Introduction to AI
-Search-

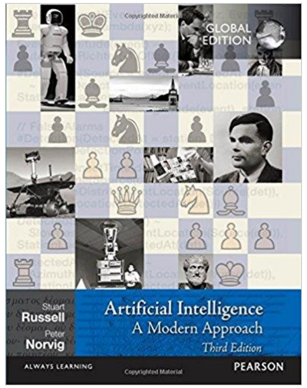
Francesca Toni

Outline

- Solving problems by search
- Basic search algorithms (uninformed/blind search)
- Heuristic-based search algorithms (informed search)
- Search for games (adversarial search)

Recommended reading:

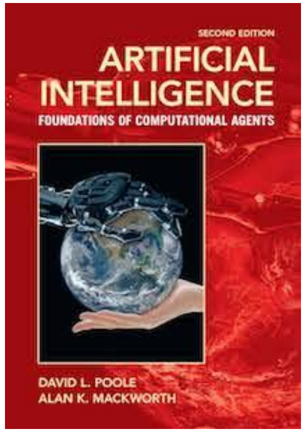
(most of) Chapters 3, 5



Additional reading:

Chapter 3

Section 11.2.2



Solving problems by search

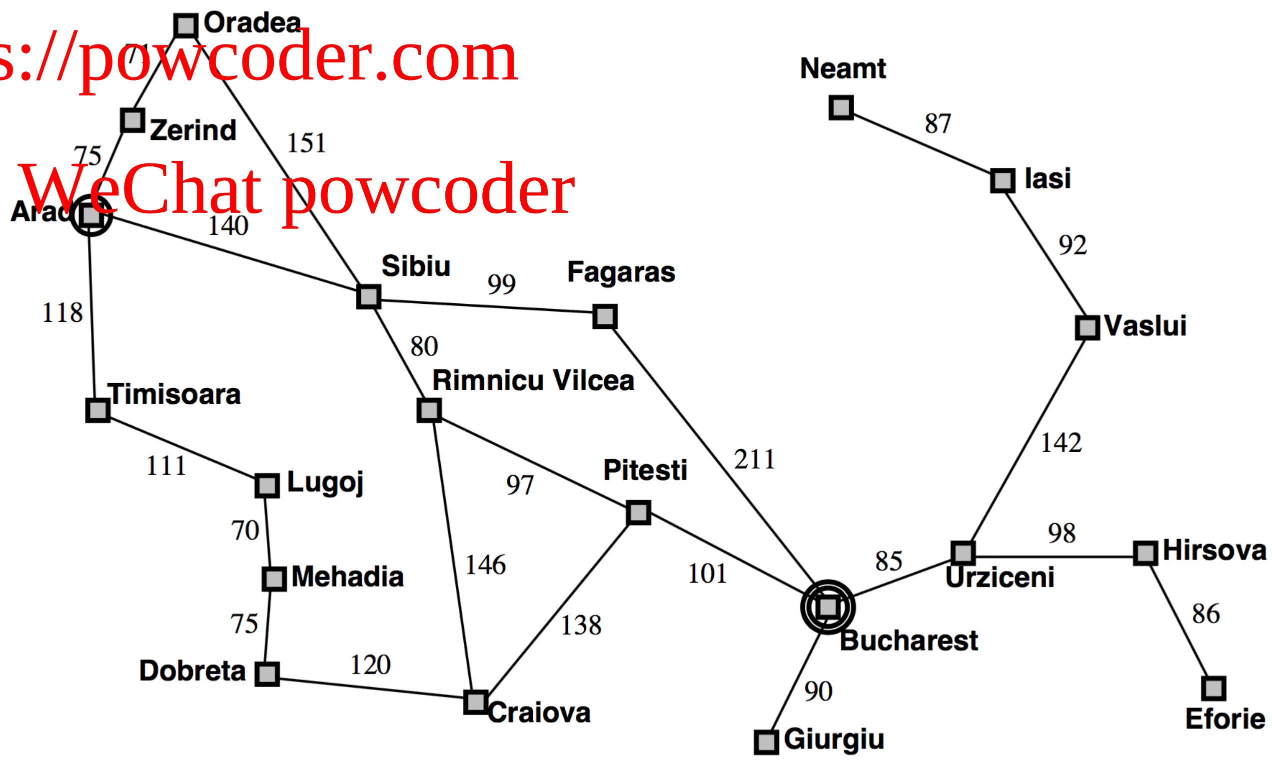
- Problem solution can be abstracted as **path** from some (**start**) node to some (**goal**) node in a suitably defined **directed graph**
- Search algorithms to find **(optimal)** paths

Route finding example

On holiday in Romania; today in Arad; flight leaves tomorrow from Bucharest

- Start: Arad
- Goal: Bucharest

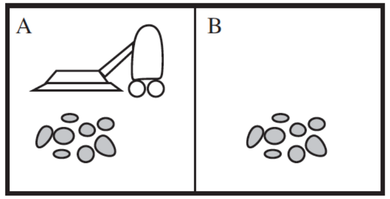
- Graph



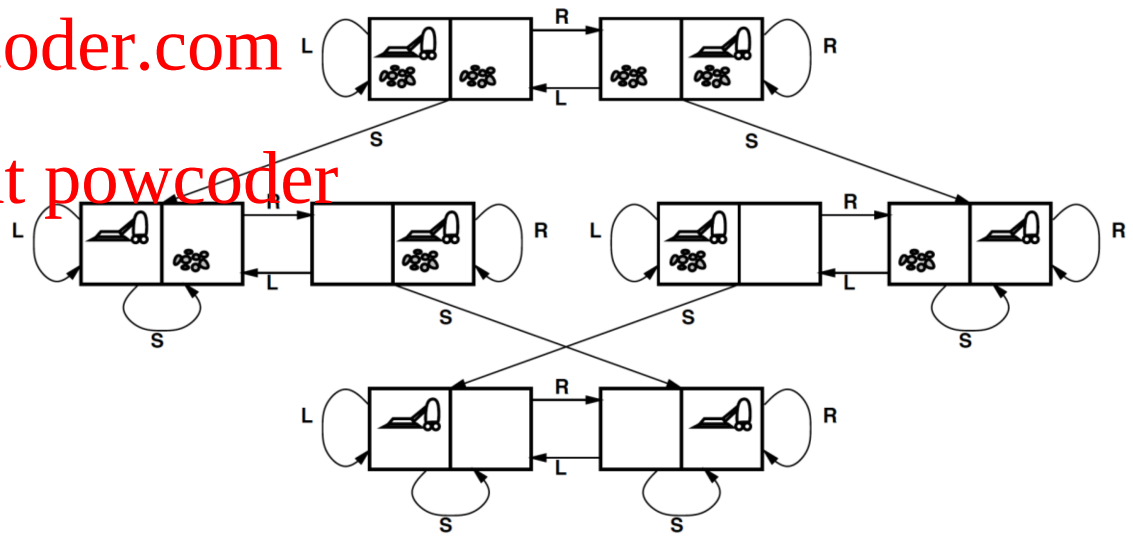
- Optimal path
  - shortest, or
  - quickest, or
  - cheapest

Vacuum world example

The vacuum cleaner needs to remove all dirt from the room

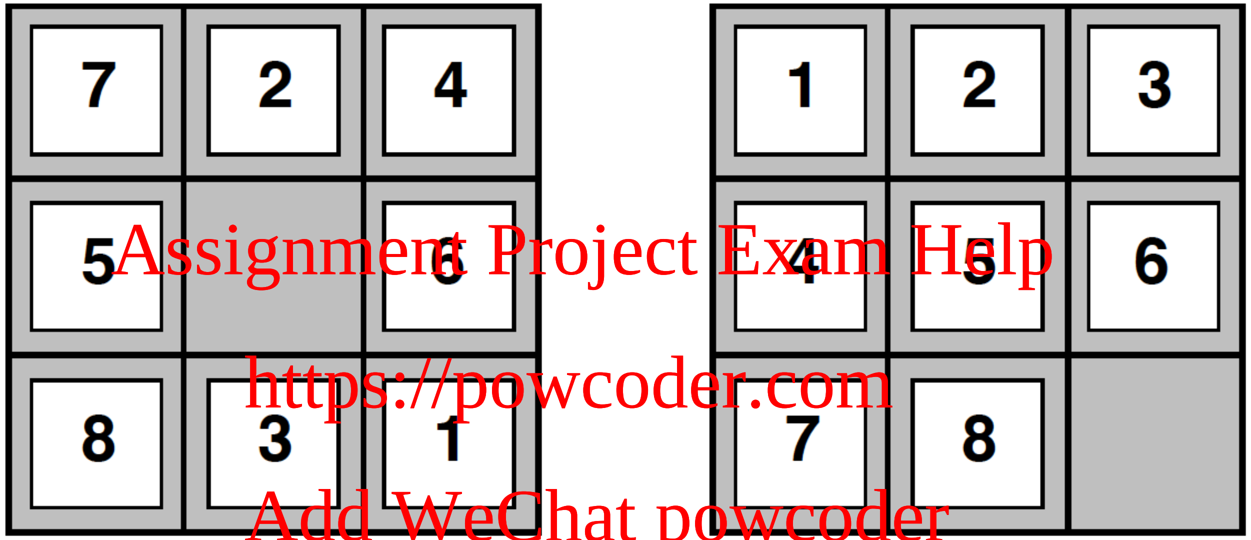


- Start: A
- Goal: no dirt anywhere in A, B
- Graph



- Optimal path
  - shortest, or
  - quickest, or
  - cheapest

8-puzzle example



Start Goal

- Optimal path?
- Graph?

Which problems can be solved by (classical) search?

Environment is

- Observable
  - Current state known
- Discrete
  - From each state finitely many next states (by executing actions)
- Deterministic
  - Each action has exactly one outcome (next state)
- Known
  - Possible actions and next states for each state

Search problems – formally

- *Initial state* (**start**) e.g. *In(Arad)*
- *Transition model* between states (**graph**)

  possibly given by *successor function* RESULT: States×Actions → States

  e.g. RESULT (*In(Arad)*, *Go*(*Zerind*))=*In(Zerind)*

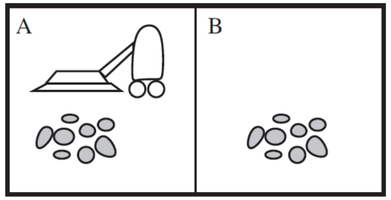
- *Goal test* (set of **goal** states)
  - explicit e.g. *In(Bucharest)*, or
  - implicit e.g. $\forall x \; \neg Dirt(x)$

- *Path cost* (additive = sum of positive *step costs*)

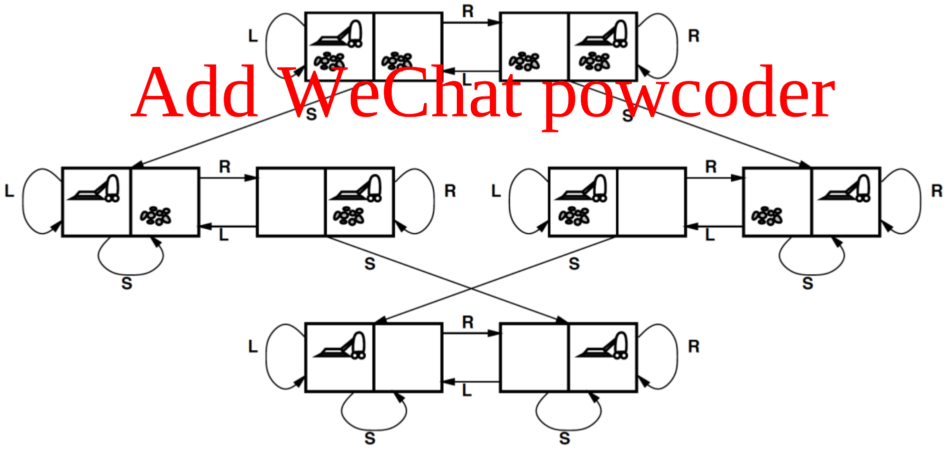
  e.g. sum of distances, number of actions

Solution=path (sequence of actions) from the initial state to a goal state
Optimal solution= solution with lowest path cost

Vacuum world example – formally

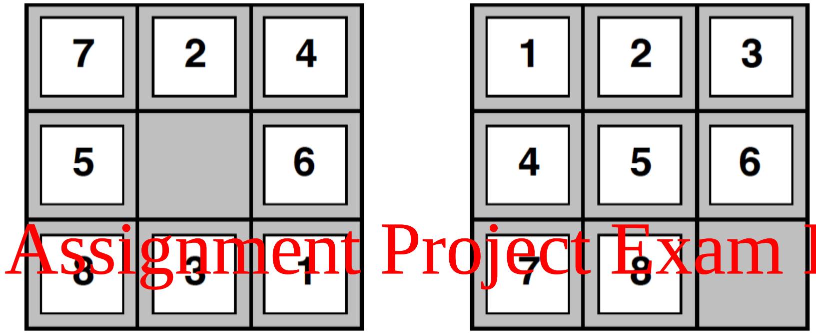


- Initial state= $In(A) \wedge Dirt(A) \wedge Dirt(B)$
- Goal test= $\{\neg Dirt(A) \wedge \neg Dirt(B)\}$
- RESULT $(In(A) \wedge Dirt(A) \wedge Dirt(B), S) = In(A) \wedge \neg Dirt(A) \wedge Dirt(B), etc$



- Step cost=1, path cost=number of actions

8-puzzle example – formally



Start Goal

- *Initial state* = Start – e.g. represented by integer locations of tiles
- *Transition model* between states?

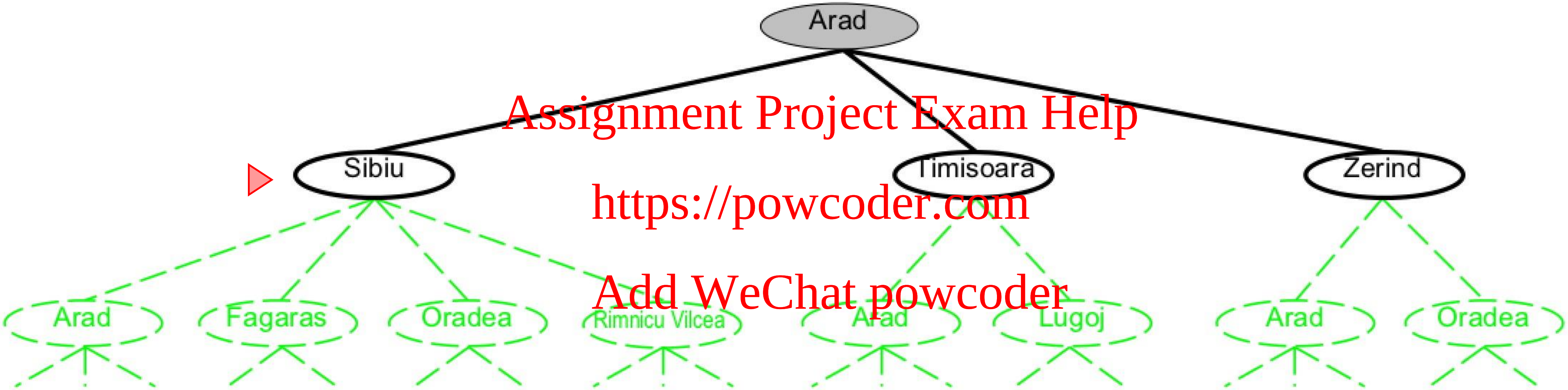
  (*successor function* RESULT: States×Actions → States ?)
- *Goal test* ={Goal} – e.g. represented by integer locations of tiles
- *Path cost*= number of actions

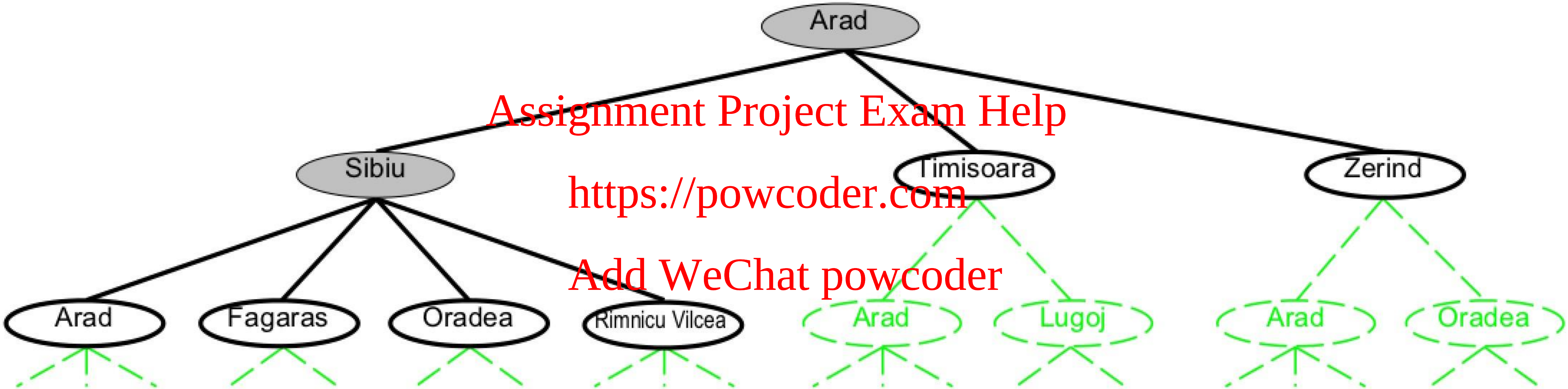
Search algorithms – basic idea

Generate a (search) tree:

1. Initialise the tree with the initial state as root (=frontier)
2. Repeat
   i. if the frontier of the tree is empty then return fail
   ii. **choose** and remove a leaf node L from the frontier
   iii. if L is a goal state (in the goal test) then return corresponding solution
   iv. expand L, adding the resulting nodes to the frontier


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Search algorithms – basic idea with loop checking

Generate a (search) tree:

1. Initialise the tree with the initial state as root
2. Repeat
   i. if the frontier of the tree is empty then return fail
   ii. **choose** and remove a leaf node L from the frontier
   iii. if L is a goal state (in the goal test) then return corresponding solution
   *****add L to the "explored set"
   iv. expand L, adding the resulting nodes to the frontier
   *only if not in the frontier or the "explored set" already

* Additions to avoid loops

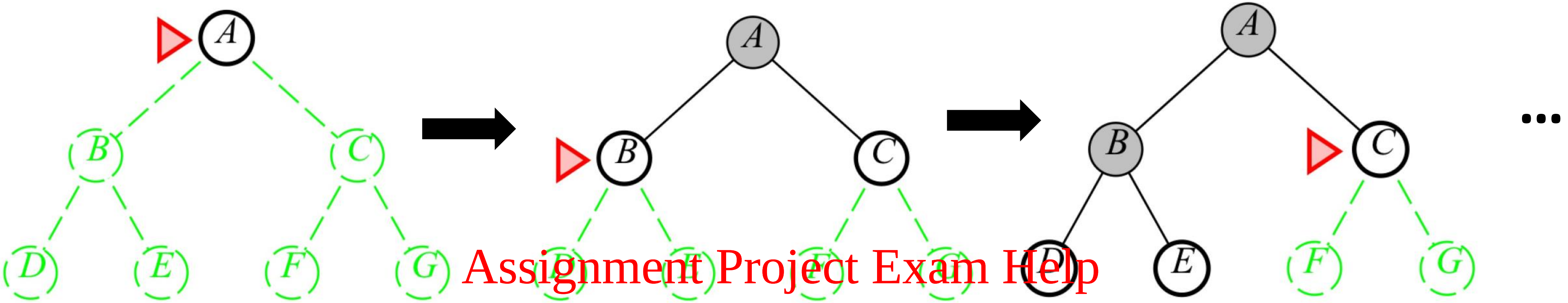
Search algorithms - concretely

Definition of **choose**

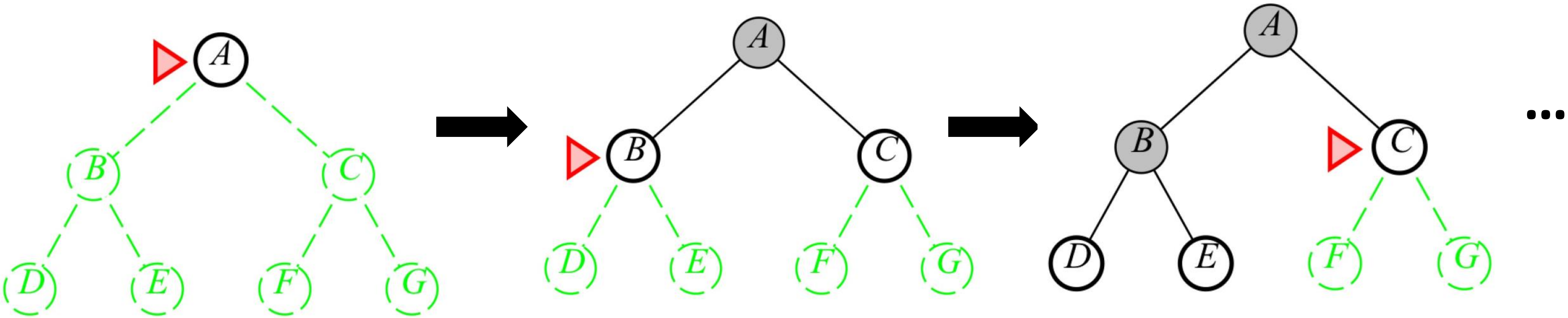
Uninformed/blind search

- Breadth-first search
  - **choose** (a) shallowest (closest to root) unexpanded node
- Uniform-cost search
  - **choose** unexpanded node with lowest path cost
- Depth-first search
  - **choose** deepest (farthest from the root) unexpanded node.
- Depth-limited search
  - depth-first search with given depth limit ℓ (nodes of depth ℓ have no children)
- Iterative deepening search
  - depth-limited search of increasing ℓ

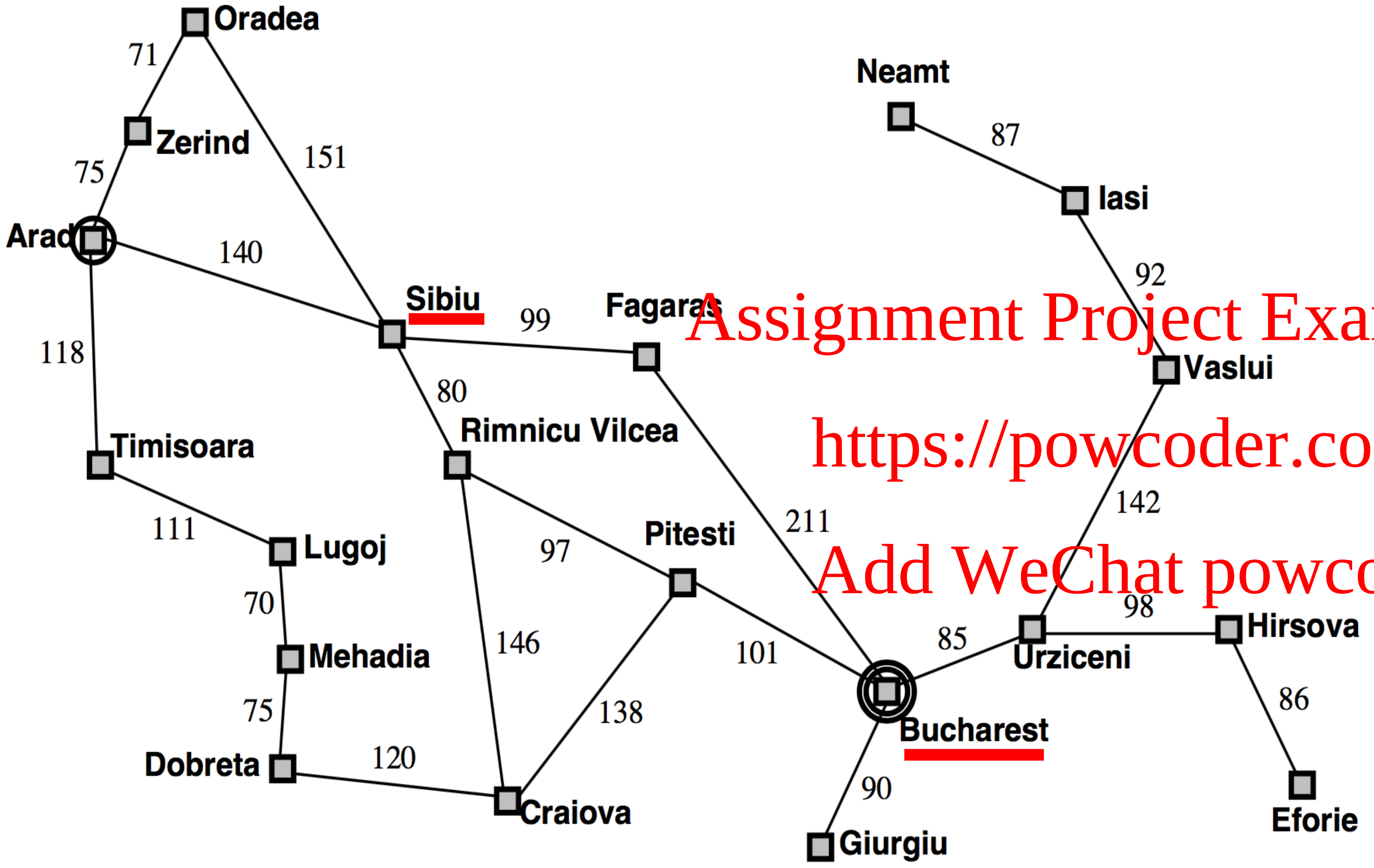
- Breadth-first (**choose** shallowest)

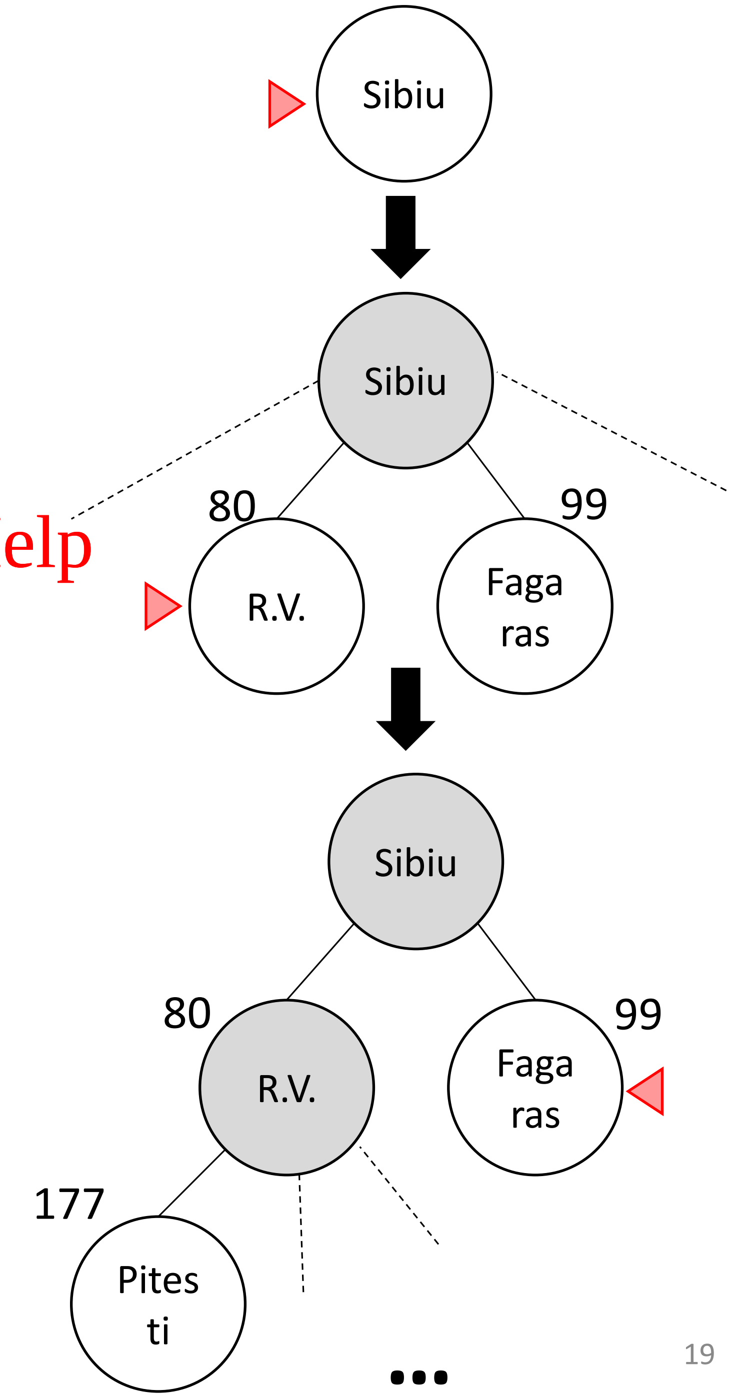


- Uniform-cost (**choose** cheapest) all costs equal = Breadth-first

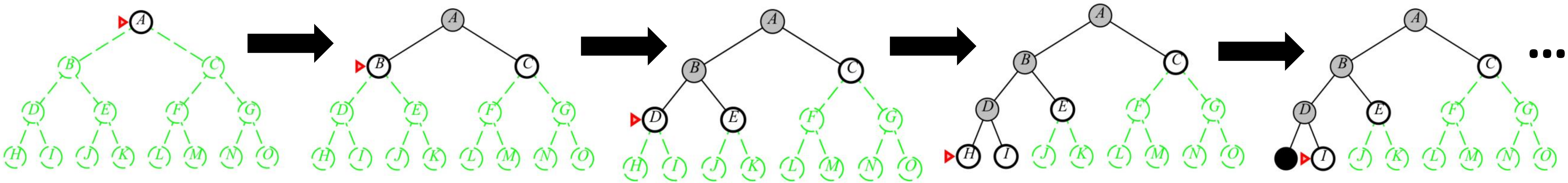


• Uniform-cost (**choose** cheapest) any costs

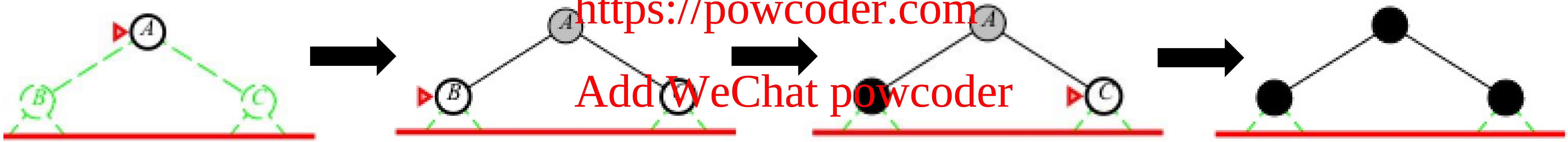




- Depth-first (**choose** deepest)



- Depth-limited (ℓ=1)



- Iterative deepening

Depth-limited (ℓ=0) ➡ Depth-limited (ℓ=1) ➡ Depth-limited (ℓ=2) •••

Uninformed search - Variants

- Backtracking search – variant of depth-first search
  - Only one (successor) node generated when expanded
  - Each (partially expanded) successor node remembers which node to generate next

- Bi-directional search
  - Two simultaneous searches (from start to goal and from goal to start)
  - Succeeds when (if) the frontiers of the two searches intersect

Properties

- *completeness*—does the algorithm always find a solution if one exists?
- *time complexity*—number of nodes generated/expanded
- *space complexity*—maximum number of nodes in memory
- *optimality*—does the algorithm always find an optimal (least-cost) solution?

Properties of uninformed search (with loop-checking)

Criterion	Breadth-first	Uniform-cost	Depth-first	Backtracking	Depth-limited	Iterative deepening	Bidirectional (breadth-first)
Complete?	Yes*	Yes*○	Yes†	Yes†	No	Yes*	Yes*
Time	$O(b^d)$	$O(b^{d+1})$ ●	s	s	$s(\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{d+1})$ ●	O(bm)	O(m)	$O(b\ell)$	O(bd)	$O(b^{d/2})$
Optimal?	Yes●	Yes	No	No	No	Yes●	Yes●

* **b** is finite
● step cost is constant
○ step cost is >0
† s is finite

b–maximum branching factor of the search tree (may be ∞)
d–depth of the optimal (least-cost) solution
m–maximum depth of the state space (may be ∞)
s–overall size of the state space
s(ℓ) –size of the state space till depth ℓ

Properties of uninformed search (without loop-checking)

Criterion	Breadth-first	Uniform-cost	Depth-first	Backtracking	Depth-limited	Iterative deepening	Bi-directional (breadth-first)
Complete?	Yes*	Yes*○	No	No	No	Yes*	Yes*
Time	$O(b^d)$	$O(b^{d+1})$ ●	$O(b^m)$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{d+1})$ ●	$O(bm)$	$O(m)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes●	Yes	No	No	No	Yes●	Yes●

* **b** is finite
● step cost is constant
○ step cost is >0

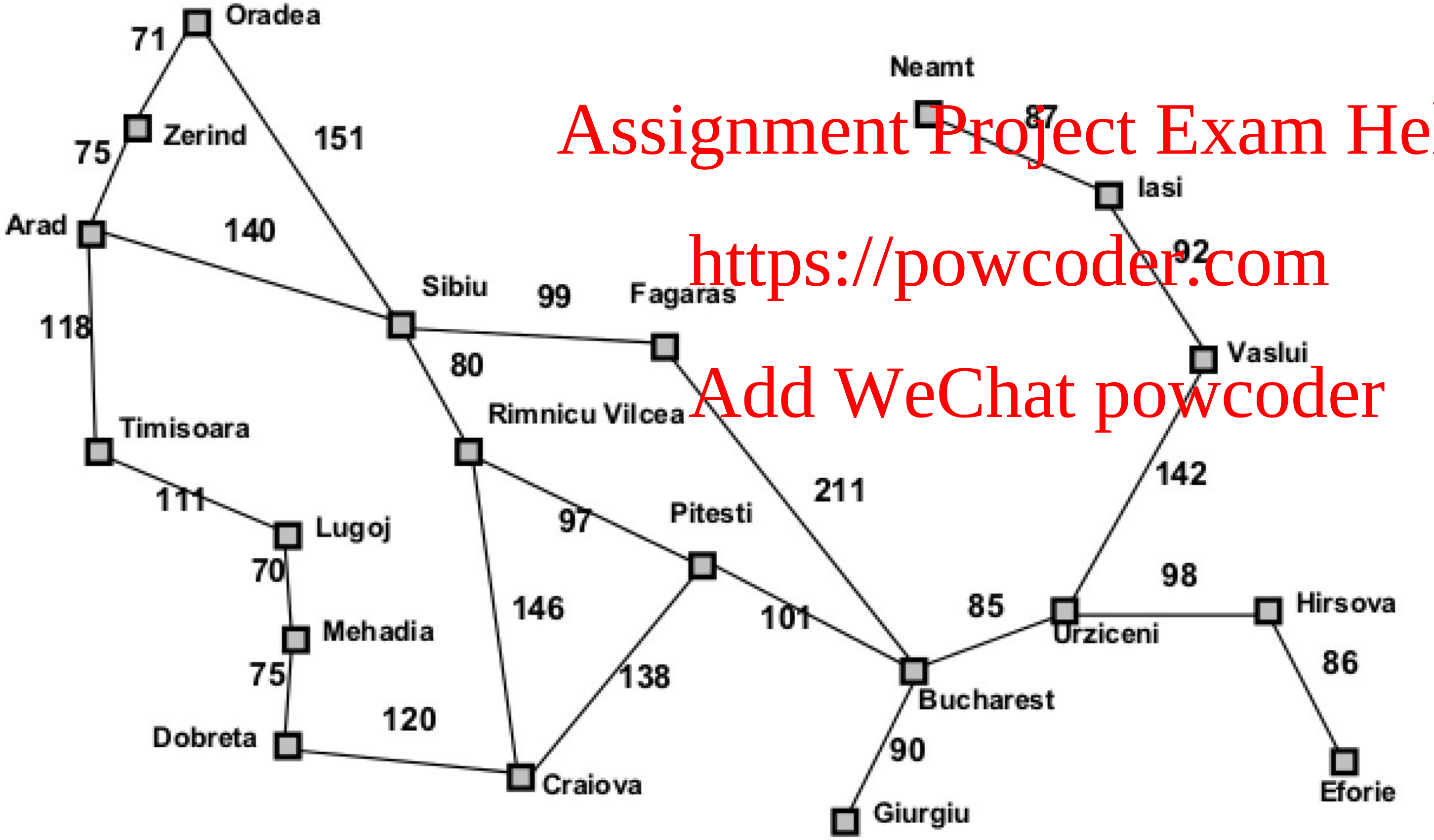
b–maximum branching factor of the search tree (may be ∞)
d–depth of the optimal (least-cost) solution
m–maximum depth of the state space (may be ∞)

Informed (heuristic-based) search

Use a *cost estimate* (of optimal path to goal) to **choose** node with the least-estimated path cost.

- Greedy-best-first search
  - cost estimate=heuristic function(node to goal)
- A* search
  - cost estimate=actual cost(to node)+heuristic function(node-to-goal)

Heuristic function: route finding example

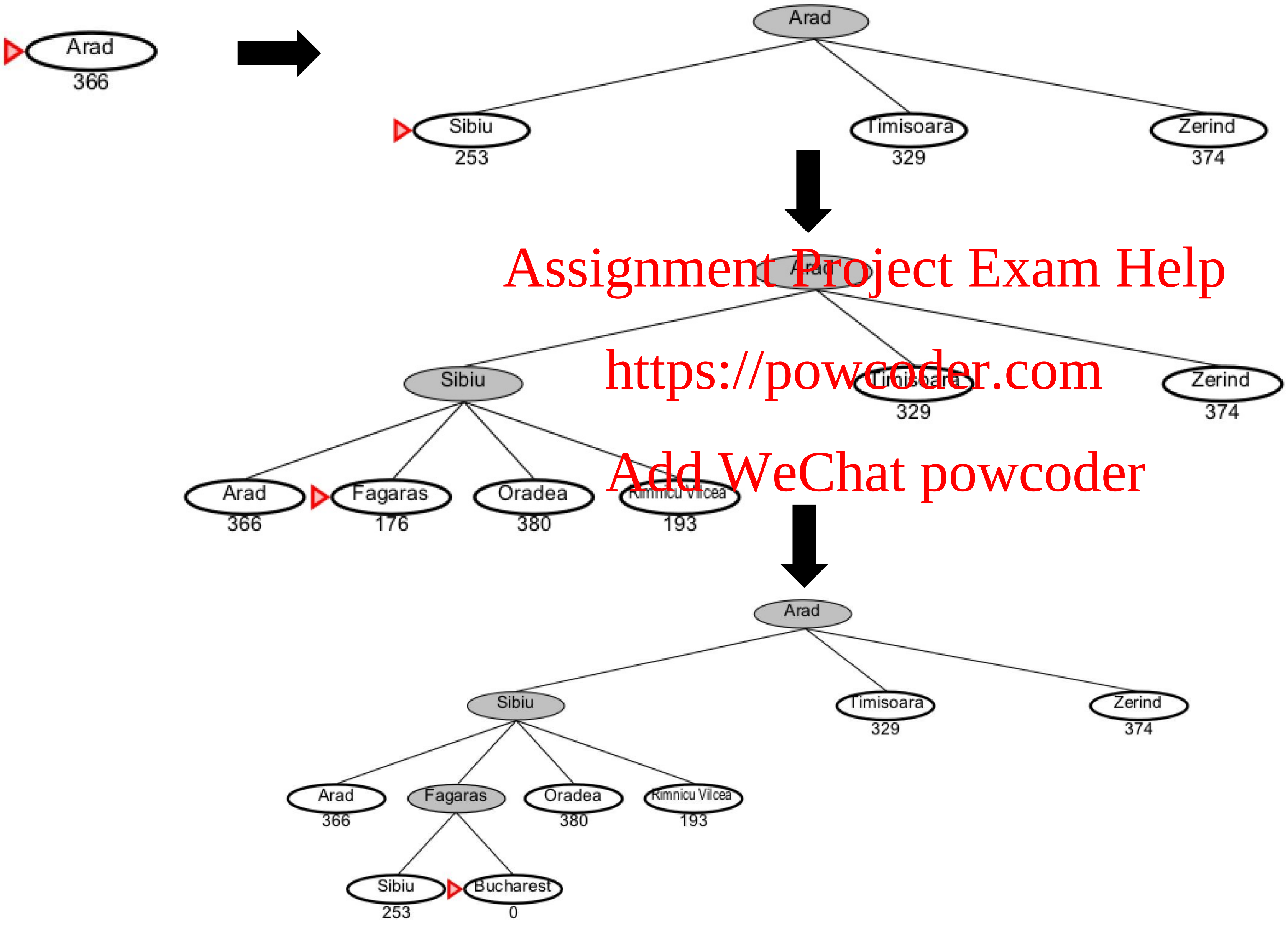


Heuristic function=

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

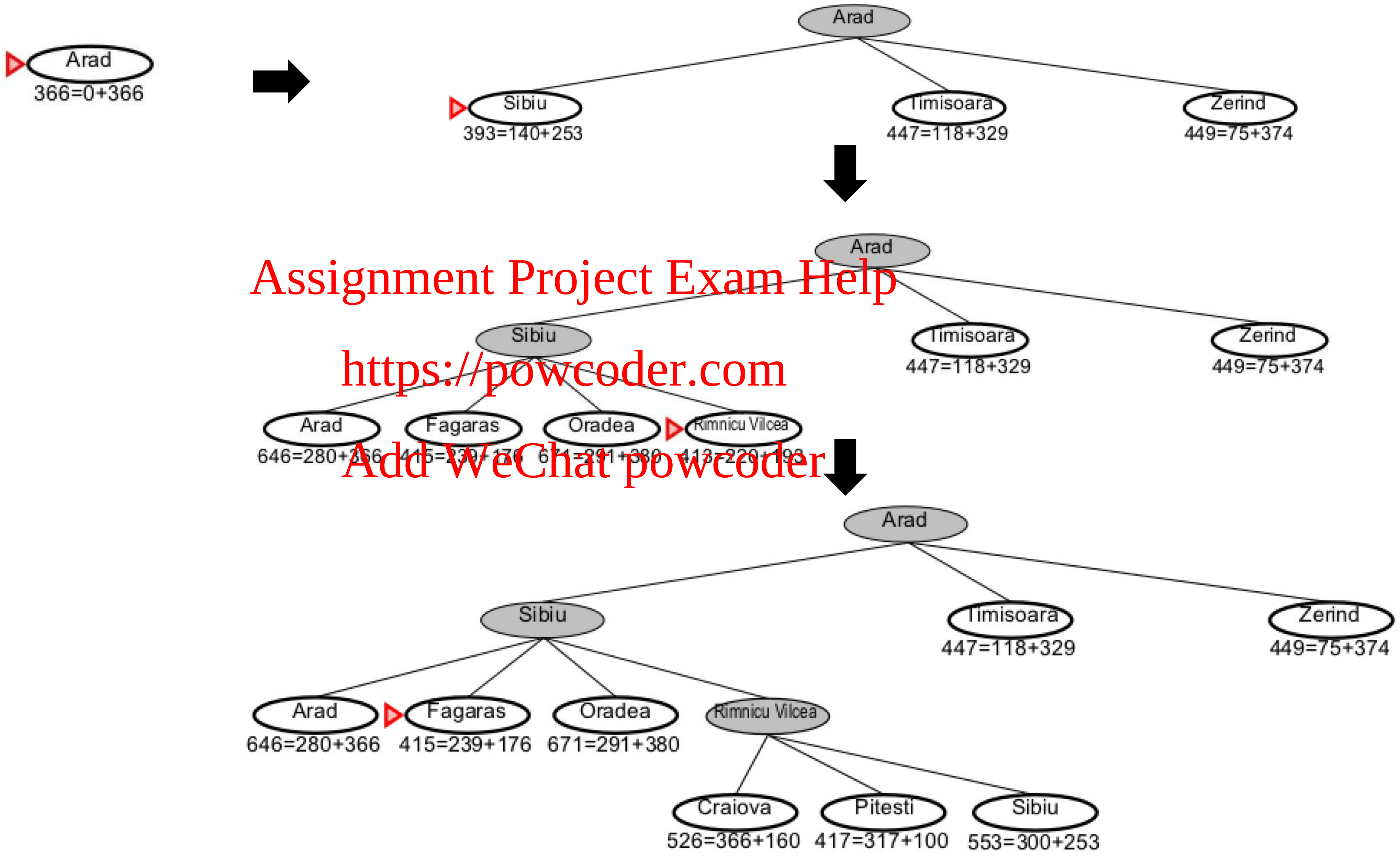
- Greedy-best-first (**choose** node with lowest value of heuristic function)

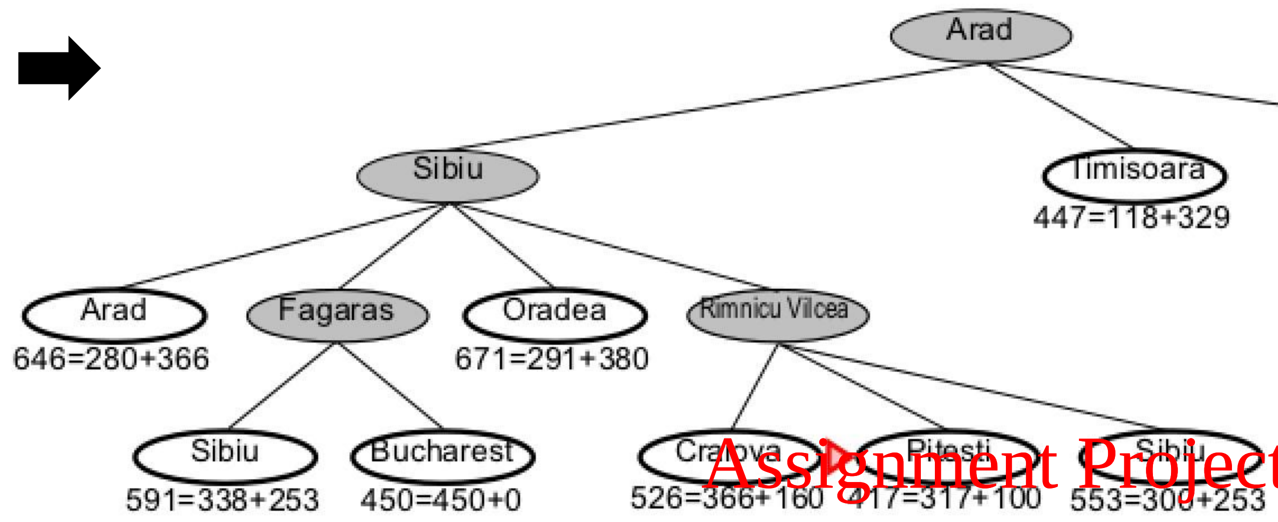


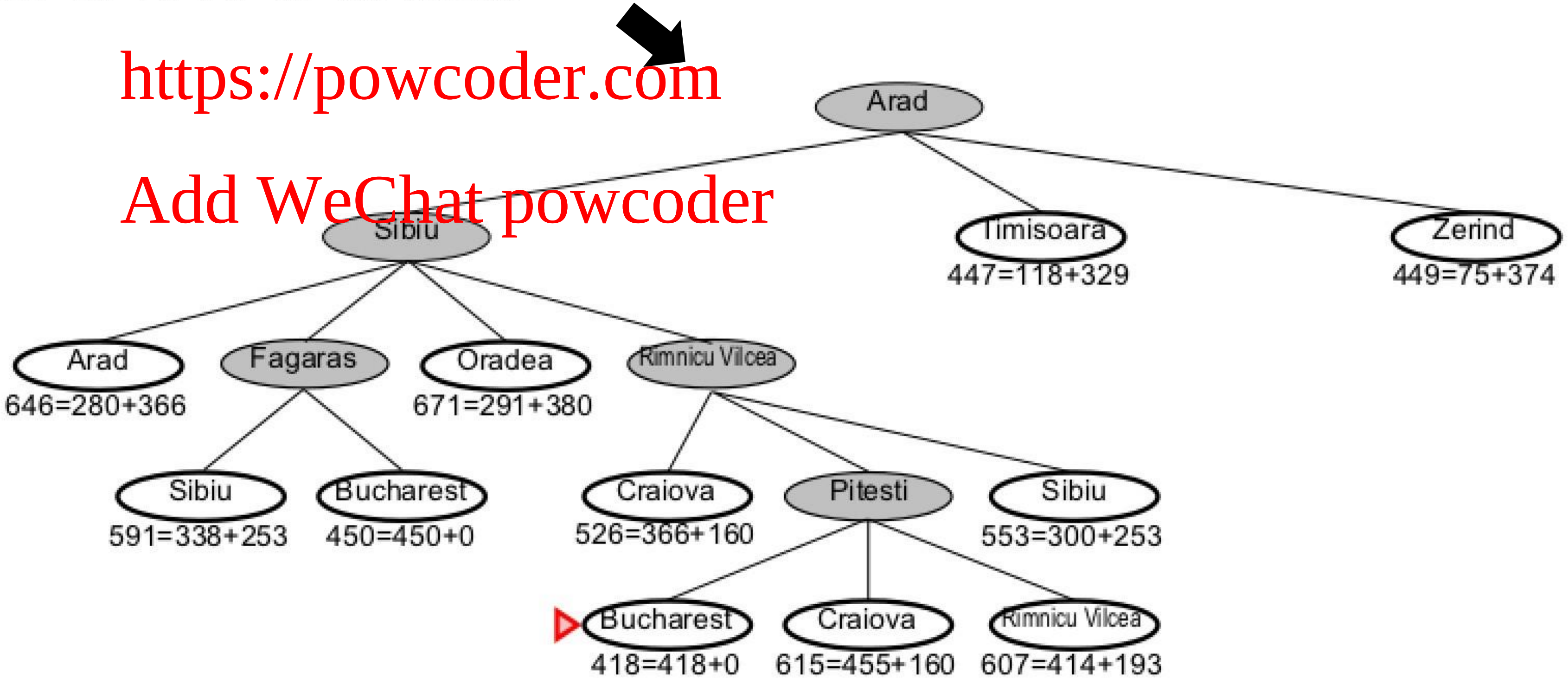
Straight–line distance to Bucharest

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

- A* (**choose** node with lowest value of actual cost+heuristic function)




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
417=317+100
Sibiu
553=300+253


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

Properties of heuristic functions

For n any node, let

$g(n, m)$ be the actual cost of reaching node m from n

$h(n)$ be the heuristic function applied to n (assume $h(n) \geq 0$ -- so that $h(goal) = 0$)

- **Admissible**: it never overestimates the actual cost (to goal) – e.g. straight-line distance

$h(n) \leq g(n, mgoal)$ for m_{goal} the cheapest goal reachable from n

- **Consistent/Monotonic**: it never overestimates the actual cost to any successor node+the heuristic function applied to that node – e.g. straight-line distance

$h(n) \leq g(n, n') + h(n')$ for n' any successor of n

Consistent/Monotonic → Admissible

Properties of informed search (with or without loop-checking)

Criterion	**Greedy** (with)	**Greedy** (without)	**A*** (with)	**A*** (without)
Complete?	Yes†	No	Yes*○	Yes*○
Optimal?	No	No	Yes ◊	Yes ‡

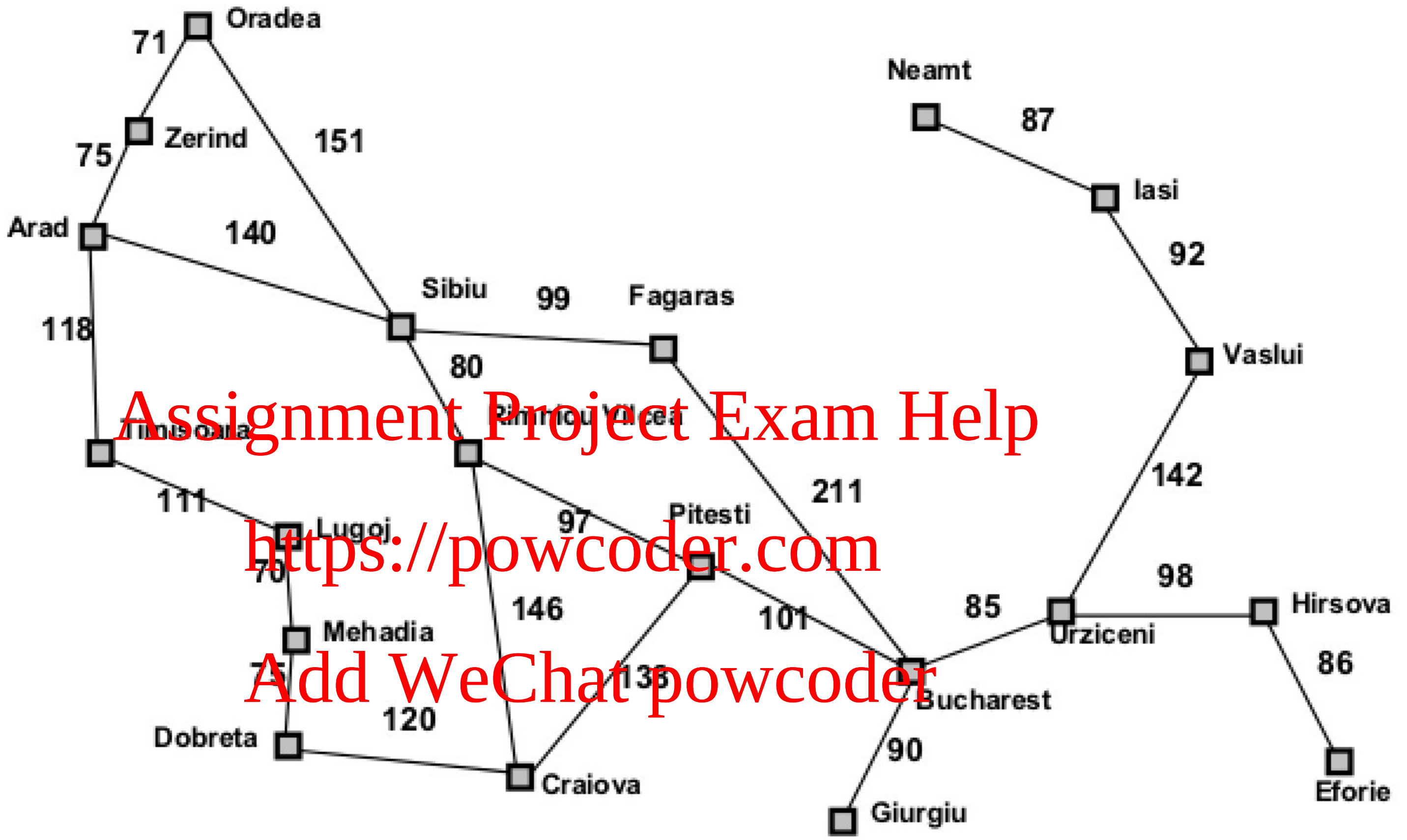
† **s** is finite
* **b** is finite
○ step cost is >0
◊ **h** consistent/monotonic
‡ **h** admissible

s–overall size of the state space
b–maximum branching factor of the search tree (may be ∞)
m–maximum depth of the state space (may be ∞)
h–heuristic function

A* is **optimally efficient** (no other optimal search algorithm is guaranteed to expand fewer nodes) but it often runs out of **space**

- Greedy best-first search not optimal:

Path via Sibiu-Fagaras is (32 km) longer than path through RV-Pitesti



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 best-first search without loop checking is not complete:

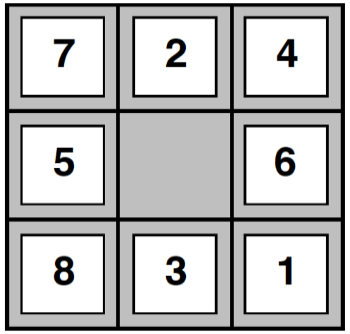
Iasi → Neamt → Iasi → Neamt →…

How to identify **admissible/consistent** heuristics?

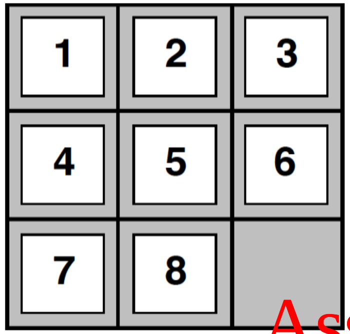
- Identify a **relaxed** version of the search problem (with a larger graph – with more edges)

- Use cost of optimal solutions for the relaxed problem as (admissible/consistent) heuristics for the original search problem

8-puzzle example – relaxed problems



Start



Goal

A tile can move from square A to square B if

- A is horizontally/vertically adjacent to B, and
- B is blank

Relaxed problems:

1. A tile can move from A to B if A is horizontally/vertically adjacent to B
2. A tile can move from A to B if B is blank
3. A tile can move from (any) A to (any) B

Heuristics:

1. $h(n)$=Manhattan distance (sum of the distances of tiles as in n from their goal positions)

2. $h(n)$= number of switches (blank-non-blank tile) to obtain goal from n

3. $h(n)$ = number of misplaced tiles in n wrt the goal

Heuristic search – today (example-non-examinable)

ELSEVIER

Artificial Intelligence

Available online 27 December 2017

In Press, Accepted Manuscript — Note to users

Star-Topology Decoupled State Space Search

Daniel Gnad, Jörg Hoffmann

Show more

https://doi.org/10.1016/j.artint.2017.12.004

Omitted

- Local search
  - Genetic algorithms
  - Simulated annealing
- Non-deterministic , partially observable or unknown environments

Adversarial search

- Competitive environment
  - opponents with conflicting goals
- “Unpredictable” opponent
  - solution is a strategy (=specifying a move for every possible opponent’s move/sequence of moves)
- Hard to solve, time limits
  - must approximate

Types of games

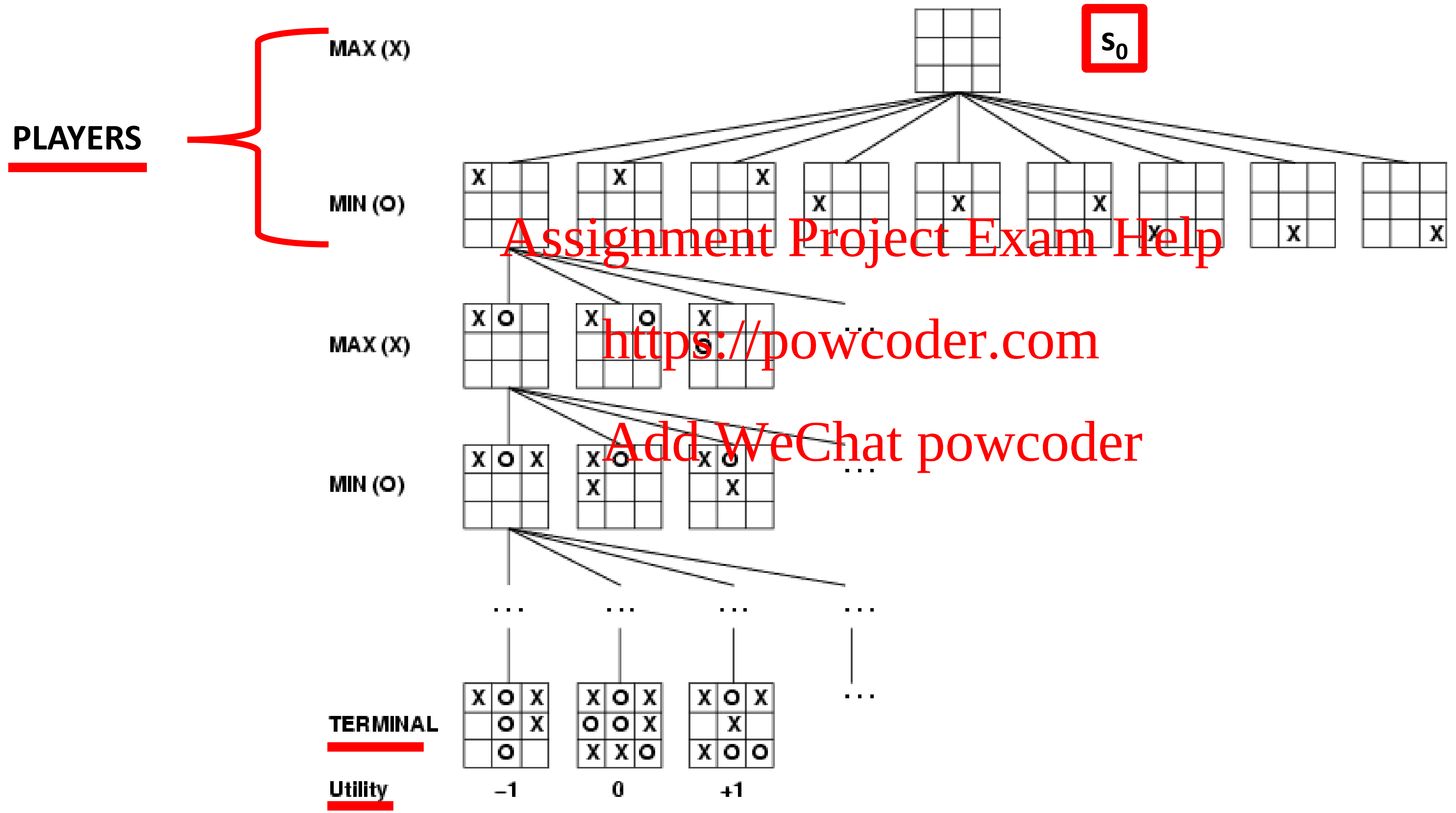
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Adversarial search problems – formally

Two-player, zero-sum (constant-sum) games

- s_0 : initial state
- PLAYER(s): determines which player (of two) moves in a state
- ACTIONS(s): returns the set of legal moves in a state
- RESULT(s,a): returns the state resulting from a move in a state
- TERMINAL(s): is true if the game has ended, false otherwise.
- UTILITY(s,p): returns 1 (win), -1 (lose), 0(draw)
  or 1 (win), 0 (lose), 1/2 (draw)

Example of two-player, zero-sum game: tic-tac-toe



Perfect play for deterministic, perfect-info games

- Minimax strategy
- α-β pruning

MINIMAX(s) – best payoff against best play

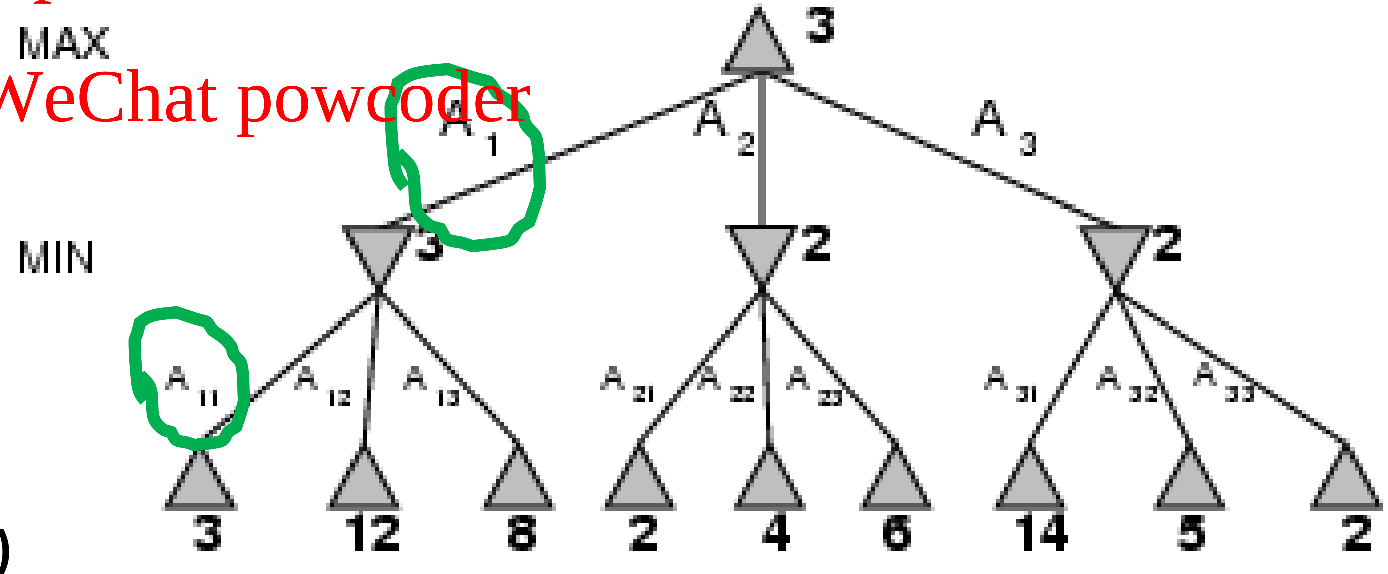
for $\neg$TERMINAL(s):

- if PLAYER(s)=MAX: return $\max_{\text{actions } a}$ MINIMAX(RESULT(s,a))
- if PLAYER(s)=MIN: return $\min_{\text{actions } a}$ MINIMAX(RESULT(s,a))

for TERMINAL(s):

- return UTILITY(s)

e.g. two-ply game



UTILITY (for MAX)

Properties of minimax

Criterion	minimax
Complete?	Yes*
Time	$O(b^m)$
Space	$O(bm)$
Optimal?	Yes

* If search tree finite

b–maximum branching factor of the search tree (may be ∞)
m–maximum depth of the state space (may be ∞)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games: not feasible

α-β pruning

- α minimum value that MAX is guaranteed to achieve so far (in some state)
- β maximum value that MAX is guaranteed to achieve so far (in some state)

Determine for initial state/node s: max-value(s,-∞,+∞) where

max-value(s,α,β):

- if TERMINAL(s) then return UTILITY(s)
- else, starting from v= -∞,
- for each action a:
  - let v=max(v,min-value(RESULT(s,a),α,β))
  - If v≥ β then return v
  - else let α=max(α,v)

return v

min-value(s,α,β):

…

min-value

… +∞

…

…min(v,max-value(RESULT(s,a),α,β))

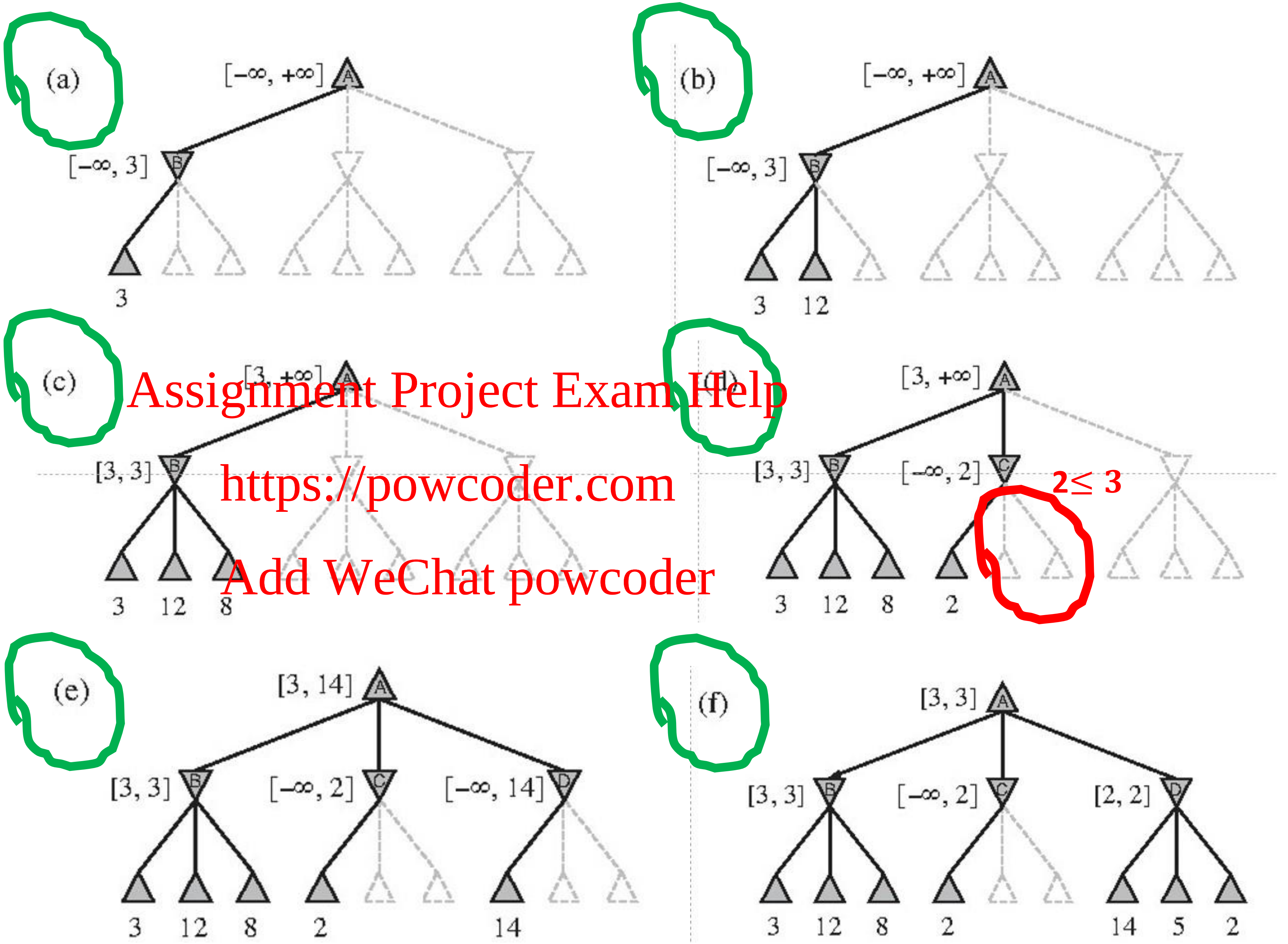
If v≤α then return v

else let β =min(β,v)

…

α-β pruning

[α,β]



Properties of α-β pruning

Criterion	minimax
Complete?	Yes*
Time	$O(b^{m/2})$ □
Space	$O(bm)$
Optimal?	Yes

* If search tree finite
□ if successors are examined best-first

b–maximum branching factor of the search tree (may be ∞)
m–maximum depth of the state space (may be ∞)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games: still slow

Omitted

Resource limits:

- Cutoff-Test instead of TERMINAL
- Evaluation instead of UTILITY
  - for chess weighted sum of features (e.g. number of white queens-number of black queens….)

Other types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Search&Games – today (example – non-examinable)

nature
International journal of science

Article

Mastering the game of Go without human knowledge

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature **550**, 354–359 (19 October 2017)
doi:10.1038/nature24270
Download Citation

Received: 07 April 2017
Accepted: 13 September 2017
Published online: 18 October 2017

Search - summary

- Formulation of search problems
- Uninformed search algorithms: breadth-first, uniform-cost, depth-first, limited-depth, iterative deepening, backtracking, bidirectional
- Informed (heuristic-based) search algorithms: greedy-best-first, A*, admissible and consistent/monotonic heuristic functions
- Adversarial search: minimax, α-β pruning
- Properties of search algorithms: completeness, optimality, time and space complexity