Artificial Intelligence

K. N Toosi University of Technology

Course Instructor:

Dr. Omid Azarkasb

Teaching Assistants:

Atena Najaf Abadi Farahani

Saeed Mahmoudian

Informed Search

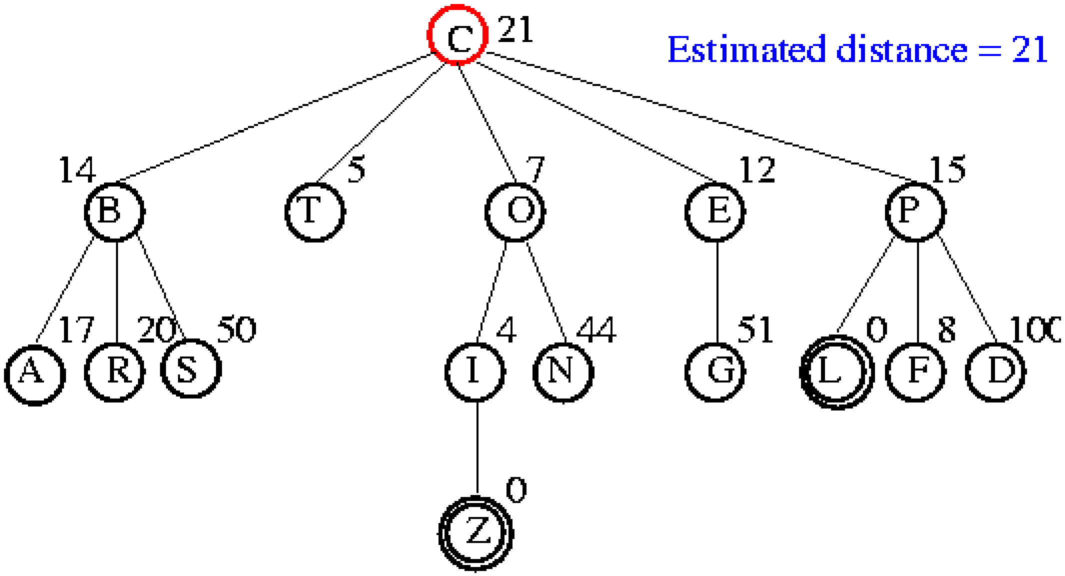
Informed Searches

- Best-first search, Hill climbing, Beam search, A*, IDA*, RBFS, SMA*
- New terms
  - Heuristics
  - Optimal solution
  - Informedness
  - Hill climbing problems
  - Admissibility
- New parameters
  - g(n) = estimated cost from initial state to state n
  - h(n) = estimated cost (distance) from state n to closest goal
  - h(n) is our heuristic
    - Robot path planning, h(n) could be Euclidean distance
    - 8 puzzle, h(n) could be #tiles out of place
- Search algorithms which use h(n) to guide search are heuristic search algorithms

Best-First Search

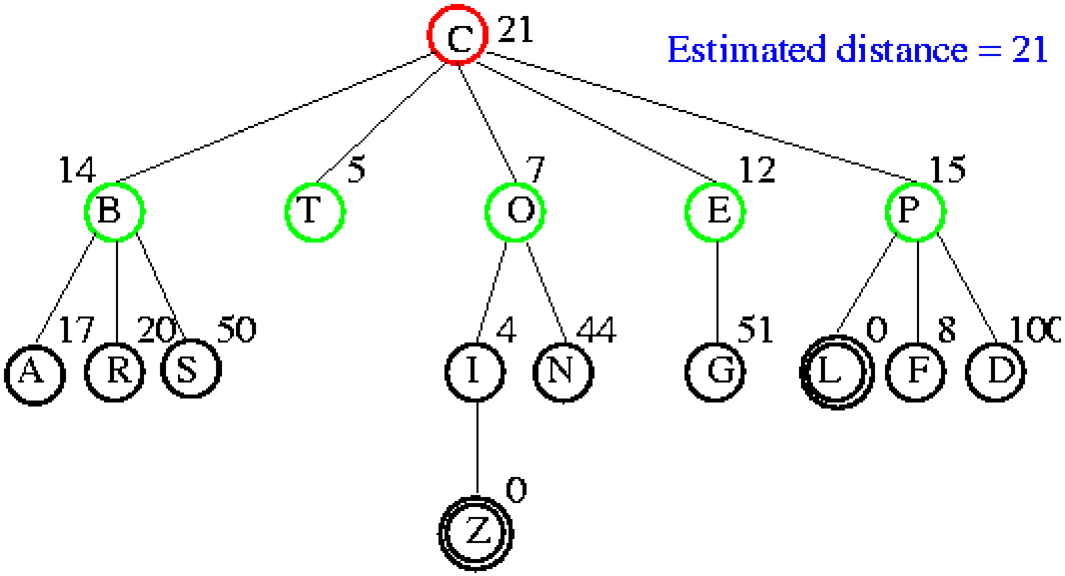
- QueueingFn is sort-by-h
- Best-first search only as good as heuristic
  - Example heuristic for 8 puzzle: Manhattan Distance

Example



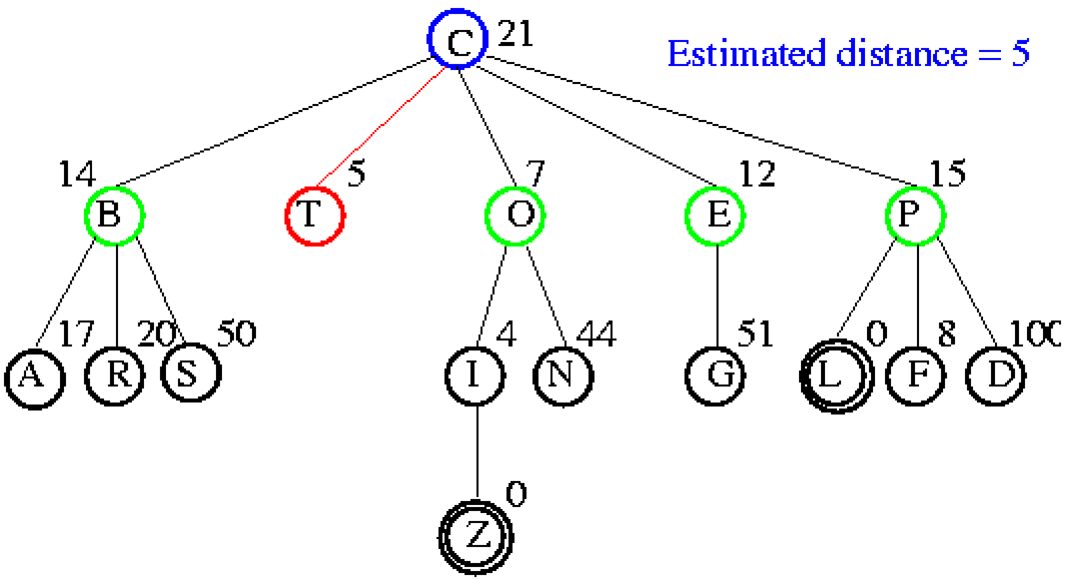
C
21
Estimated distance = 21
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example

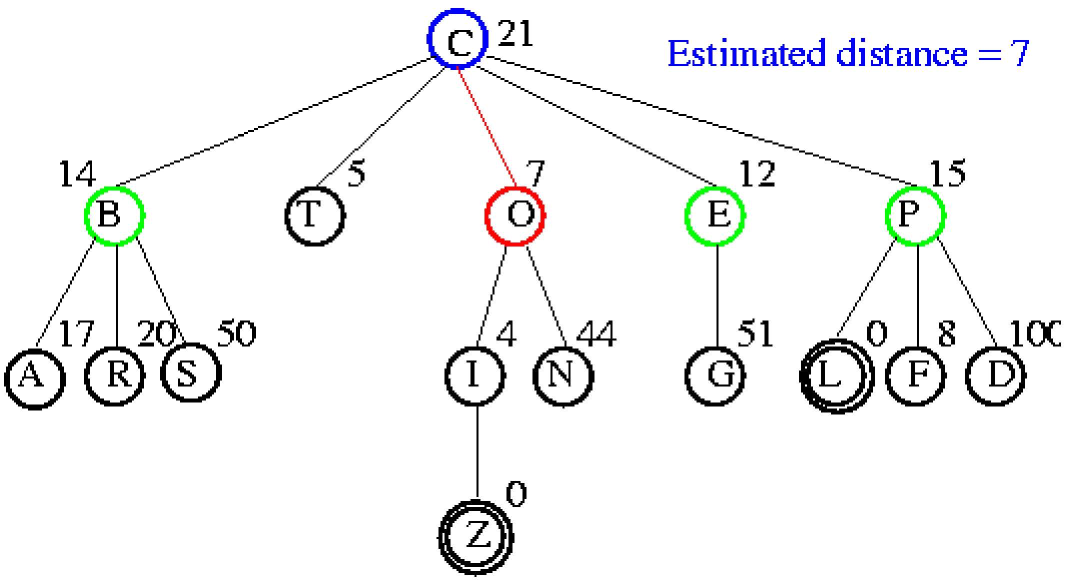


C
21
Estimated distance = 21
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

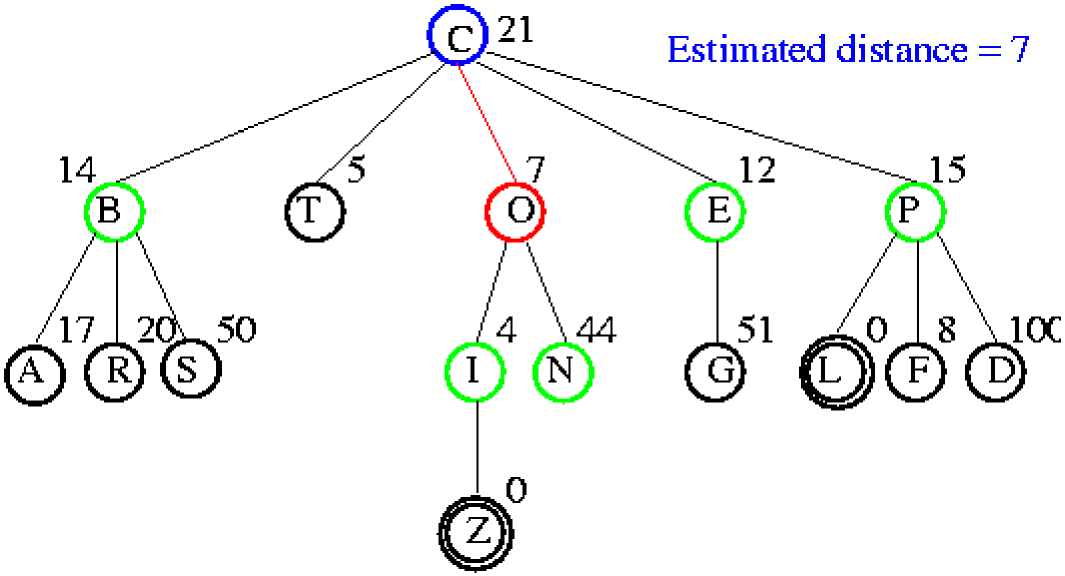
Example


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
10C D
0 Z

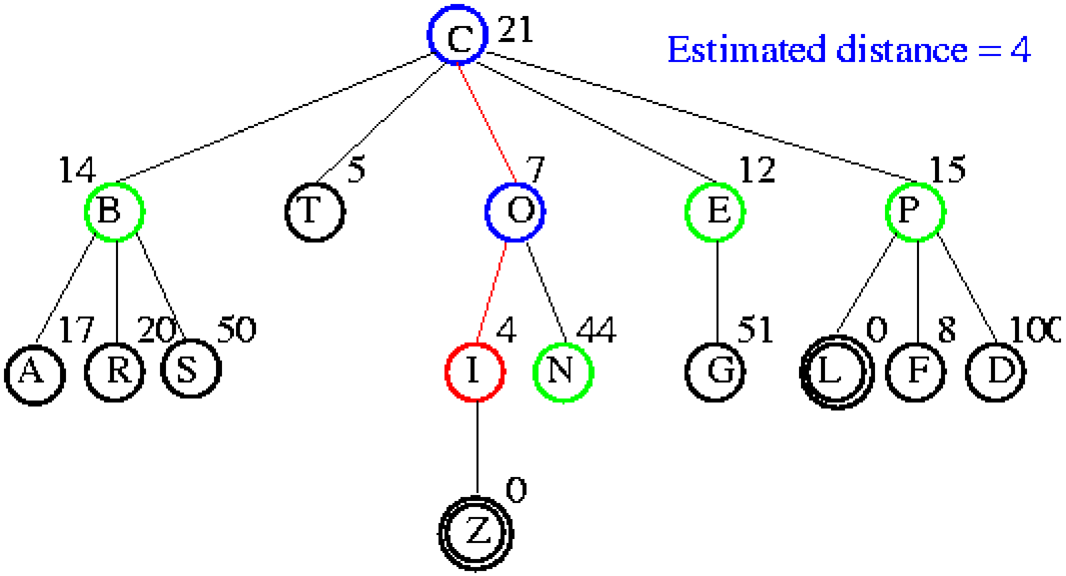
Example


C
21
Estimated distance = 7
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

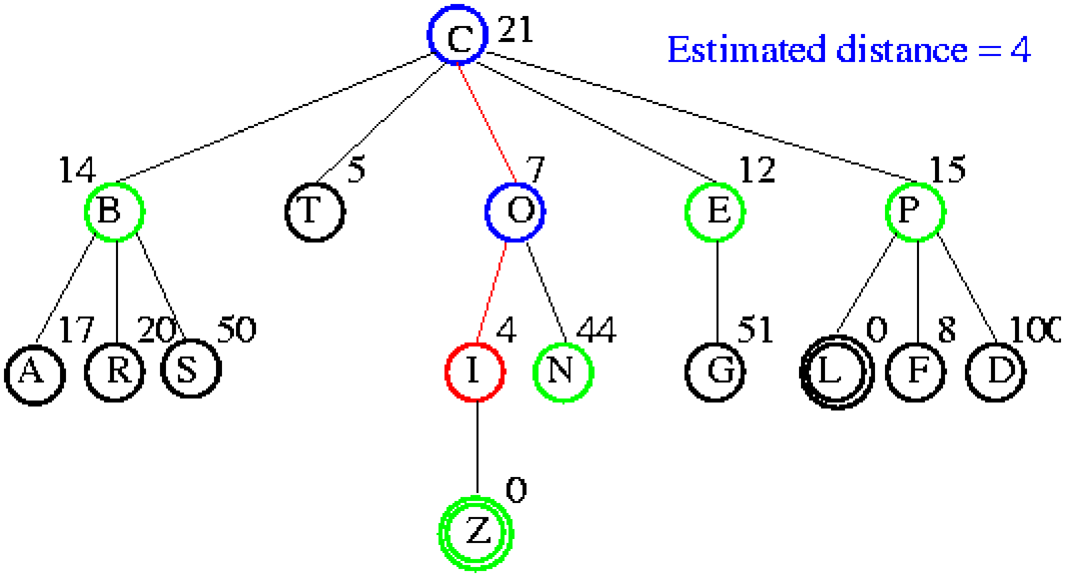
Example


C
21
Estimated distance = 7
14
B
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

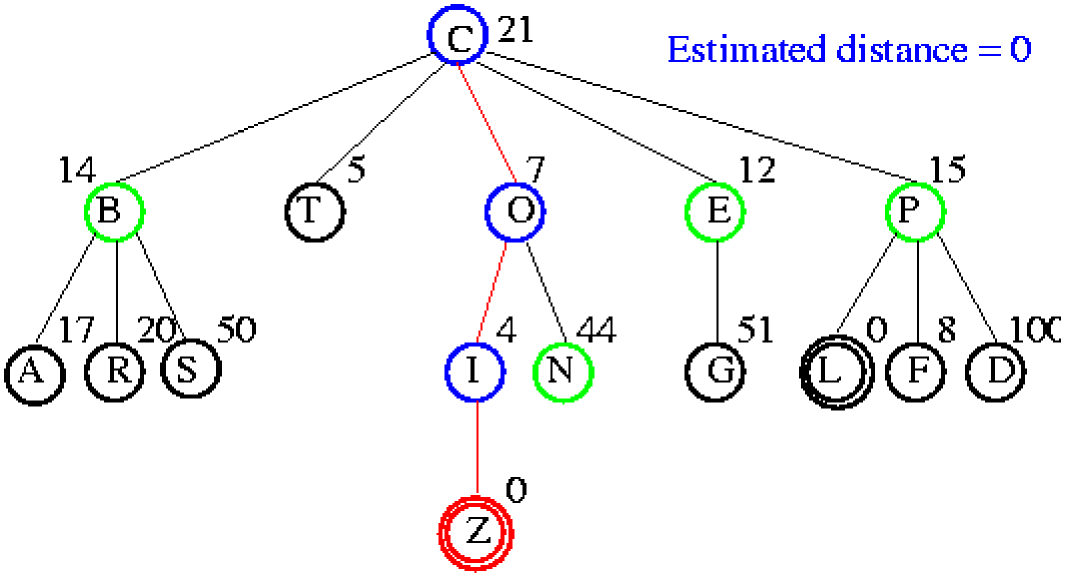

C 21
Estimated distance = 4
14 B
T 5
7 O
12 E
15 P
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Example



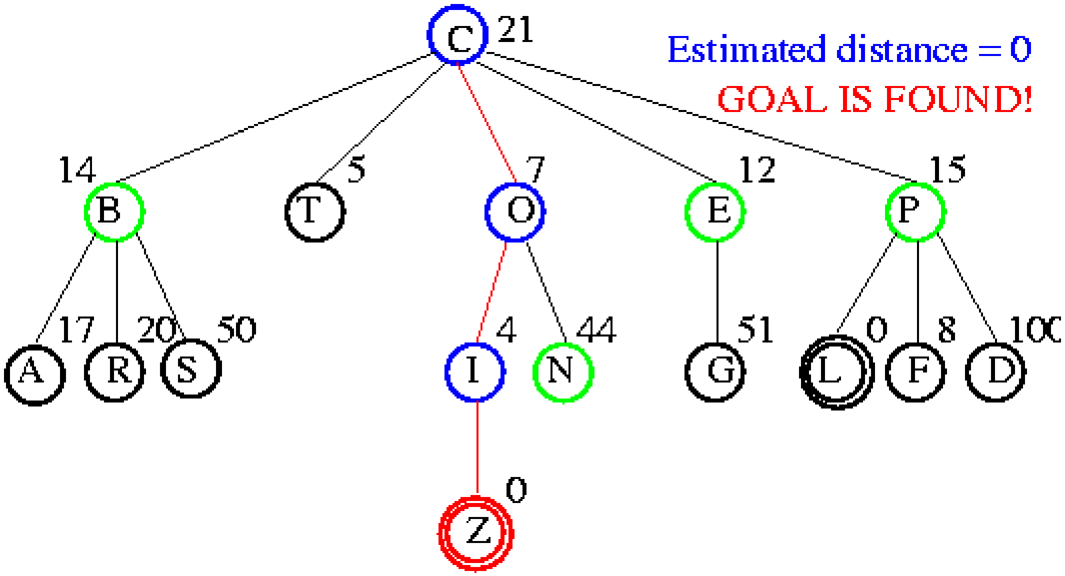
C 21
Estimated distance = 4
14
B
T 5
O 7
E 12
P 15
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Example



C 21
Estimated distance = 0
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
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
100
Z
0

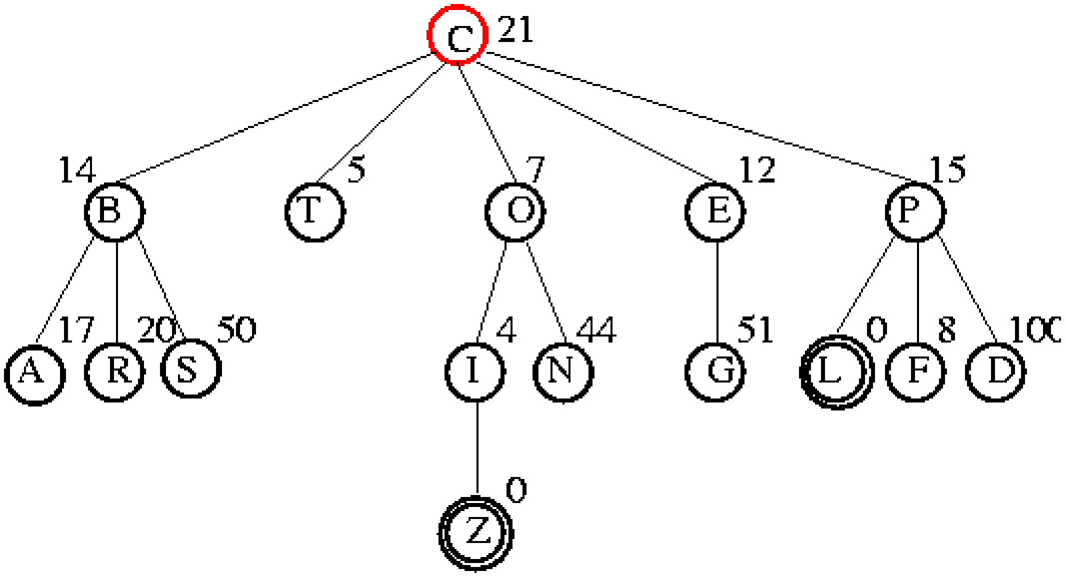
Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best
Complete	N	Y	Y	Y	N
Optimal	N	N	Y	N	N
Heuristic	N	N	N	N	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m
Space	bm	b^{d+1}	b^m	bd	b^m

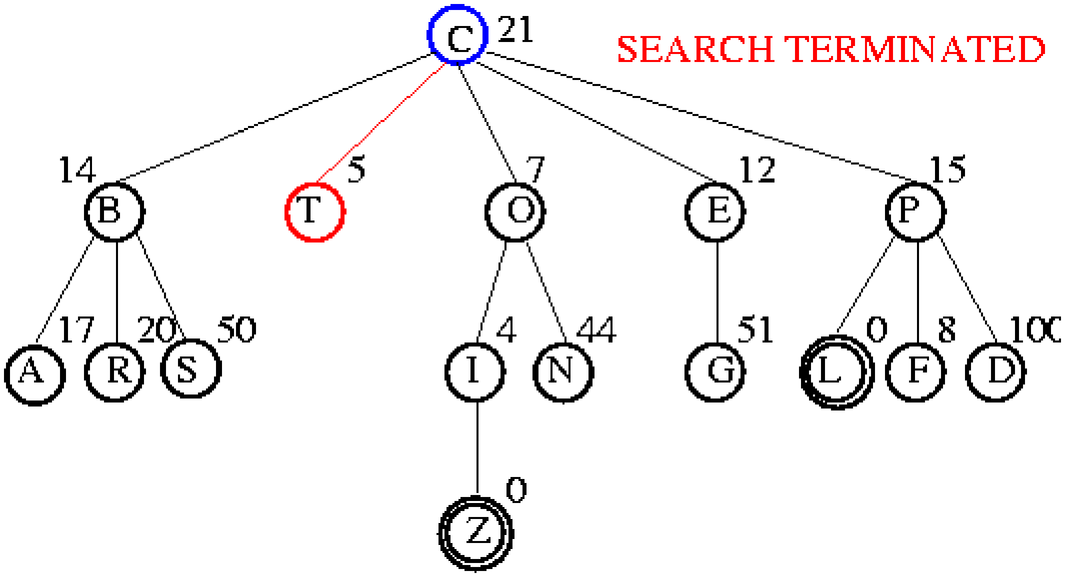
Hill Climbing (Greedy Search)

- QueueingFn is sort-by-h
  - Only keep lowest-h state on open list
- Best-first search is tentative
- Hill climbing is irrevocable
- Features
  - Much faster
  - Less memory
  - Dependent upon h(n)
  - If bad h(n), may prune away all goals
  - Not complete

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 (gradient ascent/descent)

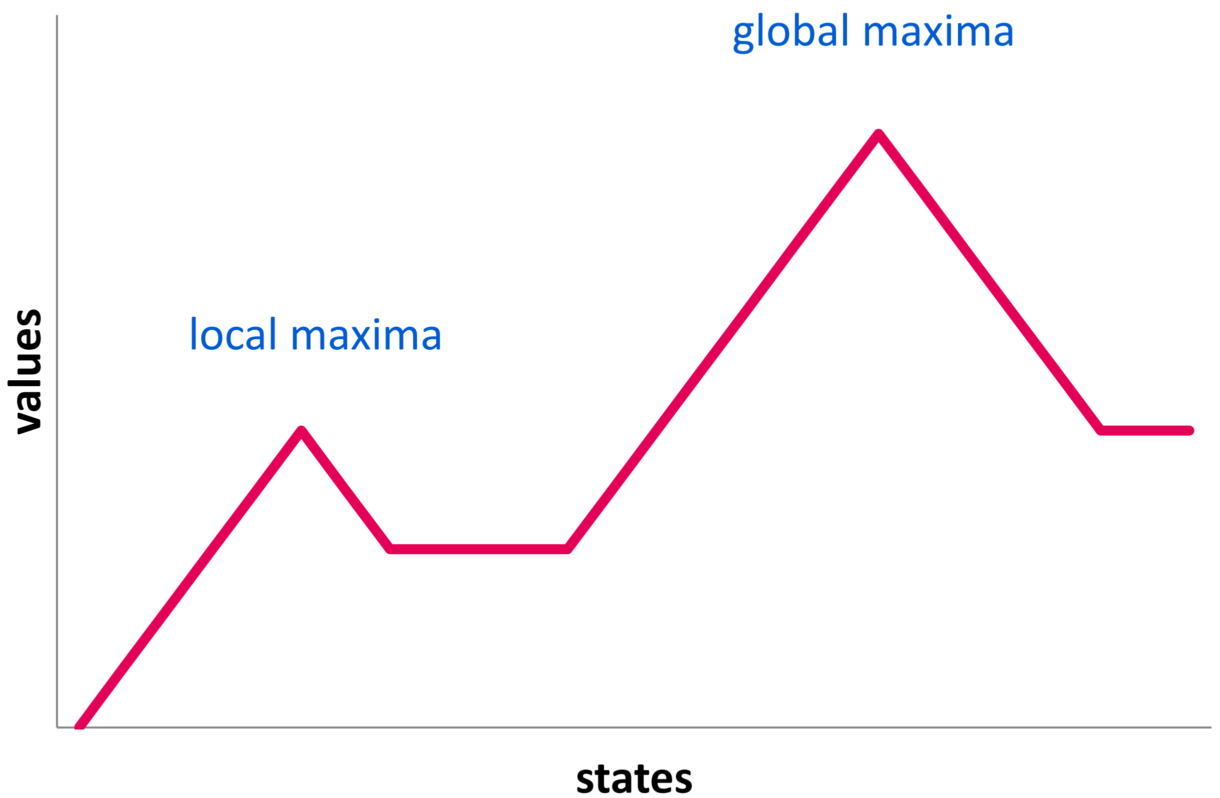
- “Like climbing Mount Everest in thick fog with amnesia”

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

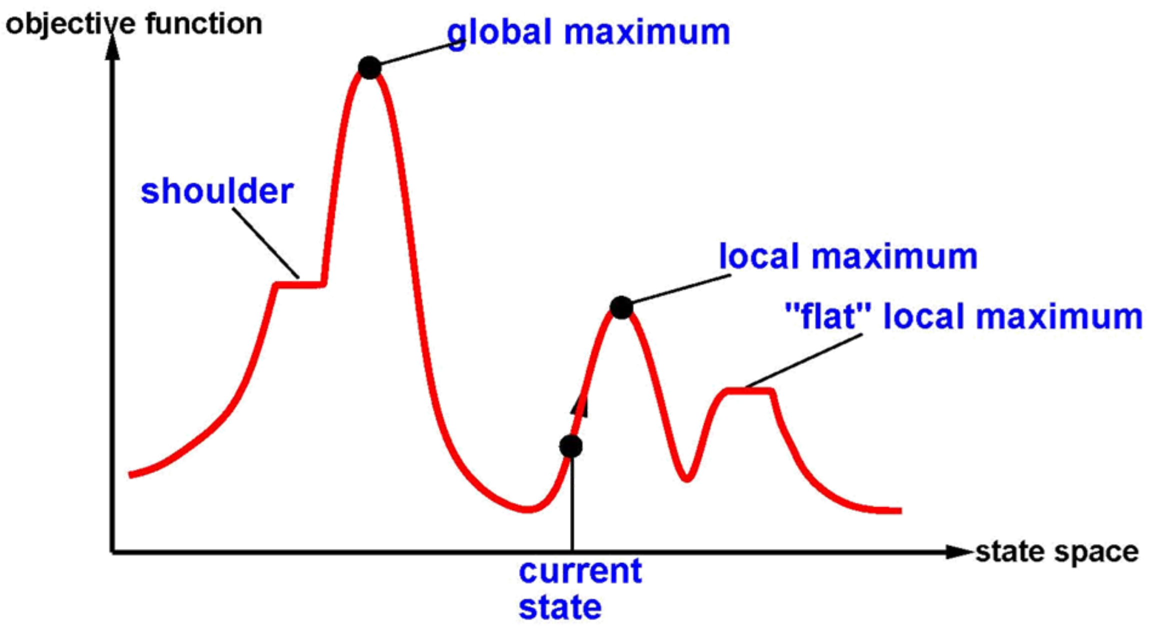
Hill Climbing Issues

- Also referred to as gradient descent
- Foothill problem / local maxima / local minima
- Can be solved with random walk or more steps
- Other problems: ridges, plateaus



Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😈escape from shoulders ☹loop on flat maxima

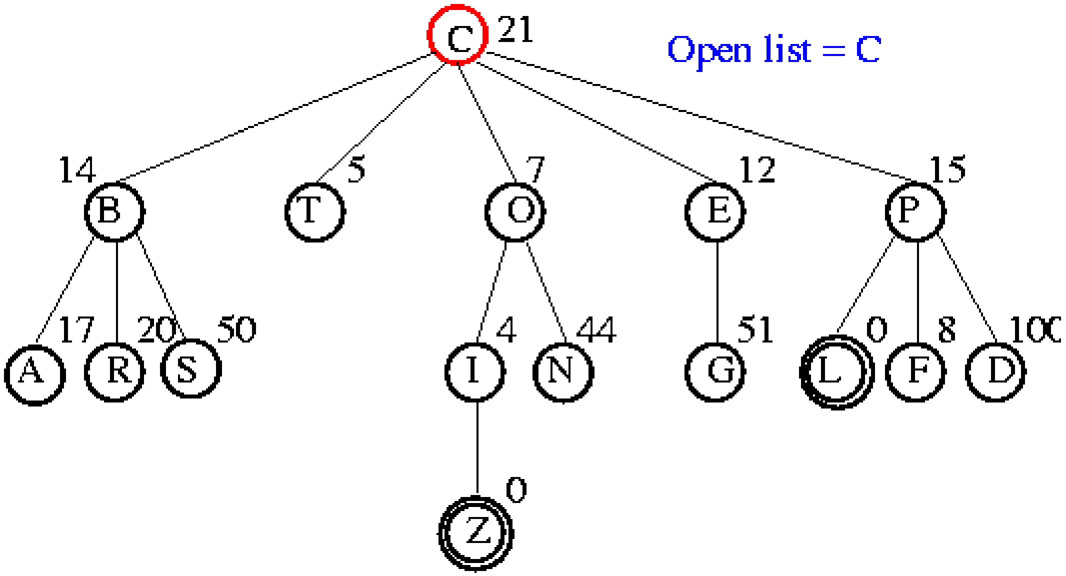
Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC
Complete	N	Y	Y	Y	N	N
Optimal	N	N	Y	N	N	N
Heuristic	N	N	N	N	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm
Space	bm	b^{d+1}	b^m	bd	b^m	b

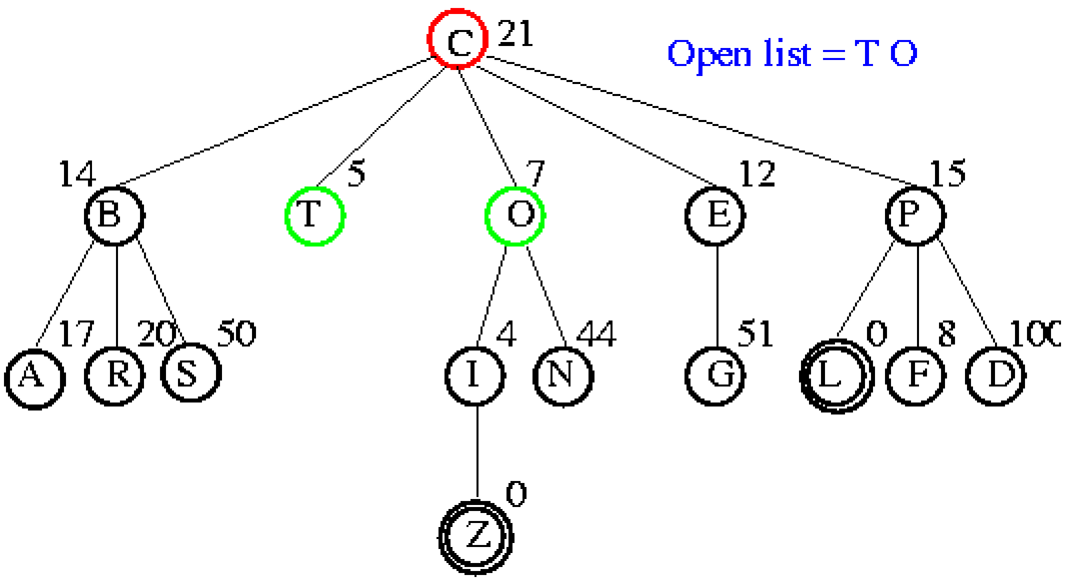
Beam Search

- QueueingFn is sort-by-h
  - Only keep best (lowest-h) n nodes on open list
- n is the “beam width”
  - n = 1, Hill climbing
  - n = infinity, Best first search

Example

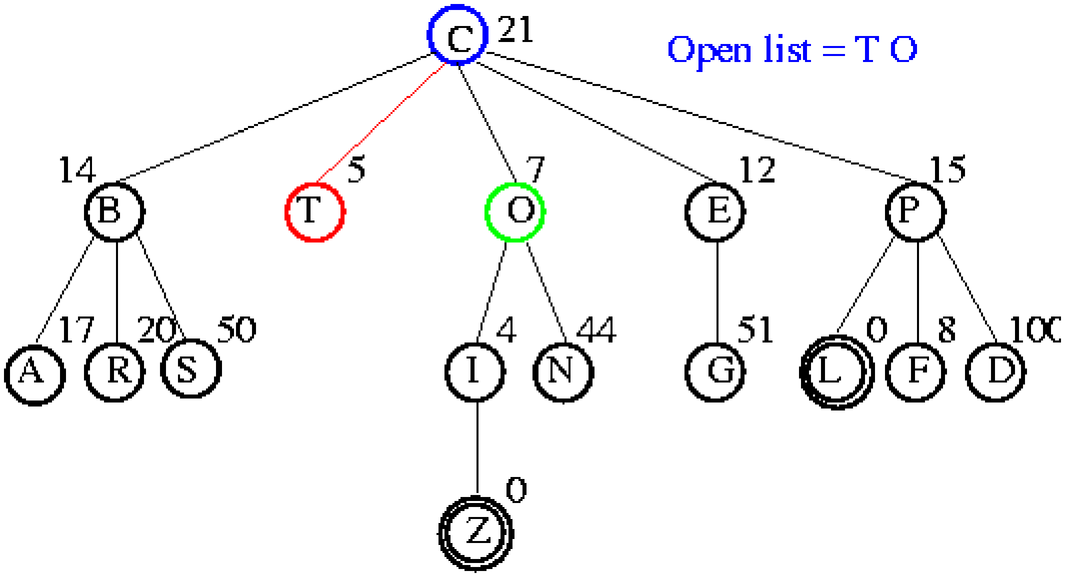

C
21
Open list = C
14
B
T
5
7
O
12
E
15
P
A
17
R
20
S
50
I
4
N
44
G
51
L
0
F
8
D
100
Z
0

Example

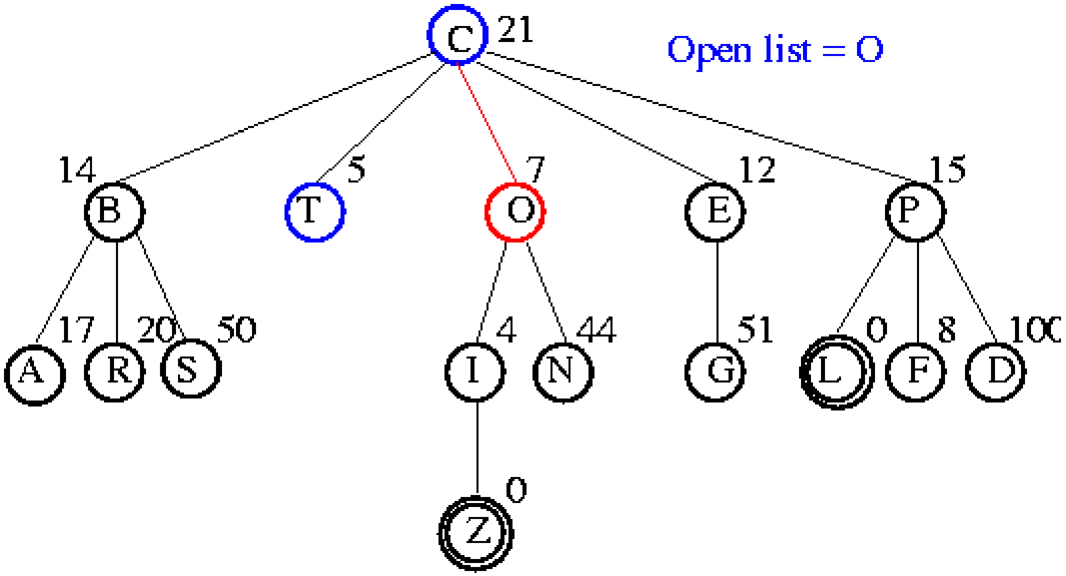


C
21
Open list = T O
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example

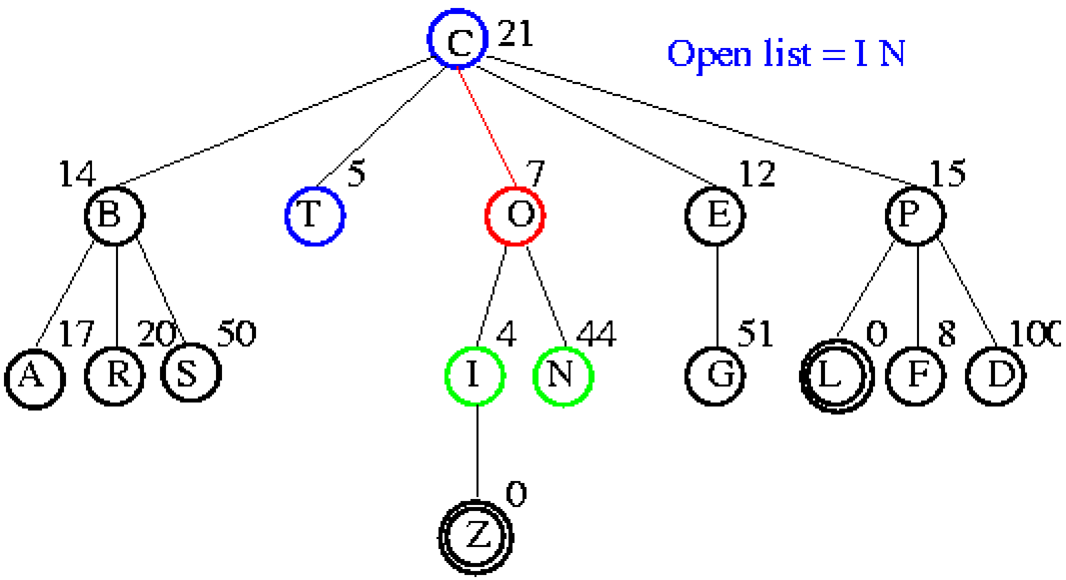

C 21
Open list = T O
14
B
T 5
O 7
E 12
P 15
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Example



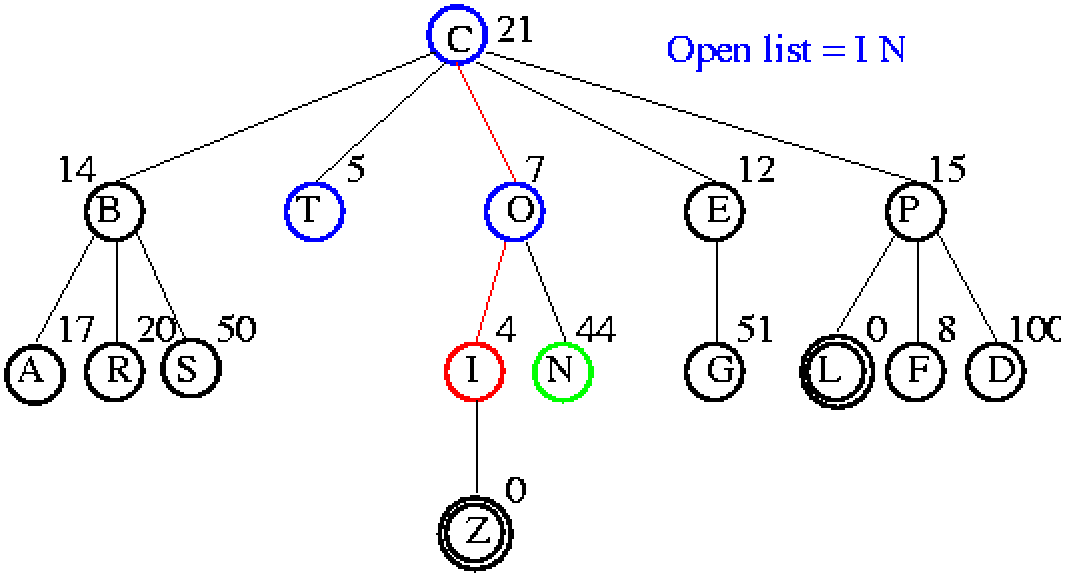
C
21
Open list = O
14
B
5
T
7
O
12
E
15
P
A
17
R
20
S
50
I
4
N
44
G
51
L
0
F
8
D
100
Z
0

Example

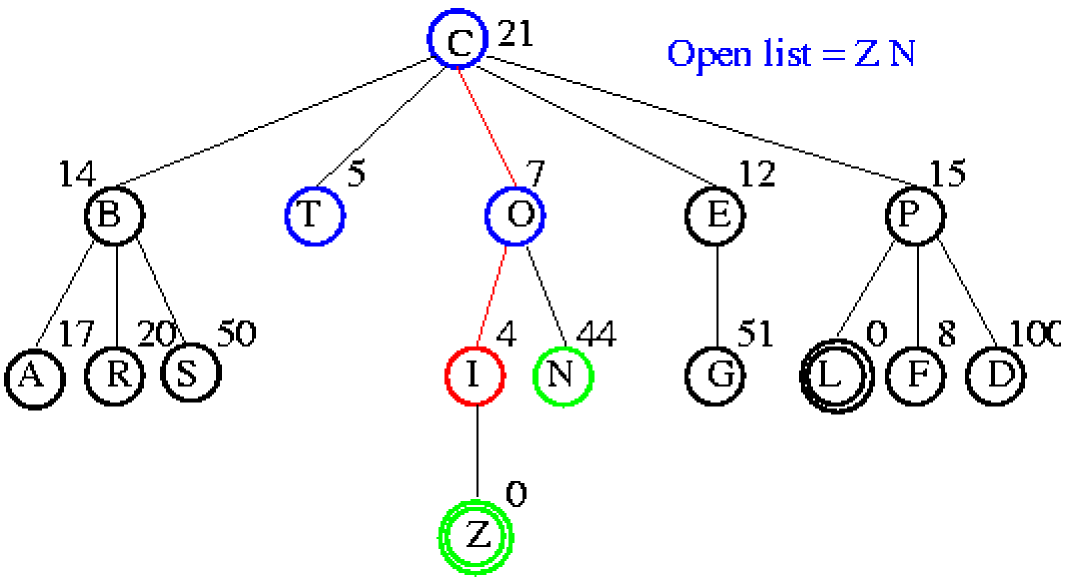


C
21
Open list = I N
14
B
5
T
7
O
12
E
15
P
A
17
R
20
S
50
I
4
N
44
G
51
L
0
F
8
D
100
Z
0

Example

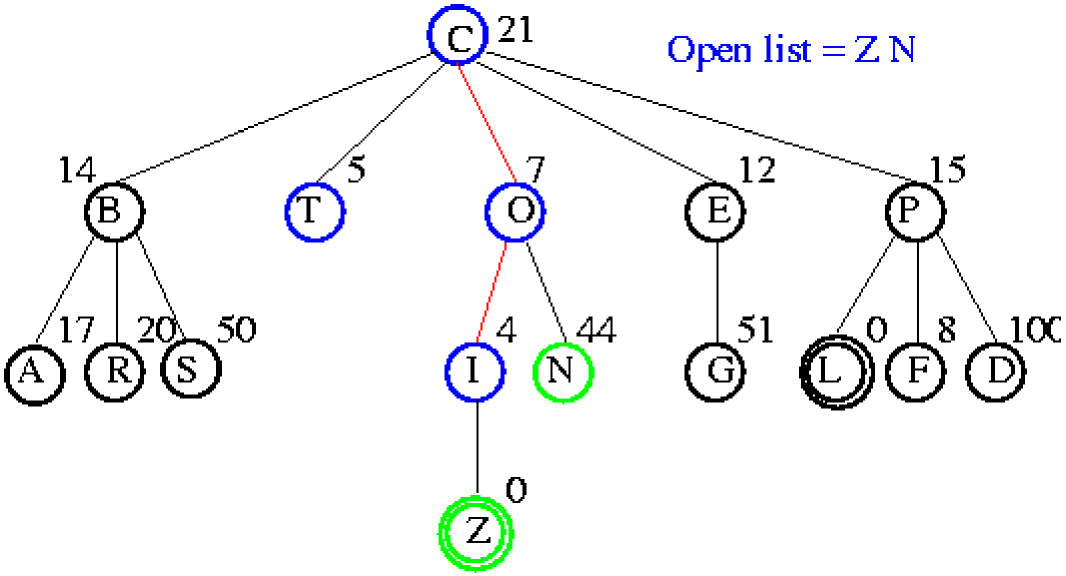

C
21
Open list = I N
14
B
5
T
7
O
12
E
15
P
17
A
R
20
S
50
4
I
44
N
G
51
L
0
F
8
D
100
Z
0

Example



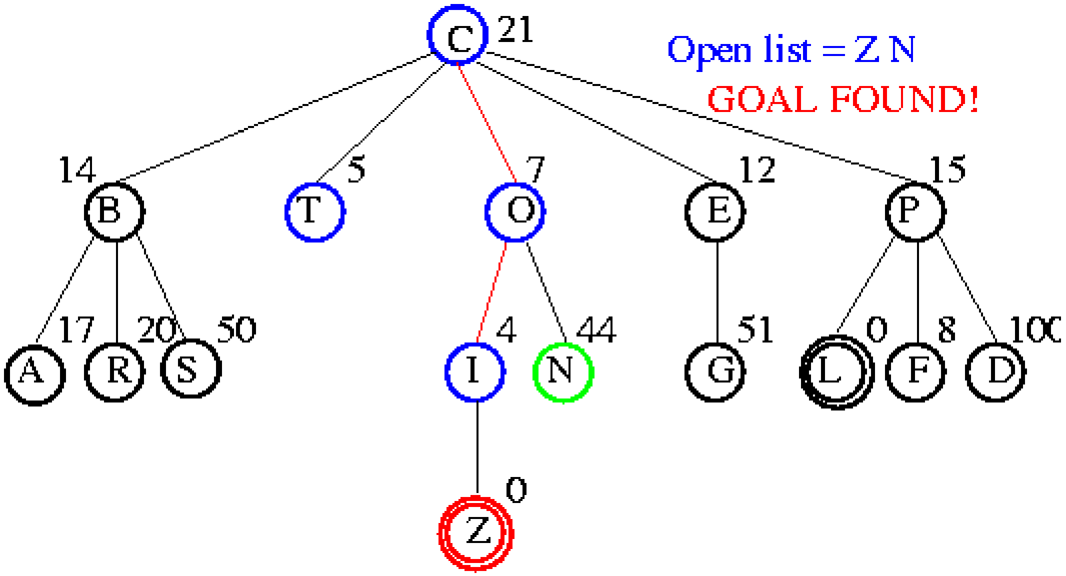
C
21
Open list = Z N
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
100
D
0
Z

Example



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

Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC	Beam
Complete	N	Y	Y	Y	N	N	N
Optimal	N	N	Y	N	N	N	N
Heuristic	N	N	N	N	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn

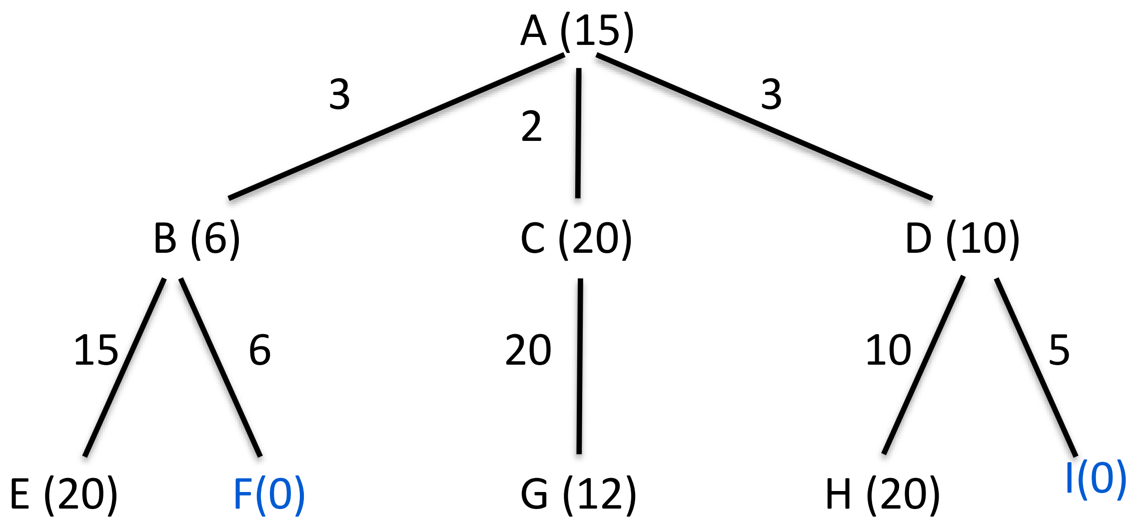
A*

- QueueingFn is sort-by-f
  - $f(n) = g(n) + h(n)$
- Note that UCS and Best-first both improve search
  - UCS keeps solution cost low
  - Best-first helps find solution quickly
- A* combines these approaches

Power of f

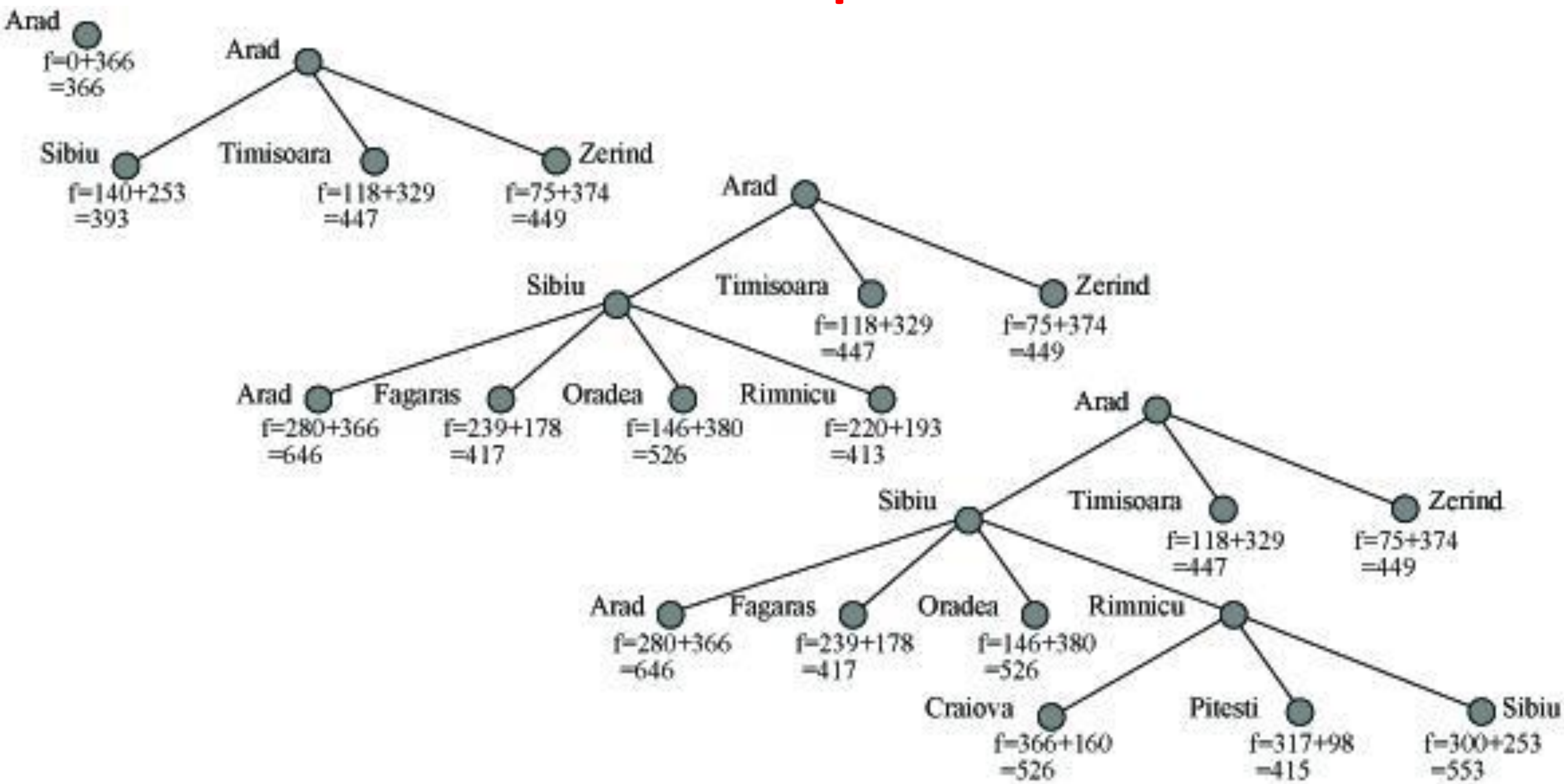
- If heuristic function is wrong it either
  - overestimates (guesses too high)
  - underestimates (guesses too low)
- Overestimating is worse than underestimating
- A* returns optimal solution if h(n) is admissible
  - heuristic function is admissible if never overestimates true cost to nearest goal
  - if search finds optimal solution using admissible heuristic, the search is admissible

Overestimating



- Solution cost:
  - ABF = 9
  - ADI = 8
- Open list:
  - A (15) B (9) F (9)
- Missed optimal solution

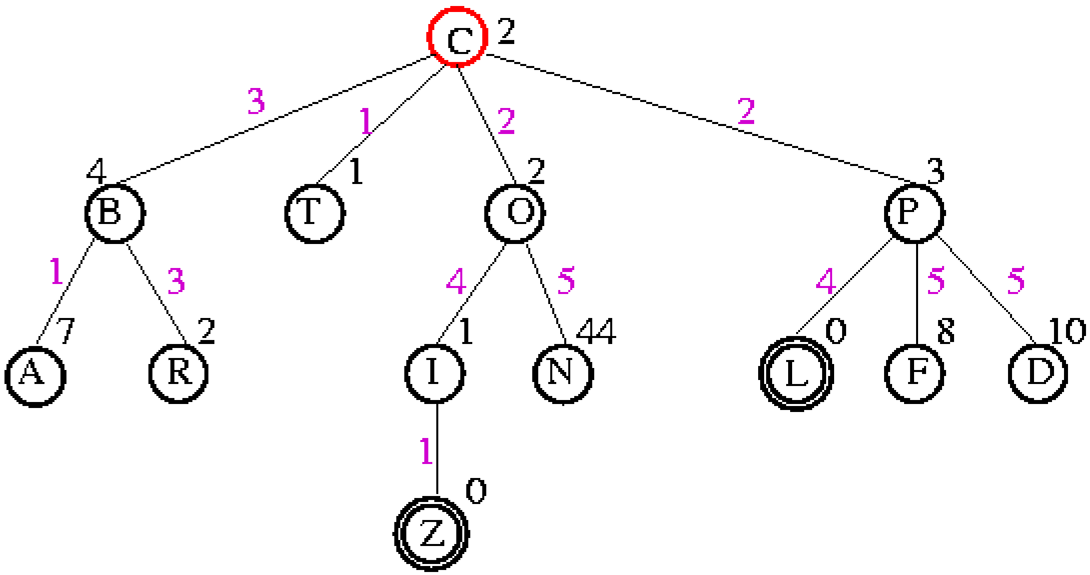
Example



[A* applied to 8 puzzle]

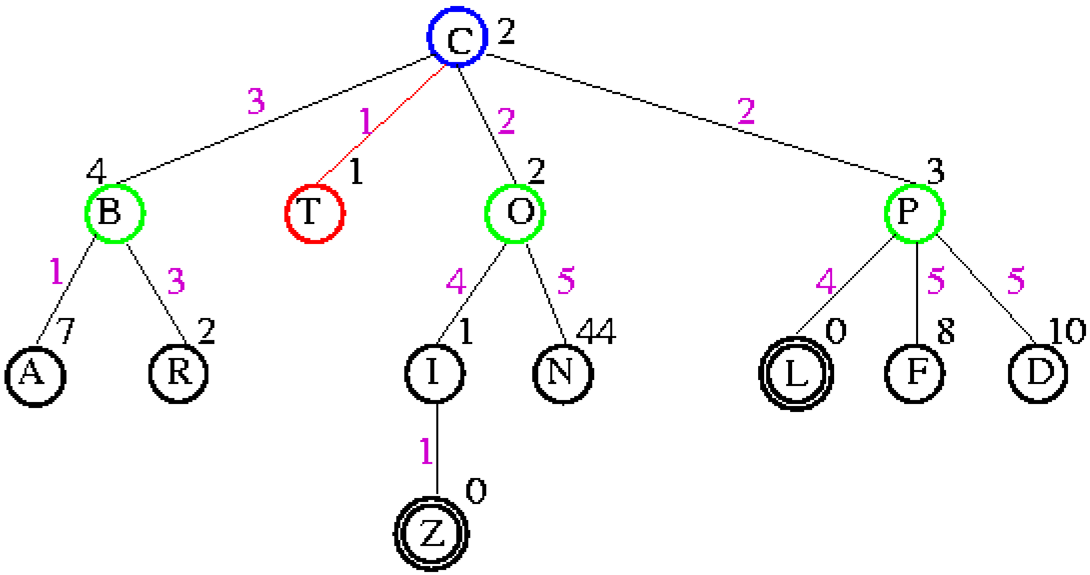
[A* search applet]

Example



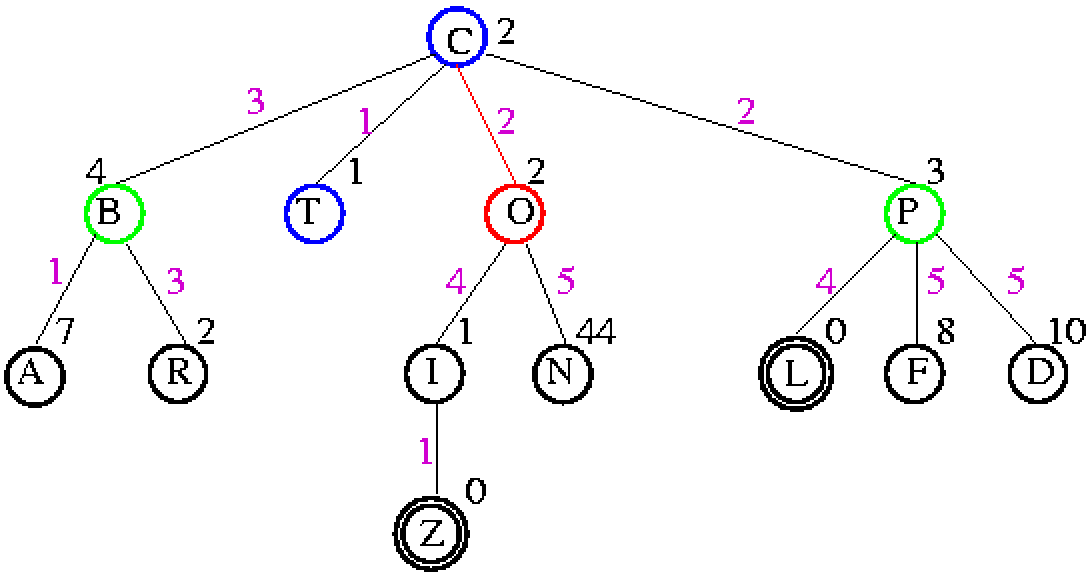
Open List = C (0+2=2)

Example



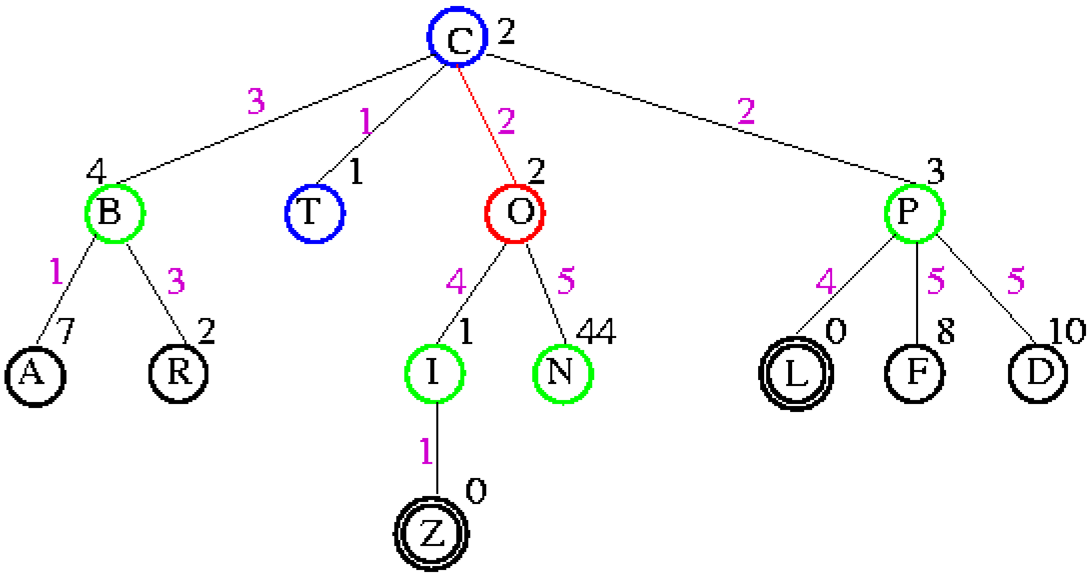
Open List = T (1+1=2), O (2+2=4), P (2+3=5), B(3+4=7)

Example



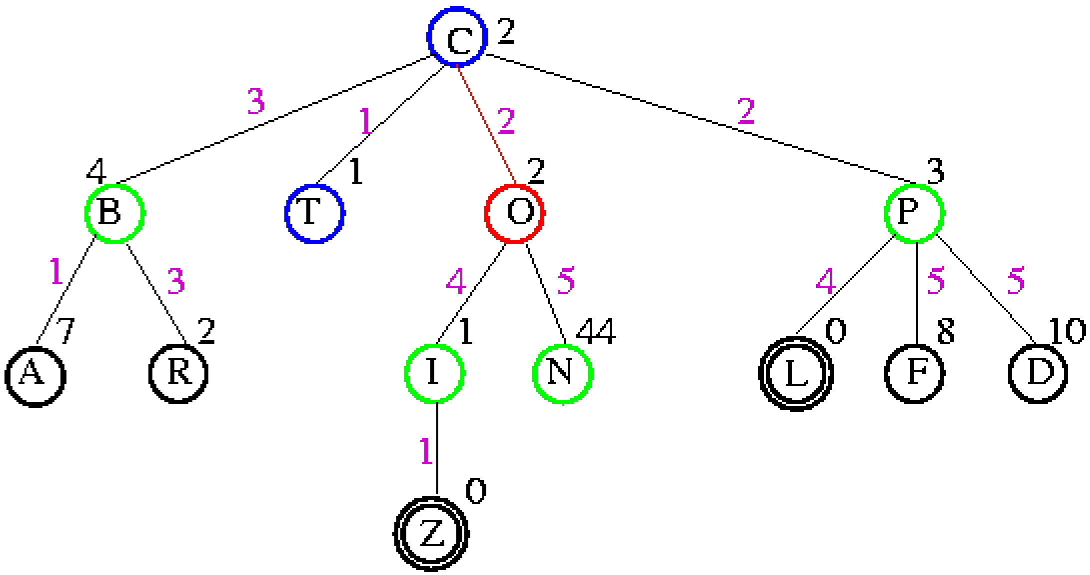
Open List = O (2+2=4), P (2+3=5), B(3+4=7)

Example



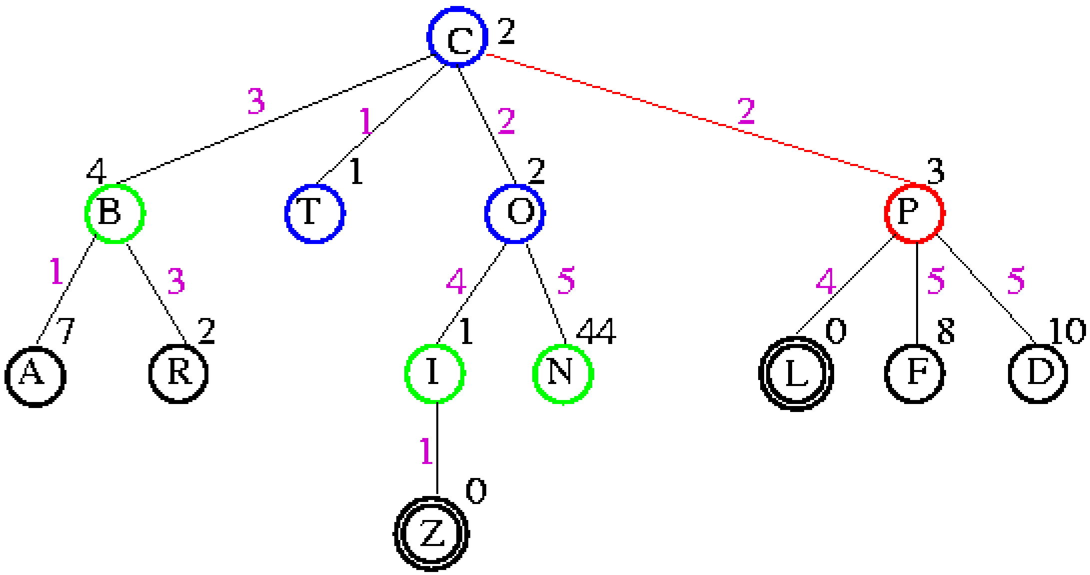
Open List = O (2+2=4), P (2+3=5), B(3+4=7)

Example



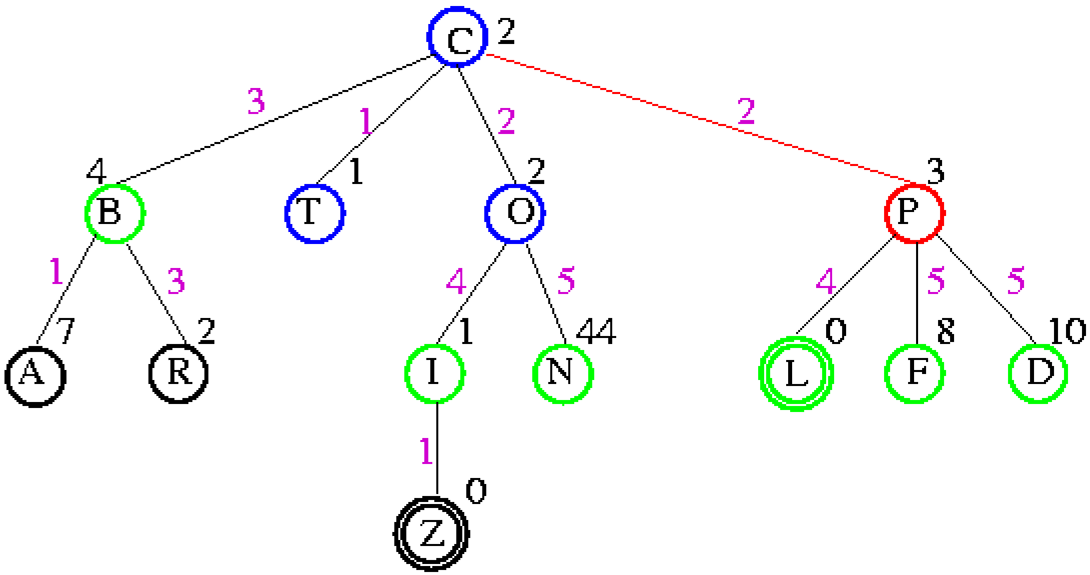
Open List = O (2+2=4), P (2+3=5), B(3+4=7)
I (6+1=7), N (7+44=51)

Example



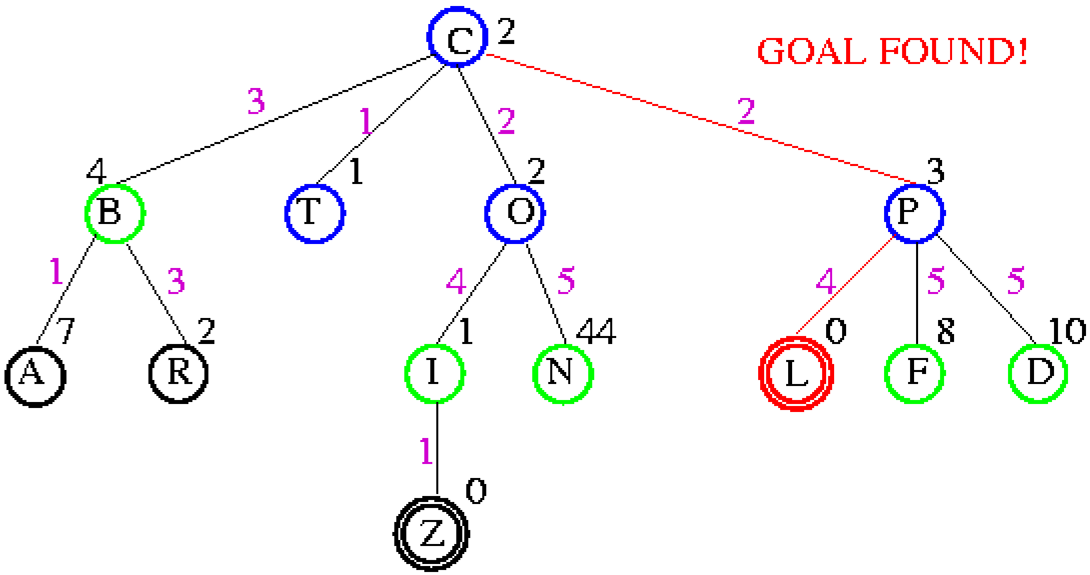
Open List = P (2+3=5), B(3+4=7)
I (6+1=7), N (7+44=51)

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)

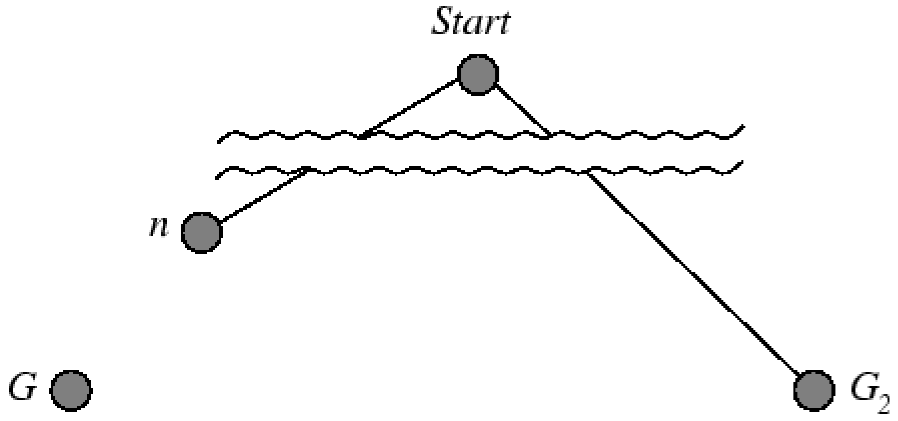
Example



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)

Optimality of A*

- Suppose a suboptimal goal G_2 is on the open list
- Let n be unexpanded node on smallest-cost path to optimal goal G_1



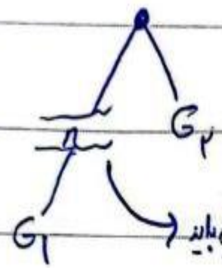
$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$>= g(G_1)$ since G_2 is suboptimal

$>= f(n)$ since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Admissibility



در صف Frontier و از این باید بگذریم تا به G برسیم.

برهان خلف: فرض می‌کنیم G_2 هدف باشد ولی بهینه نباشد.

$F(G_1) = h(G_1)$

$F(G_2) = h(G_2)$

if G_2 is subabtimal $\rightarrow h(G_2) > h(G)$

$F(G_2) > F(G)$

$F(n) = g(n) + h(n) \leq g(n) + h^*(n) \quad F(G) = h(G) < F(G_2)$

$F(n) < F(G_2)$ که این تناقض است چون در یک depth هستند.

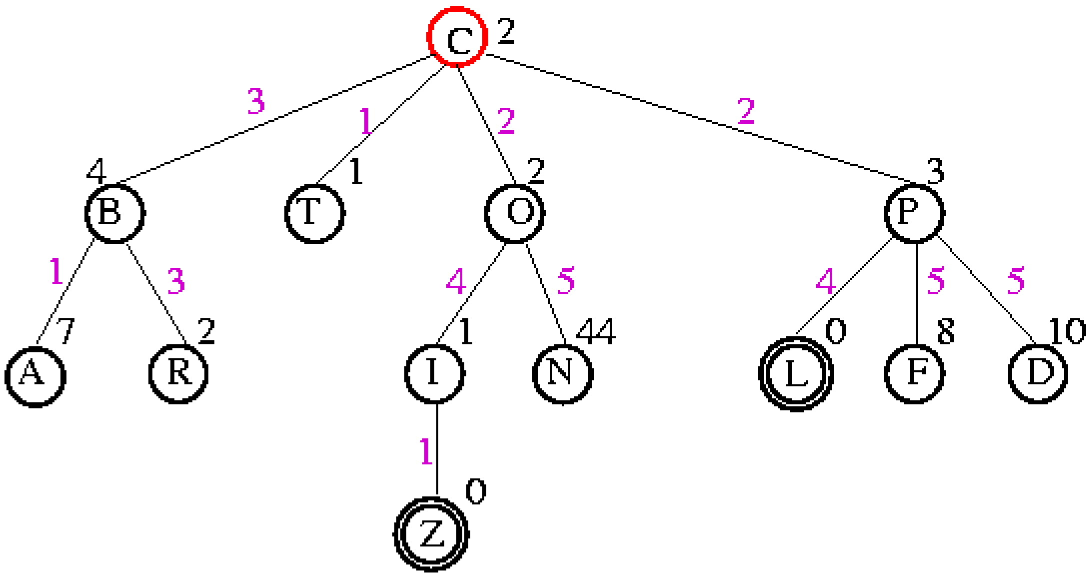
Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC	Beam	A*
Complete	N	Y	Y	Y	N	N	N	Y
Optimal	N	N	Y	N	N	N	N	Y
Heuristic	N	N	N	N	Y	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm	b^m
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn	b^m

IDA*

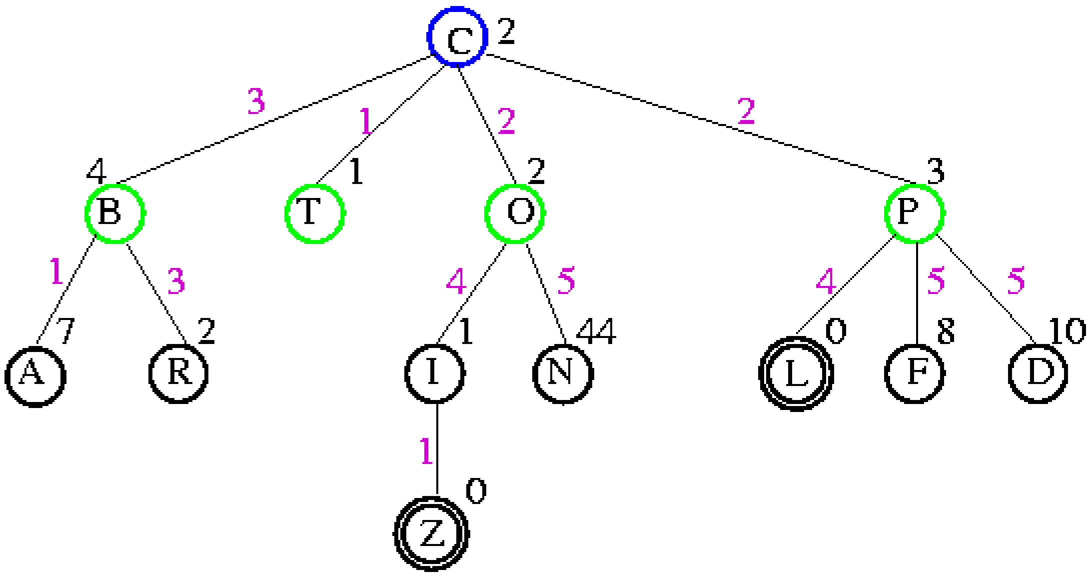
- Series of Depth-First Searches
- Like Iterative Deepening Search, except
  - Use A* cost threshold instead of depth threshold
  - Ensures optimal solution
- QueuingFn enqueues at front if f(child) <= threshold
- Threshold
  - h(root) first iteration
  - Subsequent iterations
    - f(min_child)
    - min_child is the cut off child with the minimum f value
  - Increase always includes at least one new node
  - Makes sure search never looks beyond optimal cost solution

Example



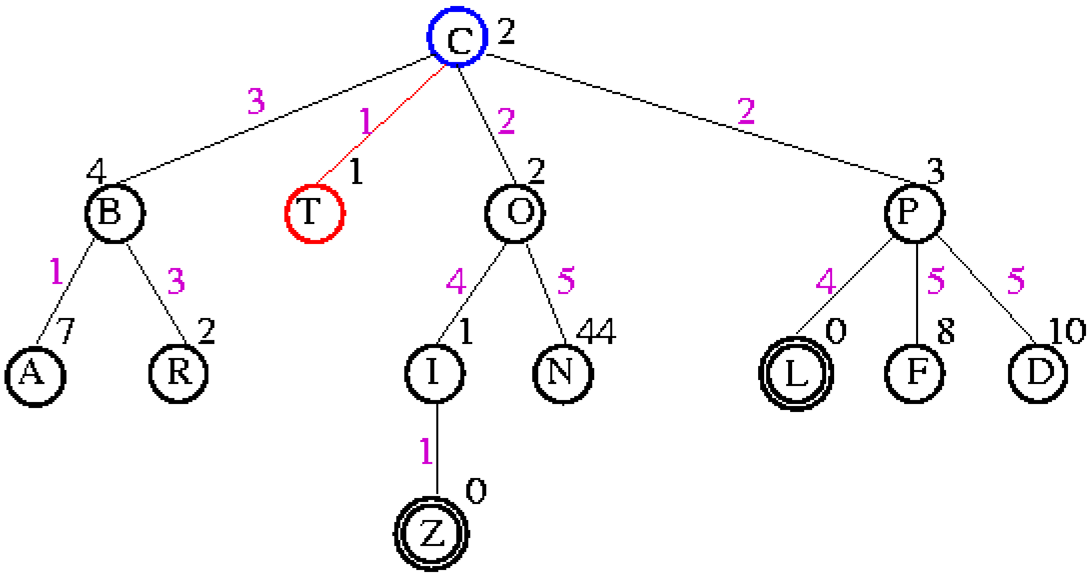
limit = f(C) = 2

Example



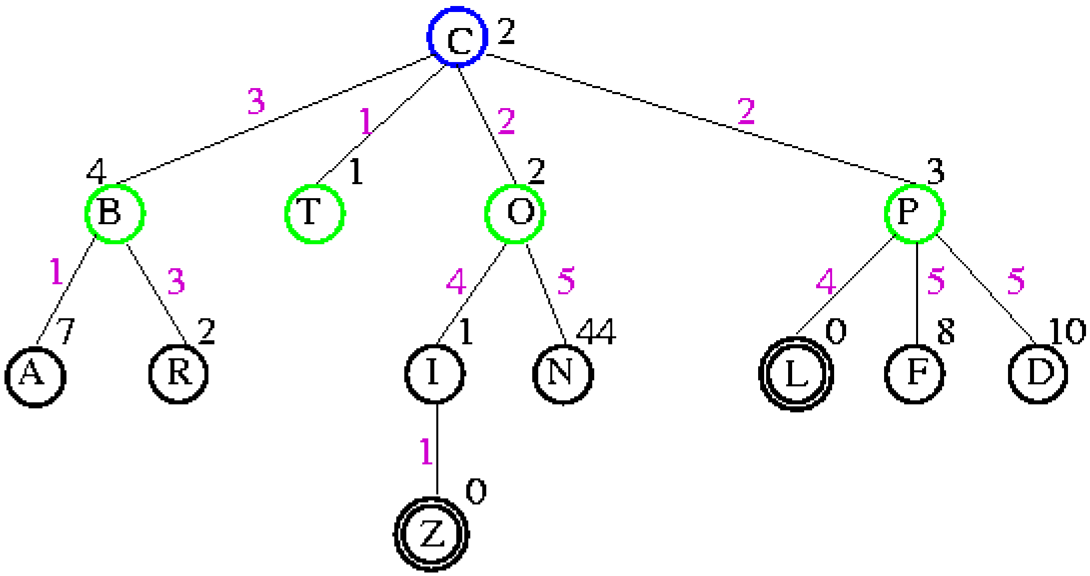
limit = f(C) = 2

Example



limit = f(C) = 2

Example

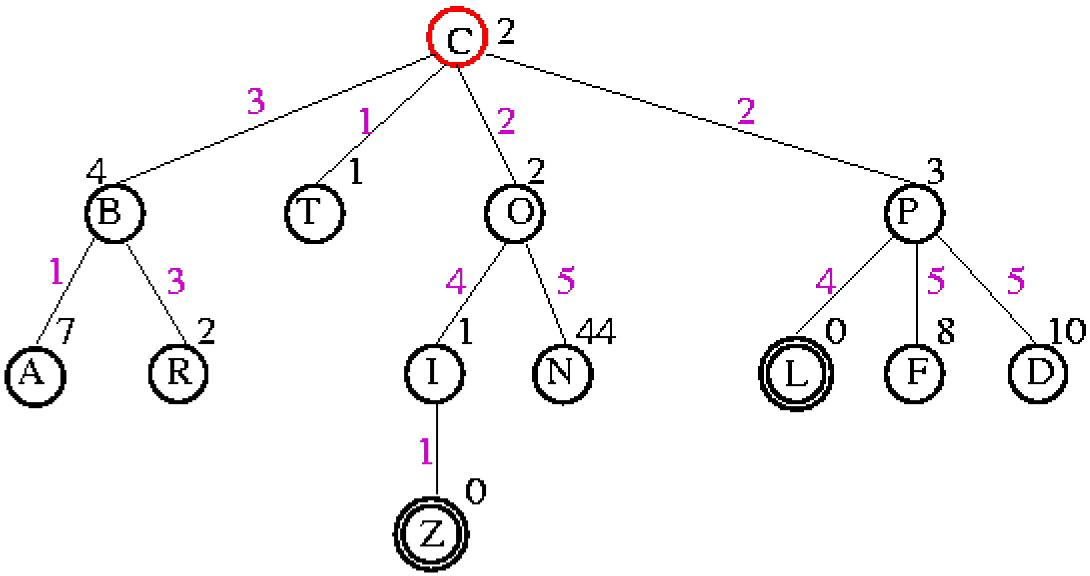


limit = f(C) = 2

Nodes on frontier: B (3+4=7), O(2+2=4), P(2+3=5)

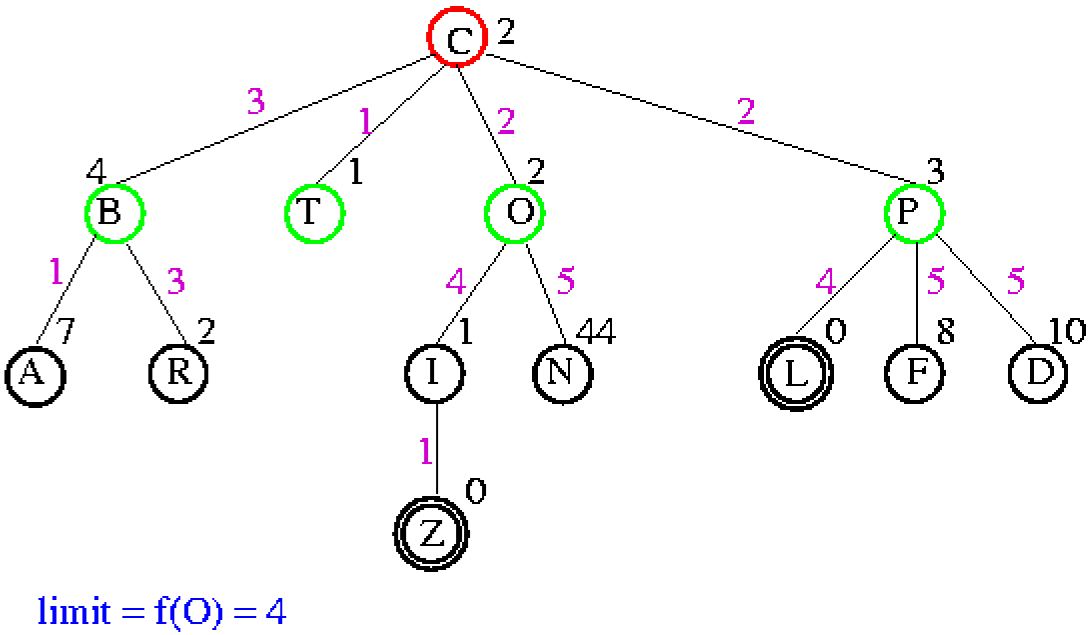
New limit = f(O) = 4

Example

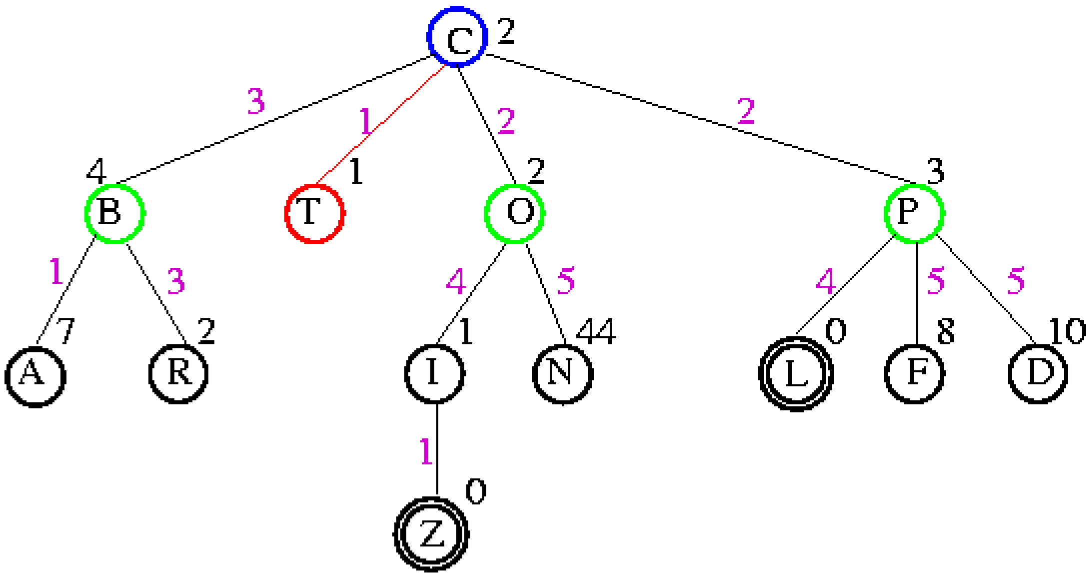


limit = f(O) = 4

Example

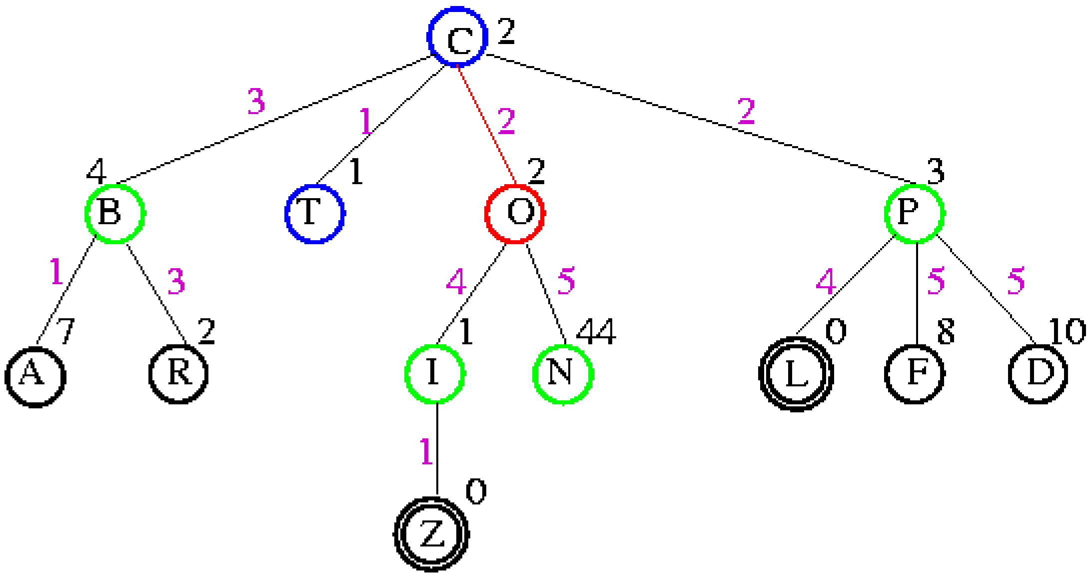


Example



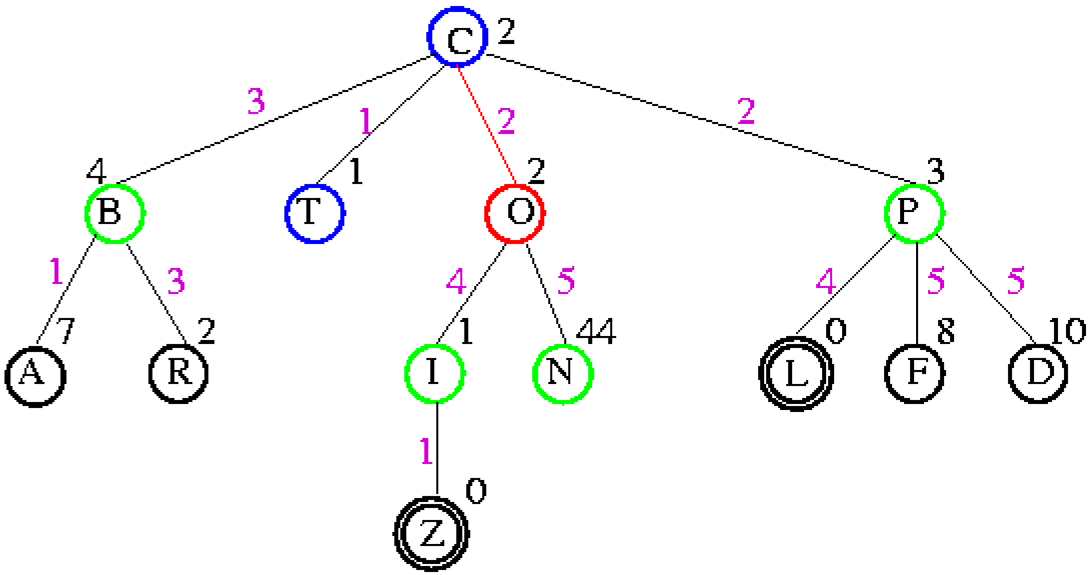
limit = f(O) = 4

Example



limit = f(O) = 4

Example

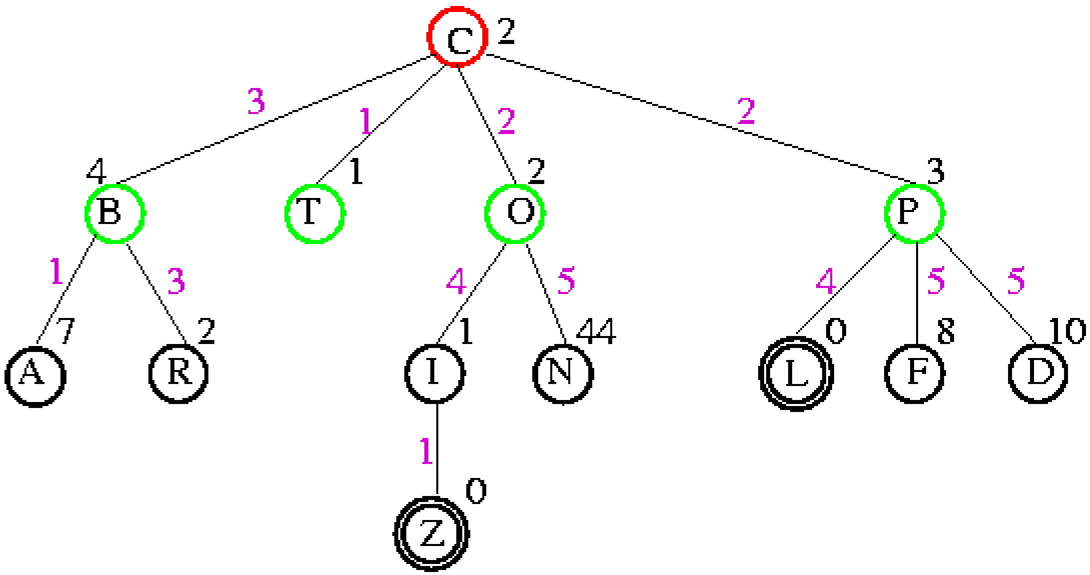


limit = f(O) = 4

Nodes on frontier: B (3+4=7), P (2+3=5)
I (6+1=7), N (7+44=51)

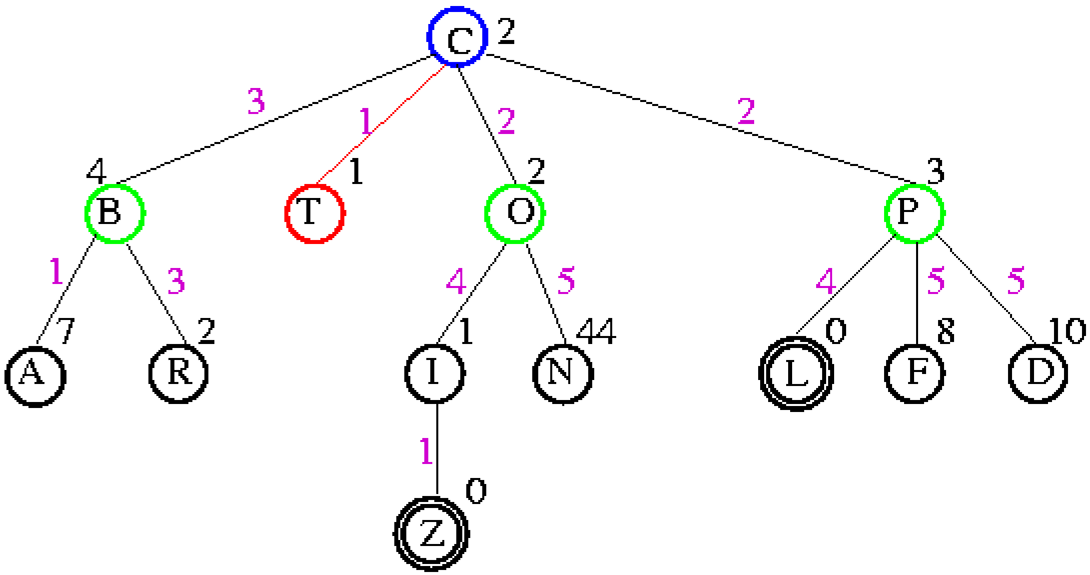
New limit = f(P) = 5

Example



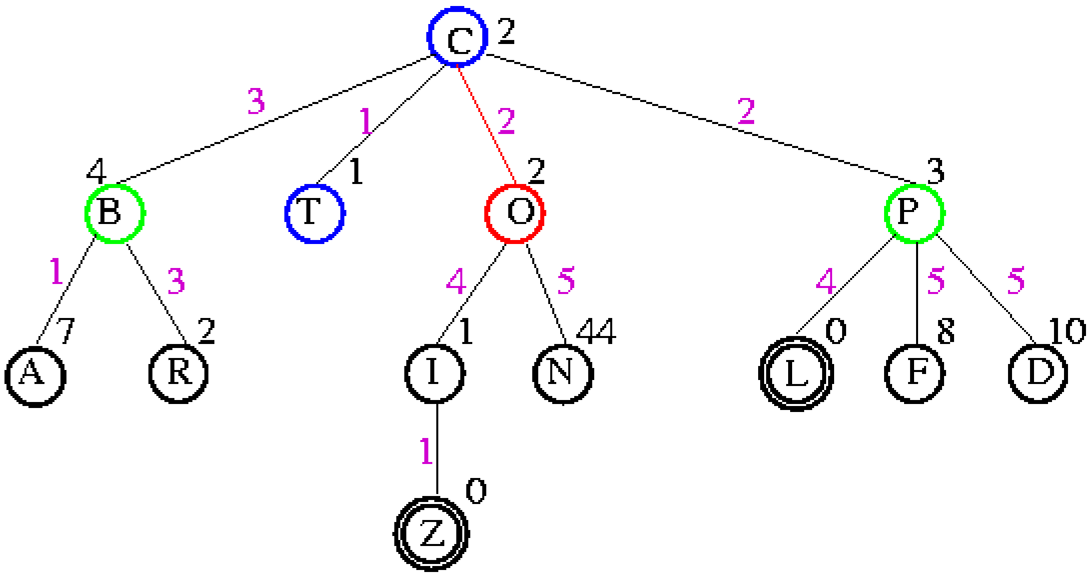
limit = f(P) = 5

Example



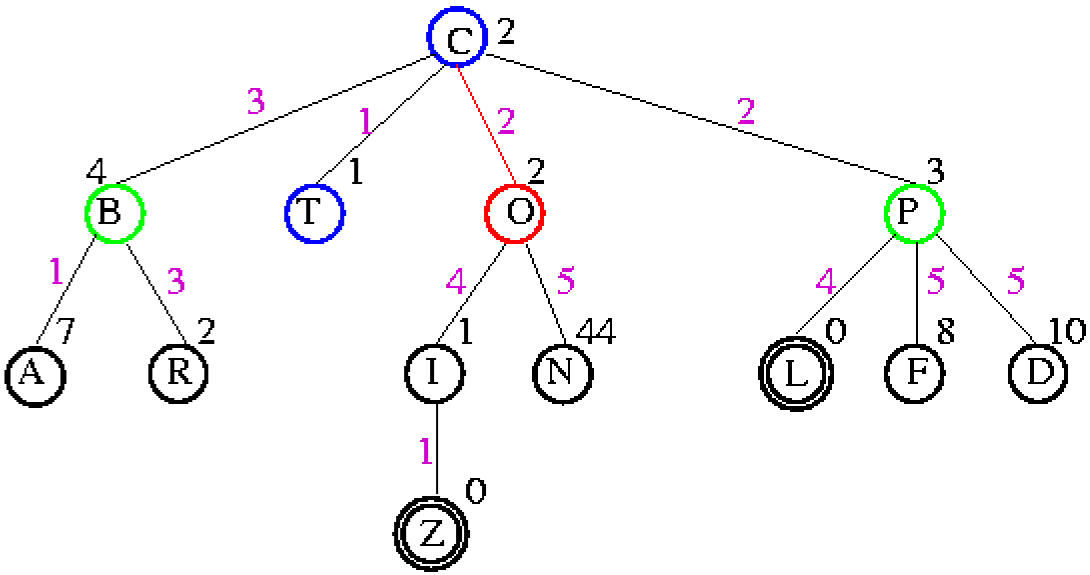
limit = f(P) = 5

Example



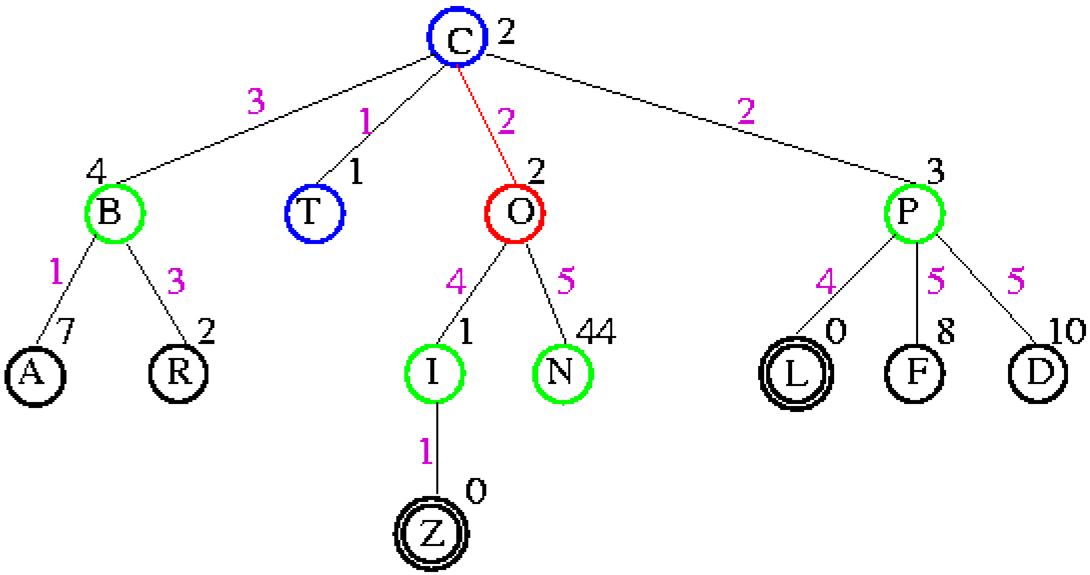
limit = f(P) = 5

Example



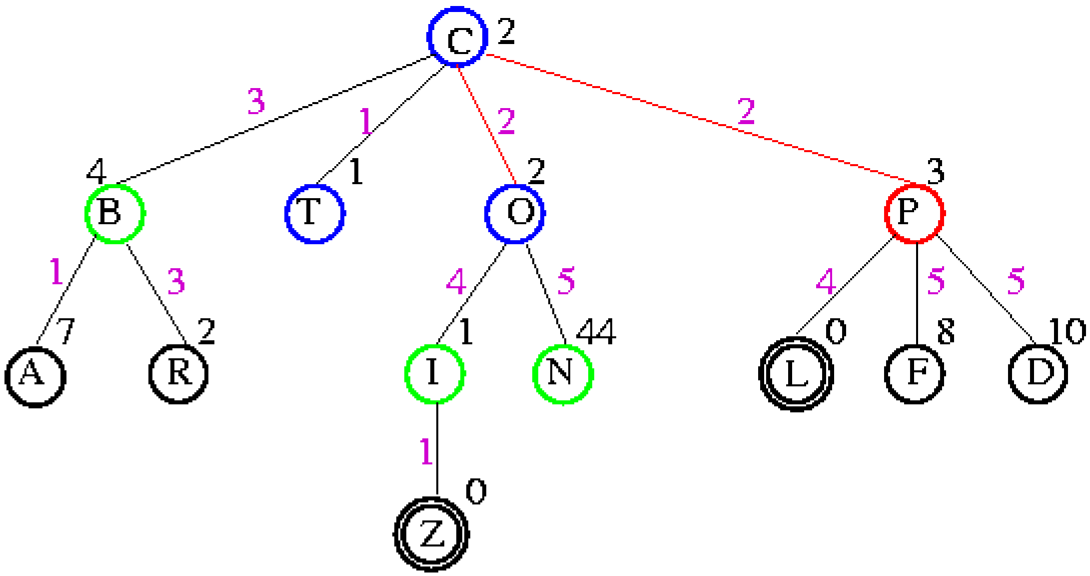
limit = f(P) = 5

Example



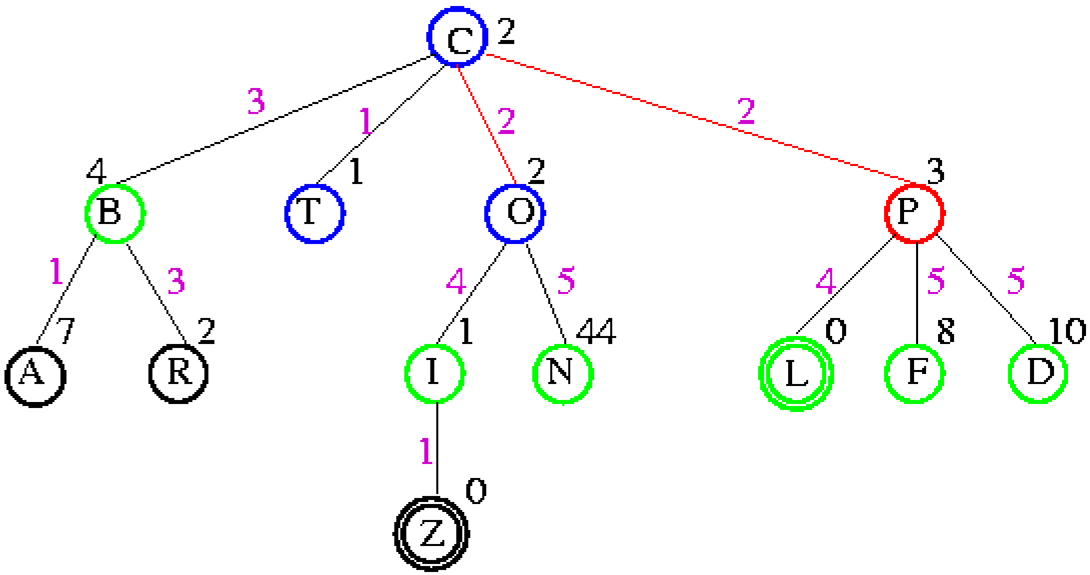
limit = f(P) = 5

Example



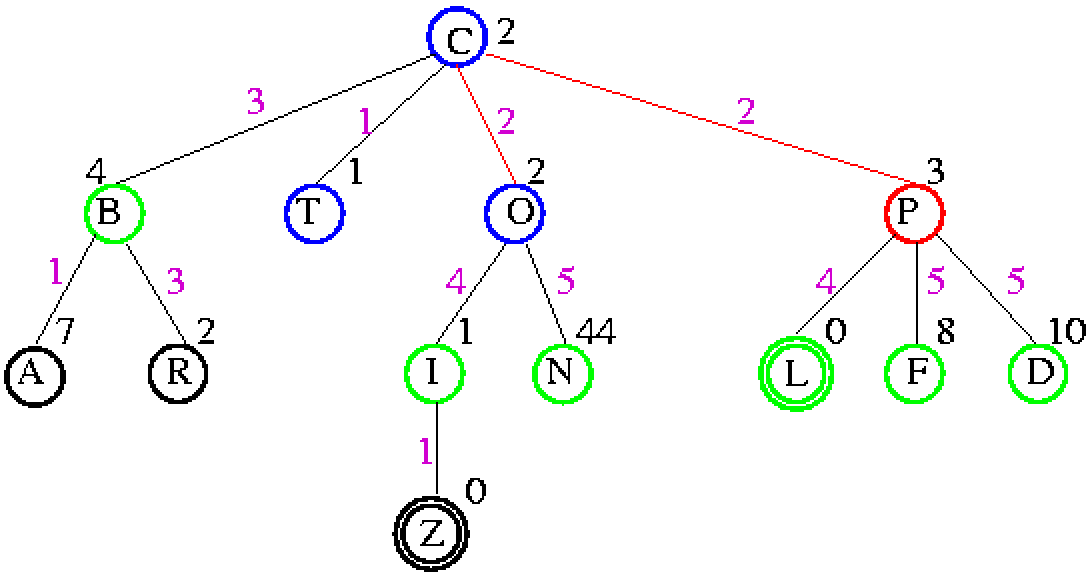
limit = f(P) = 5

Example



limit = f(P) = 5

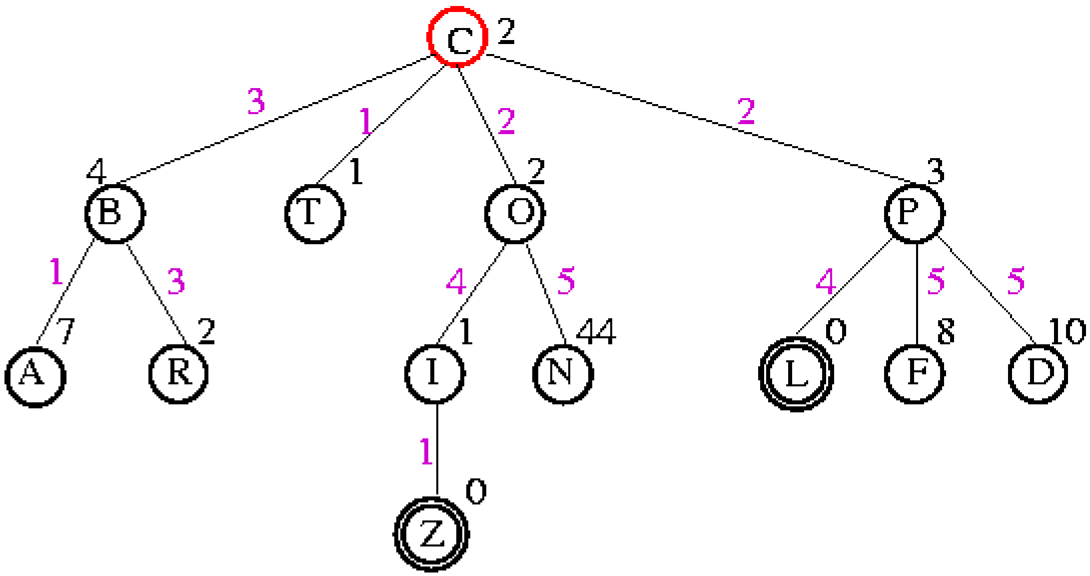
Example



limit = f(L) = 6

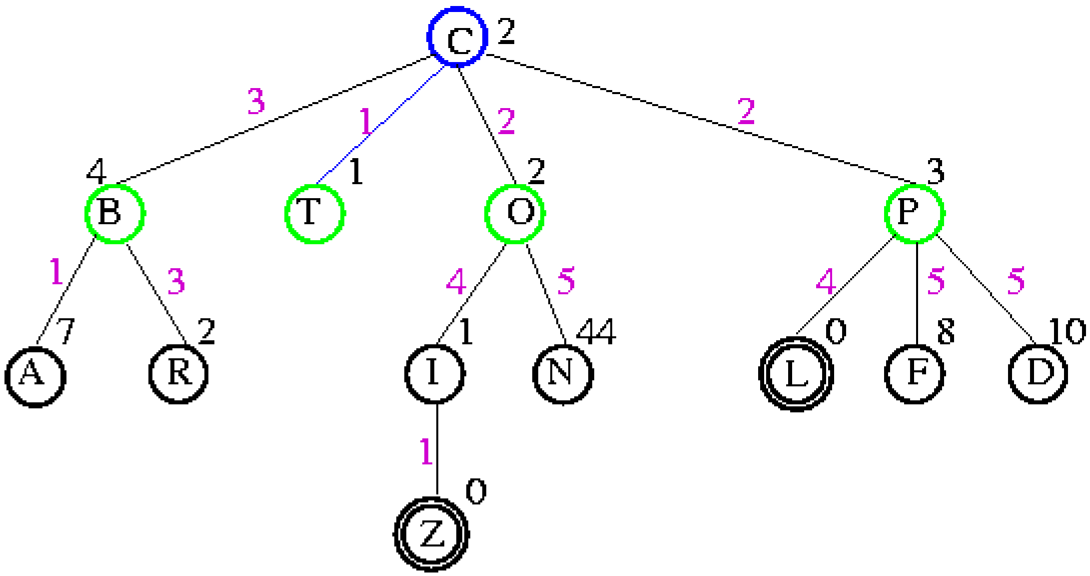
Nodes on frontier: B (3+4=7), I (6+1=7), N (7+44=51)
L (6+0=6), F (7+8=15), D (7+10=17)

Example



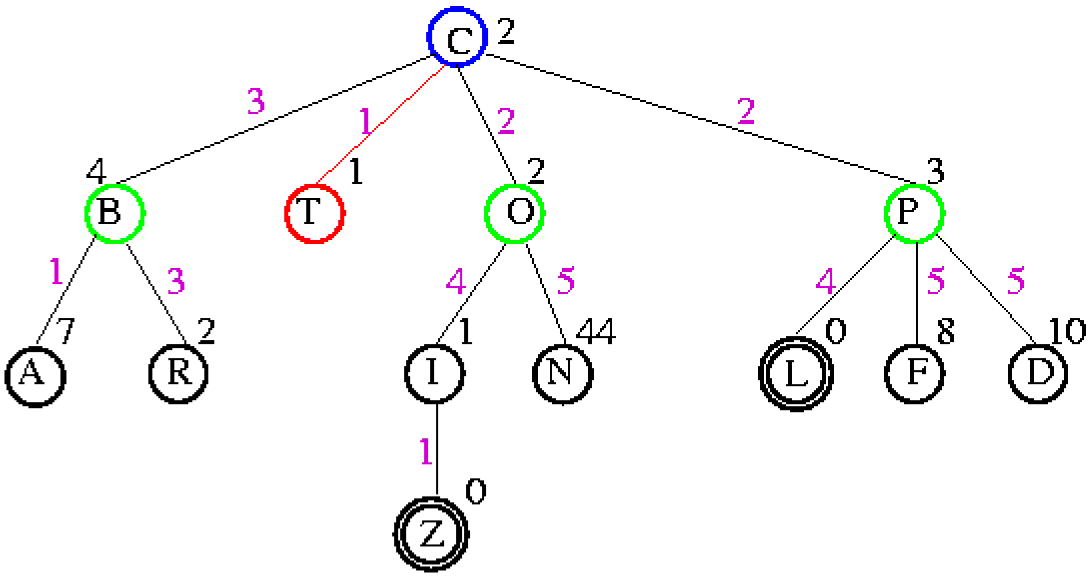
limit = f(L) = 6

Example



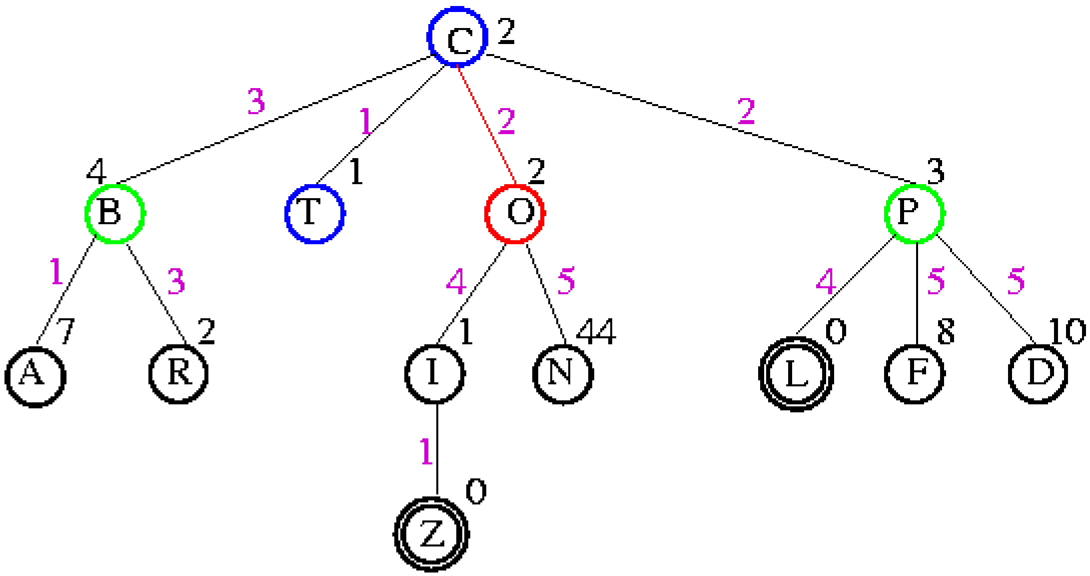
limit = f(L) = 6

Example



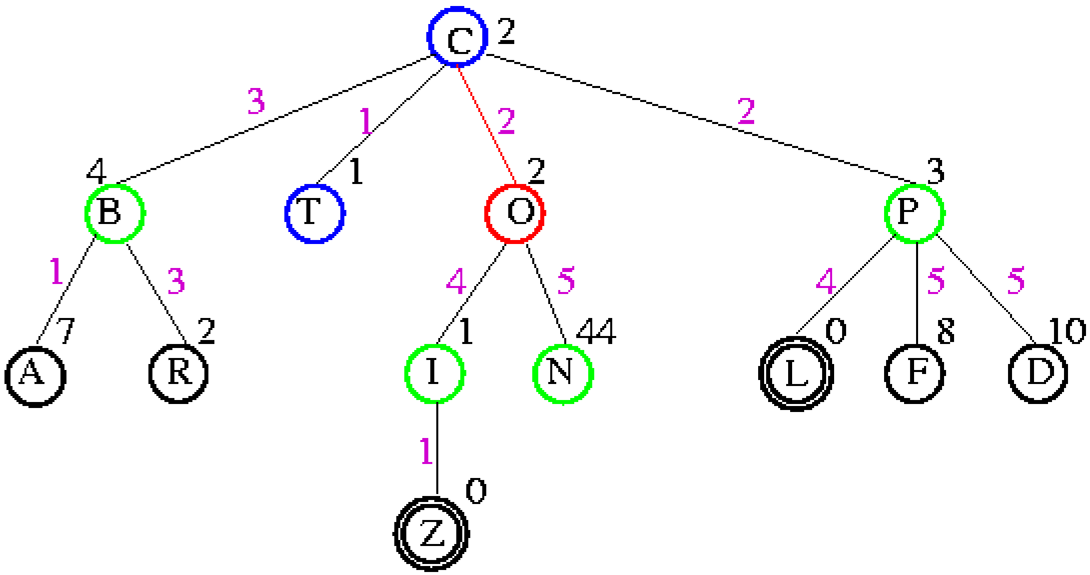
limit = f(L) = 6

Example



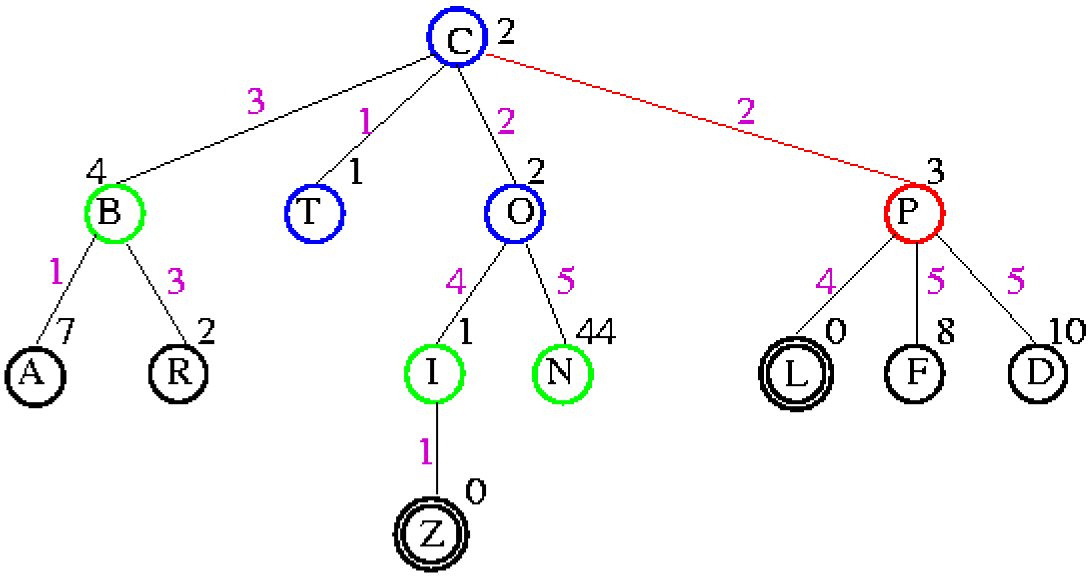
limit = f(L) = 6

Example



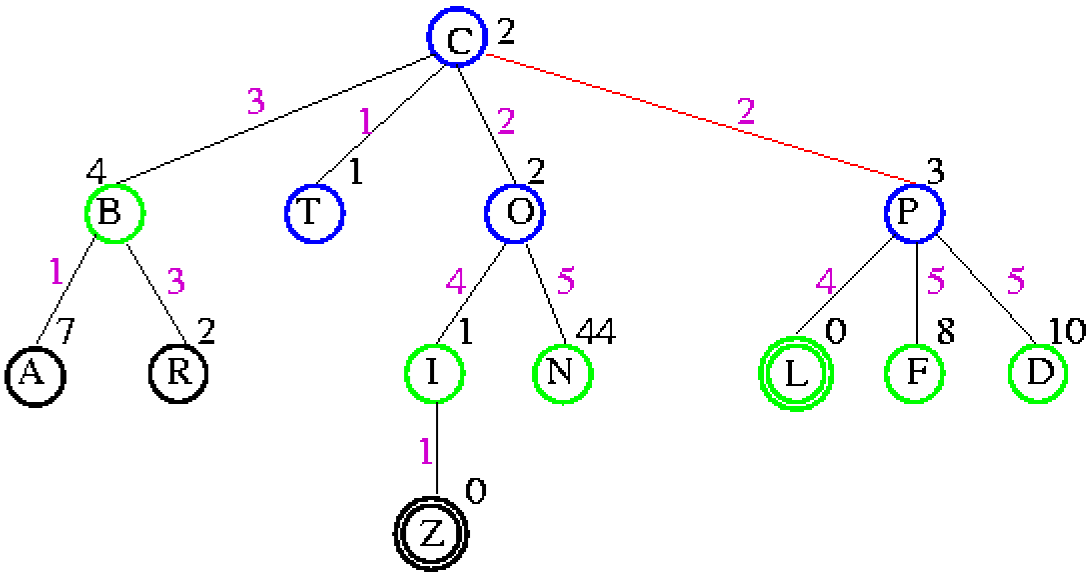
limit = f(L) = 6

Example



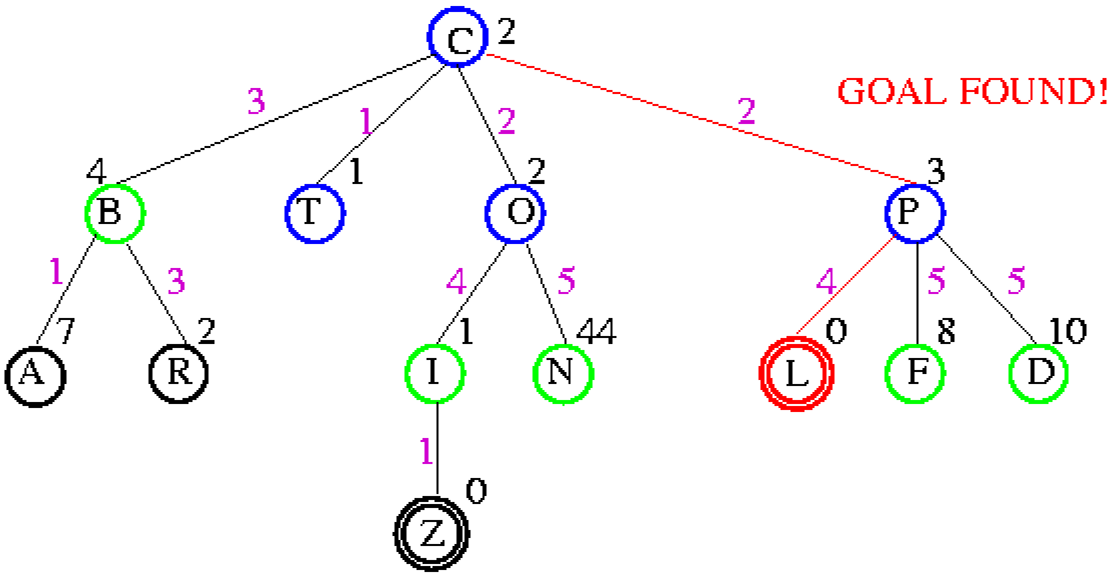
limit = f(L) = 6

Example



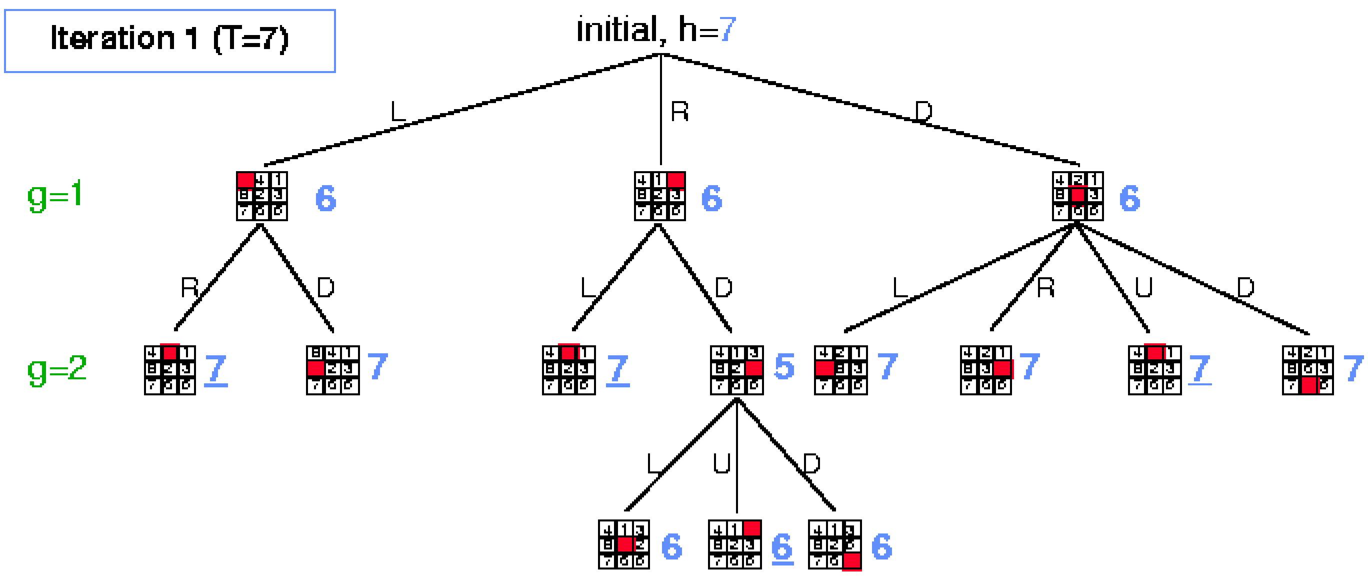
limit = f(L) = 6

Example



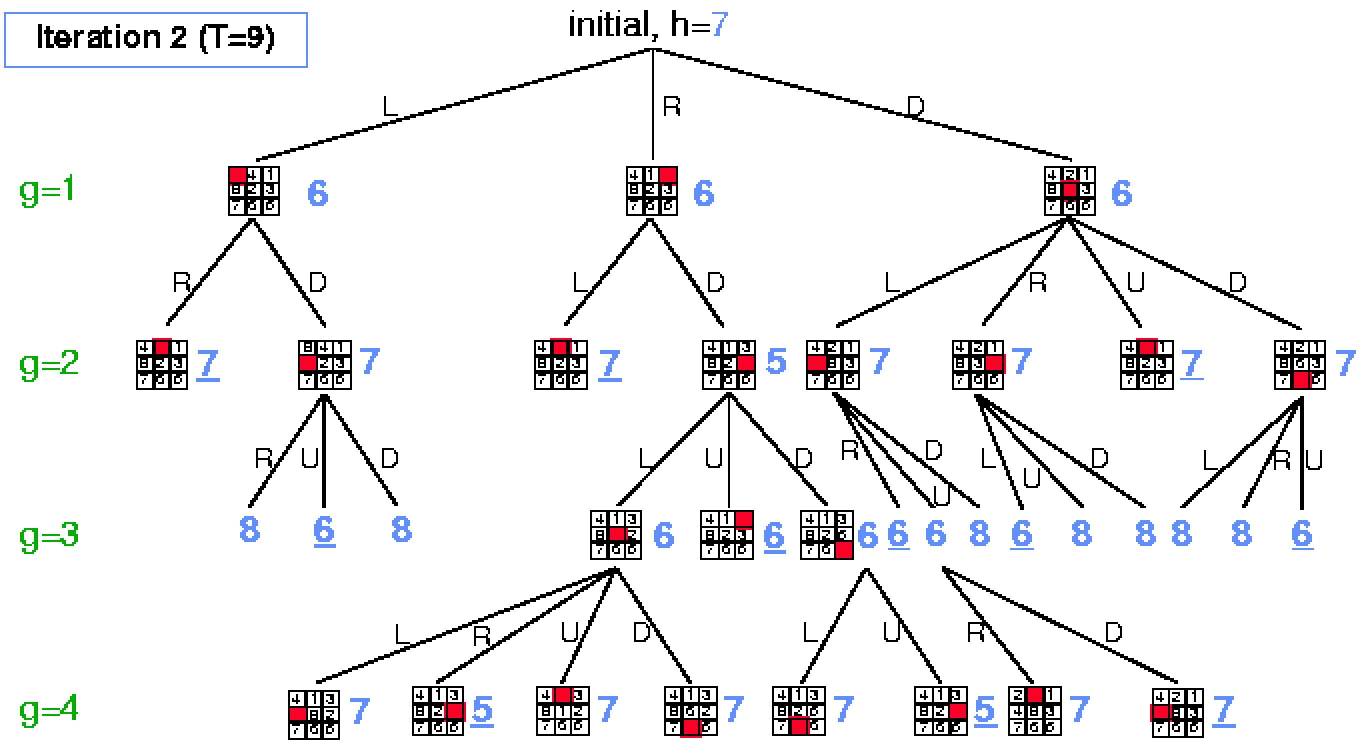
limit = f(L) = 6

Example

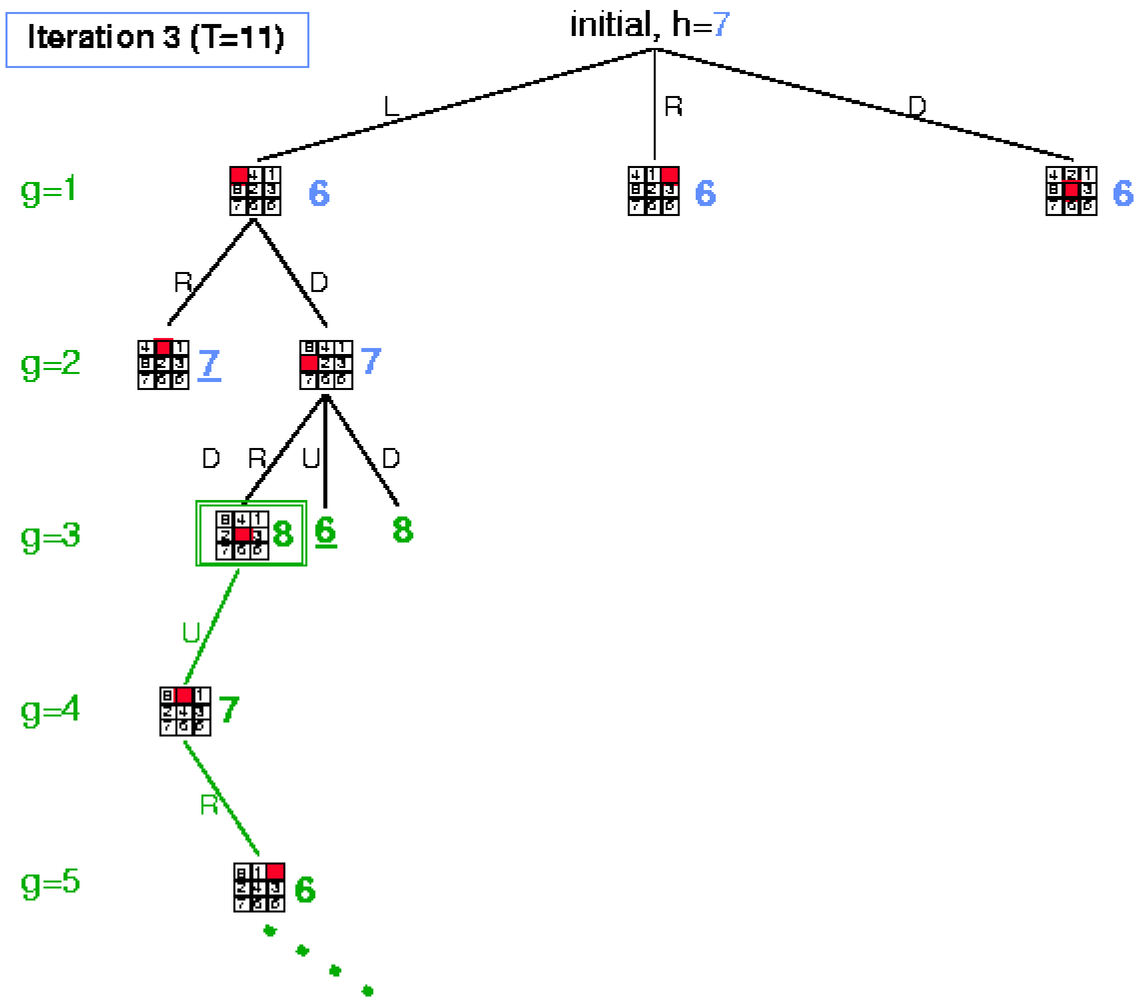


Iteration 1 (T=7)
initial, h=7
L
R
D
g=1
6
6
6
R
D
L
D
L
R
U
D
g=2
7
7
7
5
7
7
7
7
L
U
D
6
6
6

Example



Example


Iteration 3 (T=11)
initial, h=7
L
R
D
g=1
6
6
6
R
D
g=2
7
7
D
R
U
D
g=3
8
6
8
U
g=4
7
R
g=5
6

Analysis

- Some redundant search
  - Small amount compared to work done on last iteration
- Dangerous if continuous-valued h(n) values or if values very close
  - If threshold = 21.1 and value is 21.2, probably only include 1 new node each iteration
- Time complexity is $O(b^m)$
- Space complexity is $O(m)$

Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC	Beam	A*	IDA*
Complete	N	Y	Y	Y	N	N	N	Y	Y
Optimal	N	N	Y	N	N	N	N	Y	Y
Heuristic	N	N	N	N	Y	Y	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm	b^m	b^m
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn	b^m	bm

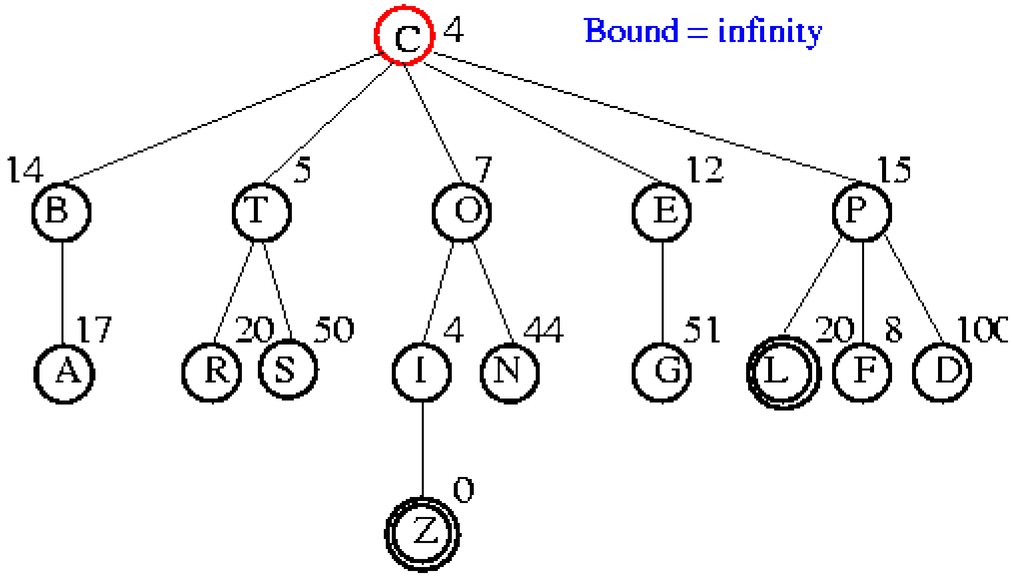
RBFS

- Recursive Best First Search
  - Linear space variant of A*
- Perform A* search but discard subtrees when perform recursion
- Keep track of alternative (next best) subtree
- Expand subtree until f value greater than bound
- Update f values before (from parent) and after (from descendant) recursive call

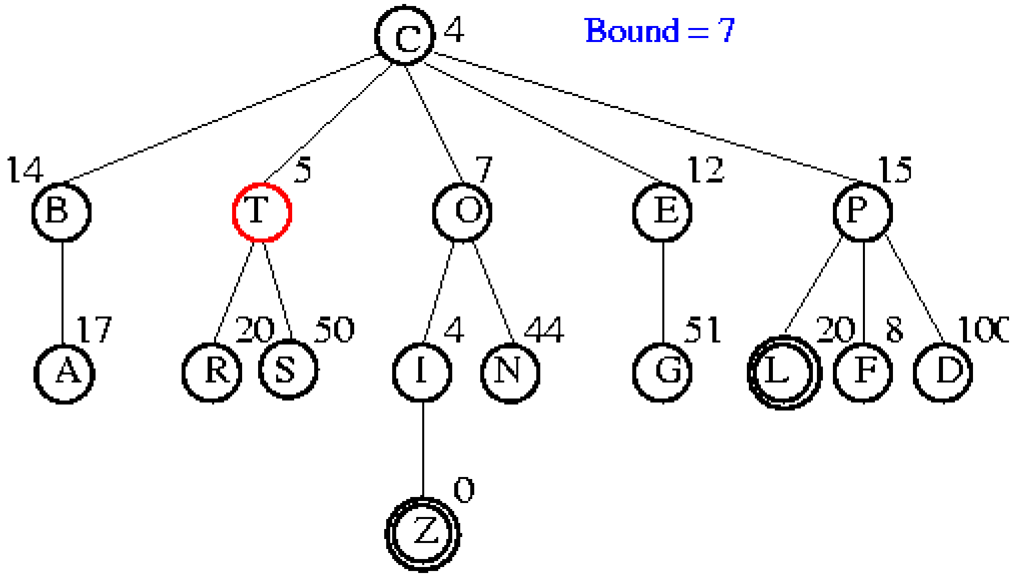
Algorithm

```
// Input is current node and f limit
// Returns goal node or failure, updated limit
RBFS(n, limit)
  if Goal(n)
    return n
  children = Expand(n)
  if children empty
    return failure, infinity
  for each c in children
    f[c] = max(g(c)+h(c), f[n])                // Update f[c] based on parent
  repeat
    best = child with smallest f value
    if f[best] > limit
      return failure, f[best]
    alternative = second-lowest f-value among children
    newlimit = min(limit, alternative)
    result, f[best] = RBFS(best, newlimit)   // Update f[best] based on descendant
  if result not equal to failure
    return result
```

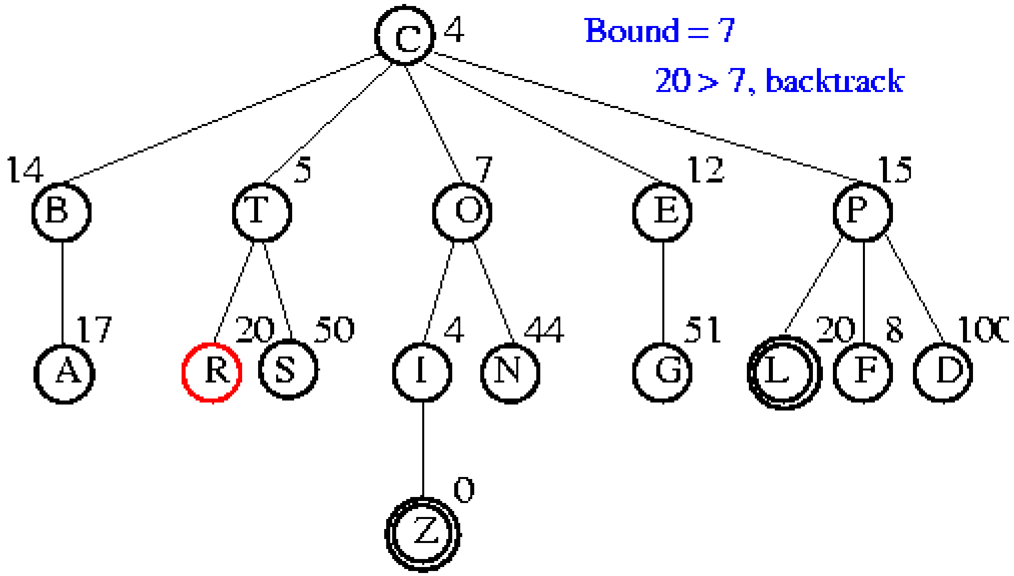
Example


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

Example

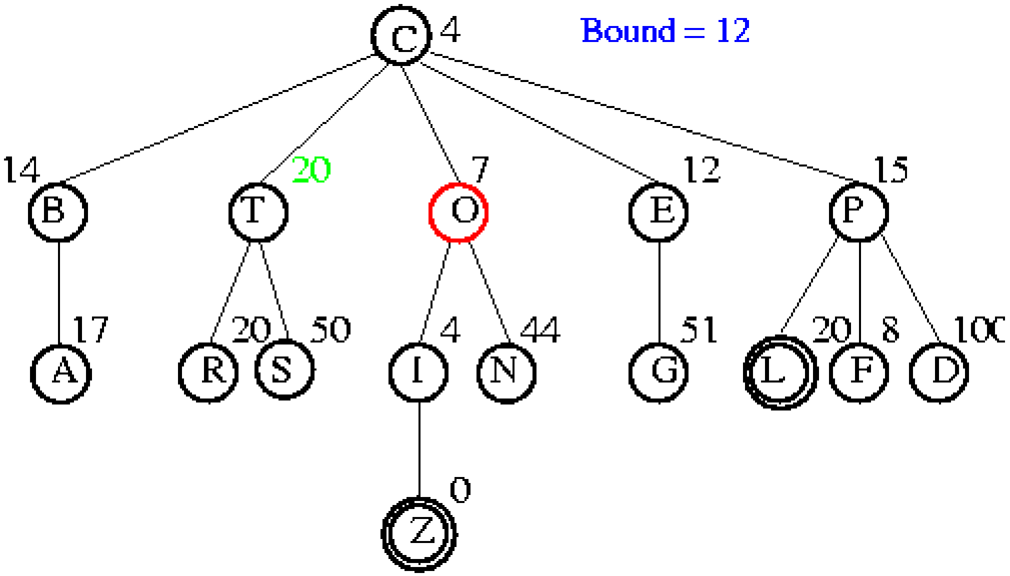


Example



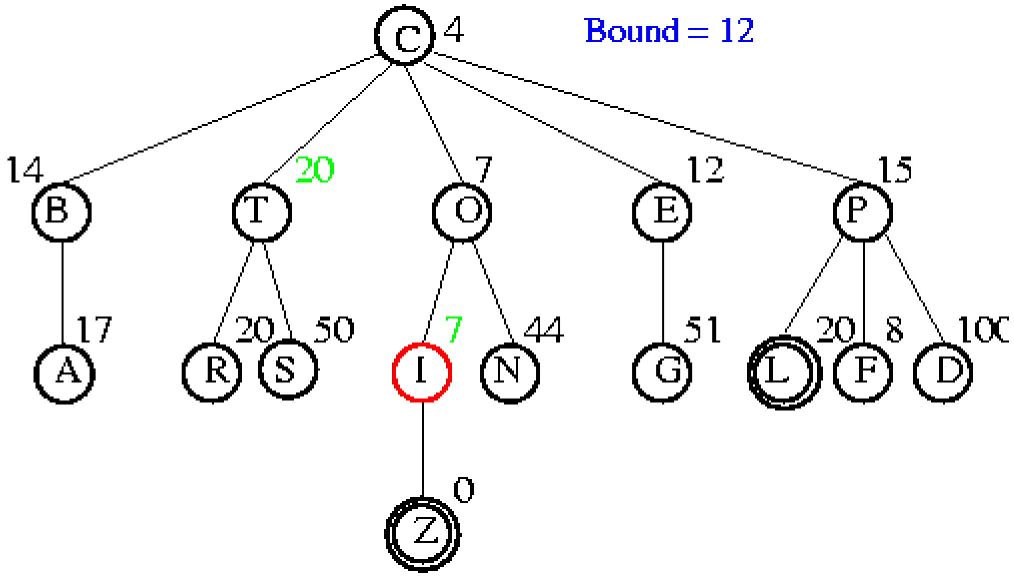
C
4
Bound = 7
20 > 7, backtrack
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
20
L
8
F
100
D
0
Z

Example



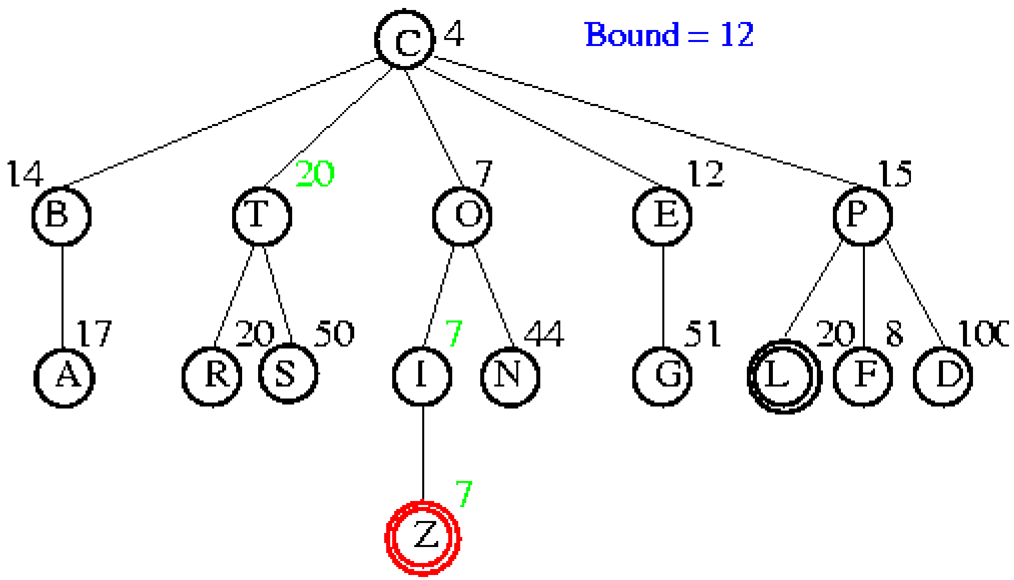
C
4
Bound = 12
14
B
20
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
20
L
8
F
100
D
0
Z

Example



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

Example



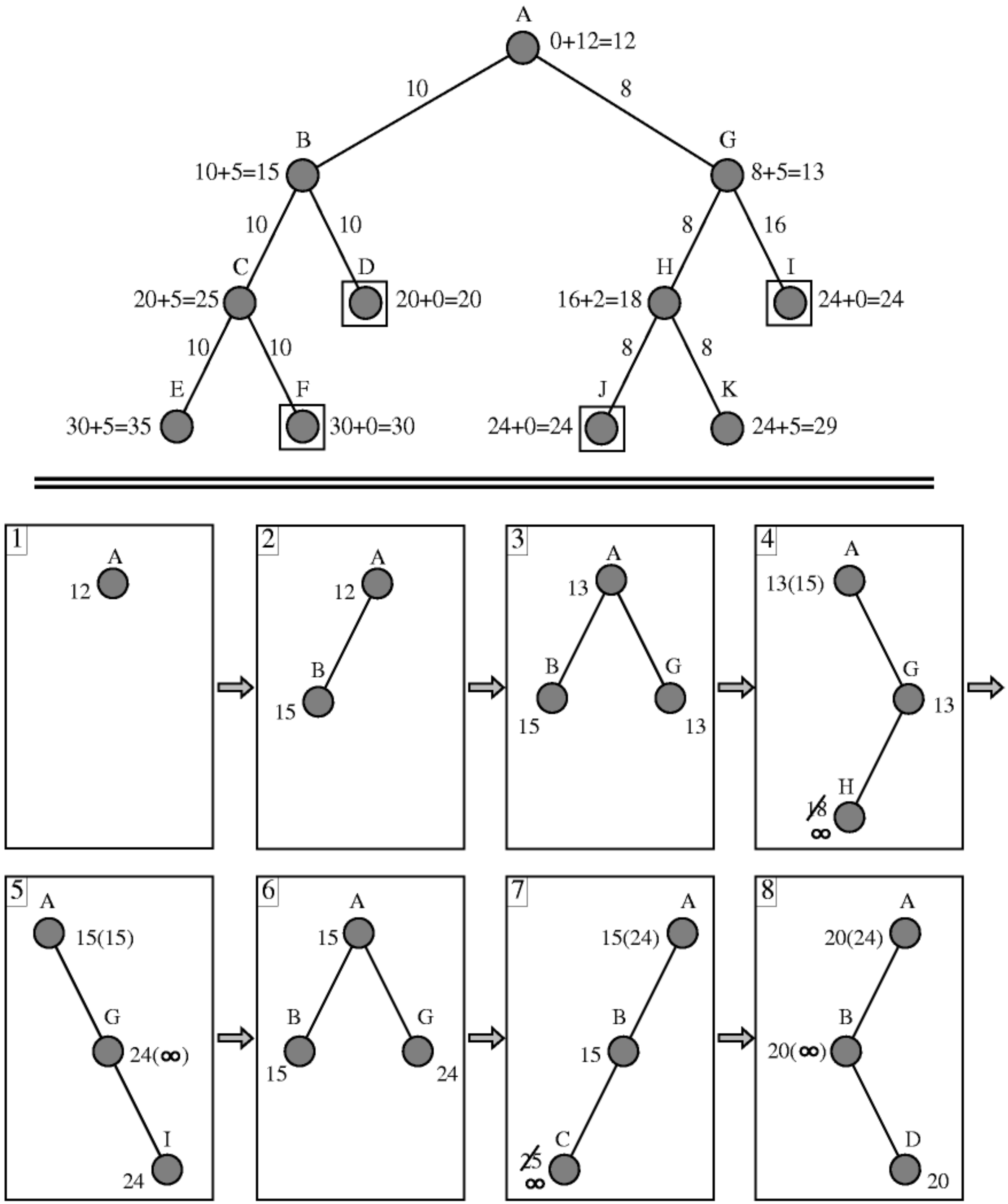
C
4
Bound = 12
14
B
20
T
7
O
12
E
15
P
17
A
20
R
50
S
7
I
44
N
51
G
20
L
8
F
100
D
7
Z

Analysis

- Optimal if h(n) is admissible
- Time is O(bm)
- Features
  - Potentially exponential time in cost of solution
  - More efficient than IDA*
  - Keeps more information than IDA* but benefits from storing this information

SMA*

- Simplified Memory-Bounded A* Search
- Perform A* search
- When memory is full
  - Discard worst leaf (largest f(n) value)
  - Back value of discarded node to parent
- Optimal if solution fits in memory



Example

- Let MaxNodes = 3
- Initially B&G added to open list, then hit max
- B is larger f value so discard but save f(B)=15 at parent A
  - Add H but f(H)=18. Not a goal and cannot go deper, so set f(h)=infinity and save at G.
- Generate next child I with f(I)=24, bigger child of A. We have seen all children of G, so reset f(G)=24.
- Regenerate B and child C. This is not goal so f(c) reset to infinity
- Generate second child D with f(D)=24, backing up value to ancestors
- D is a goal node, so search terminates.

Heuristic Functions

- Q: Given that we will only use heuristic functions that do not overestimate, what type of heuristic functions (among these) perform best?
- A: Those that produce higher h(n) values.

Reasons

- Higher h value means closer to actual distance
- Any node n on open list with
  - $f(n) < f^*(goal)$
  - will be selected for expansion by A*
- This means if a lot of nodes have a low underestimate (lower than actual optimum cost)
  - All of them will be expanded
  - Results in increased search time and space

Informedness

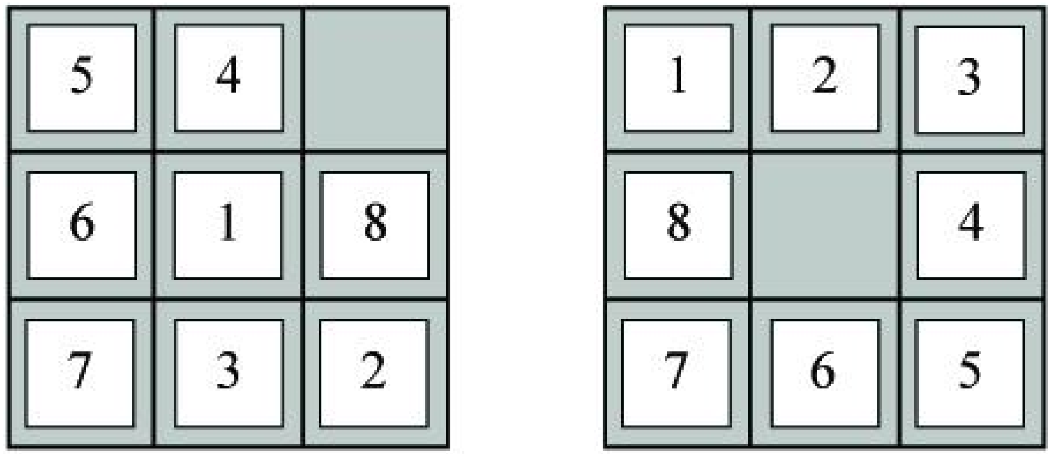
- If h1 and h2 are both admissible and
- For all x, h1(x) > h2(x), then h1 "dominates" h2
  - Can also say h1 is "more informed" than h2
- Example
  - h1(x): $|x_{goal} - x|$
  - h2(x): Euclidean distance $\sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2}$
  - h2 dominates h1

Effect on Search Cost

- If h2(n) >= h1(n) for all n (both are admissible)
  - then h2 dominates h1 and is better for search
- Typical search costs
  - d=14, IDS expands 3,473,941 nodes
    - A* with h1 expands 539 nodes
    - A* with h2 expands 113 nodes
  - d=24, IDS expands ~54,000,000,000 nodes
    - A* with h1 expands 39,135 nodes
    - A* with h2 expands 1,641 nodes

Which of these heuristics are admissible?

Which are more informed?



Start State Goal State

- h1(n) = #tiles in wrong position
- h2(n) = Sum of Manhattan distance between each tile and goal location for the tile
- h3(n) = 0
- h4(n) = 1
- h5(n) = min(2, h*[n])
- h6(n) = Manhattan distance for blank tile
- h7(n) = max(2, h*[n])

Generating Heuristic Functions

- Generate heuristic for simpler (relaxed) problem
  - Relaxed problem has fewer restrictions
  - Eight puzzle where multiple tiles can be in the same spot
  - Cost of optimal solution to relaxed problem is an admissible heuristic for the original problem
- Learn heuristic from experience