Uninformed (blind) Search

- Reading: Russell and Norvig, ch. 3
  - The material here is based on Russell's slides as well as a version of the slides at UC Irvine: https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-03-UninformedSearch.pdf
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search
  - Depth limited search

Uninformed (or “blind”) search strategies

- [A fair amount of these notes should be familiar from data structures]
- Only use information from problem statement
    - Not collecting new information during search (offline problem solving)
        - Ex: if traveling, use only distances, speed limits, other fixed data – no traffic updates
    - No measure of likelihood of goal state in subtree of a node
- All goal states are equally desirable
    - What does breadth-first search guarantee? (recall distinction between nodes and states)

Options for uninformed search

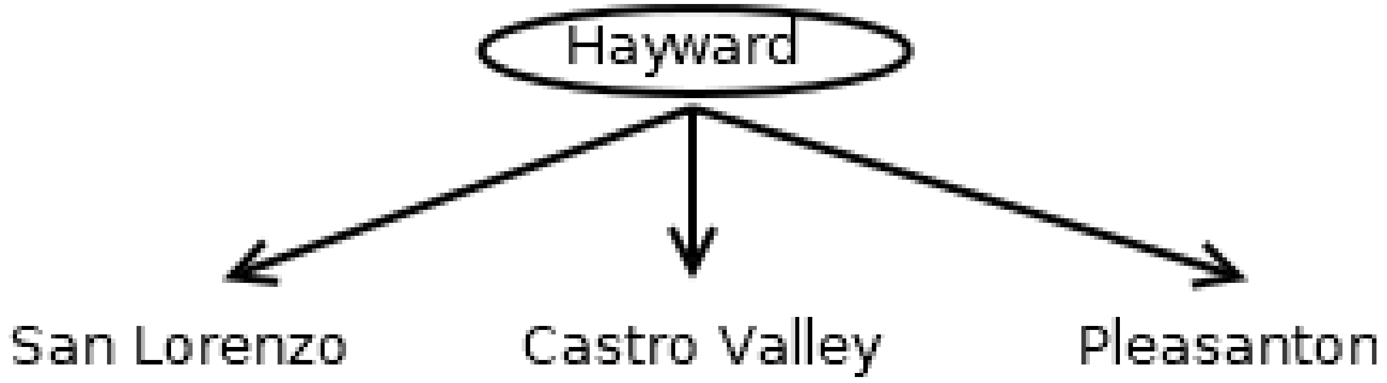
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Search Strategies (repeat from last slide of previous set of notes)

- Differ in choices of order of expanding nodes
  - [*Fringe* = current candidates for expansion]
- Can be compared on:
  - *Completeness* – does it always find a solution node if one exists?
  - Time complexity – biggest factor is number of nodes generated
  - Space complexity – number of nodes can be infinite, fitting nodes in memory also a concern
  - Optimality – guaranteed to find best solution?
- How to measure time and space complexity
  - Max branching factor = number of children
  - Depth of least cost solution
  - Max depth of state space

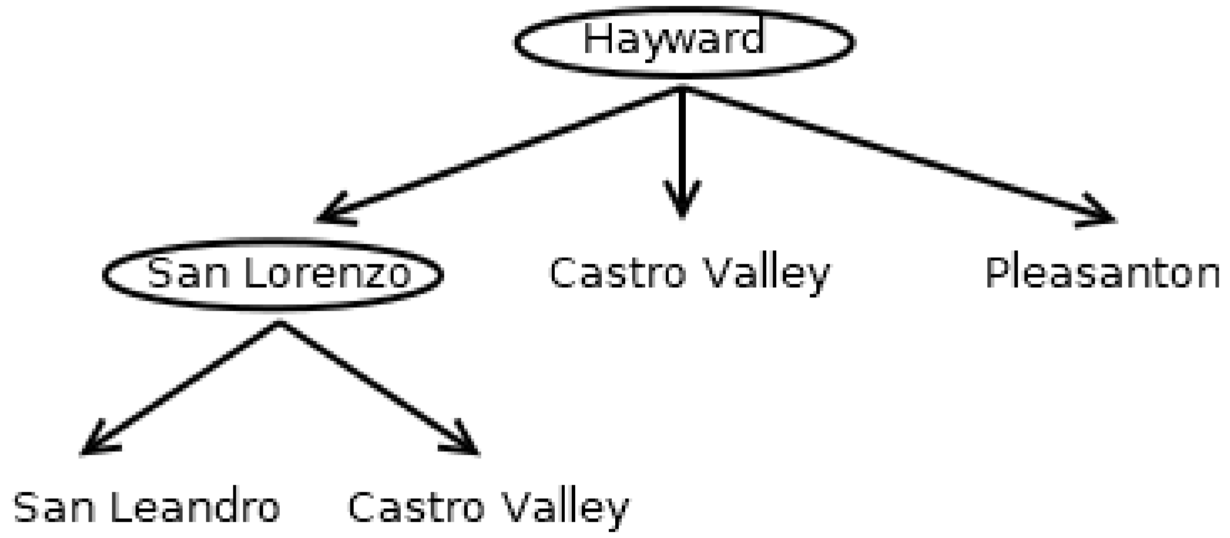
Breadth-first search

- Expand nodes in increasing order of depth within tree
- Fringe is stored in a queue
- Ex: Start node is in Hayward. Expanding it creates a queue of nodes representing San Lorenzo, Castro Valley, Pleasanton (leaving out Union City, unincorporated areas to keep things simpler)



Breadth-first search

- Ex: (continued) expanding San Leandro removes node with San Leandro and adds nodes with Oakland and Castro Valley to the end of the queue
  - Why does Castro Valley show up again? What is the order of the queue?



Breadth-first search

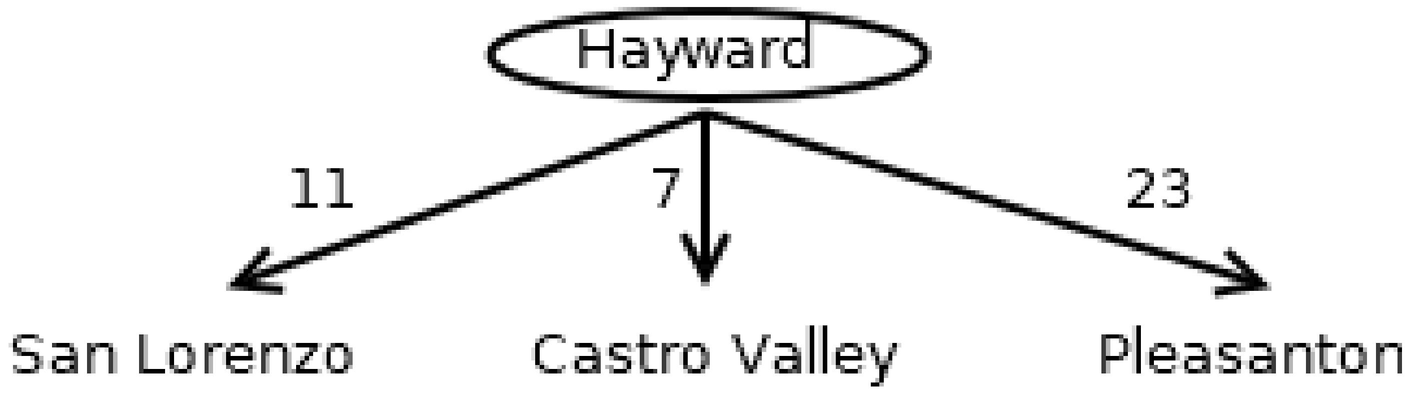
- Complete?
  - yes, if b = branching factor is finite
- Time complexity?
  - $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$ → exponential in search depth
- Space?
  - $O(b^{d+1})$ ~ almost every node is in memory
  - This is a real problem
- Optimal?
  - Under what conditions?

Uniform-cost search

- Similar to breadth-first, but use priority queue ordered by cost of path = sum of costs of actions to reach node
  - If all actions cost 1, same as breadth-first
- Complete
  - If costs are positive
    - Negative cost example: relative distance from starting point
- Time, space complexity: # nodes with total cost < optimal cost
- Optimal: yes (again, if costs > 0)

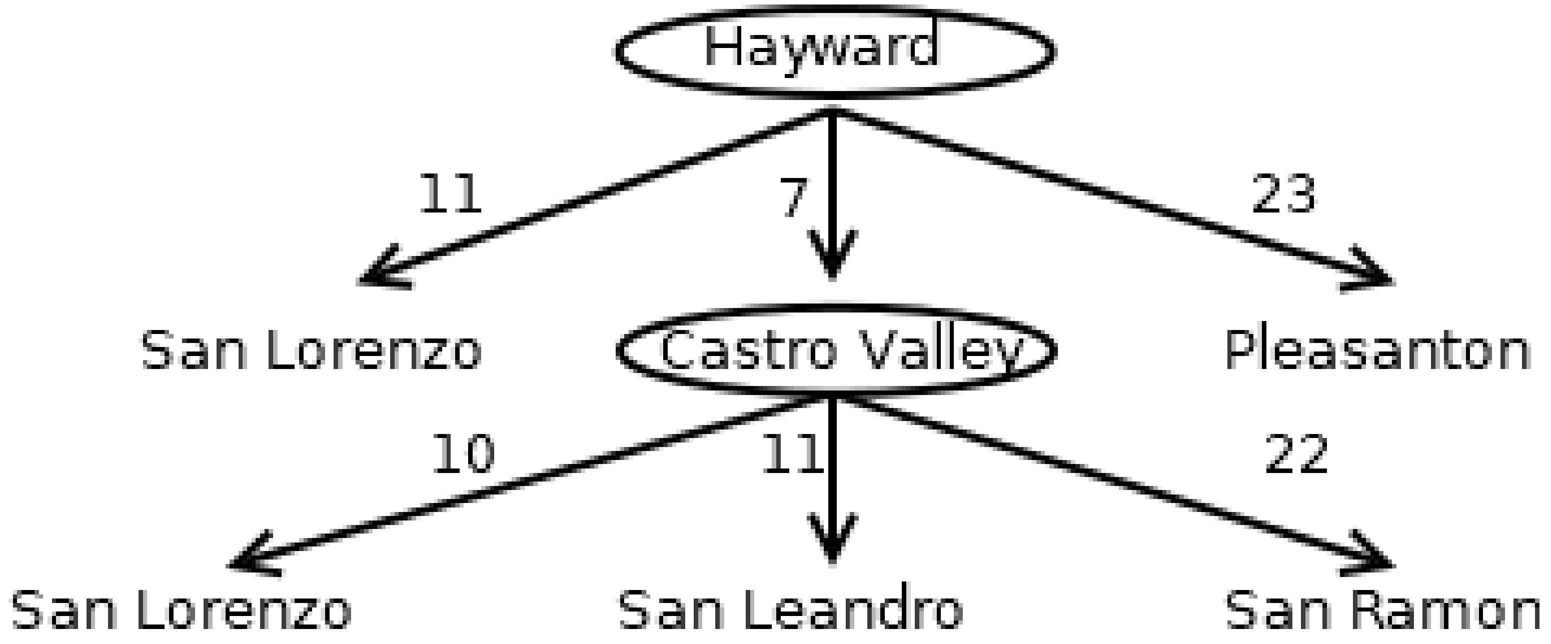
Uniform-cost search example

- Edges are weighted with cost (here, travel time in minutes taken from Google Maps on a weekday around noon)



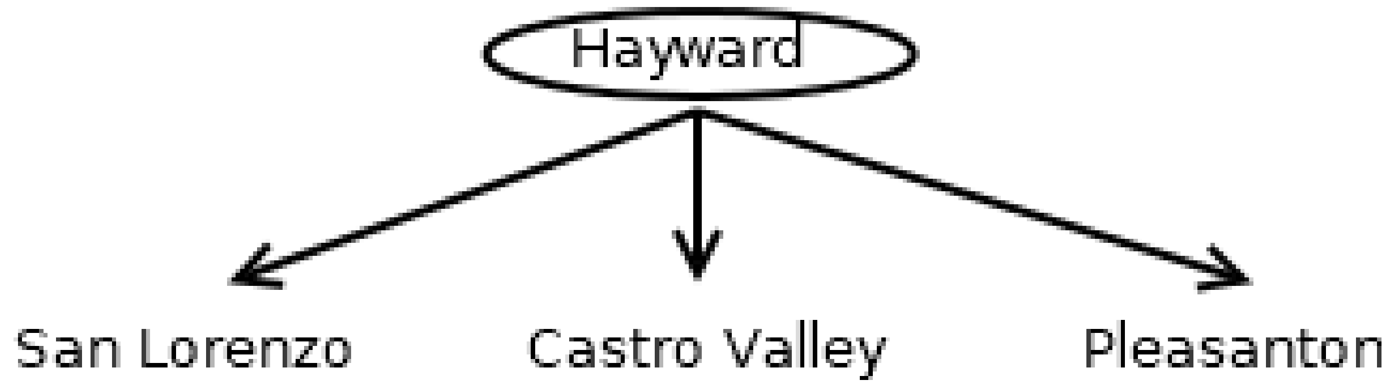
Uniform-cost search example

- Expand Castro Valley node next since travel time is lowest
  - Which node should be first in the priority queue now?



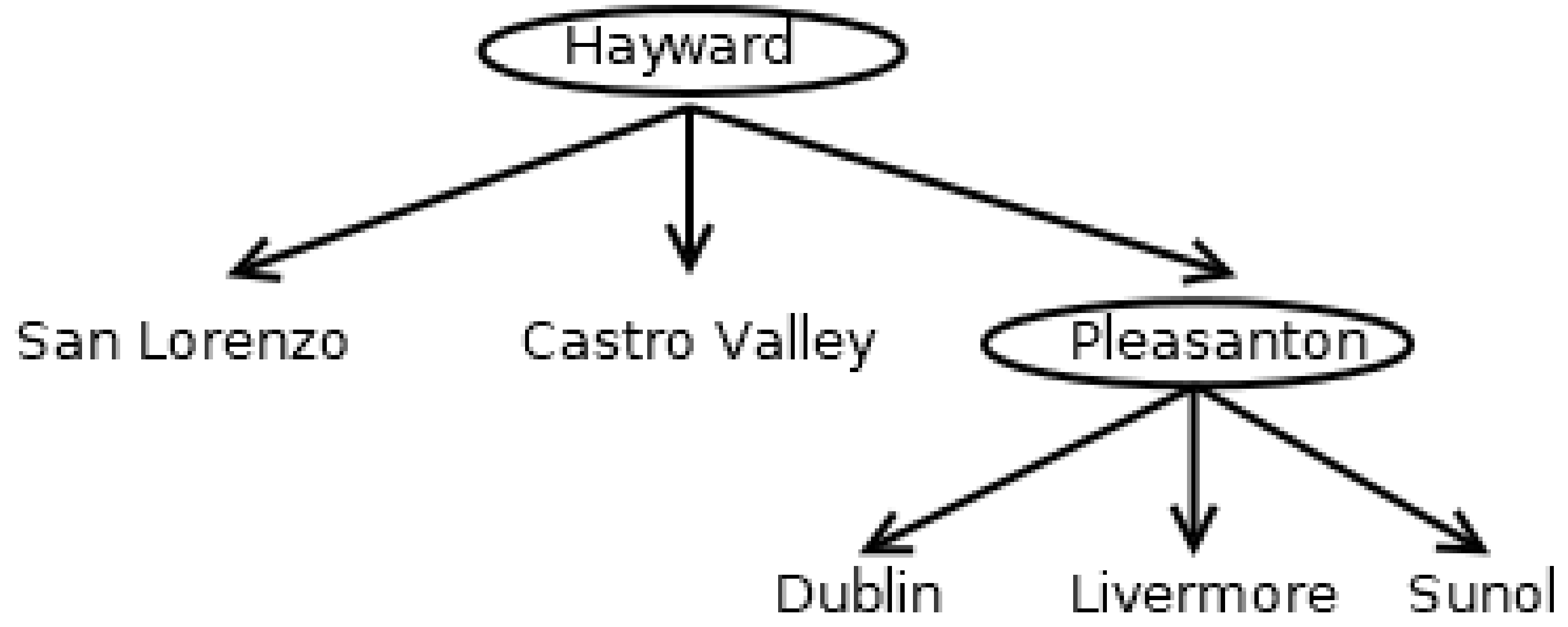
Depth-first search

- Expand nodes, deepest first
- Fringe is stored in a stack (LIFO)
- Ex: Start node is in Hayward. Expanding it creates a stack of nodes with, assuming cities are found in same order, a Pleasanton node on top



Depth-first search

- What would be the 3rd node to expand?



Depth-first search

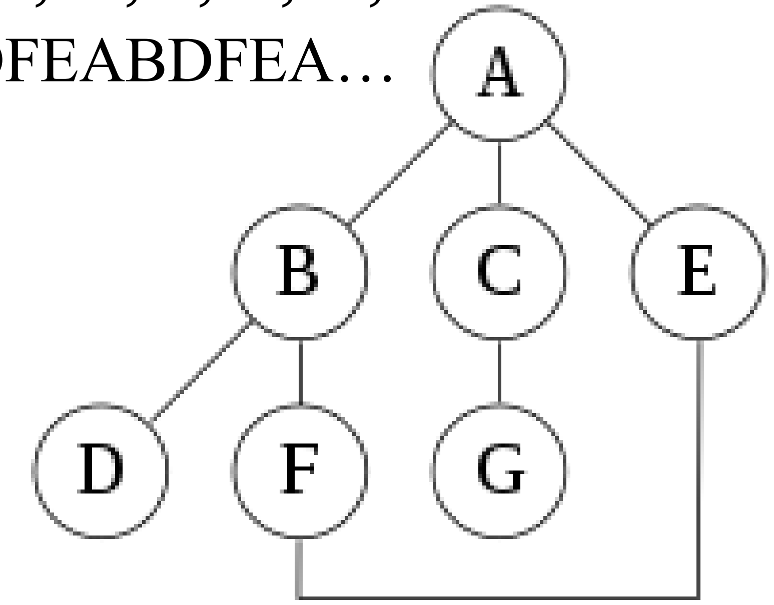
- Complete
  - only if space is finite (should check for duplicate states along any path)
- Time
  - O(b^m), m = max depth of search tree
  - When is DFS fast?
- Space
  - O(bm) – much better than BFS
- Optimal?
  - No

Iterative deepening search

- Motivation: depth-first, but handle infinite state space
    - Naïve approach: depth limited search – do depth-first, but not deeper than n levels (not complete, not optimal)
- General idea: each iteration, do depth-first search one level deeper
    - 1st iteration: check root
    - 2nd iteration: check root – if not goal state, expand and check children
    - 3rd iteration: check up to root, children, grandchildren
    - …
- Becomes similar to breadth-first search, but without having to store as much

Iterative deepening example

- From https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search#Example_2
- Depth-first order (if nodes are remembered): A, B, D, F, E, C, G
  - If nodes are not remembered, gets stuck in loop ABDFEABDFEA…
- Iterative deepening:
  - Depth 0: A
  - Depth 1: A, B, C, E
  - Depth 2: A, B, D, F, C, G, E, F
  - Depth 3: A, B, D, F, E, C, G, E, F, B



Iterative deepening search

- Complete? Yes
- Time? $O(b^d)$
  - Despite repetition, can save over breadth-first on time as well by not expanding all of the nodes at depth d
- Space? $O(bd)$
- Optimal? Yes if edges are all cost = 1