

Problem solving by search

Tomáš Svoboda, Matěj Hoffmann, and Petr Pošík

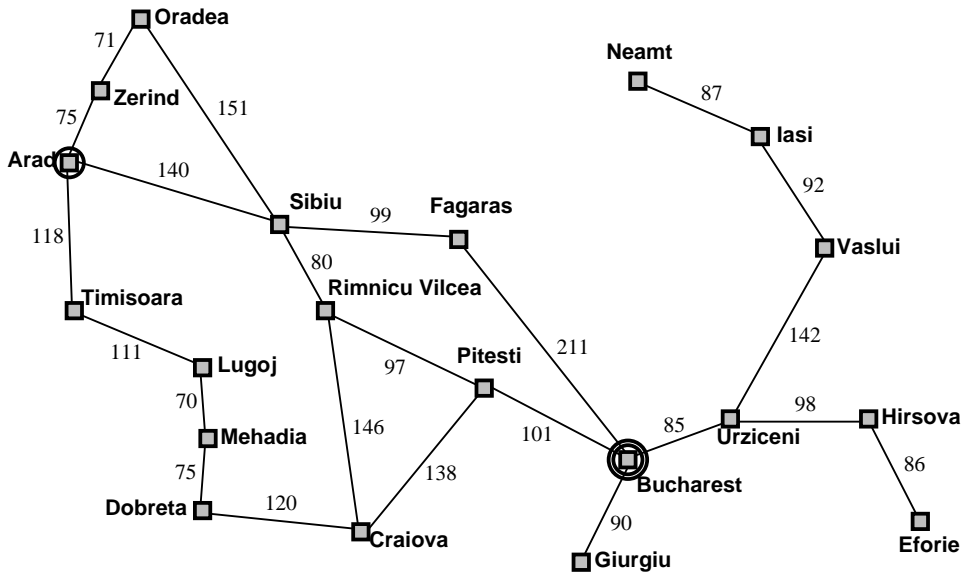
Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 1, 2023

Outline

- ▶ Search problem. *What do you want to solve?*
- ▶ State space graphs. *How do you formalize/represent the problem? Problem abstraction.*
- ▶ Search trees. *Visualization of the algorithm run.*
- ▶ Strategies: which tree branches to choose?
- ▶ Strategy/Algorithm properties. *Memory, time, ...*
- ▶ Programming infrastructure.

Example: Traveling in Romania



3 / 29

Notes

Ok, start with a simple one, almost everybody knows about the navigation - path planning problem. Waze, Garmin, ... Here, the problem can be transferred into a graph quite directly - a map is a kind of a graph, states are location in a city.

Can you think about more problems?

For example:

- Touring problems. Special case: Traveling salesperson problem – each city must be visited exactly once.
- Planning robot movements – mobile robot or manipulator.
- VLSI (chip) layout.
- ...

Traveling Example: State and Actions

Goal:

be in Bucharest

Problem formulation:

states: position in a city (cities)

actions: at a crossing, select a road

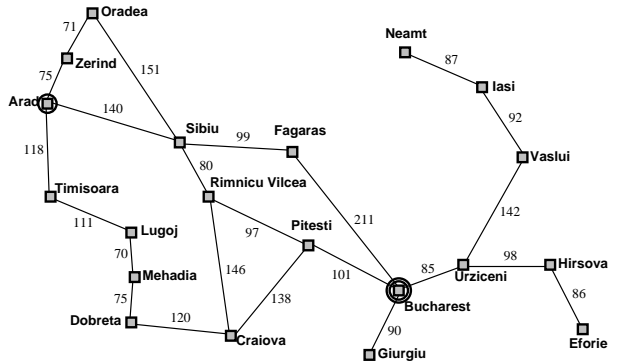
Solution:

Sequence of cities (path)

(action sequence [2])

Cost:

Energy, time, tolls, ...



Notes

Classical problem from the Book [2], we use it, too.

states and actions will be frequently discussed in several lectures and algorithms. It is important to fully understand them. At crossings, we need to decide about the next road - this is the action. We assume that we reach the next crossing - outcome of the action.

Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states?
actions?
solution?
cost?

Notes

Also known as $n - 1$ puzzle.

- States: Location of each of the 8 tiles and the blank.
- Number of states: $9!$
- Initial state: any state. (Note that any given goal state can be reached from exactly half of the initial states.)
- Actions: Movements of the blank space: Left, Right, Up, Down (or a subset of these)
- Solution / goal test: Check whether state matches the goal configuration.
- Path cost: nr. steps in the path (each step costs 1)

Toy problem (3.2.1) from [2].

A Search Problem

- ▶ **State space** (including Start/Initial state): position, board configuration,
- ▶ **Actions** : drive to, Up, Down, Left ...
- ▶ **Transition model** : Given state and action return state (and **cost**)
- ▶ **Goal test** : Are we done?

Notes

We will use the terminology through the next 5-6 lectures; also for Markov (Sequential) Decision Processes, Reinforcement Learning.

Make a mental test: You are a robot, going from home to school. What would be **states**, **actions**, **transition model**, **goal test**?

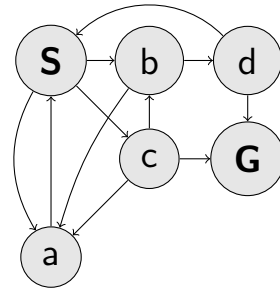
Transition model can be also understood as mapping between actions and outcome.

State Space Graphs

State space graph: a representation of a search problem

- ▶ Graph Nodes – states – are abstracted world configurations
- ▶ Arcs represent action results
- ▶ Goal test – a set of goal nodes

Each state occurs only *once* in a state (search) space.



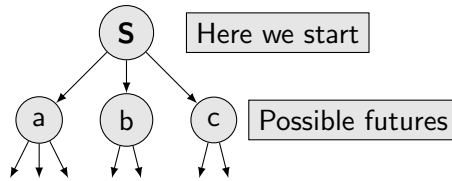
Notes

Formalizing a real world problem – (creating) a state space graph – could be a problem in itself. I put creating into brackets as it may be also infinite.

Close connection to graph algorithms like Dijkstra, Floyd-Warshall.

- Graph algorithms assume complete info about the graphs - the main input.
- For many real-world problems, the graph is not known in advance.
- The state space graph is *revealed during the search*. The graph serves as an abstraction - mental model - rather than as an actual data representation.
- Many real world problems have too many vertices, think about $n - 1$ puzzle or chess, number of possible configurations is enormous.
- A solution can be actually quite shallow.

Search Trees

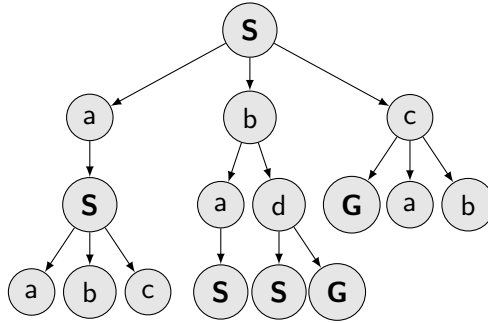
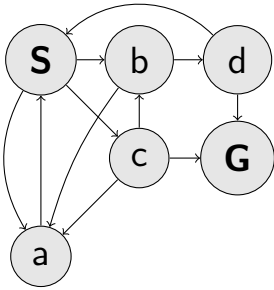


- ▶ A “what if” tree of plans and their outcomes
- ▶ Start node is the root
- ▶ Children are successors
- ▶ Nodes show/contains states, but correspond to *plans* that achieve those states

Notes

- What if decision about an action, repeats ...
- Nodes in the search tree are not the same as the nodes in the state space graph.

State Space Graphs vs. Search Trees

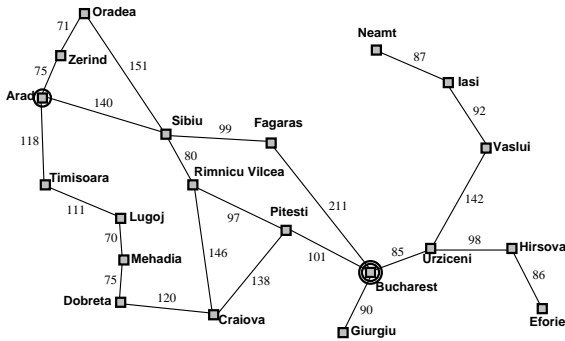


How big is the search tree?

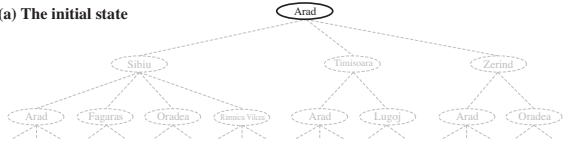
Notes

- 'S' denotes Start; 'G' denotes Goal.
- There could be *multiple* search trees above *one* state space, depending on the algorithm.
- When going through the unfolding of the *search tree* (on the right), one may already introduce that there are *leaf nodes* at the *frontier*; one of them always gets *expanded*.
- A search tree can be *much bigger* than the state space. (E.g., states 'S', 'a', ... appear more than once in the search tree...)
- Note also that search does not have stop when 'G' is reached for the first time. We may need the shortest path...
- These properties will be discussed next.

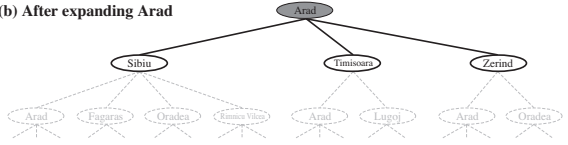
From problem/transition graph to search tree (Romania)



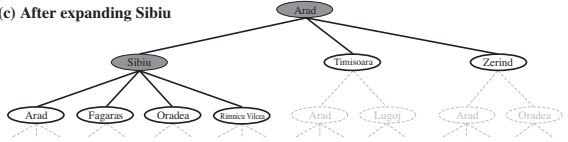
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



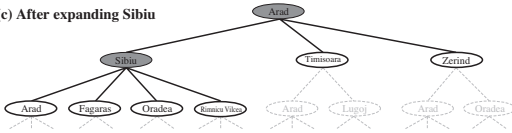
Problem/transition graph is revealed incrementally.
The revealing strategy can be visualized as a search tree.

Notes

Images from [2].

Tree search algorithm

(c) After expanding Sibiu



function TREE_SEARCH(problem) **return** a solution or failure

initialize by using the initial state of the problem

loop

if no candidates for expansion **then return** failure

else choose a leaf node for expansion

end if

if the node contains a goal state **then return** the solution

end if

Expand the node and add the resulting nodes to the tree

end loop

end function

Notes

A *general* tree search algorithm. Individual search algorithms vary primarily in how they choose which state to expand next – the “search strategy”.

Search (algorithm) properties

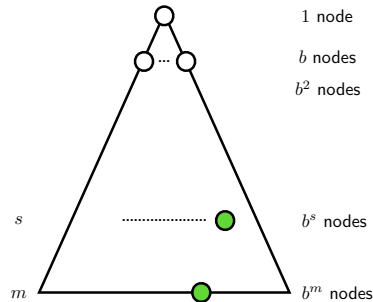
- ▶ Guaranteed to find a solution (if exists)? **Complete?**
- ▶ Guaranteed to find the least cost path? **Optimal?**
- ▶ How many steps - an operation with a node? **Time** complexity?
- ▶ How many nodes to remember? **Space/Memory** complexity?

How many nodes in a (search) tree? What are tree parameters?

Notes

Draw a (symbolic—think about a triangle) sketch of a (search) tree. It may grow upwards or downwards. How would you characterize/parametrize *size* of a tree.

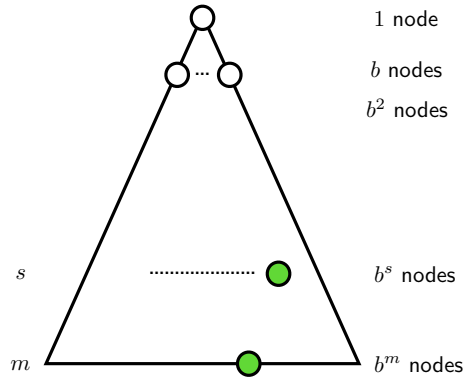
- Depth of the tree d .
- Max-Depth of the tree m . Can be ∞ .
- Branching factor b .
- s denotes the shallowest Goal.
- How many nodes in the whole tree?



Strategies

How to traverse/build a search tree?

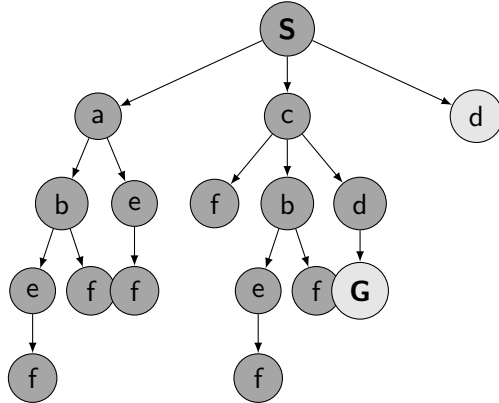
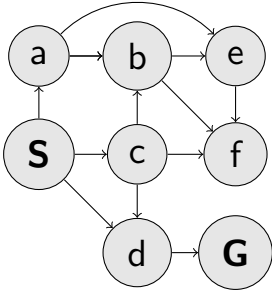
- ▶ **Depth** of the tree d .
- ▶ **Max-Depth** of the tree m . Can be ∞ .
- ▶ **Branching** factor b .
- ▶ s denotes the **shallowest Goal**.
- ▶ How many nodes in the whole tree?



Notes

It is perhaps worth to remember that the search tree is built as the algorithm goes. Or better said, the tree is a human friendly representation of the machine run.

Depth-First Search (DFS)



What are the DFS properties (complete, optimal, time, space)?

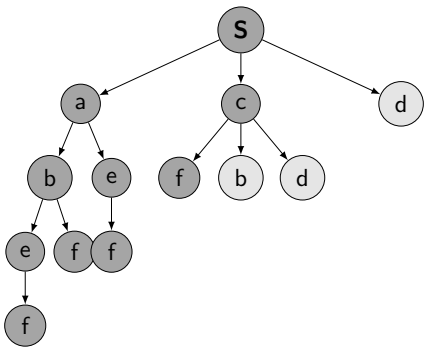
Notes

- In animation, we will do the expansion step at once.
- Expanded (explored) nodes become darker gray.
- *frontier* - set of nodes are light gray.
- When to stop the search?
- Thinking about optimality, what is the best solution we seek?

DFS properties

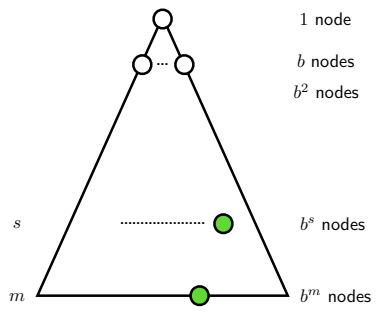
b, m, s , Time complexity?

- ▶ Time complexity? **A** $\mathcal{O}(bm)$
- ▶ Space complexity? **B** $\mathcal{O}(b^m)$
- ▶ Complete? **C** $\mathcal{O}(m^b)$
- ▶ Optimal? **D** ∞



Notes

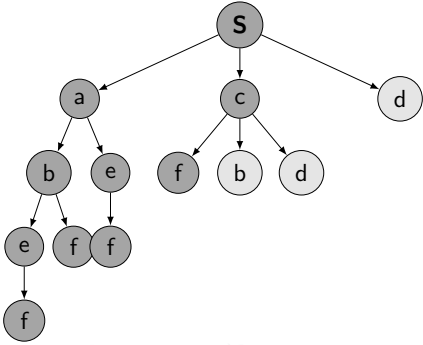
- Time, can process the whole tree: b^m
- Space, only the path so far: bm (a path from root to leaf (m), plus siblings on the path are also on the frontier ($b \times m$))
- Completeness: m may be ∞ hence, not in general
- Optimality: No! It just takes the first solution found.



DFS properties

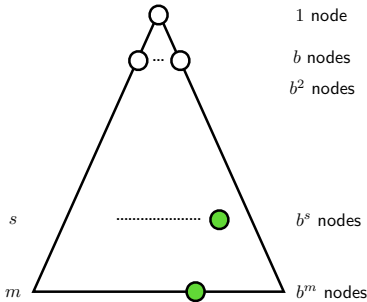
b, m, s , Space complexity?

- ▶ Time complexity? **A** $\mathcal{O}(bm)$
- ▶ Space complexity? **B** $\mathcal{O}(b^m)$
- ▶ Complete? **C** $\mathcal{O}(m^b)$
- ▶ Optimal? **D** ∞

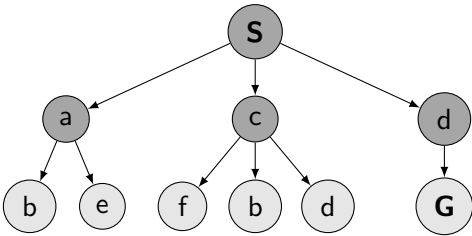
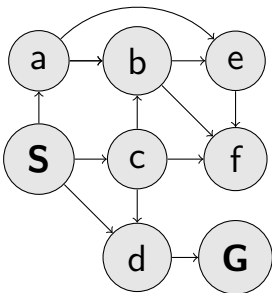


Notes

- Time, can process the whole tree: b^m
- Space, only the path so far: bm (a path from root to leaf (m), plus siblings on the path are also on the frontier ($b \times m$))
- Completeness: m may be ∞ hence, not in general
- Optimality: No! It just takes the first solution found.



Breadth-First Search (BFS)



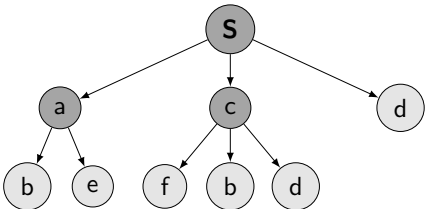
What are the BFS properties?

Notes

BFS properties

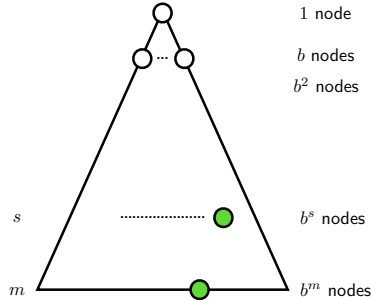
b, m, s , Time complexity?

- ▶ Time complexity? **A** $\mathcal{O}(bm)$
- ▶ Space complexity? **B** $\mathcal{O}(b^m)$
- ▶ Complete? **C** $\mathcal{O}(m^b)$
- ▶ Optimal? **D** $\mathcal{O}(b^s)$



Notes

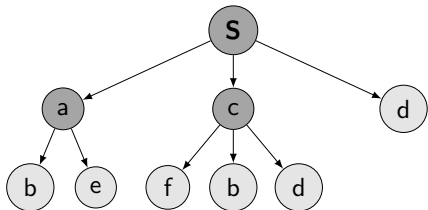
- Time, can process the whole tree until s : b^s , well actually $b + b^2 + b^3 + \dots + b^s$ but the last layer vastly dominates. Try some calculations for various b .
- Space, all the frontier: b^s
- Completeness: Yes!
- Optimality, it does not miss the shallowest solution, hence if all the transition costs are 1: Yes!



BFS properties

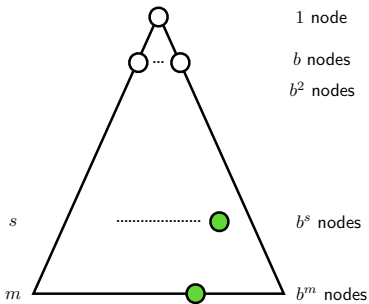
b, m, s , Space complexity?

- ▶ Time complexity? **A** $\mathcal{O}(bm)$
- ▶ Space complexity? **B** $\mathcal{O}(b^m)$
- ▶ Complete? **C** $\mathcal{O}(m^b)$
- ▶ Optimal? **D** $\mathcal{O}(b^s)$



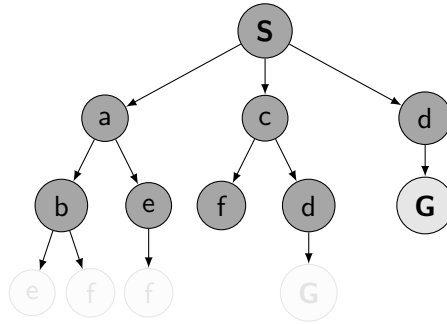
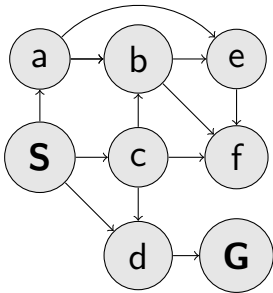
Notes

- Time, can process the whole tree until s : b^s , well actually $b + b^2 + b^3 + \dots + b^s$ but the last layer vastly dominates. Try some calculations for various b .
- Space, all the frontier: b^s
- Completeness: Yes!
- Optimality, it does not miss the shallowest solution, hence if all the transition costs are 1: Yes!



DFS with limited depth, $\text{maxdepth}=2$

Do not follow nodes with $\text{depth} > \text{maxdepth}$



Notes

- Remedy to DFS failing in infinite state spaces.
- Supplying a predetermined max depth – nodes at this depth are treated as if they had no successors.
- However, an additional source of algorithm *incompleteness*. Solution can obviously be deeper than maxdepth – unless we know something about the problem. Think about our map of Romania. There are 20 cities. Hence, $\text{maxdepth} = 19$ is a possible choice. Taking a closer look, any city can be reached from any other city in max. 9 steps (*state space diameter*), giving a more strict and hence better limit.

Iterative deepening DFS (ID-DFS)

- ▶ Start with `maxdepth = 1`
- ▶ Perform DFS with limited depth. Report success or failure.
- ▶ If failure, forget everything, increase `maxdepth` and repeat DFS

Is it not a terrible waste to forget everything between steps?

21 / 29

Notes

Really, how much do we repeat/waste? The “upper levels”, close to the root, are repeated many times. However, in a tree, most nodes are the bottom levels and nr. nodes traversed is what counts. More specifically, for a solution at depth s , the nodes on the bottom level are generated only once, those on the next-to-bottom level $2x$... children of the root are generated $s \times$. Compare the number of nodes generated ID-DFS vs. BFS:

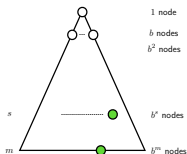
$$N(\text{ID-DFS}) = (s)b + (s-1)b^2 + (s-2)b^3 + \dots + (1)b^s$$

$$N(\text{BFS}) = b + b^2 + b^3 + \dots + b^s$$

Try some calculations for various s and b . For $b = 10$ and $d = 5$:

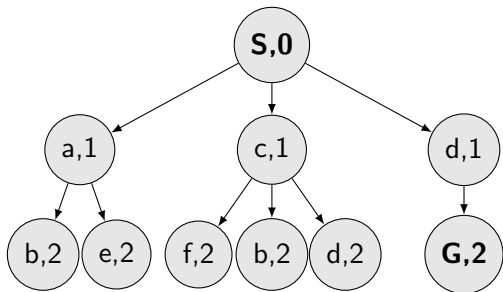
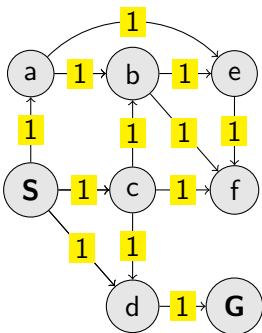
$$N(\text{ID-DFS}) = 50 + 400 + 3000 + 20000 + 100000 = 123450$$

$$N(\text{BFS}) = 10 + 100 + 1000 + 10000 + 100000 = 111110$$



(Example from [2].)

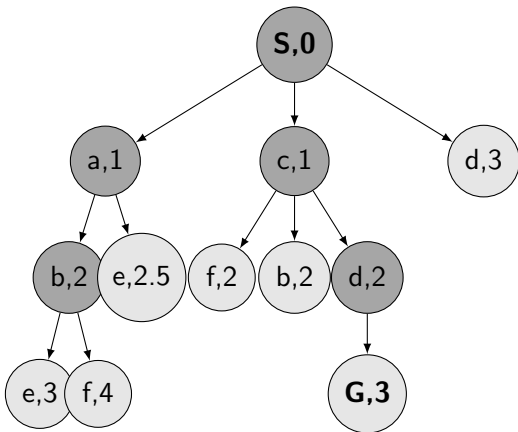
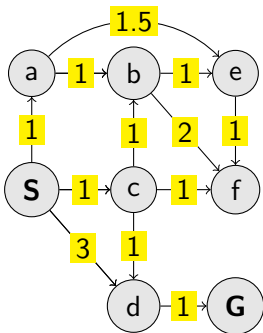
Cost sensitive search



► In BFS, DFS, node ±depth was the node-value.

Notes

Uniform Cost Search (UCS)



When to **check the goal** (and stop) the search? When visiting or expanding the node?

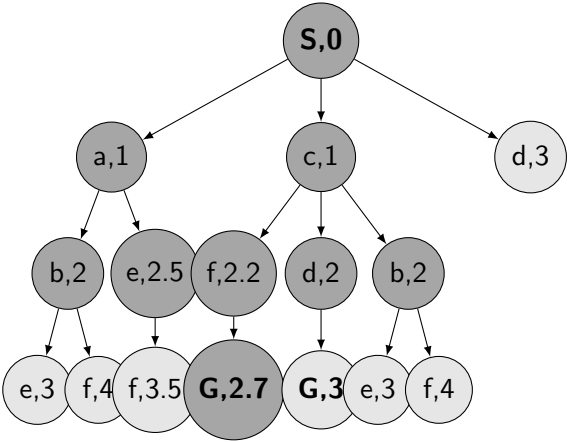
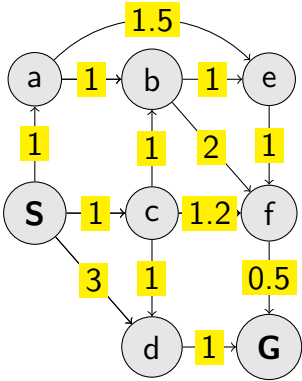
Notes

Simple extension of BFS. Instead of expanding shallowest node, the node with smallest path cost so far is expanded.

Two differences:

- Goal test applied to a node when *selected for expansion* – not when first generated. (First goal generated may be on a suboptimal path.)
- Test is added in case a better path is found to a node currently on the frontier.

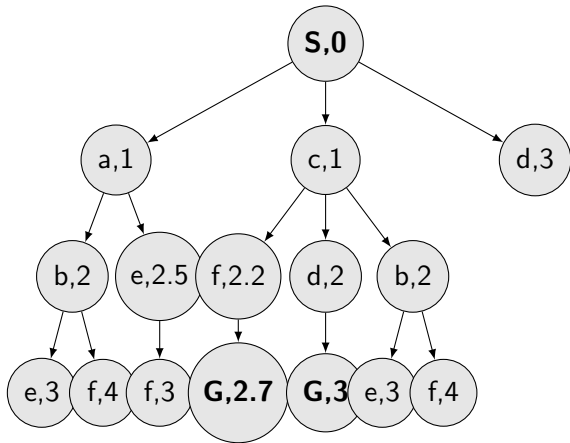
When to stop, when visiting or expanding?



Notes

UCS properties

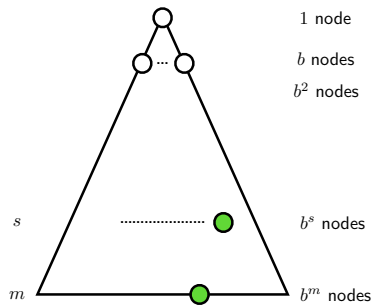
- ▶ Time complexity?
- ▶ Space complexity?
- ▶ Complete?
- ▶ Optimal?



Notes

Solution cost C^* , transition cost at least ϵ . Effective depth, roughly C^*/ϵ .

- Time: $b^{C^*/\epsilon}$
- Space: $b^{C^*/\epsilon}$
- Completeness: Yes!
- Optimality: Yes! Why?



How to organize nodes?

The Python examples are just suggestions, ...

- ▶ A dynamically linked structure (`list()`).
- ▶ Add a node (`list.insert(node)`).
- ▶ Take a node and remove from the structure (`node=list.pop()`).
- ▶ Check the Python modules `heapq`¹ and `queue`² for inspiration.

¹<https://docs.python.org/3.5/library/heapq.html>

²<https://docs.python.org/3.5/library/queue.html>

Notes

Very likely, you discussed `heapq` and `queue` in some programming, algorithms or data structures related courses.

What is the solution?

- ▶ We stop when **Goal** is reached.
- ▶ How do we construct the **path**?

Summary

- ▶ State space graph – an abstraction of a search problem.
- ▶ Search tree – visualization of the search algorithm run.
- ▶ Properties of search algorithms.

References, further reading

Some figures if from [2]. Chapter 2 in [1] provides a compact/dense intro into search algorithms.

[1] Steven M. LaValle.

Planning Algorithms.

Cambridge, 1st edition, 2006.

Online version available at: <http://planning.cs.uiuc.edu>.

[2] Stuart Russell and Peter Norvig.

Artificial Intelligence: A Modern Approach.

Prentice Hall, 3rd edition, 2010.

<http://aima.cs.berkeley.edu/>.