Bookkeeping

- HW1 due 9/16 at 11:59 PM
  - Writing (please read the integrity page statement on GenAI use carefully)
  - Problem sets
  - Programming
- Today:
  - Last of uninformed search
    - Uniform-Cost, Iterative Deepening, Bidirectional
  - Informed search, heuristics
- Soon: Constraint satisfaction

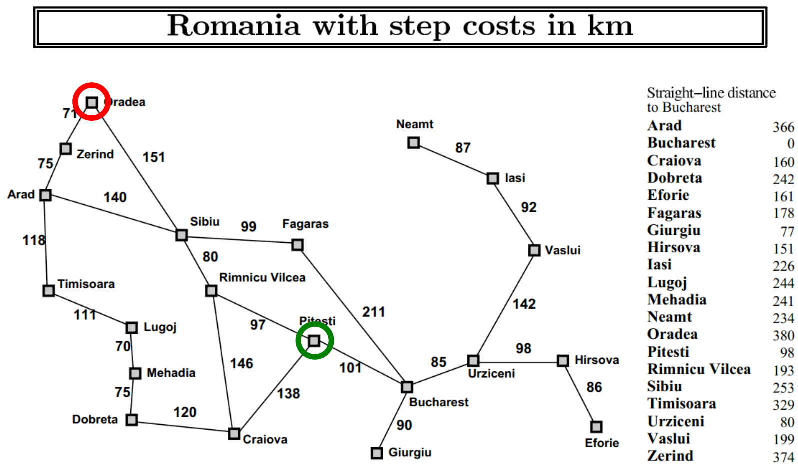
1

Uninformed Search: Uniform-Cost (UCS)

- Enqueue nodes by **path cost**:
  - Let g(n) = cost of path from start node to current node *n*
  - Sort nodes by increasing value of *g*
  - Identical to breadth-first search **if** all operators have equal cost
- *"Dijkstra's Algorithm"* in algorithms literature
- *"Branch and Bound Algorithm"* in operations research literature
- **Complete** (*)
- **Optimal/Admissible** (*)
  - Admissibility depends on the goal test being applied *when a node is removed from the nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$

2

Example: Path Costs



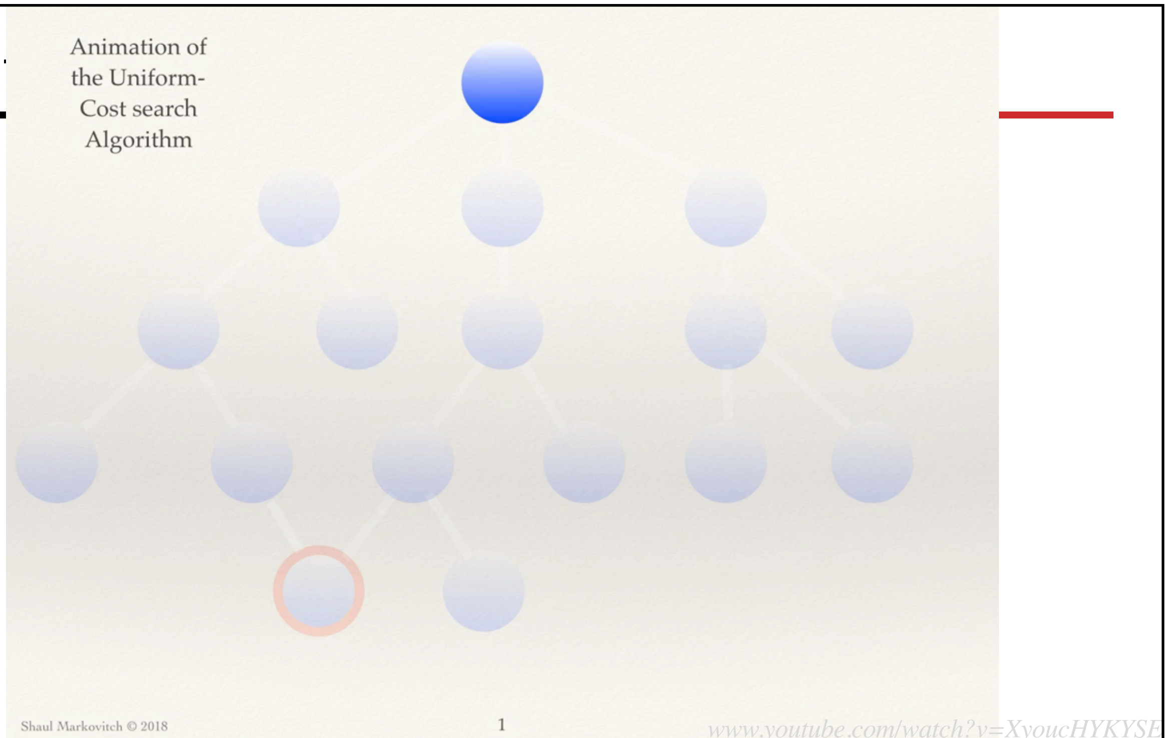
3

UCS Implementation

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Equivalent to breadth-first if step costs all equal
- Equivalent to Dijkstra's algorithm in general

4

Uni



5

Depth-First Iterative Deepening (DFID)

1. DFS to depth 0 (i.e., treat start node as having no successors)
2. Iff no solution, do DFS to depth 1

```
until solution found do:
  DFS with depth cutoff c;
  c = c+1
```

- **Complete**
- **Optimal/Admissible** if all operators have the same cost
  - Otherwise, not optimal, but guarantees finding solution of shortest length
- **Time complexity** is a little worse than BFS or DFS
- **Nodes near the top of the tree are generated multiple times**
  - Because most nodes are near the bottom of a tree, worst case time complexity is still exponential, O(bd)

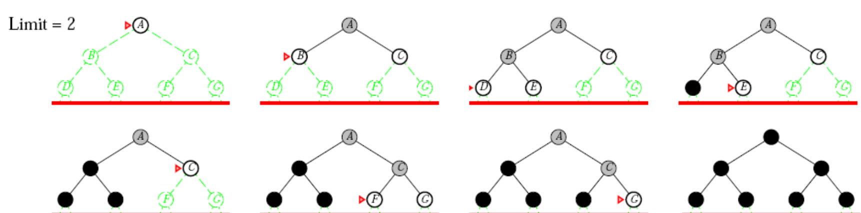
8

Iterative deepening search (c=1)

Nodes visited: 3

9

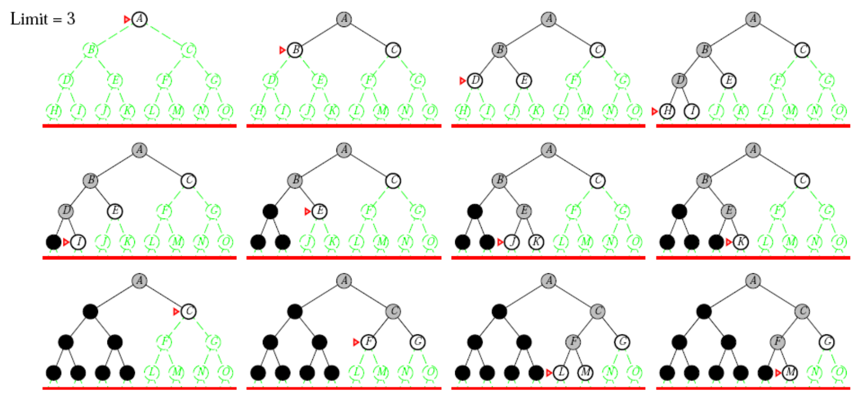
Iterative deepening search (c=2)



Nodes visited: 3+4 = 7

10

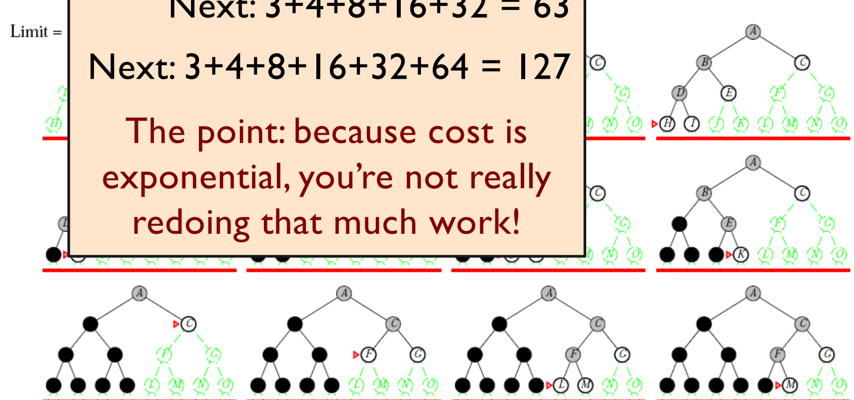
Iterative deepening search (c=3)



Nodes visited: 3+4+8 = 15

11

Iterative deepening search (c=3)



Next: 3+4+8+16 = 31

Next: 3+4+8+16+32 = 63

Next: 3+4+8+16+32+64 = 127

The point: because cost is exponential, you're not really redoing that much work!

Nodes visited: 3+4+8 = 15

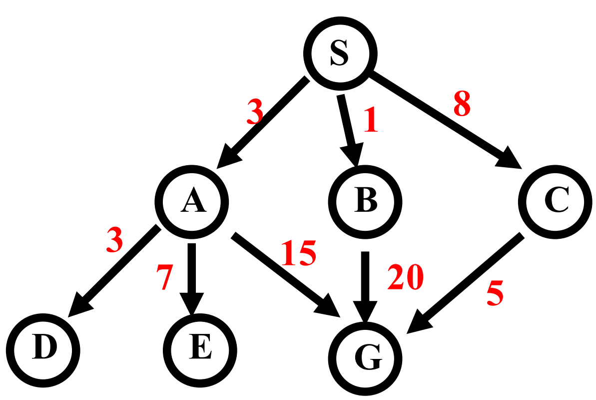
12

Depth-First Iterative Deepening

- If branching factor is *b* and solution is at depth *d*, then nodes at depth *d* are generated once, nodes at depth *d*-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db \leq b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, then worst case is 1.78 * 4^d, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- **Linear space complexity**, O(bd), like DFS
- Has advantage of both BFS (completeness) and DFS (limited space, finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

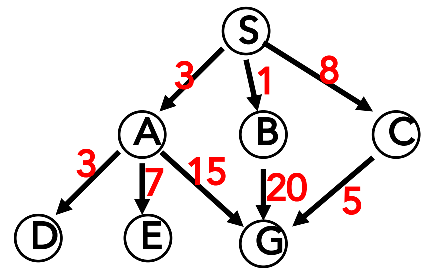
13

Example for Illustrating Search Strategies



14

Depth-First Search



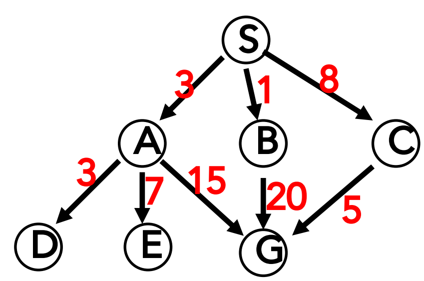
Expanded node	**Nodes list**
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S → A → G, cost 18

Number of nodes expanded (including goal node) = 5

15

Breadth-First Search



Expanded node	**Nodes list**
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

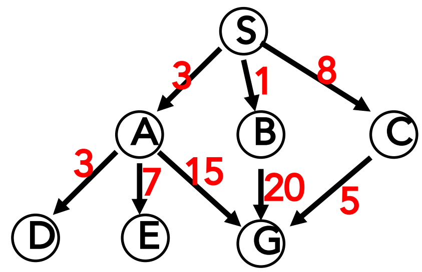
Solution path found is S → A → G , cost 18

Number of nodes expanded (including goal node) = 7

16

Uniform-Cost Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }



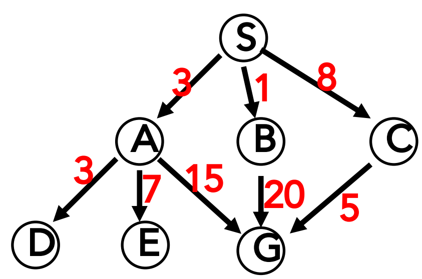
Solution path found is S → C → G, cost 13

Number of nodes expanded (including goal node) = 7

17

How they Perform

- **Depth-First Search:**
  - Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search:**
  - Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search:**
  - Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)
  - *This is the only **uninformed** search that worries about costs.*
- **Iterative-Deepening Search:**
  - nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)



18

Comparing Search Strategies

	Complete	Optimal	Time complexity	Space complexity
Breadth first search:	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	$O(bm)$
Depth limited search	if l >= d	no	$O(b^l)$	$O(bl)$
depth first iterative deepening search	yes	yes	$O(b^d)$	$O(bd)$
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

b is branching factor, d is depth of the shallowest solution,
m is the maximum depth of the search tree, l is the depth limit

19

Blind Search (Redux)

- Last time:
  - Search spaces
  - Problem states
  - Goal-based agents
  - Breadth-first
  - Depth-first
  - Uniform-cost
  - Iterative deepening
- From the book:
  - Bidirectional
  - Holy Grail Search

20

Comparing Search Strategies

- b is branching factor, d is depth of the shallowest solution, m is the maximum depth of the search tree, l is the depth limit

	Complete	Optimal	Time complexity	Space complexity
Breadth first search:	yes	yes (Given unit arc costs)	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	$O(bm)$
Depth limited search	if $l >= d$	no	$O(b^l)$	$O(bl)$
depth first iterative deepening search	yes	yes	$O(b^d)$	$O(bd)$
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

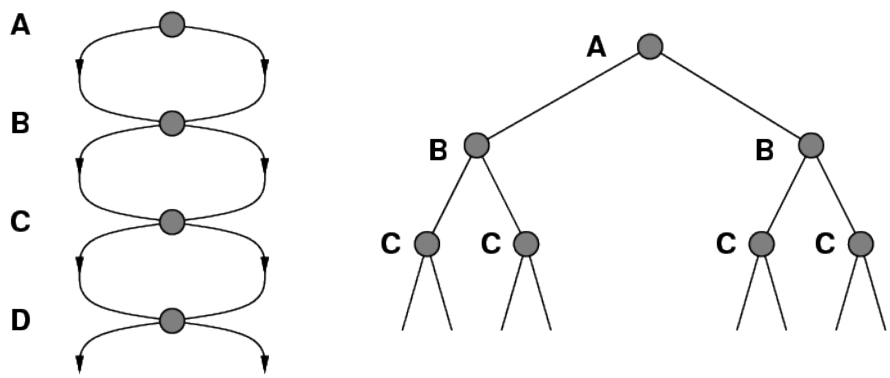
21

Avoiding Repeated States

- Ways to reduce size of state space (with increasing computational costs)
- In increasing order of effectiveness *and* cost:
  - Do not return to the state you just came from.
  - Do not create paths with cycles in them.
  - Do not generate any state that was ever created before.
- Effect depends on frequency of loops in state space.
  - Worst case, storing as many nodes as exhaustive search!

22

State Space → An Exponentially Growing Search Space



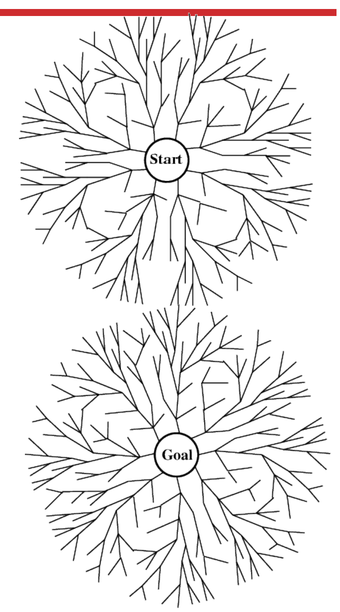
23

Bi-directional Search

- Alternate searching from
  - start state → goal
  - goal state → start
- Stop when the fr
- Works well only w   nd goal states
- Requires ability t   es
- Can (sometimes) find a solution fast

What's a real world problem where you can generate predecessors?

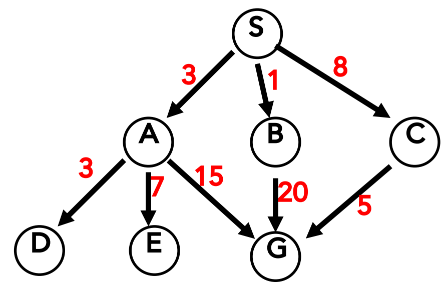
What's a problem where you cannot?



24

Holy Grail Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ C^8 A^3 B^1 }
C^8	{ G^{13} A^3 B^1 }
G^{13}	{ A^3 B^1 }



Solution path found is S C G, cost 13 (optimal)

Number of nodes expanded (including goal node) = 3

(minimum possible!)

25

Holy Grail Search

- Why not go straight to the solution, without any wasted detours off to the side?
- If we knew where the solution was we wouldn't be searching!

If only we knew where we were headed…

26

"Satisficing"

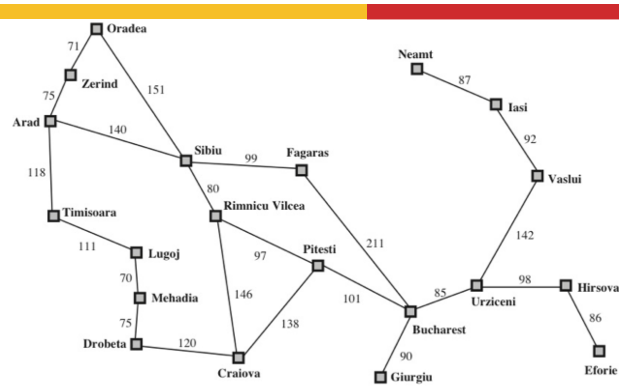
- Wikipedia: "**Satisficing** is … searching until an **acceptability threshold** is met"
- Contrast with **optimality**
  - Satisficable problems *do not get more benefit from* finding an optimal solution
- Ex: You have an A in the class. Studying for 8 hours will get you a 98 on the final. Studying for 16 hours will get you a 100 on the final. What to do?
- A combination of *satisfy* and *suffice*
- Introduced by Herbert A. Simon in 1956

Another piece of **problem definition**

27

Informed Search (Ch. 3.5-3.7)

"An informed search strategy—one that uses problem specific knowledge… can find solutions more efficiently then an uninformed strategy." – R&N pg. 92



Based Some material adapted from slides by Dr. Matuszek @ Villanova University, which are based on Hwee Tou Ng at Berkeley, which are based on Russell at Berkeley. Some diagrams based on AIMA.

28

Overview of Informed Search

- Heuristic search
- Heuristic functions
- Admissibility
- Best-first search
  - Greedy search, beam search, A*
  - Examples
- Memory-conserving variations of A*

Questions?

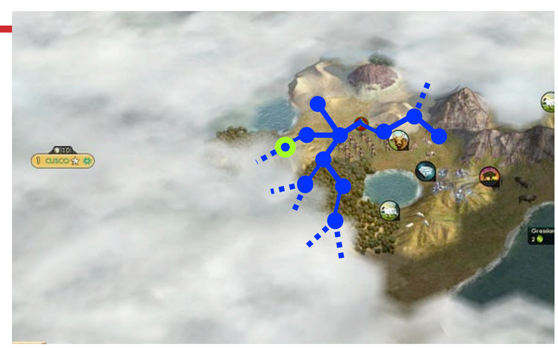
"An informed search strategy—one that uses problem specific knowledge... can find solutions more efficiently then an uninformed strategy."

– R&N pg. 92

29

The Core Idea

- How can we make search smarter?
  - Use problem-specific knowledge beyond the definition of the problem
  - Specifically, incorporate knowledge of how good a non-goal state is



- Informed or **Best-First Search**
  - Node selected for expansion is based on an evaluation function *f(n)*
    - I.e., expand the node that **appears to be** the best bet
  - Node with lowest evaluation is selected for expansion
    - Uses a priority queue

Slide from Dr. Rebecca Hutchinson @ Oregon State
Image: medium.com/blockchain-gaming/fog-of-war-7dba2b7faa72

30

Definition: Heuristic

- Free On-line Dictionary of Computing*: **A rule of thumb, simplification, or educated guess**
- WordNet (r) 1.6*: **Commonsense rule (or set of rules) intended to increase the probability of solving some problem**
- Reduces, limits, or guides search in particular domains
- Does not guarantee feasible solutions; often with no theoretical guarantee
  - **Playing chess:** try to take the opponent's queen
  - **Getting someplace:** head in that compass direction when possible

**Heavily edited for clarity*

31

Heuristic Search

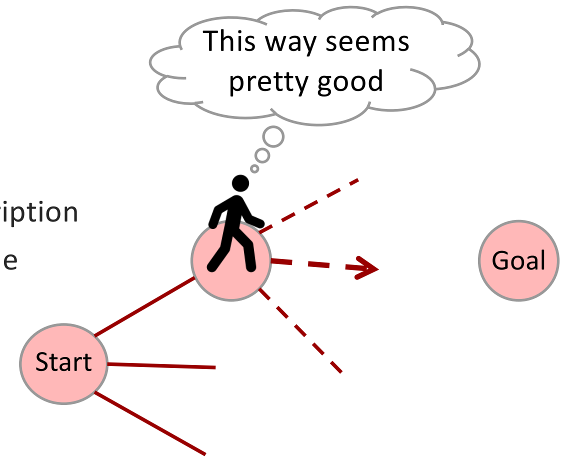
- Uninformed search is **generic**
  - Node selection depends only on shape of tree and node expansion strategy
- **Domain knowledge*** → better decisions (sometimes)
  - Knowledge about the specific problem
  - Often calculated based on **state**

* *Domain knowledge* is a general term in AI. A *domain* is a specific problem space, which you may or may not know something about. Examples: game playing; chess; medicine; perfumery; …

32

Is It A Heuristic?

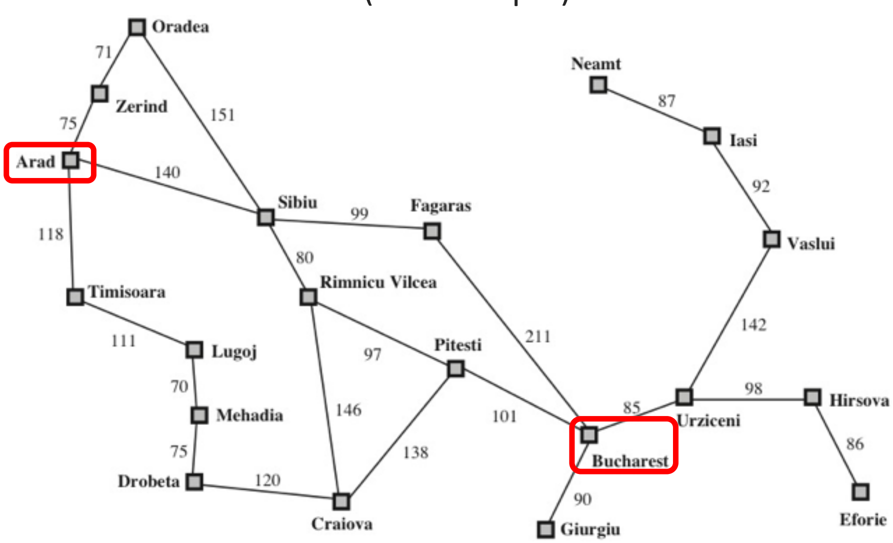
- A **heuristic function** is:
  - An **estimate** of how close we are to a goal
    - We don't assume perfect knowledge
      - That would be holy grail search
    - So, the estimate can be wrong
  - Based on domain-specific information
  - Computable from the current state description
  - A function over nodes that returns a value
    - Node = particular problem state



33

Heuristic Search

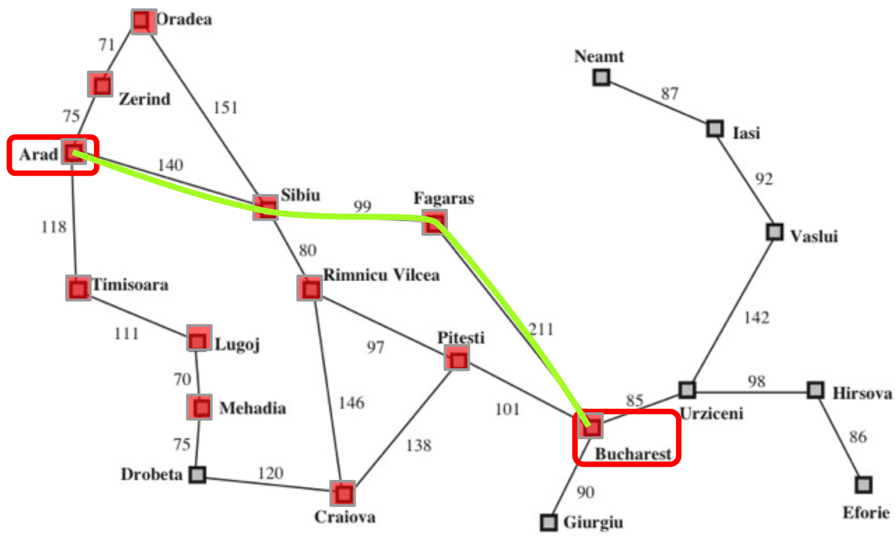
- Romania: Arad→ Bucharest (for example)



34

Breadth-First Search

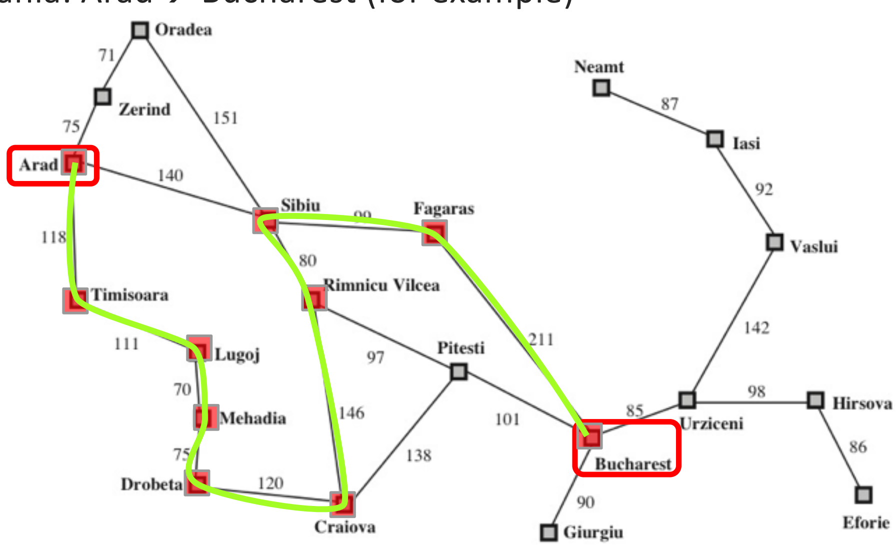
- Romania: Arad→ Bucharest (for example)



35

Depth-First Search

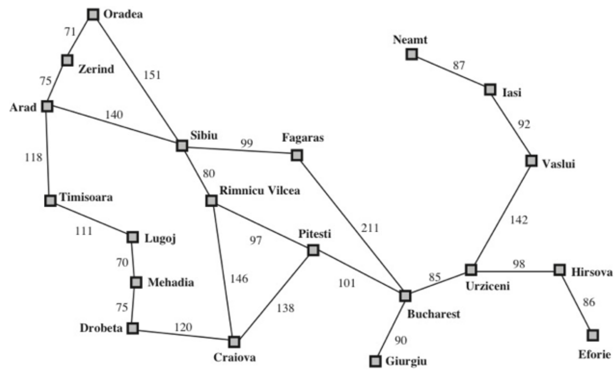
- Romania: Arad→ Bucharest (for example)



36

Heuristic Search

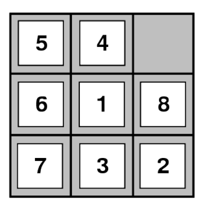
- Romania:
  - Eyeballing it → certain cities first
  - They "look closer" to where we are going
- Can domain knowledge be captured in a **heuristic**?

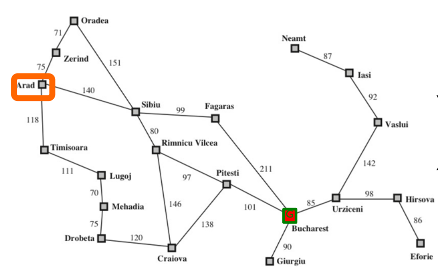


37

Heuristics Examples

- 8-puzzle:
  - # of tiles in wrong place
- 8-puzzle (better):
  - Sum of distances from goal
  - Captures distance and number of nodes
- Romania:
  - Straight-line distance from current node to goal
  - Captures "closer to Bucharest"

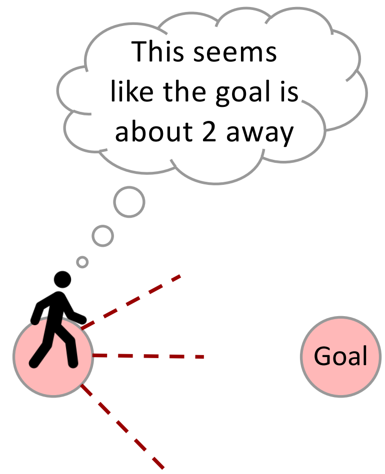
f() = ?

f() = ?

38

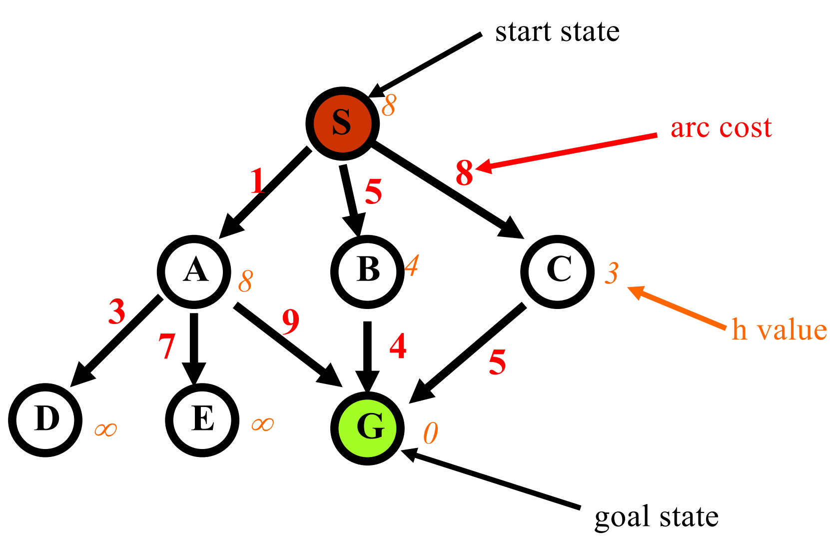
Heuristic Function

- **All** domain-specific knowledge is encoded in heuristic function *h*
- *h* is some **estimate** of how desirable a move is
  - How "close" (we think, maybe) it gets us to our goal
- Usually:
  - $h(n) \geq 0$: for all nodes *n*
  - $h(n) = 0$: *n* is a goal node
  - $h(n) = \infty$: *n* is a dead end (no goal can be reached from *n*)



39

Example Search Space Revisited



40

Weak vs. Strong Methods

- **Weak methods**:
  - Extremely general, not tailored to a specific situation
- Examples
  - **Subgoaling**: split a large problem into several smaller ones that can be solved one at a time.
  - **Space splitting:** try to list possible solutions to a problem, then try to rule out *classes* of these possibilities
  - **Means-ends analysis:** consider current situation and goal, then look for ways to shrink the differences between the two
- Called "weak" methods because they do not take advantage of more powerful domain-specific heuristics

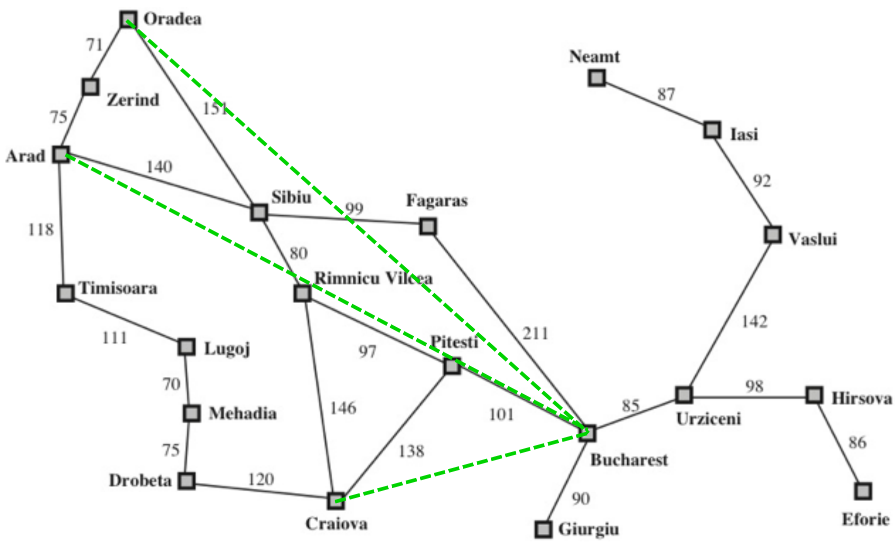
41

Domain Information

- Informed methods add domain-specific information!
- Goal: select the best path to continue searching
  - Uninformed methods (BFS, DFS, UCS) push nodes onto the search list based only on the order in which they are encountered and the cost of reaching them
  - Informed methods try to explore the best ("most likely looking") nodes first
- Define ***h*(*n*)** to estimate the "goodness" of node *n*
  - *h*(*n*) = **estimated cost** (or distance) of minimal cost path from *n* **to a goal state**

42

Straight Lines to Bucharest (km)



$h_{SLD}(n)$

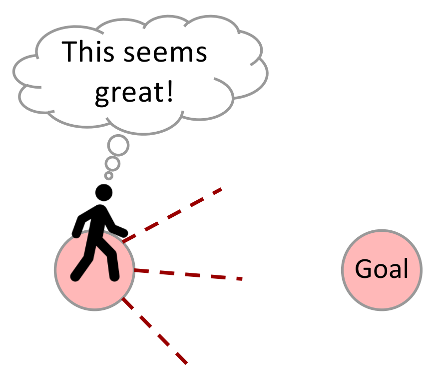
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

R&N pg. 68, 93

43

Admissible Heuristics

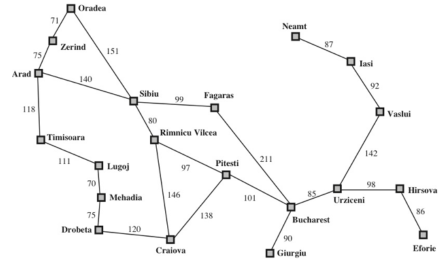
- Admissible heuristics never <u>over</u>estimate cost
  - They are *optimistic* – think goal is closer than it is
- $h(n) \leq h^*(n)$
  - where $h^*(n)$ is **true** cost to reach goal from n
- $h_{SLD}(\text{Lugoj}) = 244$
  - Can there be a shorter path?



44

Admissibility

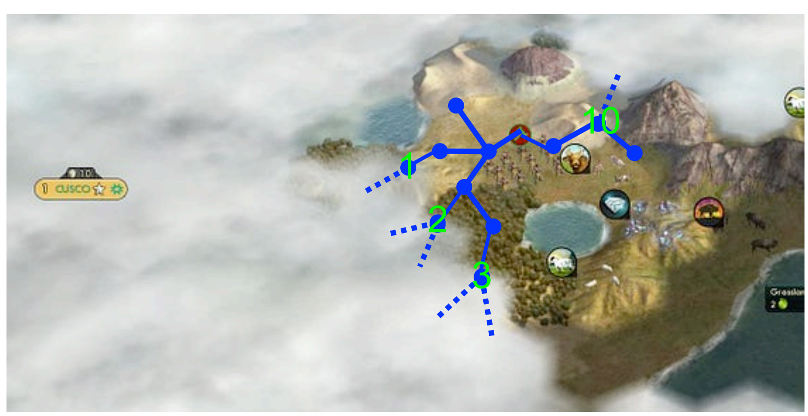
- Admissibility is a property of **heuristics**
  - They are *optimistic* – think goal is closer than it is
  - (Or, exactly right)
- Is “∀n, $h(n)$=1 kilometer” admissible?
- Admissible heuristics can be pretty bad!
- Using admissible heuristics guarantees that the first solution found will be optimal, **for some algorithms** (A*).



45

Best-First Search

- A generic way of referring to informed methods
- Use an **evaluation function** $f(n)$ over nodes
  - Gives an estimate of “desirability”
  - $f(n)$ incorporates domain-specific information
  - Different $f(n)$ → Different searches
  - $f(n)$ can incorporate knowledge from $h(n)$
- So let’s estimate $f(n)$ for these nodes…



46

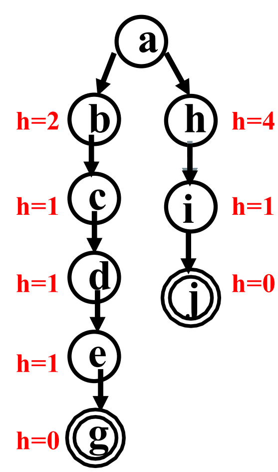
Best-First Search (more)

- Order nodes on the list by increasing value of *f(n)*
- Expand most desirable unexpanded node
  - Implementation:
  - Order nodes in frontier in decreasing order of desirability
- Special cases:
  - Greedy best-first search
  - A* search

47

Greedy Best-First Search

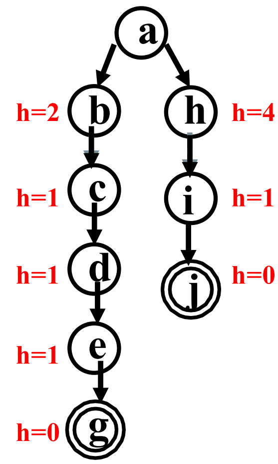
- Idea: always choose "closest node" to goal
  - Most likely to lead to a solution quickly
- So, evaluate nodes based only on heuristic function
  - $f(n) = h(n)$
- Sort nodes by increasing values of *f*
- Select node believed to be closest to a goal node (hence "greedy")
  - That is, select node with smallest *f* value



48

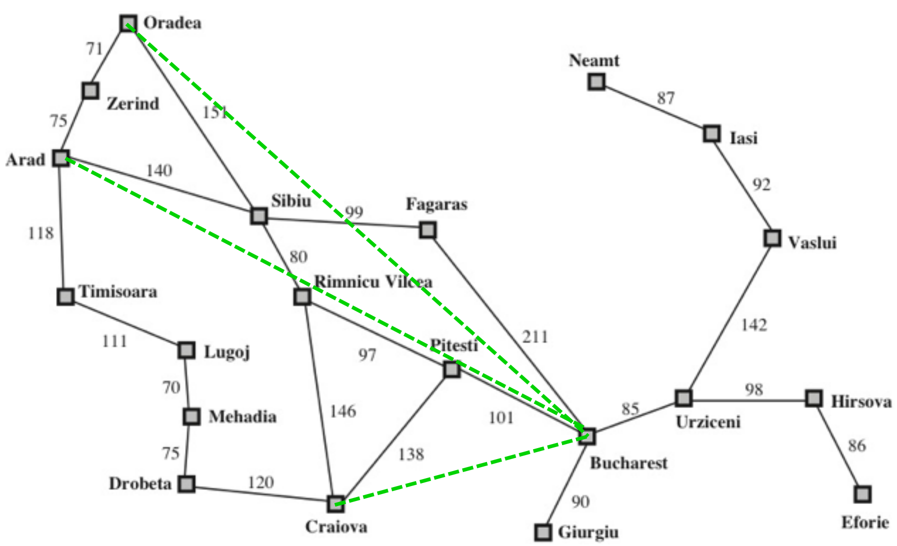
Greedy Best-First Search

- Optimal?
  - Why not?
- Example:
  - Greedy search will find:
    a→b→c→d→e→g ; cost = 5
  - Optimal solution:
    a→h→i→j ; cost = 3
- Not complete (why?)



49

Straight Lines to Bucharest (km)



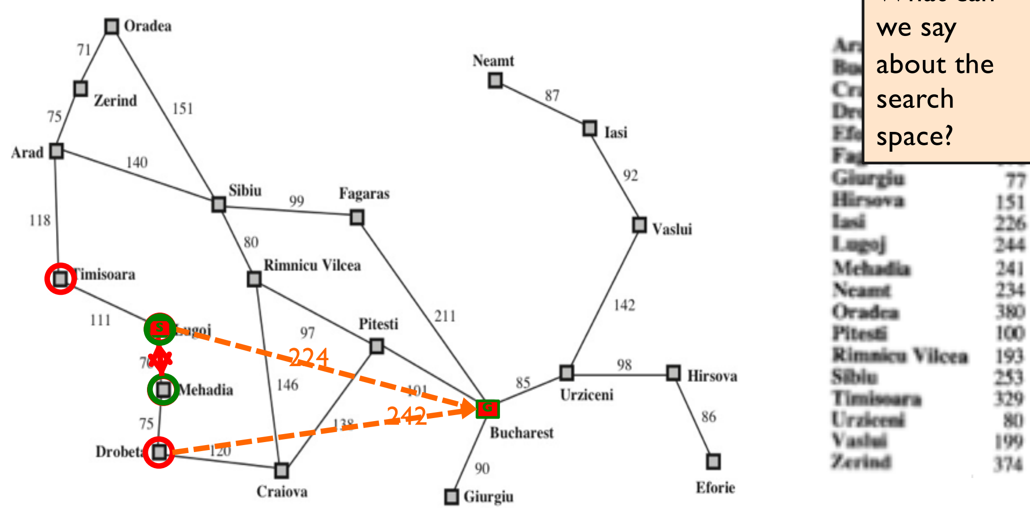
$h_{SLD}(n)$

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

R&N pg. 68, 93

50

Greedy Best-First Search: Ex. 1



What can we say about the search space?

51

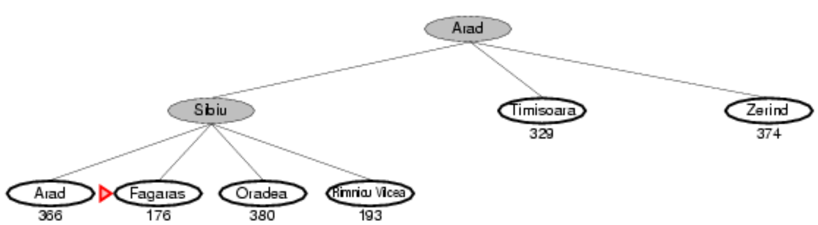
Greedy Best-First Search: Ex. 2

52

Greedy Best-First Search: Ex. 2

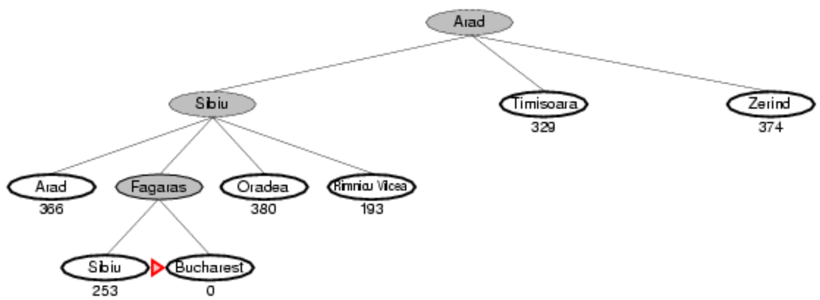
53

Greedy Best-First Search: Ex. 2



54

Greedy Best-First Search: Ex. 2



55

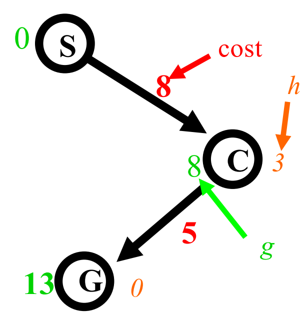
Beam Search

- Use an evaluation function *f(n)* = *h(n)*, but the maximum size of the nodes list is *k*, a fixed constant
- Only keeps *k* best nodes as candidates for expansion, and throws the rest away—can **never** explore those nodes
- More space-efficient than greedy search, but may throw away a node that is on a solution path
- Not complete
- Not admissible

56

A* Search

- **Idea:** Evaluate nodes by combining **g(n), the cost of reaching a node**, with **h(n), the cost of getting from the node to the goal**.
  - A* because $h(n) \leq h^*(n)$
- 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



66

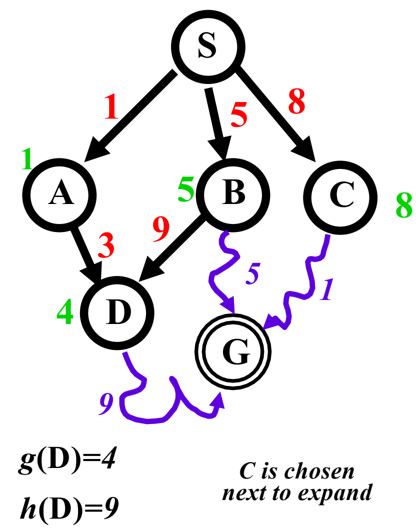
Quick Terminology Reminders

- What is *f(n)*?
  - An **evaluation function** that gives…
  - A cost <u>estimate</u> of…
  - The distance from *n* to *G*
- What is *h*(*n*)?
  - A **heuristic function** that…
  - Encodes domain knowledge about...
  - The search space
- What is *h**(*n*)?
  - A **heuristic function** that gives the…
  - **True** cost to reach goal from *n*
  - Why don't we just use that?
- What is *g*(*n*)?
  - The **path cost** of getting from *S* to *n*
  - describes the "already spent" costs of the current search

67

Algorithm A*

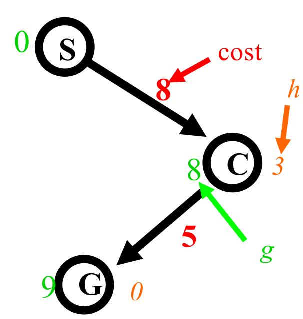
- Use evaluation function ***f(n) = g(n) + h(n)***
- *g(n)* = minimal-cost path from S to state n
  - That is, the cost of getting to the node so far
- Ranks nodes on frontier by estimated cost of solution
  - From start node, through given node, to goal
- Not complete if *h(n)* can = ∞



68

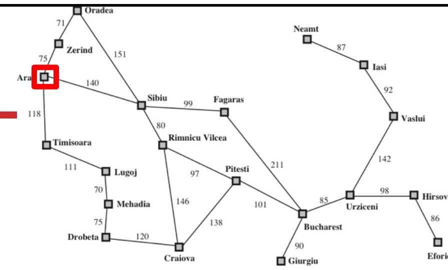
A* Search

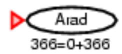
- Avoid expanding paths that are already expensive
  - Combines costs-so-far with expected-costs
- Is **complete** iff
  - Branching factor is finite
  - Every operator has a fixed positive cost
- Is **admissible** iff
  - *h*(*n*) is admissible



69

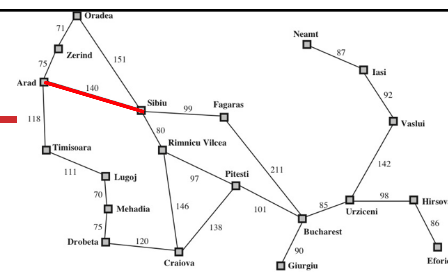
A* Example 1





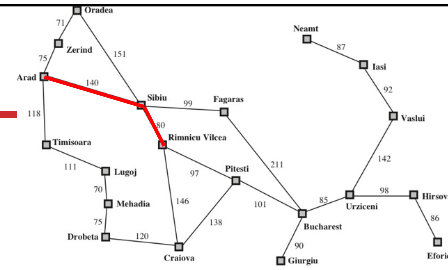
70

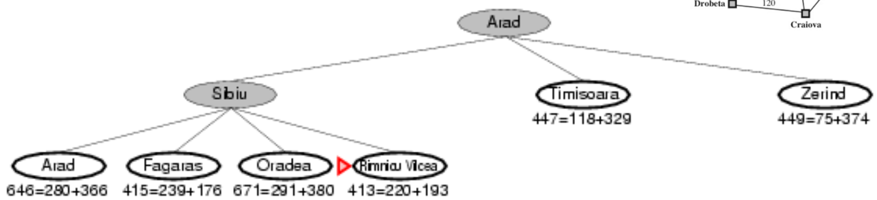
A* Example 1



71

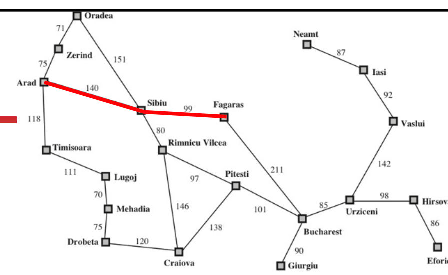
A* Example 1

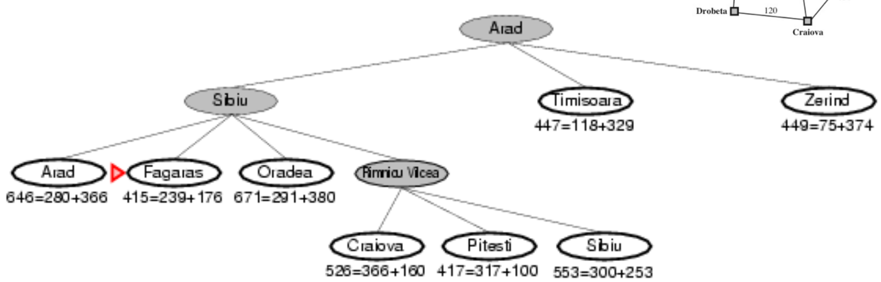




72

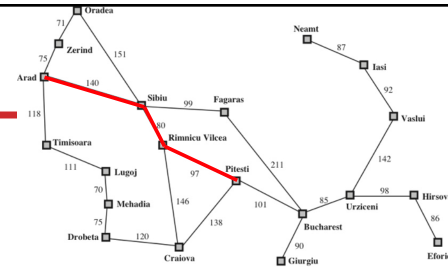
A* Example 1

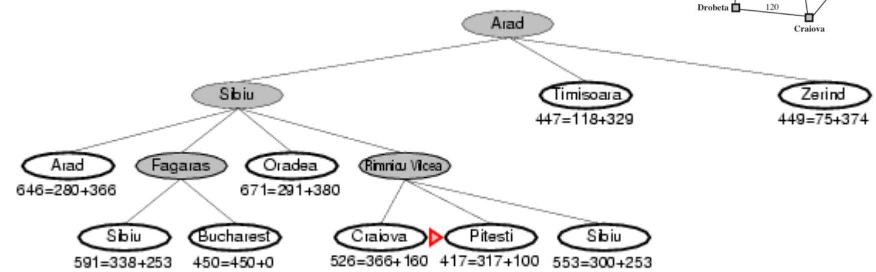




73

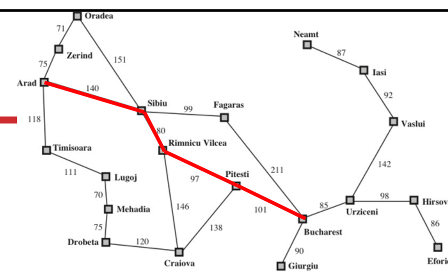
A* Example 1

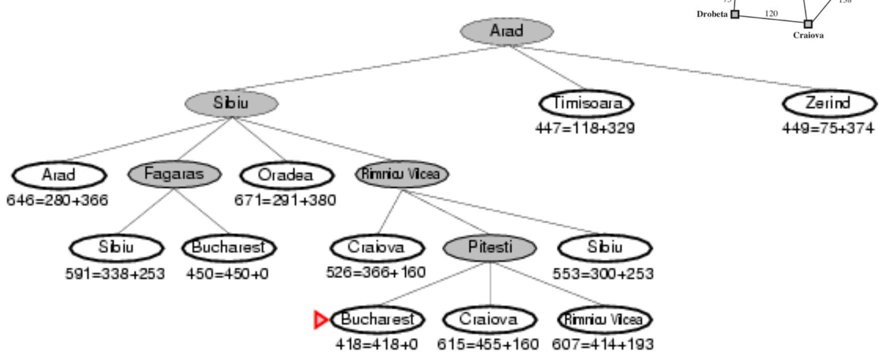




74

A* Example 1





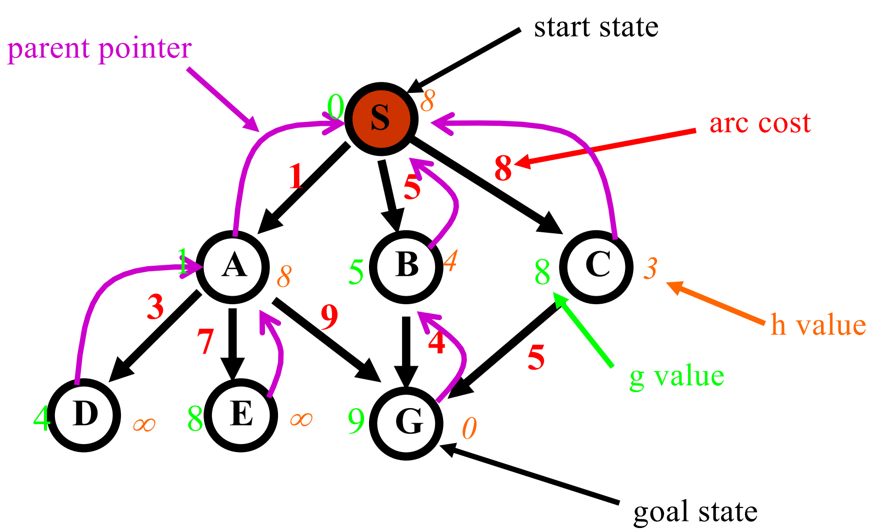
75

Algorithm A*

- Algorithm A with constraint that $h(n) \leq h^*(n)$
  - ***h*(n)*** = true cost of the minimal cost path from *n* to a goal.
- Therefore, ***h*(*n*)** is an **underestimate** of the distance to the goal
- *h*() is **admissible** when $h(n) \leq h^*(n)$
  - Guarantees optimality
- A* is **complete** whenever the branching factor is finite, and every operator has a fixed positive cost
- A* is **admissible**

76

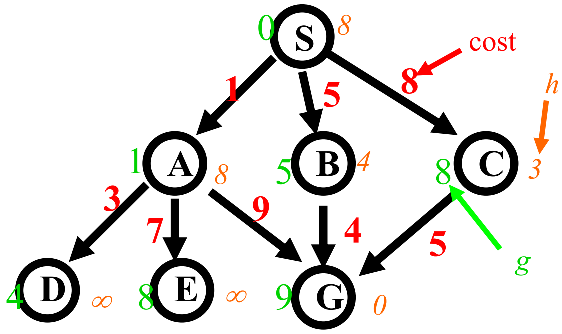
Example Search Space Revisited



77

Example

n	$g(n)$	$h(n)$	$f(n)$	$h^*(n)$
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	∞	∞	∞
E	8	∞	∞	∞
G	9	0	9	0



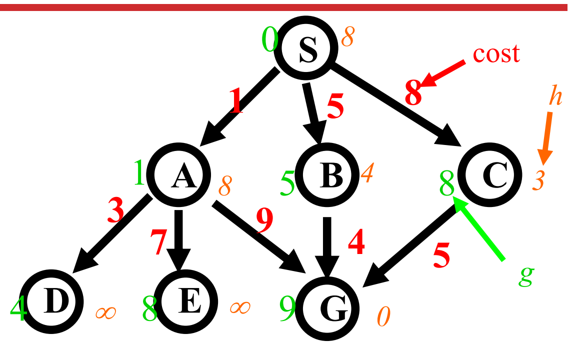
- $h^*(n)$ is the (hypothetical) perfect heuristic.
- Since $h(n) \leq h^*(n)$ for all n, h is admissible
- Optimal path = S B G with cost 9.

78

Greedy Search

$f(n) = h(n)$

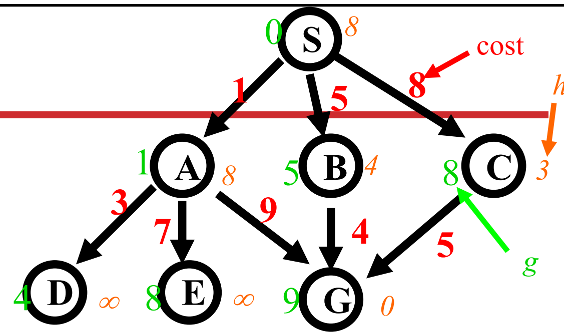
Node	**exp. node list**
	{ S(8) }
S	{ C(3) B(4) A(8) }
C	{ G(0) B(4) A(8) }
G	{ B(4) A(8) }



- Solution path found is S C G, 3 nodes expanded.
- Fast!! But NOT optimal.

79

A* Search



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

```
node      exp. nodes list
          { S(8) }
 S        { A(9) B(9)  C(11) }
 A        { B(9) G(10) C(11) D(∞) E(∞) }
 B        { G(9) G(10) C(11) D(∞) E(∞) }
 G        { C(11) D(∞) E(∞) }
```

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast, *and* optimal

80

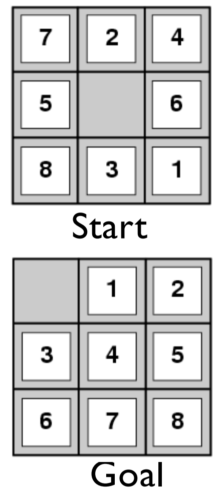
Admissibility and Optimality

- Intuitively:
  - When A* finds a path of length *k*, it has already tried **every other path which can have length ≤ k**
  - Because all frontier nodes have been sorted in ascending order of *f*(*n*)=*g*(*n*)+*h*(*n*)
- Does an admissible heuristic guarantee optimality for greedy search?
  - Reminder: *f*(*n*) = *h*(*n*), always choose node “nearest” goal
  - No sorting beyond that

81

Admissible heuristics

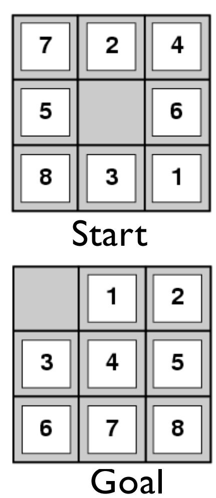
- E.g., for the 8-puzzle:
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance
    - (i.e., # of squares each tile is from desired location)
- $h_1(S)$ = ?
- $h_2(S)$ = ?



83

Admissible heuristics

- E.g., for the 8-puzzle:
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance
    - (i.e., # of squares each tile is from desired location)
- $h_1(S) = 8$
- $h_2(S) = 3+1+2+2+2+3+3+2 = 18$



84

Dealing with Hard Problems

- For large problems, A* often requires too much space.
- Two variations conserve memory: IDA* and SMA*
- IDA* – iterative deepening A*
  - uses successive iteration with growing limits on f. For example,
    - A* but don't consider any node n where $f(n) > 10$
    - A* but don't consider any node n where $f(n) > 20$
    - A* but don't consider any node n where $f(n) > 30$, ...
- SMA* – Simplified Memory-Bounded A*
  - Uses a queue of restricted size to limit memory use
  - Throws away the "oldest" worst solution

85

What's a Good Heuristic?

- If $h_1(n) < h_2(n) \leq h^*(n)$ for all n, then:
  - Both are admissible
  - h_2 is strictly better than (**"dominates"**) h_1
- So… how do we find one?

1. Relaxing the problem:
   - Remove constraints to create a (much) easier problem
   - Use the solution cost for this problem as the heuristic function
2. Combining heuristics:
   - Take the max of several admissible heuristics
   - Still have an admissible heuristic, and it's better!

86

Finding a Good Heuristic (2)

3. Use statistical estimates to compute *h*
   - May lose admissibility
4. Identify good features, then use a learning algorithm to find a heuristic function
   - Also may lose admissibility

- **Why are these a good idea, then?**
  - Machine learning can give you answers you don't "think of"
  - Can be applied to new puzzles without human intervention
  - Often works

87

Some Examples of Heuristics?

- 8-puzzle?
  - Manhattan distance
- Driving directions?
  - Straight line distance
- Crossword puzzle?
- Making a medical diagnosis?

88

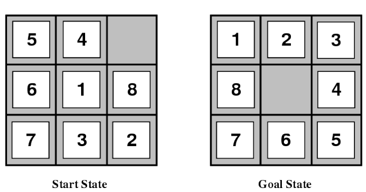
Summary: Informed Search

- **Best-first search:** general search where the *minimum-cost nodes* (according to some measure) are expanded first.
- **Greedy search:** uses *minimal estimated cost h*(*n*) to the goal state as measure. Reduces search time, but is neither complete nor optimal.
- **A* search:** combines UCS and greedy search
  - $f(n) = g(n) + h(n)$
  - A* is complete and optimal, but space complexity is high.
  - Time complexity depends on the quality of the heuristic function.
- IDA* and SMA* reduce the memory requirements of A*.

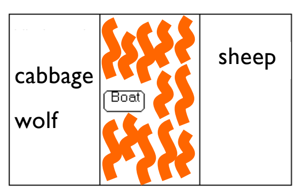
89

Class Exercise: Creating Heuristics

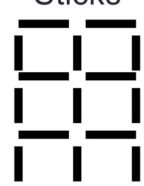
8-Puzzle



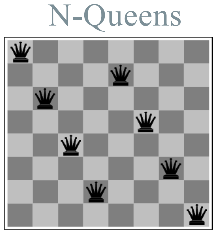
Boat Problems



Remove 5 Sticks



N-Queens

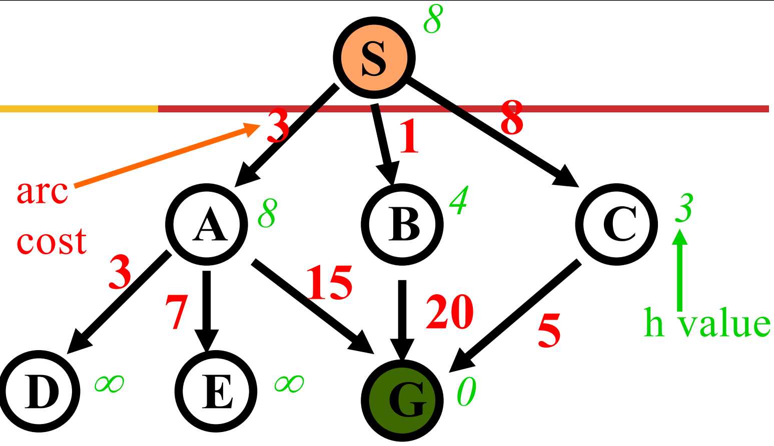


Water Jug Problem

Route Planning

90

Class Exercise



Apply the following to search this space. At each search step, show: the current node being expanded; *g*(*n*) (path cost so far); *h*(*n*) (heuristic estimate); *f(n)* (evaluation function); and *h**(*n*) (true goal distance).

Depth-first search
Breadth-first search
A* search
Uniform-cost search
Greedy search

91