CSC 442: ARTIFICIAL INTELLIGENCE

LEC 05: ADVERSARIAL SEARCH

ANNOUNCEMENTS

- First quiz is *today*, but it's online via so you have until 1159PM tonight.
- Open notes, but take it alone. The quiz is meant to help you gauge your own understanding and give formative feedback.
- First project is up on blackboard – let's discuss after the main lecture.

REVIEW: SEARCH ALGORITHMS

- Uninformed search: BFS, DFS, IDS, UCS
- Informed search: A*
- A* expands fewest number of nodes possible while retaining optimality of solution if heuristic is consistent.

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

AIMA Figure 3.14, p84

```
function A-STAR-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

Insert with cost g(n)+h(n).

AIMA Figure 3.14, p84

HEURISTICS

- A **heuristic** - $h(n)$ - is a special kind of function which estimates the cost of an optimal solution through a particular node n.

- A heuristic is **admissible** if it never over-estimates the true cost of a solution.

- A heuristic is **consistent** if it obeys the triangle inequality:

  - $h(n) \leq c(n, a, n') + h(n')$

Evaluation function

$$f(n) = h(n)$$

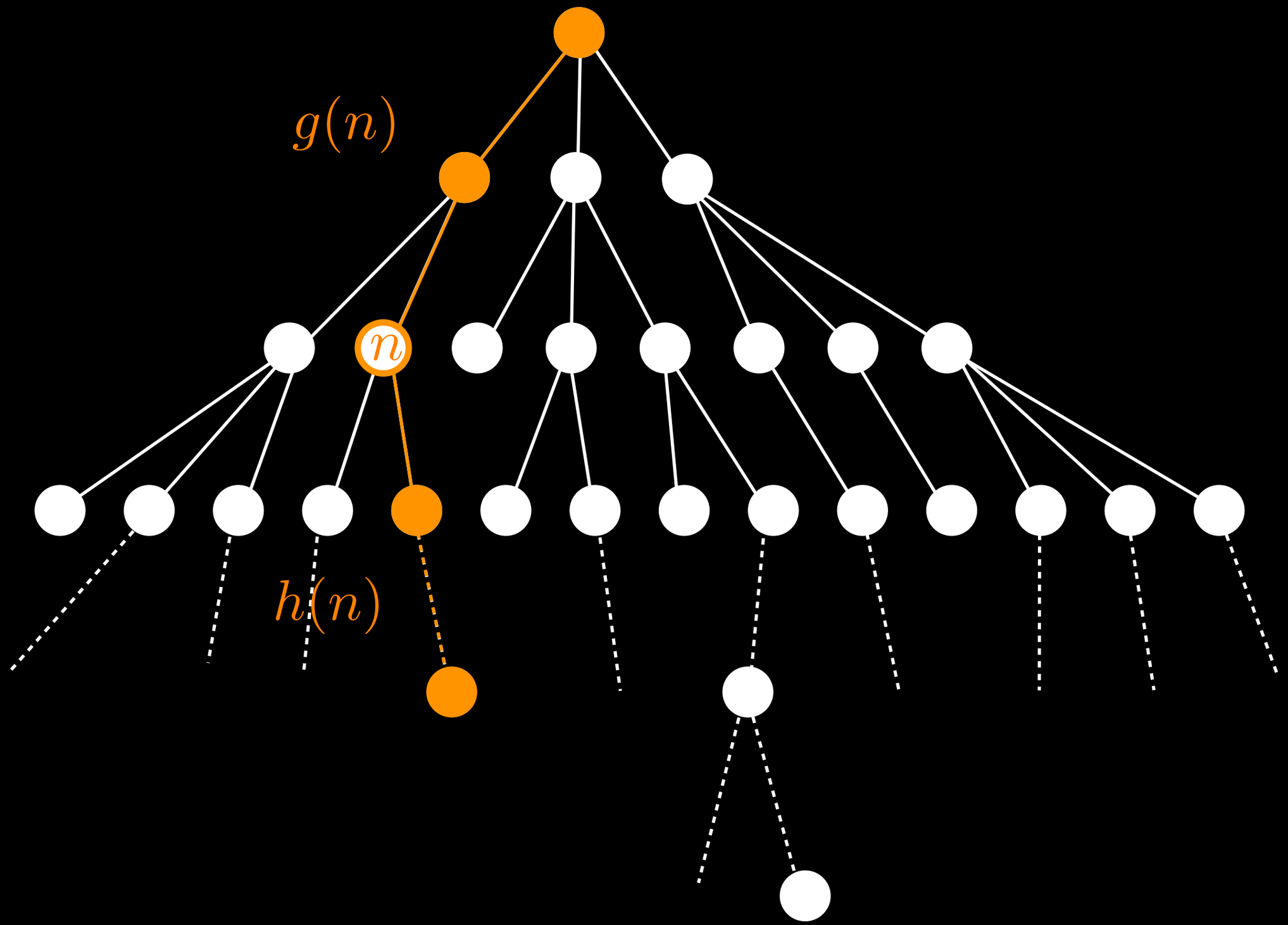
Estimated cost of cheapest path from n to a goal node

Evaluation function

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

True cost of path from start node to node n

Estimated cost of cheapest path from n to a goal node


$g(n)$
n
$h(n)$

Evaluation function

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

$\equiv$ Estimated cost of cheapest solution through n

Frontier: A

0+366
=366

$$f(n) = g(n) + h_{SLD}(n)$$

A

S

T

Z

Oradea
71
Zerind
75
Arad
151
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
86
Bucharest
90
Giurgiu
Eforie

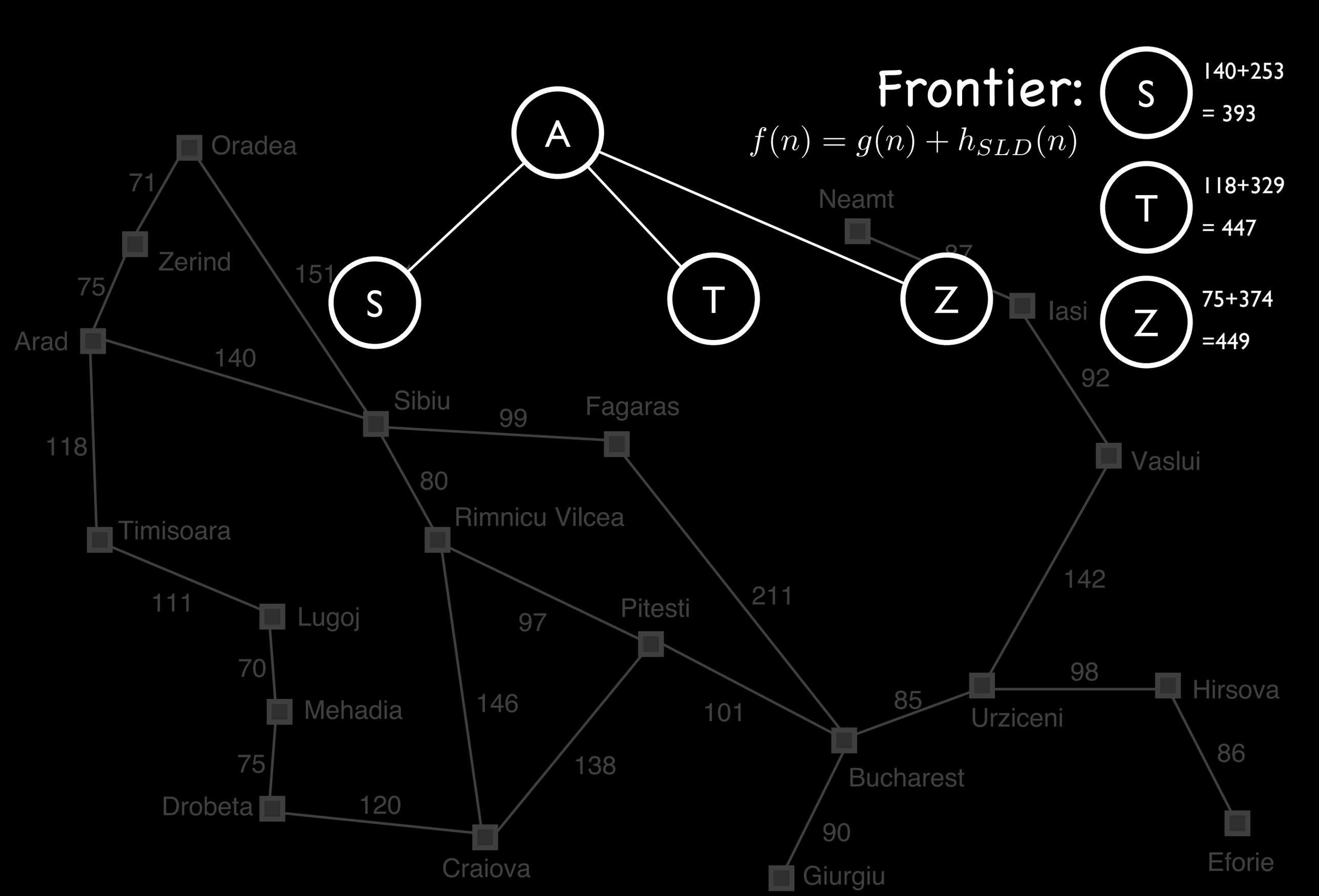
Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$

S — 140+253 = 393

T — 118+329 = 447

Z — 75+374 =449



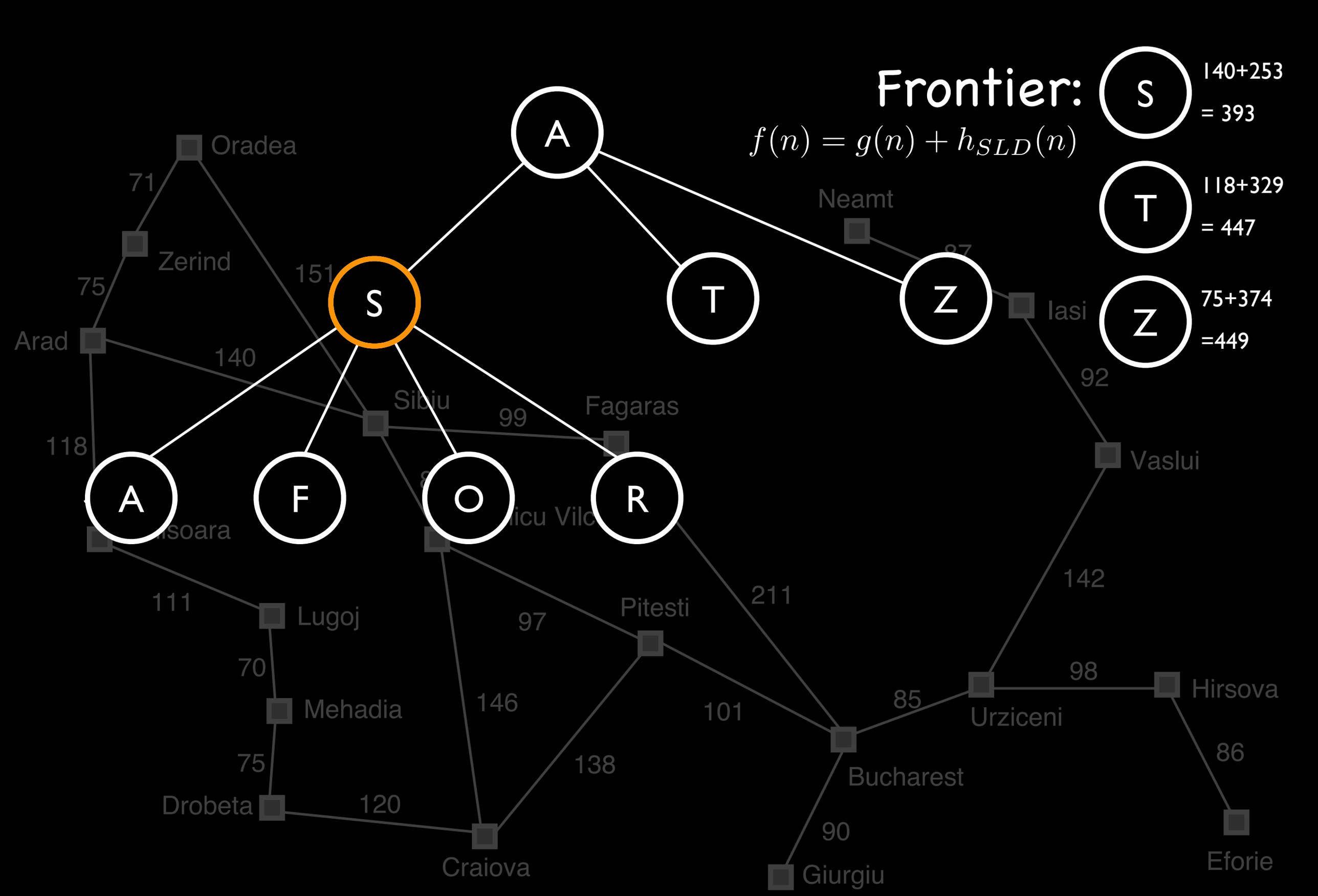
Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$

S 140+253 = 393

T 118+329 = 447

Z 75+374 =449



Frontier:

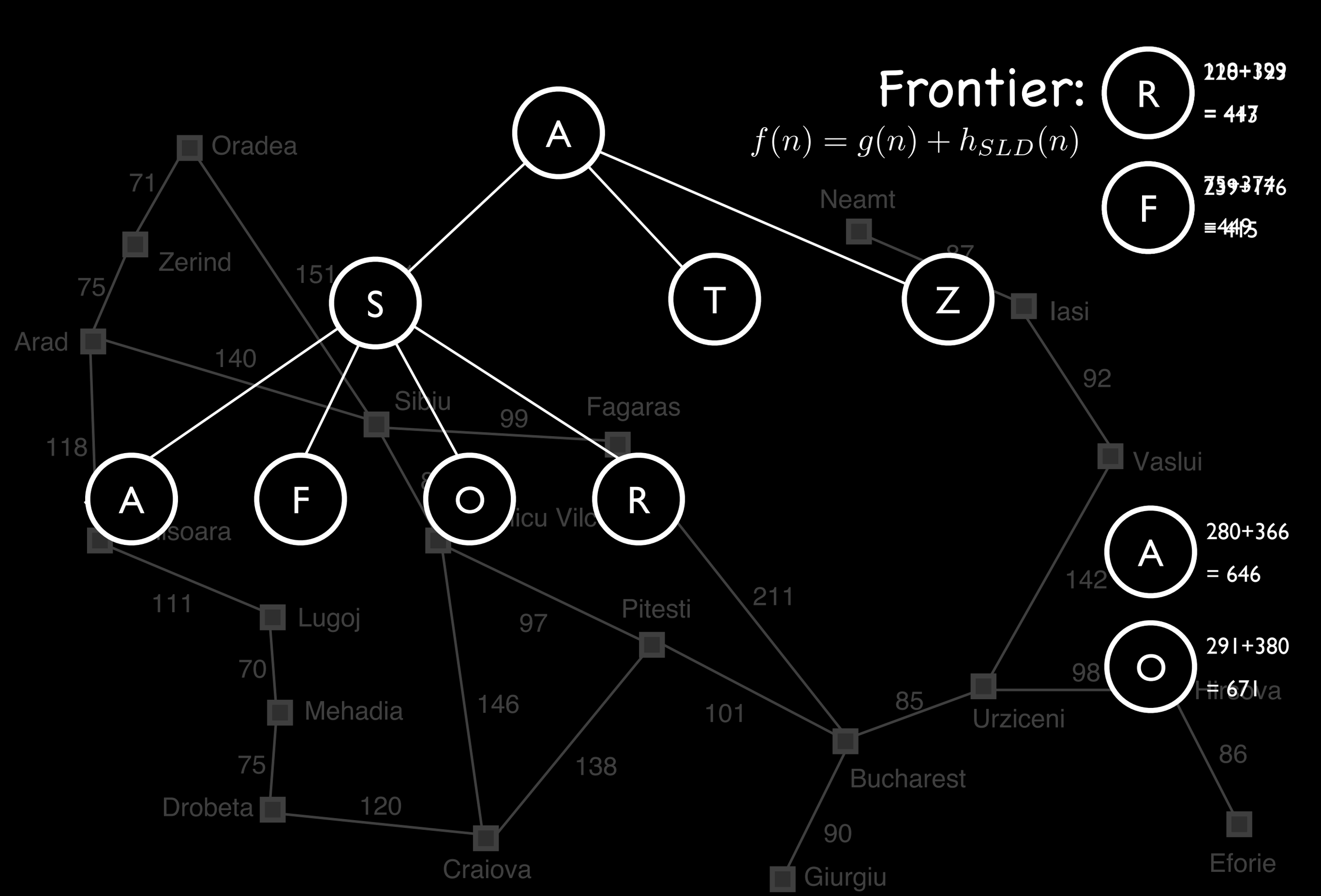
$$f(n) = g(n) + h_{SLD}(n)$$

R 220+329 = 447

F 239+176 = 415

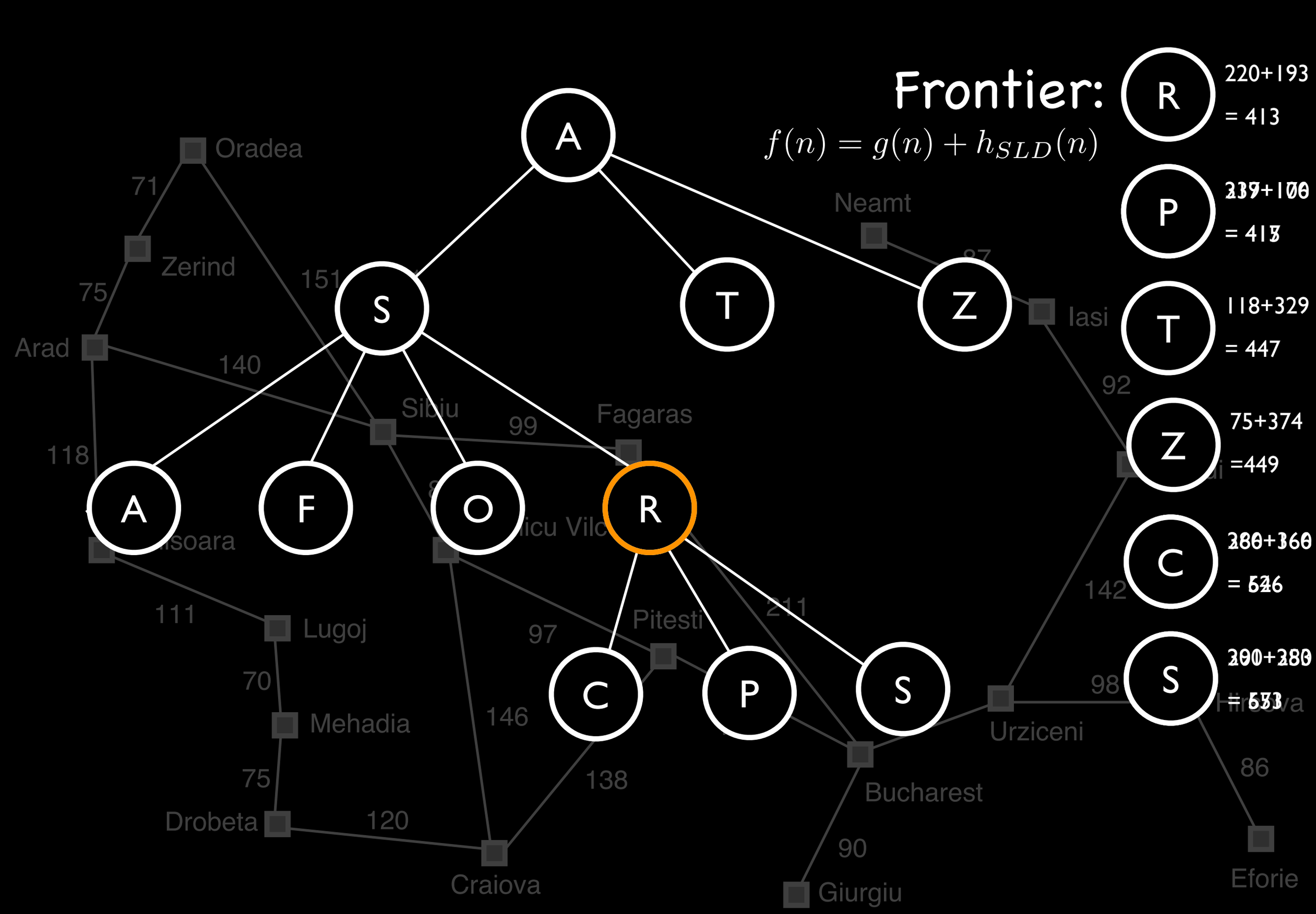
A 280+366 = 646

O 291+380 = 671



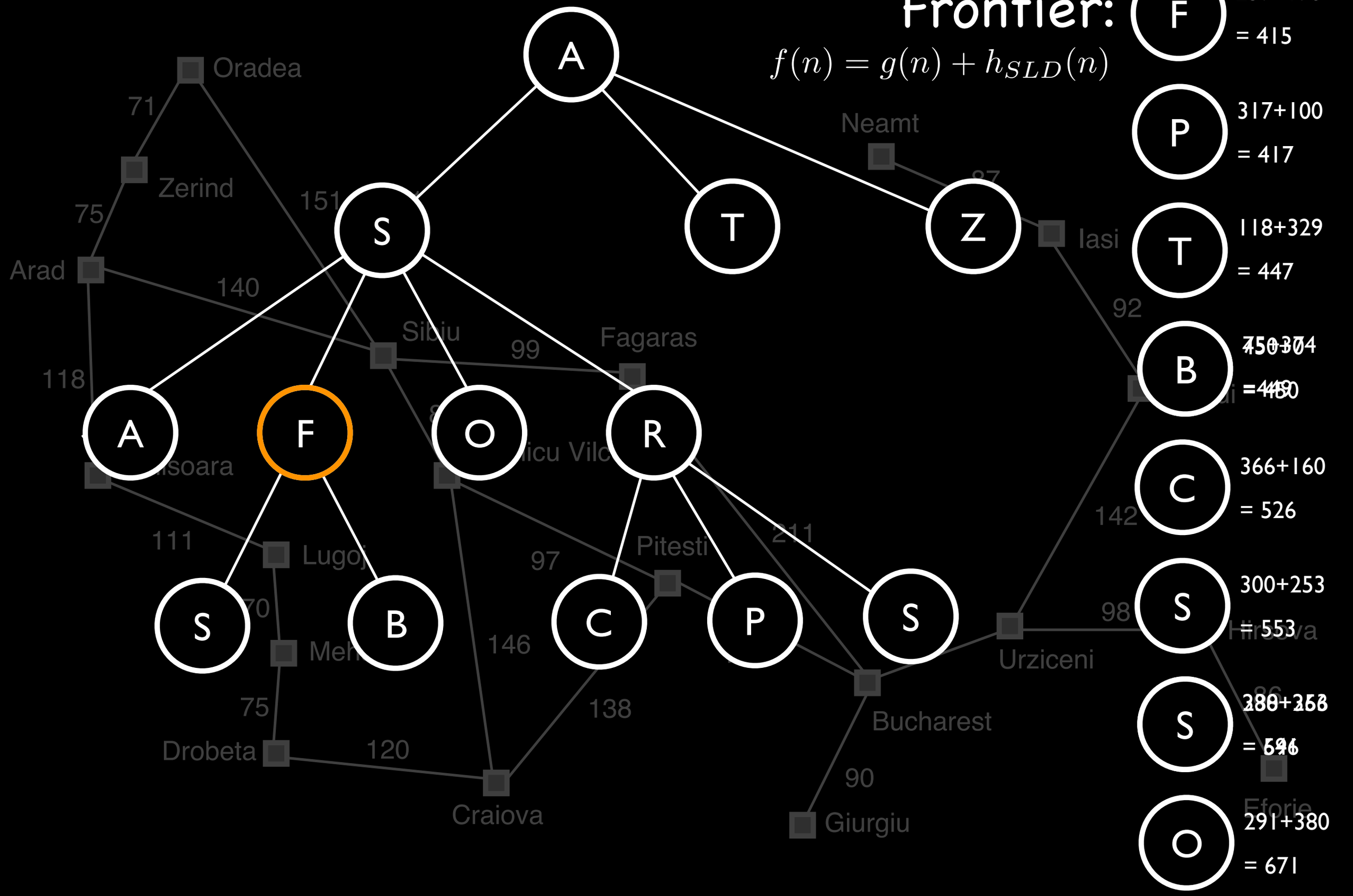
Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$



Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$



F 239+176 = 415

P 317+100 = 417

T 118+329 = 447

B ~~450~~ 418 = ~~449~~ 417

C 366+160 = 526

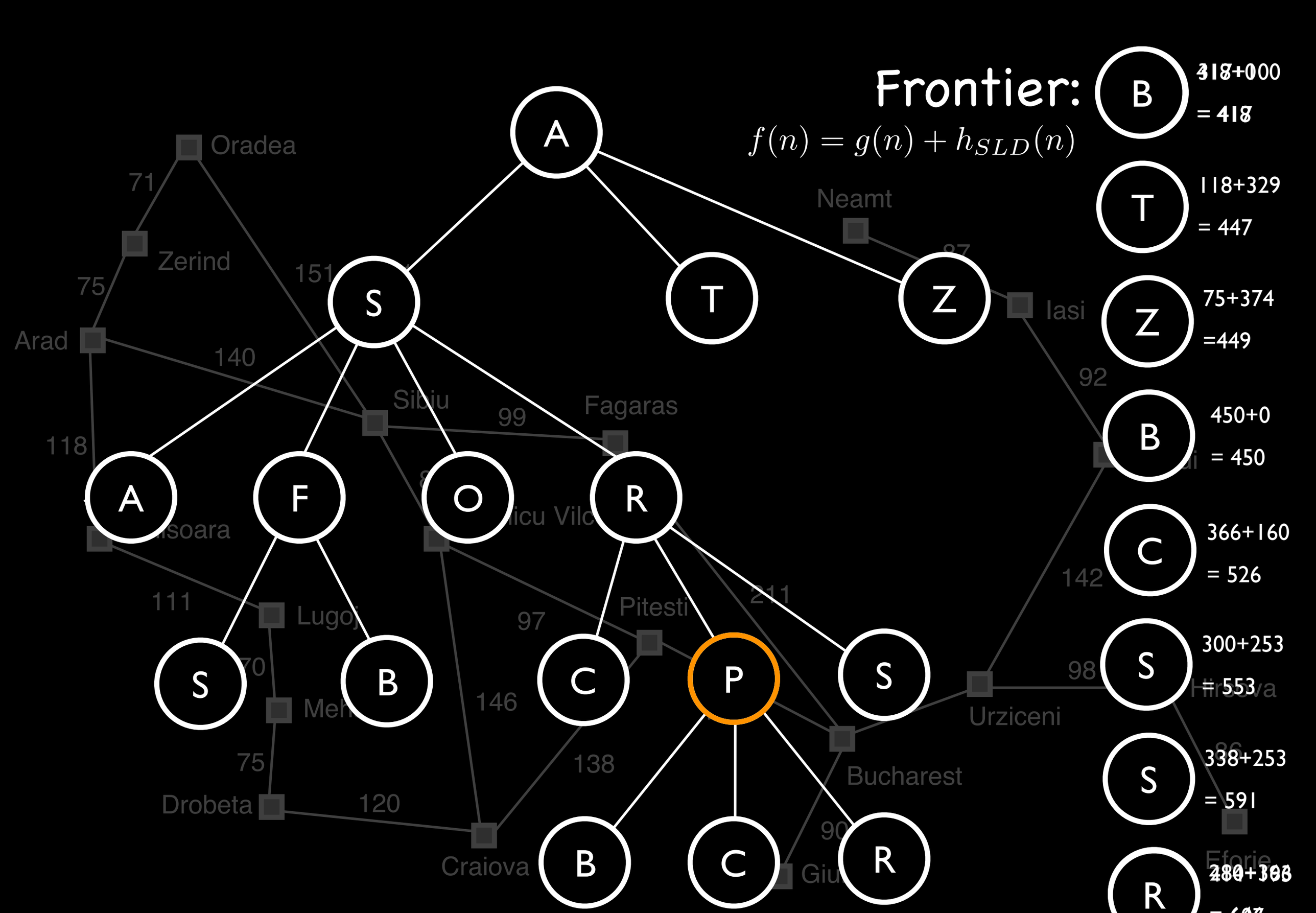
S 300+253 = 553

S ~~280+253~~ = ~~533~~ 531

O 291+380 = 671

Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$



B — 418+000 = 418

T — 118+329 = 447

Z — 75+374 =449

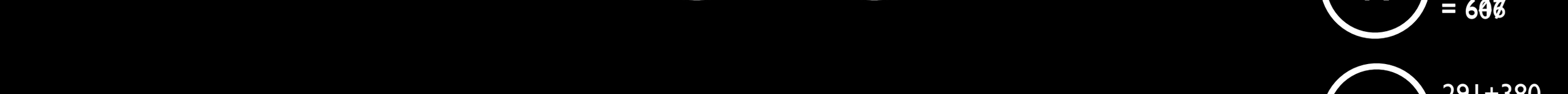
B — 450+0 = 450

C — 366+160 = 526

S — 300+253 = 553

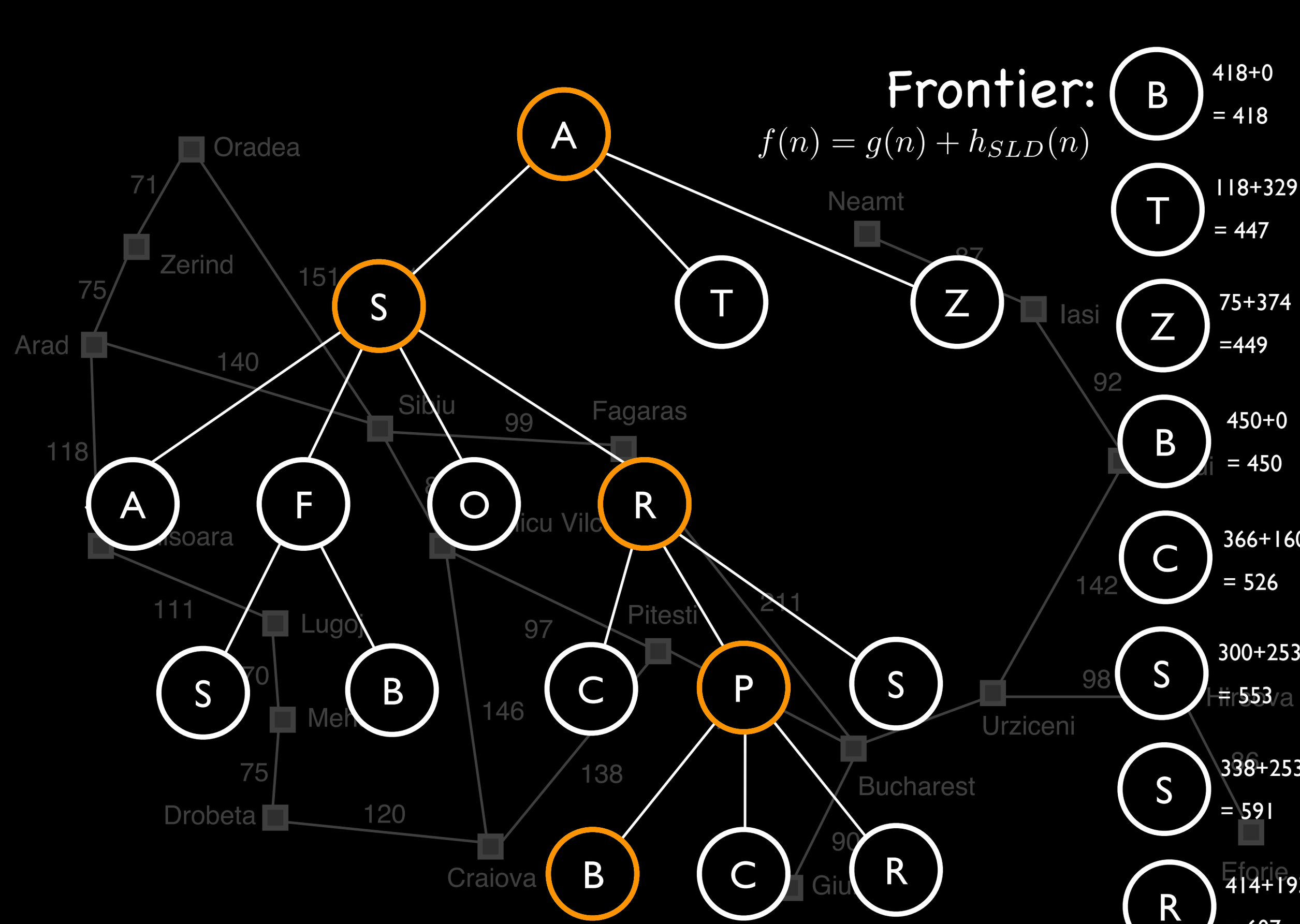
S — 338+253 = 591

R — 480+303 = 607



Frontier:

$$f(n) = g(n) + h_{SLD}(n)$$



B 418+0 = 418

T 118+329 = 447

Z 75+374 =449

B 450+0 = 450

C 366+160 = 526

S 300+253 = 553

S 338+253 = 591

R 414+193 = 607

Evaluation function

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

True cost of path from start node to node *n*

Estimated cost of cheapest path from *n* to a goal node

If $h(n)$
is admissible

Completeness

Optimality

Never overestimates the true cost
of a solution

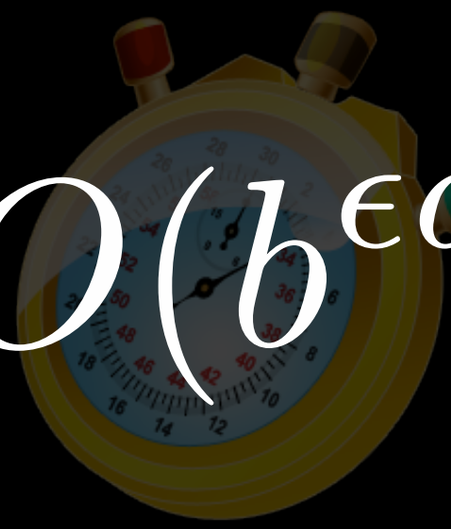
$$f(n) = g(n) + h_{SLD}(n)$$

A* SEARCH

If $h(n)$ is admissible

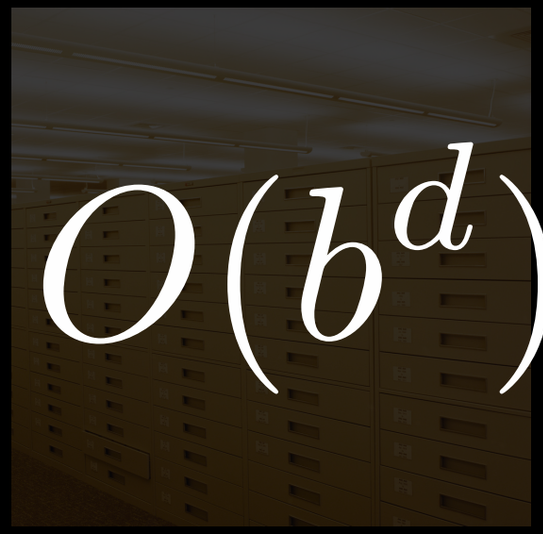
Completeness

Optimality



$O(b^{\epsilon d})$

$O(b^{d})$



Time Complexity

Space Complexity

GAMES

- ... are interesting because they are too hard to solve.
- ... require the ability to make some decision even when calculating the optimal decision is infeasible.
- ... require fast algorithms because of time limits

GAME TERMINOLOGY

- A game is **zero-sum** if one person's loss means another person's game. Wealth is neither created or destroyed.
- We want maximize our score. In a zero-sum game, such as checkers or chess, an optimal opponent would try to minimize our score, so we choose the action which gives us the highest guaranteed payout.
- This is called the **minimax strategy**.

GAME: A 6-TUPLE

- S0 – The initial state, which specifies how the game is set up
- PLAYER(s) – a function to define which player has the move in a state.
- ACTIONS(s) – the set of legal moves in a state.
- RESULT(s, a) – the transition model, defines the result of a move.
- TERMINAL-TEST(s) – true when the game is over and false otherwise.
- UTILITY(s, p) – a utility function, defines the final numeric value for a game that ends in terminal state s for a player p.

STATE-SPACE GAMES

- Game: Chess, Checkers, Tic-Tac-Toe, Mancala
- S0 – initial board (need some way to represent), blank board, 4x6 board, player
- PLAYER(s) – integers? strings? boolean? self (pov of algorithm) vs opponent?
- ACTIONS(s) – action encodings (integer, string), action set (vector/array, list), available actions (for each piece check all moves, for TTT/Mancala check all locations)
- RESULT(s, a) – copy and return a new state datastructure after the move
- TERMINAL-TEST(s) – checkmate/draw detection, no pieces/stuck,cross or diagonal, no stones for one player
- UTILITY(s, p) – who won?

MINIMAX STRATEGY – CHOOSE THE ACTION WITH HIGHEST MINIMAX VALUE

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

SAMPLE SEARCH SPACES

	Branching Factor	Average Depth
Tic-Tac-Toe	4	9
Checkers	2.8	70
Chess	35	70
Go	250	150
Mancala	3.13	26

Solving Kalah. Irving et al. ICGA 2000

https://en.wikipedia.org/wiki/Game_complexity

A CAREFUL DISTINCTION

- Minimax Value – the value of a state assuming both players play optimally from the state until the game ends.
- Minimax Strategy - the strategy of always choosing an action with greatest minimax value (or randomly among them, if there are several options with maximum value)
- Minimax Algorithm - an algorithmic implementation of the minimax strategy

MINIMAX AND OPTIMALITY

- This definition of optimal play for MAX assumes that MIN also plays optimally—it maximizes the worst-case outcome for MAX.
- Other strategies against suboptimal opponents may do better than the minimax strategy, but these strategies necessarily do worse against optimal opponents.

GAME TREES

- The initial state, actions function, and result function define the **game tree** for the game—a tree where the nodes are game states and the edges are moves.

Suppose we go first.

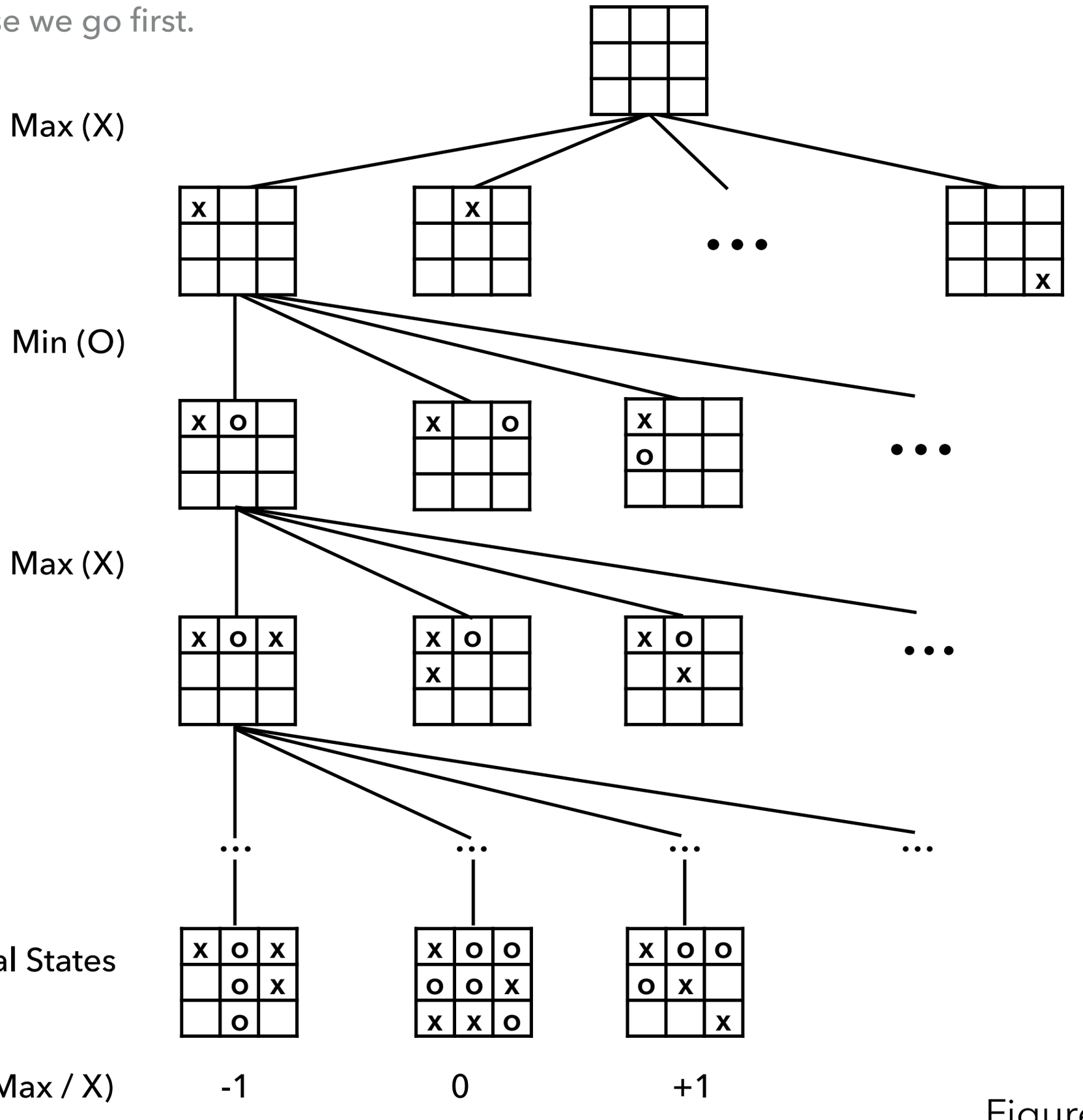
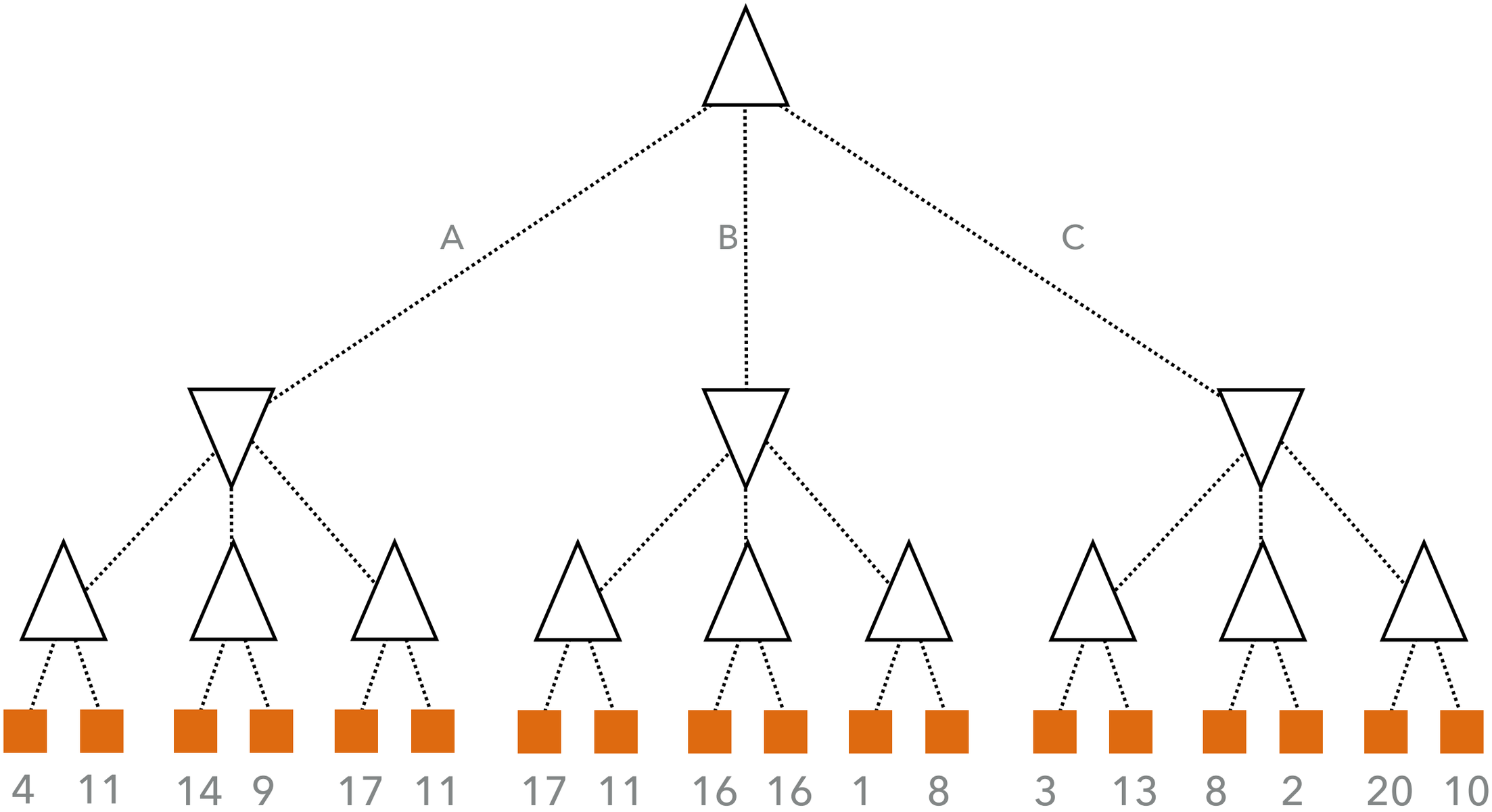
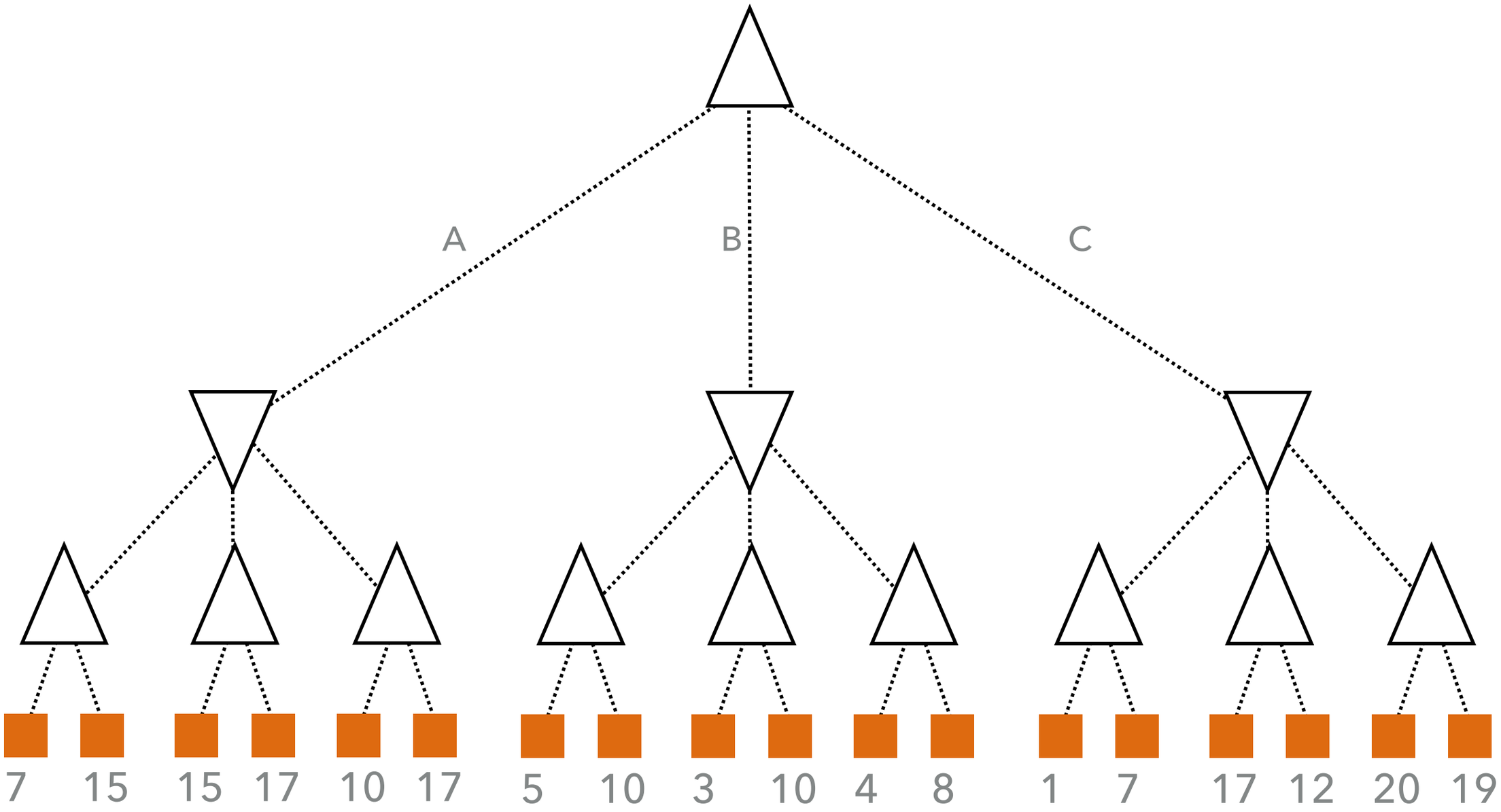


Figure from AIMA, p163


A
B
C
4 11 14 9 17 11 17 11 16 16 1 8 3 13 8 2 20 10


A
B
C
7
15
15
17
10
17
5
10
3
10
4
8
1
7
17
12
20
19

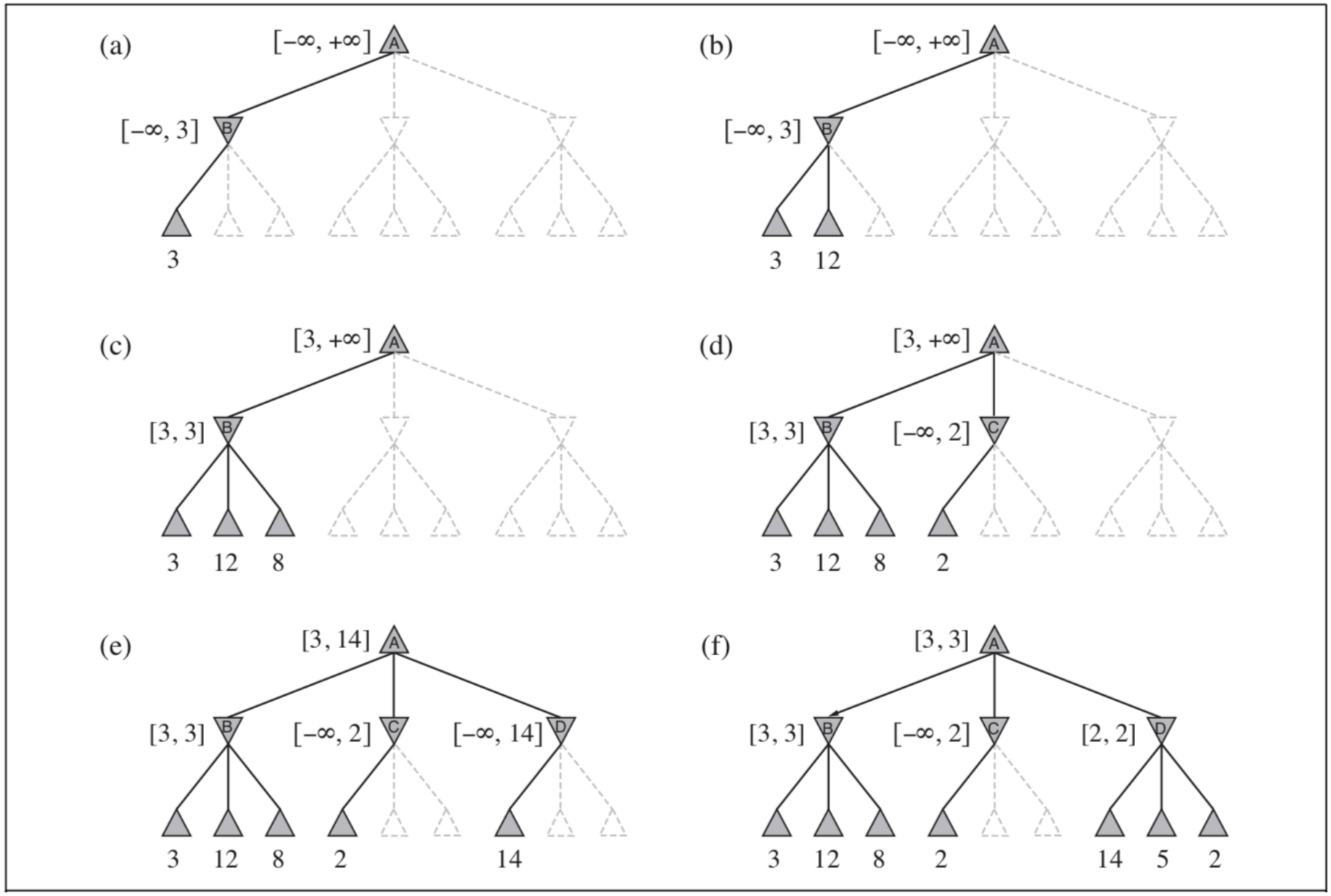
MINIMAX COMPLEXITY

- Minimax performs a complete depth-first exploration of the game tree. Let b be the branching factor, and m be the maximum depth of the tree. Then computational requirements of minimax are:

- Time complexity – $\mathcal{O}(b^m)$  This is a problem.
- Space complexity – $\mathcal{O}(bm)$ or $\mathcal{O}(b)$

- Minimax ✓
- Alpha-Beta Pruning
- Heuristic Minimax
- Expecti-Minimax

- It is possible to compute the correct minimax decision without looking at every node in the game tree.



$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}
$$

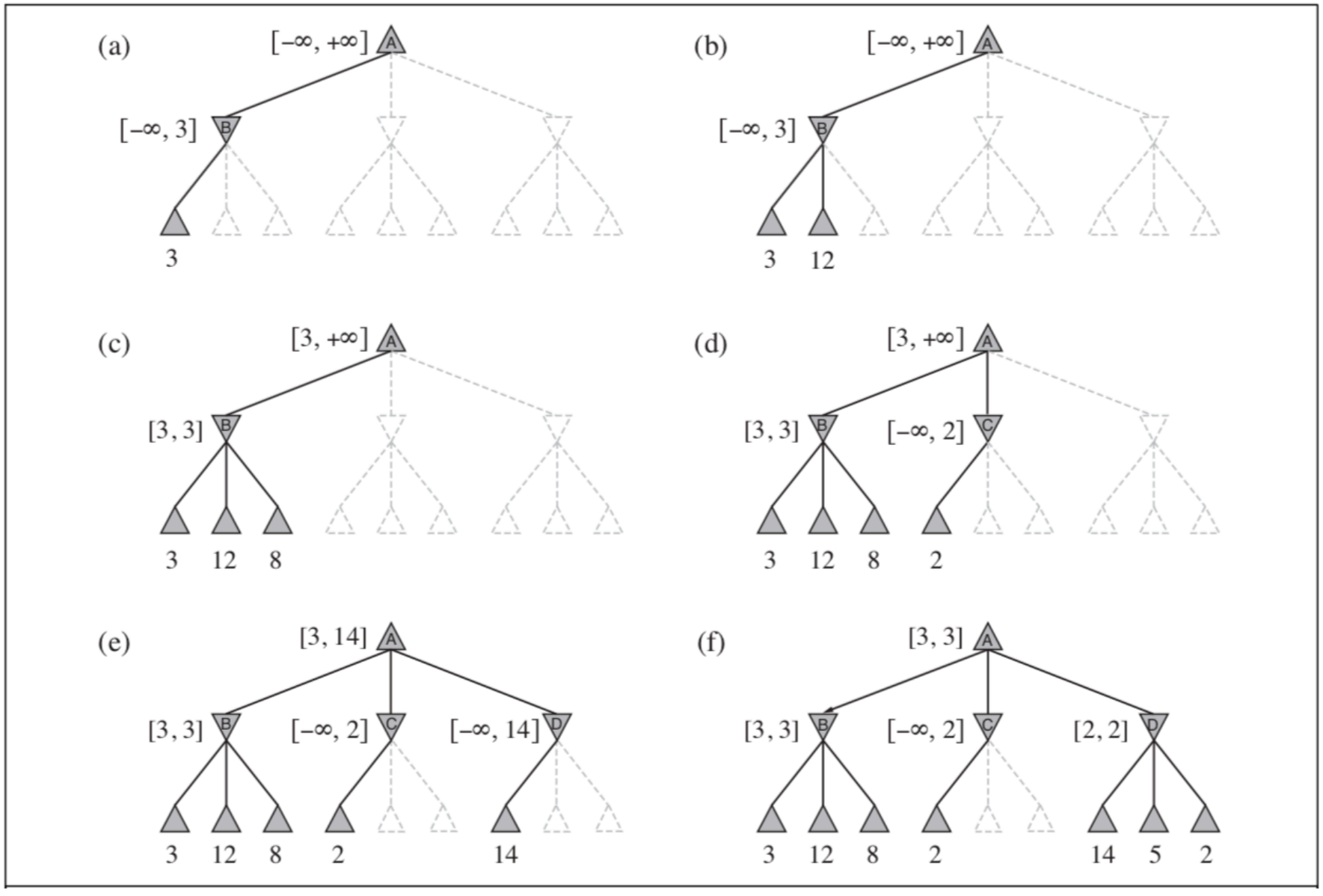
ALPHA-BETA PRUNING

- A pruning technique that can double the effective search depth by avoiding unnecessary state expansions.

ALPHA–BETA PRUNING

- Alpha (α) – the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- Beta (β) – the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

ALPHA-BETA PRUNING



AIMA, Fig 5.5, p168

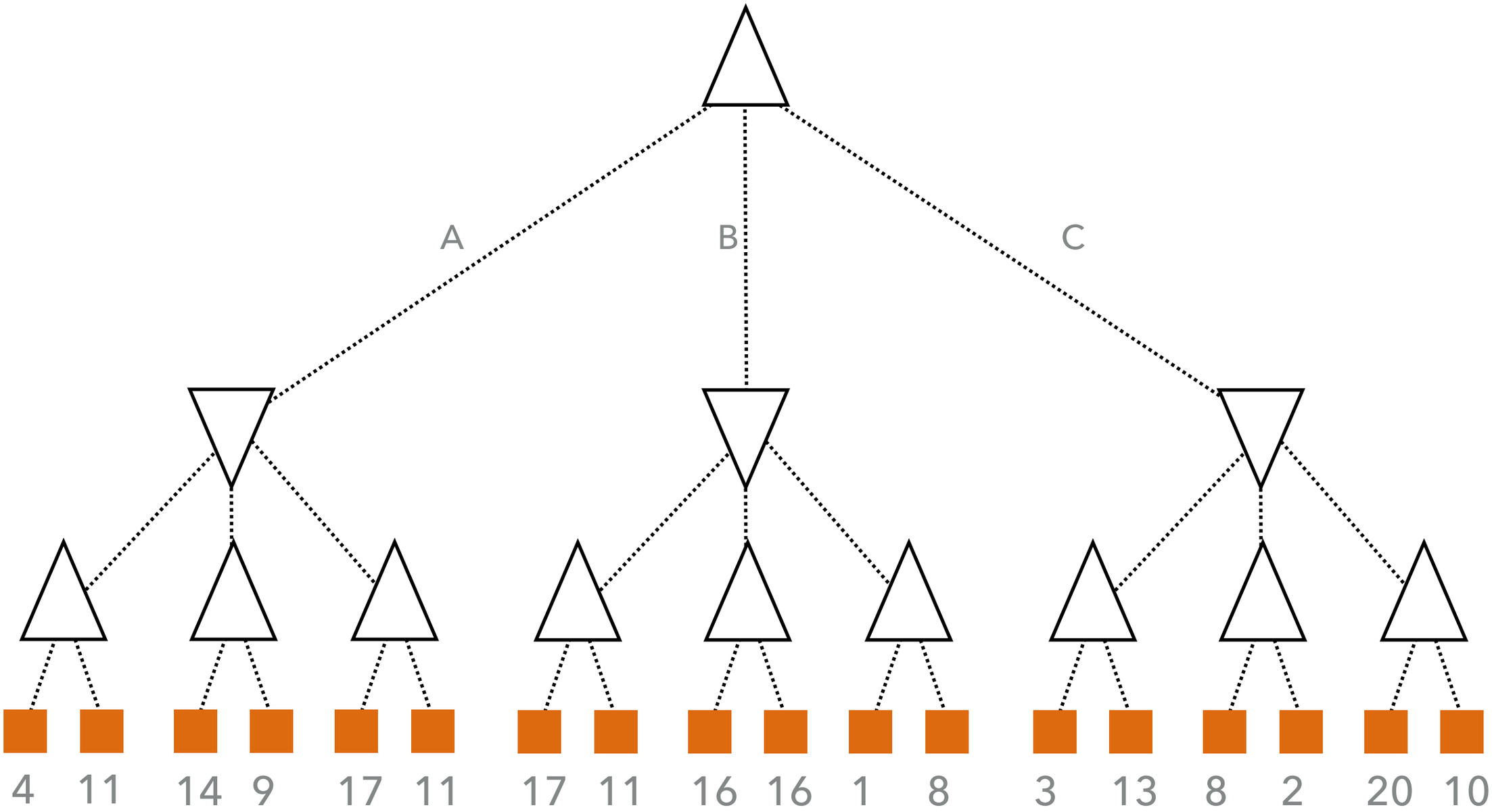
ALPHA-BETA PRUNING

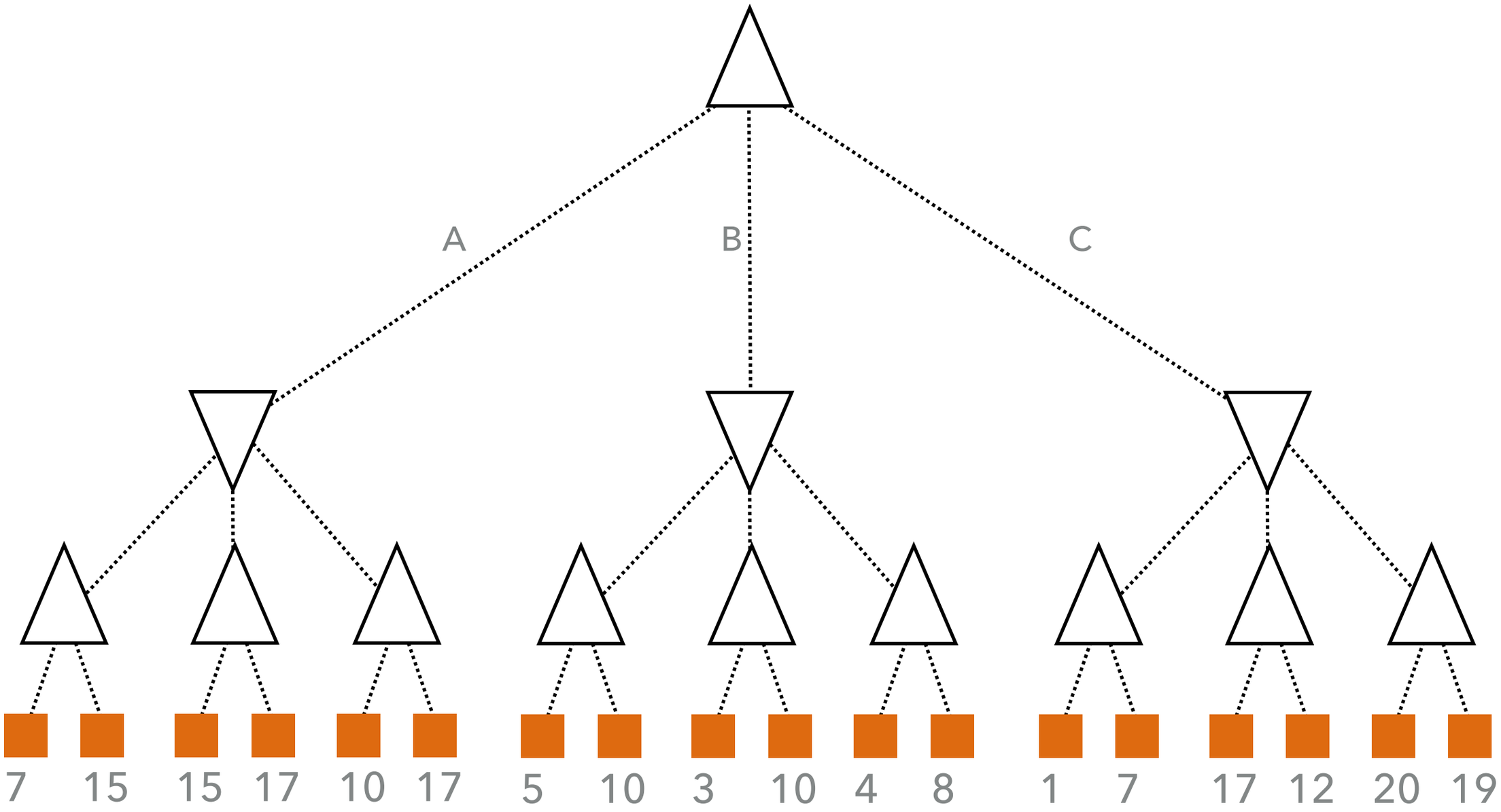
```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

AIMA, Fig 5.7, p170


A
B
C
4 11 14 9 17 11 17 11 16 16 1 8 3 13 8 2 20 10


A
B
C
7
15
15
17
10
17
5
10
3
10
4
8
1
7
17
12
20
19

ALPHA-BETA PRUNING

- Move ordering strongly effects the effective reduction in branching factor when using alpha-beta.
- with optimal move ordering – $\mathcal{O}(b^{\frac{m}{2}})$
- with random move ordering – $\mathcal{O}(b^{\frac{3m}{4}})$

ALPHA–BETA PRUNING

- Alpha-beta can effectively double maximum search depth within a given time limit; however, ...
- Minimax still has to search the tree to the leaves, which is wildly impractical for most games.
- What to do?

HEURISTIC MINIMAX

- Generalize terminal-test to **cutoff-test** and use **heuristic evaluation** after the cutoff, instead of searching all the way to terminals.

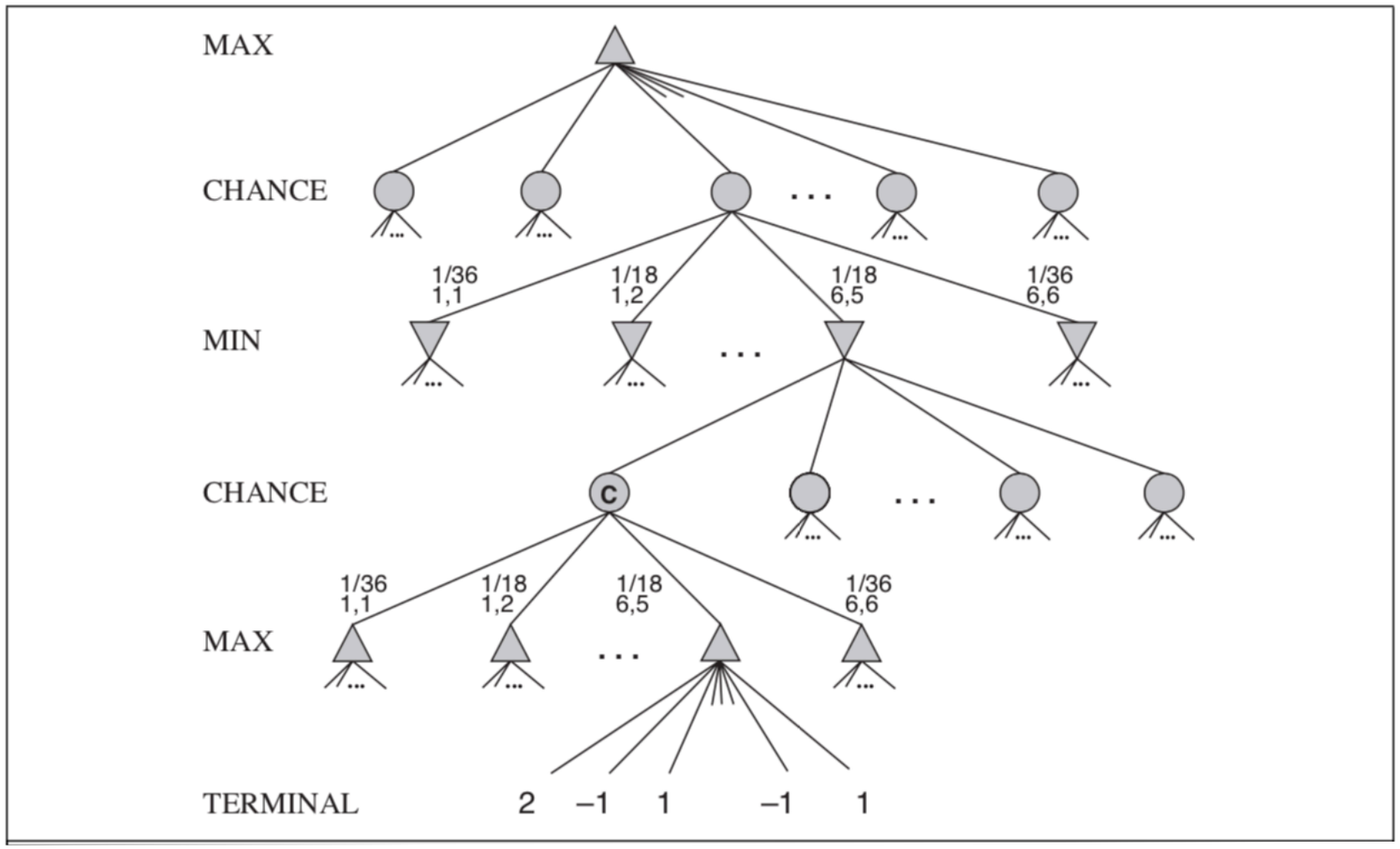
$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

AIMA, p171

ALPHA-BETA PRUNING

- What about stochastic games?
- We can no longer predict the actual value of a state, but if we know the probabilities of transitions and utilities we can calculate **expected values.**

EXPECTI-MINIMAX



AIMA, Fig 5.11, p178

EXPECTI-MINIMAX

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

AIMA, Fig 5.11, p178

SUMMARY OF SEARCH

- Computational (State-Space) Problem Solving
- Uninformed Search – BFS, DFS, IDS, UCS
- Informed Search – A*
- Adversarial Search – Minimax (+alpha-beta & heuristics)
- Next topic – logic