EECS 3401 — AI and Logic Prog. — Lecture 11

Adapted from slides of Yves Lesperance

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

October 28, 2020

- Today: **Search Algorithms**
- Required reading: Russell & Norvig Chapters 3.1–3.4

Algorithms for Search

Inputs:

- an **initial state**—a specific world state or a set of world states representing the agent's knowledge, etc.
- a **successor function** $S(x)$ = a set of states that can be reached from state x via a single action
- a **goal test**—a function that can be applied to a state and returns *True* if the state satisfies the goal condition
- a **step cost function** $C(x, a, y)$ which determines the cost of moving from state x to state y using action a.
  $C(x, a, y) = \infty$ if a does not lead to y from x

Algorithms for Search

Output:

- a sequence of states leading from the initial state to a state satisfying the goal test
- The sequence might be
  - annotated by the name of the actions used;
  - optimal in cost (for some algorithms)

Algorithms for Search

Obtaining the action sequence:

- The set of successors of a state x might arise from different actions, e.g.,
  - $x \Rightarrow a \Rightarrow y$
  - $x \Rightarrow b \Rightarrow z$
- Successor function $S(x)$ yields a set of states that can be reached from x via any single action
- Rather than just return a set of states, we might want to annotate these states by the action used to obtain them:
  - $S(x) = \{\langle y, a\rangle, \langle z, b\rangle\}$
    y via action a, z via action b
  - $S(x) = \{\langle y, a\rangle, \langle y, b\rangle\}$
    y via action a, also y via action b

Tree Search

Tree Search: a way to explore the state space in a tree-like fashion

- Root = initial state
- *Frontier* = set of unexplored nodes/states (so far)
- Branch = action that is possible in current node/state + the resulting node/state

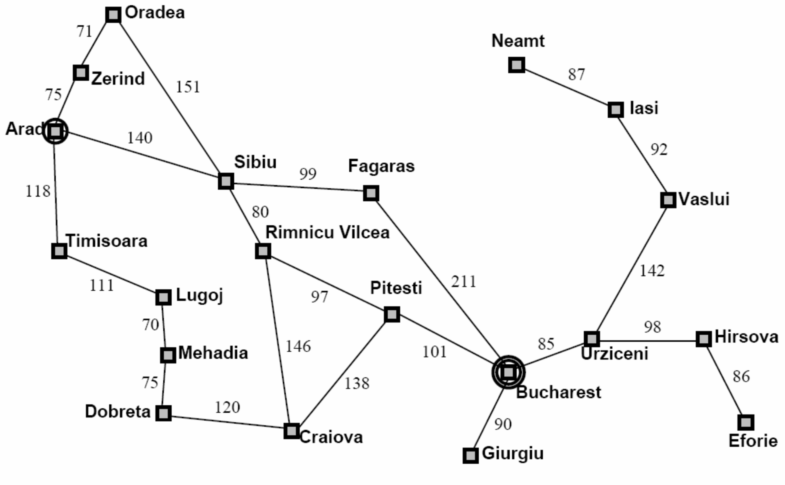
  Use the successor state function to **expand** the current state

```
TreeSearch(Frontier, Successors, Goal?)
    if Frontier is empty
        return failure
    Current := select state from Frontier
    if Goal?(Current)
        return Current
    NewFrontier := (Frontier - {Current})
                        + Successors(Current)
    return TreeSearch(NewFrontier, Successors, Goal?)
```

Tree search in Prolog

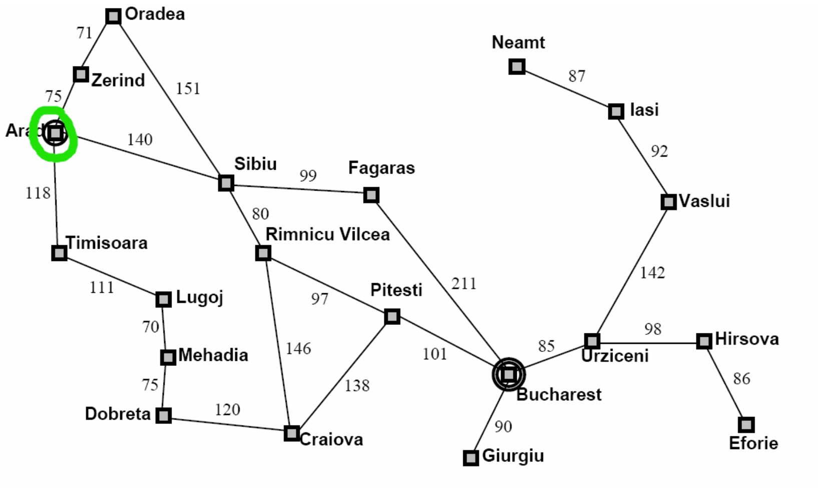
```
treeS([[State|Path]|_],Soln) :-
        Goal(State), reverse([State|Path], Soln).

treeS([[State|Path]|Frontier],Soln) :-
        GenSuccessors(State,Path,NewPaths),
        merge(NewPaths,Frontier,NewFrontier),
        treeS(NewFrontier,Soln).
```

Example: Travelling in Romania



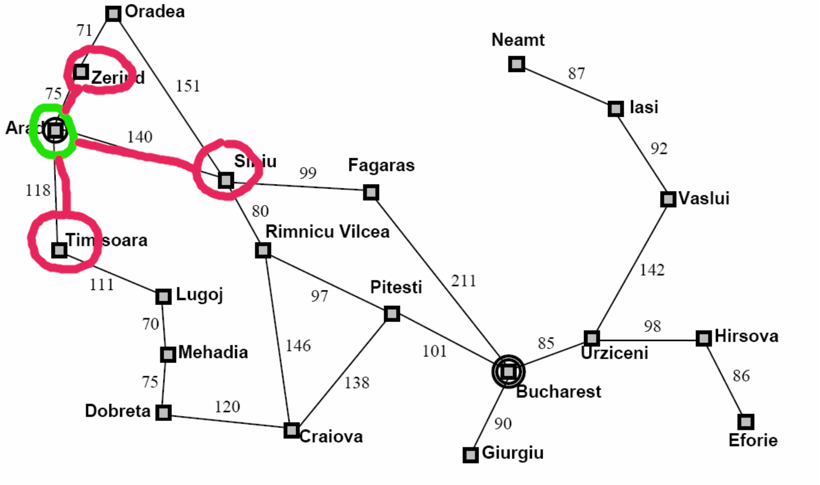
Currently in Arad, need to get to Bucharest by tomorrow to catch a flight

Example: Travelling in Romania



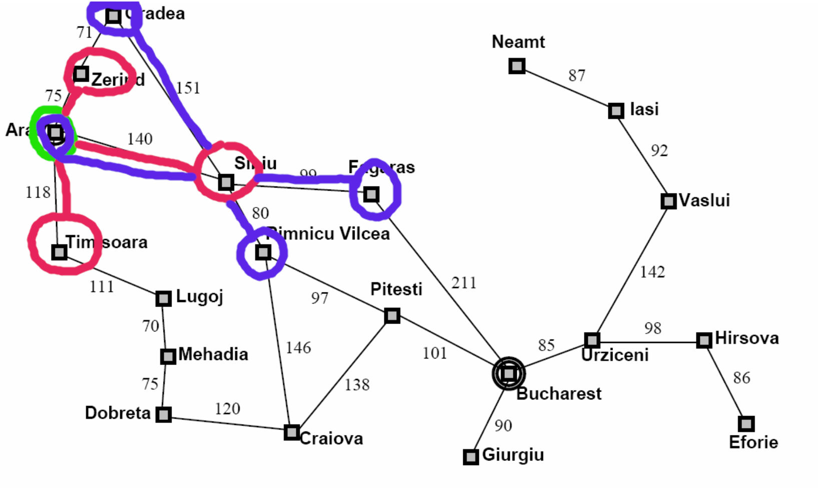
Frontier: {Arad}

Example: Travelling in Romania



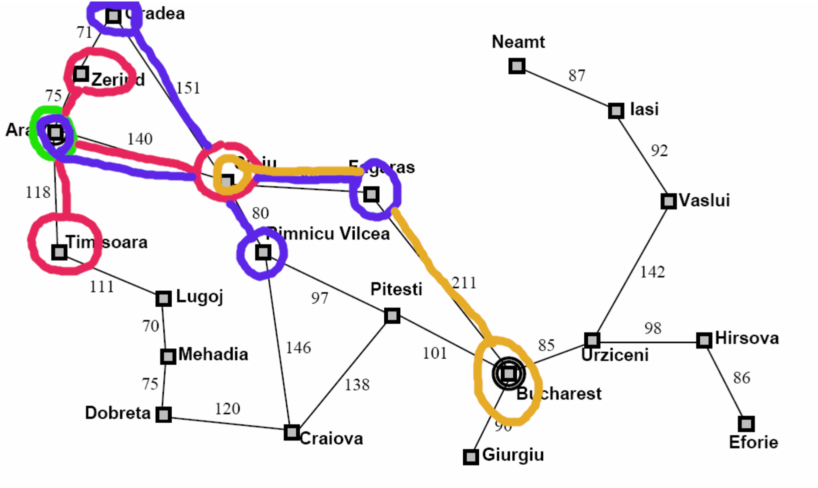
Frontier: {Zerind, Timisoara, Sibiu}

Example: Travelling in Romania



Frontier: {Zerind, Timisoara, Arad, Oradea, Fagaras, RimnicuVilcea }

Example: Travelling in Romania



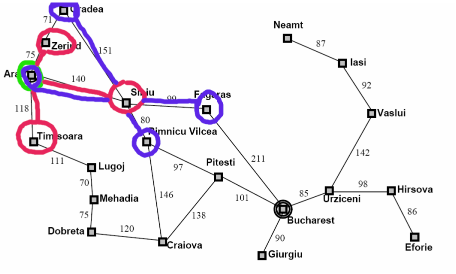
Frontier: {Zerind, Timisoara, Arad, Oradea, Sibiu, Bucharest, ... }

Example: Travelling in Romania

- Solution: Arad–Sibiu–Fagaras–Bucharest
- Cost: $140 + 99 + 211 = 450$
- That's not the only solution
- Alternative: Arad–Sibiu–Rimnicu Vilcea–Pitesti–Bucharest
- Cost: $140 + 80 + 97 + 101 = 418$
- Alternative is cheaper!

- *Way of picking the next node to expand is important*

Example: Travelling in Romania



- It gets worse with cycles
- Frontier: {Zerind, Timisoara, Arad, Oradea, Fagaras, RimnicuVilcea}
- What if we chose to expand Arad instead?
- Infinite search tree, no solution

Selection Rule

The order of selecting frontier states to expand is of critical importance to

- whether the solution will be found at all
- the cost of the solution (if one is found)
- the time and space consumed by search.

Critical Properties of Search

- **Completeness** — Will the search always find a solution if one exists?
- **Optimality** — Will the search always find the cheapest solution?
- **Time complexity** — What is the maximum number of nodes that can be generated or expanded?
- **Space complexity** — What is the maximum number of nodes that have to be stored in memory?

Uninformed Search Strategies

- Adopt a fixed selection rule
- Rule is always the same regardless of the specific search problem
- Do not take into account any domain-specific information
- Popular uninformed search techniques:
  - Breadth-first
  - Uniform-cost
  - Depth-first
  - Depth-limited
  - Iterative-deepening

Selecting versus Sorting

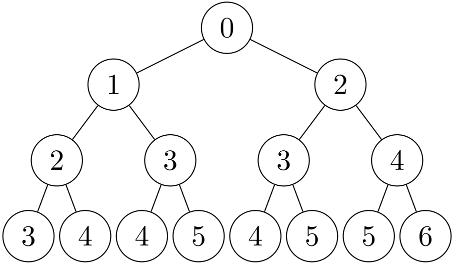
- The problem of selection can be reframed as the problem of sorting
- Commit to the following selection rule:
    1. Arrange frontier elements according to some order
    2. Always select the first element
- Then, the sorting criteria define the search strategy
- Will adopt this approach

Breadth-First

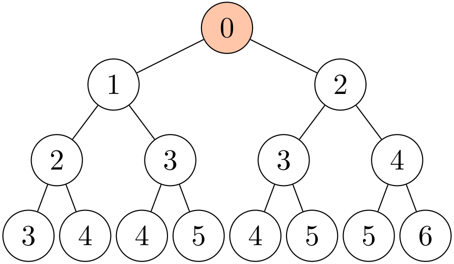
- Place the successors of the current state at the **end** of the frontier
- Then, frontier behaves as a First-In-First-Out queue
- Example:
  - Let the states be non-negative integers {0, 1, 2, 3, ...}
  - Successor state: $S(n) = \{n+1, n+2\}$
  - Initial state: 0
  - Goal state: 5

Example

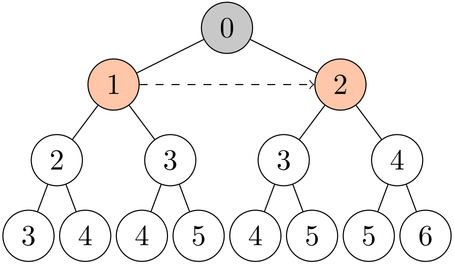
- Let the states be non-negative integers {0, 1, 2, 3, ...}
- Successor state: $S(n) = \{n+1, n+2\}$
- Initial state: 0
- Goal state: 5



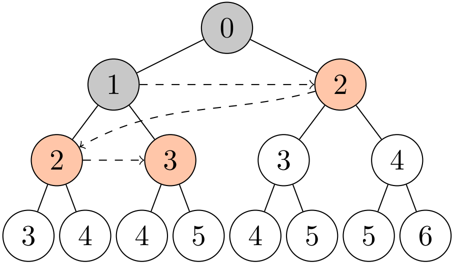
Example



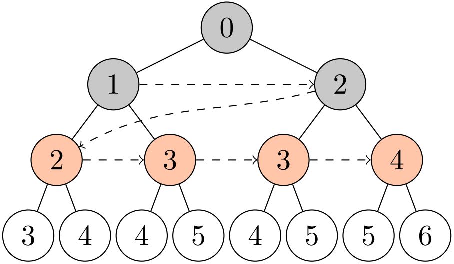
Example



Example



Example



Breadth-First Properties

Time Complexity — number of nodes generated (Let b the maximum number of children per node)

- Level 0 (root): 1
- Level 1: b
- Level 2: $b \cdot b = b^2$
- Level 3: $b \cdot b^2 = b^3$
- ...
- Level d: b^d
- Level $d+1$: $b^{d+1} - b = b(b^d - 1)$

  when last node in level d is the goal and does not need to be expanded
- **Total**: $1 + b + b^2 + \ldots + b^{d-1} + b^d + b(b^d - 1)$
- $O(b^{d+1})$ — exponential, so only works for small instances

Breadth-First Properties

Space Complexity — number of nodes stored

- $O(b^{d+1})$: If the goal node is the last node at level d, all of the successors of the other nodes will be on the frontier when we get to the goal, i.e., $b(b^d - 1)$

Breadth-First Properties

Optimality

- Will find the shortest solution
- Least-cost solution? Generally, **no**. Will if all step costs are equal.

Breadth-First Properties

Space complexity is a real problem

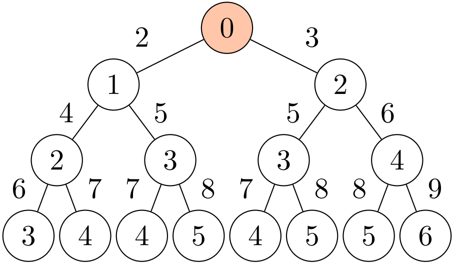
- Let $b = 10$, and say 1000 nodes can be expanded per second and each node requires 100 bytes of storage.

Depth	Nodes	Time	Memory
1	1	1 ms	100 bytes
6	10^6	18 min	111 MB
8	10^8	31 hrs	11 GB

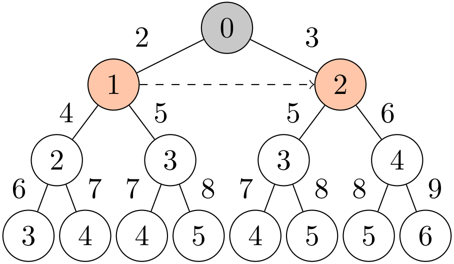
Uniform-Cost Search

- Keep the frontier sorted in increasing cost of the path to a node (behaves like a priority queue)
- Always expand the least-cost node
- Identical to breadth-first if each transition has the same cost
- Example:
    - Let the states be non-negative integers {0, 1, 2, 3, ...}
    - Successor state: $S(n) = \{n+1, n+2\}$
    - Action $n+1$ has cost 2, action $n+2$ has cost 3

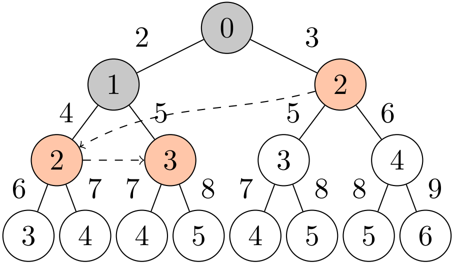
Example



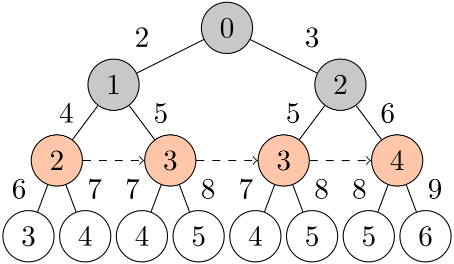
Example



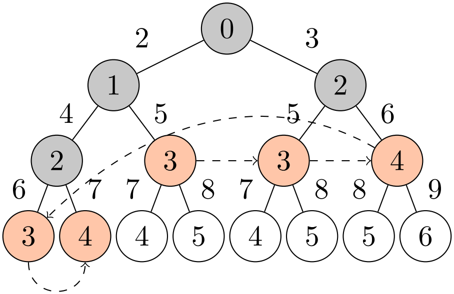
Example



Example



Example



Uniform-Cost Search

Completeness

- Assume each transition has costs $\geq \epsilon > 0$ (otherwise can have an infinite path with finite cost)
- The previous argument used for breadth-first search holds: the cost of the expanded state must increase monotonically
- Algorithm expands nodes in order of increasing path costs

Uniform-Cost Search

Time and Space Complexity

- $O\left(b^{\frac{C^\star}{\epsilon}}\right)$, where $C^\star$ is the cost of the optimal solution
- Difficulty is that there may be many long paths with cost $\leq C^\star$ — uniform cost search must explore them all

Uniform-Cost Search

Optimality

- If each transition has cost $\geq \epsilon > 0$, **finds optimal solution**
- Explores paths in the search space in increasing order of cost. Thus, must find minimum cost path to a goal before finding any higher cost paths.

Uniform-Cost Search: proof of optimality

Let $c(n)$ be the cost of the path to node n.

1. If node n_2 is expanded after node n_1, then $c(n_1) \leq c(n_2)$.

Proof:

- If n_2 was already on the frontier when n_1 was expanded, then $c(n_2) \geq c(n_1)$ (otherwise, n_1 wouldn't have been selected for expansion)
- If n_2 was added to the frontier as the result of expanding n_1, then $c(n_2) \geq c(n_1)$ (since the path to n_2 extends that to n_1)
- If n_2 is a successor of a node n_3 that was already on the frontier or added as the result of expanding n_1, then $c(n_2) > c(n_3)$ and $c(n_3) \geq c(n_1)$ by the previous arguments

Uniform-Cost Search: proof of optimality

2. When n is expanded, every path with cost strictly less than $c(n)$ has already been expanded.

Proof:

- Let $\langle Start, n_0, n_1, \ldots, n_k \rangle$ be a path with cost less than $c(n)$. Len n_i $(0 \leq i < k)$ be the last node on this path that has been expanded
- n_{i+1} must be on the frontier, also $c(n_{i+1}) < c(n)$ since the cost of the entire path to n_k is less than $c(n)$
- But then uniform-cost would have expanded n_{i+1} and not n
- Thus, every node on this path must already be expanded, i.e., this path has already been expanded.

Uniform-Cost Search: proof of optimality

2. The first time uniform-cost expands a state, it has found the minimal-cost path to it (it may later find other paths to the same state)

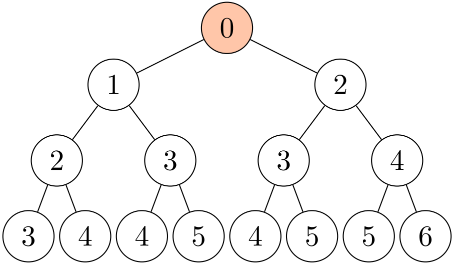
Proof:

- No cheaper path exists, else that path would have been expanded before
- No cheaper path will be discovered later, as all those paths must be at least as expensive
- So, when a goal state is expanded, the path to it must be optimal

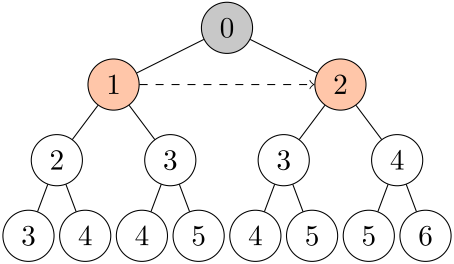
Depth-First Search

- Place the successors of the current state at the **front** of the frontier
- Frontier behaves like a stack

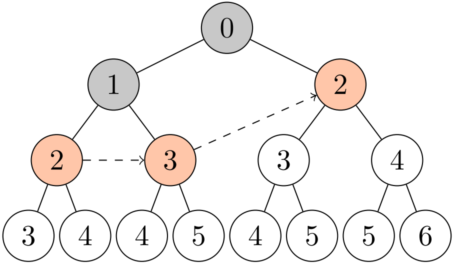
Example



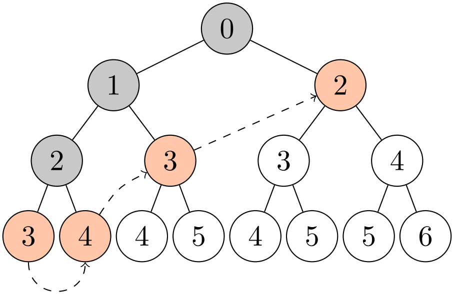
Example



Example



Example



Depth-First Properties

Completeness? No

- Infinite paths cause incompleteness. Typically come from cycles in the search space.
- If we prune paths with duplicate states, we obtain completeness, provided that the search space is finite

Optimality? No

- Can find goal along a longer branch

Depth-First Properties

Time Complexity

- $O(b^m)$, where m is the length of the longest path in the state space
- Why? Worst case expands $1 + b + b^2 + \ldots + b^{m-1} + b^m = O(b^m)$ nodes (assuming no cycles)

- Very bad if m is much larger than d, but if there are many solution paths, depth-first can be much faster than breadth-first
- At each step, frontier nodes are backtrack points

Depth-First Properties

Space Complexity

- $O(b \cdot m)$ — linear space!
- Only explores one path at a time
- The frontier only contains the deepest states on the current path along with the backtrack points
- Can even reduce to $O(m)$ if we generate siblings one at a time

End of Lecture

- Next time: **Algorithms for Search continued**