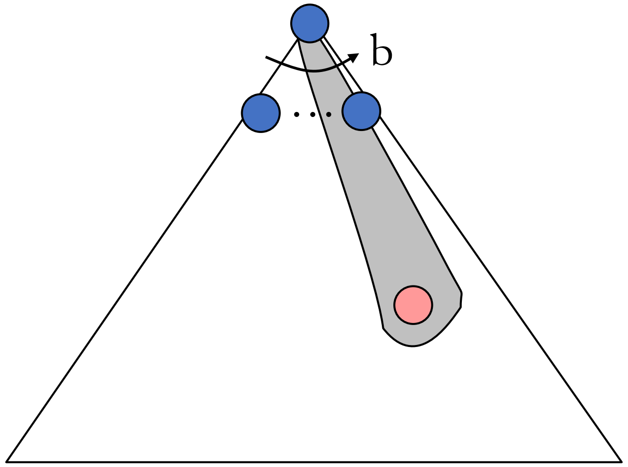


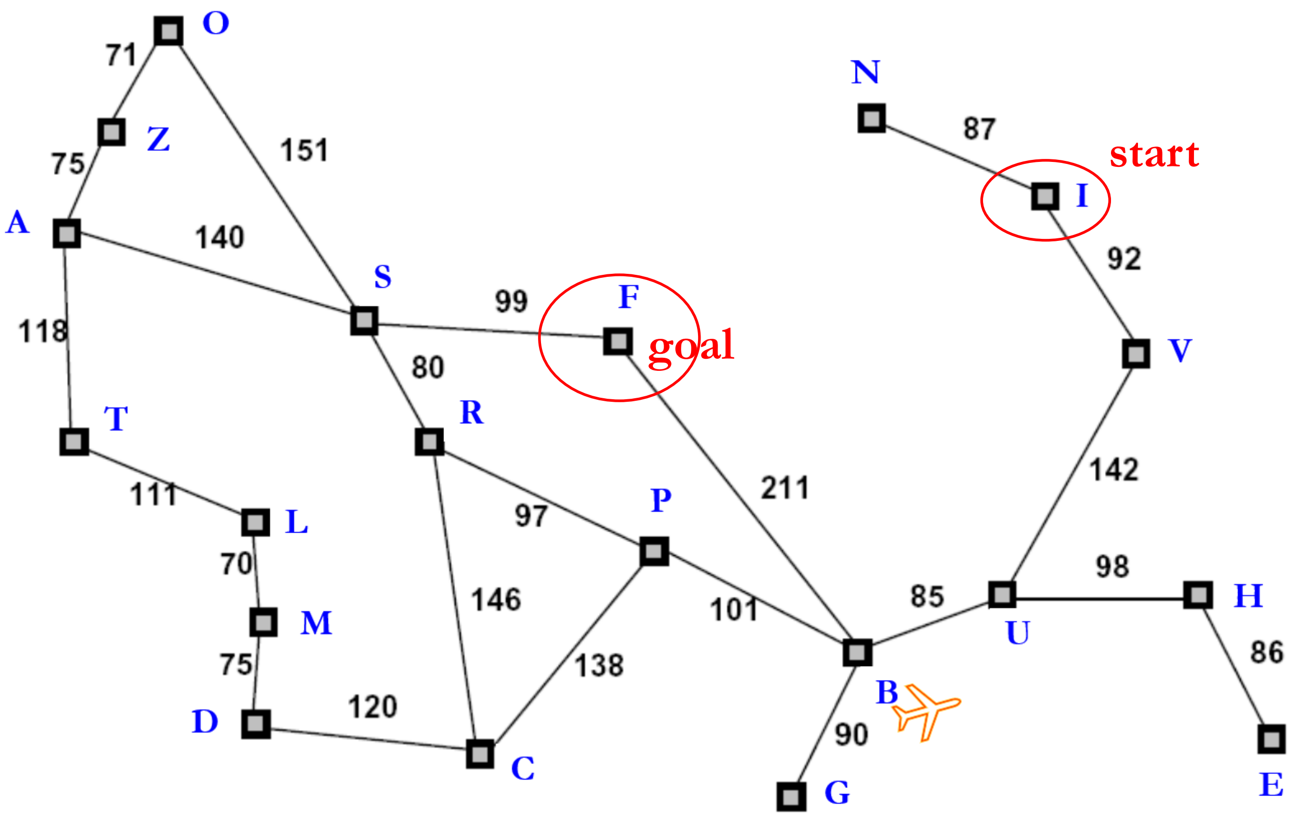
- Return path: A→S→F→B

Properties of Greedy Search

- **Complete?**

  No – can get stuck in loops





Left image is Adopted from Anca Dragan, University of California, Berkeley

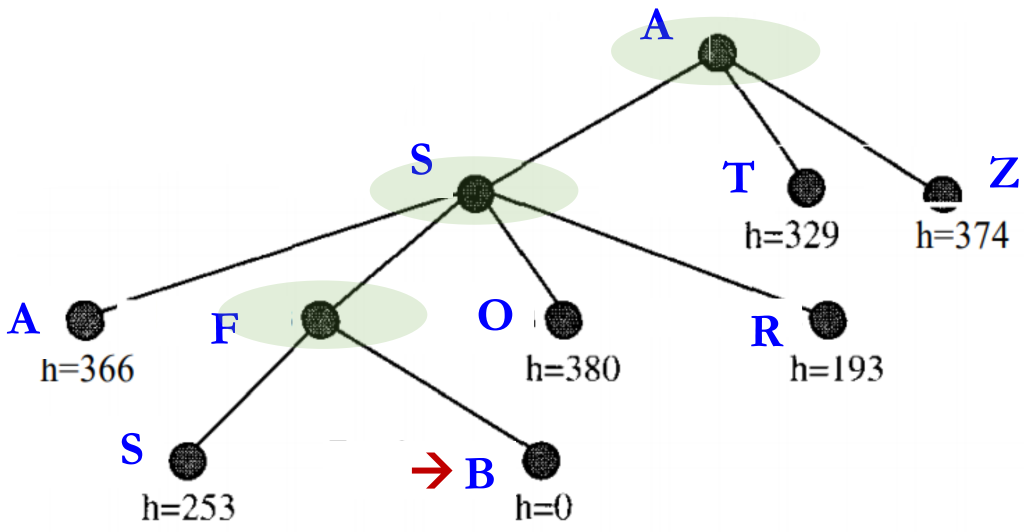
Properties of Greedy Search Cont.

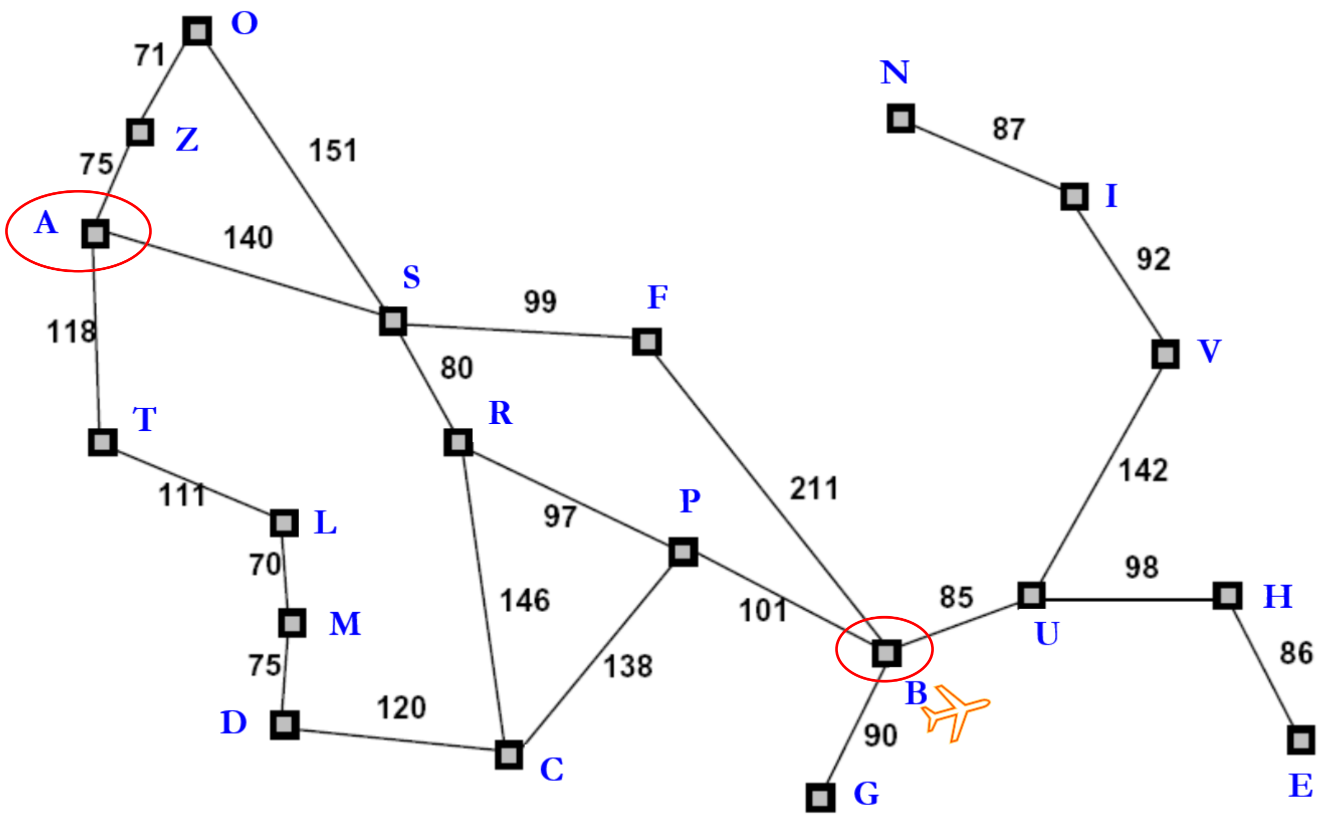
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





What is the path cost of:

A→S→F→B ?________

A→S→R→P→B ?________

Properties of Greedy Search Cont.

- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No

- **Time?**

  Worst case: $O(b^m)$

  Best case: $O(bd)$ – If $h(n)$ is 100% accurate

- **Space?**

  Worst case: $O(b^m)$

Greedy Search Problem

- It **minimizes the estimated cost** from node *n* to the goal (forward path), $h(n)$, thus cuts the search cost considerably.

- **Issue:** neither optimal nor complete.

- **What we know:**
  - UCS minimizes the cost of the path so far, i.e., from node *n* to the **start** state (backward path), $g(n)$;
  - It is optimal and complete, but can be very inefficient.

- **Solution:** $f(n) = g(n) + h(n)$.
  - $g(n)$ gives the path cost from the start node to node n,
  - $h(n)$ is the estimated cost of the cheapest path from n to the goal,
  - So, $f(n)$ is the estimated cost of the cheapest solution through n