CS 156:Introduction to Artificial Intelligence

Instructor: Dr. Sayma Akther

San José State University

Criteria for Strategy Evaluation

- **Completeness**: Can the strategy always find a solution if one exists?
- **Optimality**: Can it find the least-cost solution?
- **Time Complexity**: How long does it take to find a solution?
- **Space Complexity**: How much memory does it need?

Search Strategies

- **Uninformed Search (Blind Search)**: No additional information about states and the quality of actions. Example: Breadth-First Search (BFS), Depth-First Search (DFS).
- **Informed Search**: Uses knowledge about the problem to find solutions more efficiently. Often relies on heuristics. Example: A* search.

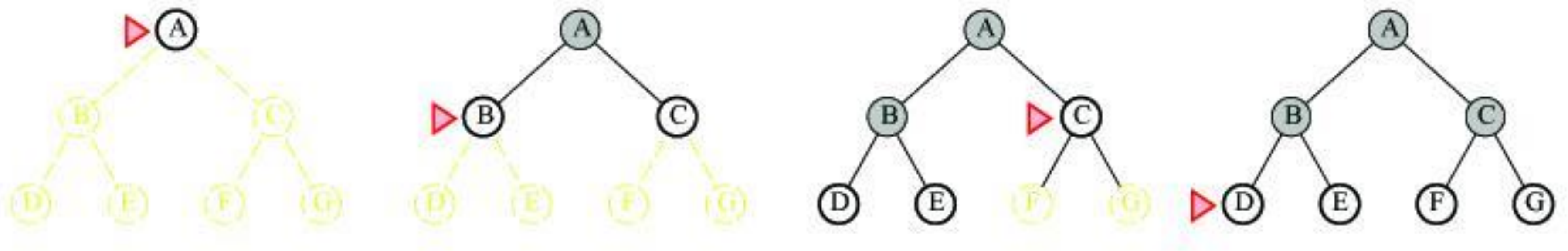
Uninformed Search

Breadth-First Search (BFS)

Explores all neighbor nodes at the current depth before moving to nodes at the next depth level.

- Generate children of a state, adds the children to the end of the open list
- Level-by-level search
- Order in which children are inserted on open list is arbitrary
- In tree, assume children are considered left-to-right unless specified differently
- Number of children is “branching factor” b

BFS Example



branching factor b = 2

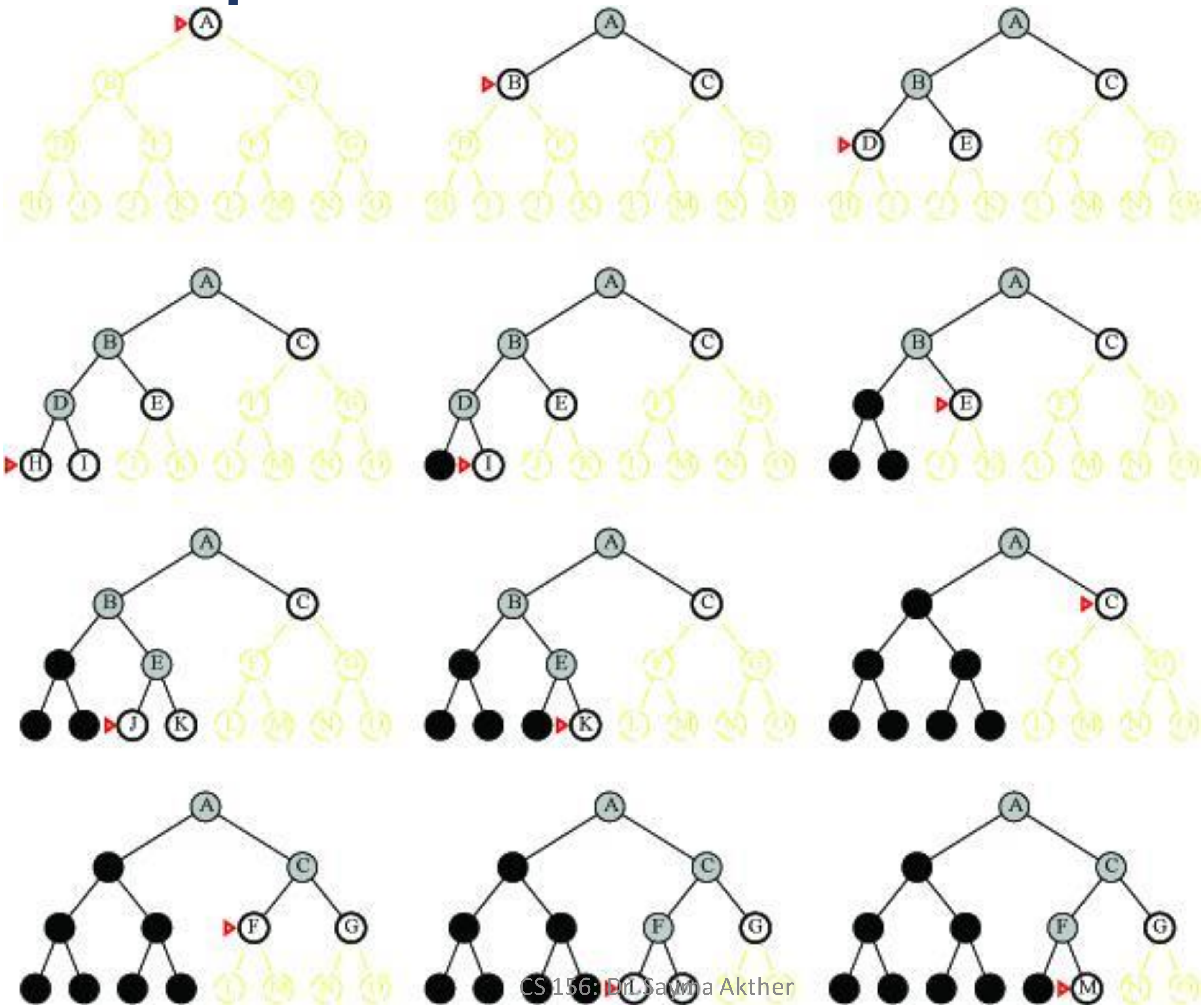
Breadth-First Search (BFS)

- Assume goal node at level d with constant branching factor b
- Time complexity (measured in #nodes generated)
  - ➢ 1 (1st level) + b (2nd level) + b^2 (3rd level) + … + b^d (goal level) + (b^{d+1} – b) = O(b^d+1)
- This assumes goal on far right of level
- Space complexity
  - ➢ At most majority of nodes at level d + majority of nodes at level d+1 = O(b^{d+1})
  - ➢ Exponential time and space
- Features
  - ➢ Simple to implement
  - ➢ Complete
  - ➢ Finds shortest solution (not necessarily least-cost unless all operators have equal cost)

Depth-First Search (DFS)

- Adds the children to the front of the open list
- Follow leftmost path to bottom, then backtrack
- Expand deepest node first

DFS Example



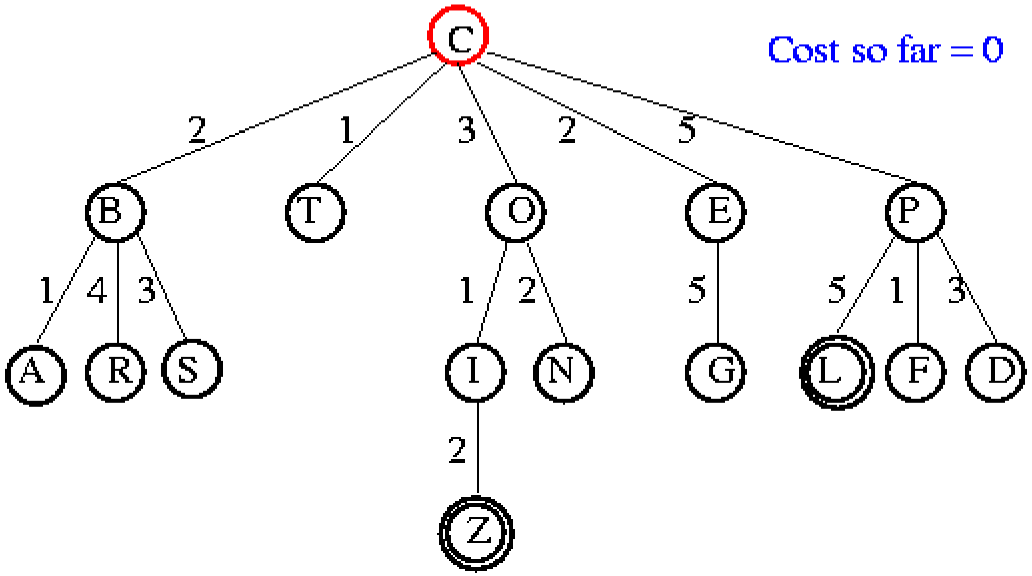
Depth-First Search (DFS)

- **Time complexity**
  - In the worst case, search entire space
  - Goal may be at level d but tree may continue to level m, m>=d
  - $O(b^m)$
  - Particularly bad if tree is infinitely deep
- **Space complexity**
  - Only need to save one set of children at each level
  - 1 + b + b + … + b (m levels total) = O(bm)
- **Benefits**
  - May not always find solution
  - Solution is not necessarily shortest or least cost
  - If many solutions, may find one quickly (quickly moves to depth d)
  - Simple to implement
  - Space often bigger constraint, so more usable than BFS for large problems

Uniform Cost Search

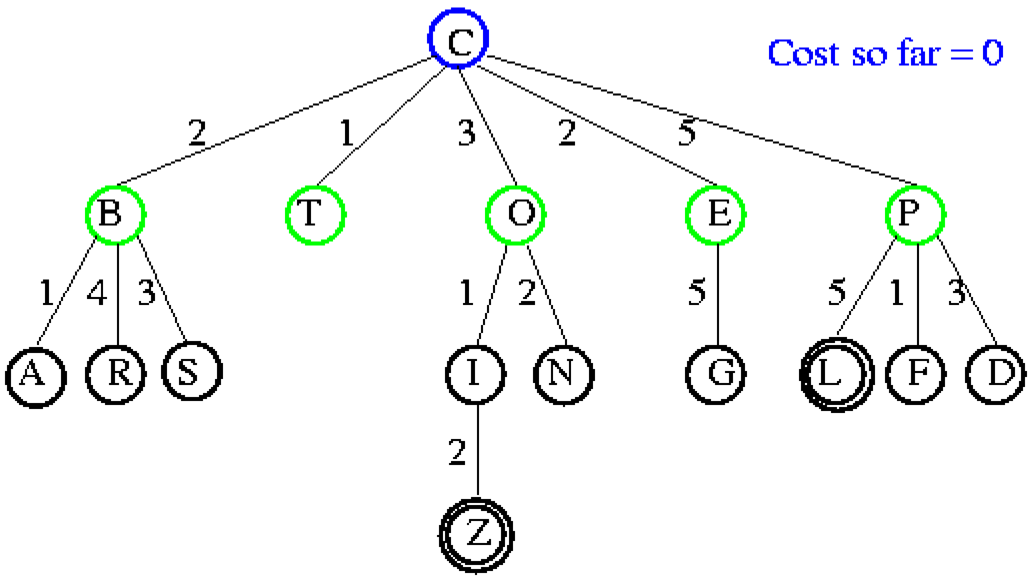
- Cost from root to current node n is g(n)
  - Add operator costs along path
- First goal found is least-cost solution
- Space & time can be exponential because large subtrees with inexpensive steps may be explored before useful paths with costly steps
- If costs are equal, time and space are $O(b^d)$
  - Otherwise, complexity related to cost of optimal solution

UCS Example



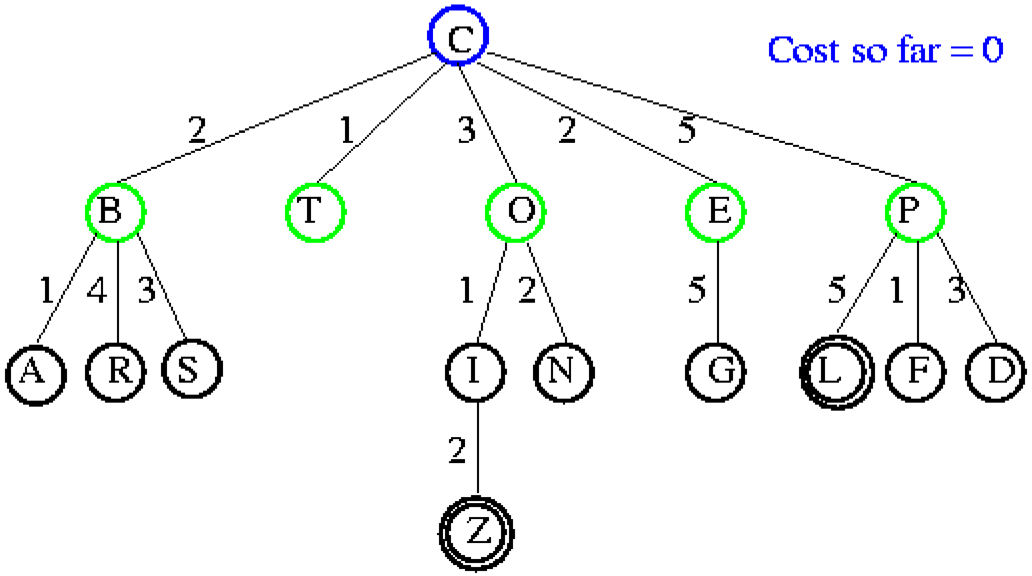
Open list: C

UCS Example



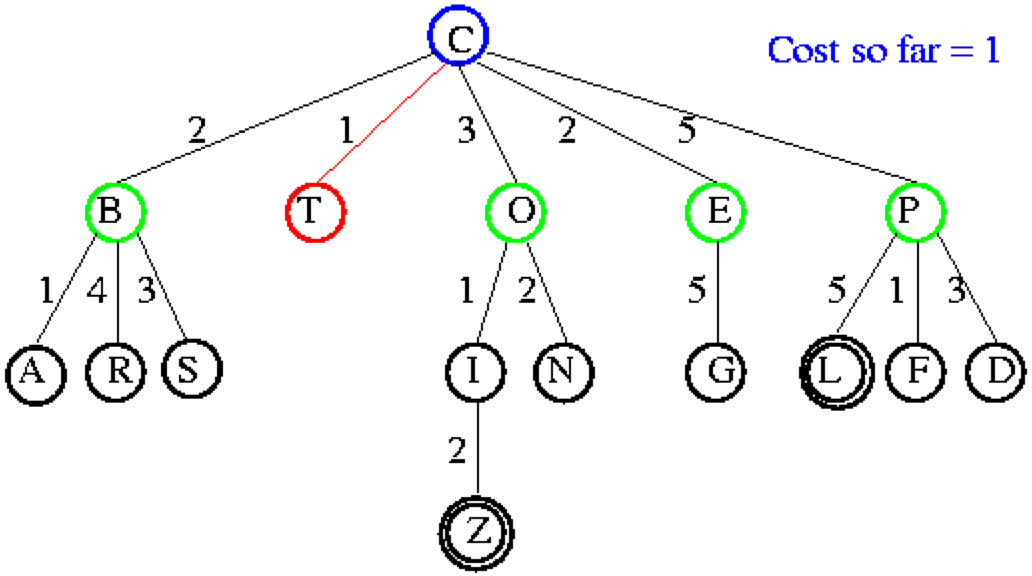
Open list: B(2) T(1) O(3) E(2) P(5)

UCS Example



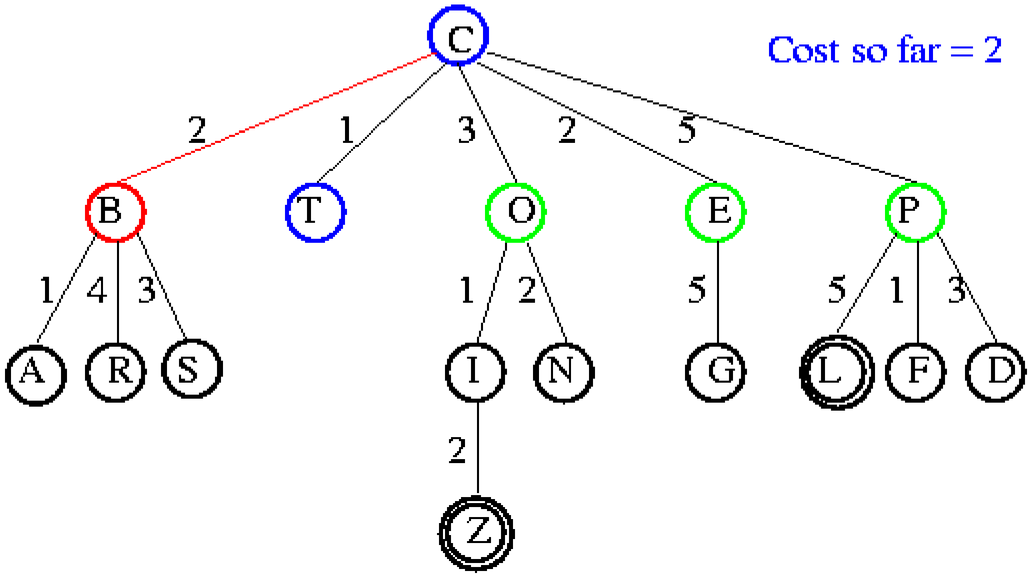
Open list: T(1) B(2) E(2) O(3) P(5)

UCS Example



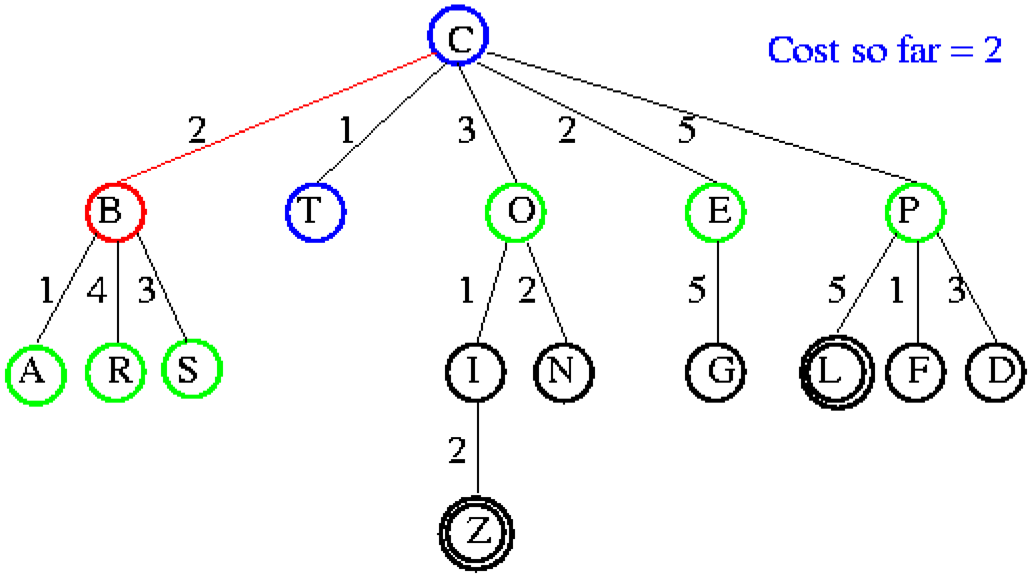
Open list: B(2) E(2) O(3) P(5)

UCS Example



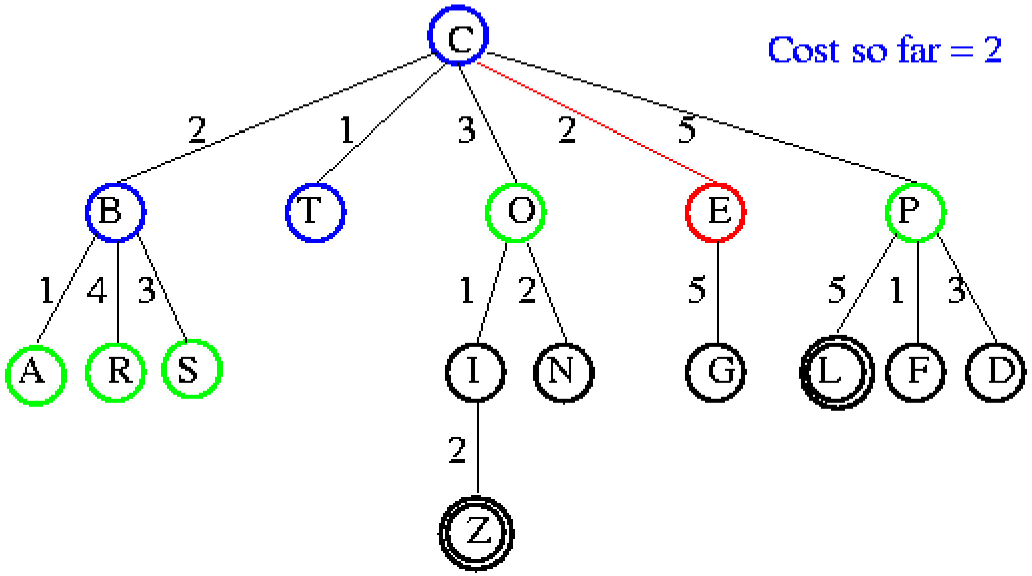
Open list: E(2) O(3) P(5)

UCS Example



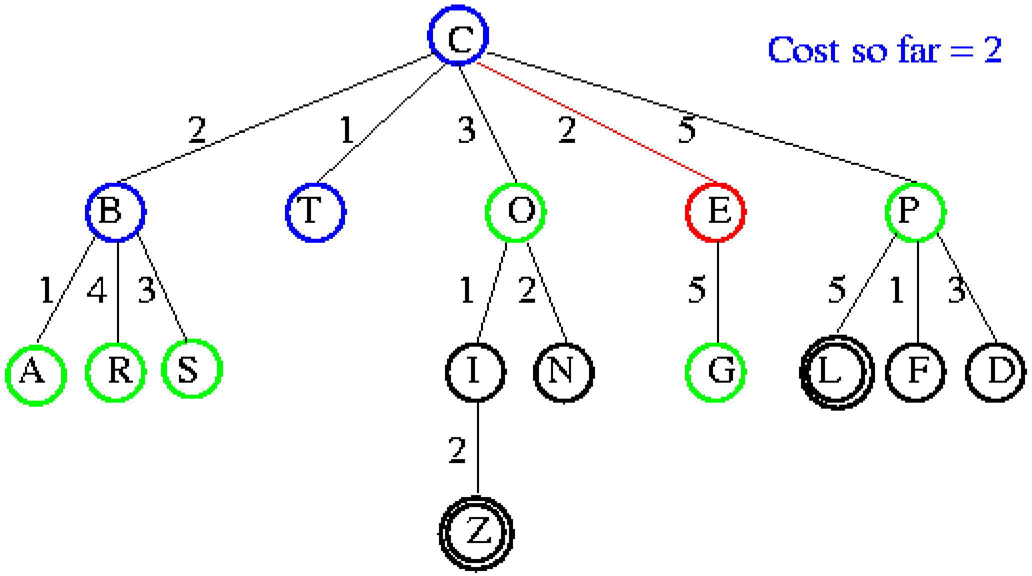
Open list: E(2) O(3) A(3) S(5) P(5) R(6)

UCS Example



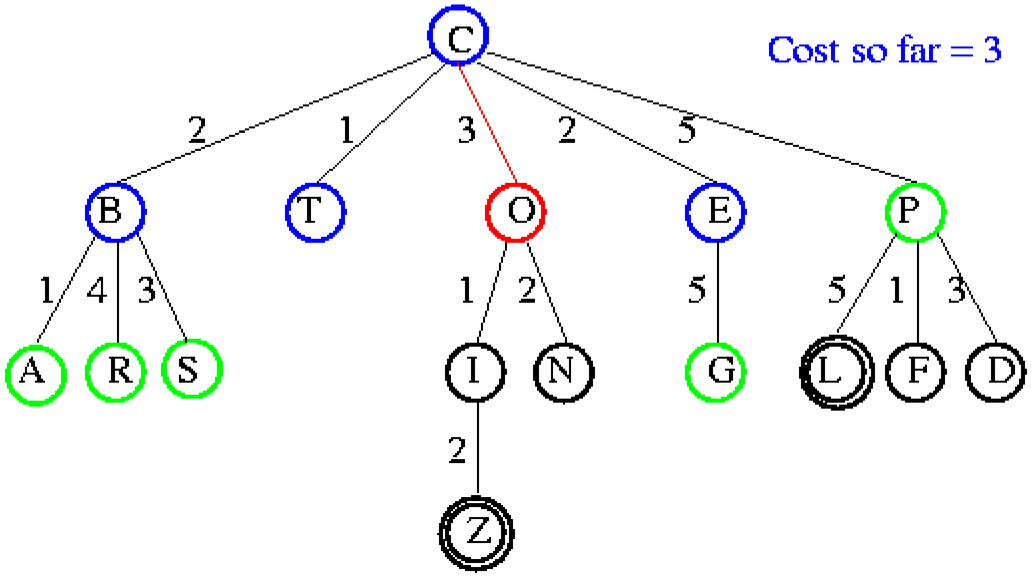
Open list: O(3) A(3) S(5) P(5) R(6)

UCS Example



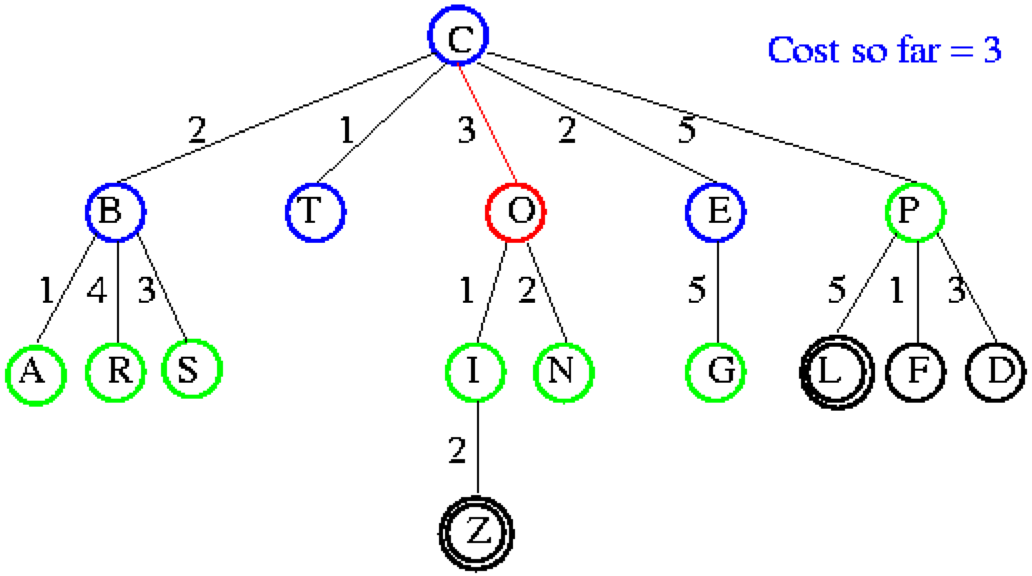
Open list: O(3) A(3) S(5) P(5) R(6) G(7)

UCS Example



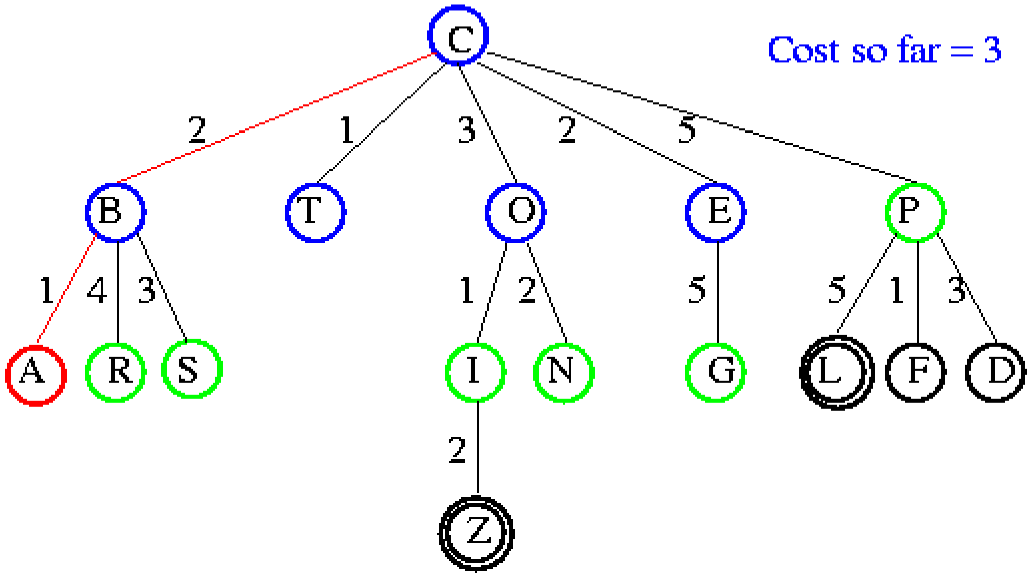
Open list: A(3) S(5) P(5) R(6) G(7)

UCS Example



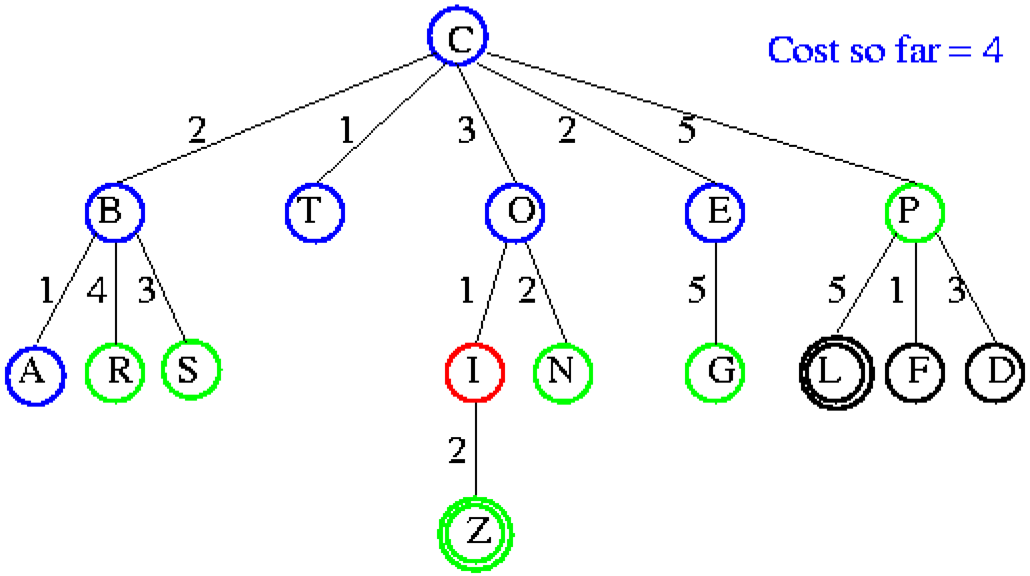
Open list: A(3) I(4) S(5) N(5) P(5) R(6) G(7)

UCS Example



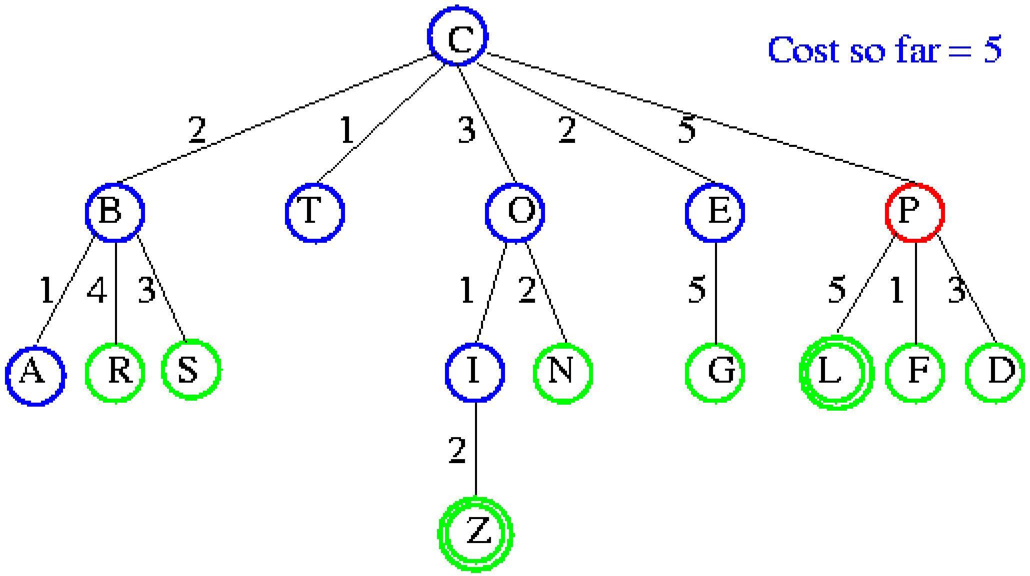
Open list: I(4) P(5) S(5) N(5) R(6) G(7)

UCS Example



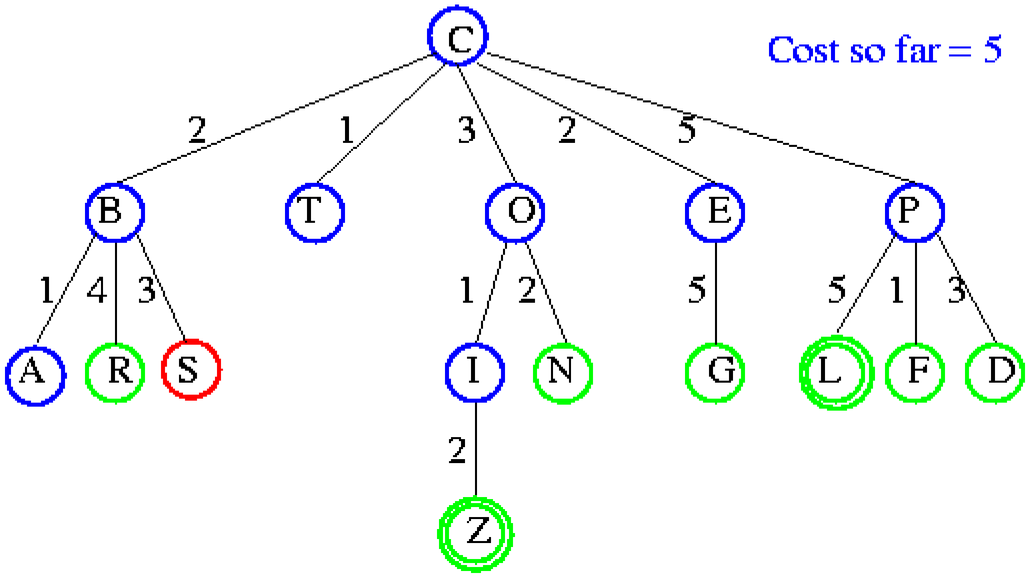
Open list: P(5) S(5) N(5) R(6) Z(6) G(7)

UCS Example



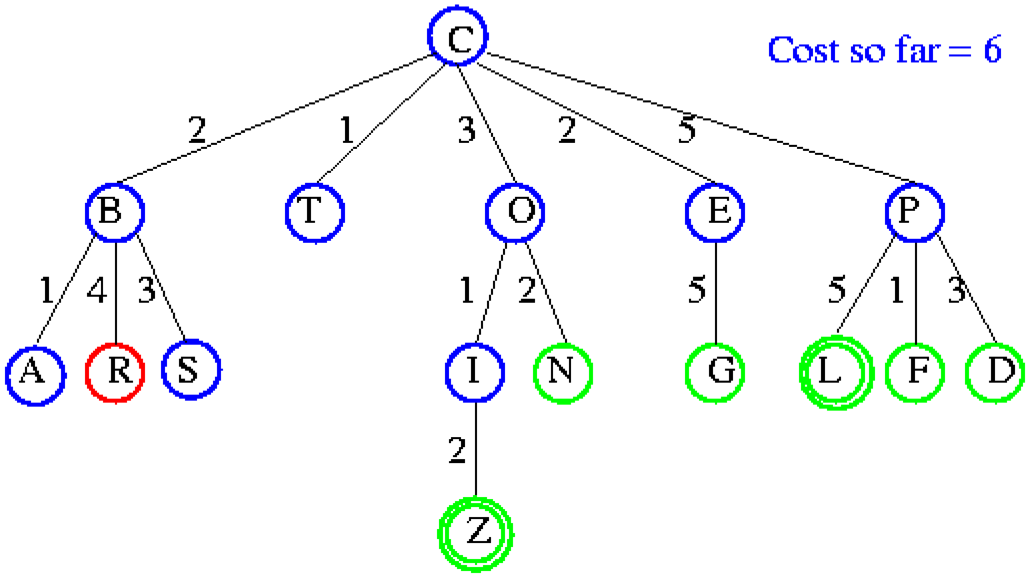
Open list: S(5) N(5) R(6) Z(6) F(6) G(7) D(8) L(10)

UCS Example



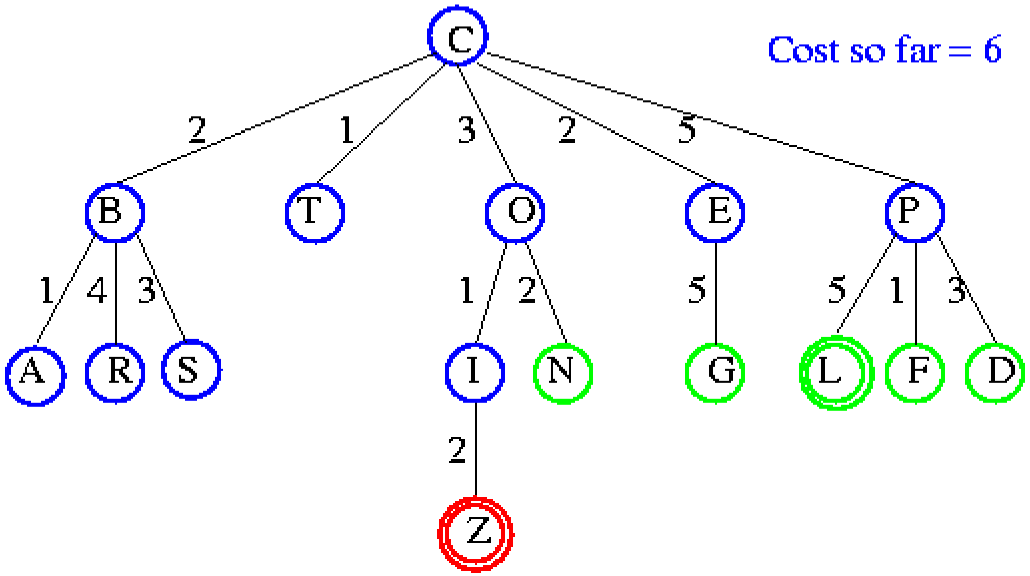
Open list: N(5) R(6) Z(6) F(6) G(7) D(8) L(10)

UCS Example



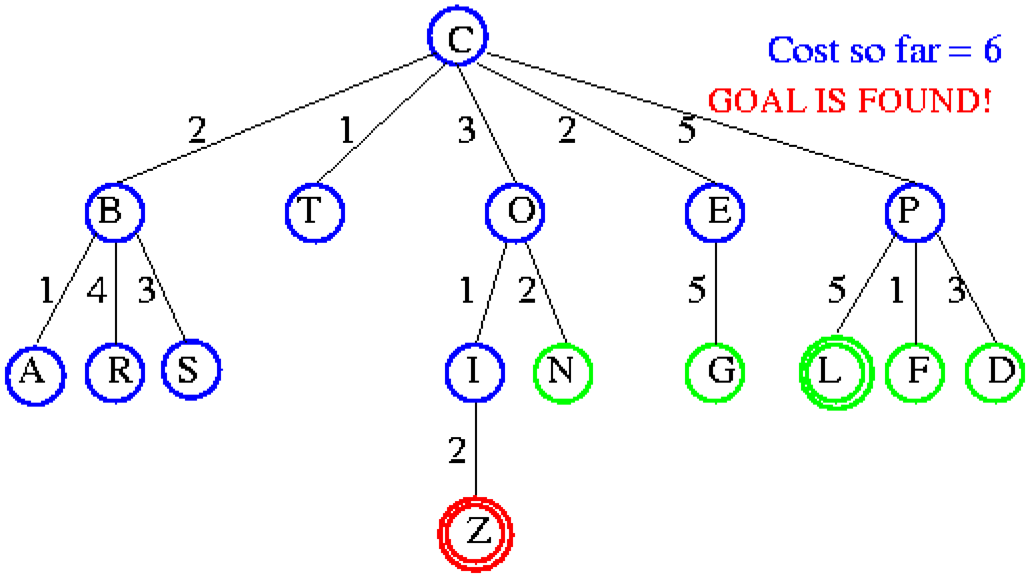
Open list: Z(6) F(6) G(7) D(8) L(10)

UCS Example



Open list: F(6) G(7) D(8) L(10)

UCS Example



Cost so far = 6
GOAL IS FOUND!
C
2
1
3
2
5
B
T
O
E
P
1
4
3
1
2
5
5
1
3
A
R
S
I
N
G
L
F
D
2
Z

Informed Search

Comparison of Uninformed and Informed

1. **Knowledge**
2. **Efficiency**
3. **Good**
4. **Completeness**

Heuristic Functions

- **Definition**: A heuristic is a function that estimates how close a state is to a goal. It doesn't guarantee a perfect solution, but it guides the search in a promising direction.
- **Usage**: Represented as h(n) where n is a node. The function returns an estimated cost from node n to the goal.

h(N) = Heuristic function

Calculate Heuristic Functions

$h_1(N)$ = number of misplaced numbered tiles = 6

$h_2(N)$ = sum of the (Manhattan) distances of every tile to its goal position

= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

5		8
4	2	1
7	3	6

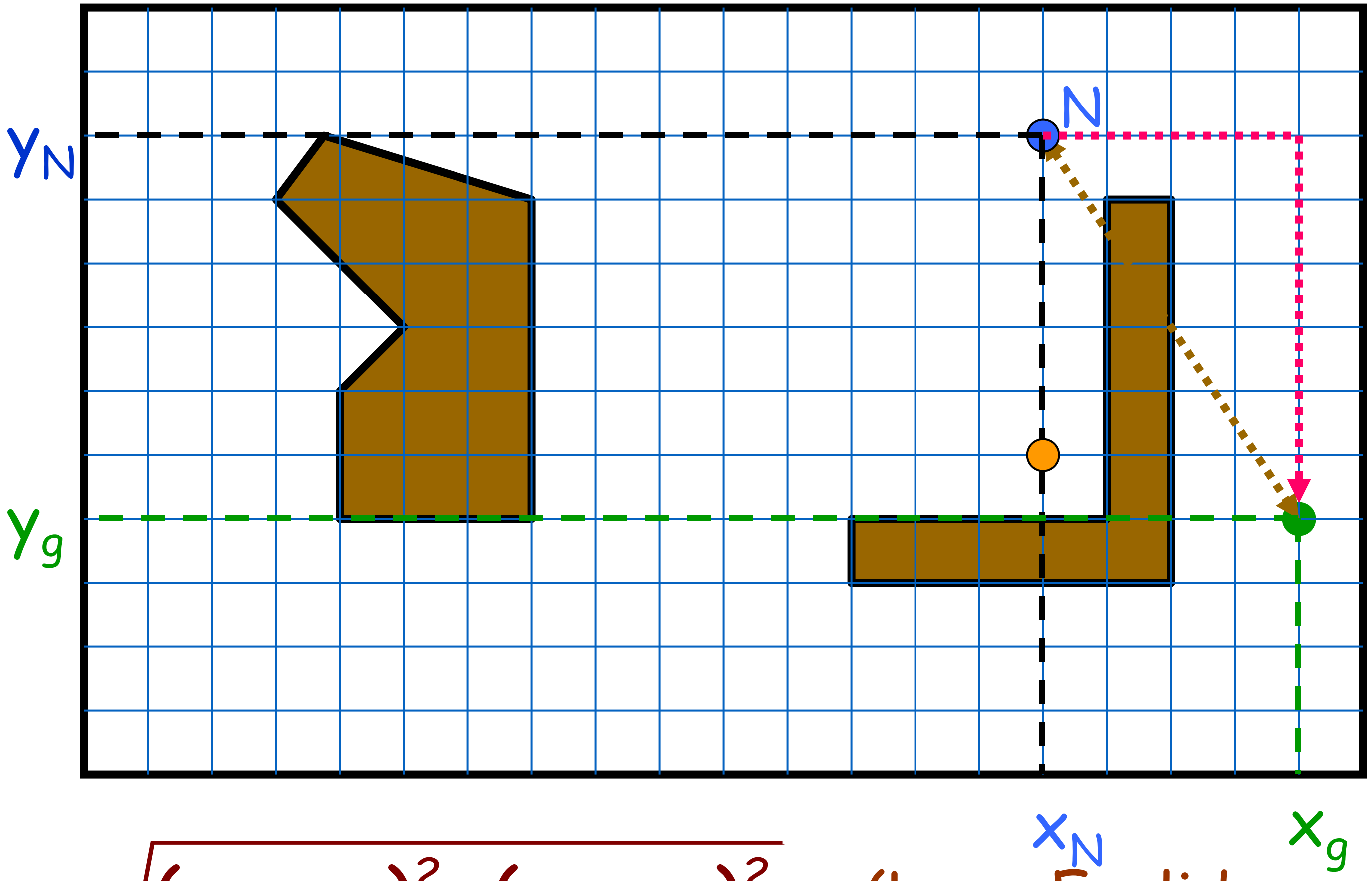
STATE(N)

1	2	3
4	5	6
7	8	

Goal state

$h_2(N) = |x_N - x_g| + |y_N - y_g|$ (L_1 or Manhattan distance)

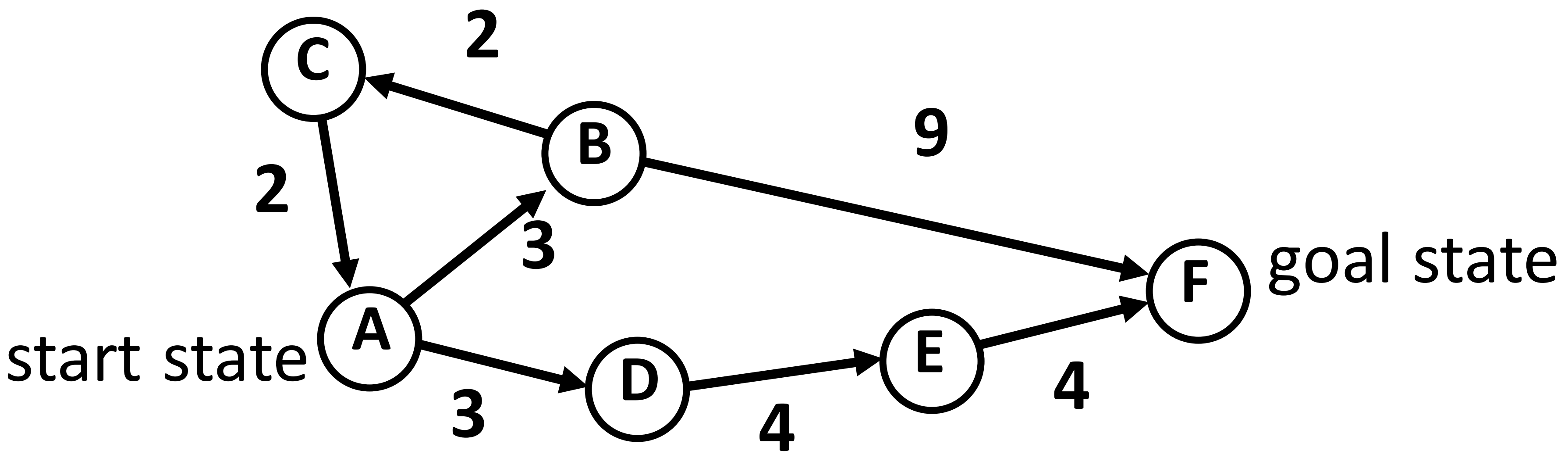
Calculate Heuristic Functions



$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$ (L_2 or Euclidean distance)

Best-First Search

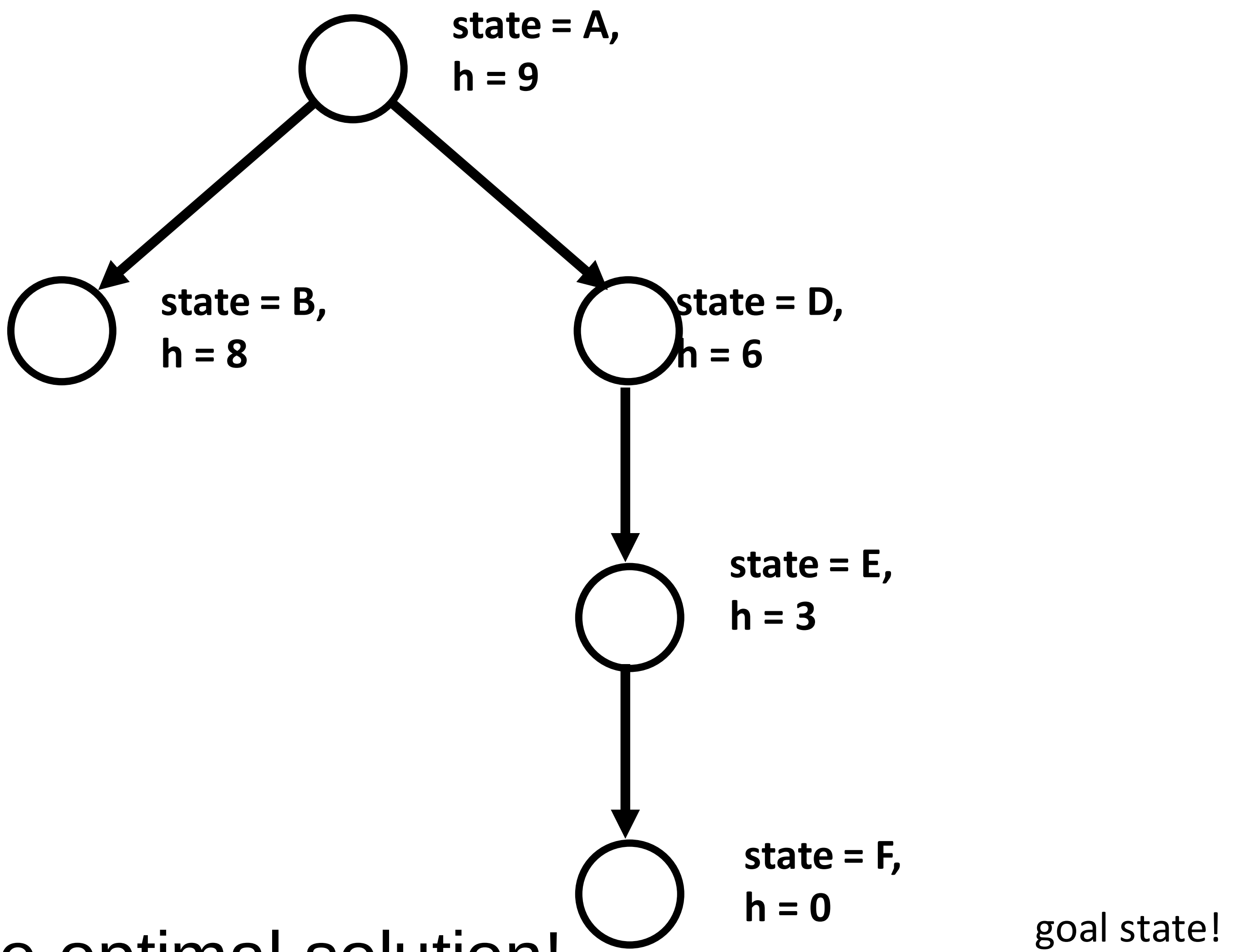
- heuristic function h(n) gives an estimate of the distance from n to the goal
  - h(n)=0 for goal nodes
- E.g. straight-line distance for traveling problem



- Say: h(A) = 9, h(B) = 8, h(C) = 9, h(D) = 6, h(E) = 3, h(F) = 0
  - Typically assume that h is 0 at goal states

Best-First Search

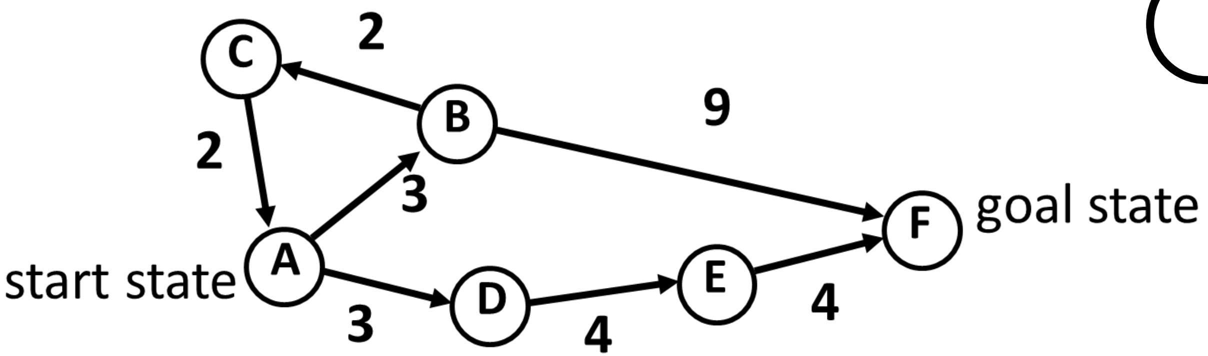
- expand nodes with lowest h values first



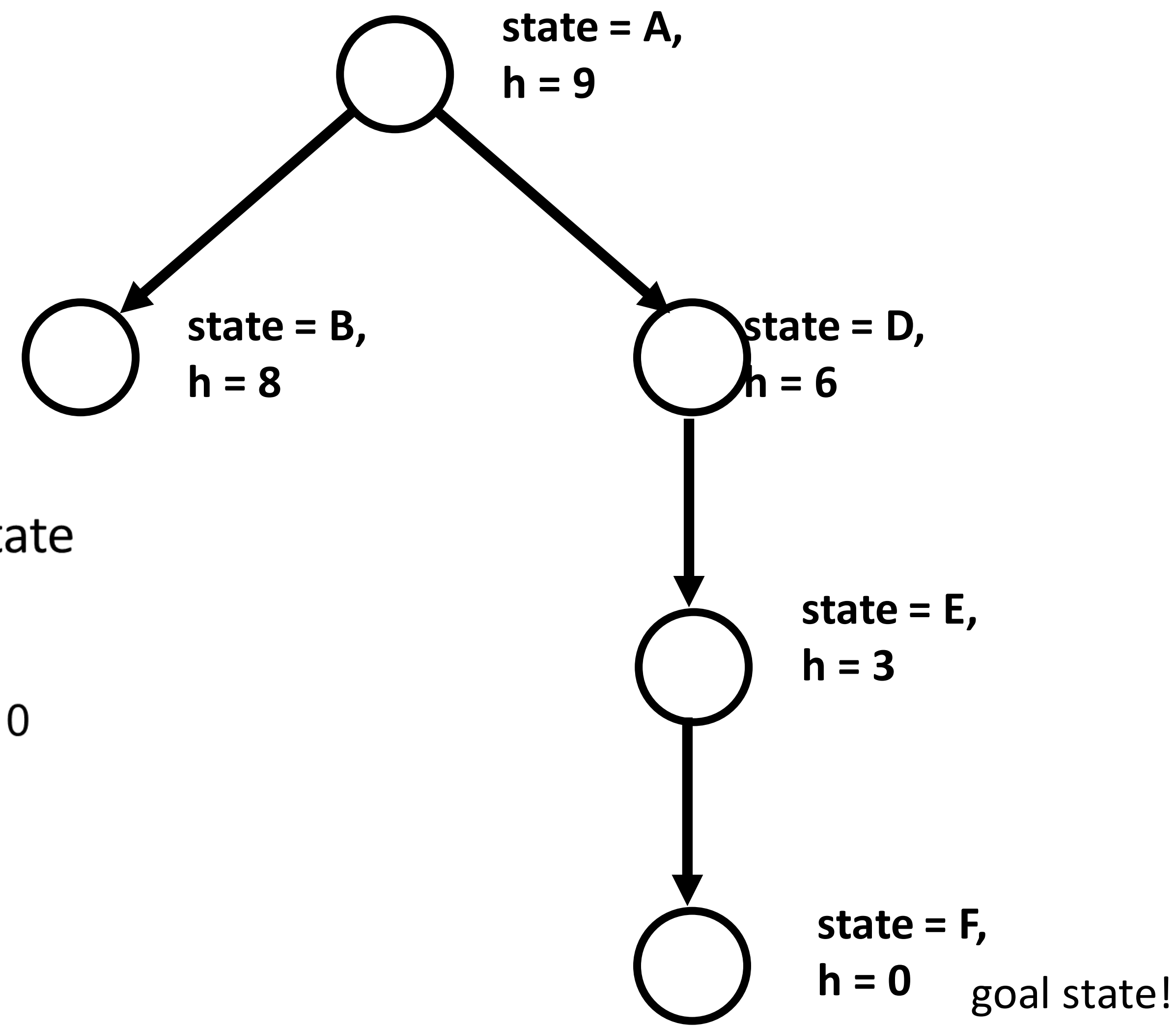
- Rapidly finds the optimal solution!

Best-First Search

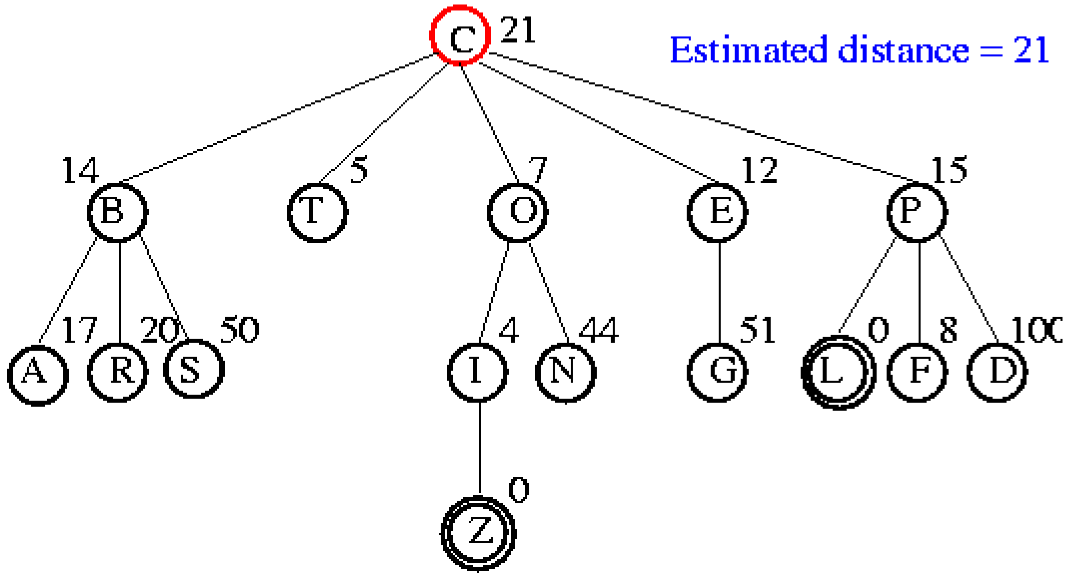
- expand nodes with lowest h values first



- Say: h(A) = 9, h(B) = 8, h(C) = 9, h(D) = 6, h(E) = 3, h(F) = 0
  - Typically assume that h is 0 at goal states

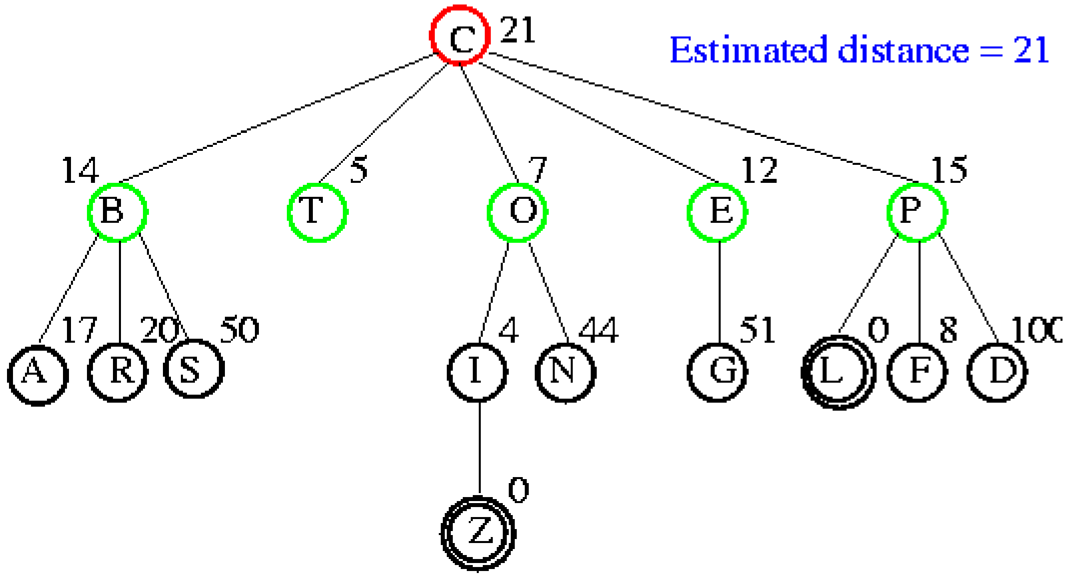


Example



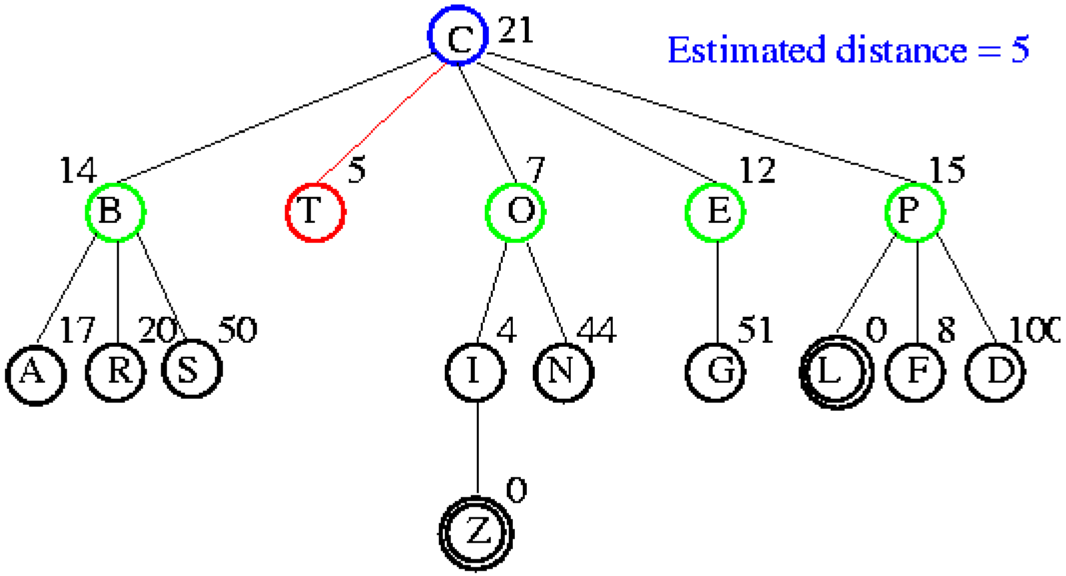
C 21
Estimated distance = 21
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Example



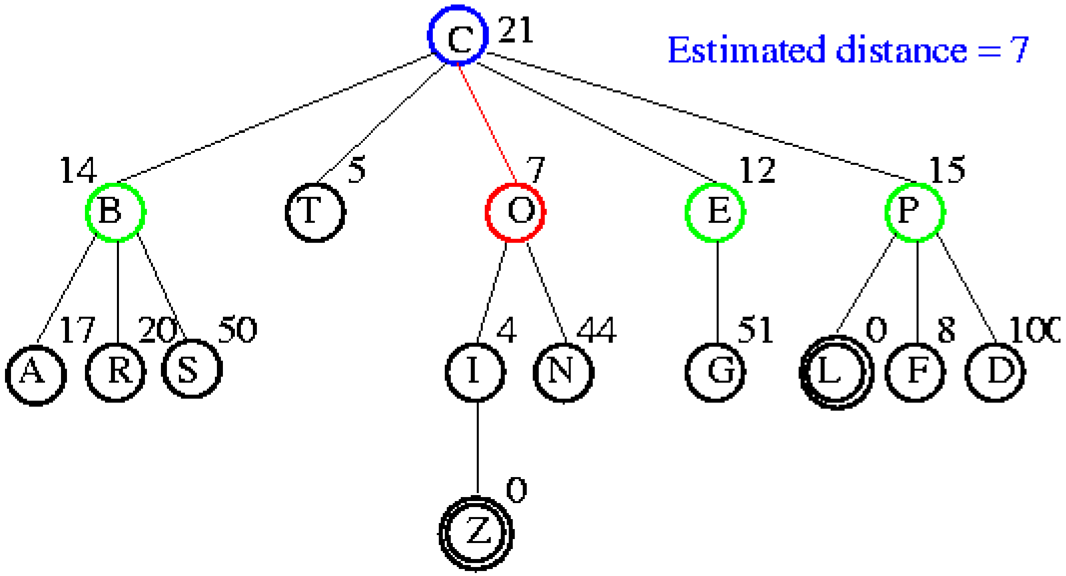
C 21
Estimated distance = 21
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Example



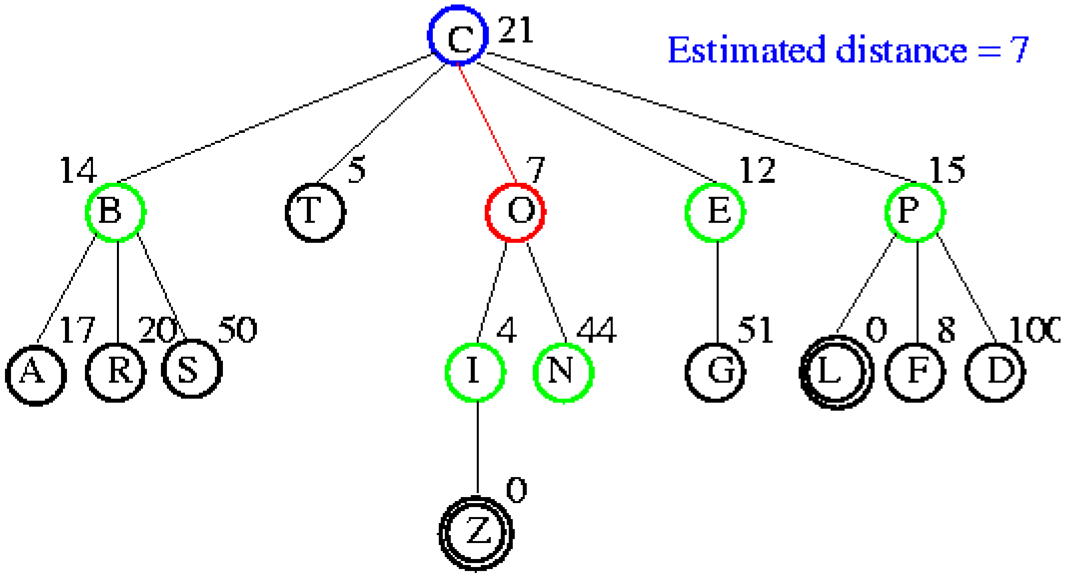
C
21
Estimated distance = 5
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example



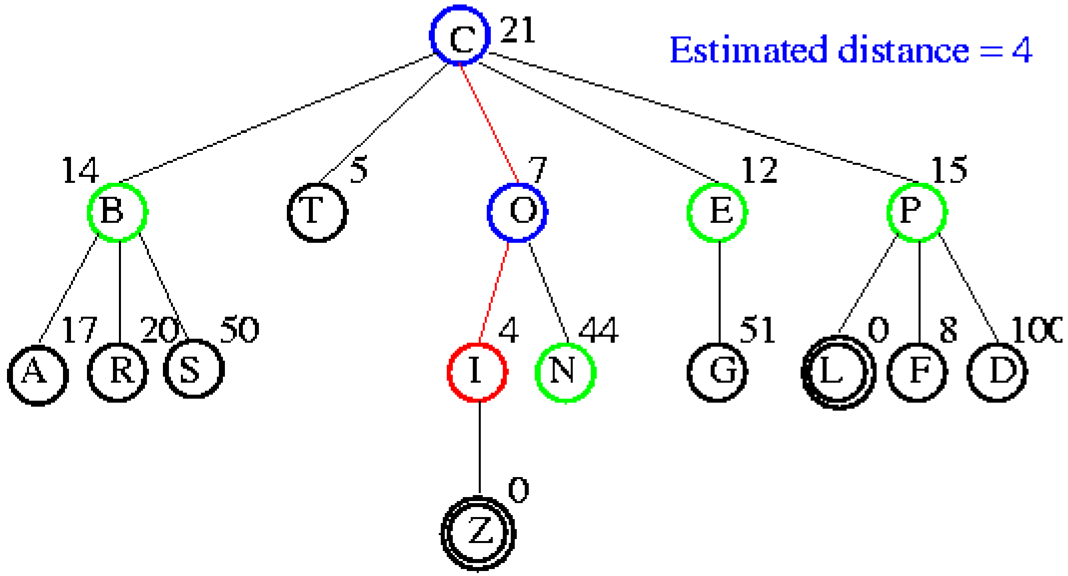
C
21
Estimated distance = 7
14
B
5
T
7
O
12
E
15
P
17
20
50
A
R
S
4
44
I
N
51
G
0
8
100
L
F
D
0
Z

Example

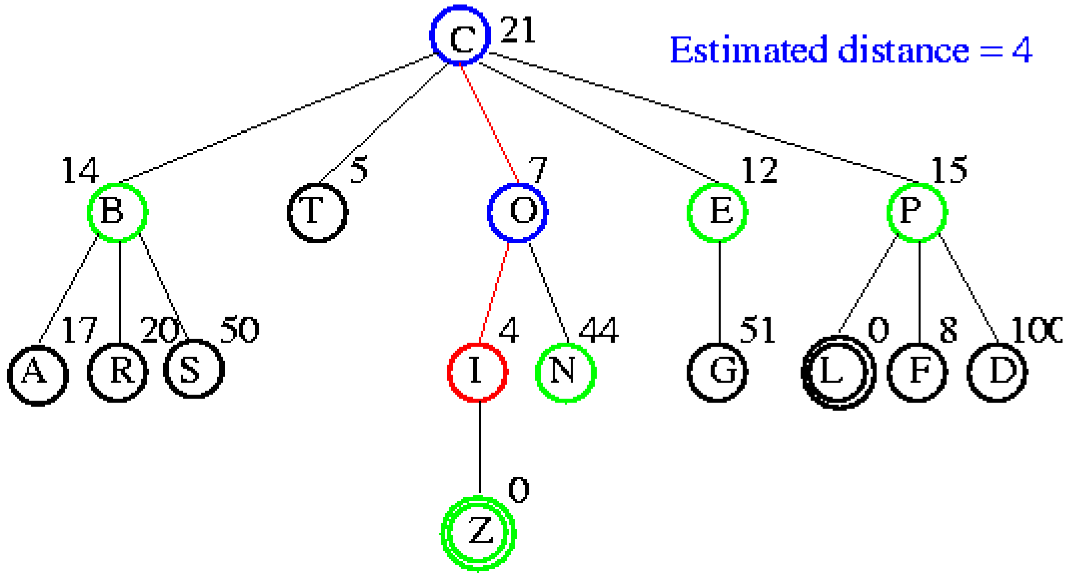


C
21
Estimated distance = 7
14
B
5
T
7
O
12
E
15
P
17
20
50
A
R
S
4
44
I
N
51
G
0
8
10C
L
F
D
0
Z

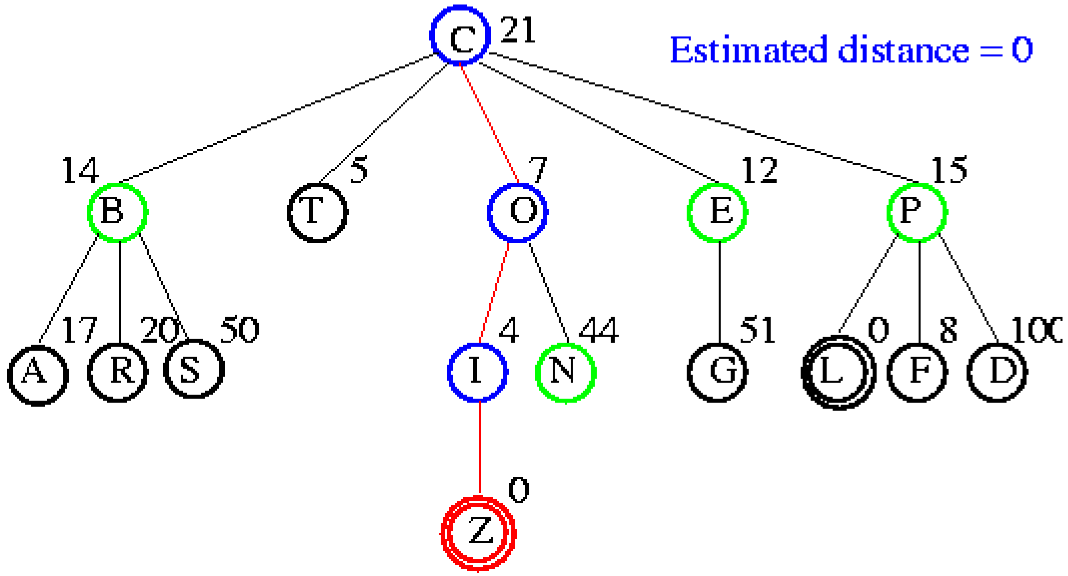
Example


C
21
Estimated distance = 4
14
B
5
T
7
O
12
E
15
P
17
20
50
A
R
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example

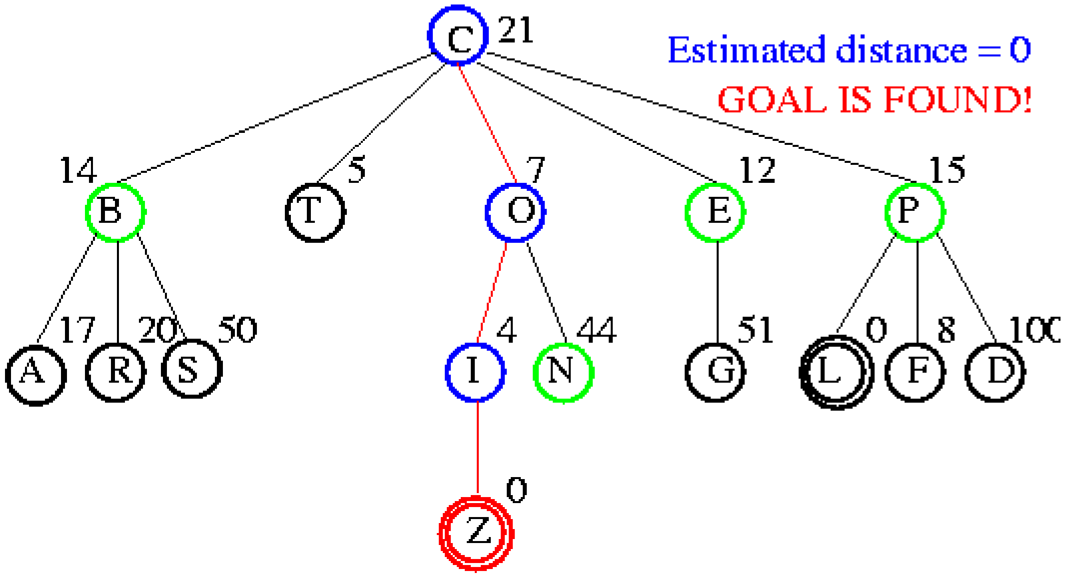

C 21
Estimated distance = 4
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example



C 21
Estimated distance = 0
14
B
T 5
7
O
12
E
15
P
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

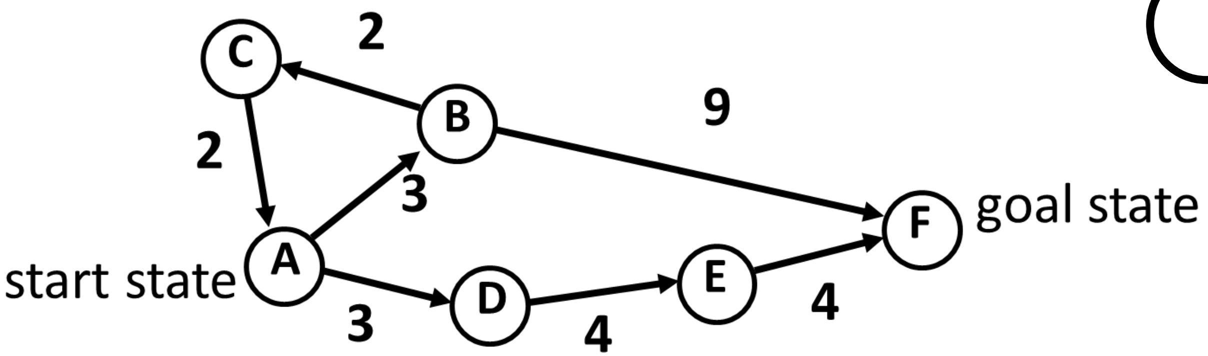
Example

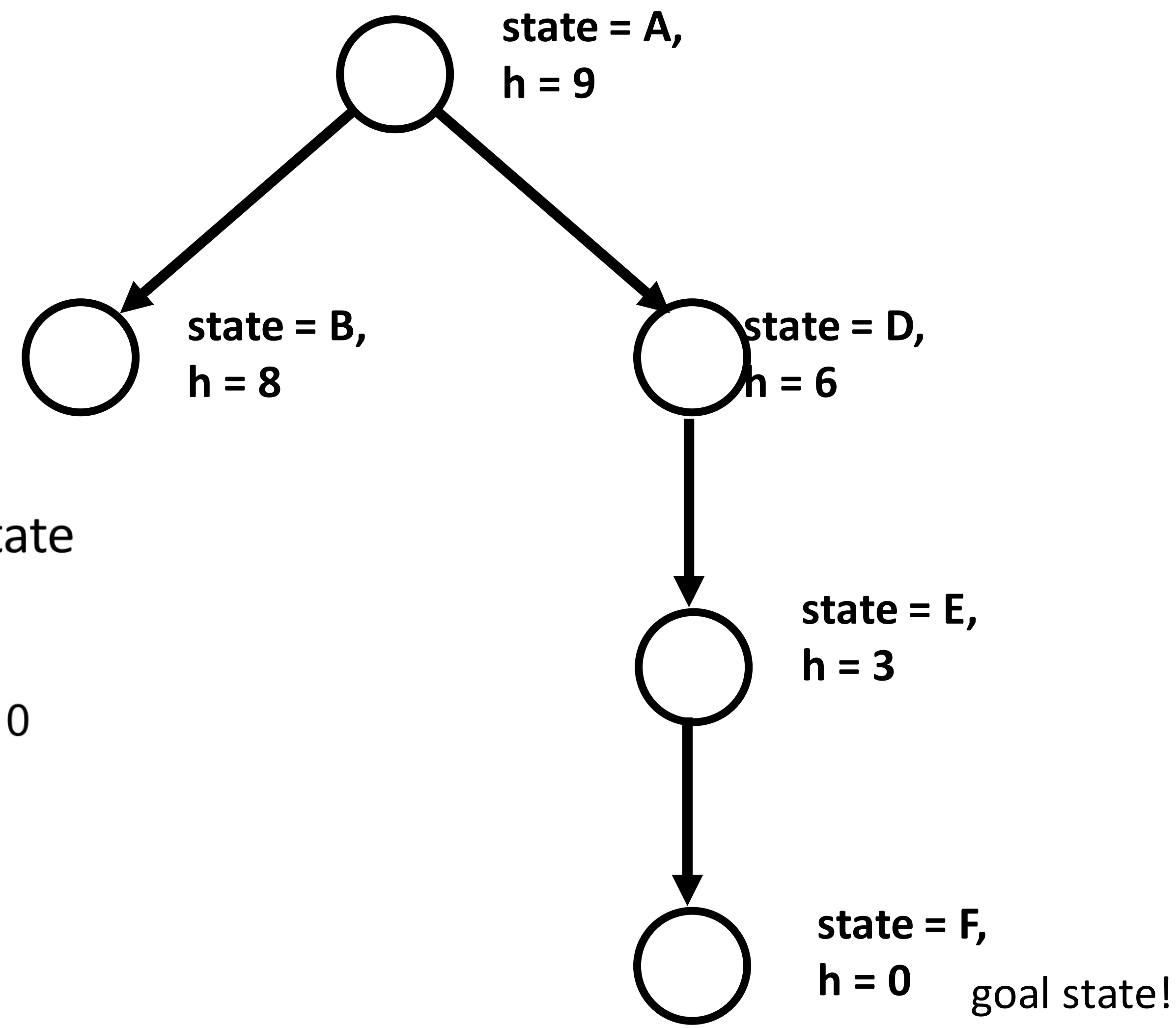


C 21
Estimated distance = 0
GOAL IS FOUND!
14 B
T 5
7 O
12 E
15 P
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 10C
Z 0

Best-First Search

- expand nodes with lowest h values first

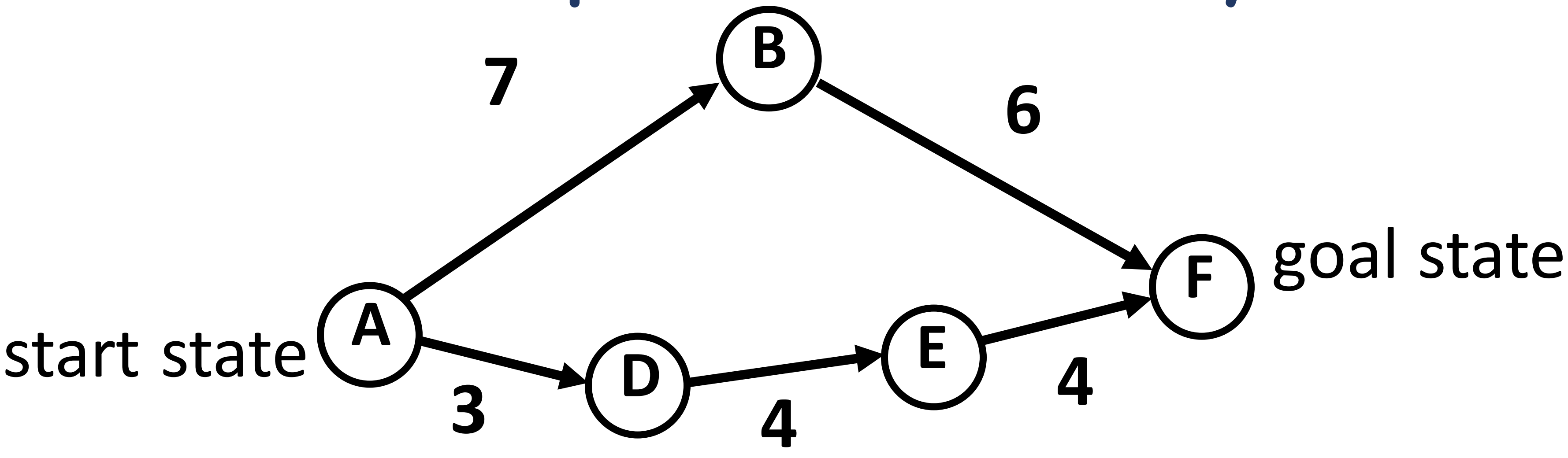




- Say: h(A) = 9, h(B) = 8, h(C) = 9, h(D) = 6, h(E) = 3, h(F) = 0
  - Typically assume that h is 0 at goal states

- Rapidly finds the optimal solution!
- Does it always?

A Bad Example for Greedy

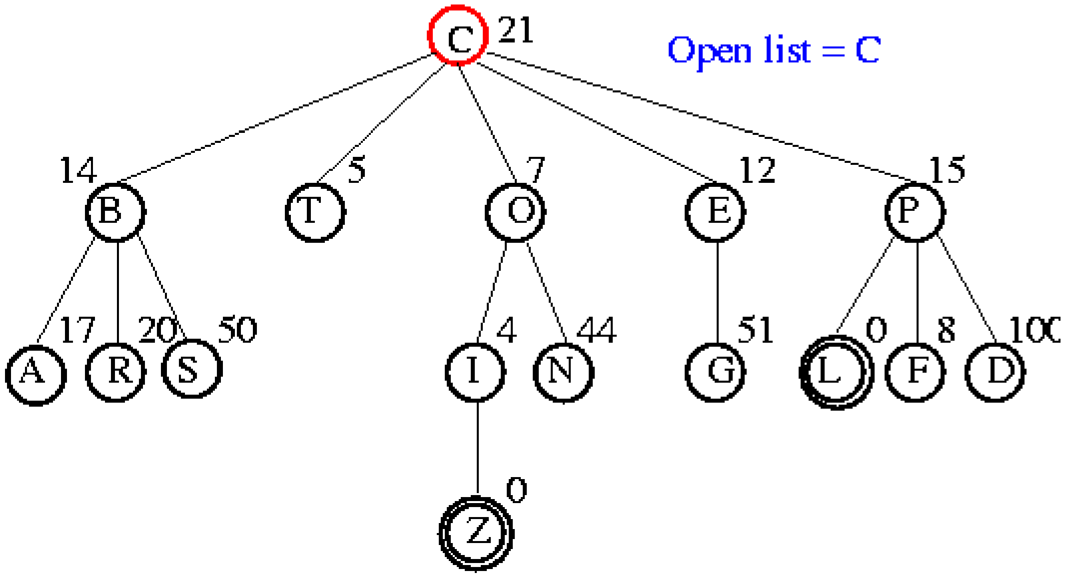


- Say: h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0
- Problem: greedy evaluates the promise of a node only by how far is left to go, does not take cost incurred already into account

Beam Search

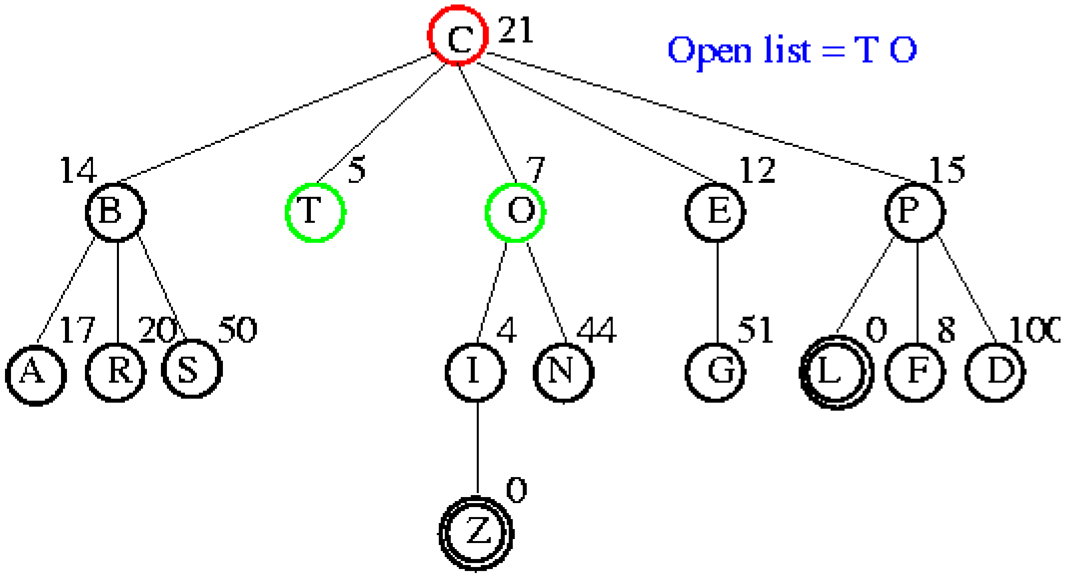
- A version of Best-First Search that uses only a predetermined number of best nodes.
- Limits its search to the kk most promising nodes, ensuring less memory usage.
  - **Pros**: Memory efficient.
  - **Cons**: May miss an optimal solution due to its limited scope.
  - n is the “beam width”
  - Only keep best (lowest-h) n nodes on open list

Example

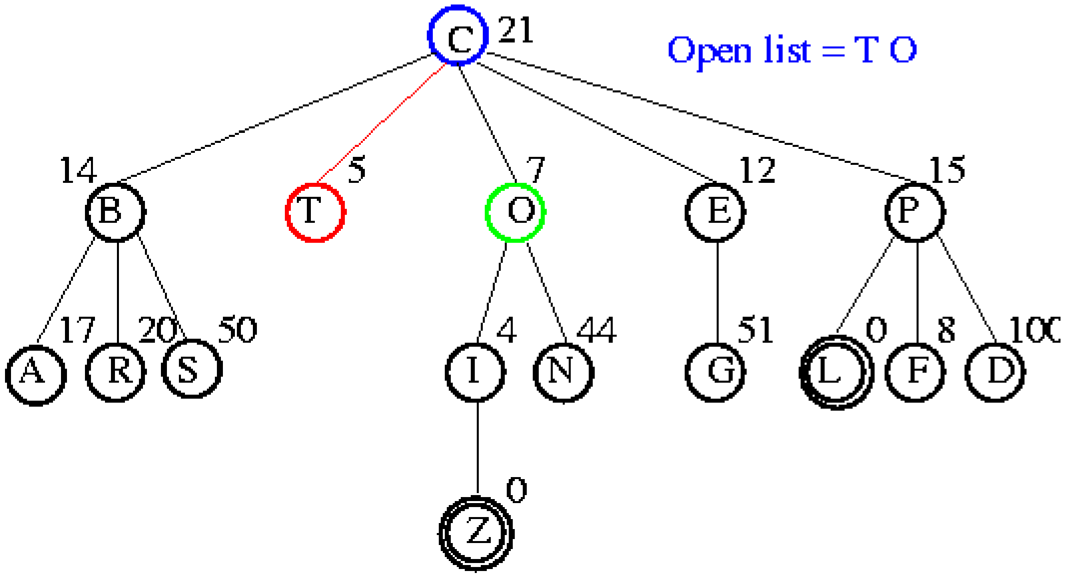


C 21
Open list = C
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

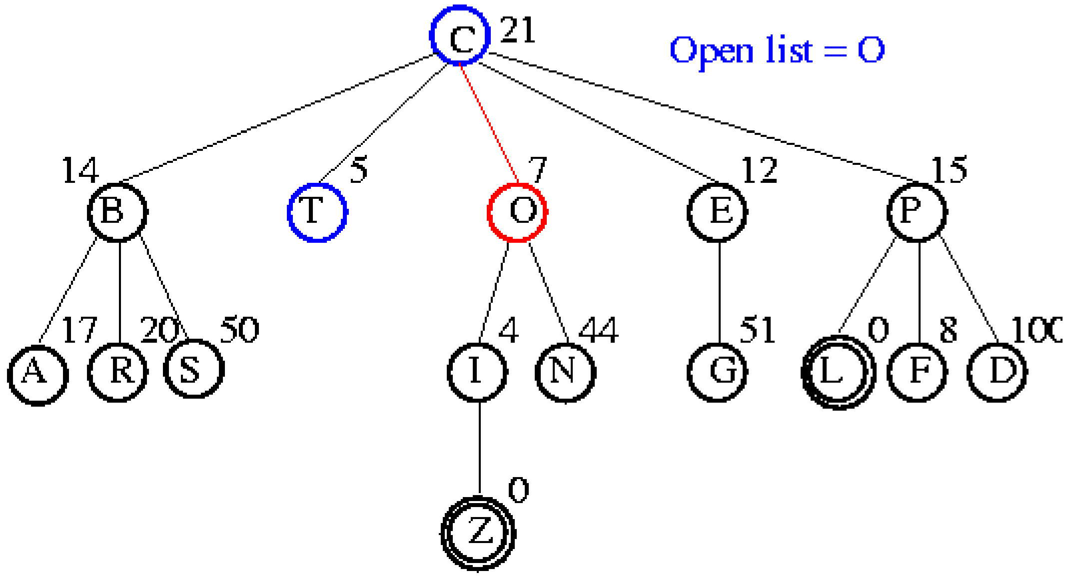
Example


C 21
Open list = T O
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

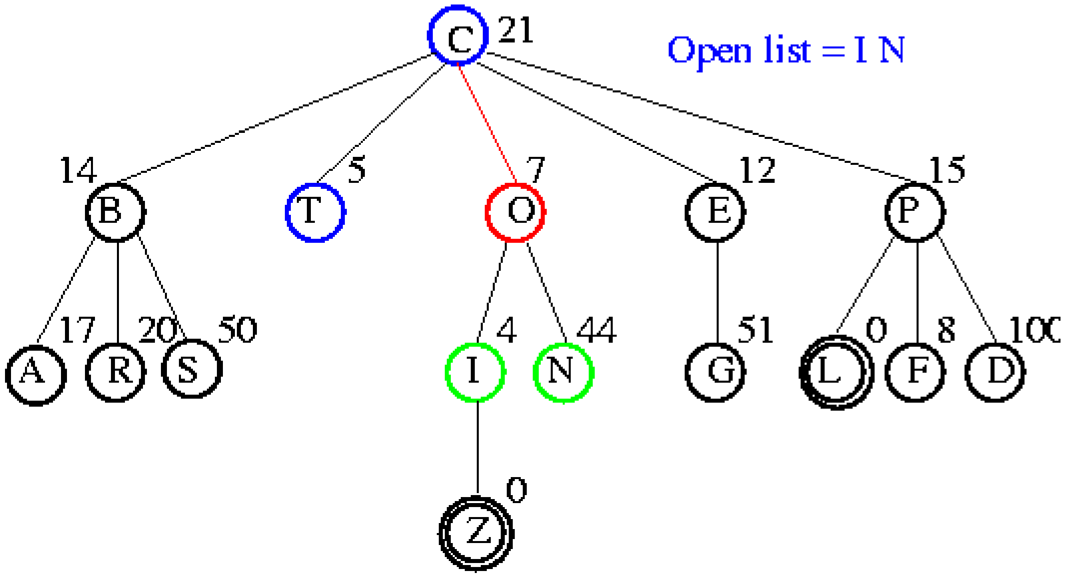
Example


C
21
Open list = T O
14
B
5
T
7
O
12
E
15
P
17
20
50
A
R
S
4
44
I
N
51
G
0
8
100
L
F
D
0
Z

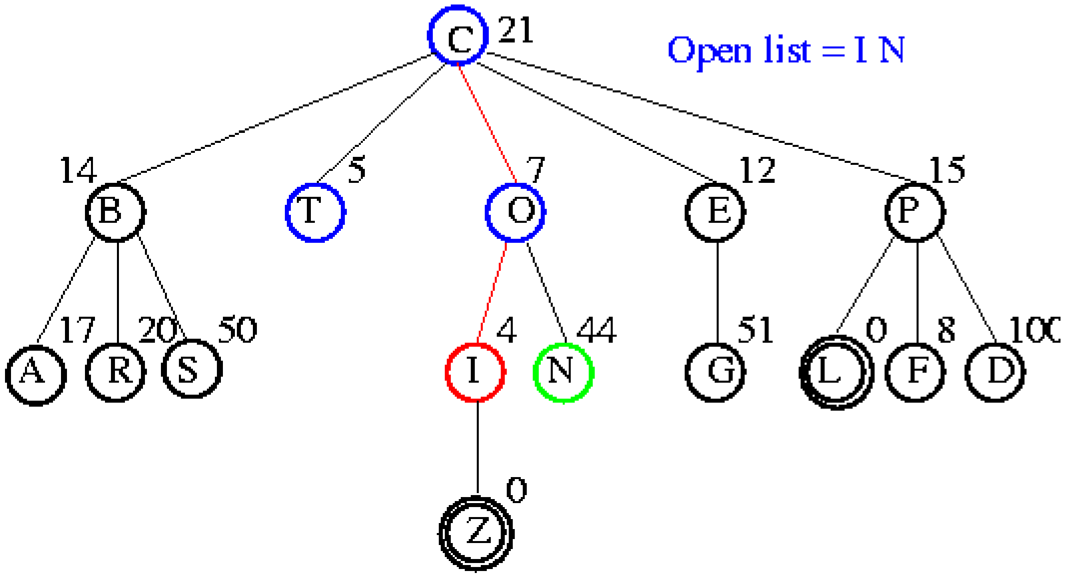
Example


C 21
Open list = O
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

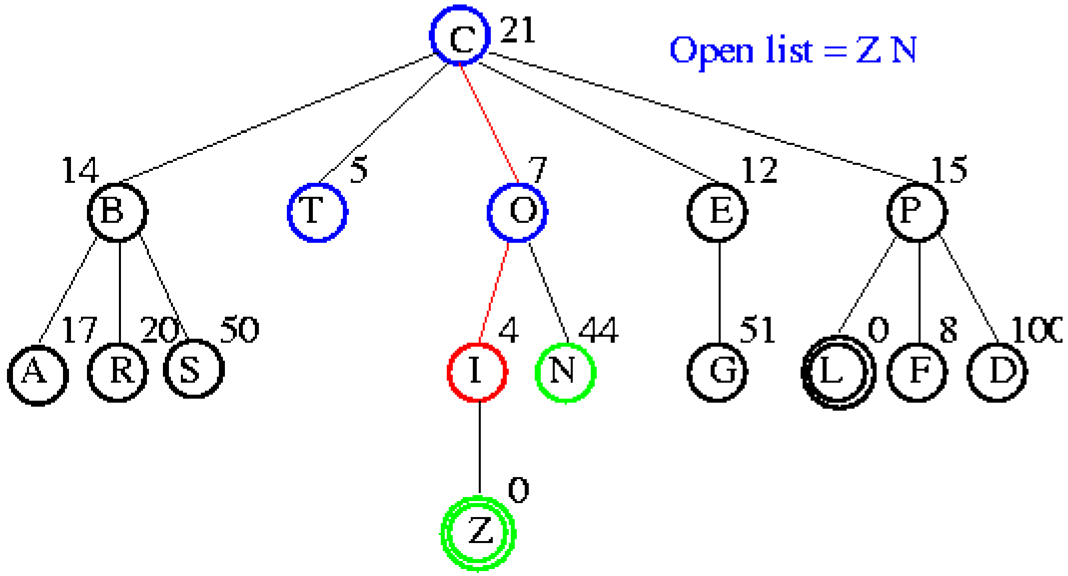
Example


C 21
Open list = I N
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Example

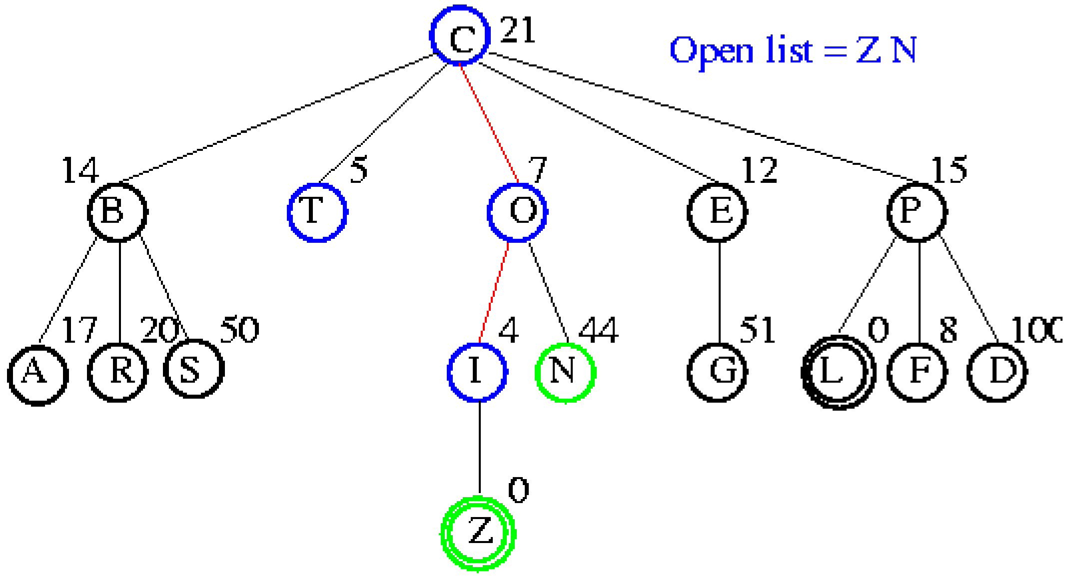

C 21
Open list = I N
B 14
T 5
O 7
E 12
P 15
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Example

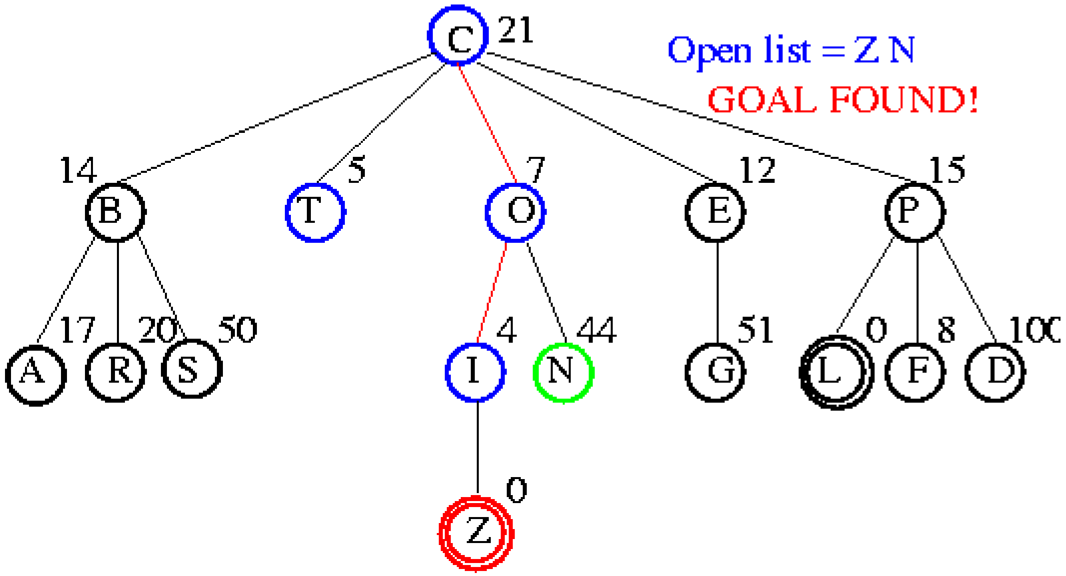


C 21
Open list = Z N
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example


C
21
Open list = Z N
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example

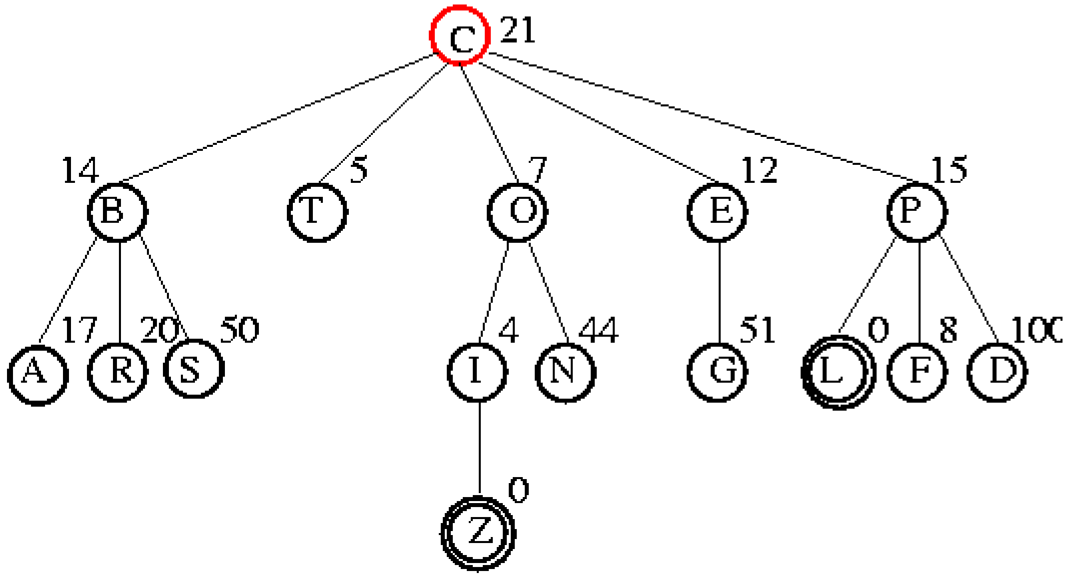


C 21
Open list = Z N
GOAL FOUND!
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

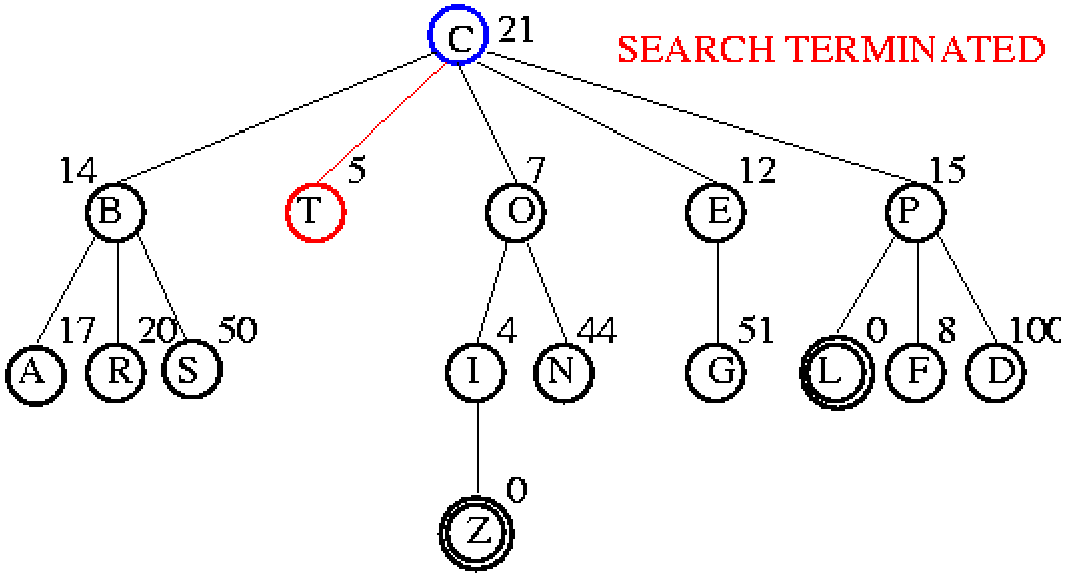
Hill Climbing

- A type of local search that starts with an arbitrary solution and iteratively makes small changes to the solution, selecting the neighbor with the highest fitness value.
  - **Pros**: Simple to implement.
  - **Cons**: Prone to getting stuck in local maxima.
  - n is the “beam width”
    - n = 1, Hill climbing
    - n = infinity, Best first search

Example



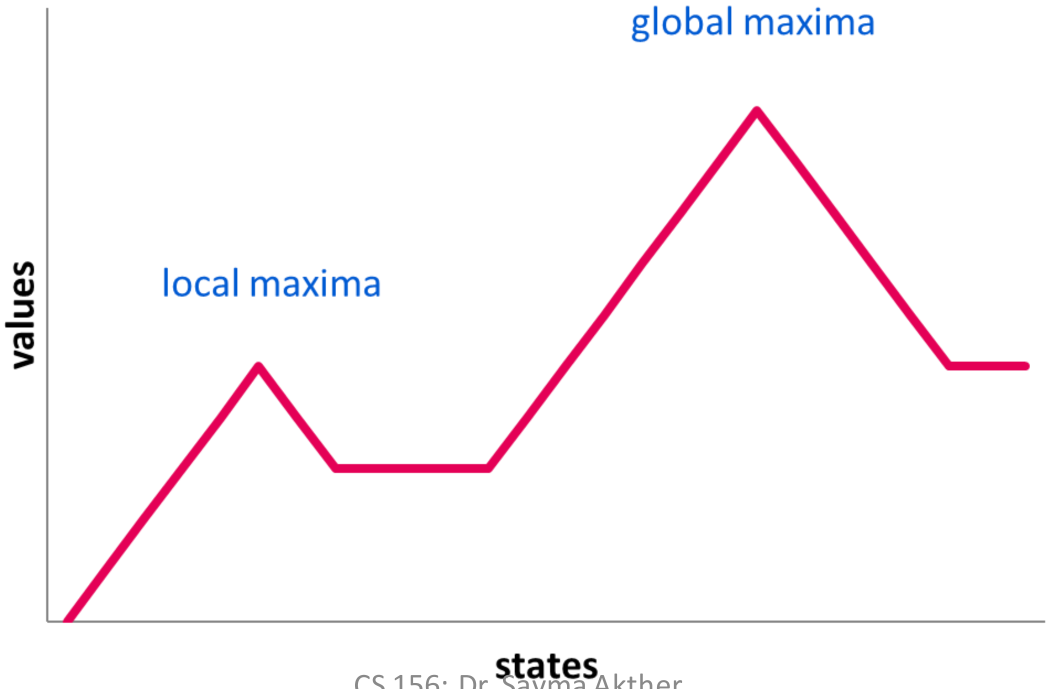
Example



C 21
SEARCH TERMINATED
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

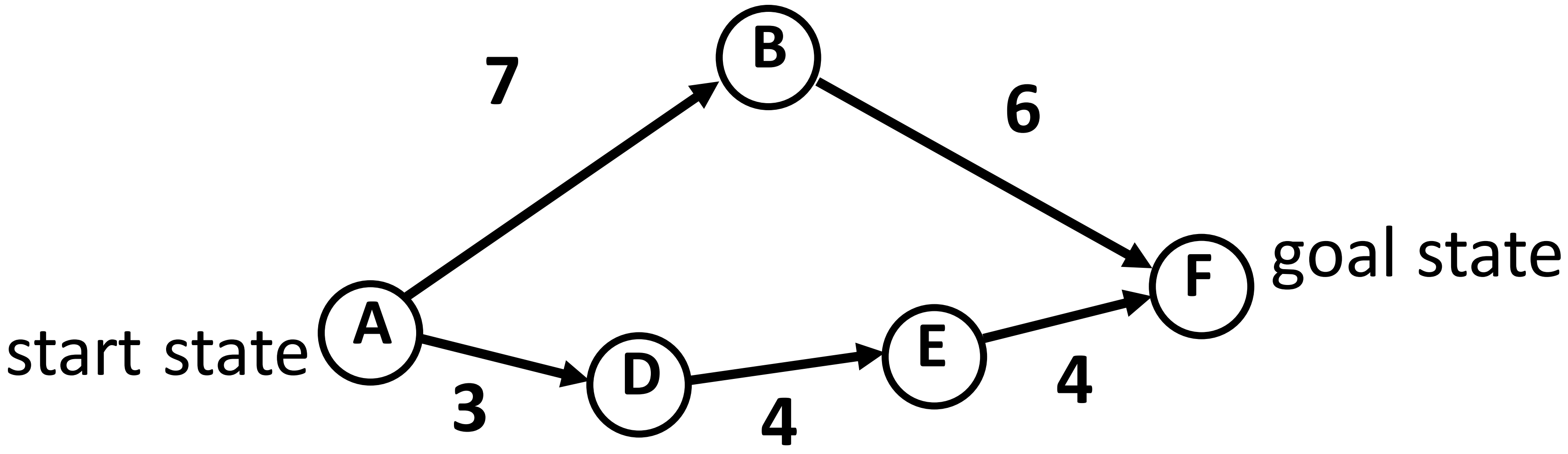
Hill Climbing Issues

- Also referred to as gradient descent
- Foothill problem / local maxima / local minima
- Can be solved with random walk or more steps



A* Search

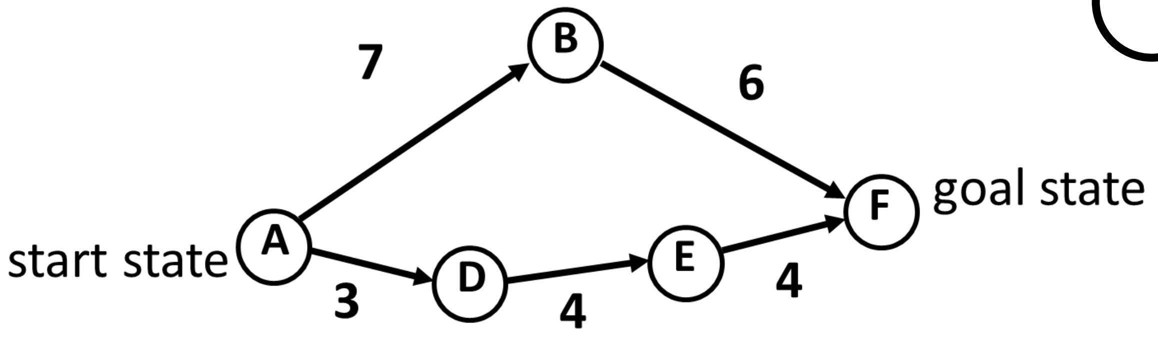
- Let g(n) be cost incurred already on path to n
- Expand nodes with lowest g(n) + h(n) first



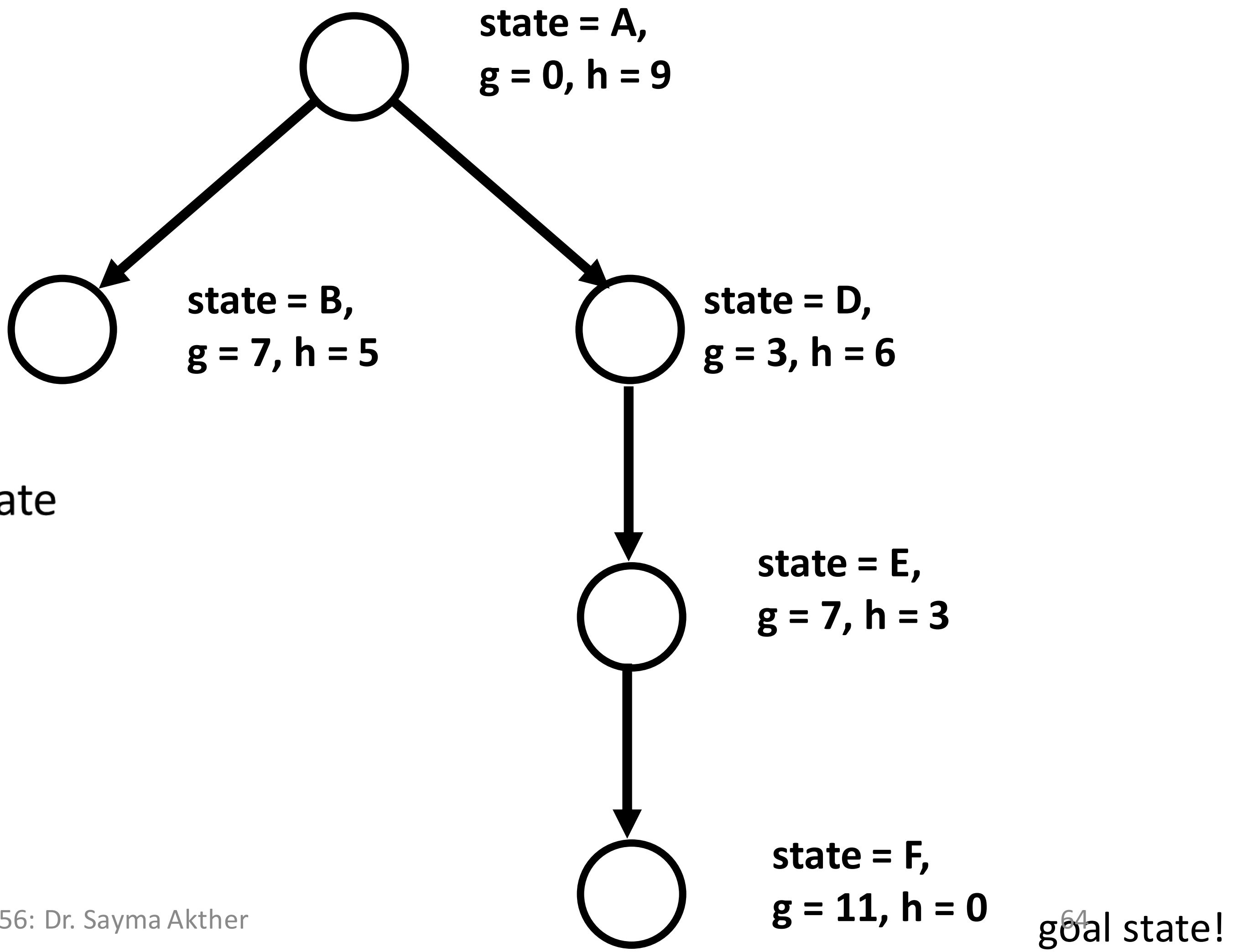
- Say: h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0
- Note: if h=0 everywhere, then just uniform cost search

A* Search

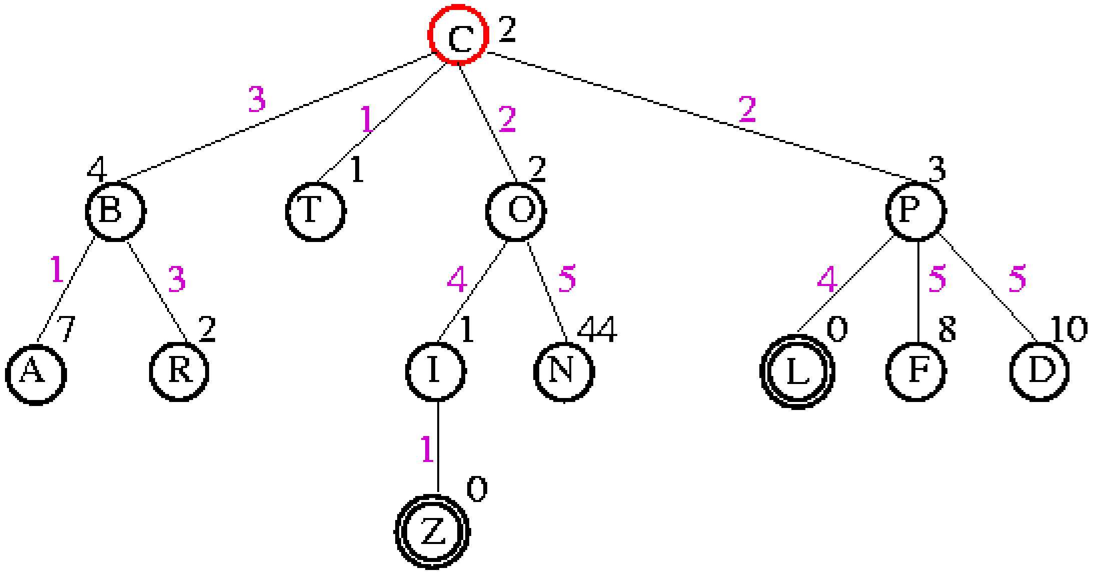
$f(N) = g(N) + h(N)$, where:
$g(N)$ = cost of best path found so far to N



- Say: h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0

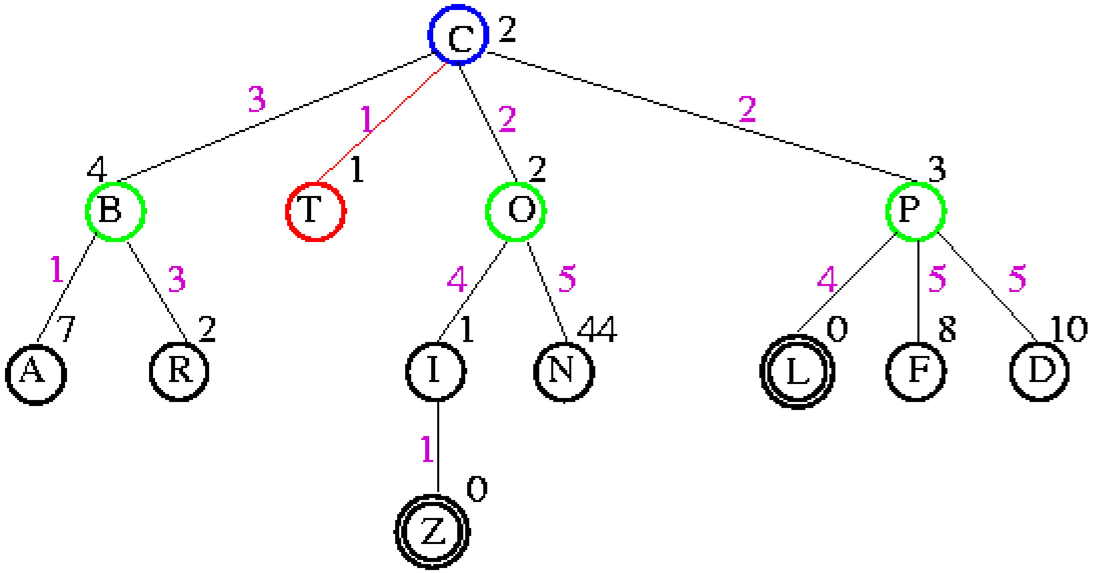


Example



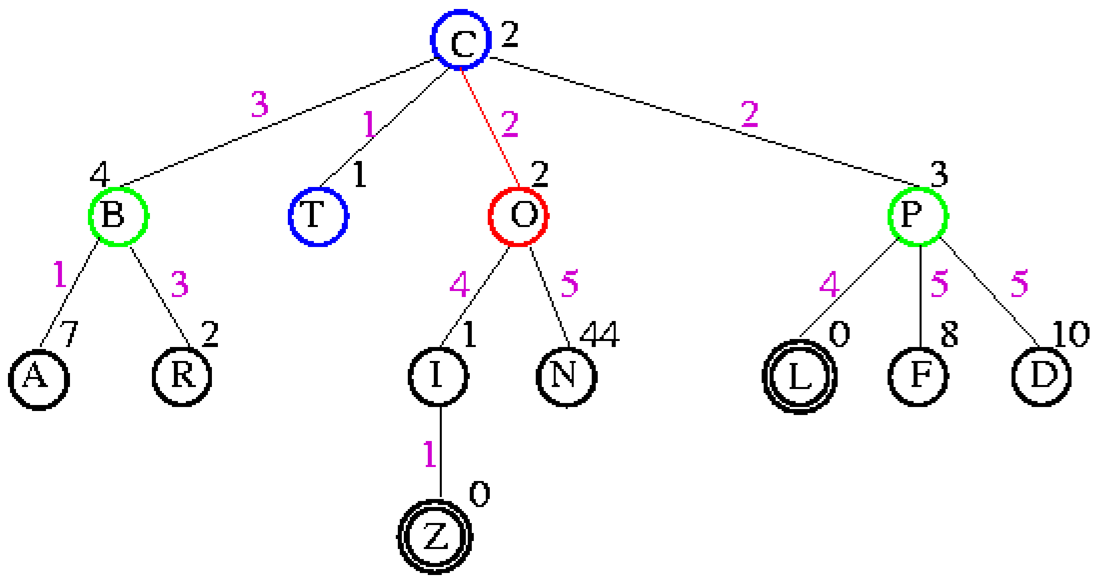
Open List = C (0+2=2)

Example



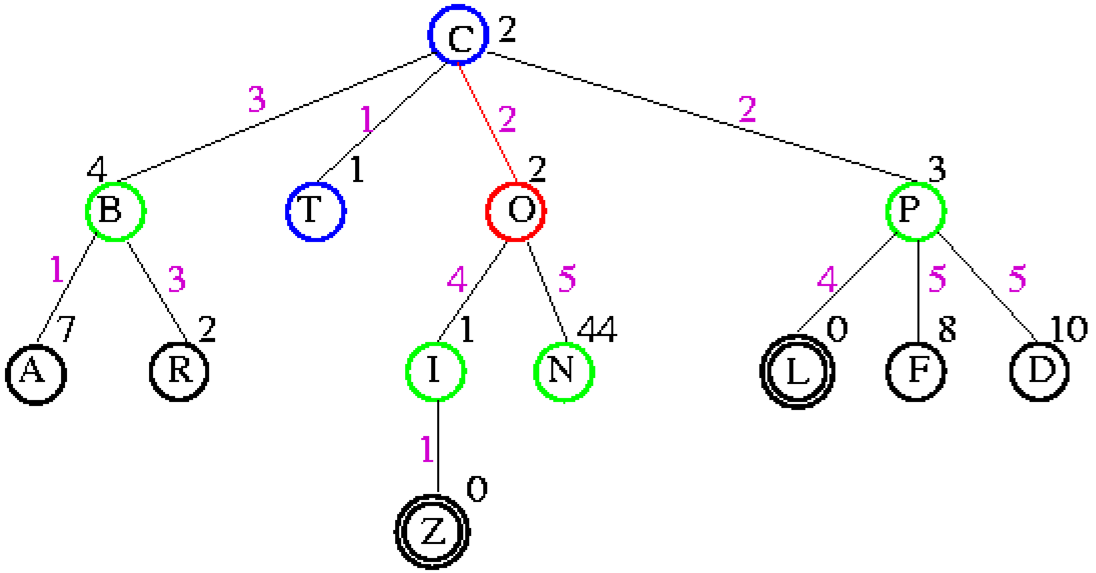
Open List = T (1+1=2), O (2+2=4), P (2+3=5), B(3+4=7)

Example



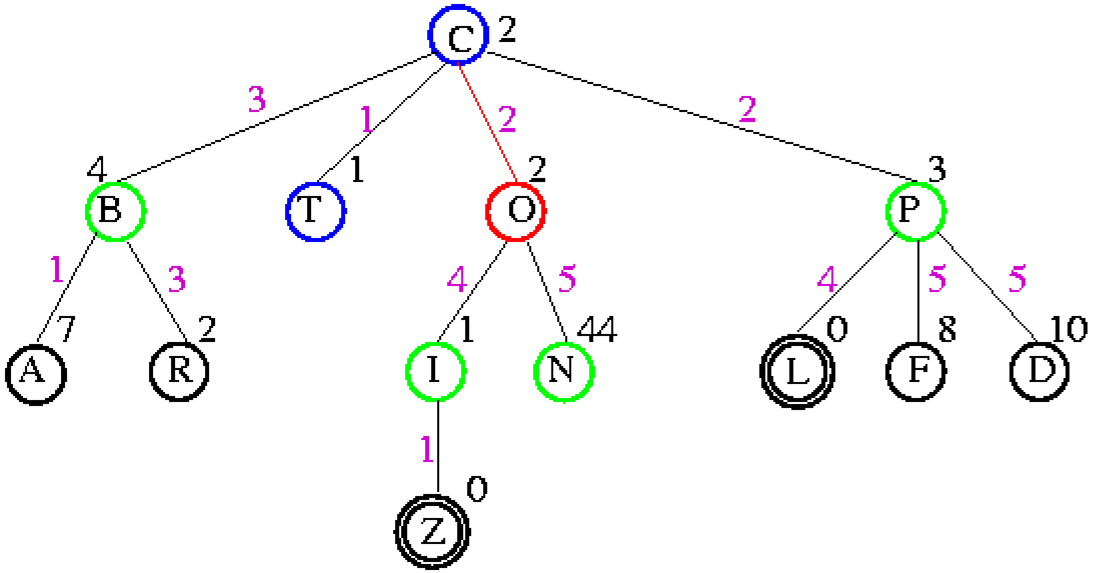
Open List = O (2+2=4), P (2+3=5), B(3+4=7)

Example



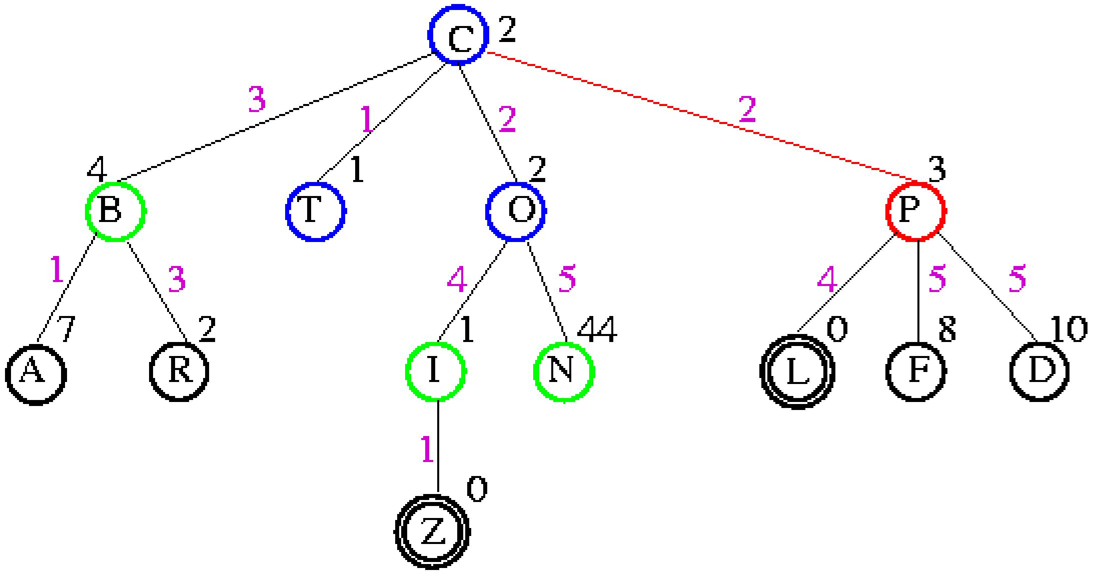
Open List = O (2+2=4), P (2+3=5), B(3+4=7)

Example



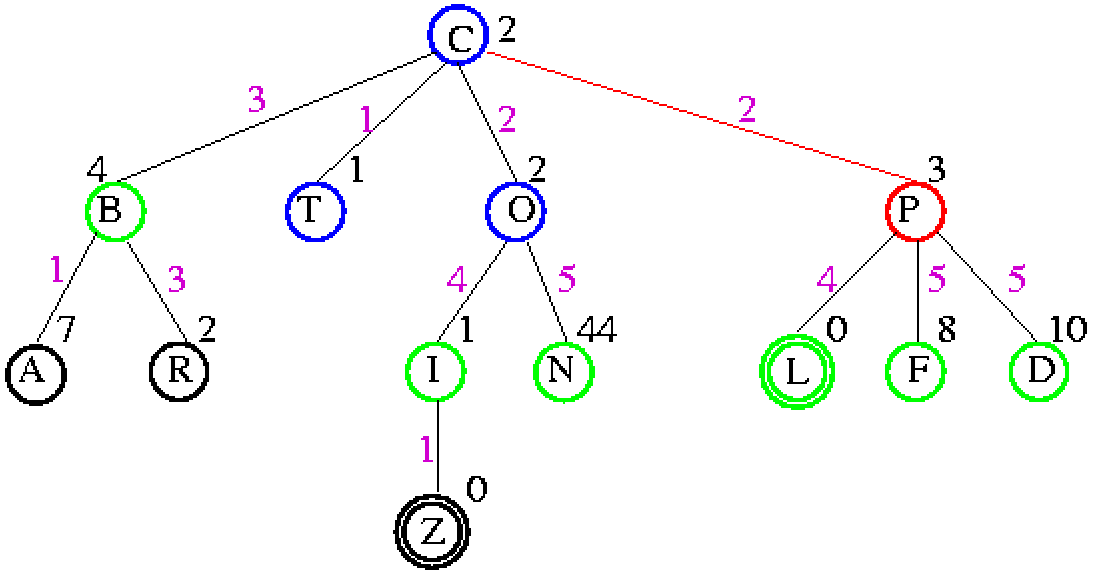
Open List = O (2+2=4), P (2+3=5), B(3+4=7)
I (6+1=7), N (7+44=51)

Example



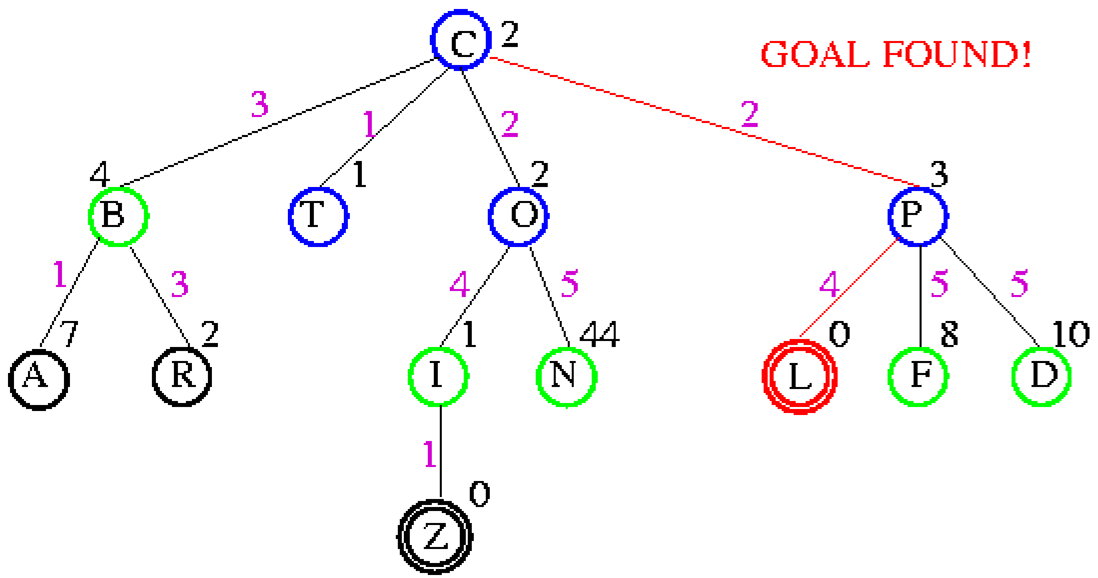
Open List = P (2+3=5), B(3+4=7)
I (6+1=7), N (7+44=51)

Example



Open List = P (2+3=5), L (6+0=6), B (3+4=7)
I (6+1=7), F (7+8=15), D (7+10=17), N (7+44=51)

Example



Open List = L (6+0=6), B (3+4=7)
I (6+1=7), F (7+8=15), D (7+10=17), N (7+44=51)