Informed search algorithms

Chapter 4, Sections 1–2

Outline

- ♢ Best-first search
- ♢ A^* search
- ♢ Heuristics

Review: Tree search

```
function Tree-Search( problem, fringe) returns a solution, or failure
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← Remove-Front(fringe)
        if Goal-Test[problem] applied to State(node) succeeds return node
        fringe ← InsertAll(Expand(node, problem), fringe)
```

A strategy is defined by picking the **order of node expansion**

Best-first search

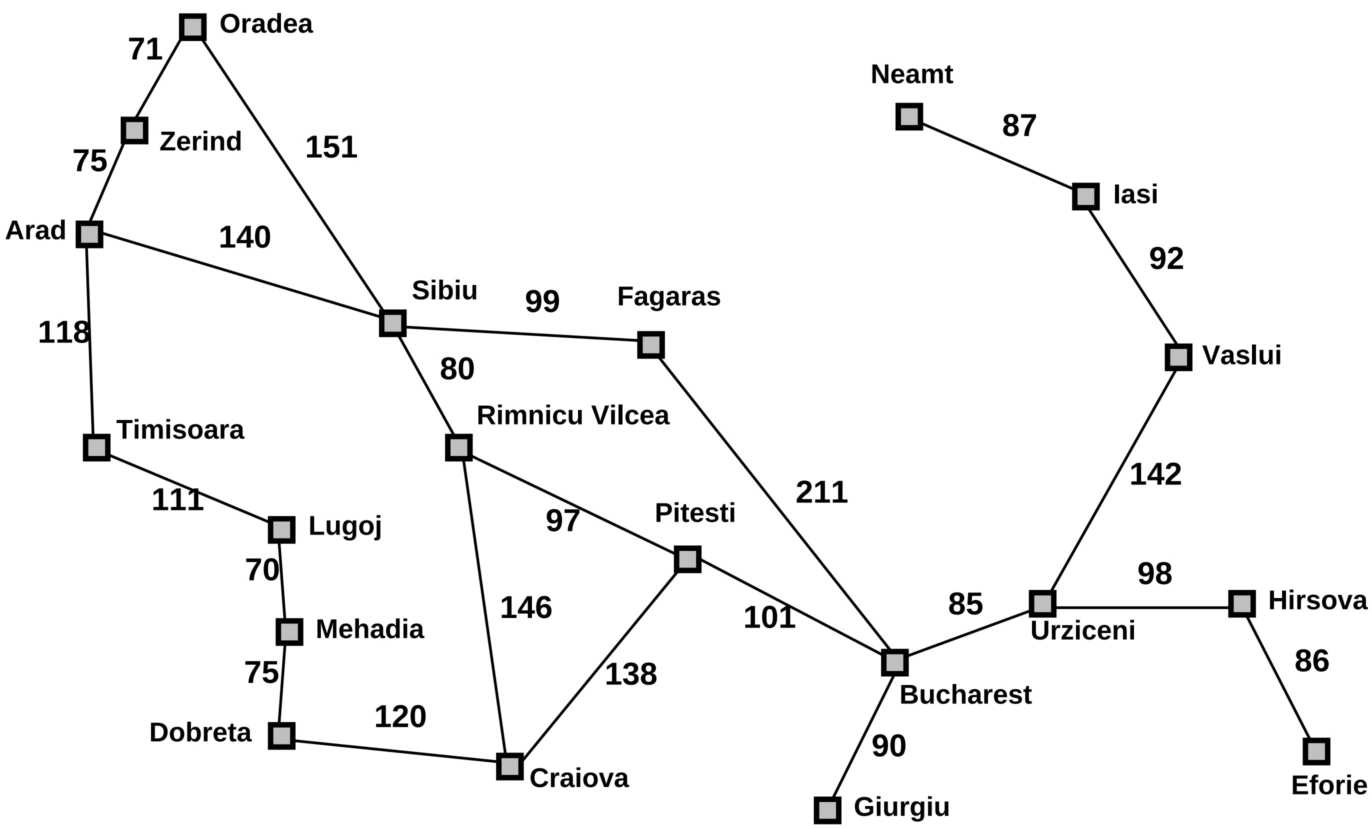
Idea: use an evaluation function for each node
– estimate of "desirability"

⇒ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy 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	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

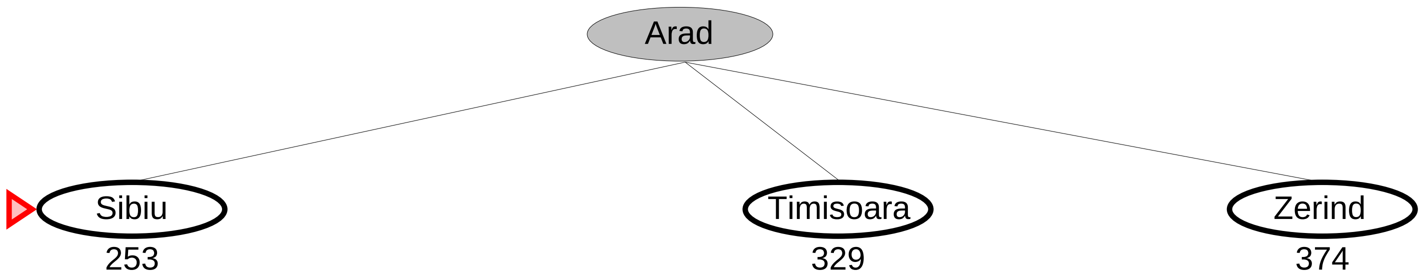
Evaluation function $h(n)$ (**h**euristic)
$= $ estimate of cost from n to the closest goal

E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

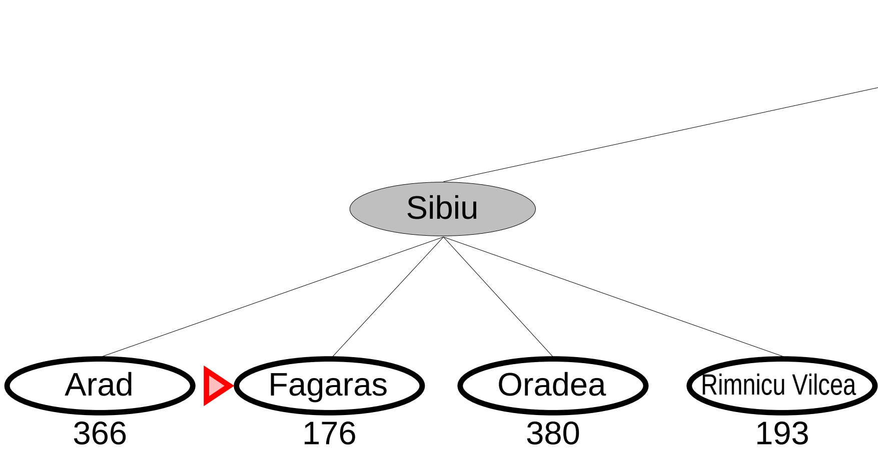
Greedy search expands the node that **appears** to be closest to goal

Greedy search example

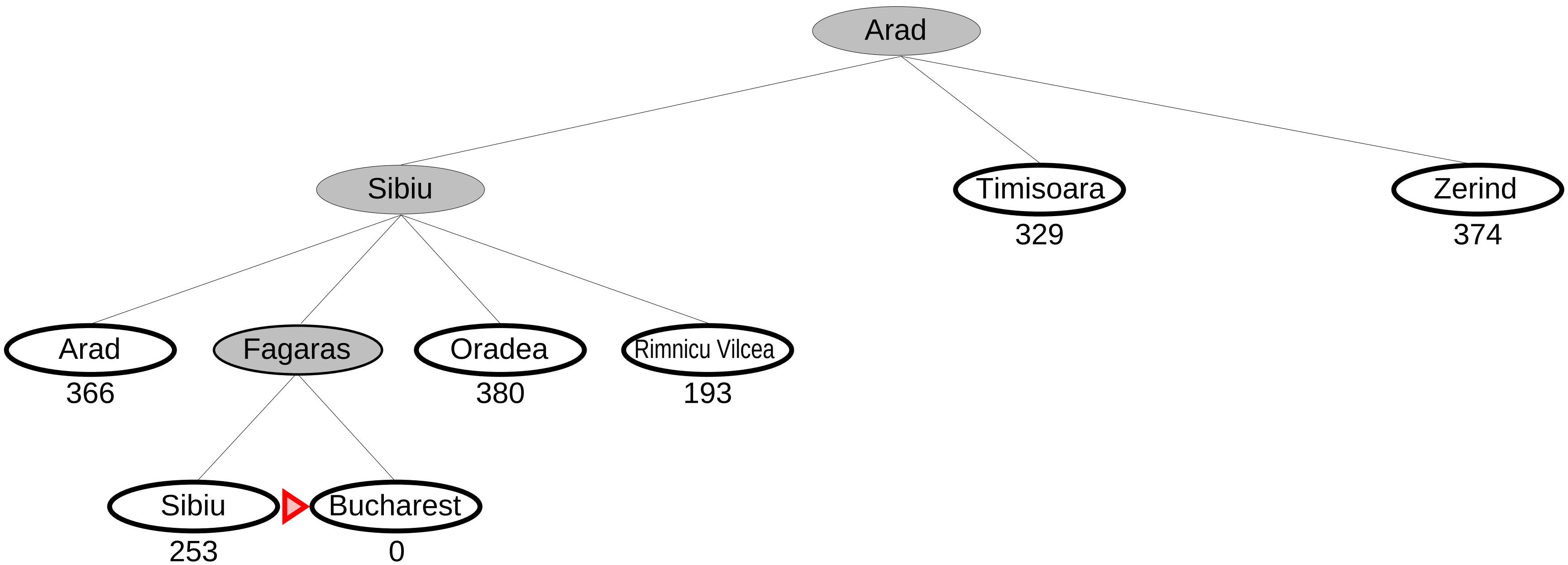
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A^* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

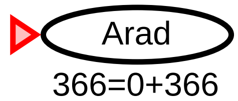
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A^* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

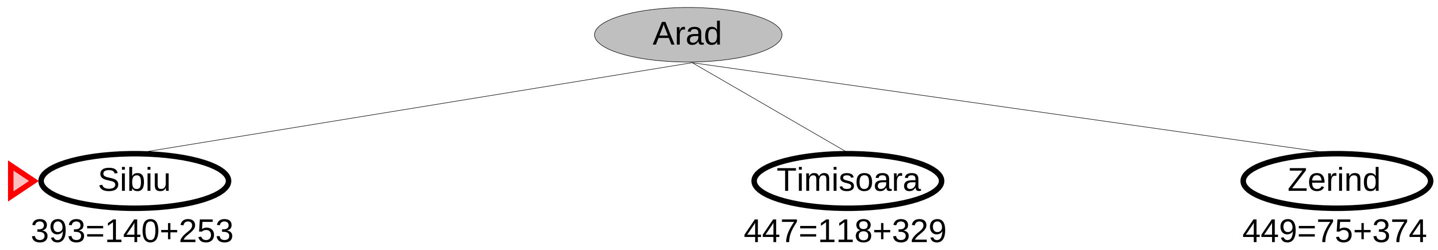
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A^* search is optimal

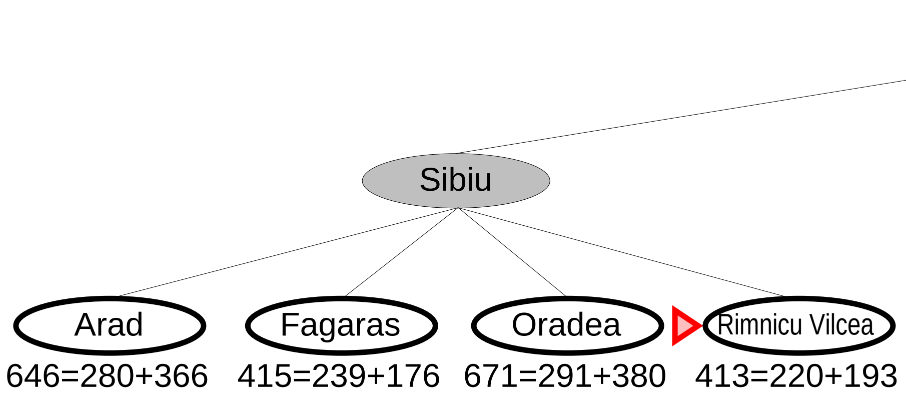
A* search example



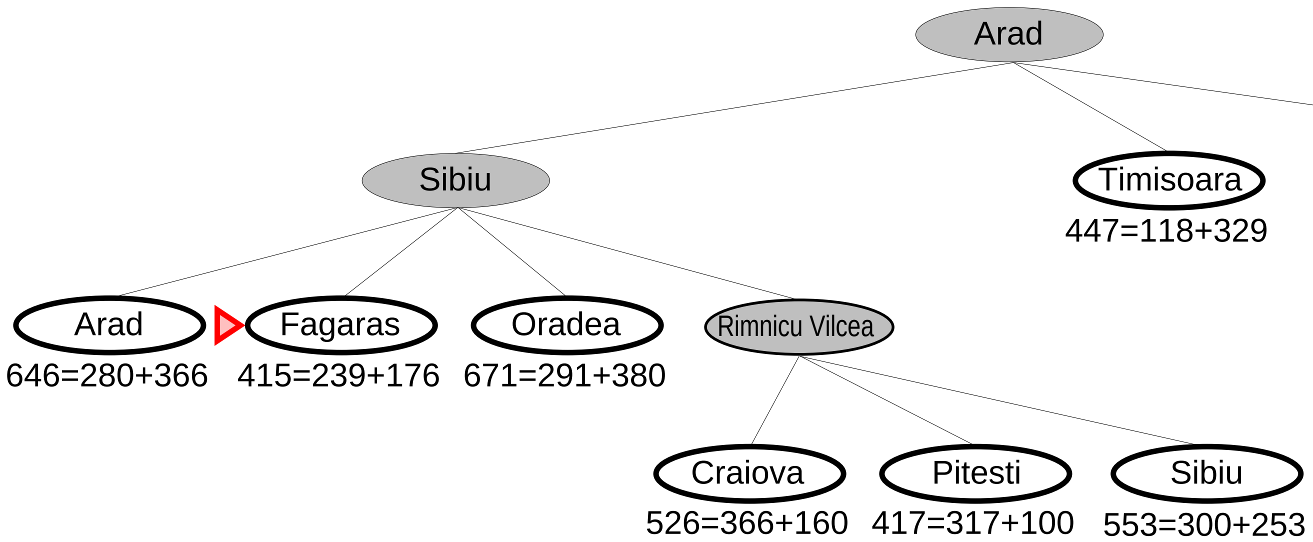
A* search example



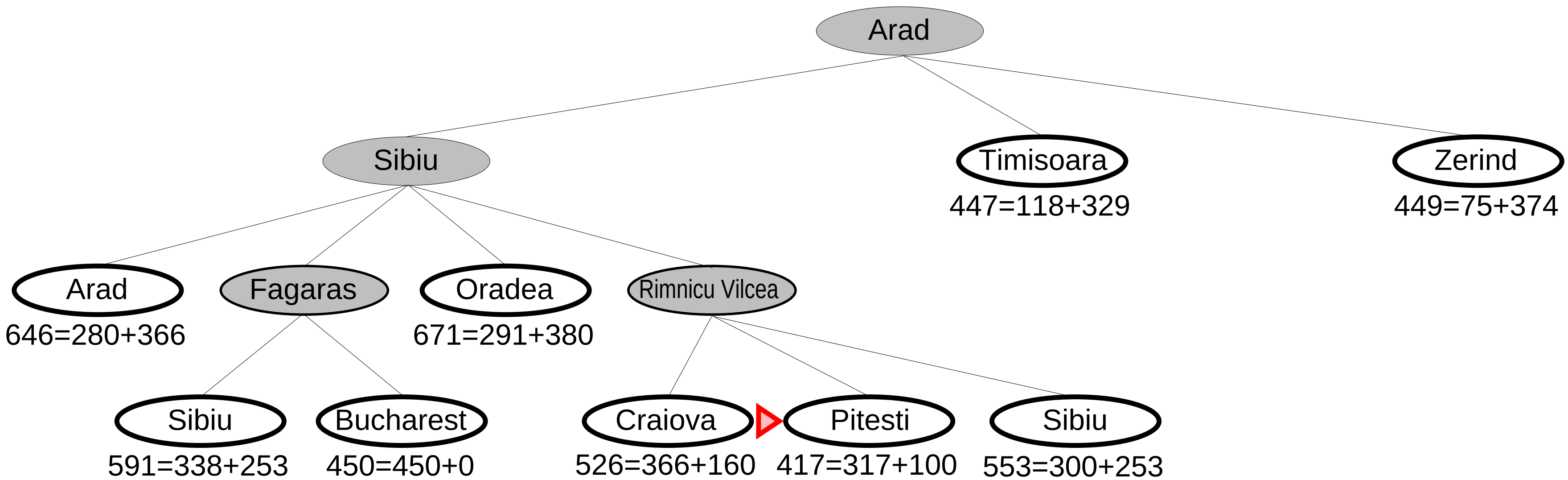
A* search example



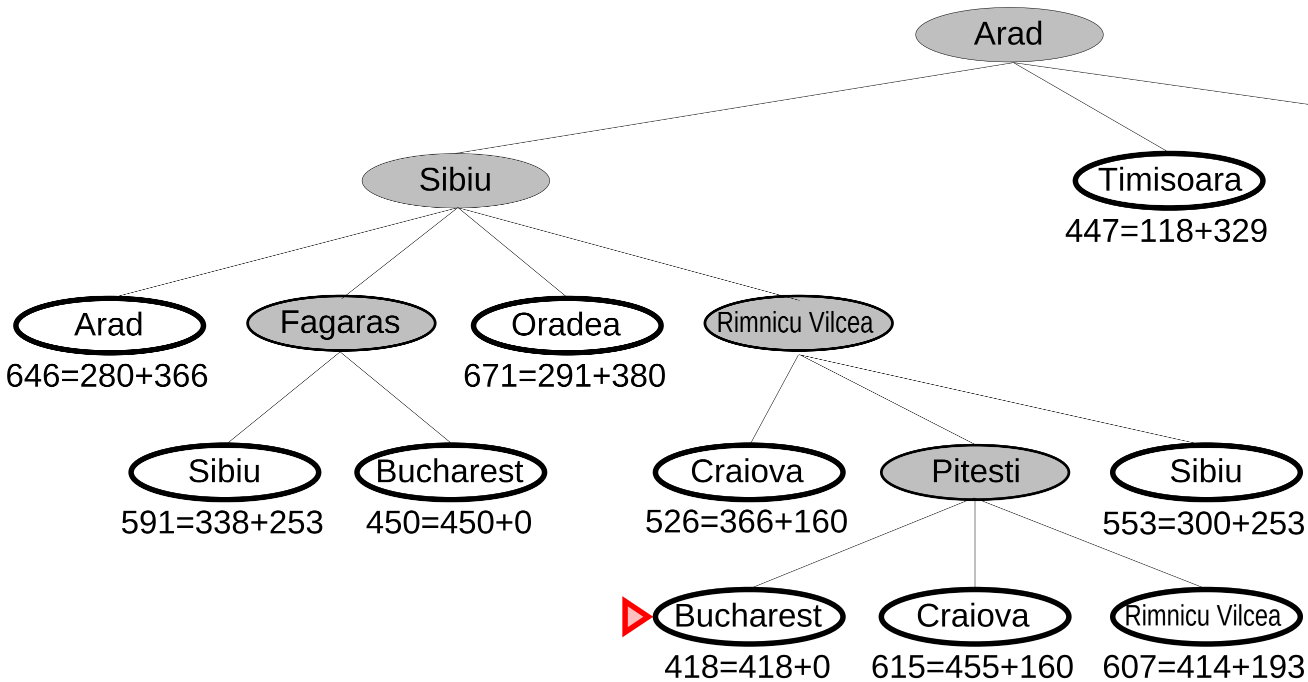
A* search example



A* search example

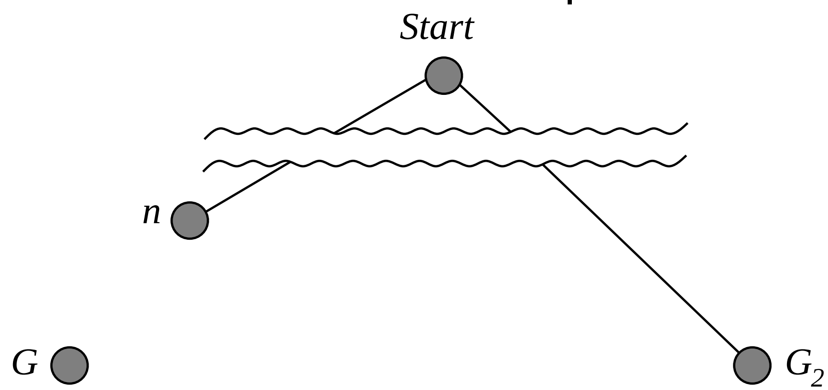


A* search example



Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue.
Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2)=0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

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

Properties of A*

Complete??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

IDA*

Series of Depth-First Searches

Like Iterative Deepening Search, except use A* cost threshold instead of depth threshold

Ensures optimal solution

queueing-fn is enqueue-at-front if f(child) $\leq$ threshold

Threshold is h(root) for first pass

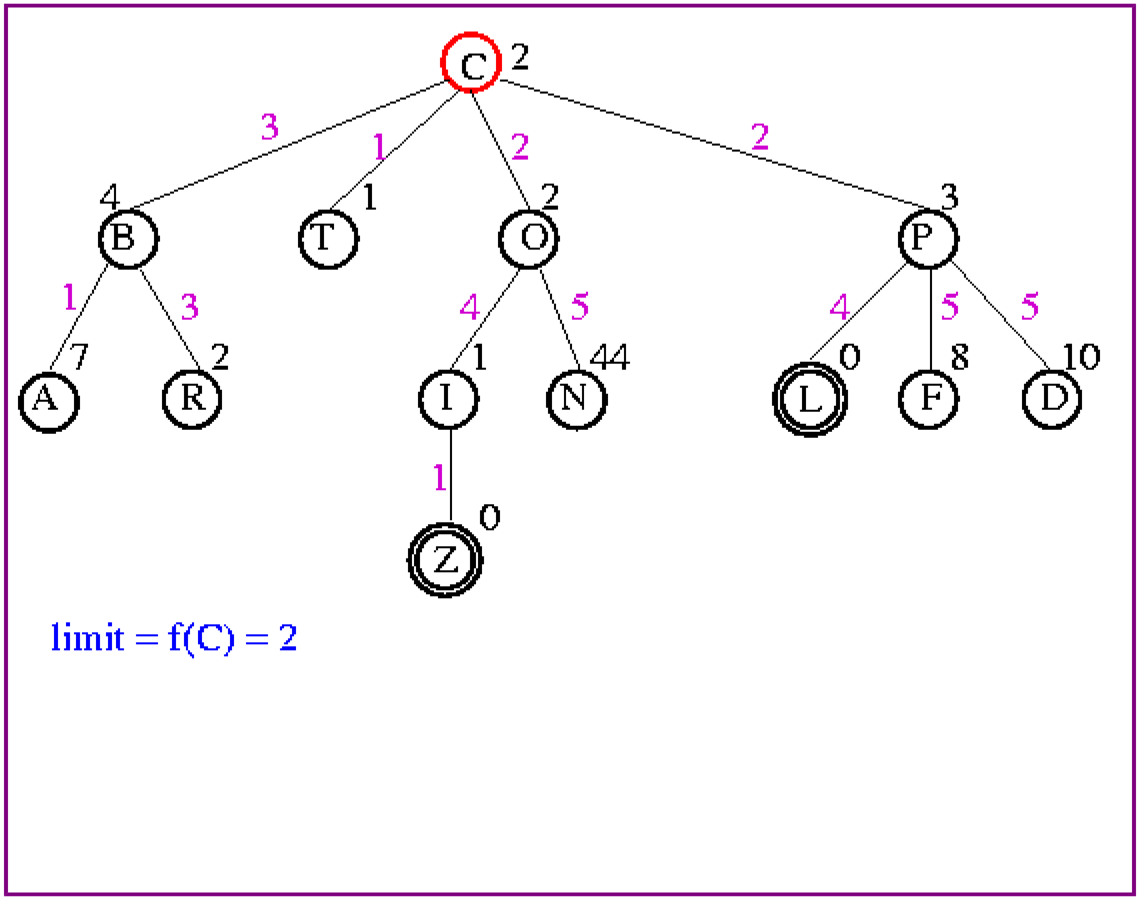
Next threshold is f(min_child),
where min_child is cutoff child with minimum f value

This conservative increase ensures cannot look past optimal cost solution

---

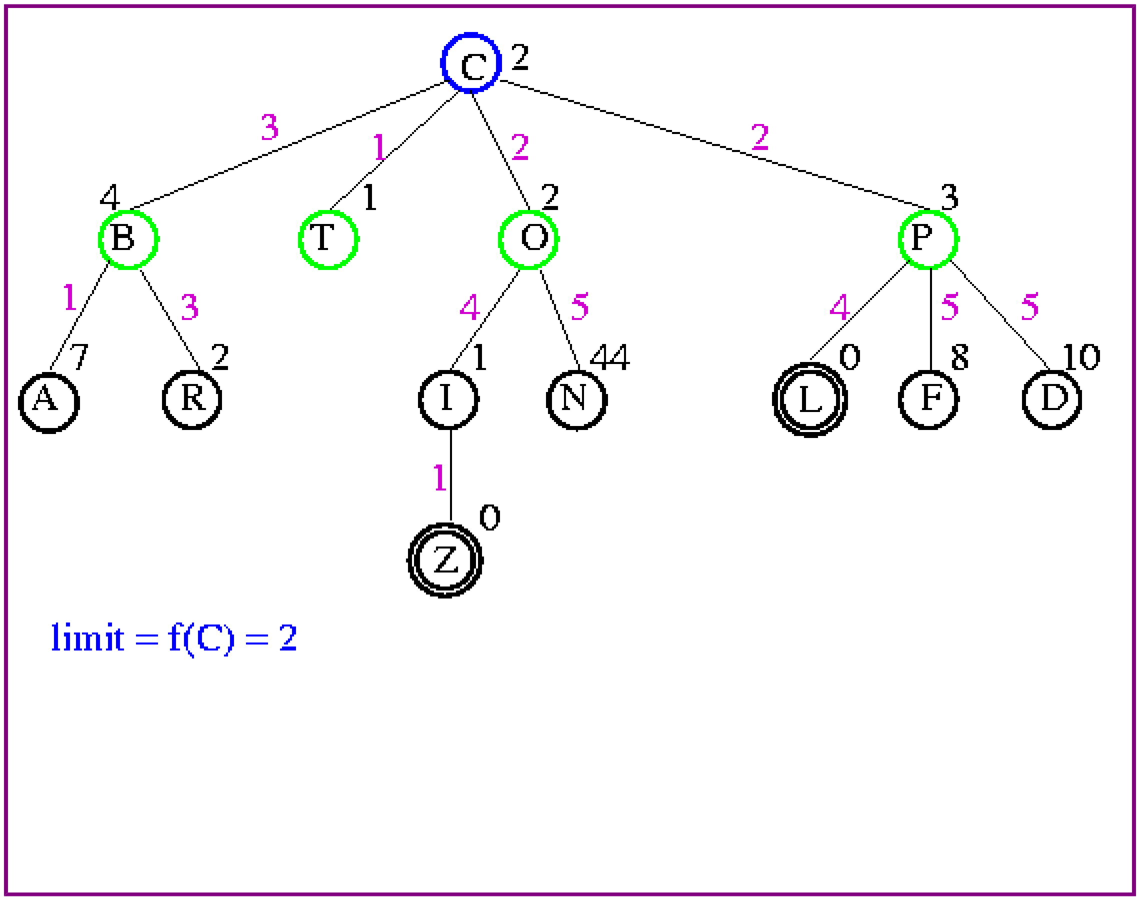
Example

Click mouse to advance to next frame.



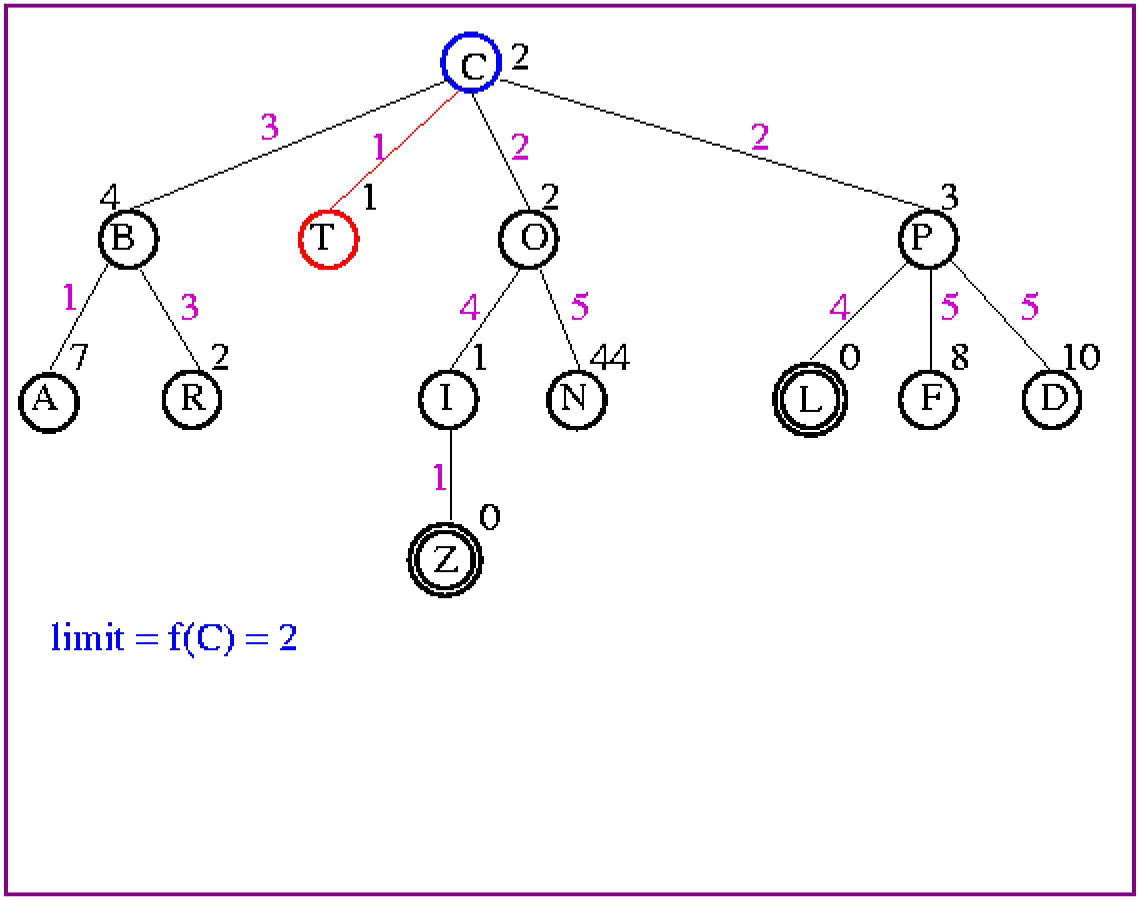
Example

Click mouse to advance to next frame.



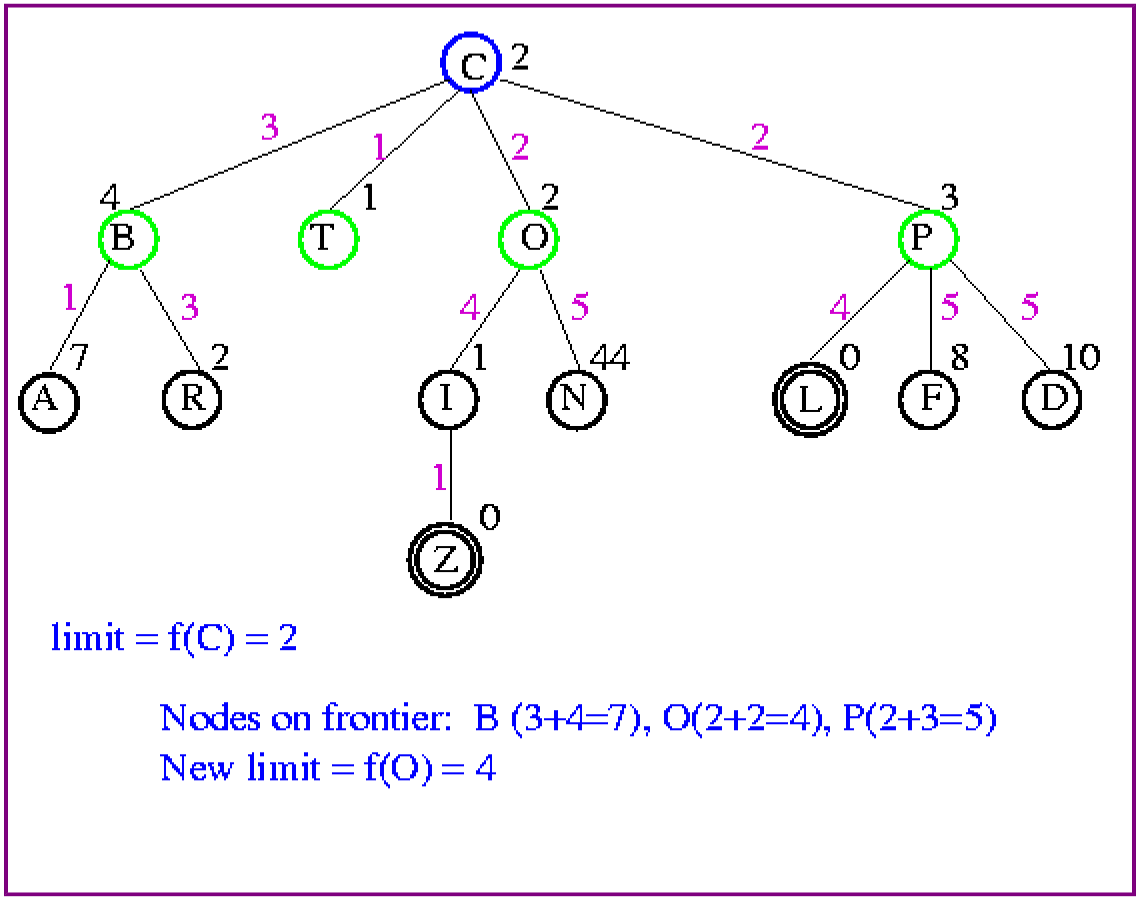
Example

Click mouse to advance to next frame.



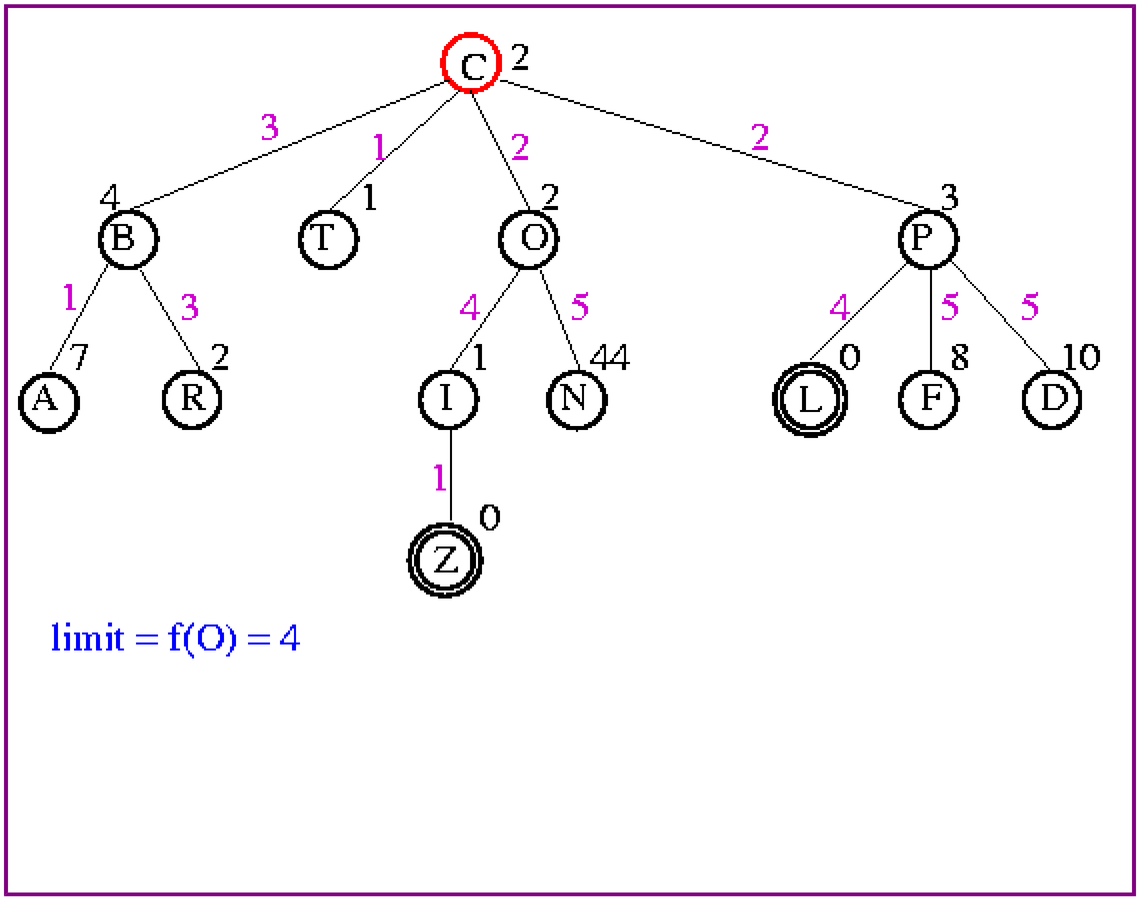
Example

Click mouse to advance to next frame.



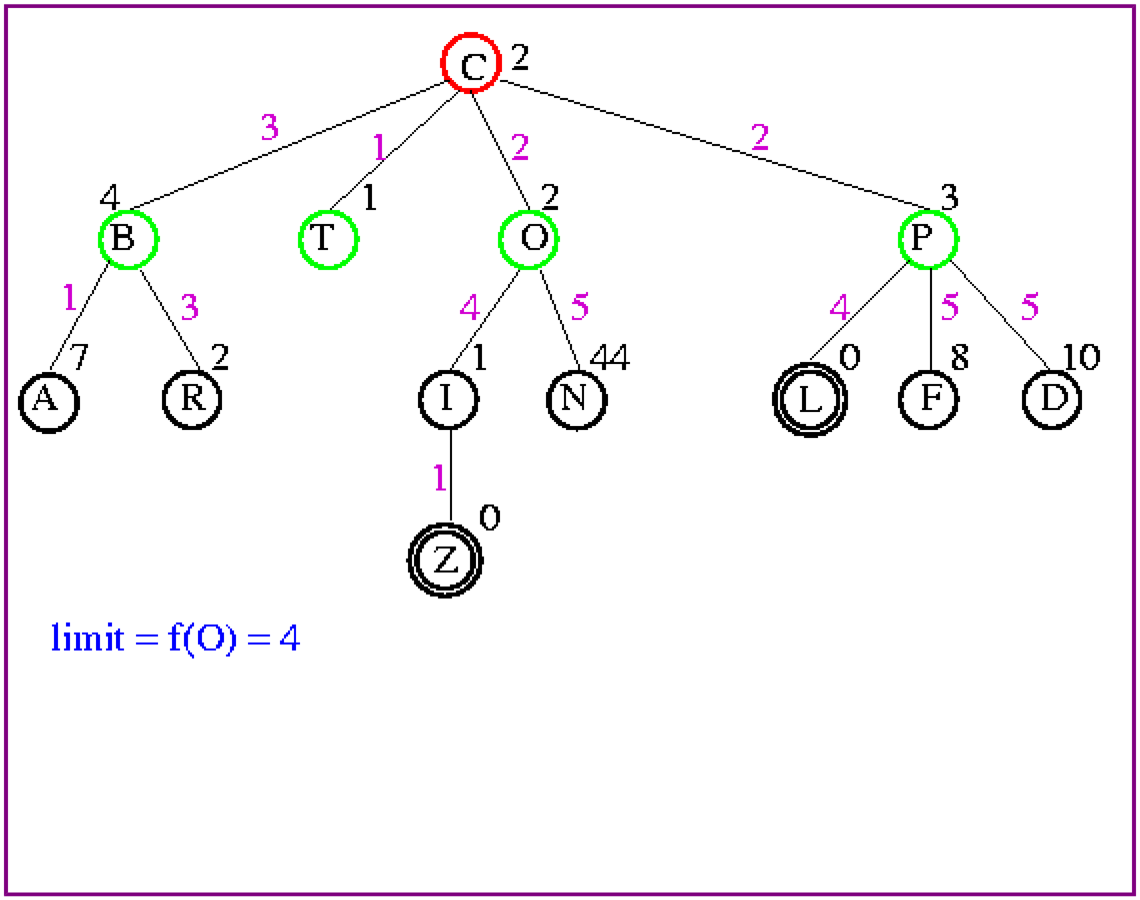
Example

Click mouse to advance to next frame.



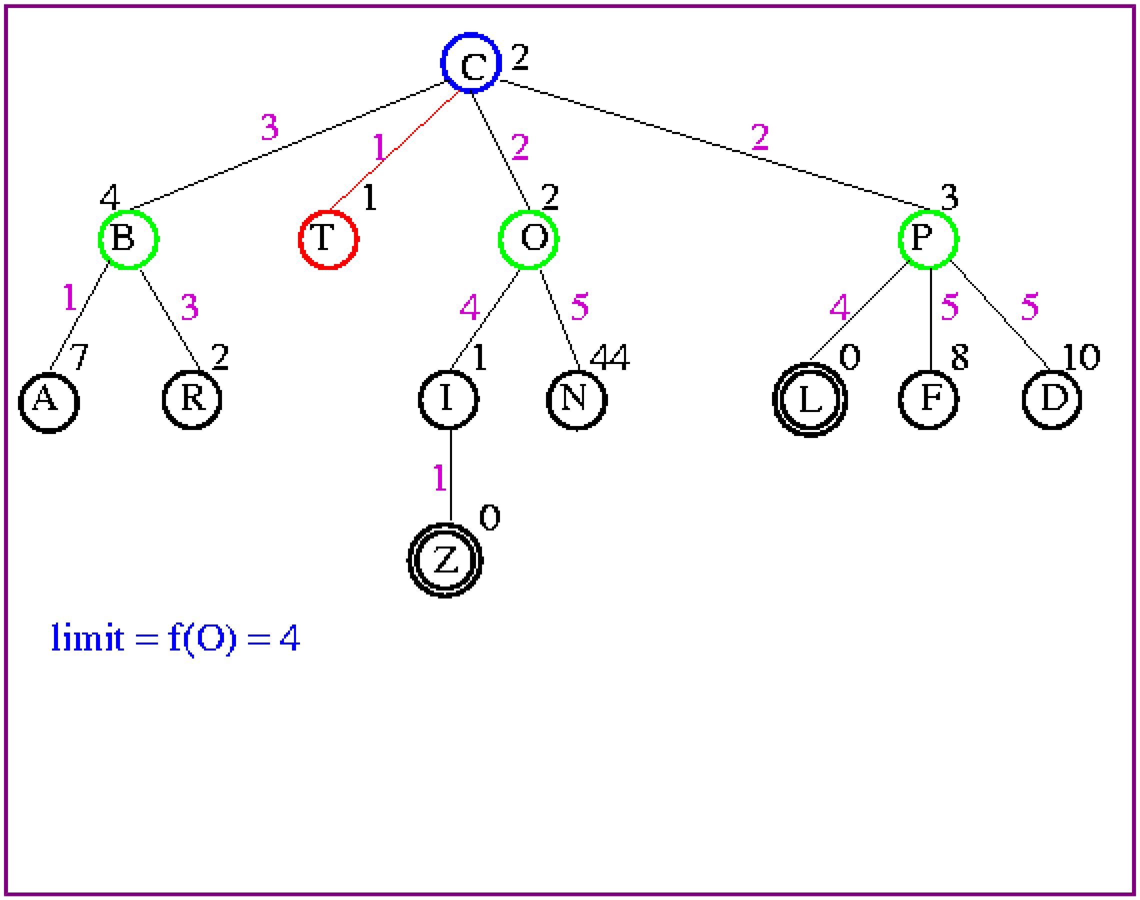
Example

Click mouse to advance to next frame.



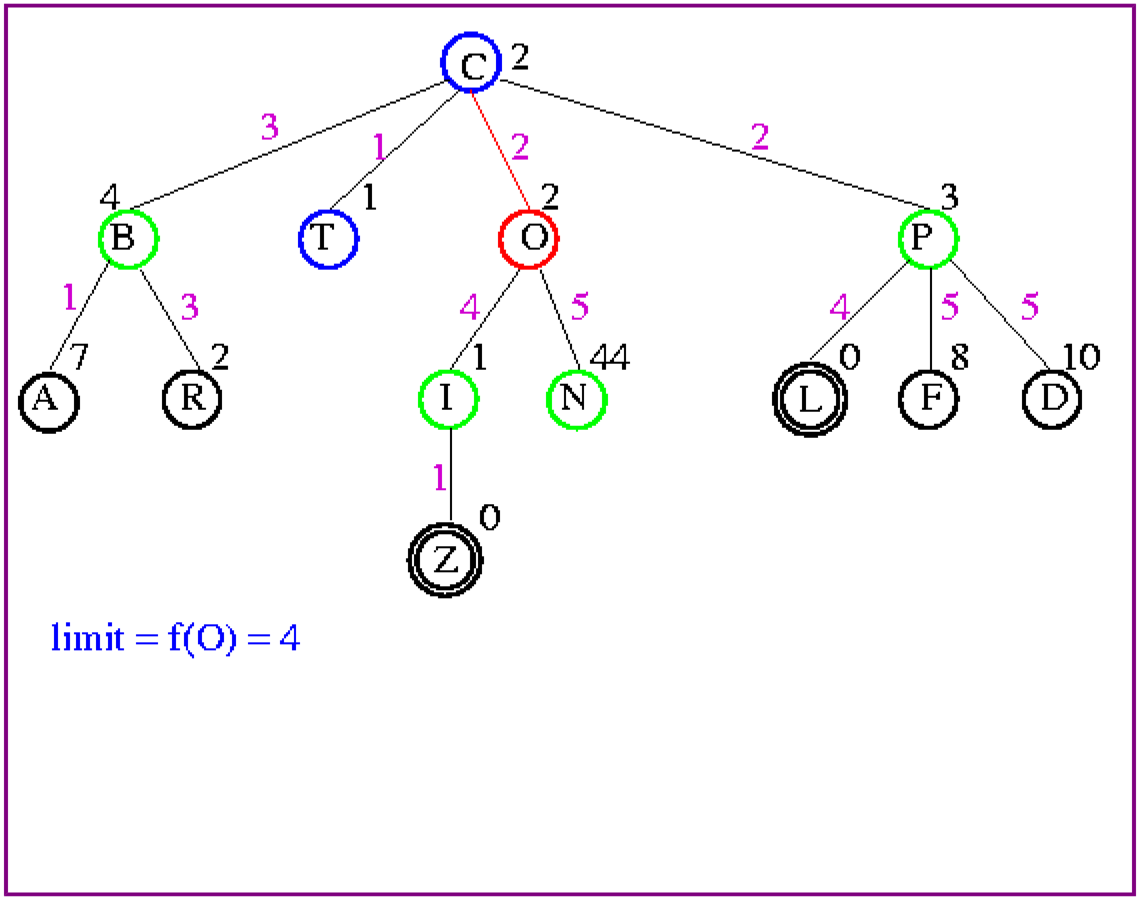
Example

Click mouse to advance to next frame.



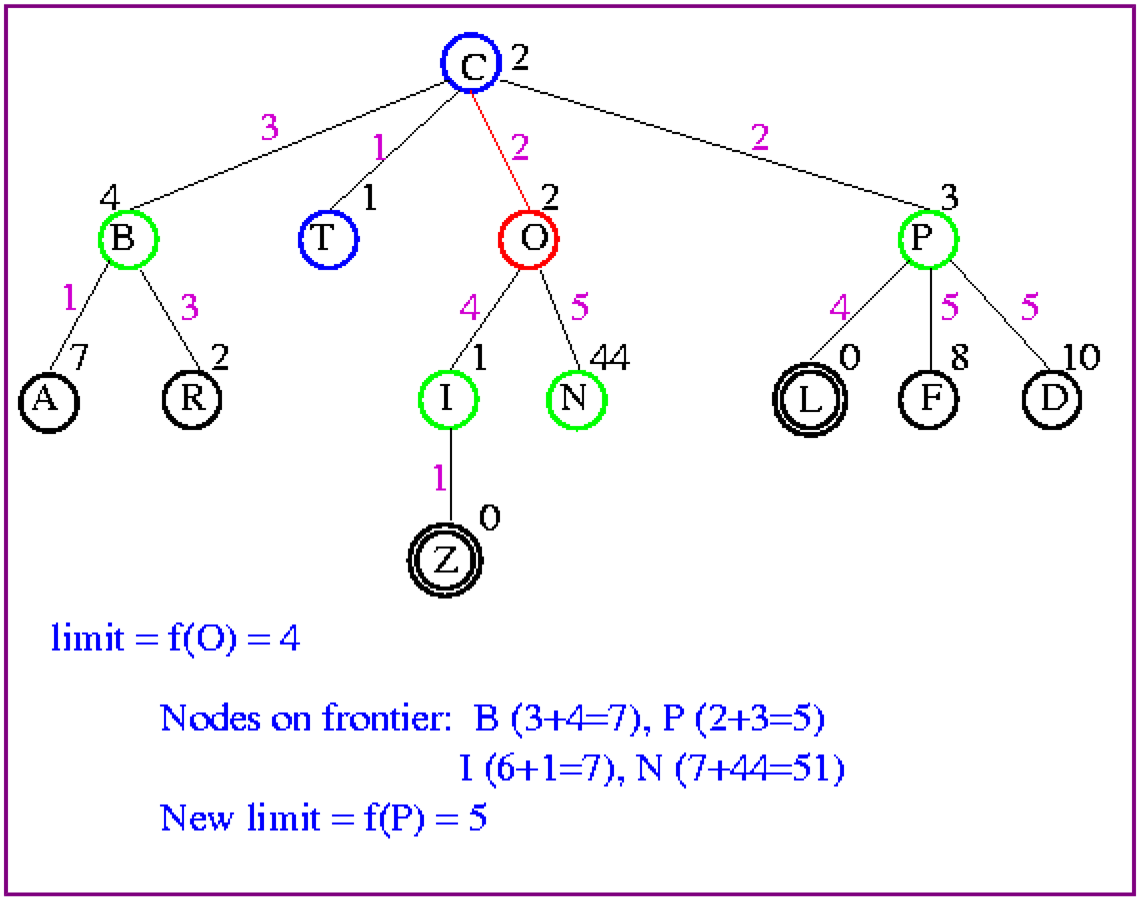
Example

Click mouse to advance to next frame.



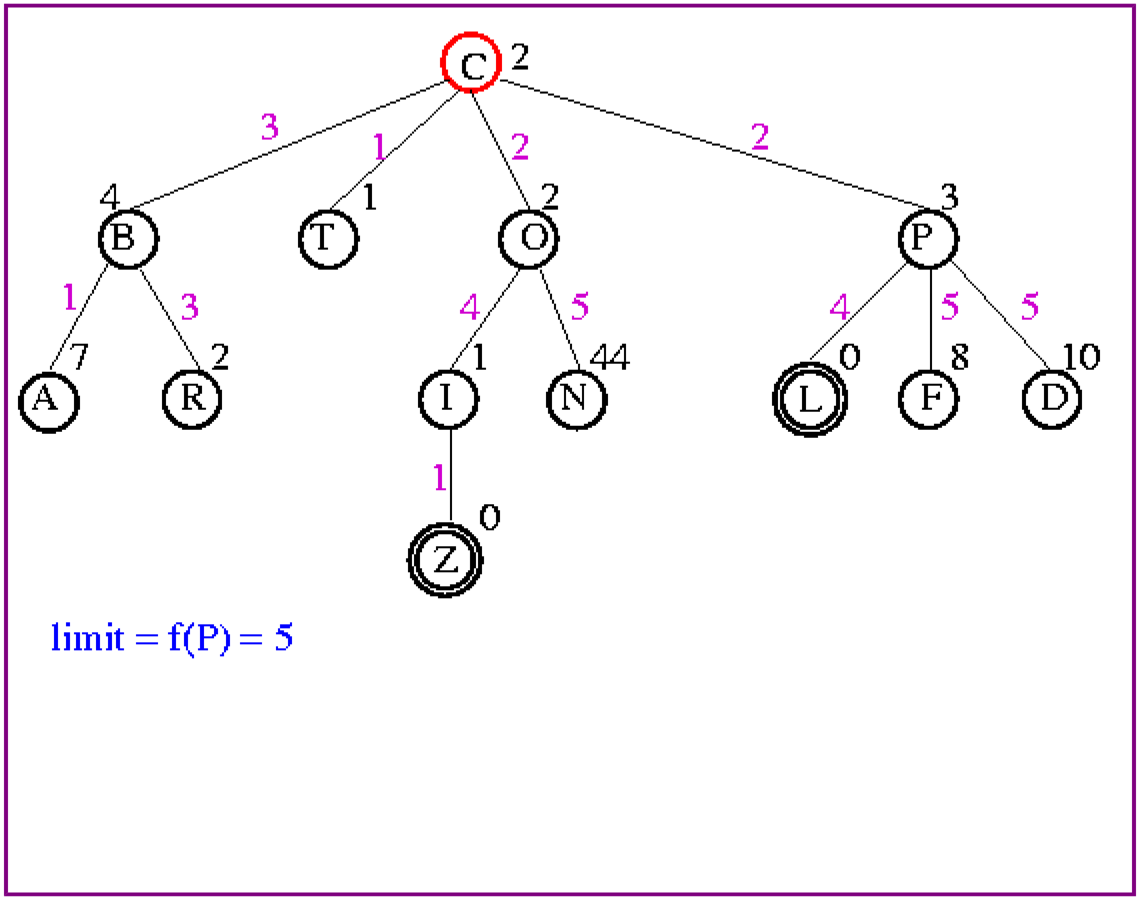
Example

Click mouse to advance to next frame.



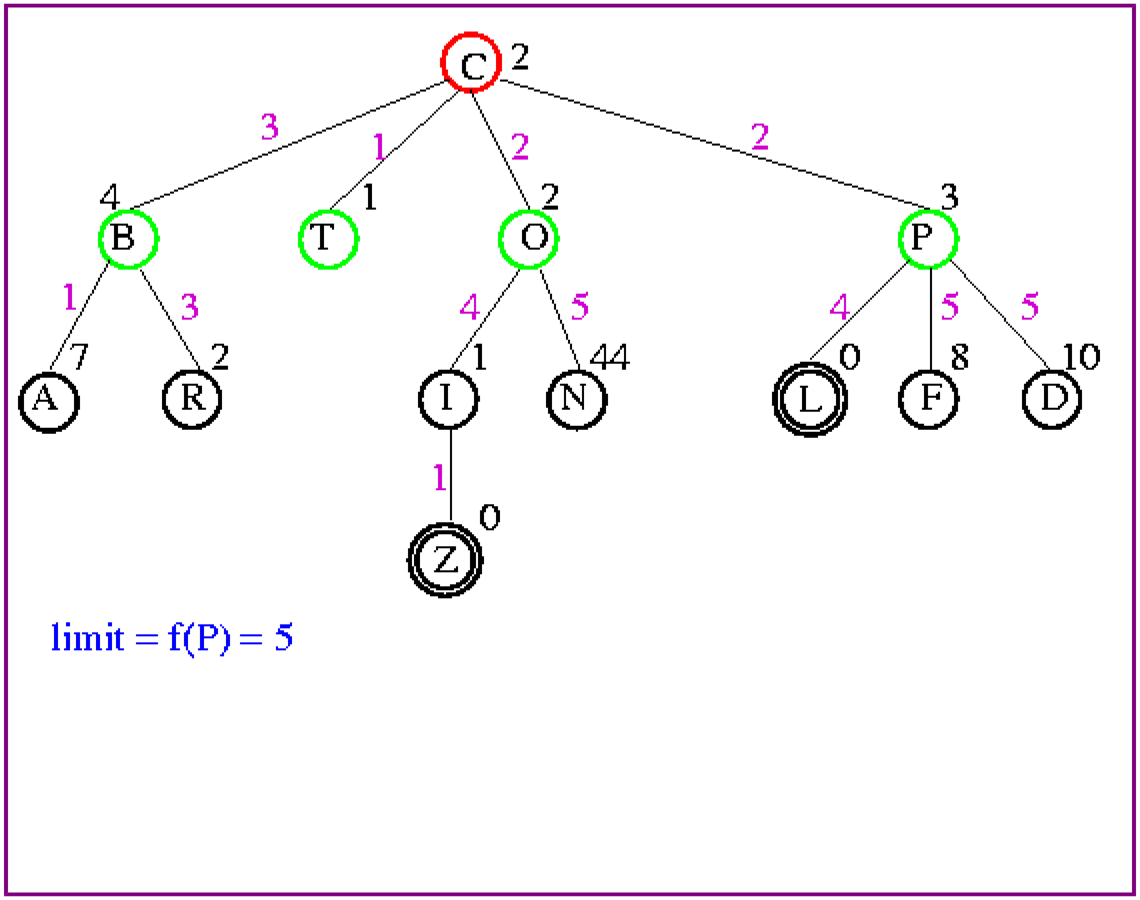
Example

Click mouse to advance to next frame.



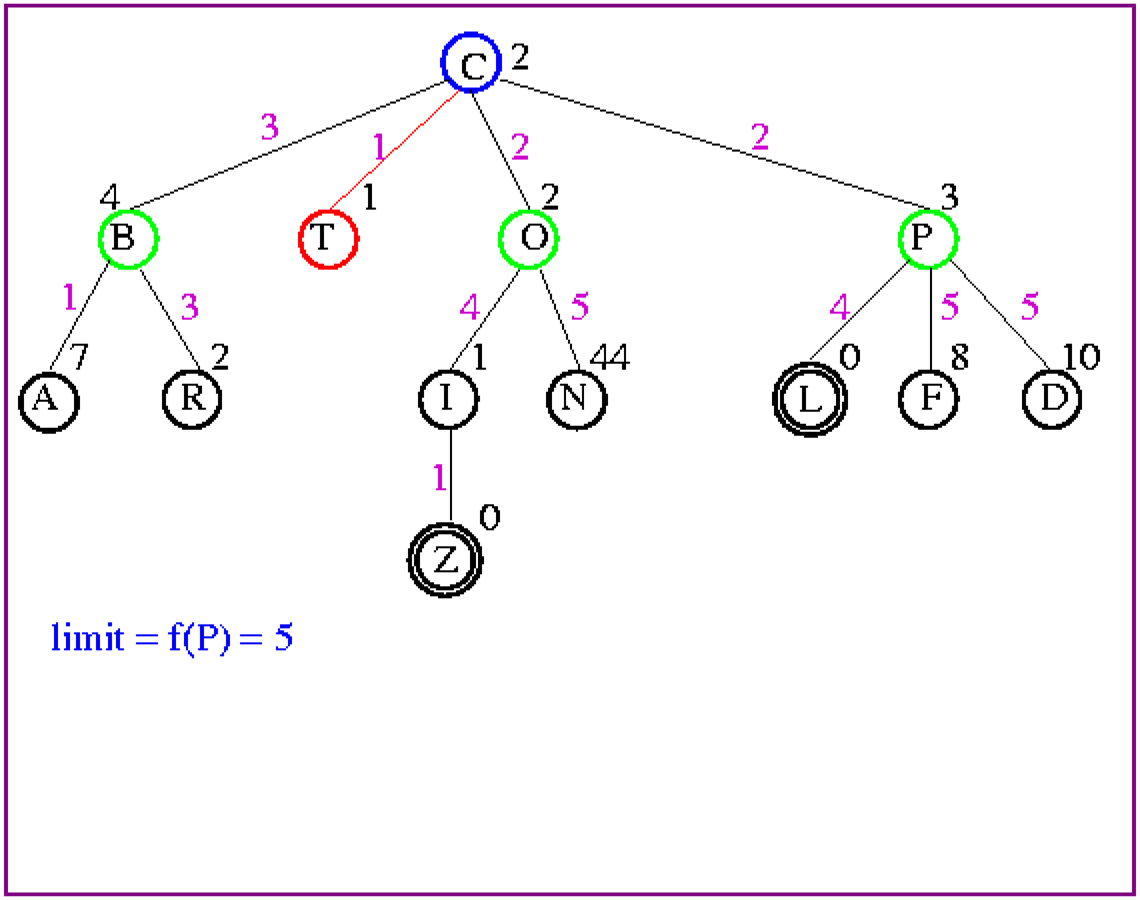
Example

Click mouse to advance to next frame.



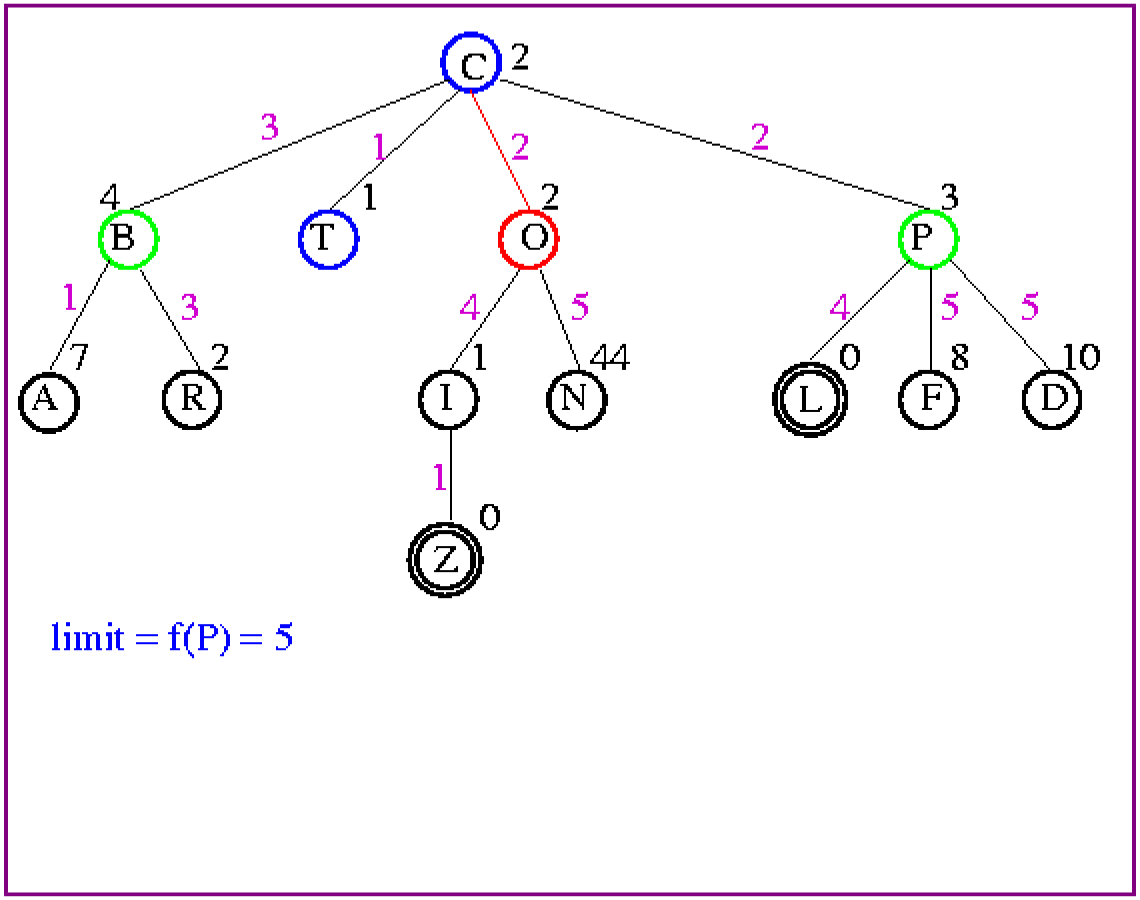
Example

Click mouse to advance to next frame.



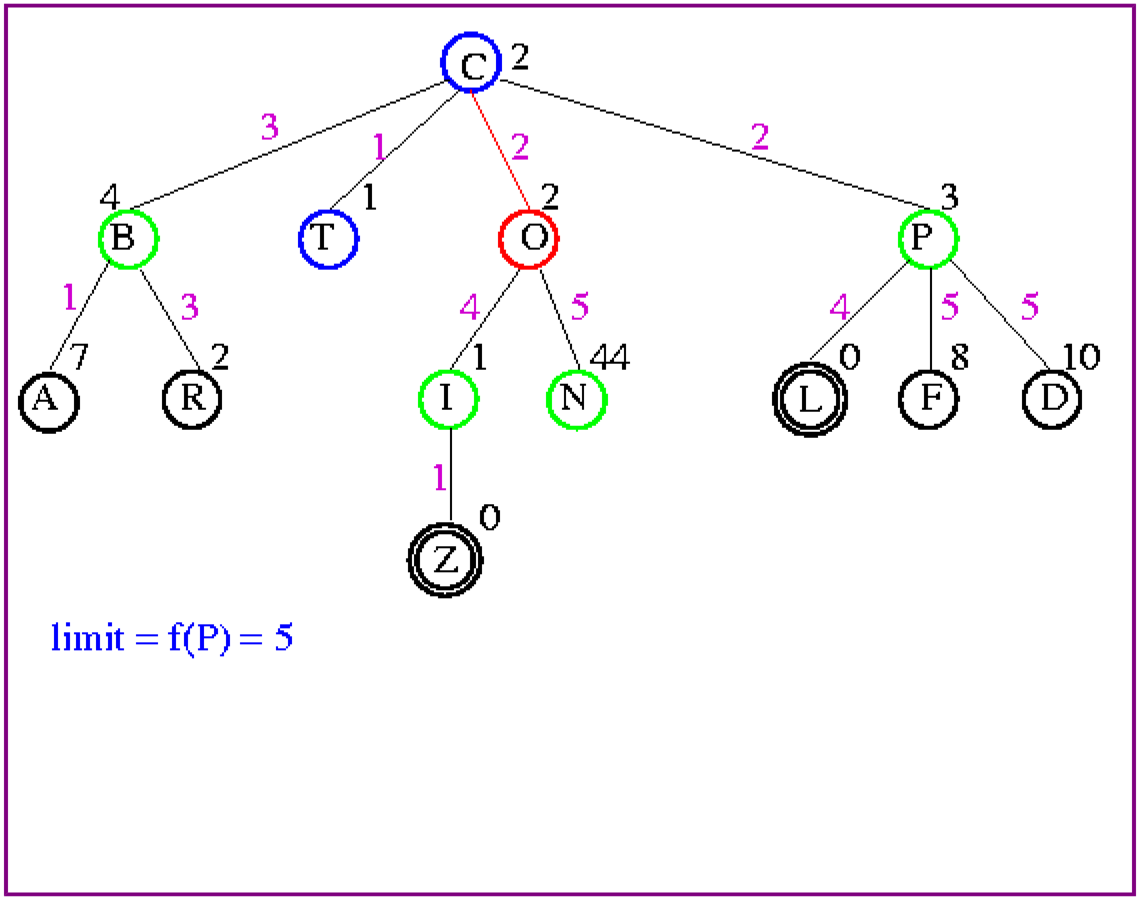
Example

Click mouse to advance to next frame.



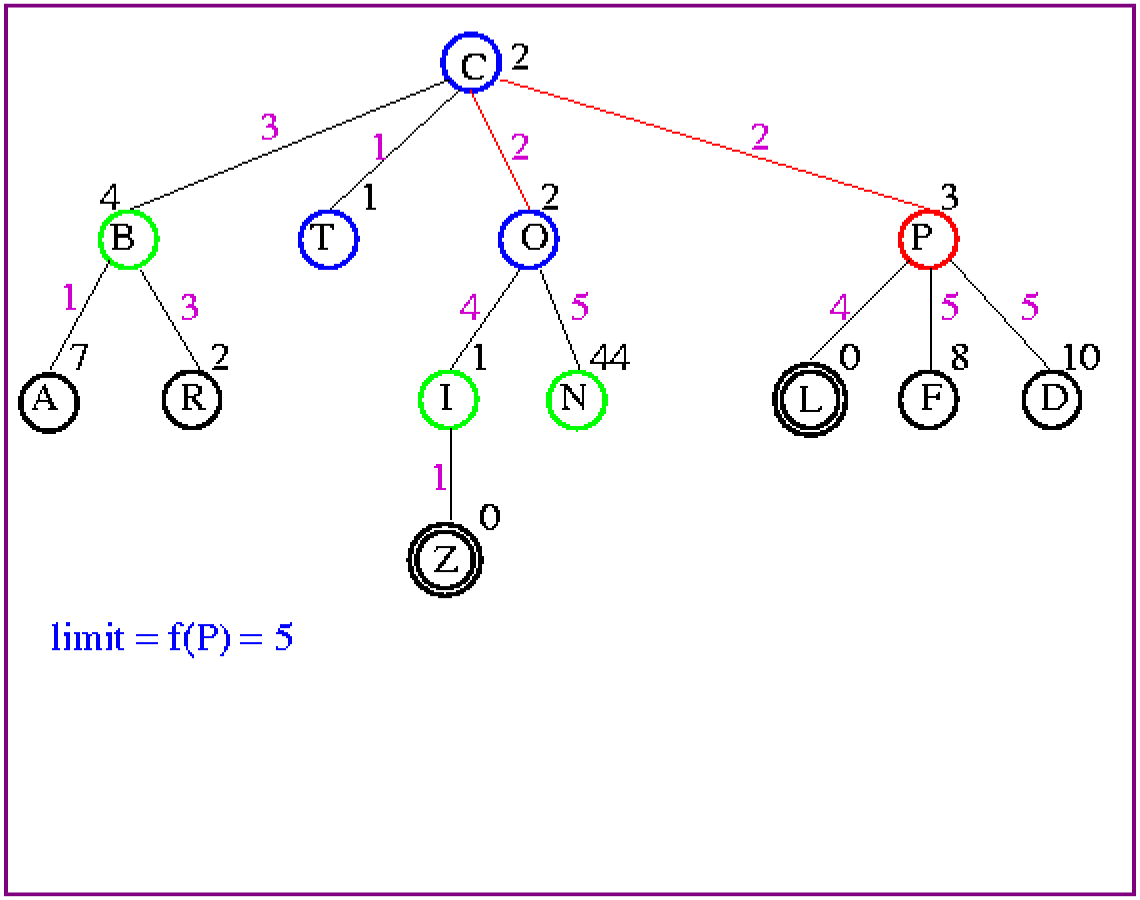
Example

Click mouse to advance to next frame.



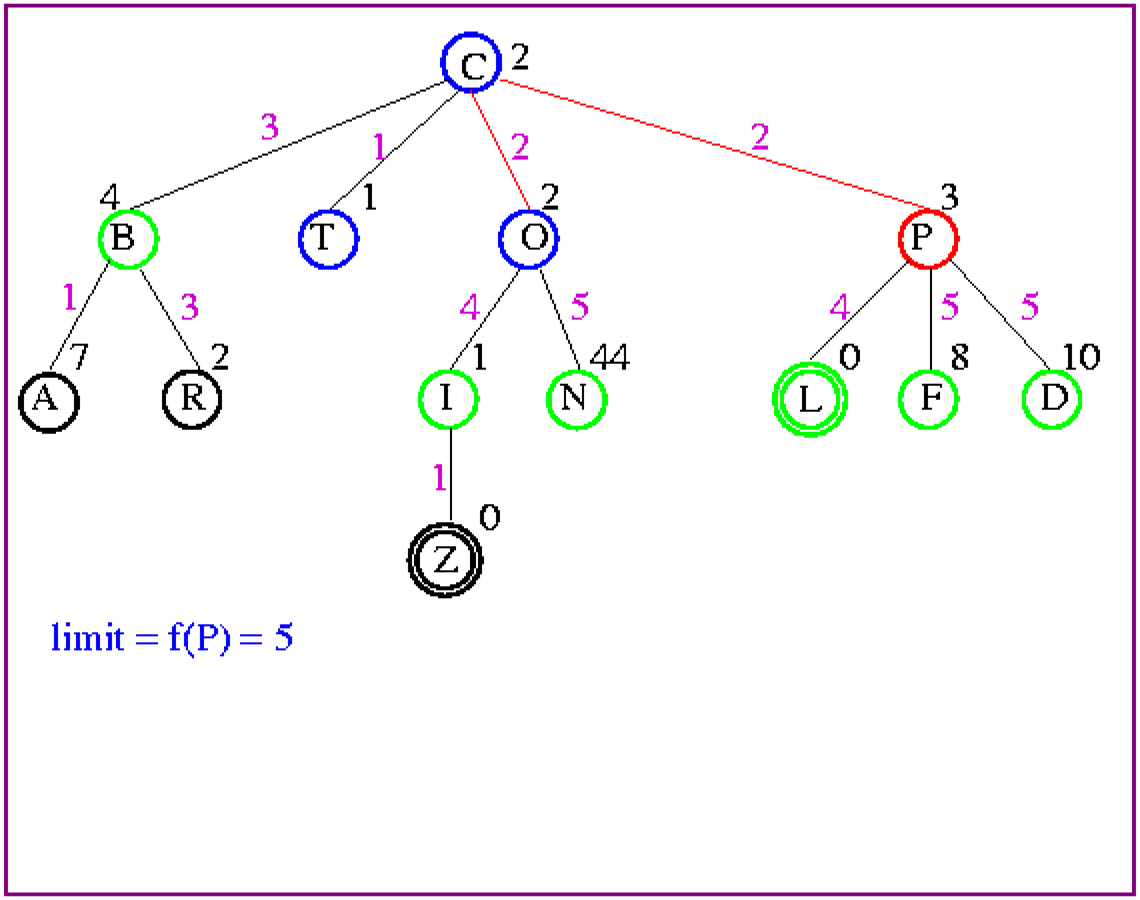
Example

Click mouse to advance to next frame.



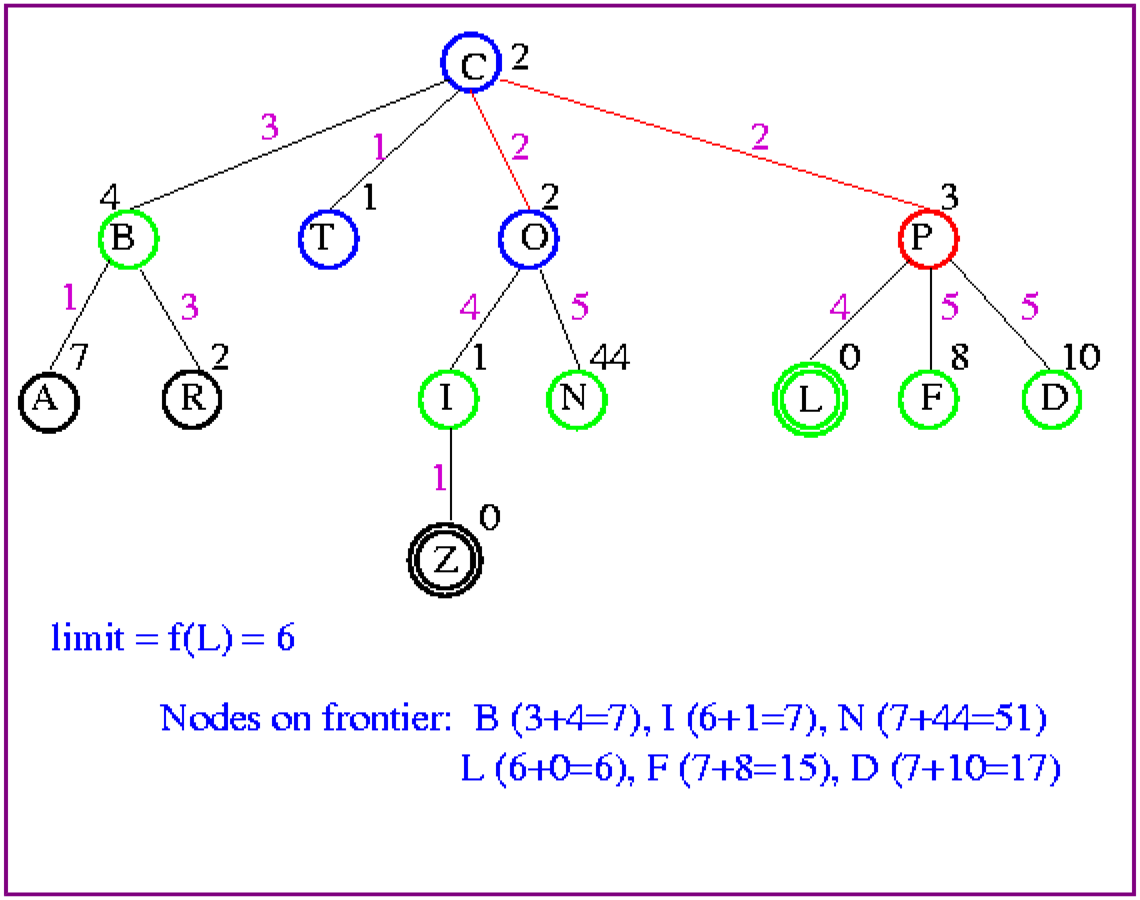
Example

Click mouse to advance to next frame.



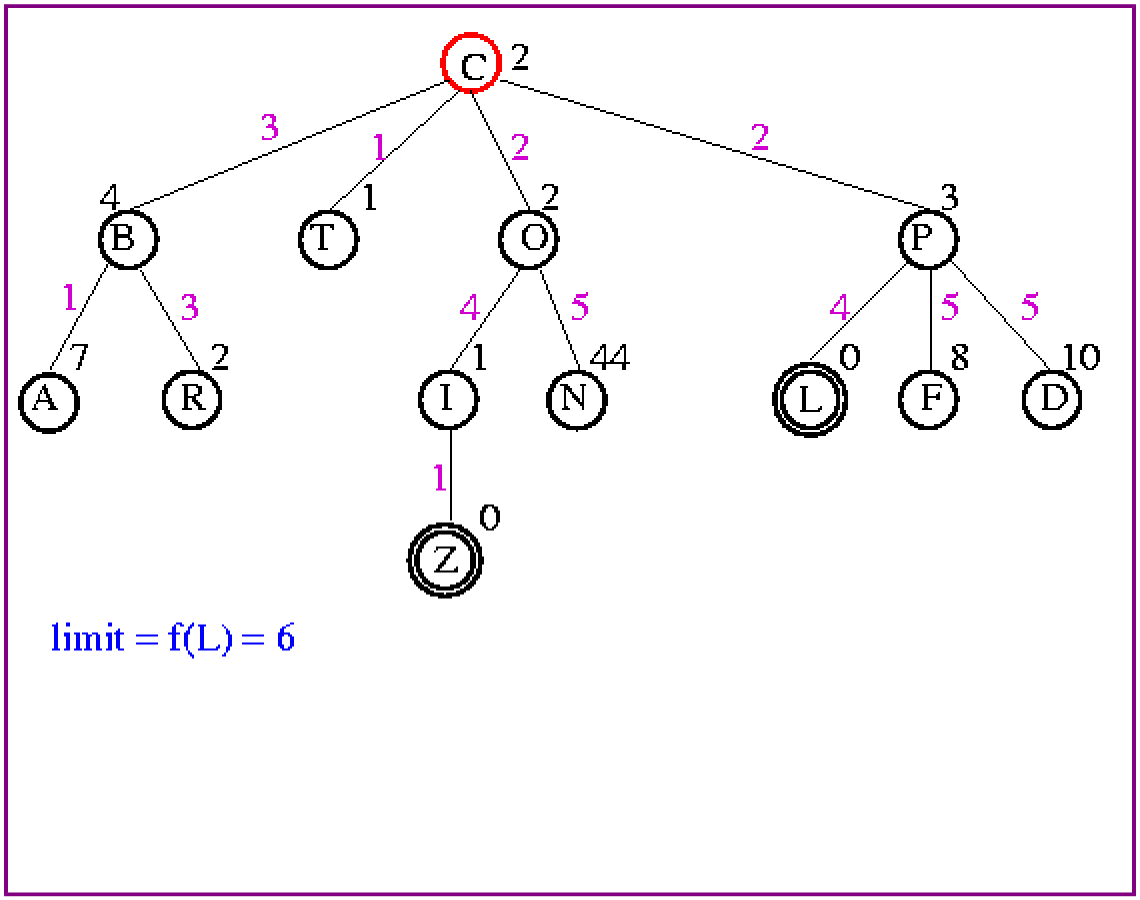
Example

Click mouse to advance to next frame.



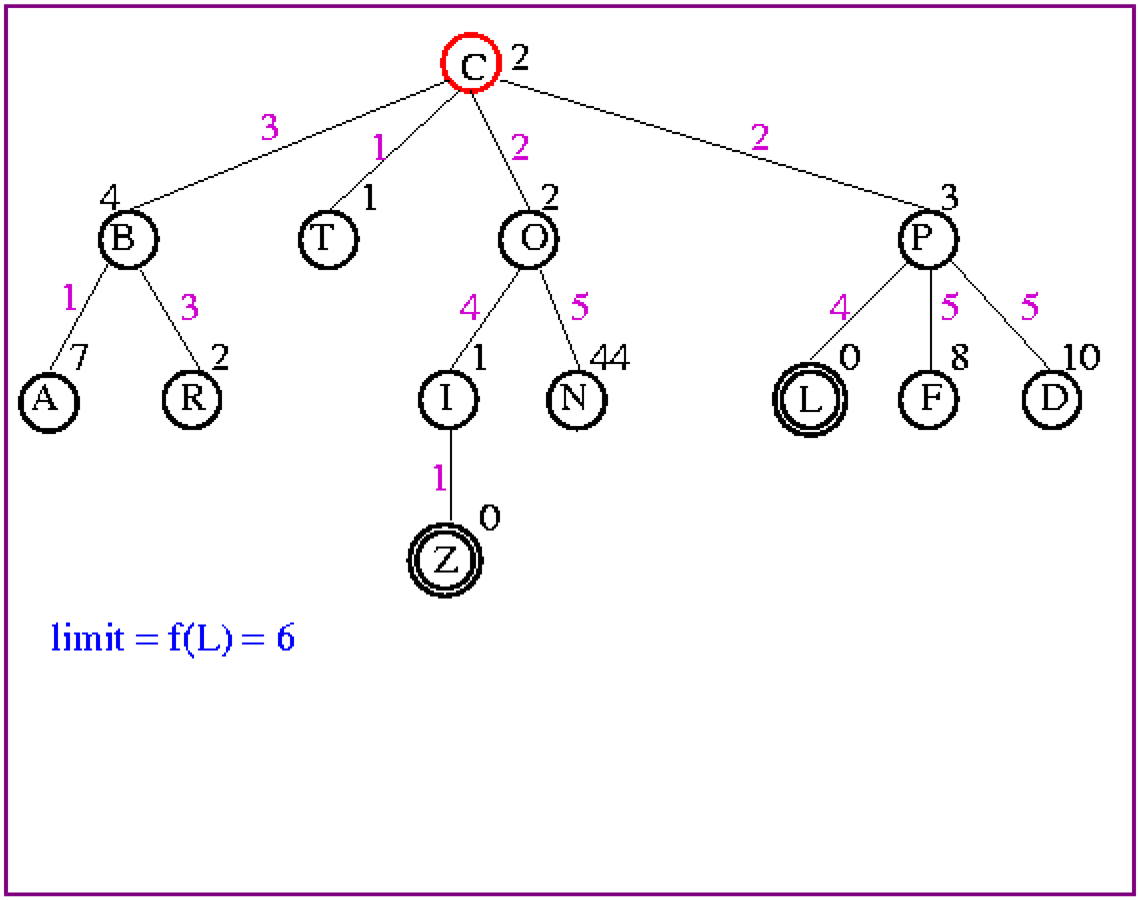
Example

Click mouse to advance to next frame.



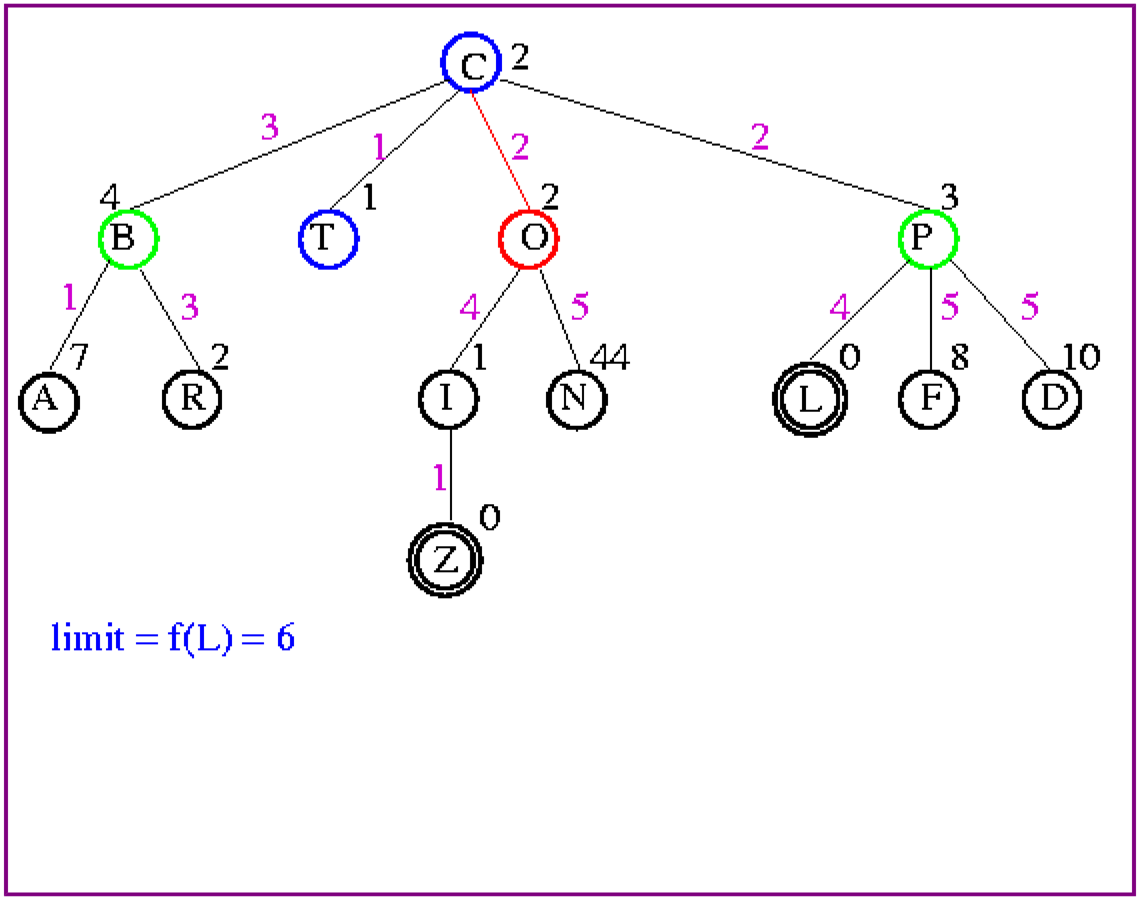
Example

Click mouse to advance to next frame.



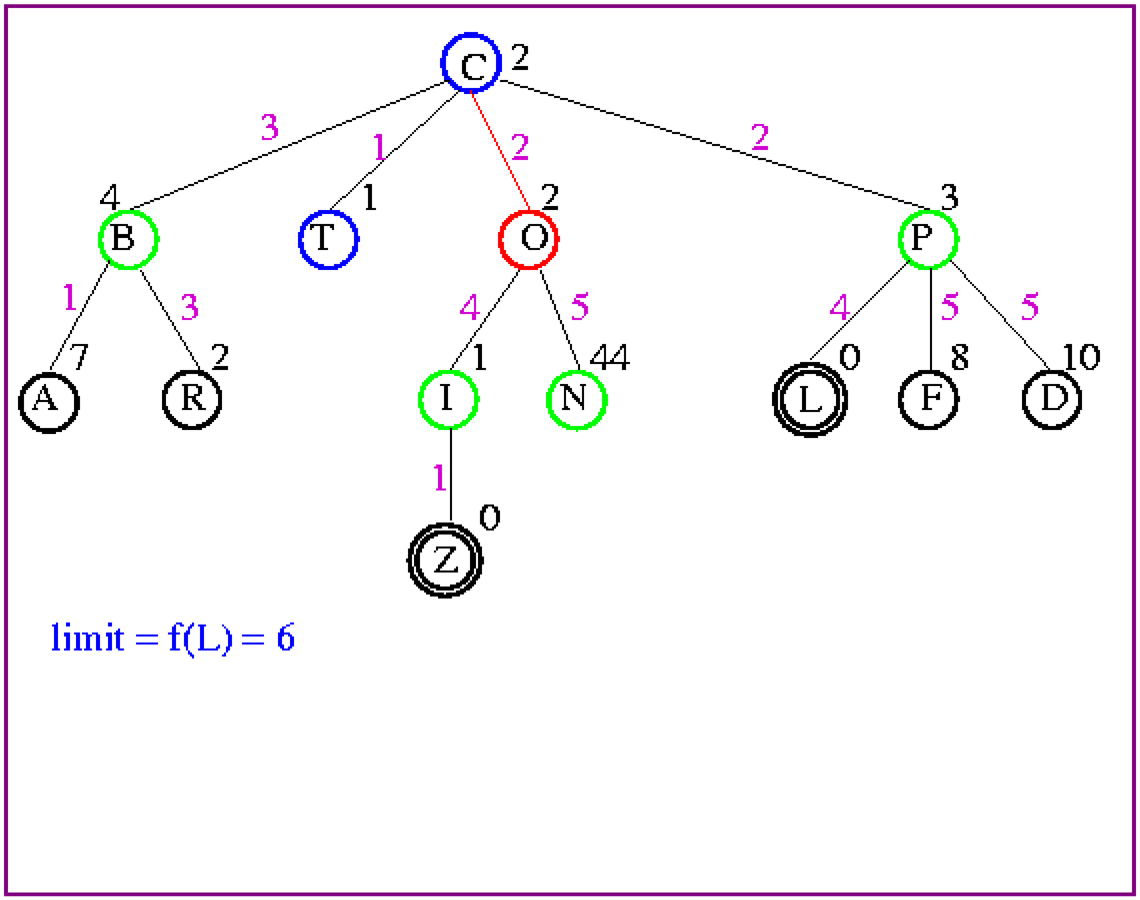
Example

Click mouse to advance to next frame.



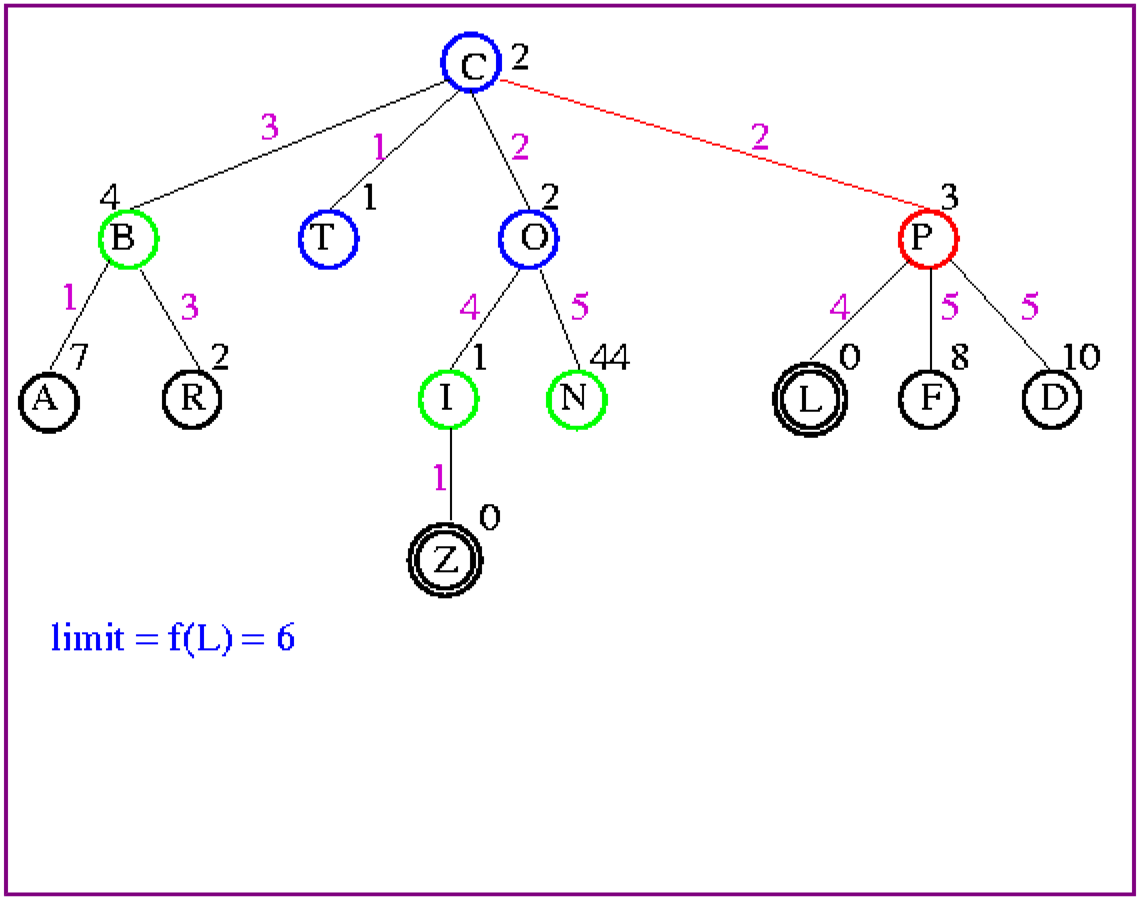
Example

Click mouse to advance to next frame.



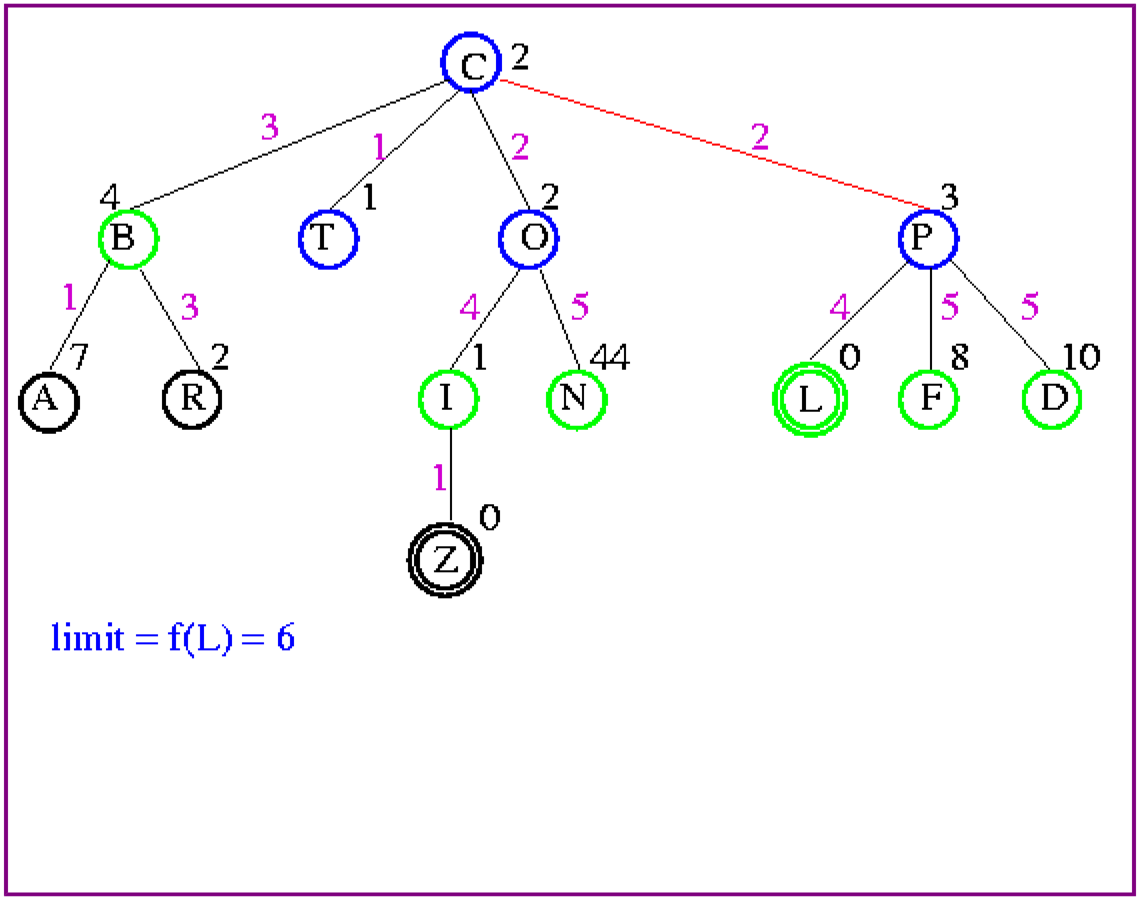
Example

Click mouse to advance to next frame.



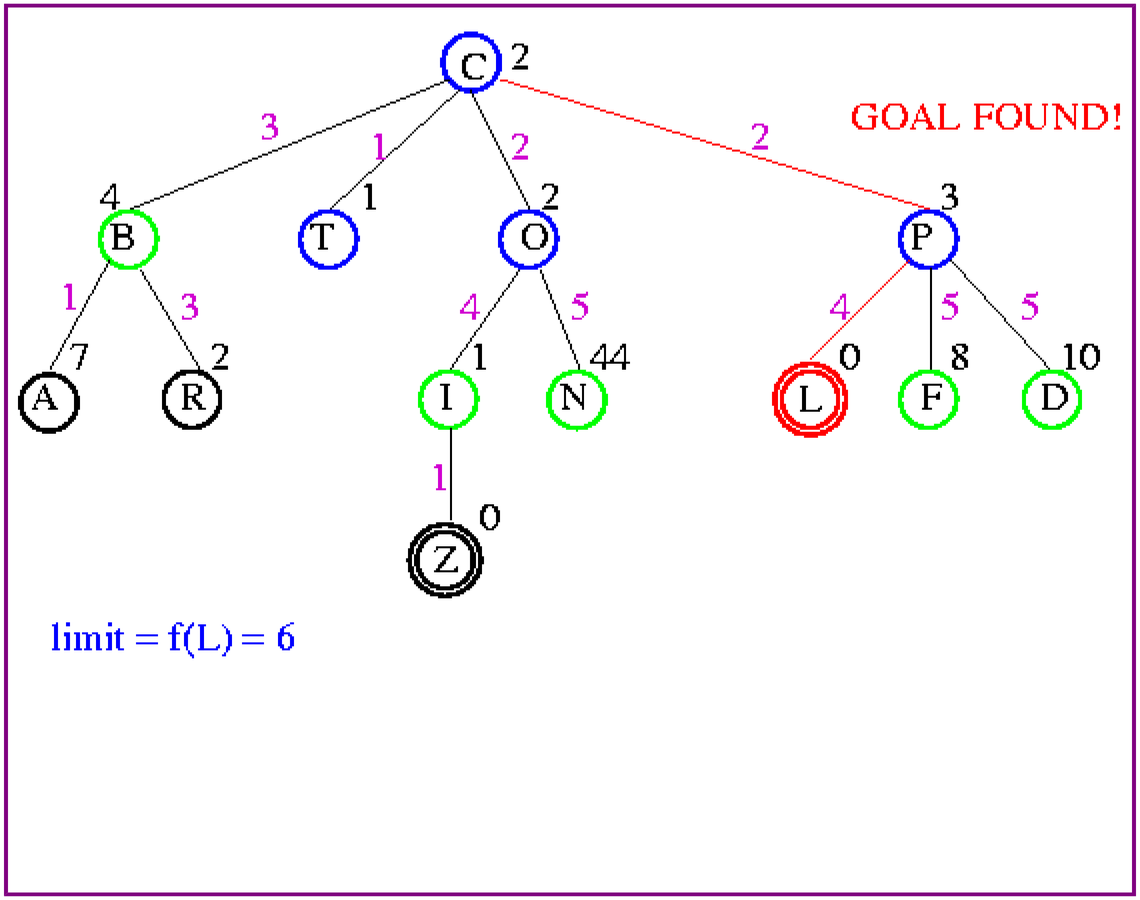
Example

Click mouse to advance to next frame.

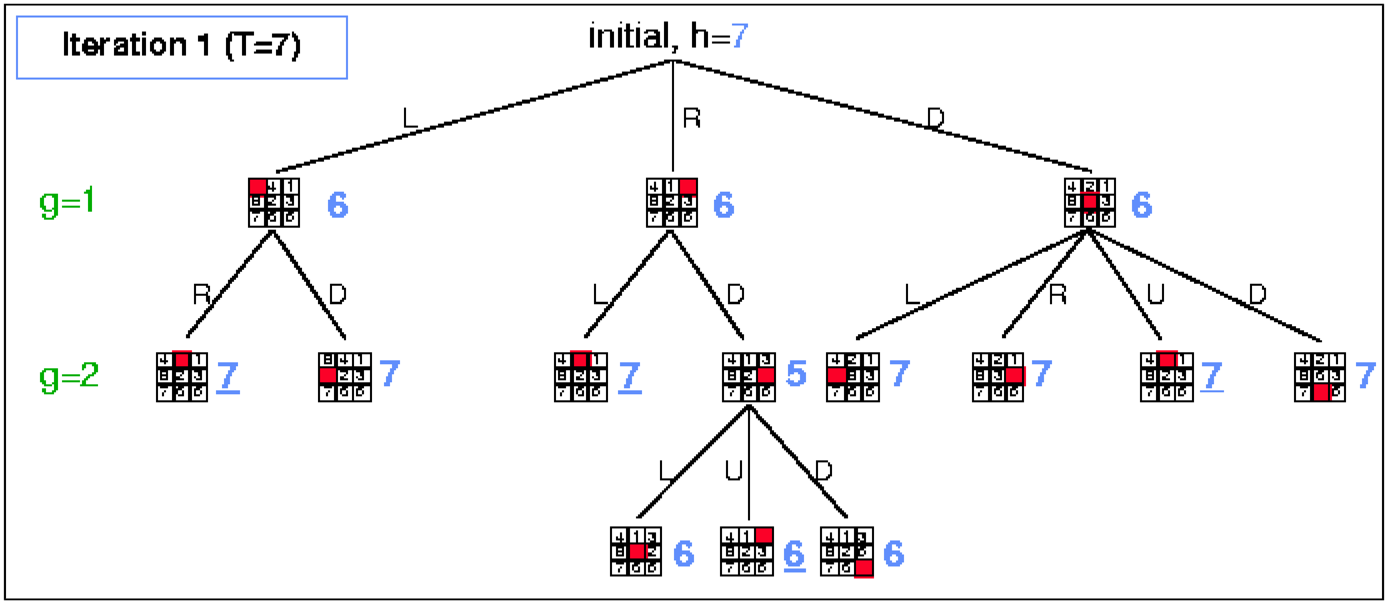


Example

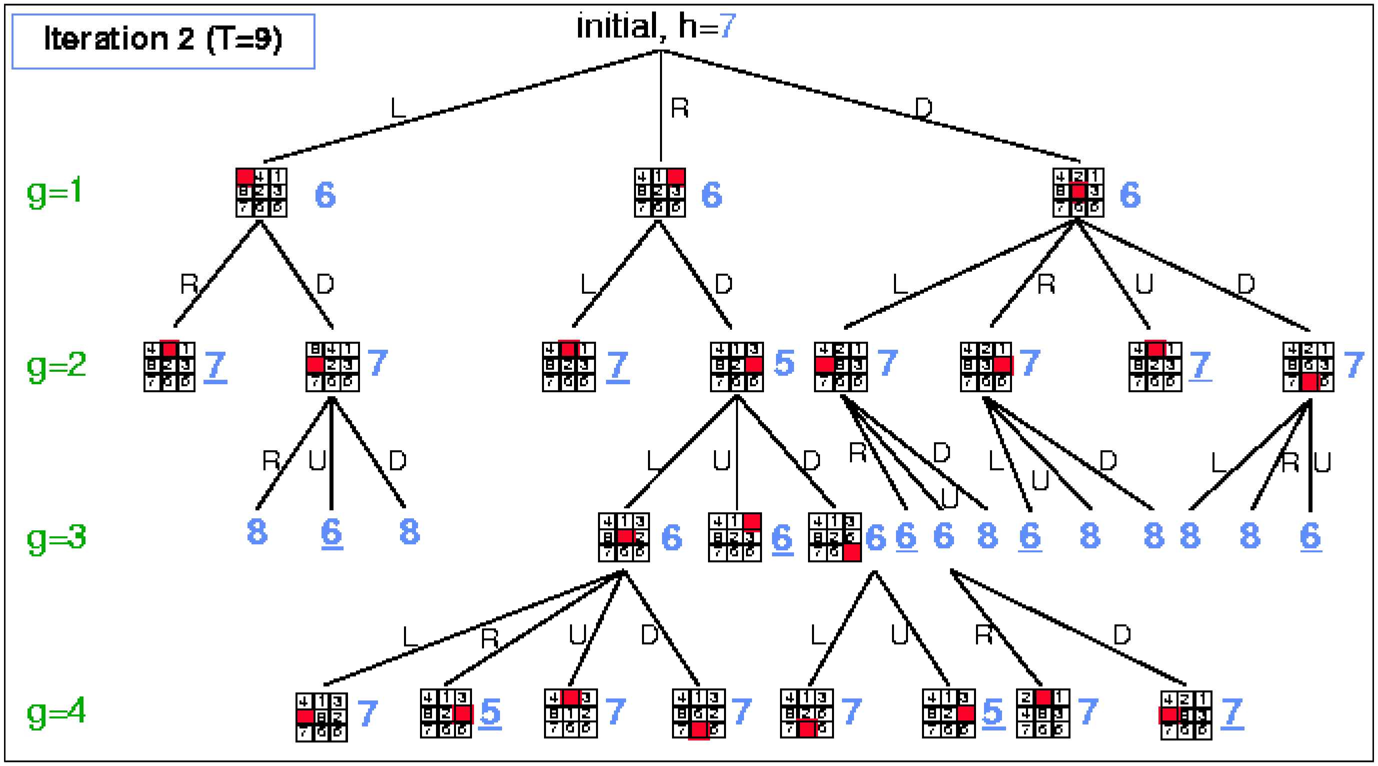
Click mouse to advance to next frame.



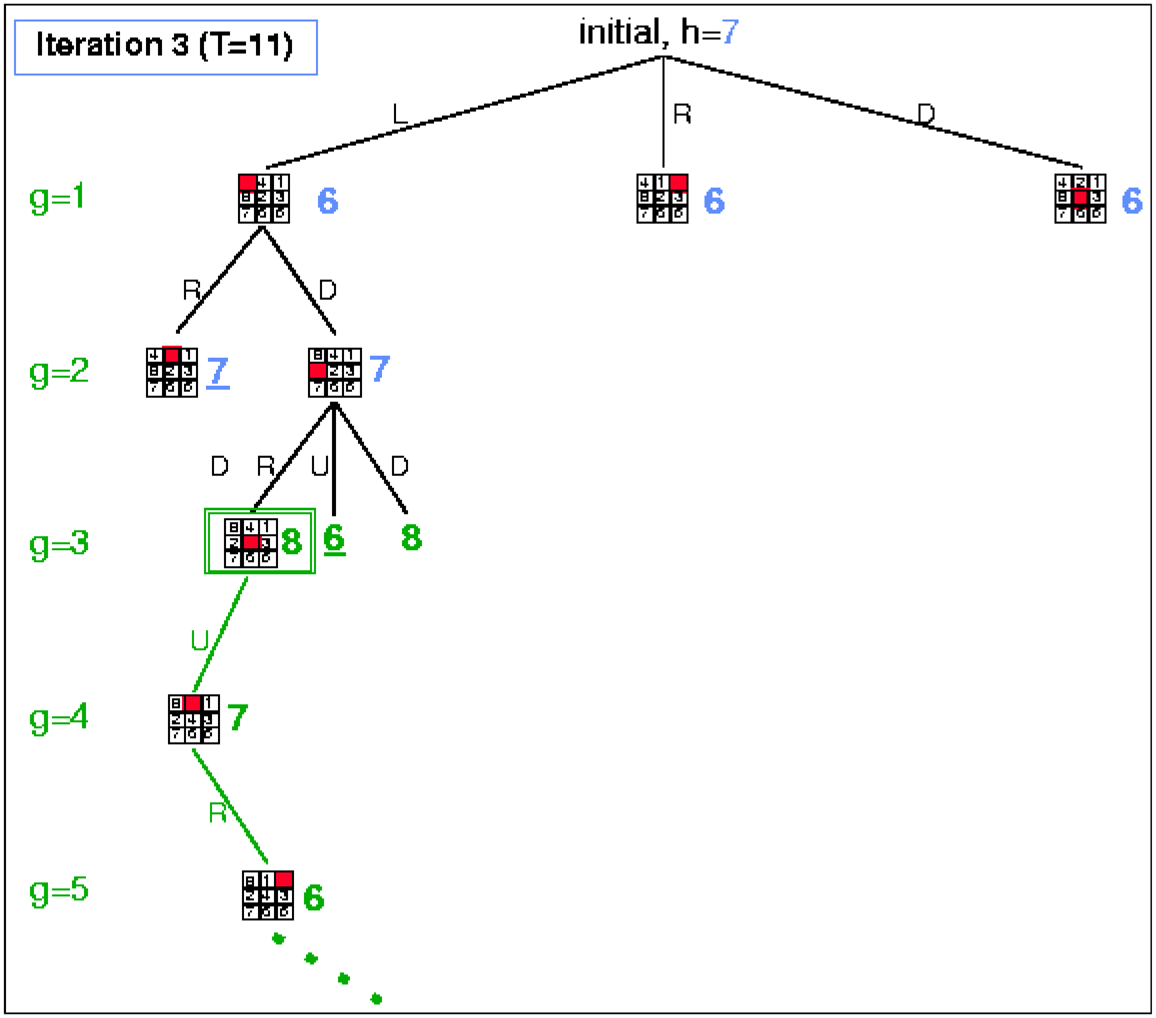
Eight Puzzle Example



Eight Puzzle Example



Eight Puzzle Example



Analysis

Some redundant search, but small amount compared to work done on last iteration

Dangerous if f values are very close

If threshold = 21.1 and next value is 21.2, probably only include 1 new node each iteration

$$\text{Time: } O(b^m) \text{ Space: } O(m)$$

SMA* search can be used to remember some nodes from one iteration to the next.

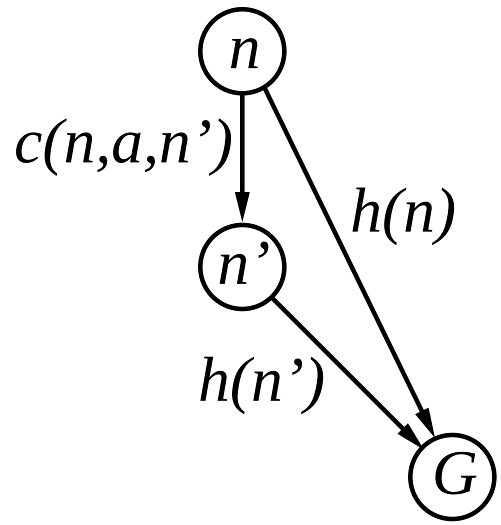
Proof of lemma: Consistency

A heuristic is consistent if

$$h(n) \le c(n,a,n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &= f(n) \end{aligned}$$

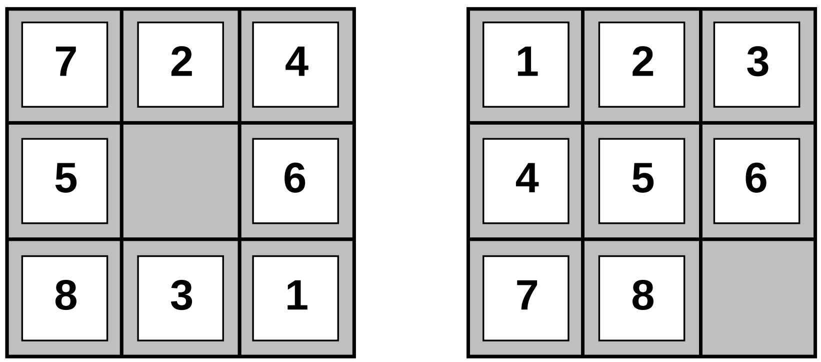


I.e., $f(n)$ is nondecreasing along any path.

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



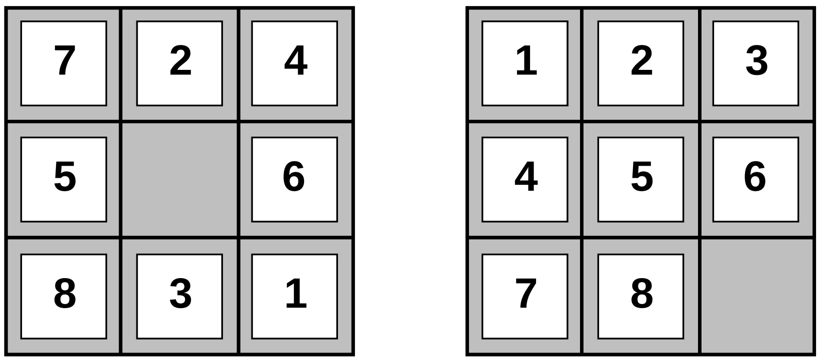
Start State Goal State

$h_1(S) =$??
$h_2(S) =$??

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



Start State **Goal State**

$h_1(S)$ =?? 6
$h_2(S)$ =?? 4+0+3+3+1+0+2+1 = 14

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 dominates h_1 and is better for search

Typical search costs:

$d = 14$ IDS = 3,473,941 nodes
A$^*(h_1)$ = 539 nodes
A$^*(h_2)$ = 113 nodes
$d = 24$ IDS ≈ 54,000,000,000 nodes
A$^*(h_1)$ = 39,135 nodes
A$^*(h_2)$ = 1,641 nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Relaxed problems

Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem

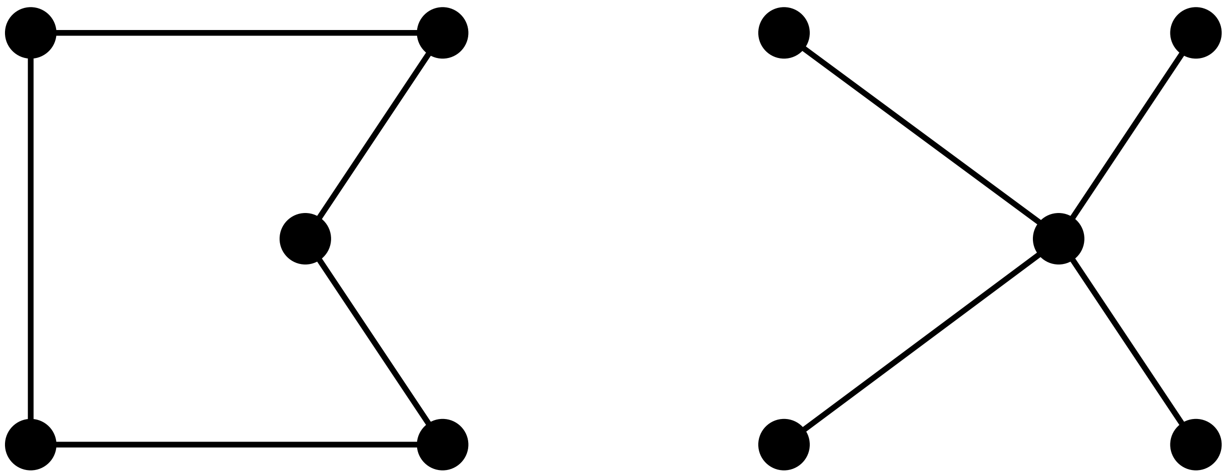
If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ gives the shortest solution

If the rules are relaxed so that a tile can move to **any adjacent square**, then $h_2(n)$ gives the shortest solution

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest h
– incomplete and not always optimal

A* search expands lowest $g + h$
– complete and optimal
– also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems