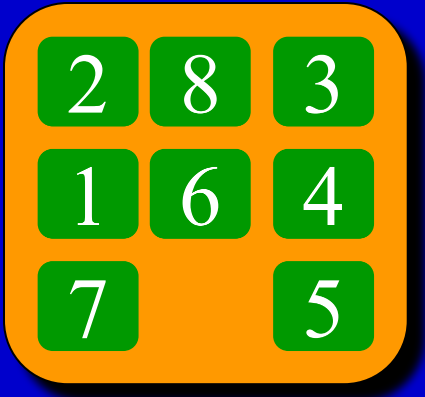
Artificial Intelligence
CSE 351

Dr. M Firoz Mridha
Associate Professor
Department of CSE
Bangladesh University of Business and Technology(BUBT)

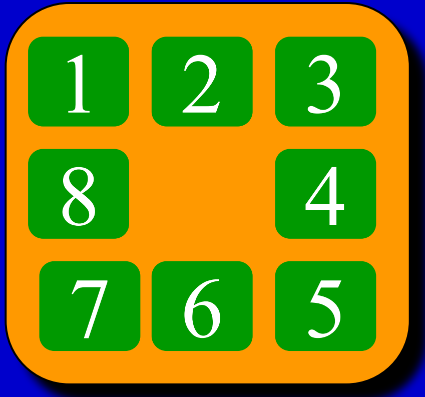
Problem Solving and Search

Breadth-first search
Depth-first search
Iterative deepening search
Hill-Climbing

Example: The 8-puzzle



Start



Goal

- States: Integer location of tiles (ignore intermediate positions)
- Operators: Move blank left, right, up, down
- Goal Test: = goal state (given)
- Path Cost: 1 per move

Missionaries and cannibals

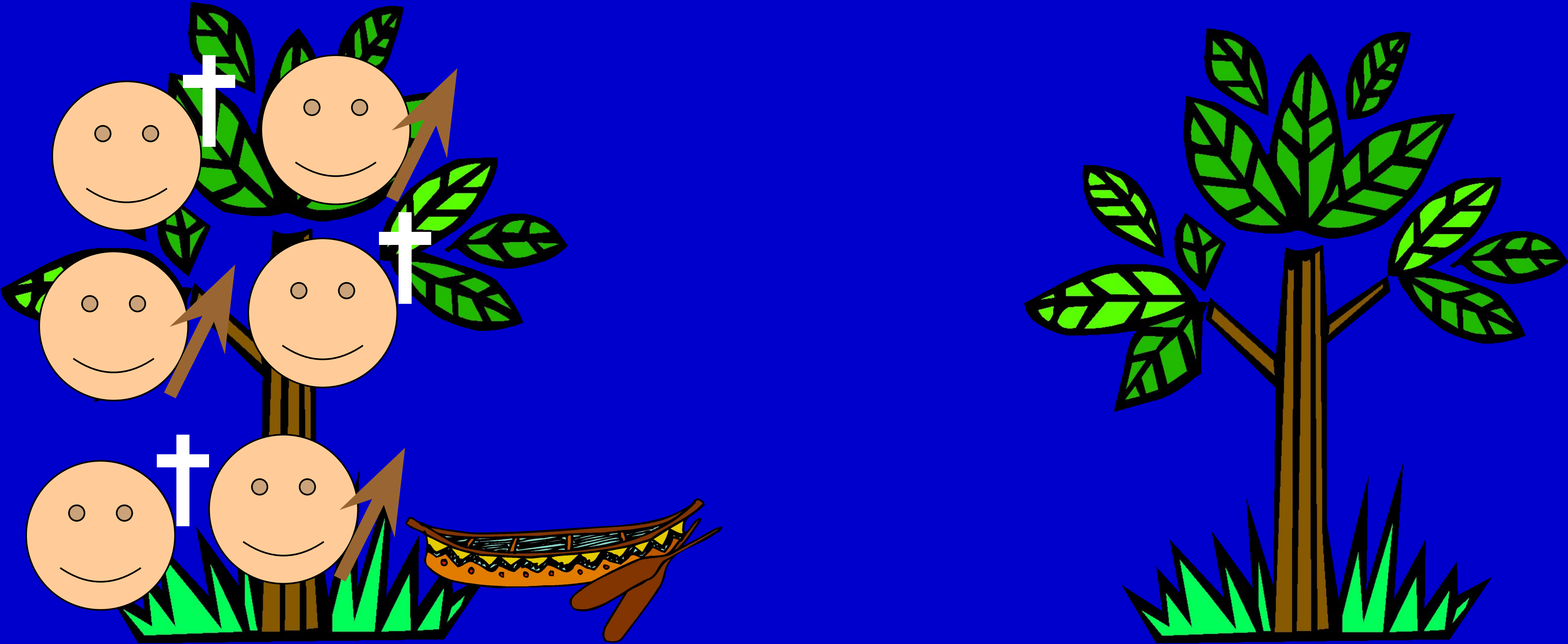
- Three missionaries and three cannibals are on the left bank of a river.
- There is one boat which can hold one or two people.
- Find a way to get everyone to the right bank, without ever leaving a group of missionaries in one place outnumbered by cannibals in that place.

Missionaries and cannibals

- States: three numbers representing the number of missionaries, cannibals, and boat on the left bank of the river.
- Initial state: (3, 3, 1)
- Operators: take one missionary, one cannibal, two missionaries, two cannibals, one missionary and one cannibal across the river in a given direction.
- Goal test: reached state (0, 0, 0)
- Path cost: Number of crossings.

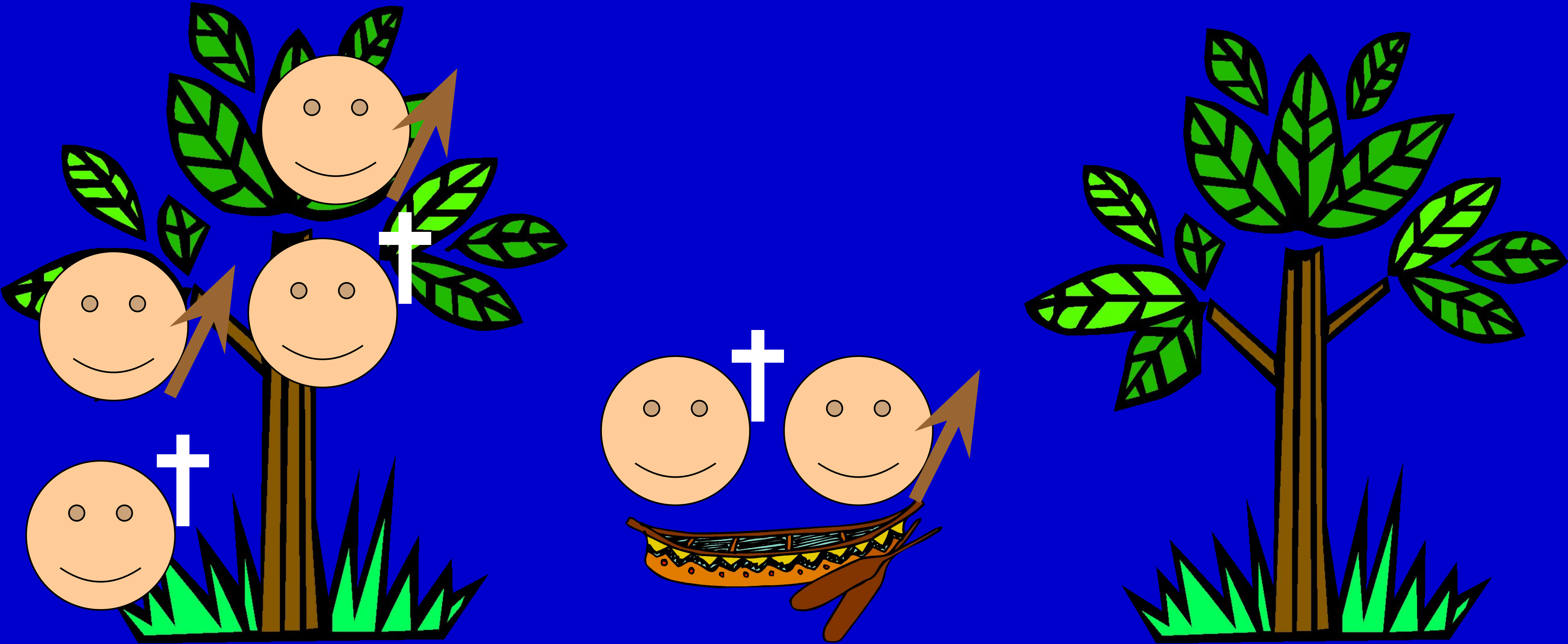
Missionaries and Cannibals

(3,3,1)



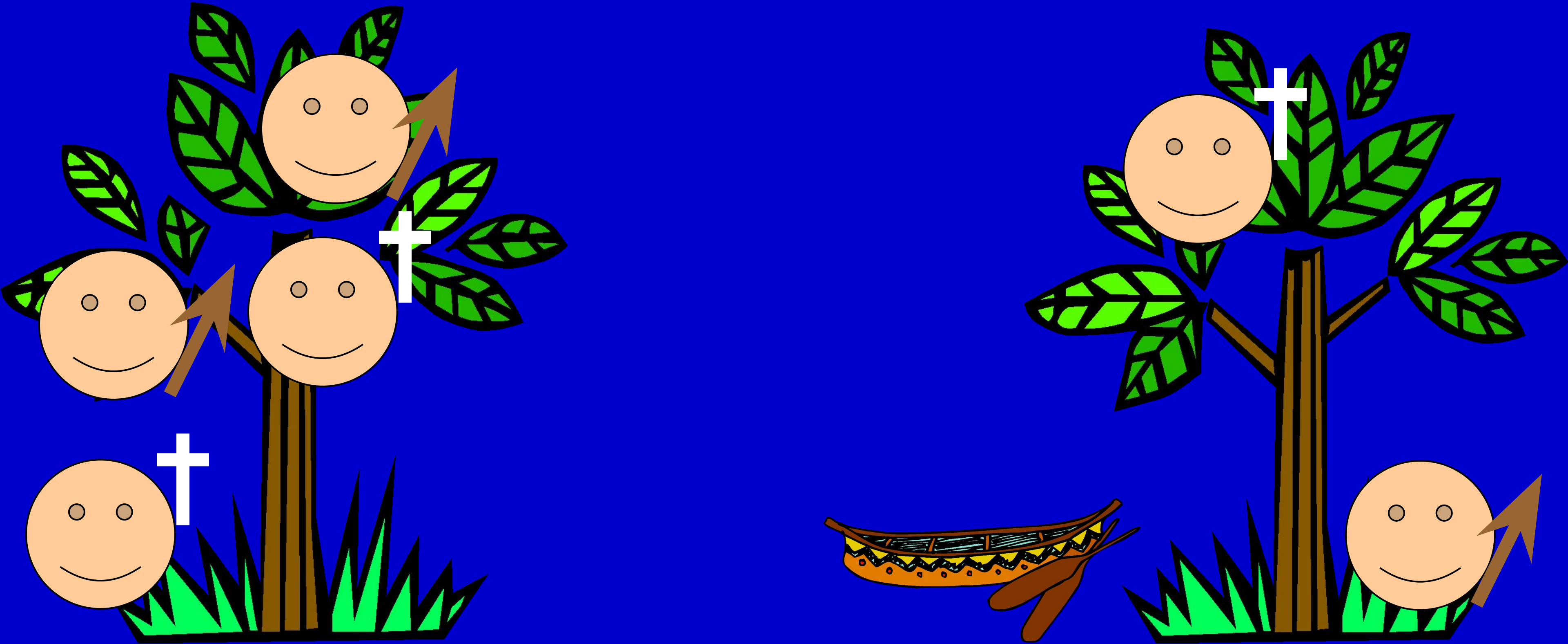
Missionaries and Cannibals

A missionary and cannibal cross



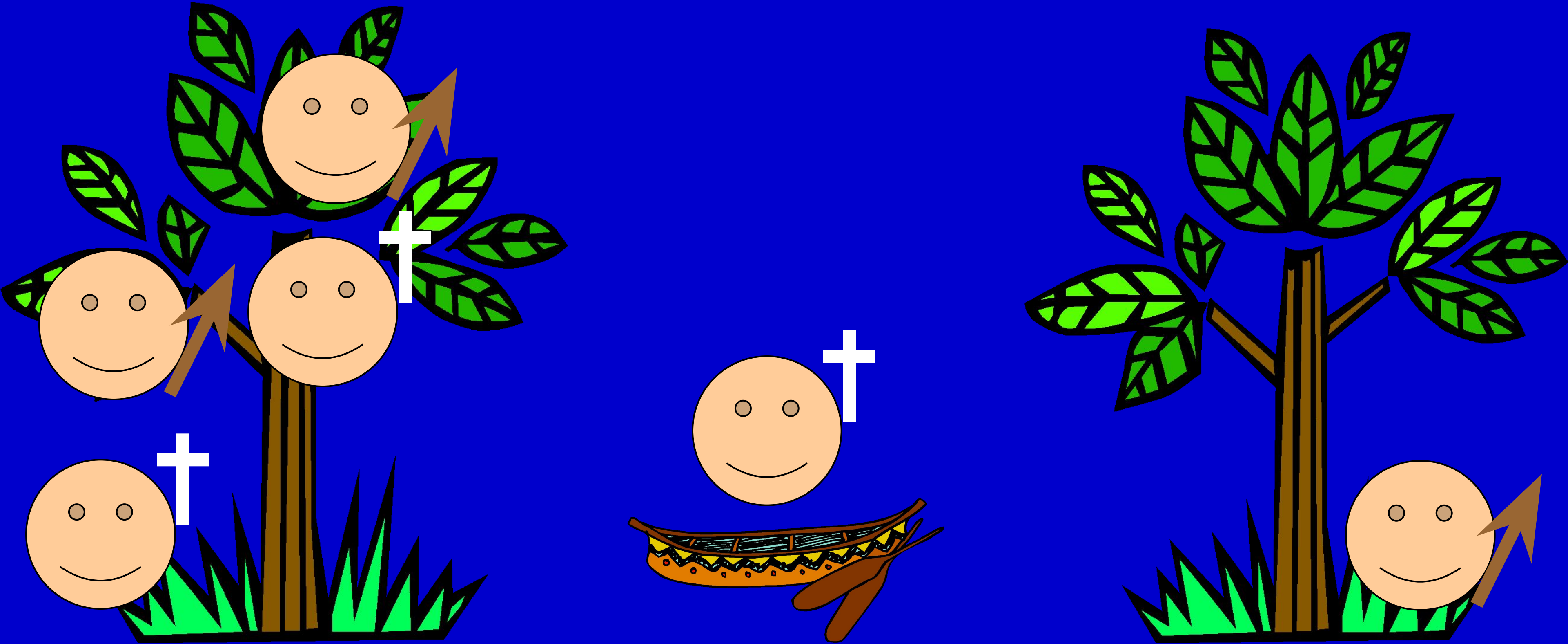
Missionaries and Cannibals

(2,2,0)



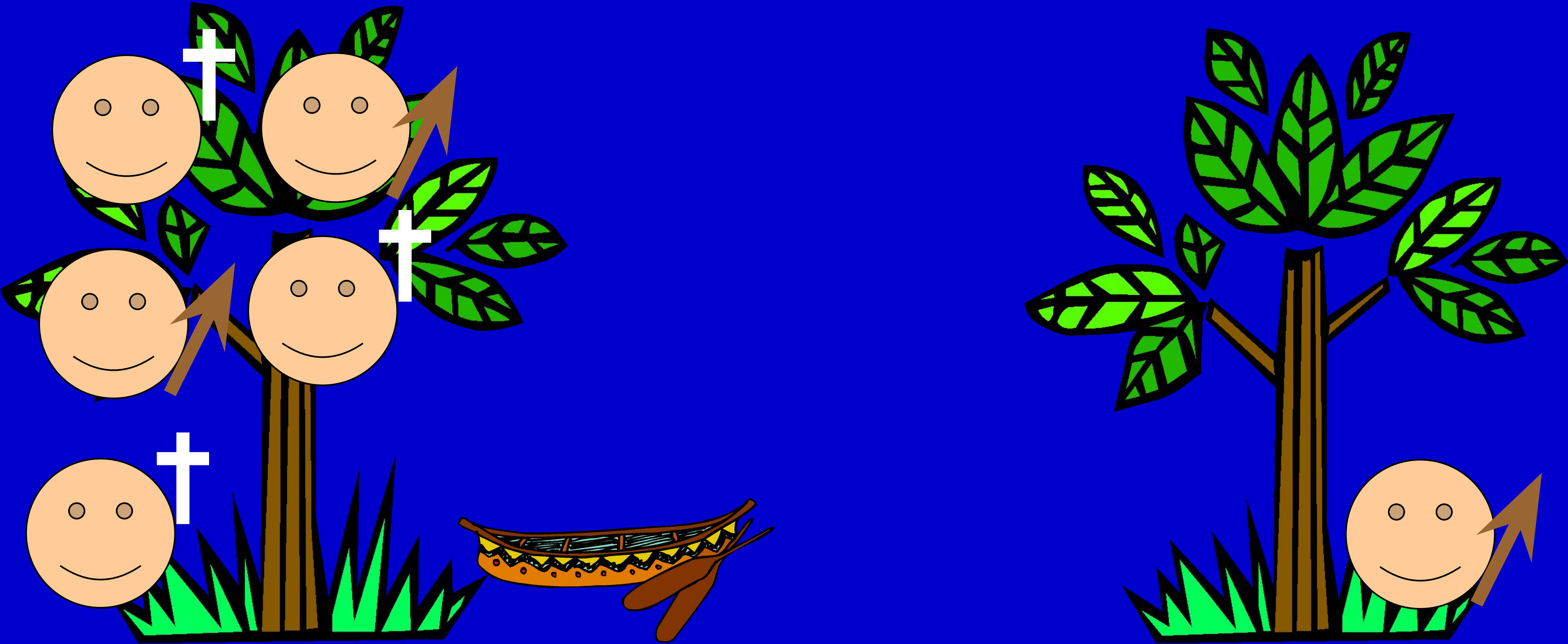
Missionaries and Cannibals

One missionary returns



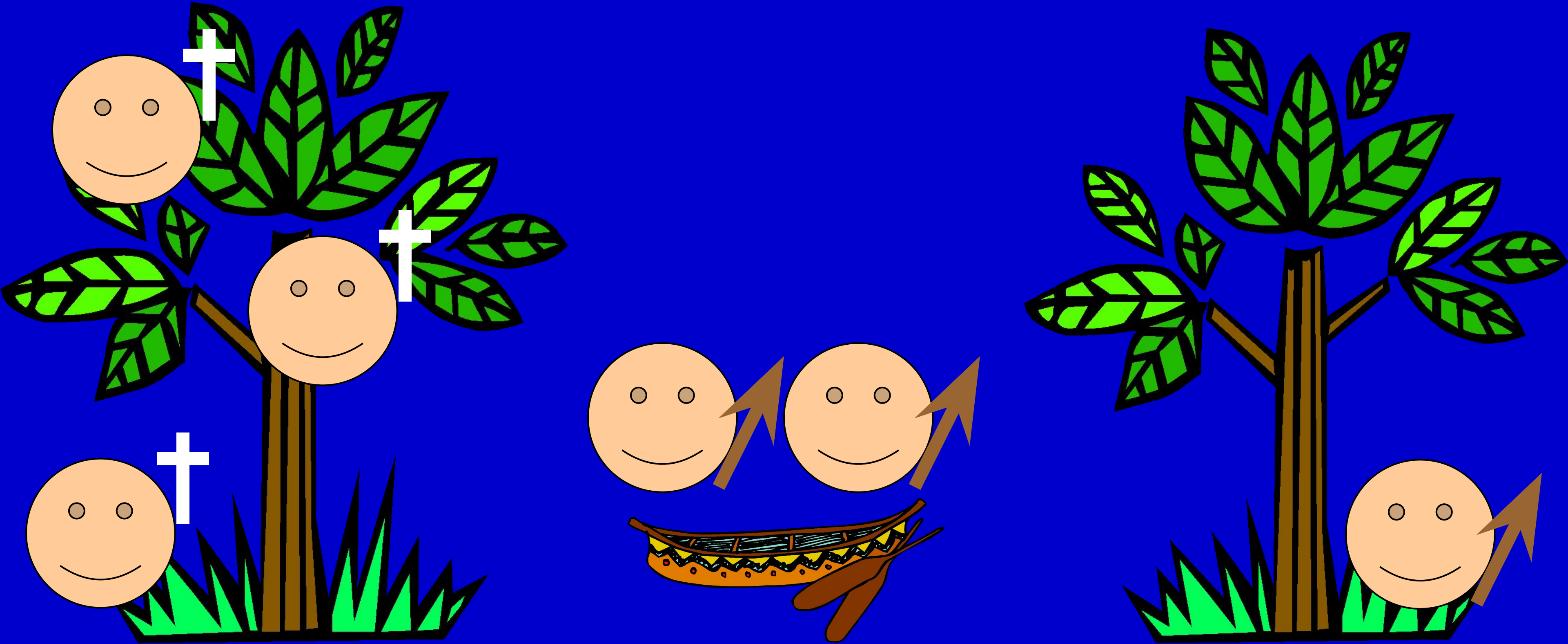
Missionaries and Cannibals

(3,2,1)



Missionaries and Cannibals

Two cannibals cross

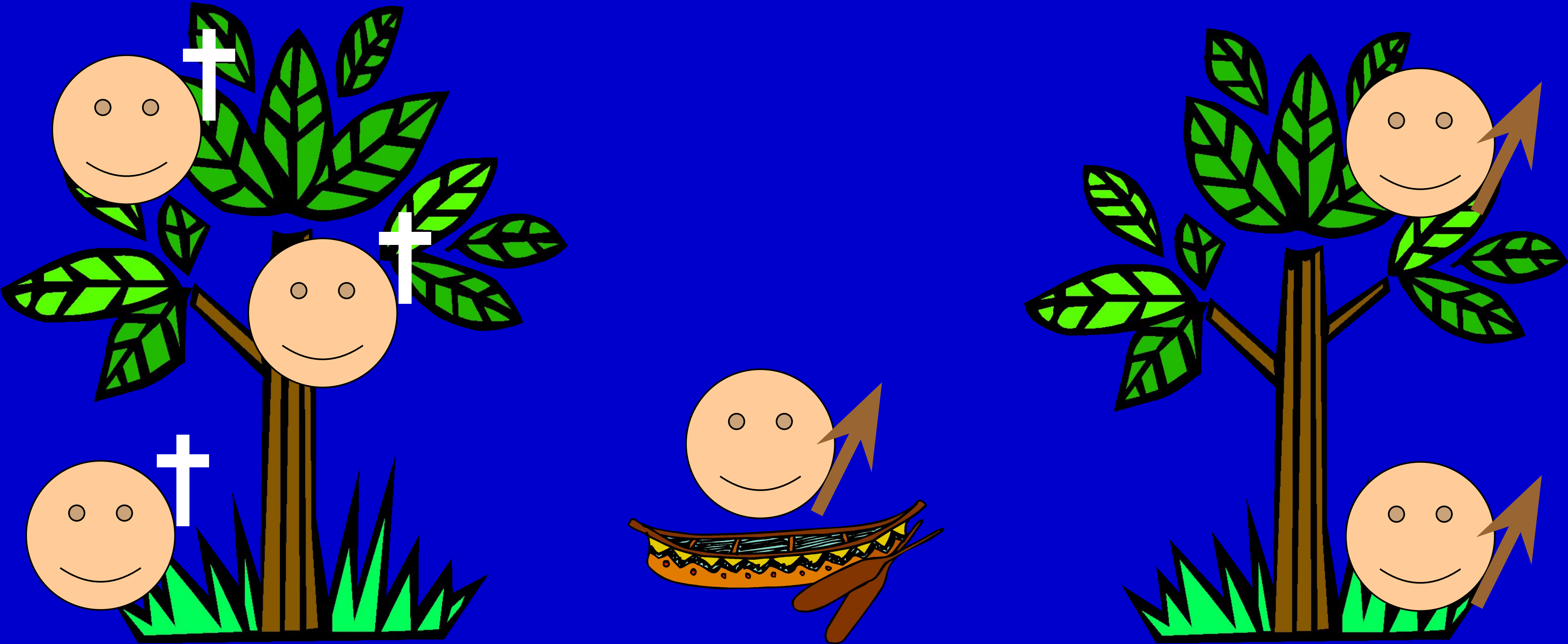


Missionaries and Cannibals

(3,0,0)

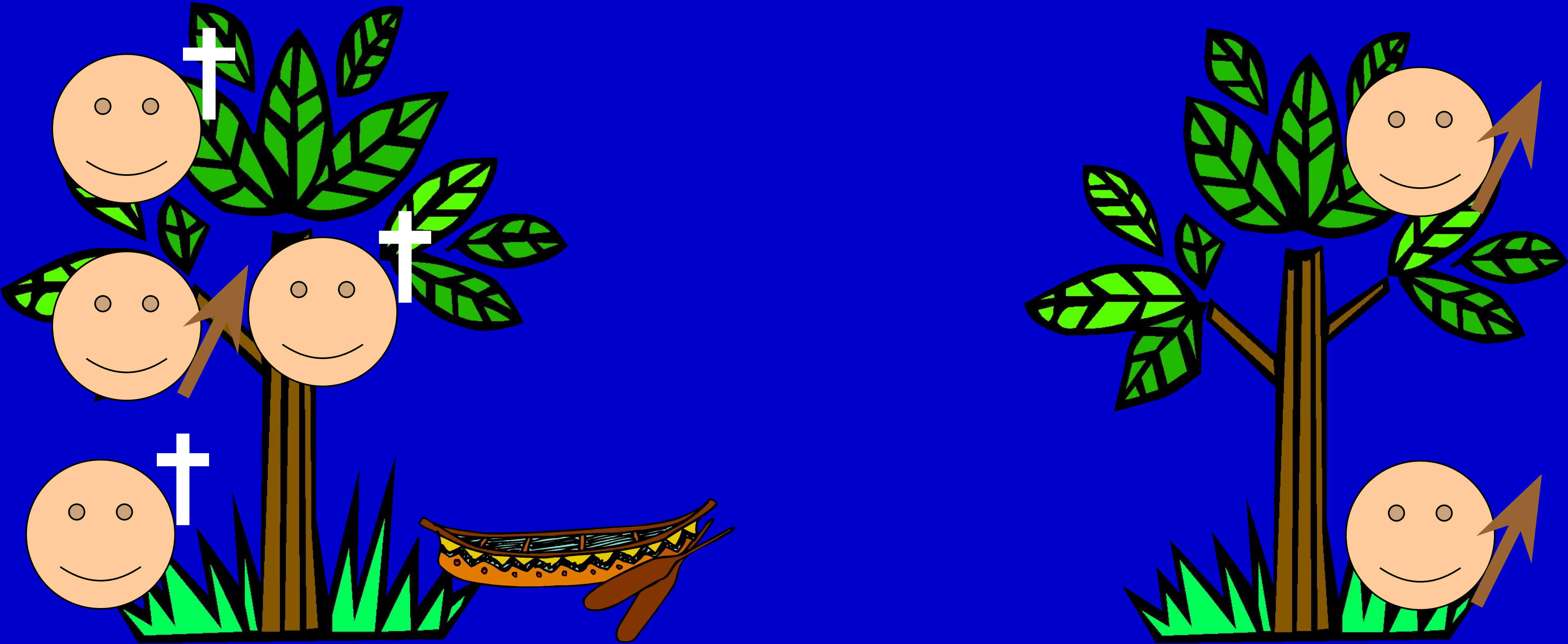
Missionaries and Cannibals

A cannibal returns



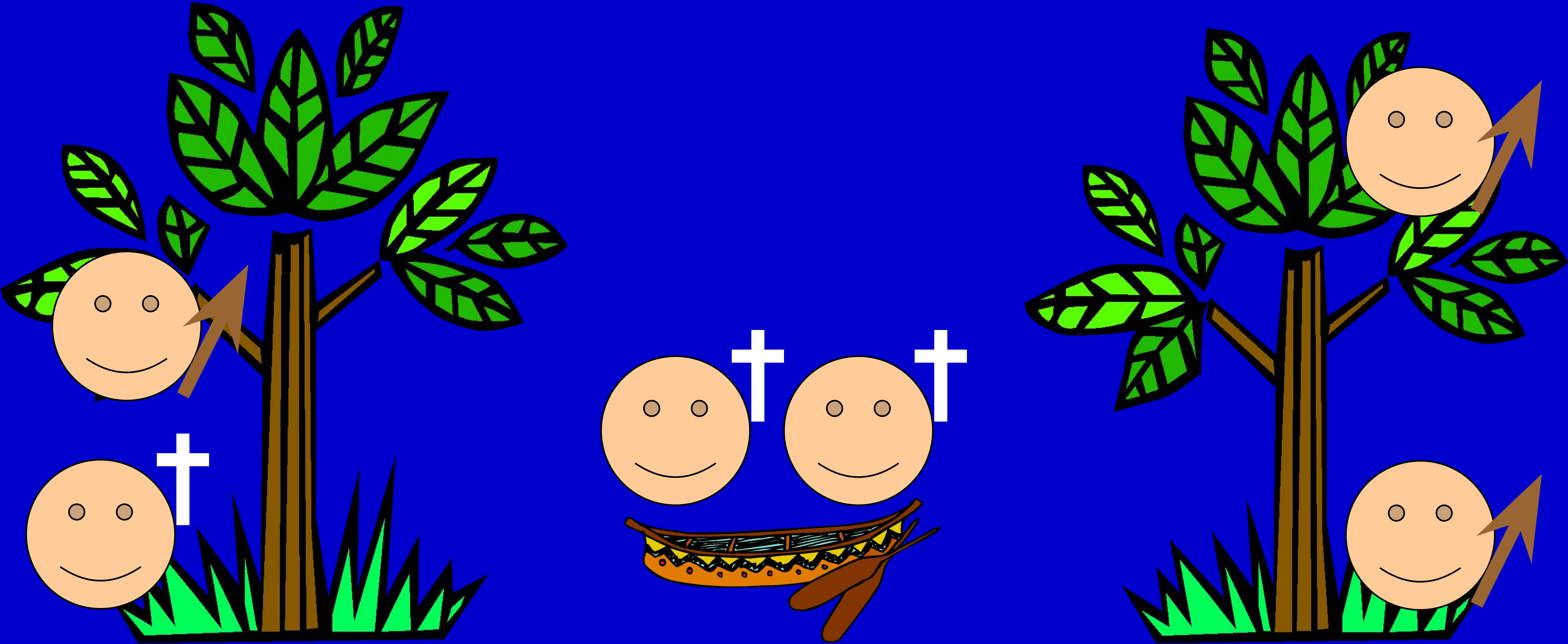
Missionaries and Cannibals

(3,1,1)



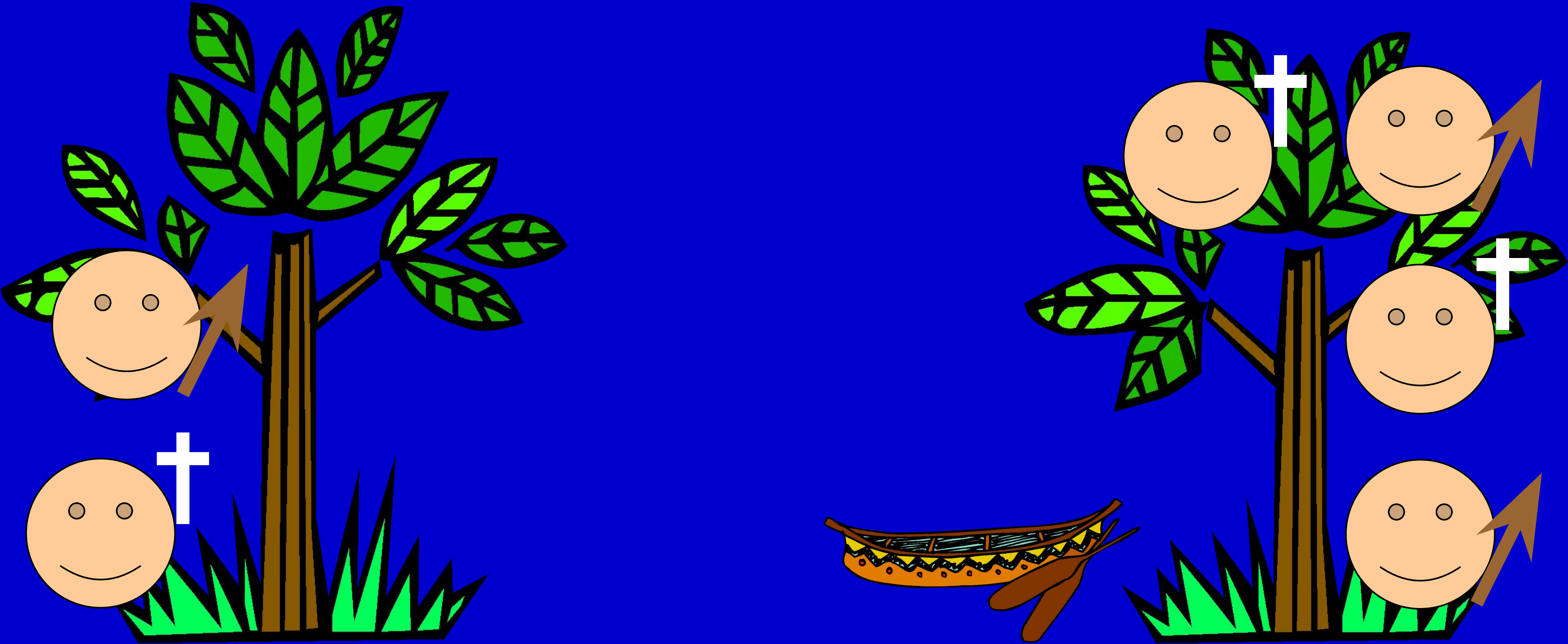
Missionaries and Cannibals

Two missionaries cross



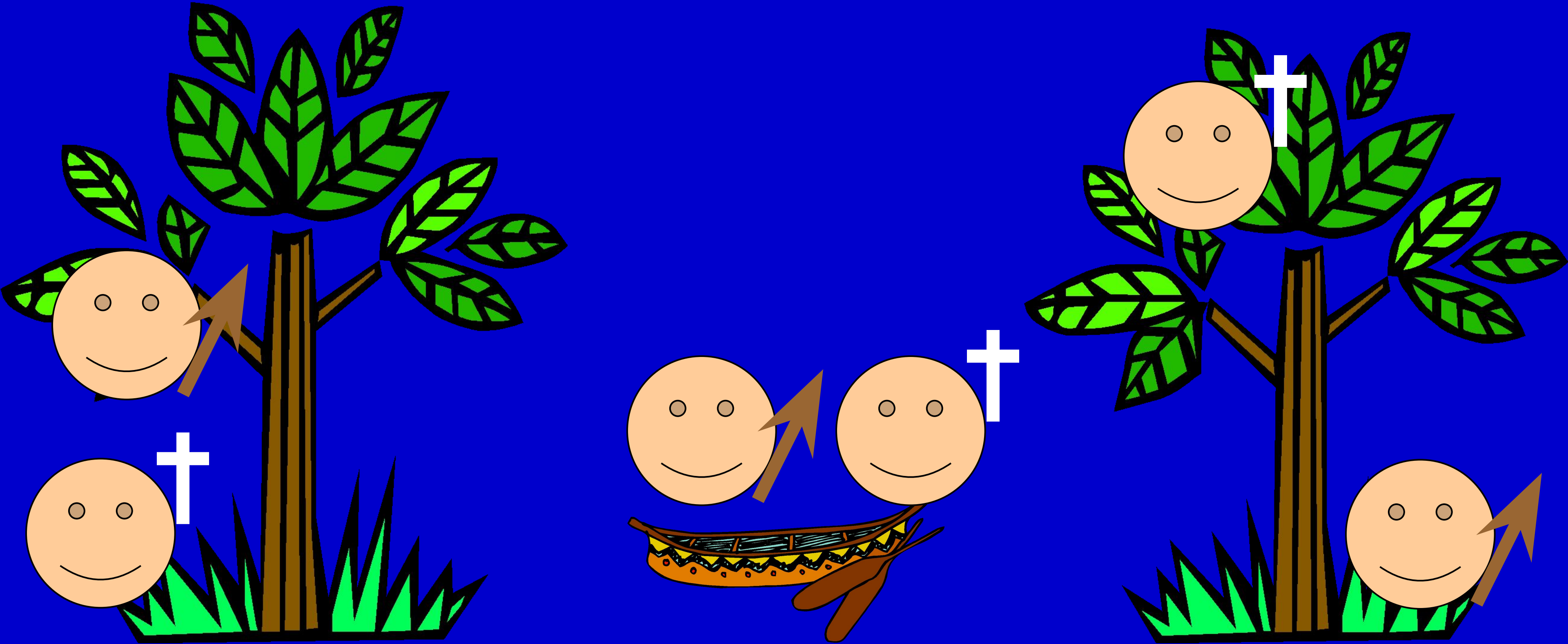
Missionaries and Cannibals

(1,1,0)



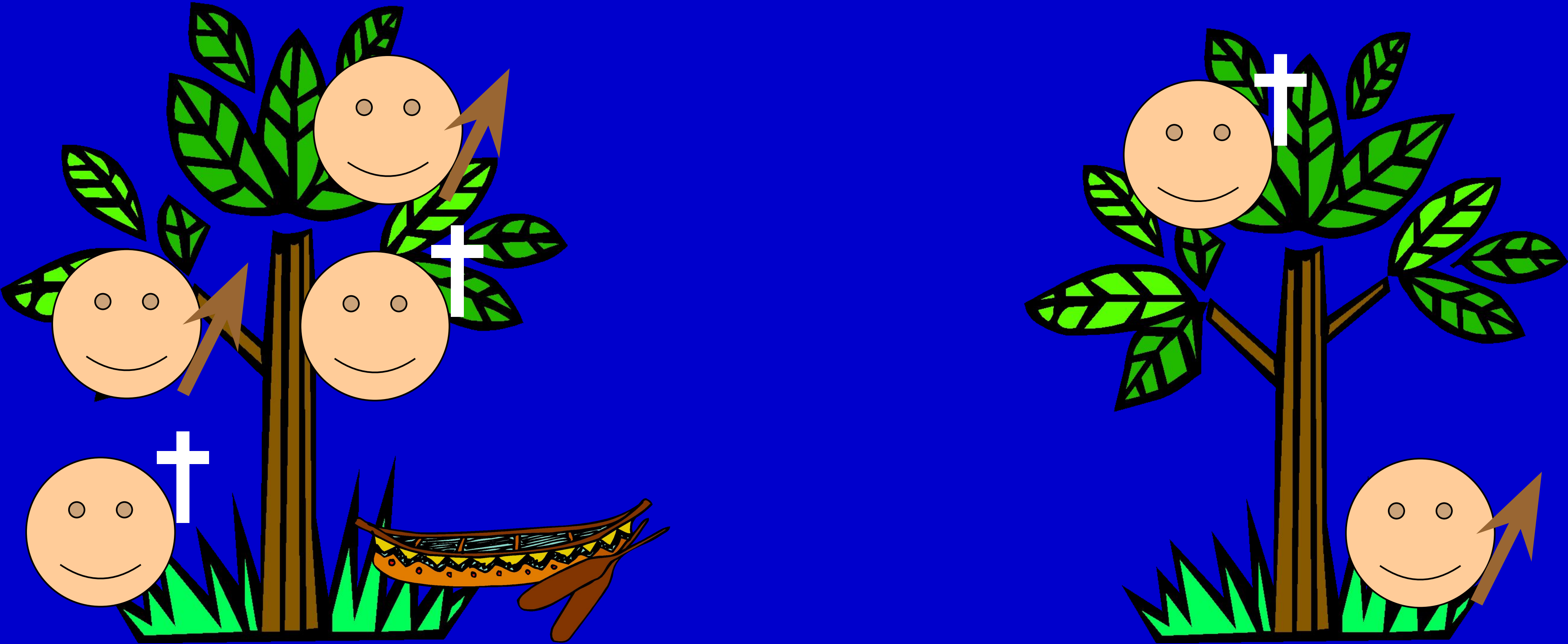
Missionaries and Cannibals

A missionary and cannibal return



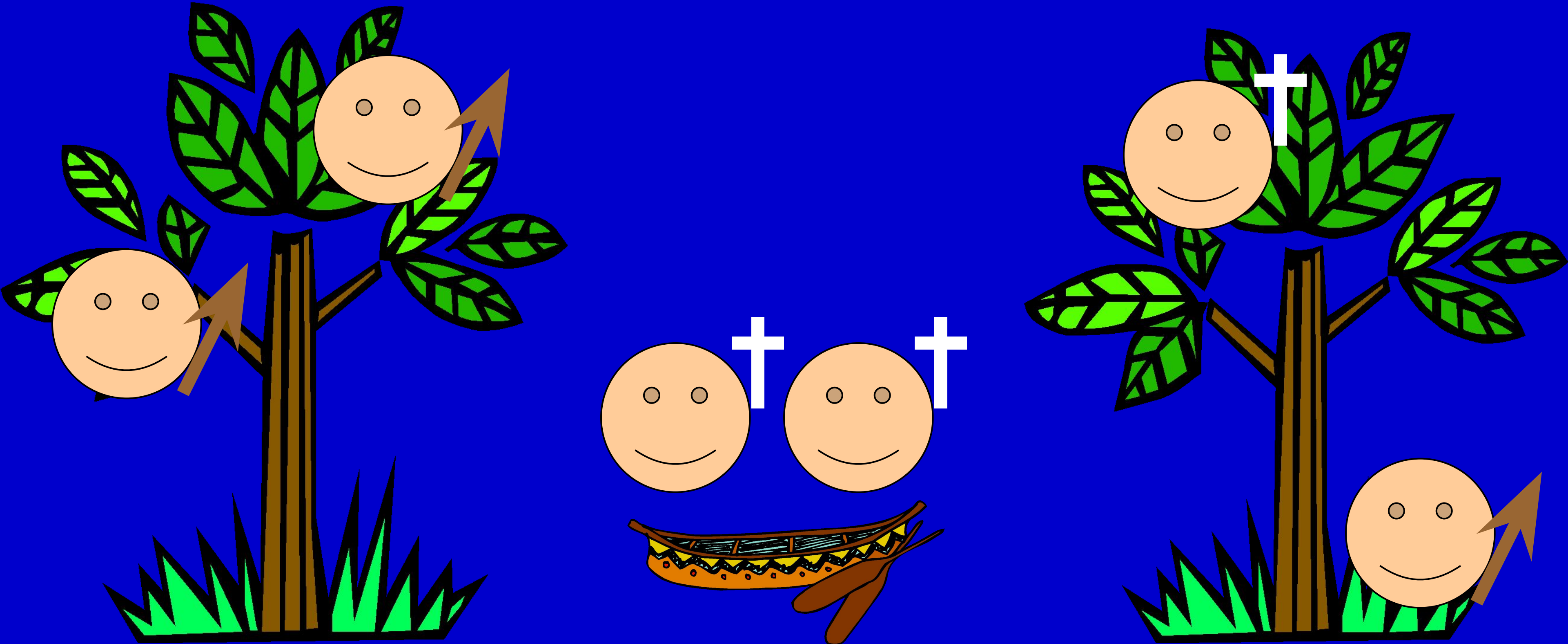
Missionaries and Cannibals

(2,2,1)



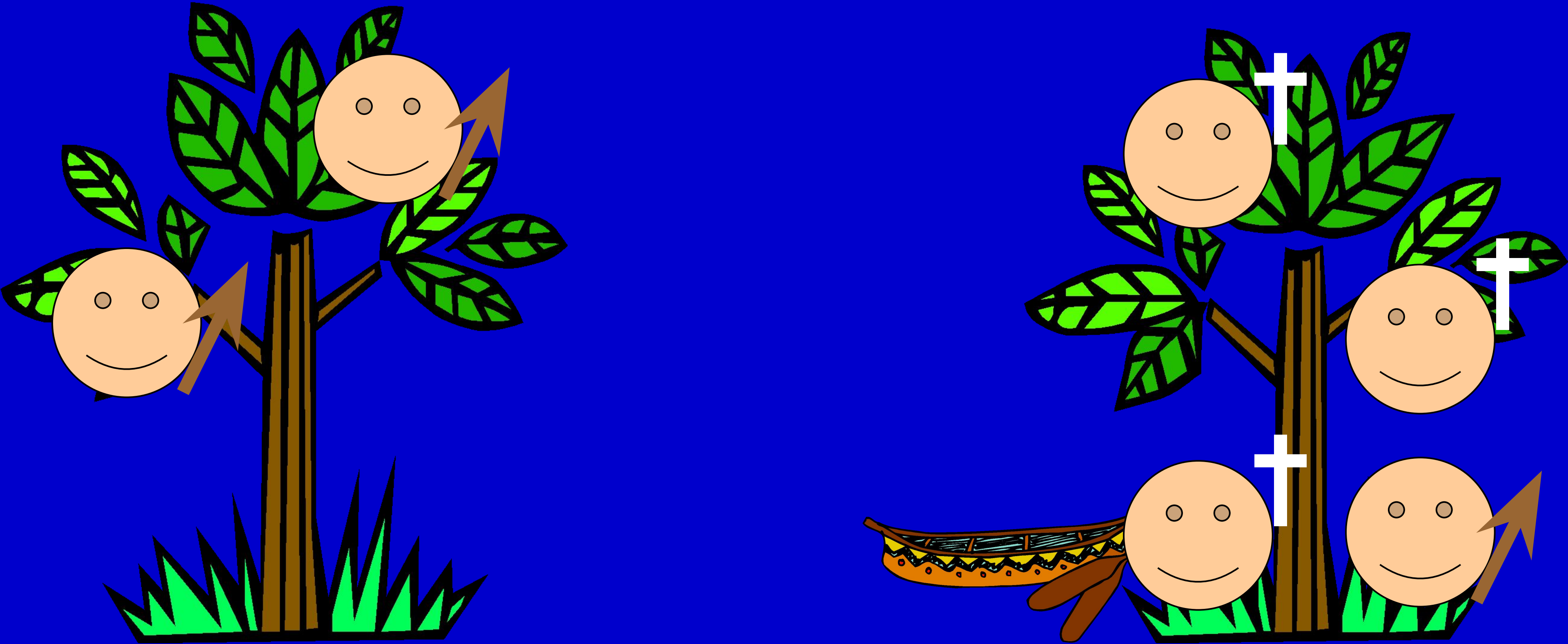
Missionaries and Cannibals

Two Missionaries cross



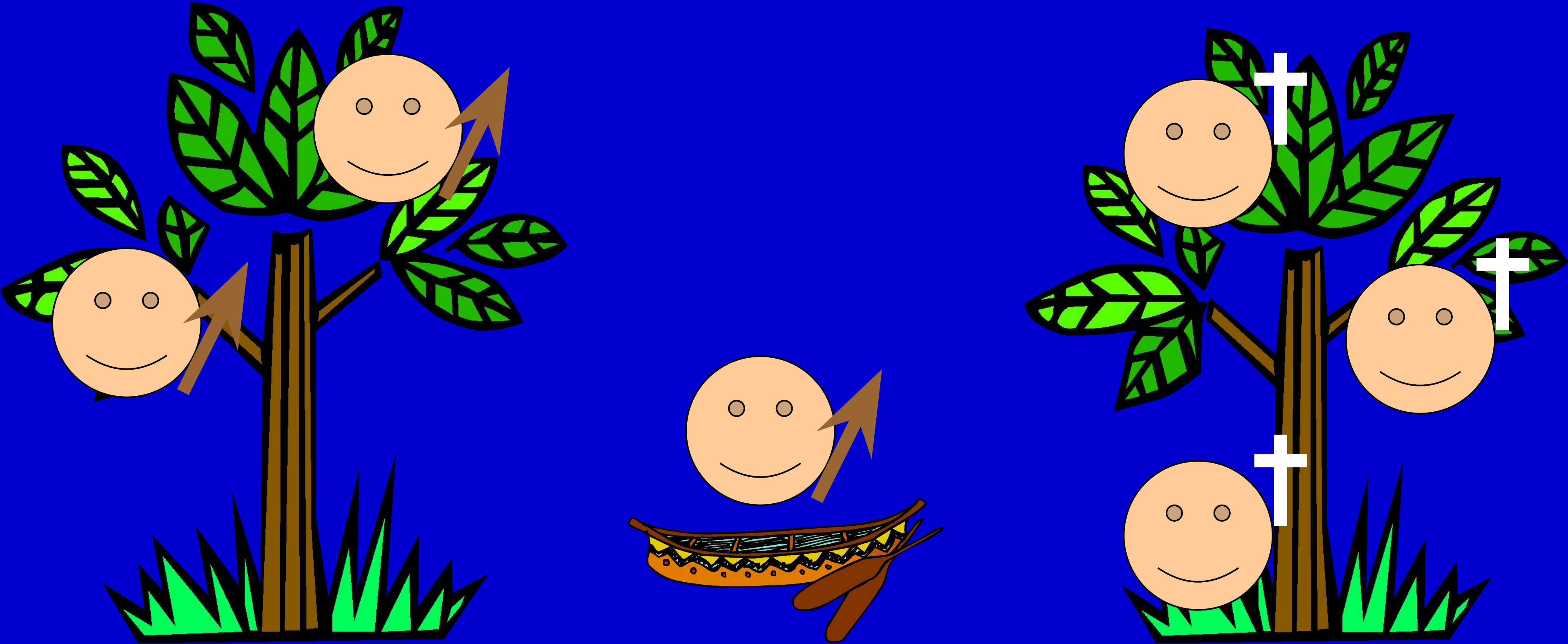
Missionaries and Cannibals

(0,2,0)



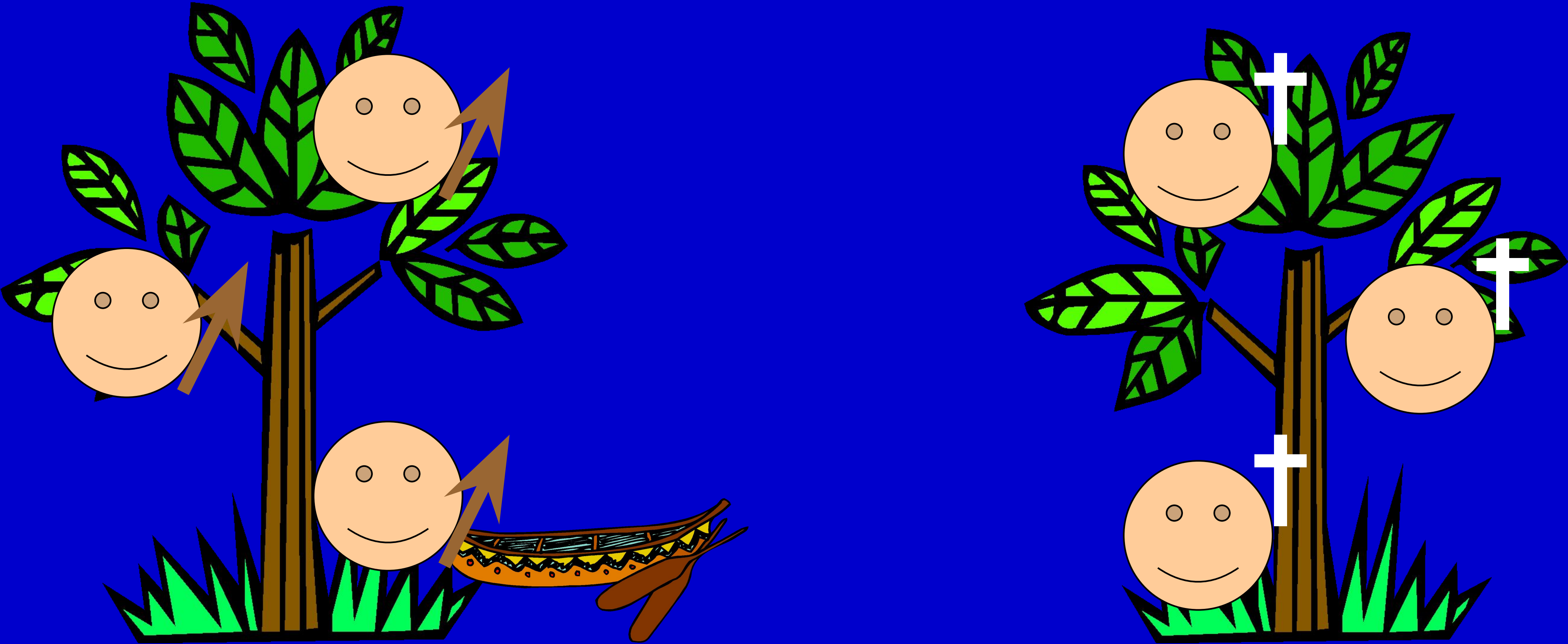
Missionaries and Cannibals

A cannibal returns



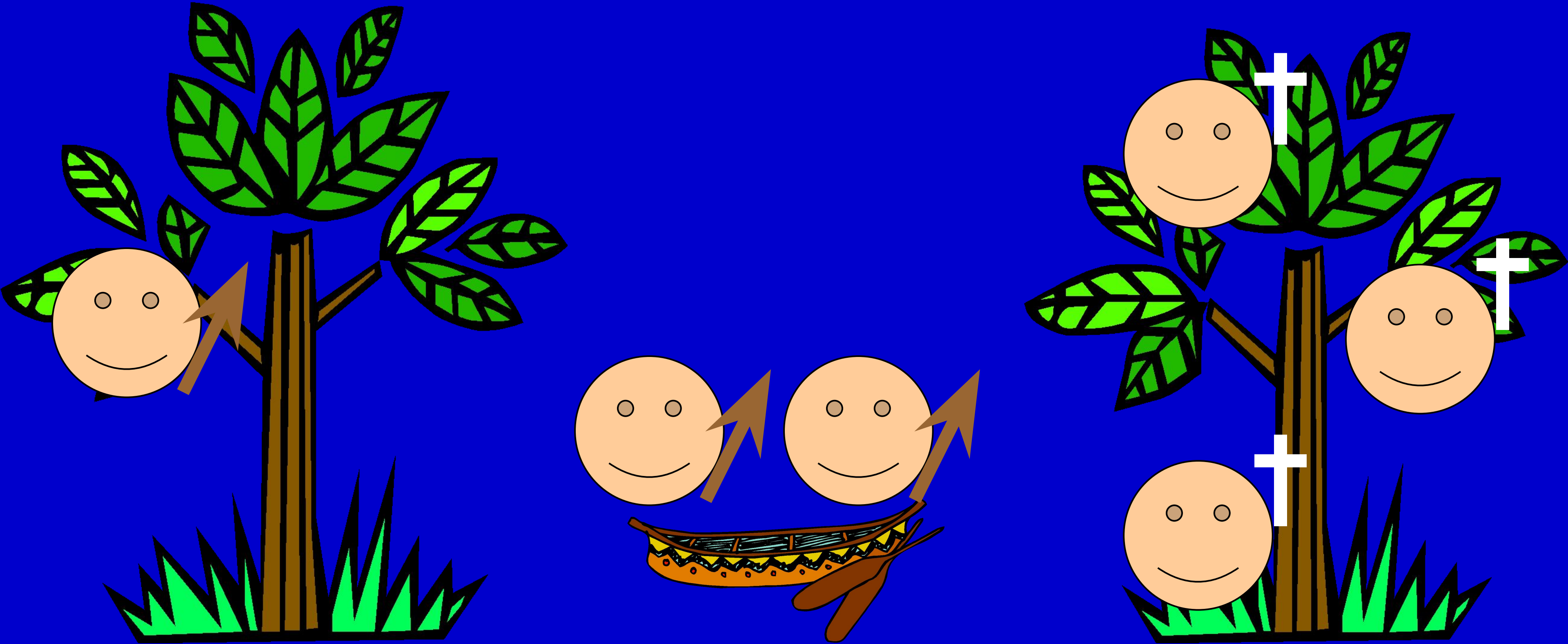
Missionaries and Cannibals

(0,3,1)



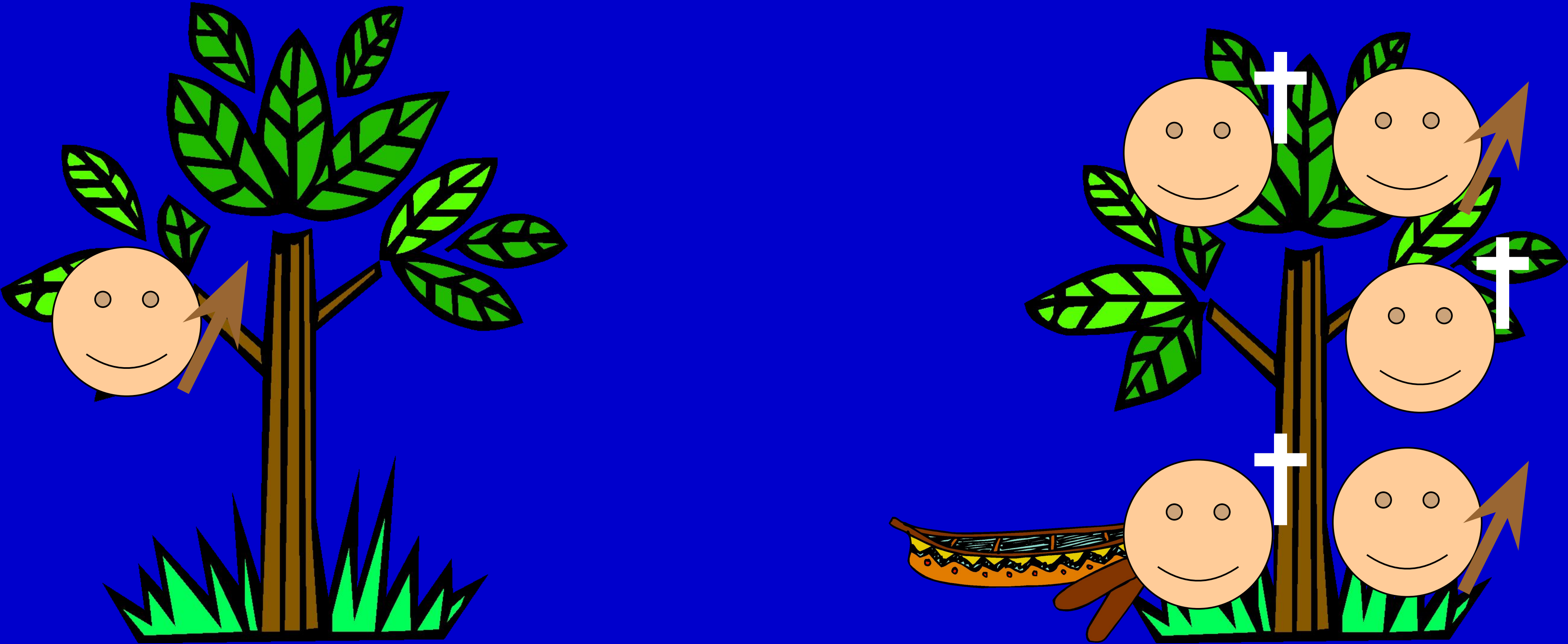
Missionaries and Cannibals

Two cannibals cross



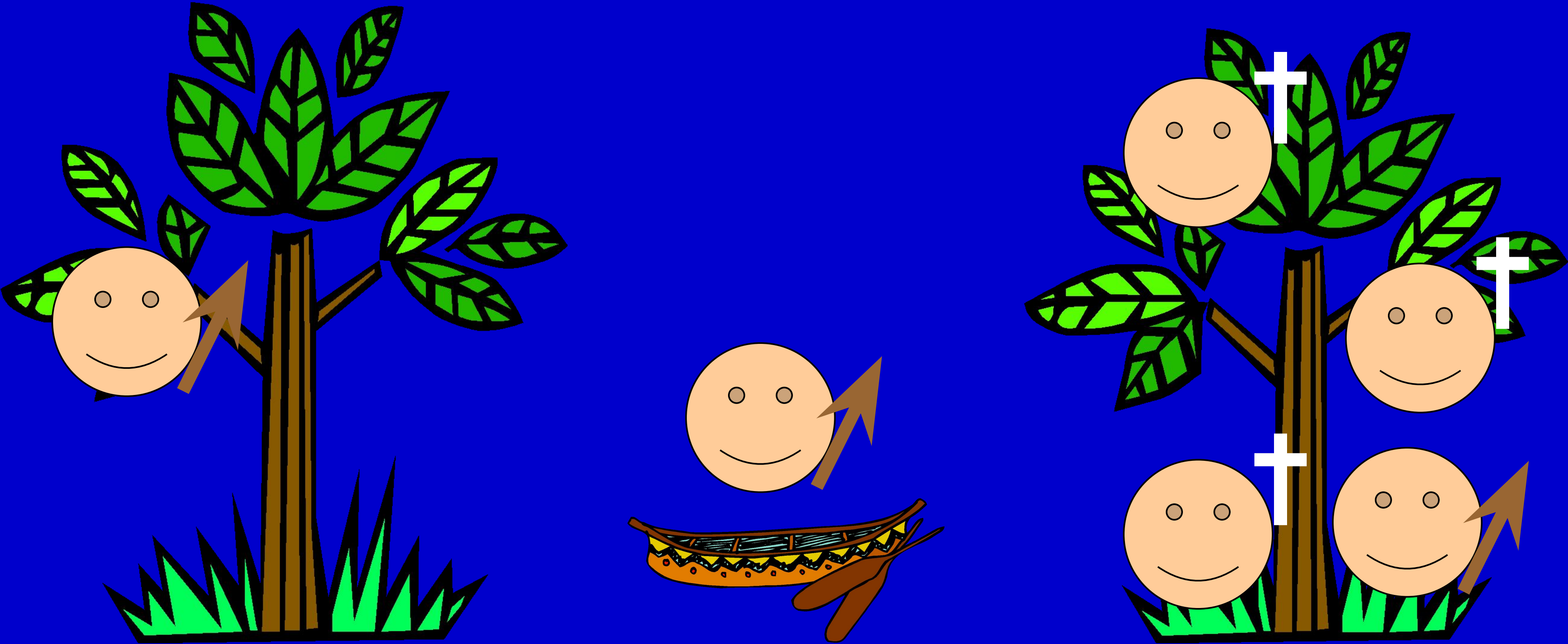
Missionaries and Cannibals

(0,1,0)



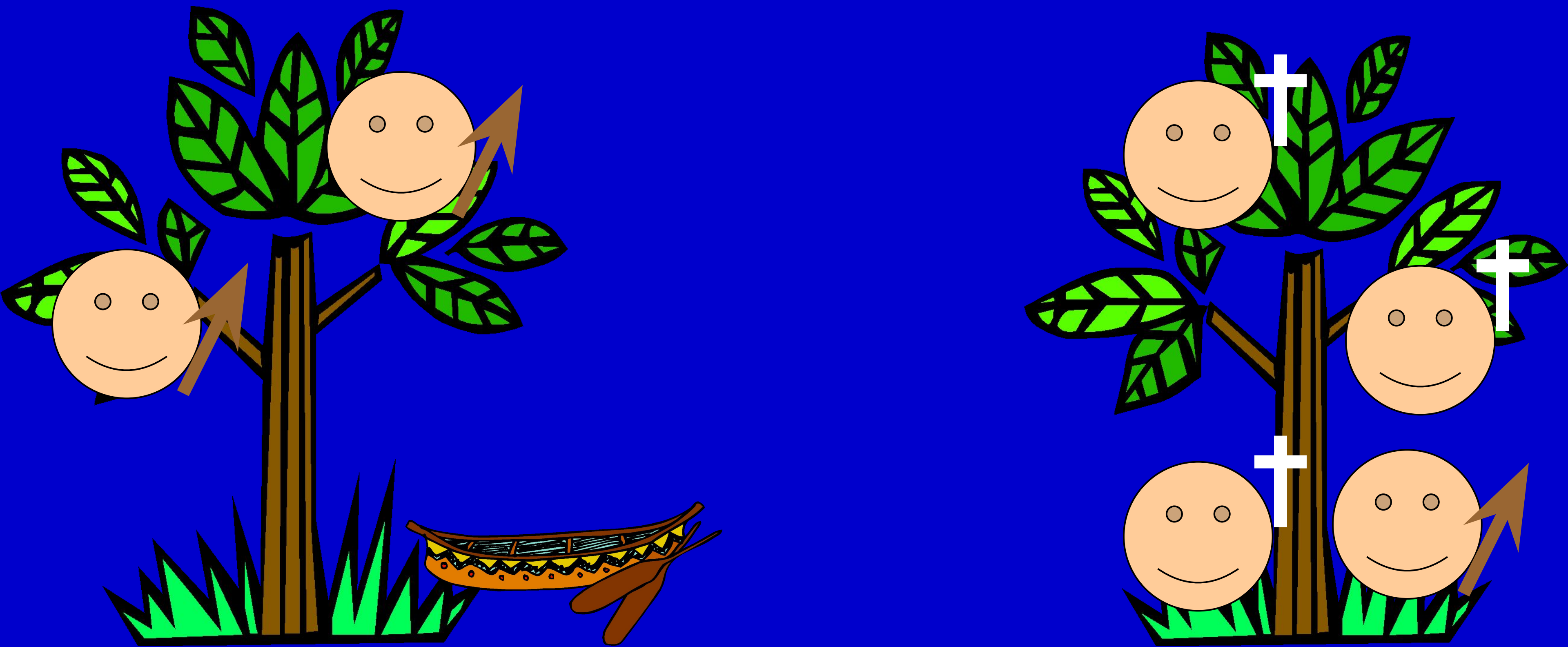
Missionaries and Cannibals

A cannibal returns



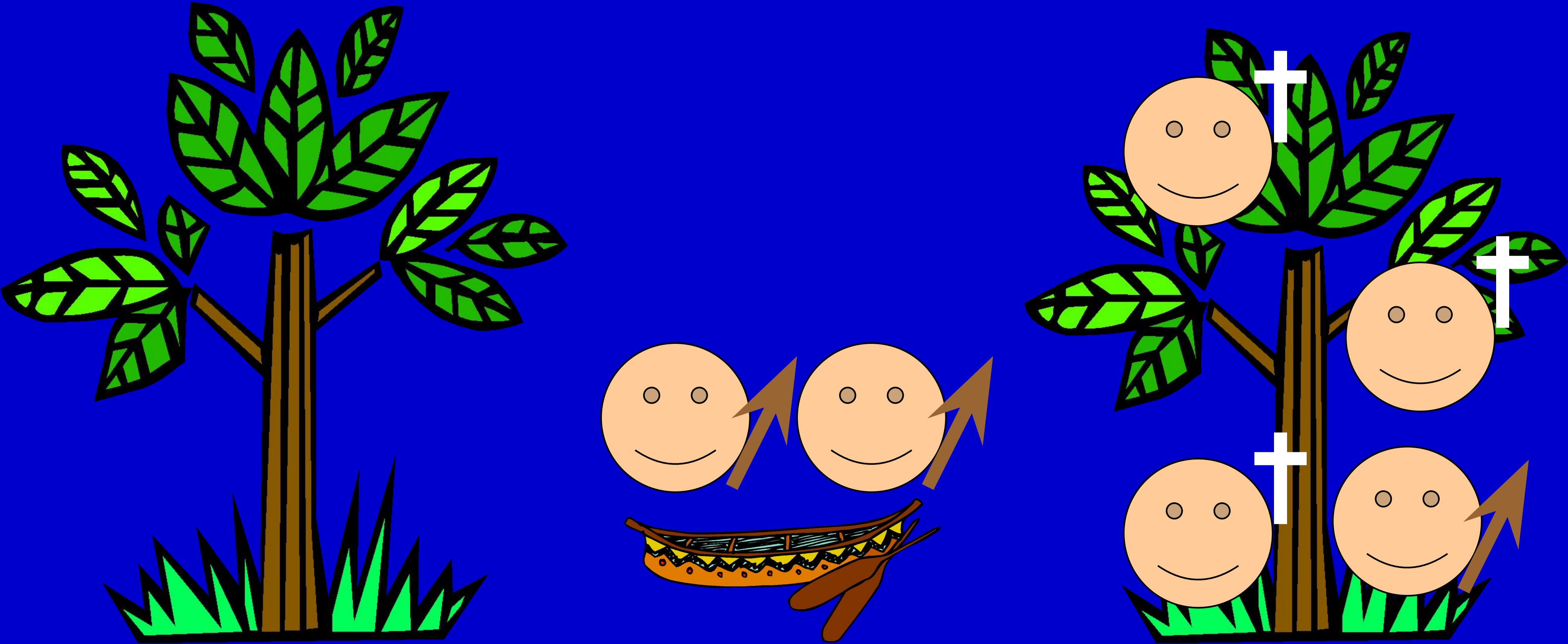
Missionaries and Cannibals

(0,2,1)



Missionaries and Cannibals

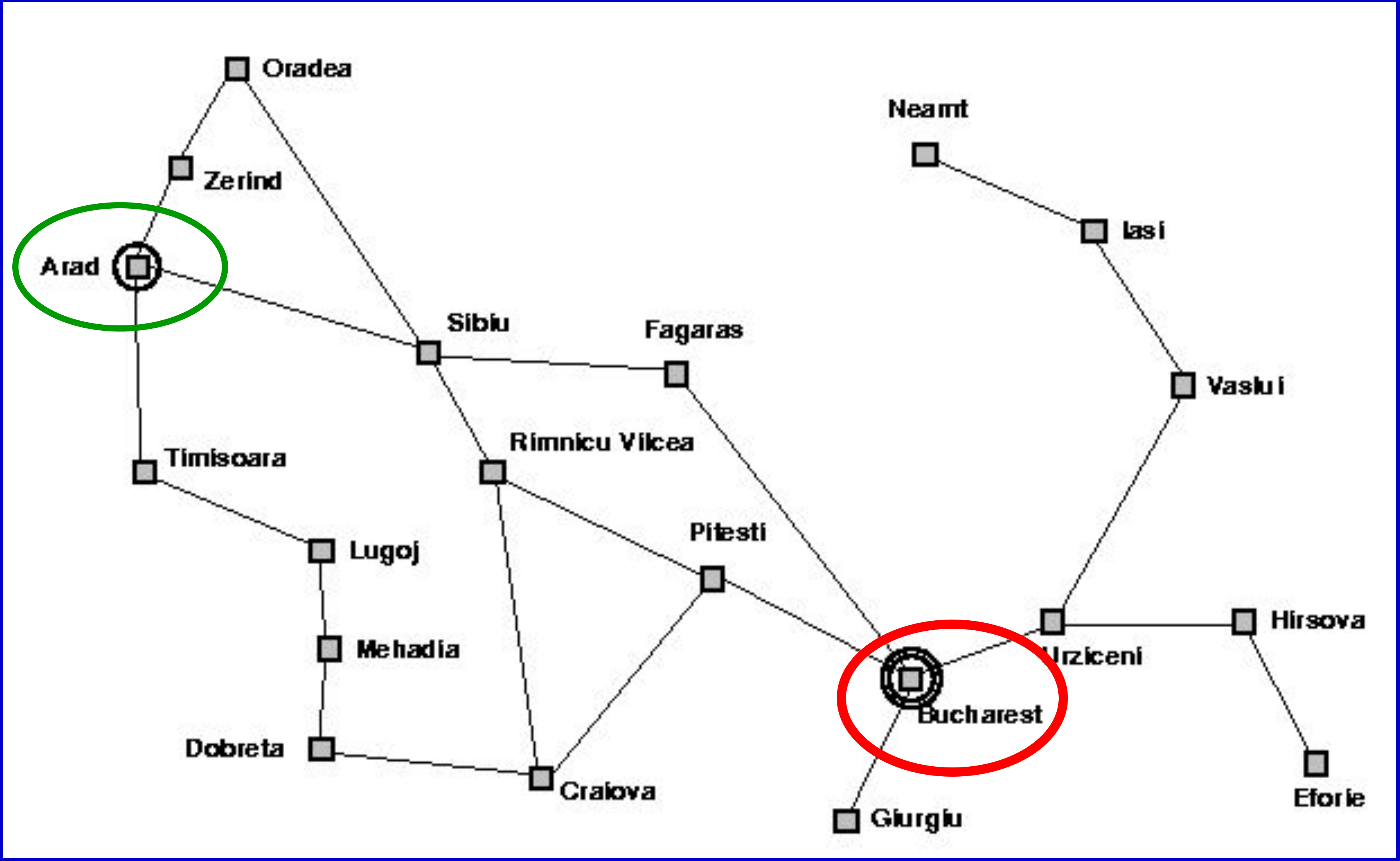
The last two cannibals cross



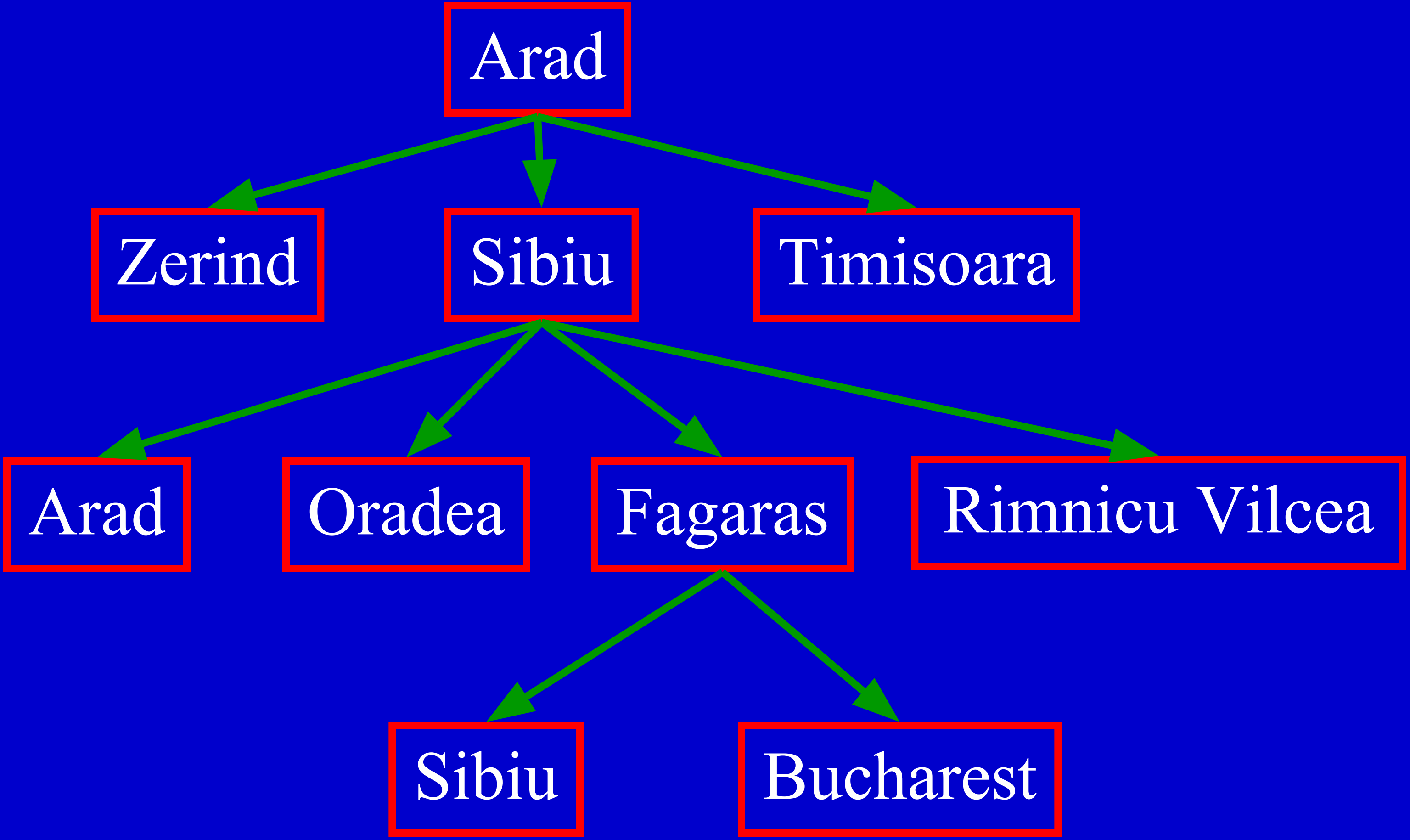
Missionaries and Cannibals

(0,0,0)

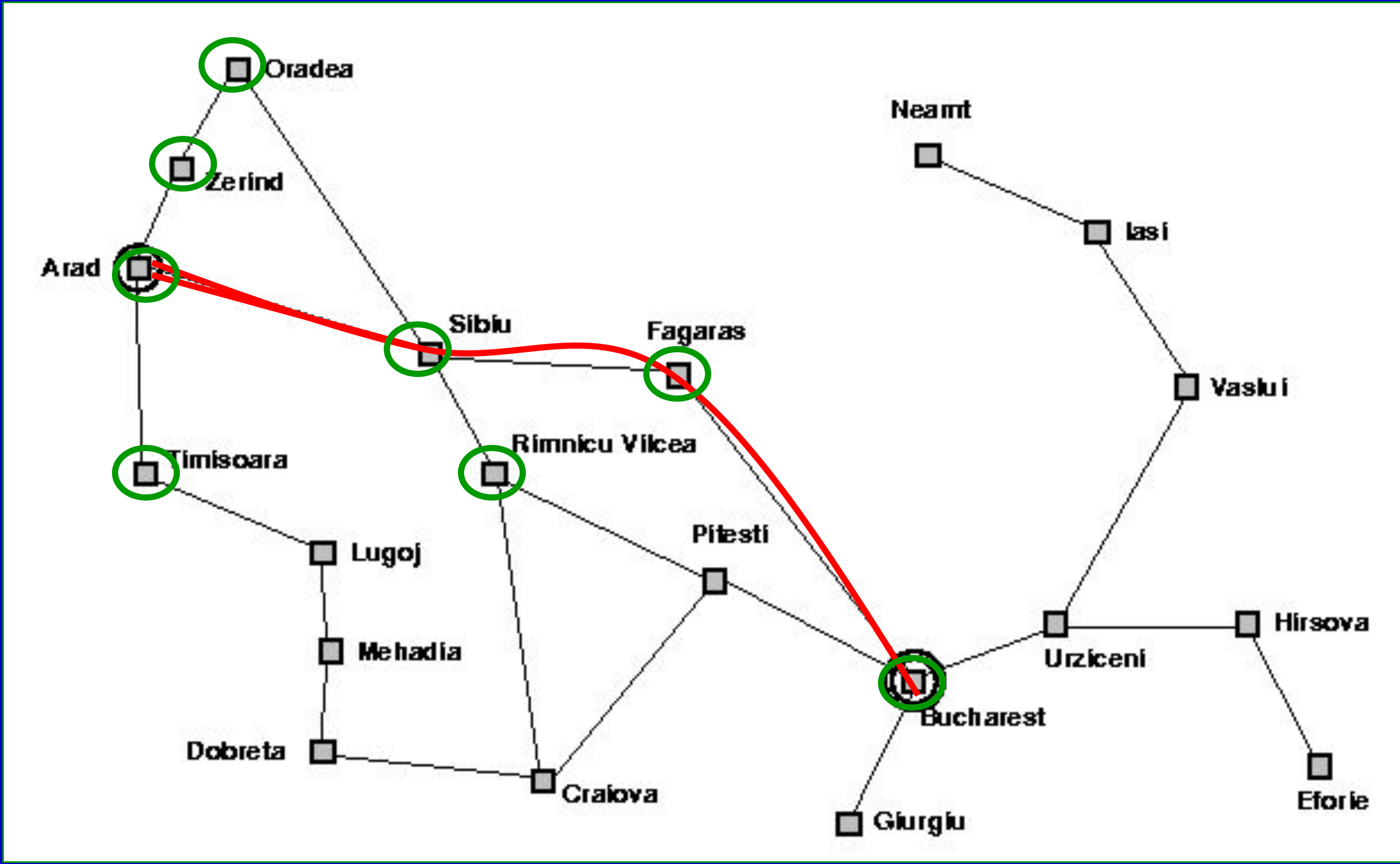
The state space



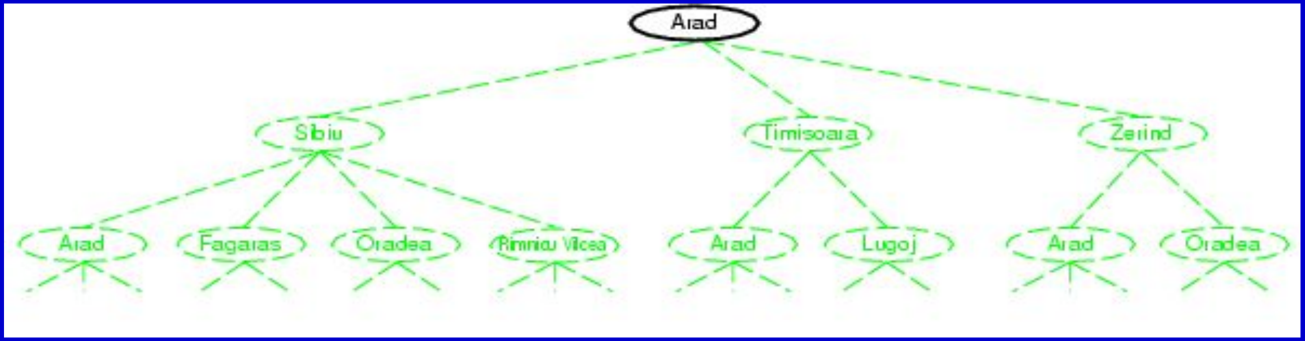
General Search Example



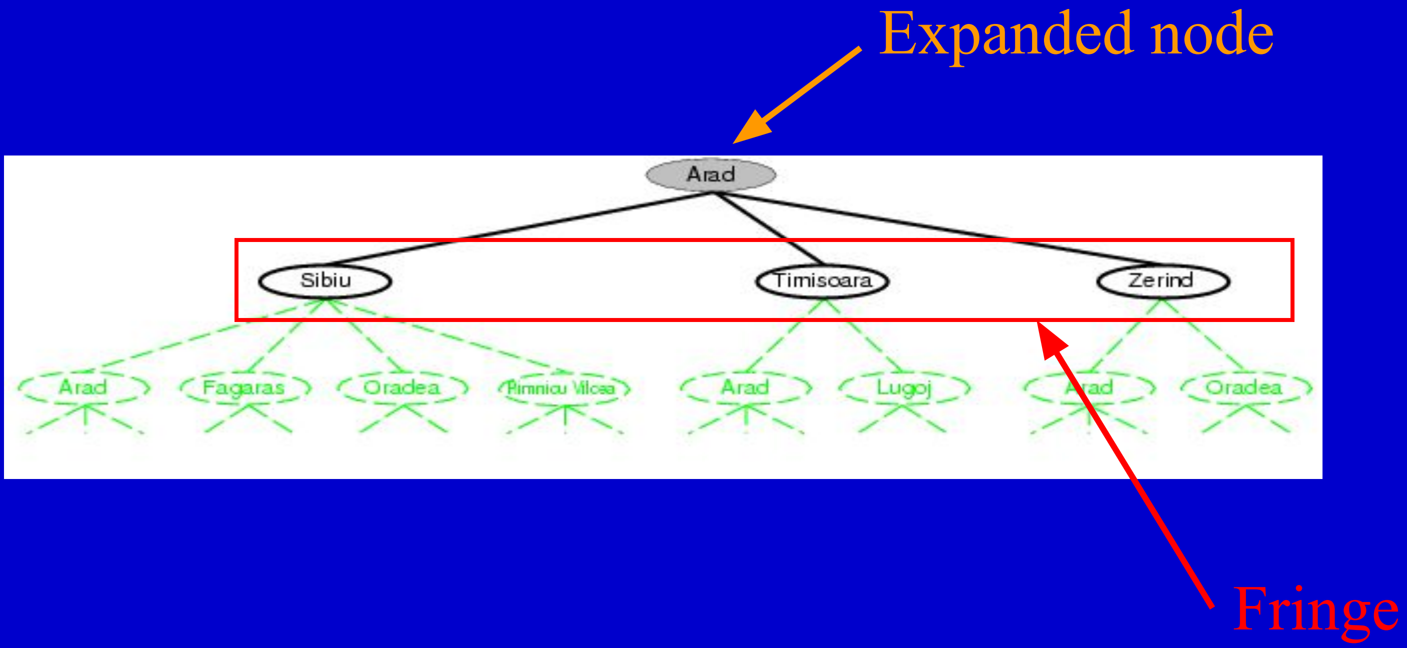
The solution



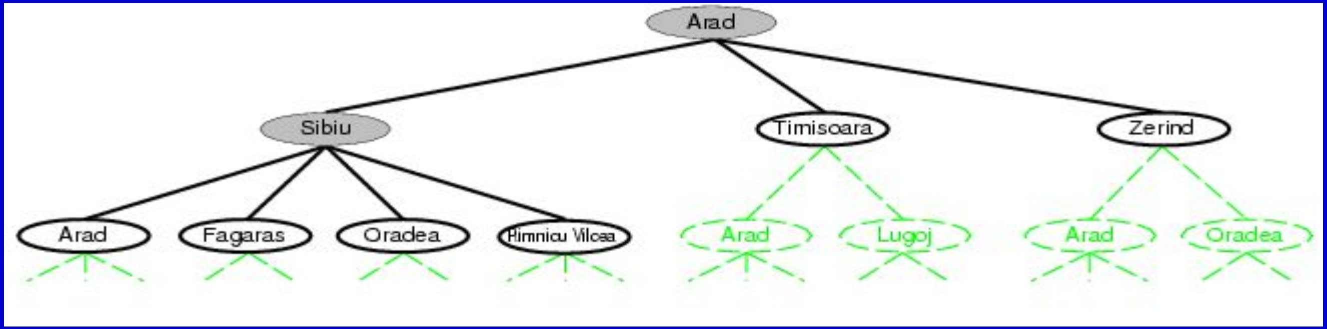
Tree search example



Tree search example



Tree search example



States vs. nodes

- A *state* is a (representation of a) physical configuration.
- A *node* is a data structure constituting part of a search tree includes *parent, children, depth, path cost g(n)*.
- *States* do not have parents, children, depth, or path cost!

Search Strategies

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

Strategies are evaluated along the following dimensions:

- **completeness** – does it always find a solution if one exists?
- **optimality** – does it always find a least-cost solution?
- **time complexity** – number of nodes generated/expanded
- **space complexity** – maximum number of nodes in memory

Time and space complexity are measured in terms of:

b	maximum branching factor of the search tree
d	depth of the least-cost solution
m	maximum depth of the state space (may be infinite)

Search Strategies

Uninformed (blind) strategies use only the information available in the problem definition.

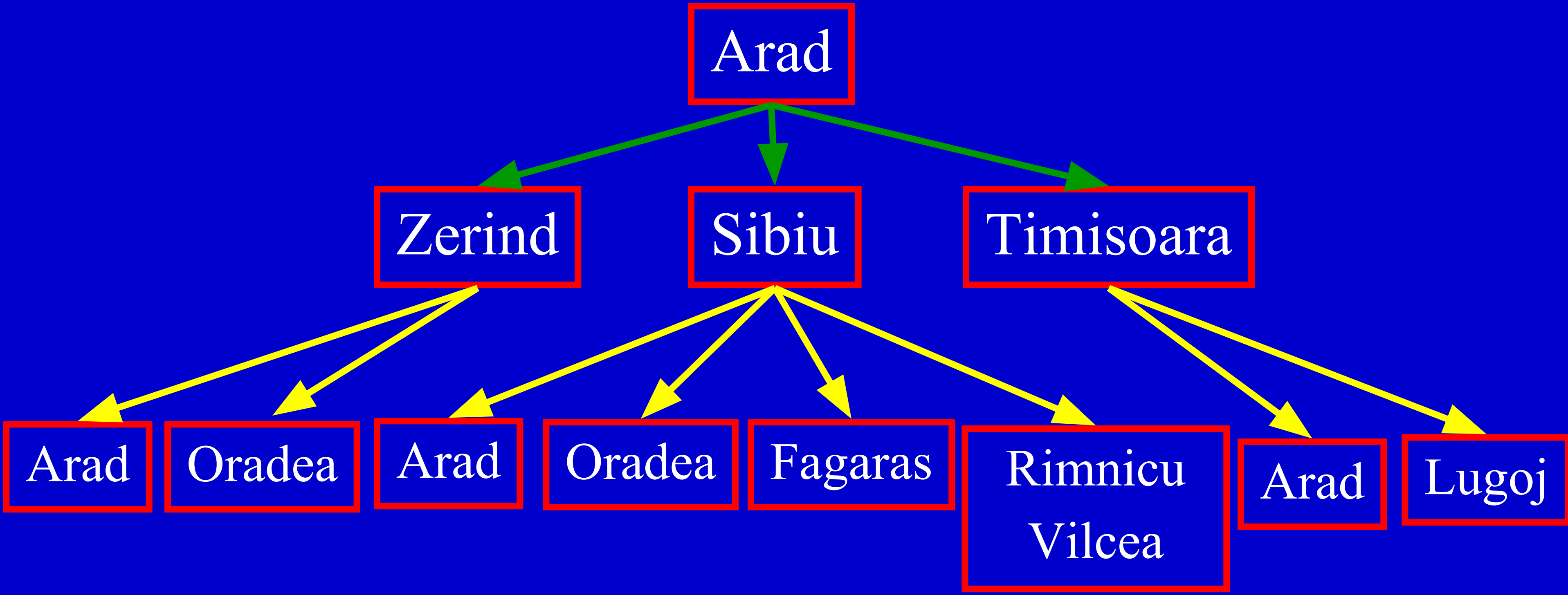
Informed search techniques which might have additional information (e.g. a compass).

- Breadth-first search
- Depth-first search
- Iterative deepening search

Breadth-first Search

- Expand shallowest unexpanded node
- Implementation:

  QueueingFn = put successors at end of queue.

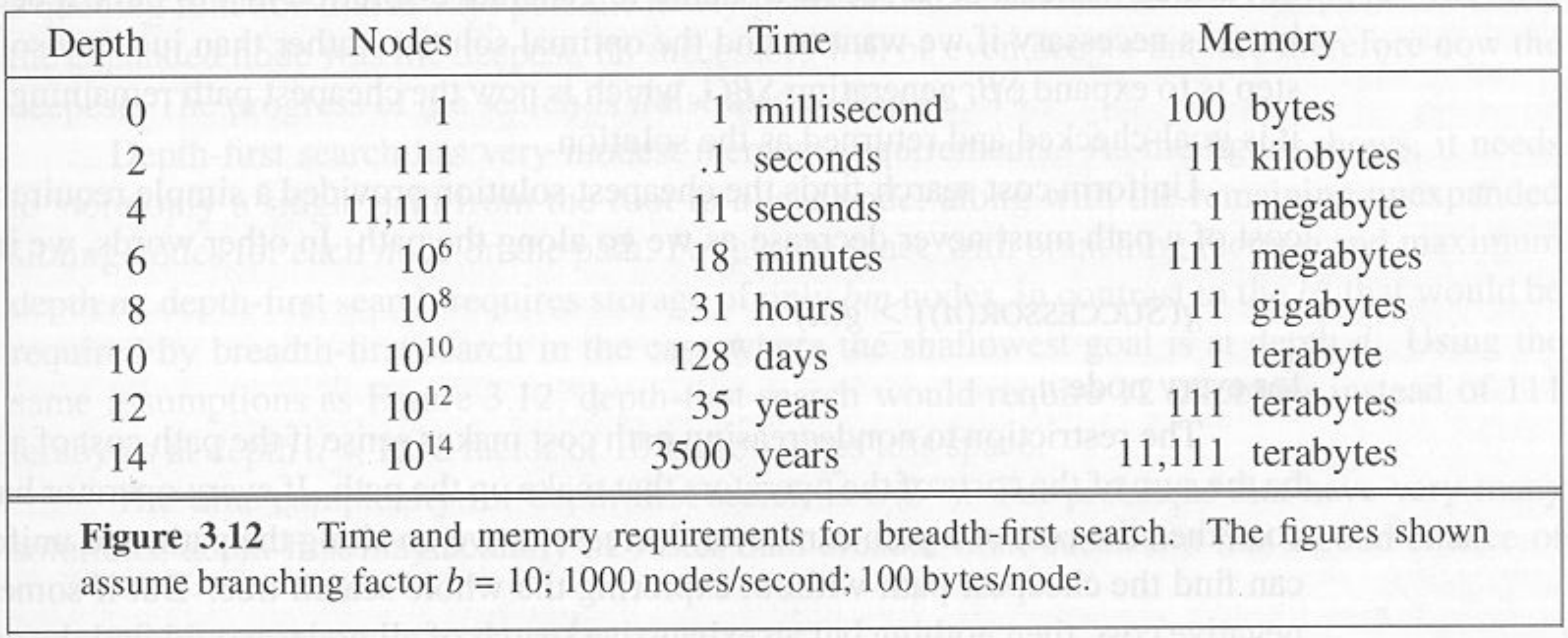


Properties of Breadth-first Search

- Complete: Yes (if *b* is finite)
- Optimal: Yes (if cost = 1 per step); not in general
- Time: $1 + b + b^2 + b^3 + b^4 + \ldots + b^d = O(b^d)$
- Space: $O(b^d)$ -- Keeps every node in memory

Let b: Branching factor
d: Solution depth
m: Maximum depth

Breadth-First Search: Time & Memory

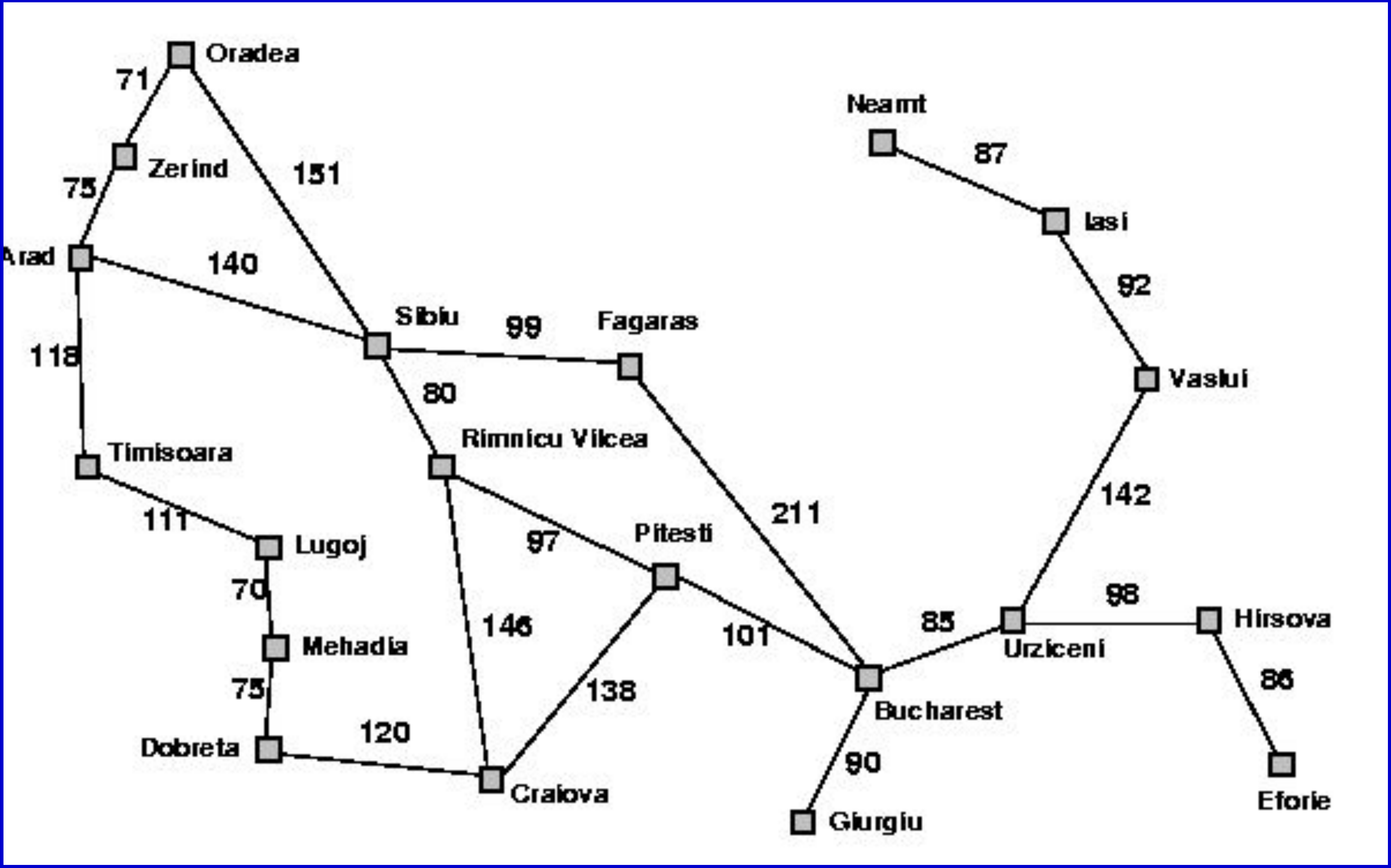


Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Figure 3.12 Time and memory requirements for breadth-first search. The figures shown assume branching factor $b = 10$; 1000 nodes/second; 100 bytes/node.

- Branching (b) =10
- 1000 nodes per second
- 100 bytes per node

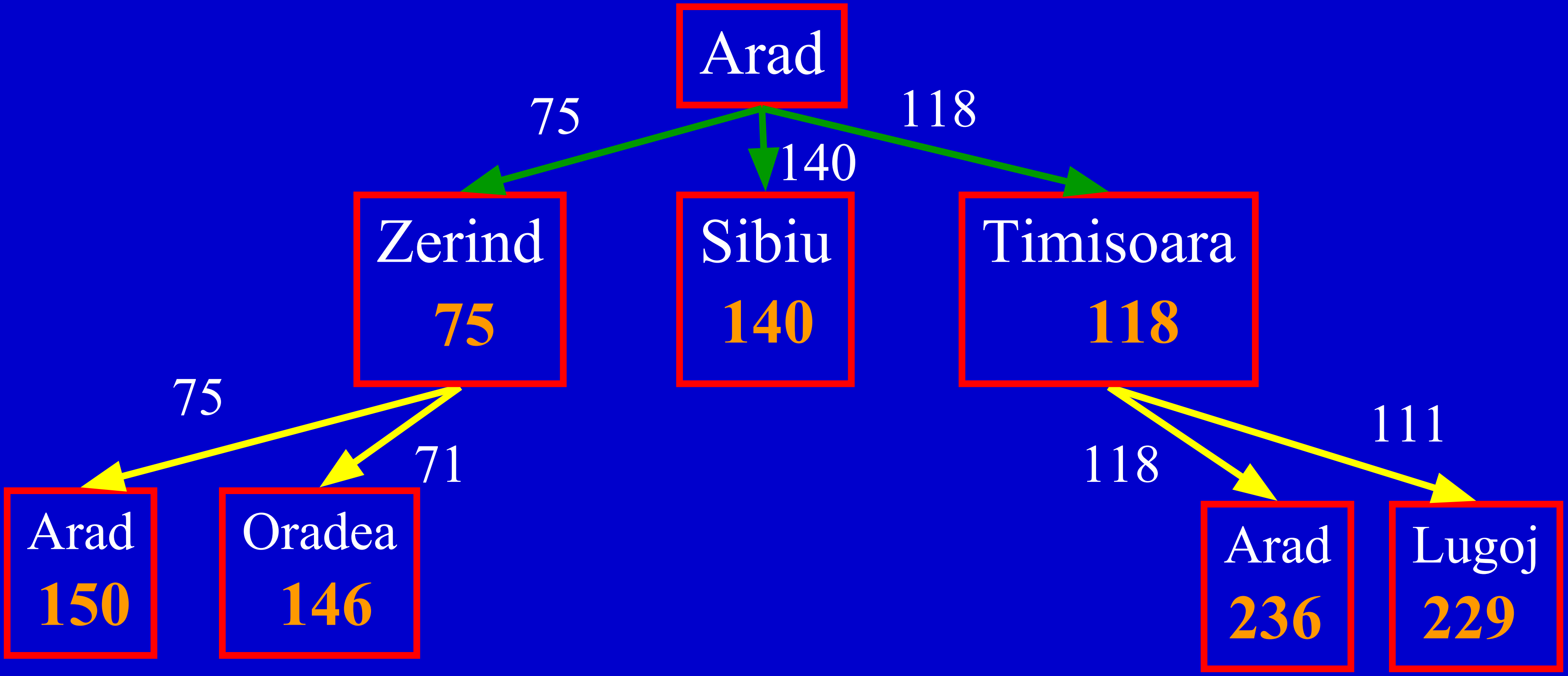
Romania with Edge Costs



Uniform-cost Search

- Let g(n) be path cost of node n.
- Expand least-cost unexpanded node
- Implementation:

  QueueingFn = insert in order of increasing path length



Properties of Uniform-Cost Search

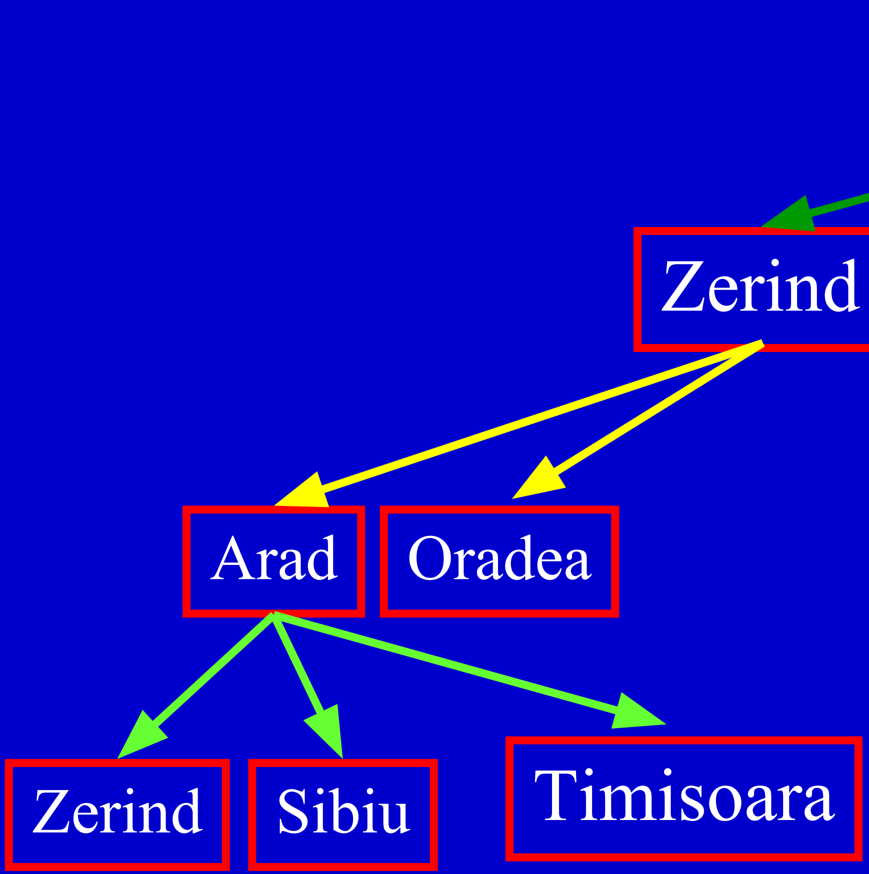
- Complete: Yes if arc costs bounded below by $\varepsilon > 0$.
- Optimal: Yes
- Time: # of nodes with $g(n) \leq$ cost of optimal solution
- Space: # of nodes with $g(n) \leq$ cost of optimal solution

Note: Breadth first is equivalent to Uniform-Cost Search with edge cost equal a constant.

Depth-First Search

- Expand deepest unexpanded node
- Implementation

  QueueingFn = insert successors at front of queue



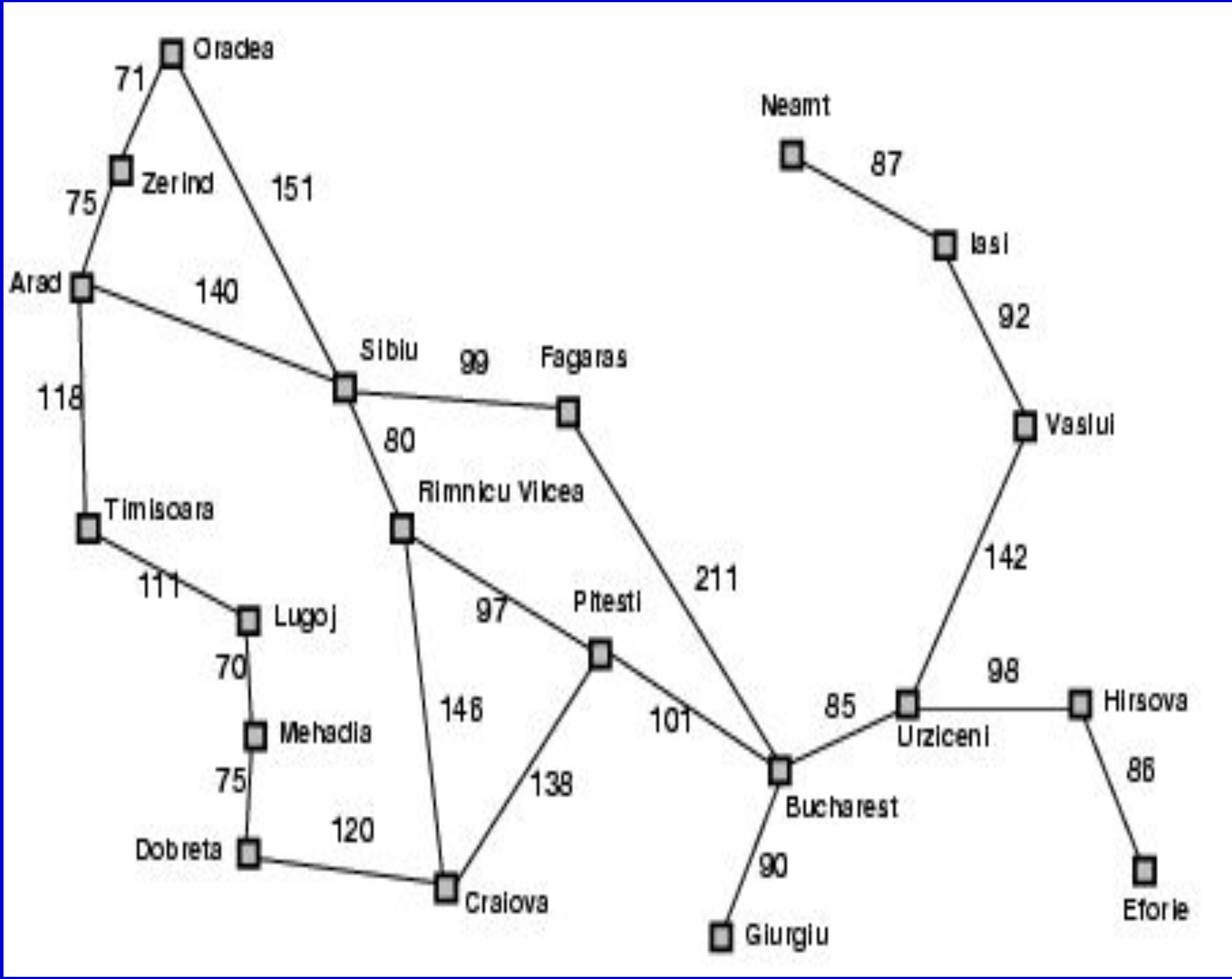
Note: Depth-first search can perform infinite cycle excursions. Need a finite, non-cyclic search space or repeated-state checking.

Properties of Depth-First Search

- Complete: No: Fails in infinite-depth spaces, spaces with loops. Modify to avoid repeated states on path
   Complete in finite spaces
- Optimal: No.
- Time: $O(b^m)$: terrible if m is much larger than d. but if solutions are dense may be much faster than breadth first.
- Space: $O(bm)$ i.e. linear in depth.

Let b: Branching factor
m: Maximum Depth

Romania with step costs in km

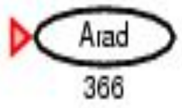


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

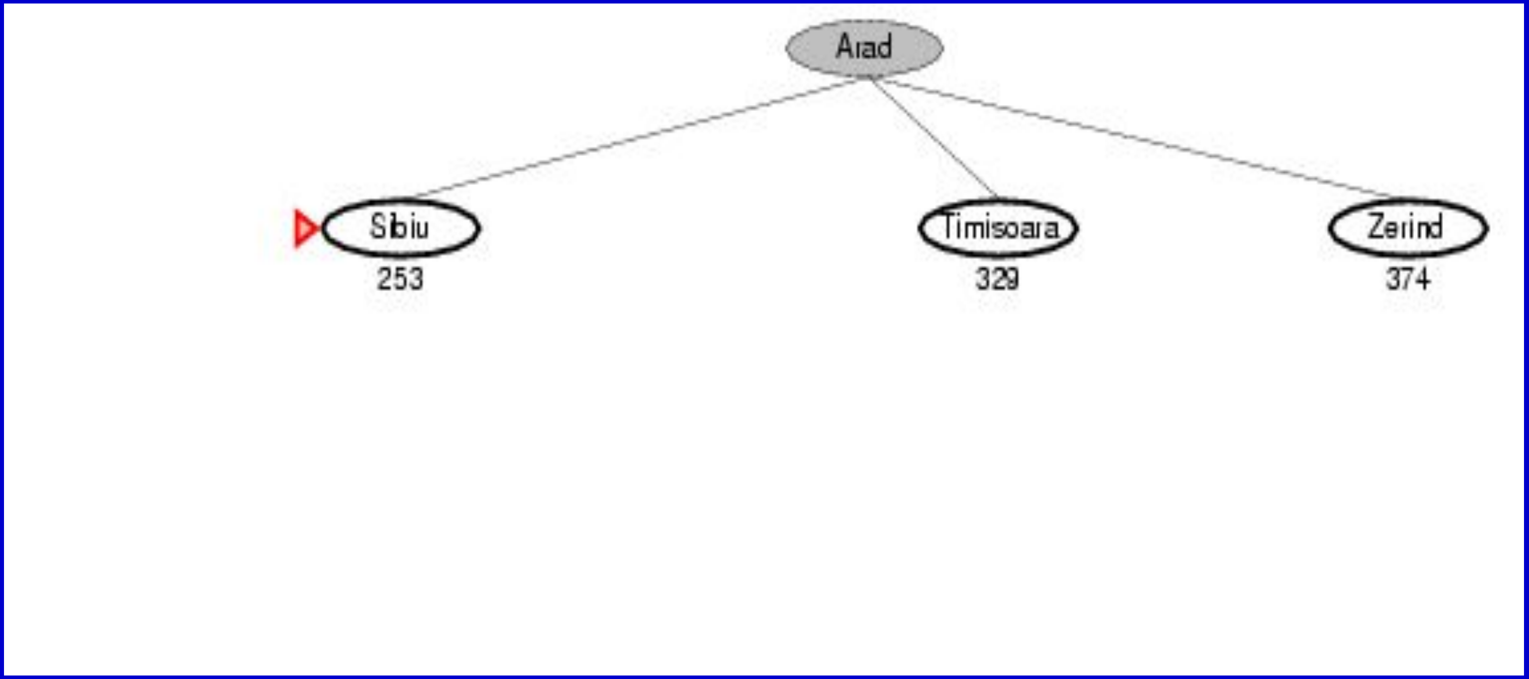
Greedy best-first search

- Evaluation function *f(n) = h(n)* (heuristic)
- h(n)= estimate of cost from *n* to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

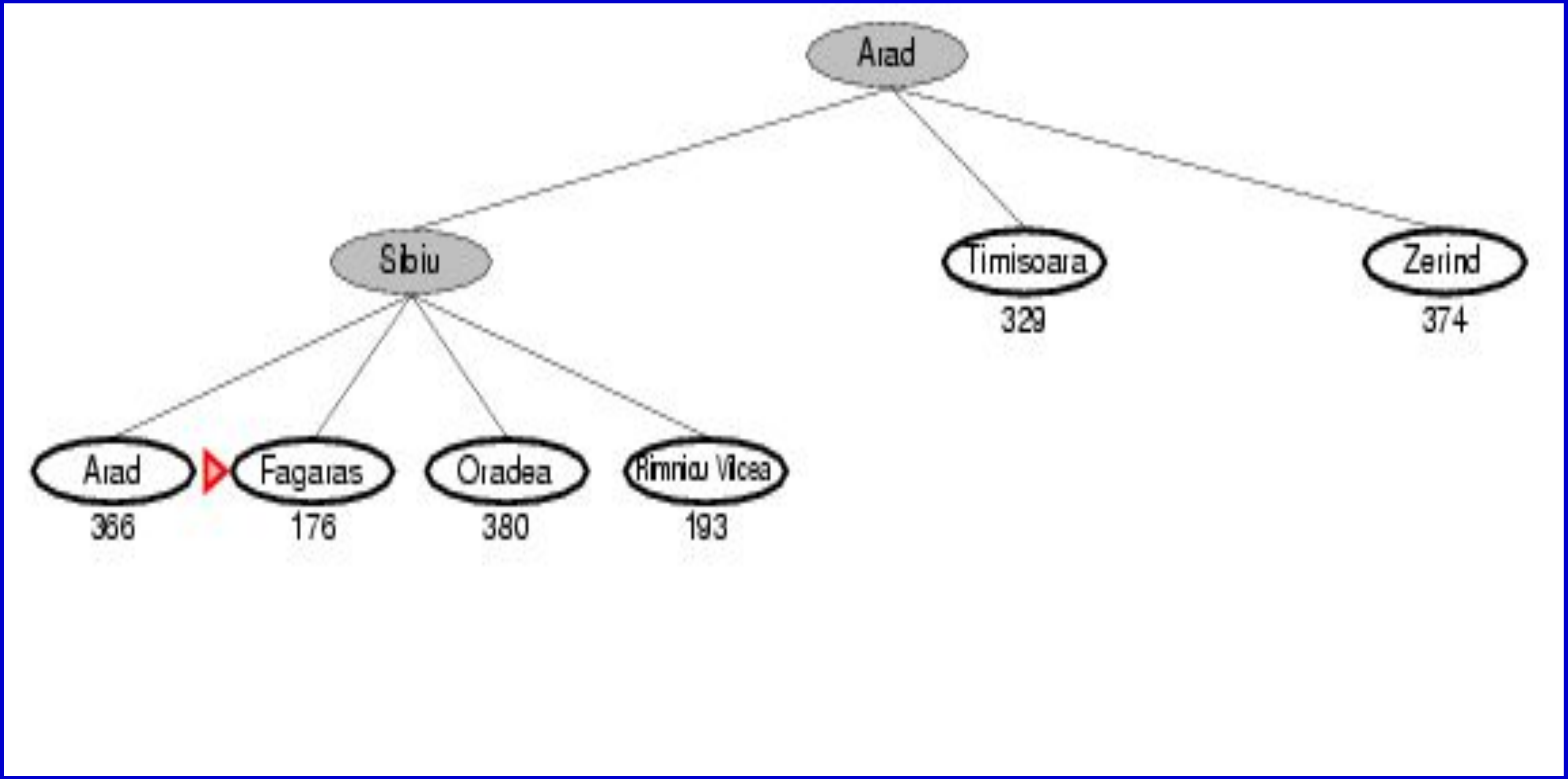
Greedy best-first search example (Arad to Bucharest)



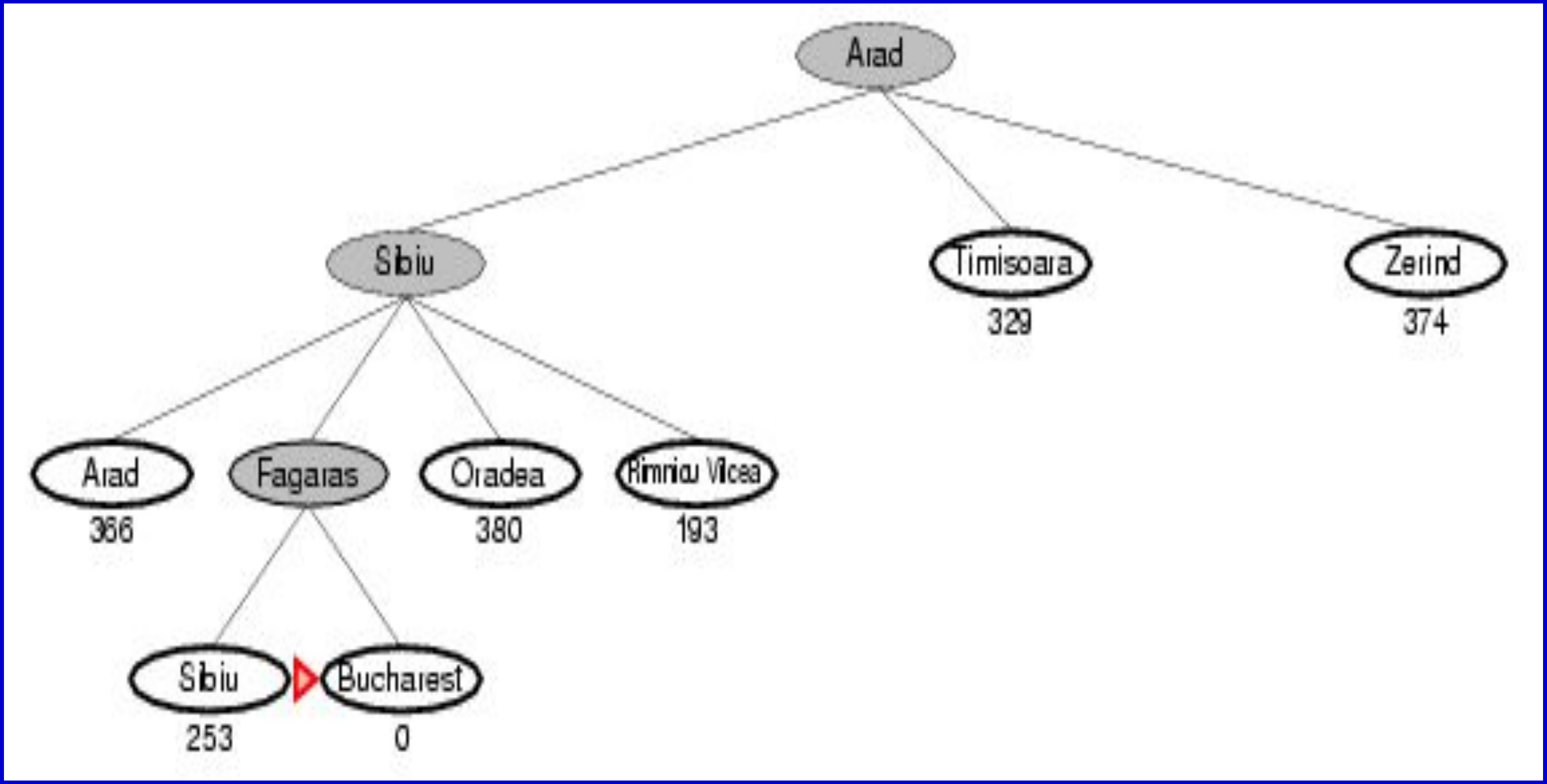
Greedy best-first search example (Arad to Bucharest)



Greedy best-first search example (Arad to Bucharest)



Greedy best-first search example (Arad to Bucharest)



Some points of the example

- For this particular problem, greedy best search using h_{SLD} finds a solution without ever expanding a node that is not on the solution path
- *Its search cost is minimal*
- *It is not optimal*

Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi □ Neamt □ Iasi □ Neamt □
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory

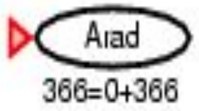
- Optimal? No

A^* search

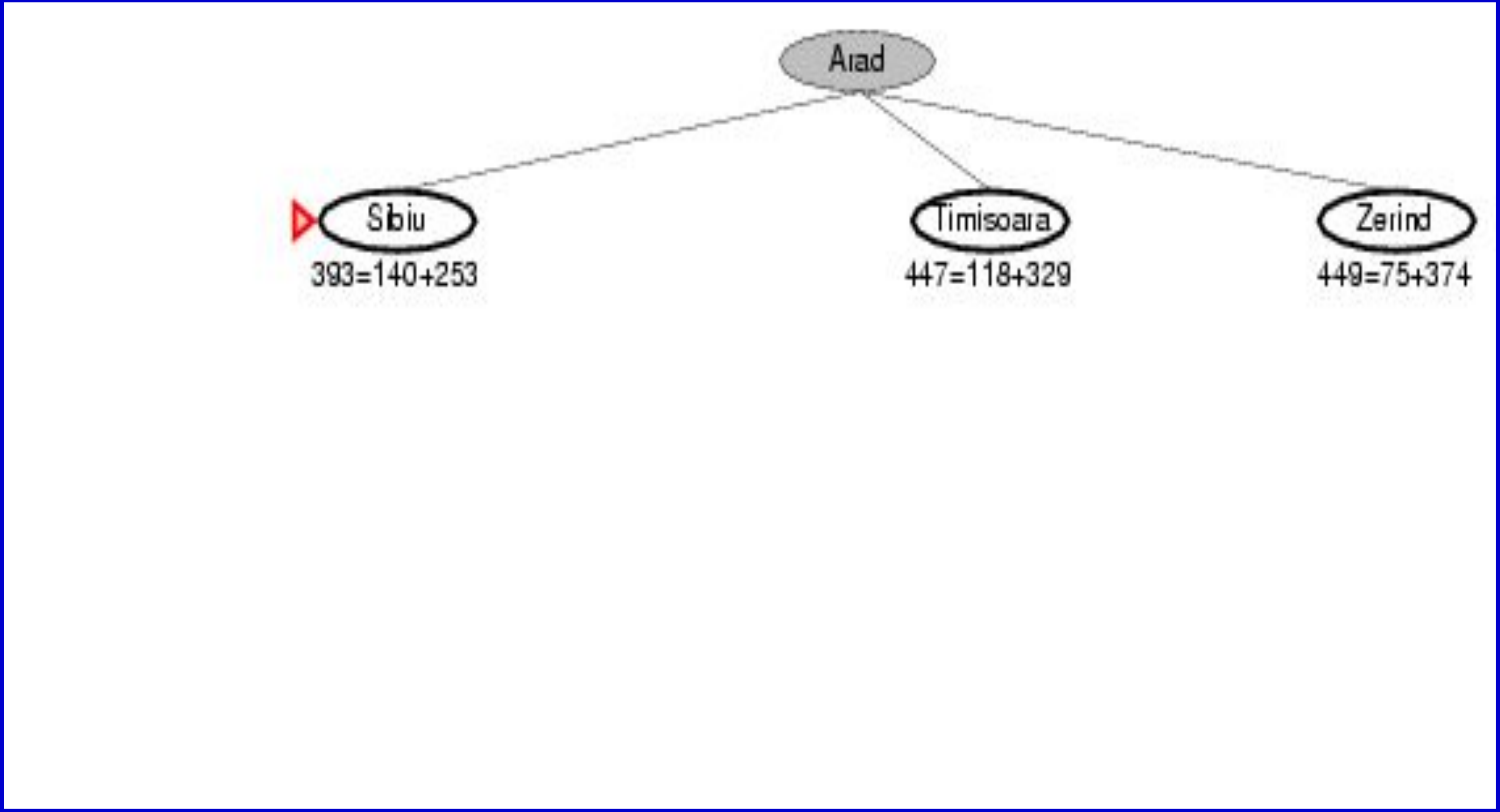
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$

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

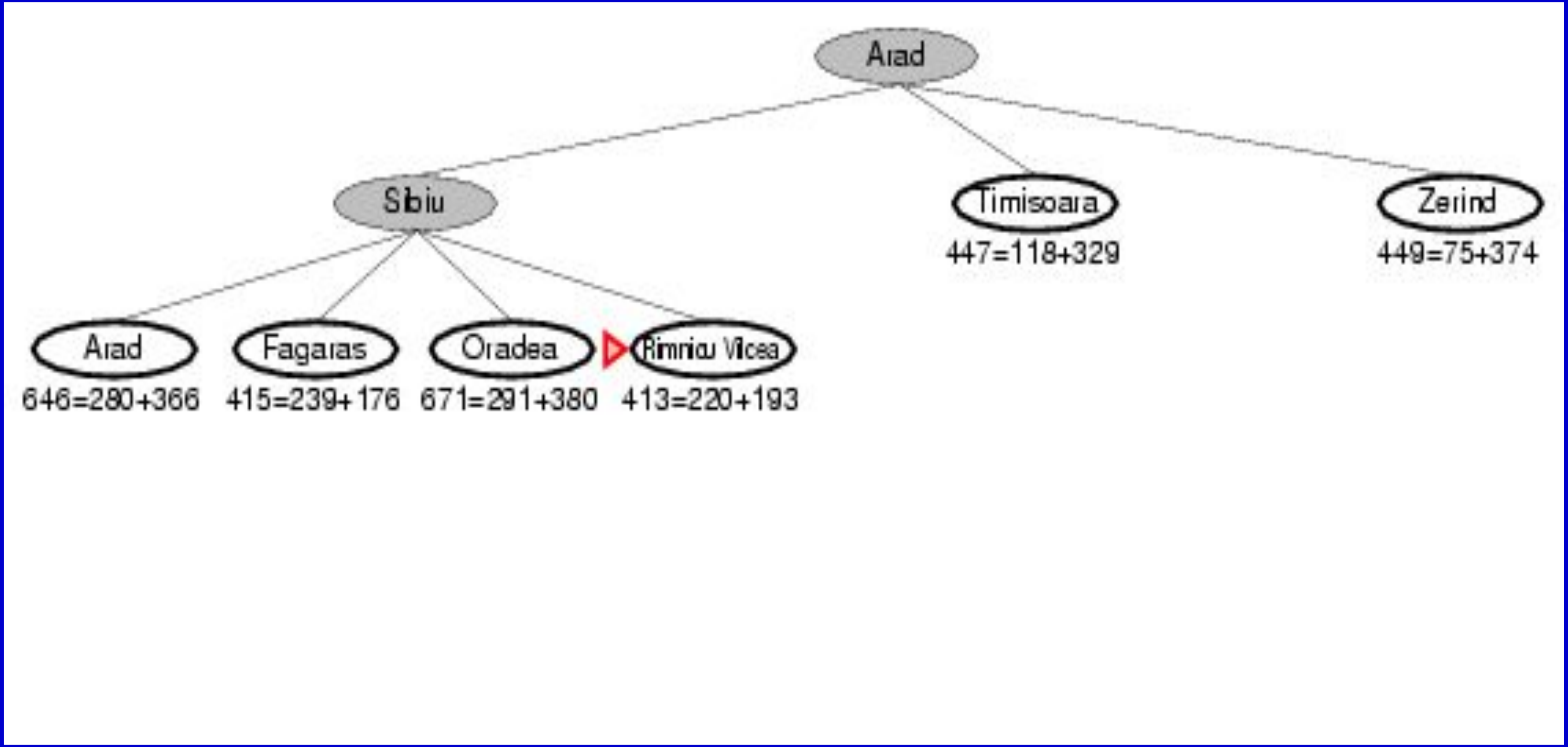
A^* search example



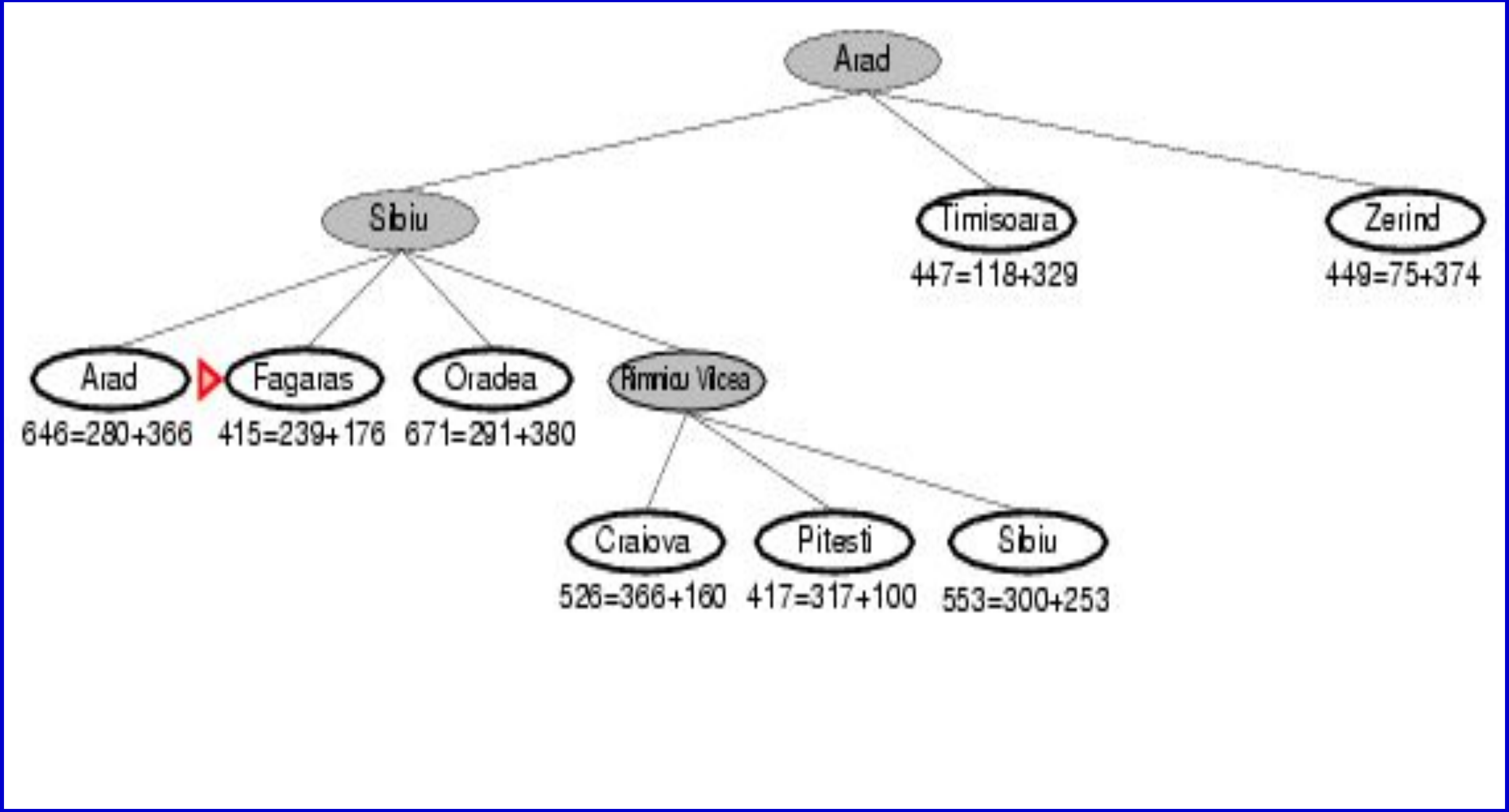
A* search example



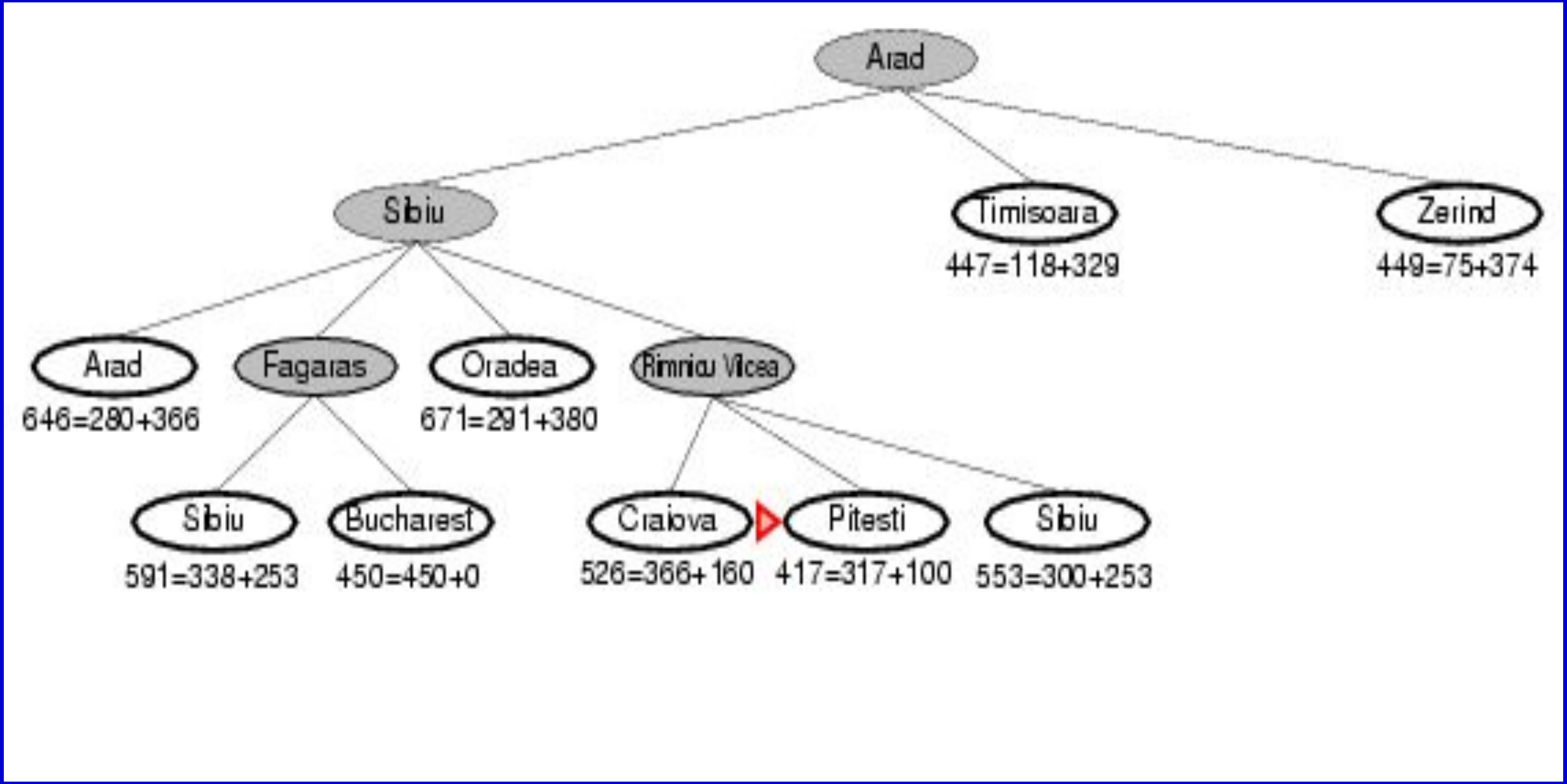
A* search example



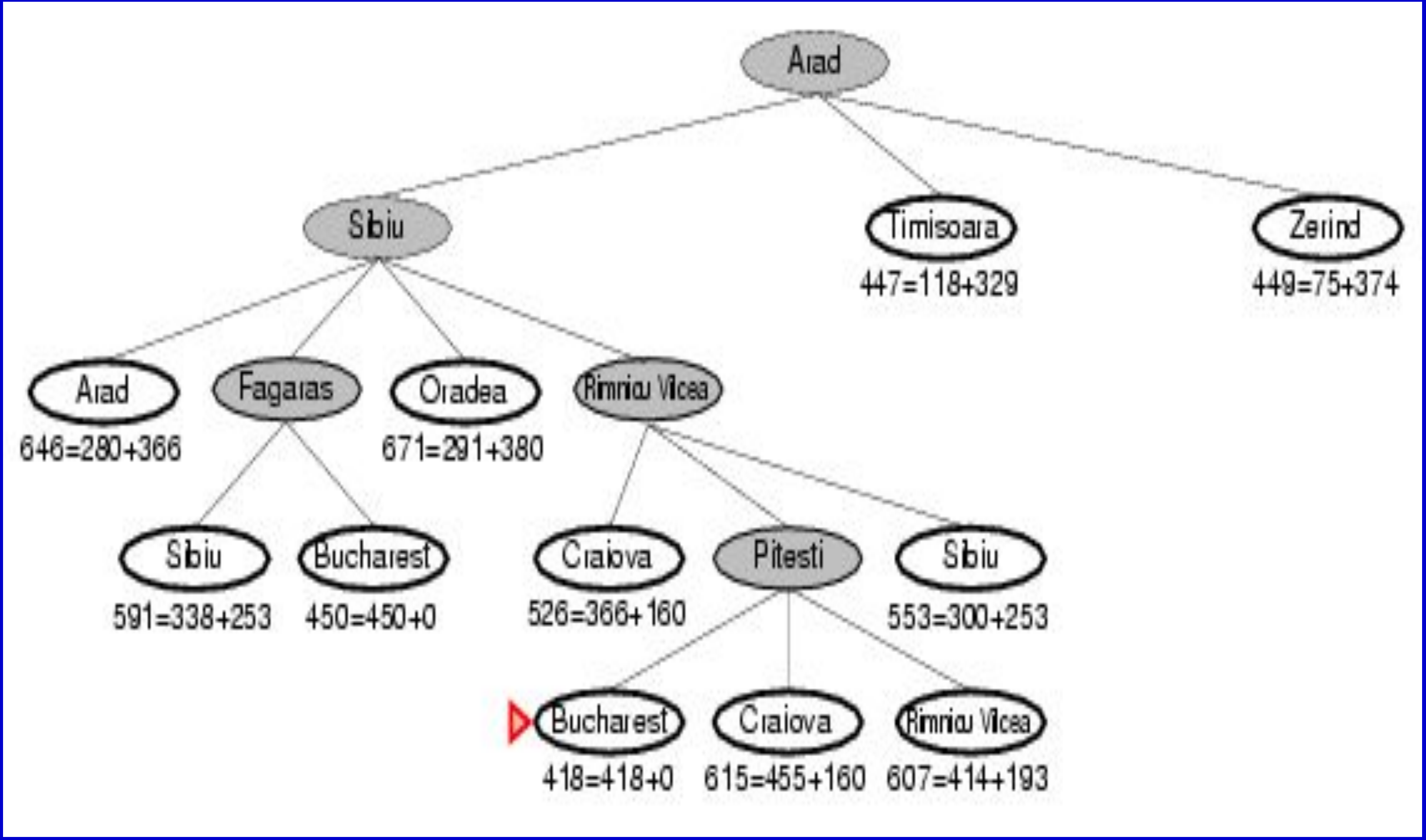
A* search example



A* search example



A* search example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193