A* Search
Tree search algorithms

- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a. expanding states).
  - Every state is evaluated: *is it a goal state?*
Tree search example
Tree search example
Tree search example

function TREE-SEARCH(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
Best-first search

- Idea: use an evaluation function $f(n)$ for each node
  - $f(n)$ provides an estimate for the total cost.
    - Expand the node $n$ with smallest $f(n)$.

- Implementation:
  Order the nodes in fringe increasing order of cost.
Romania with straight-line dist.
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
- Best First search has \( f(n)=h(n) \)
- Uniform Cost search has \( f(n)=g(n) \)
Admissible heuristics

- A heuristic \( h(n) \) is admissible if for every node \( n \), \( h(n) \leq h^*(n) \), where \( h^*(n) \) is the true cost to reach the goal state from \( n \).

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

- Example: \( h_{SLD}(n) \) (never overestimates the actual road distance)

- **Theorem**: If \( h(n) \) is admissible, A* using TREE-SEARCH is optimal.
**Dominance**

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
- then $h_2$ dominates $h_1$
- $h_2$ is better for search: it is guaranteed to expand less or equal nr of nodes.

- Typical search costs (average number of nodes expanded):
  - $d=12$
    - IDS = 3,644,035 nodes
    - $A^*(h_1) = 227$ nodes
    - $A^*(h_2) = 73$ nodes
  - $d=24$
    - IDS = too many nodes
    - $A^*(h_1) = 39,135$ nodes
    - $A^*(h_2) = 1,641$ nodes
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.
Consistent heuristics

- A heuristic is **consistent** if for every node \( n \), every successor \( n' \) of \( n \) generated by any action \( a \),

\[
h(n) \leq c(n,a,n') + h(n')
\]

- If \( h \) is consistent, we have

\[
f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') = g(n) + h(n) = f(n)
\]

- i.e., \( f(n) \) is non-decreasing along any path.

- **Theorem:**
  If \( h(n) \) is consistent, \( A^* \) using \texttt{GRAPH-SEARCH} is optimal

It’s the triangle inequality! keeps all checked nodes in memory to avoid repeated states
A* search example
A* search example
A* search example
A* search example
A* search example
A* search example
Properties of A*

- **Complete?** Yes (unless there are infinitely many nodes with \( f \leq f(G) \), i.e. step-cost > \( \varepsilon \))
- **Time/Space?** Exponential: \( b^d \)
  except if: \( |h(n) - h^*(n)| \leq O(\log h^*(n)) \)
- **Optimal?** Yes
- **Optimally Efficient:** Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)
Memory Bounded Heuristic Search: Recursive BFS

- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.
- We remember the best f-value we have found so far in the branch we are deleting.
RBFS changes its mind very often in practice. This is because the $f=g+h$ become more accurate (less optimistic) as we approach the goal. Hence, higher level nodes have smaller $f$-values and will be explored first.

Problem: We should keep in memory whatever we can.
Simple Memory Bounded A*

- This is like A*, but when memory is full we delete the worst node (largest f-value).
- Like RBFS, we remember the best descendent in the branch we delete.
- If there is a tie (equal f-values) we delete the oldest nodes first.
- simple-MBA* finds the optimal *reachable* solution given the memory constraint.
- Time can still be exponential.

A Solution is not reachable if a single path from root to goal does not fit into memory.