Search : Brute force and Heuristic

- In solving problems, we sometimes have to search through many possible ways of doing something.

  Getting from Brussels to Oostende

  Winning a chess game.

- Many problems can be formalized in a general way as search problems.

See AI a modern approach

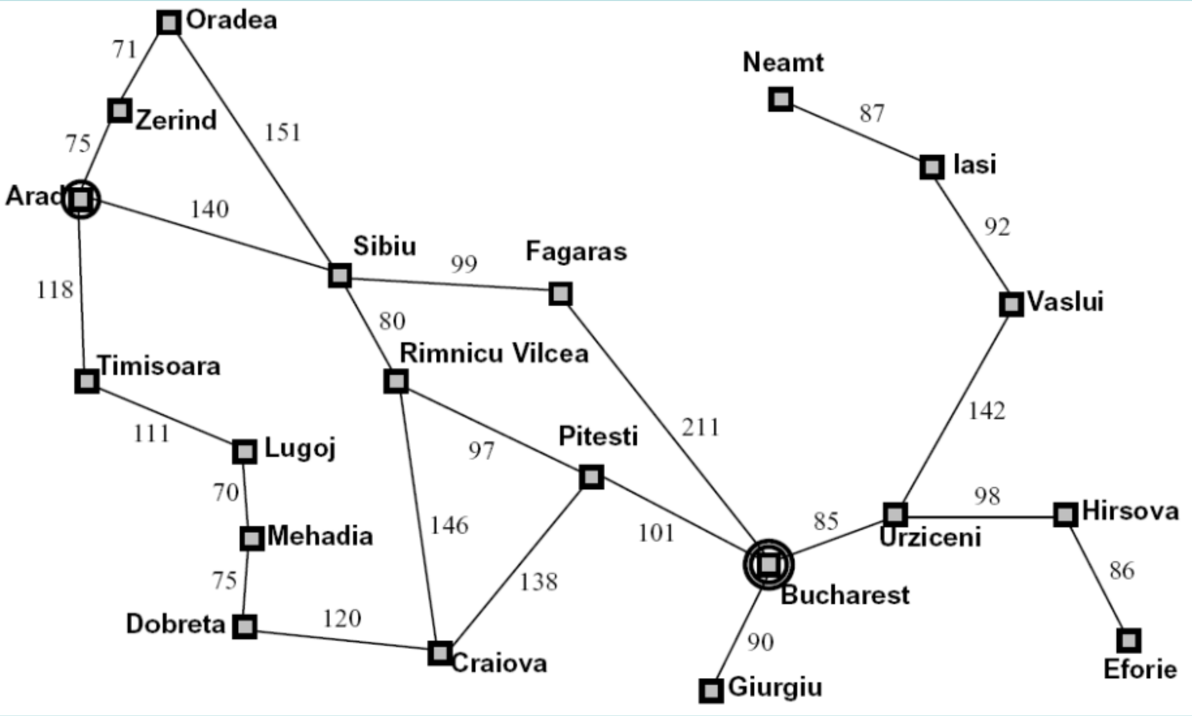
The essence of AI, A. Cawsely (chapter 4)

Search and Problem Solving

- Search problems described in terms of:
  - An initial state. (e.g., initial chessboard, current positions of objects in world, current location)
  - A target state.(e.g., winning chess position, target location)
  - Some possible actions, that get you from one state to another. (e.g. chess move, robot action, simple change in location).
- Search techniques systematically consider all possible action sequences to find a *path* from the initial to target state.

Example: Romania

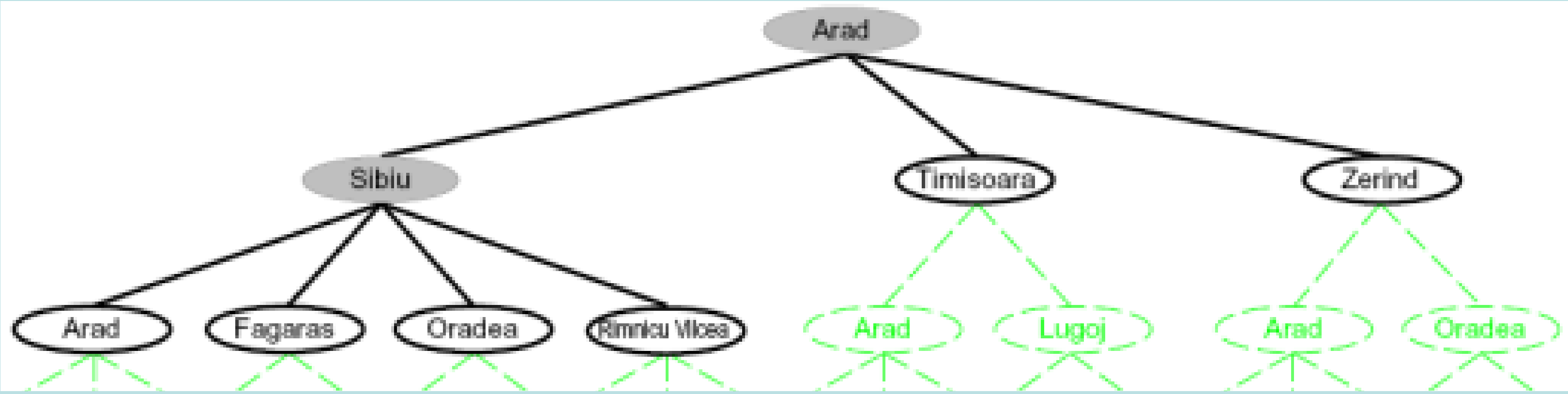
- State Space Graphs



- State space:
  – Cities
- Successor function:
  – Go to adj city with cost = dist
- Start state:
  – Arad
- Goal test:
  – Is state == Bucharest?
- Solution
  - path from Arad to Bucharest

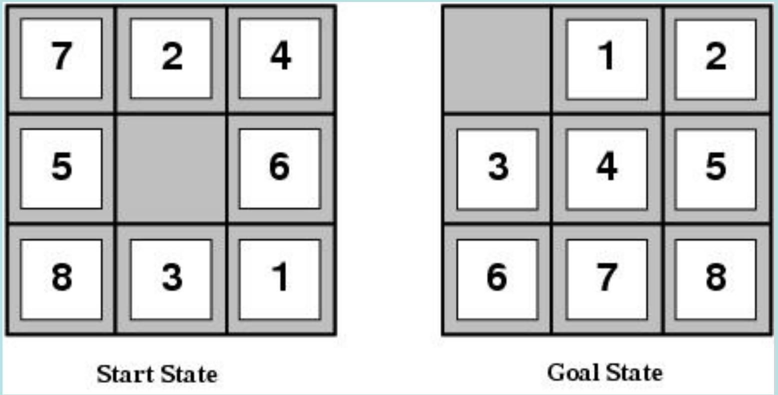
Example: Romania, cont.

- Search Tree



- Search:
  - Expand out possible plans
  - Maintain a frontier of unexpanded plans
  - Try to expand as few tree nodes as possible

Another example: 8 puzzle



- States: Integer location of each tile
- Initial state: Any state can be initial
- Actions: {*Left*, *Right*, *Up*, *Down*}
- Goal test: Check whether goal configuration is reached
- Path cost: Number of actions to reach goal

8-puzzle: as search tree

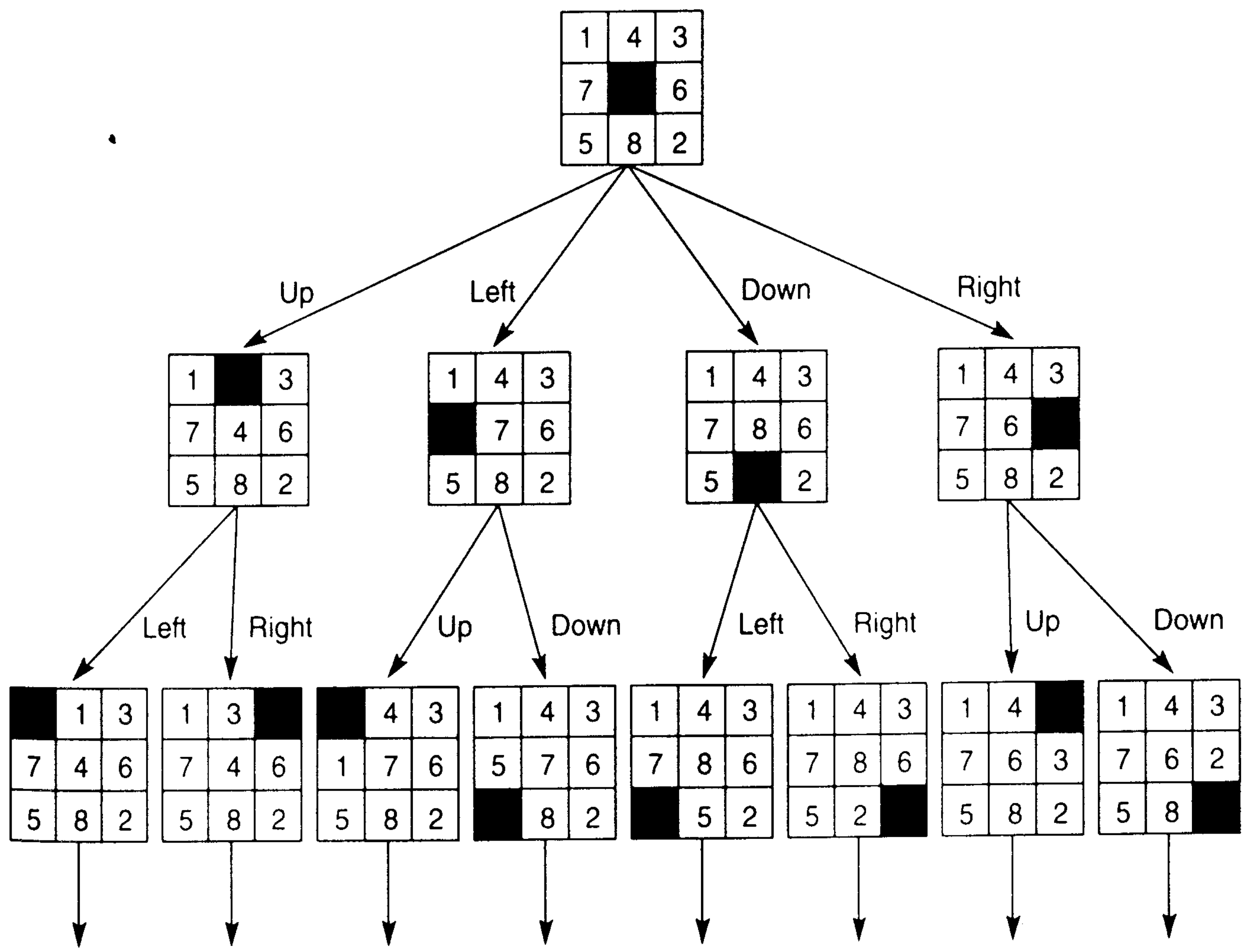
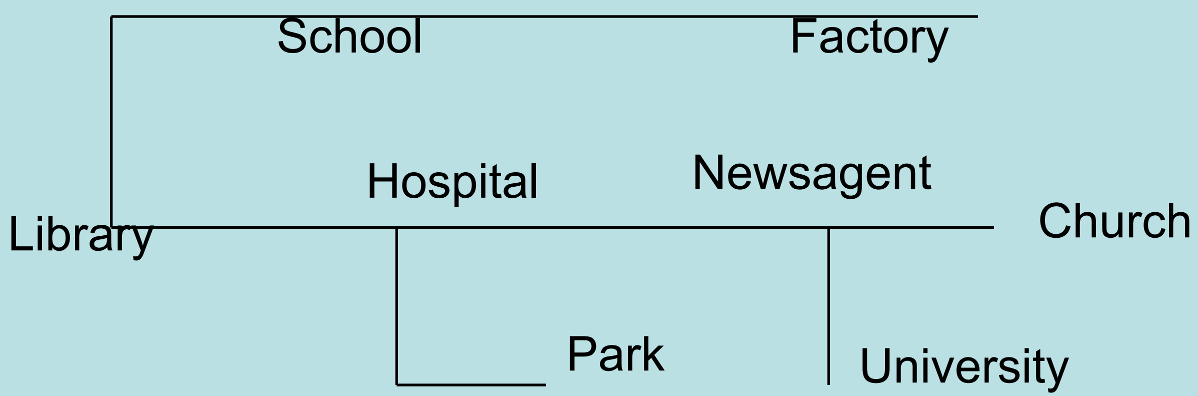


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

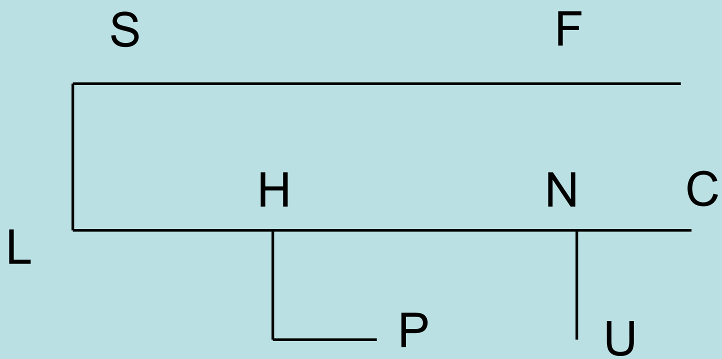
Simpler example, no loops

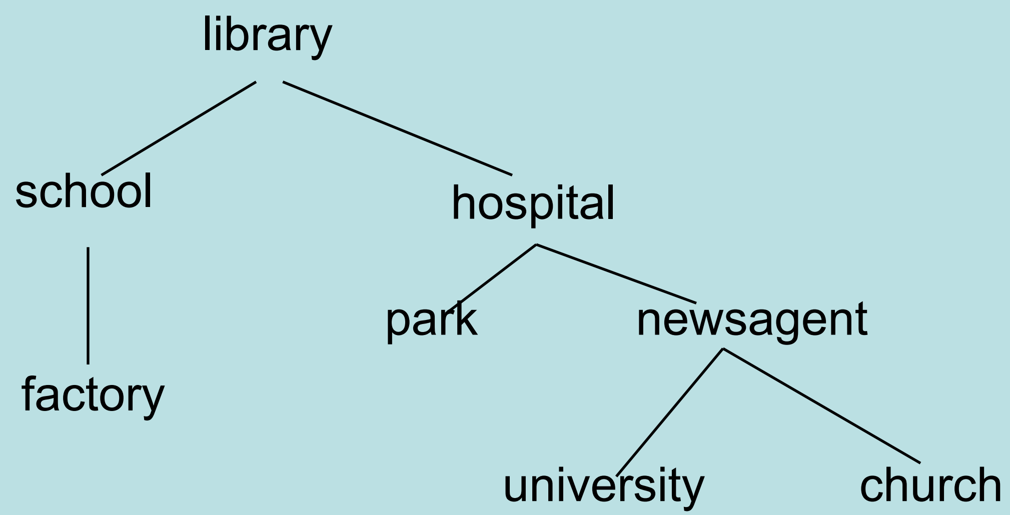
- How do we systematically and exhaustively search possible routes, in order to find, say, route from library (initial state) to university (goal state)?



Search Space

- The set of all possible states reachable from the initial state defines the *search space*.
- We can represent the search space as a tree.



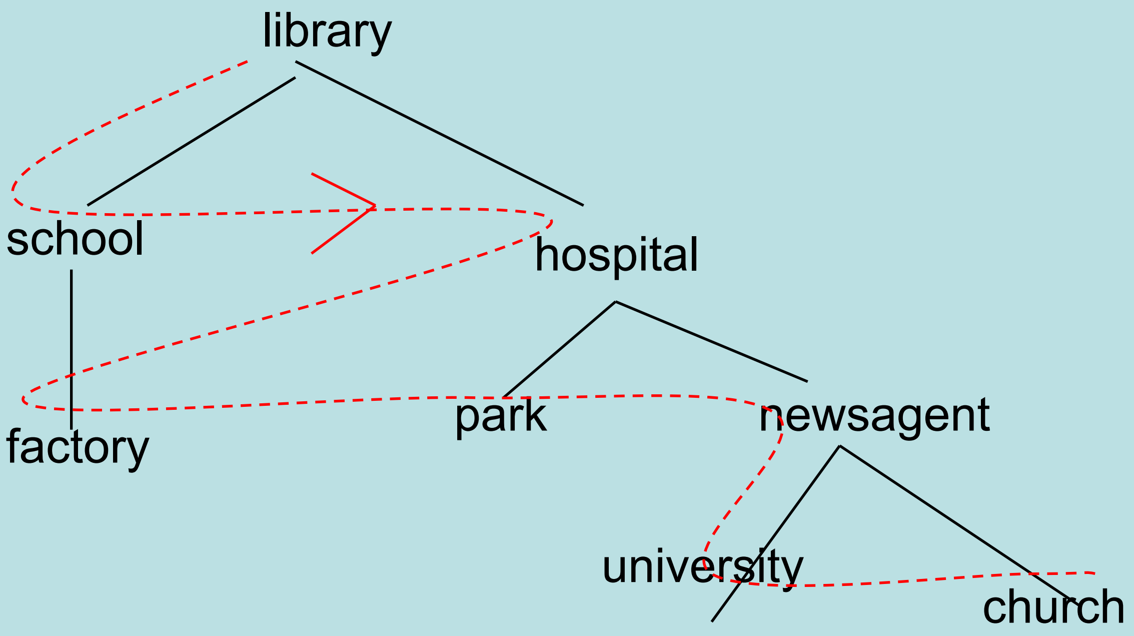


Simple Search Techniques

- How do we search this tree to find a possible route from library to University?
- May use simple systematic search techniques, which try every possibility in systematic way.
- Referred to as **brute force** or **blind** techniques
- **Breadth first search** - Try shortest paths (least hops) first.
- **Depth first search** - Follow a path as far as it goes, and when reach dead end, backup and try last encountered alternative.

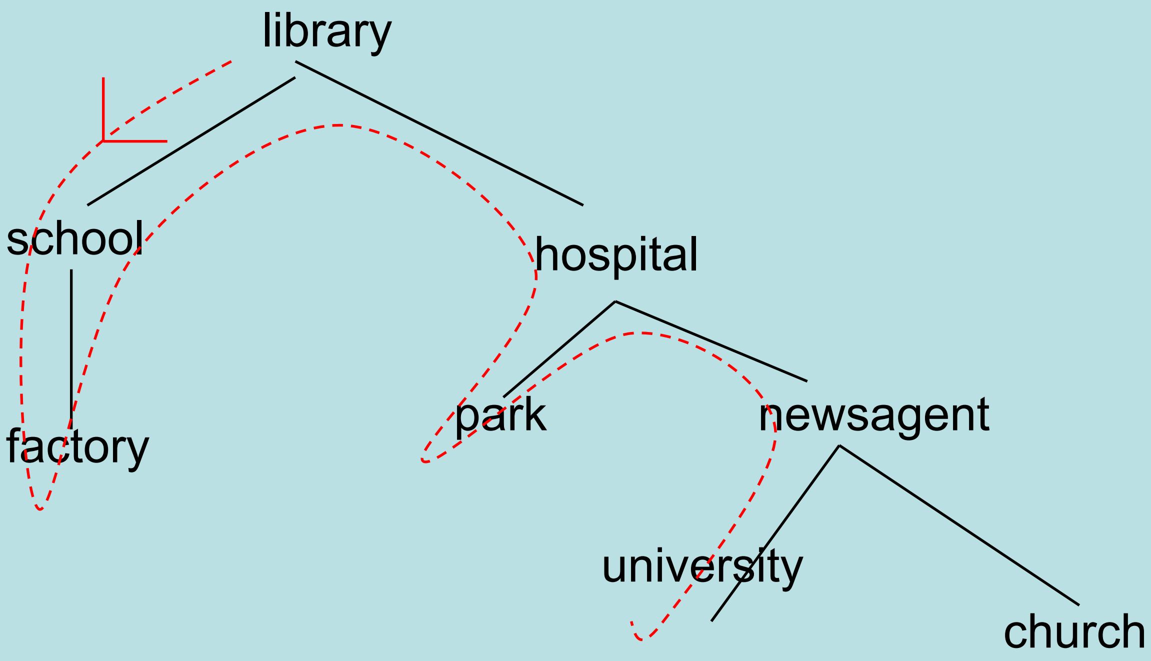
Breadth first search

Explore *nodes* in tree order: library, school, hospital, factory, park, newsagent, university, church. (conventionally explore left to right at each level)



Depth first search

- Nodes explored in order: library, school, factory, hospital, park, newsagent, university.



Algorithms for breadth first and depth first search.

- Very easy to implement algorithms to do these kinds of search.
- Both algorithms keep track of the list of nodes found.
  - E.g., [library, school, hospital, ….]
  - List is sometimes referred to as an **agenda**. But implemented using
    stack for **depth first**,
    queue for **breadth first**.

Algorithm for breadth first:

- Start with **queue** = [initial-state] and found=FALSE.
- While queue not empty and not found do:
  - Remove the first node N from queue.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, and put them at the end of the queue.

Algorithm for depth first:

- Start with **stack** = [initial-state] and found=FALSE.
- While stack not empty and not found do:
  - Remove the first node N from stack.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, and put them on the end of the stack.

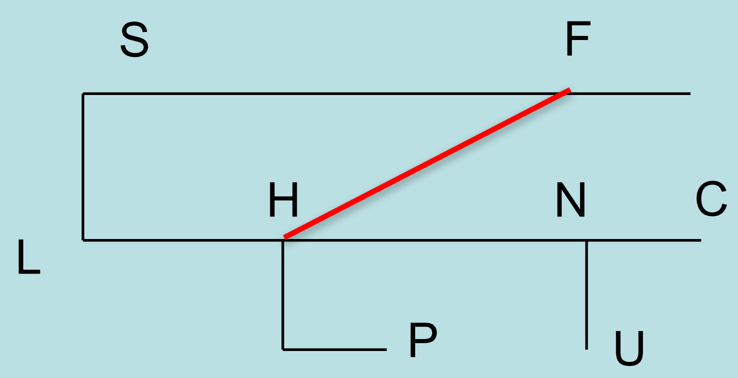
Note: Detailed work through of algorithms and discussion of trees/graphs in textbook.

Choice between algorithms

- When is one technique more appropriate than the other?
  - Shortest path? BF
  - Is memory a problem? DF
  - Do you want to find the solution quickly? Depends on the structure of the search tree
- To avoid long paths in DF search; define depth limit
- To find shortest path quickly, change DF search to *iterative deepening*

Extensions to basic algorithm

- Loops: What if there are loops (i.e., we are searching a *graph*)? How do you avoid (virtually) driving round and round in circles?



- Algorithm needs to keep track of which nodes have already been explored, and avoids redoing these nodes.

Extensions to basic algorithm

- Variation of DF search
  - Start with stack = [initial-state] and found=FALSE.
  - While stack not empty and not found do:
    - Remove the first node N from stack.
    - **If N is not in visited then**:
      - **Add N to visited**
      - If N is a goal state, then found = TRUE.
      - Find all the successor nodes of N, and put them at the end of the stack.

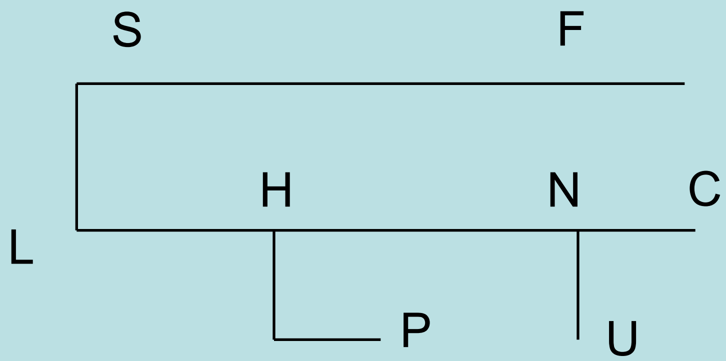
Extensions to basic algorithm

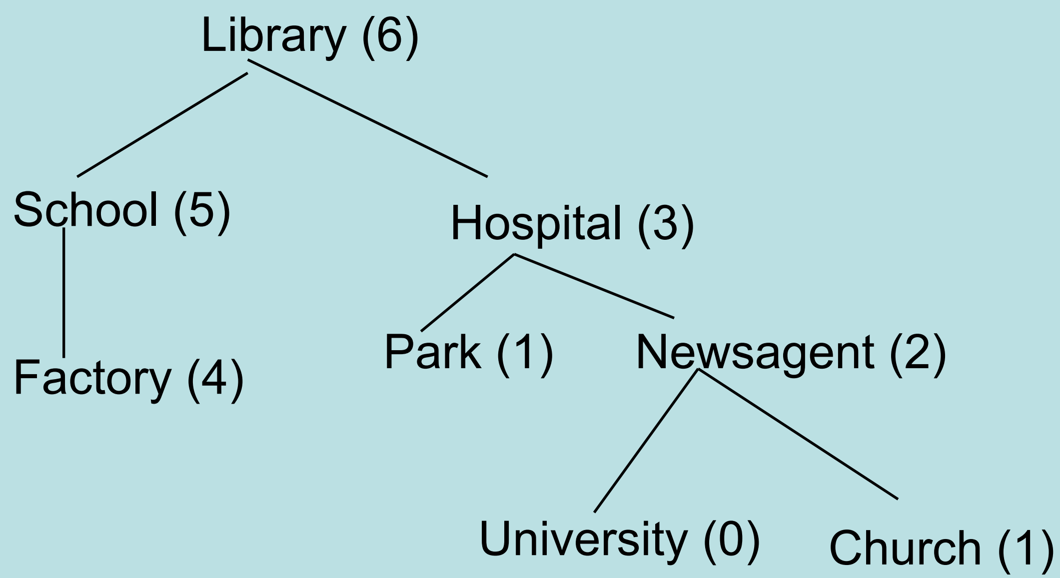
- **Other** variation of DF search
  - Start with stack = [initial-state] and found=FALSE.
  - While stack not empty and not found do:
    - Remove the first node N from stack.
      - **Add N to visited**
      - If N is a goal state, then found = TRUE.
      - Find all the successor nodes of N, and put the non-visited nodes at the end of the stack.

Heuristic search algorithms.

- Depth first and breadth first search turn out to be too inefficient for really complex problems.
- Instead we turn to “heuristic search” methods, which don’t search the whole search space, but focus on promising areas.
- To identify promising areas we need an evaluation function
- The evaluation function scores a node in the search tree on how close it is to the goal/target state.

Hill Climbing





(number) indicates the “as the crow flies”-distance to the goal

Hill Climbing

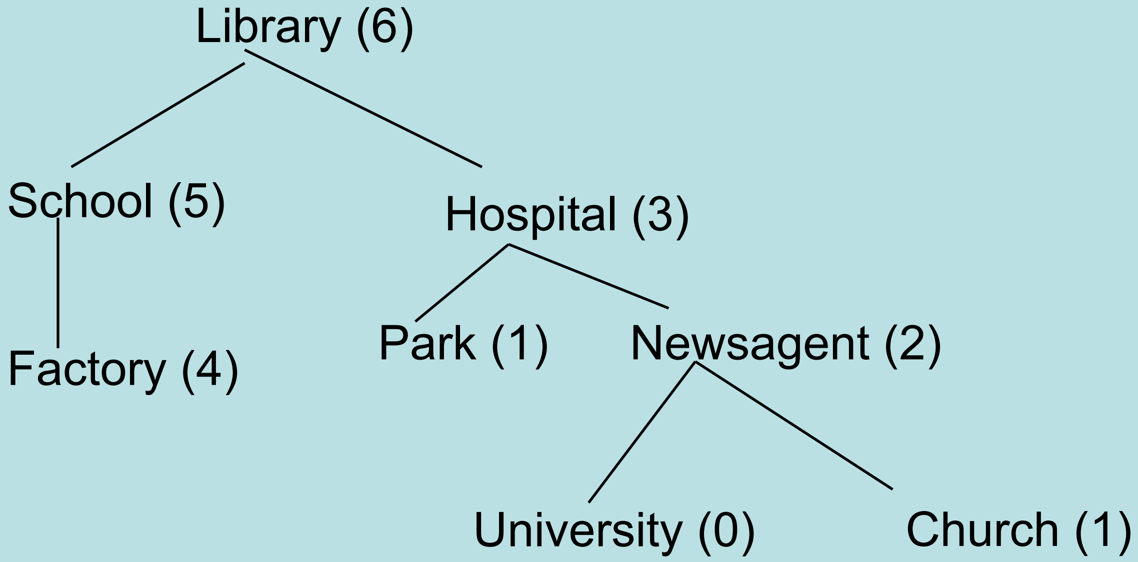
- Hill climbing: always choose successor node with highest score.
  - Start with current-state=initial-state
  - Until current-state=goal-state or there is no change in current state do:`
    - Get the successors of current state
    - Evaluate the successors and assign them a score
    - If one of the successors is better than current-state, then set the new-current state to be the successor with the best score
- Avoids loop
- Algorithm may halt without success in local optimum

Best first search algorithm

- Best first search algorithm almost same as depth/breadth. But we use a **priority queue**, where nodes with high scores are taken off the queue first.
- Hence still exhaustive search and performance depends on the quality of the evaluation function
- Start with agenda=(initial-state)
- While agenda not empty and not found do:
  - Remove the BEST node N from agenda.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, assign them a score, and put them on the **agenda organised as a priority queue.**

Best first search

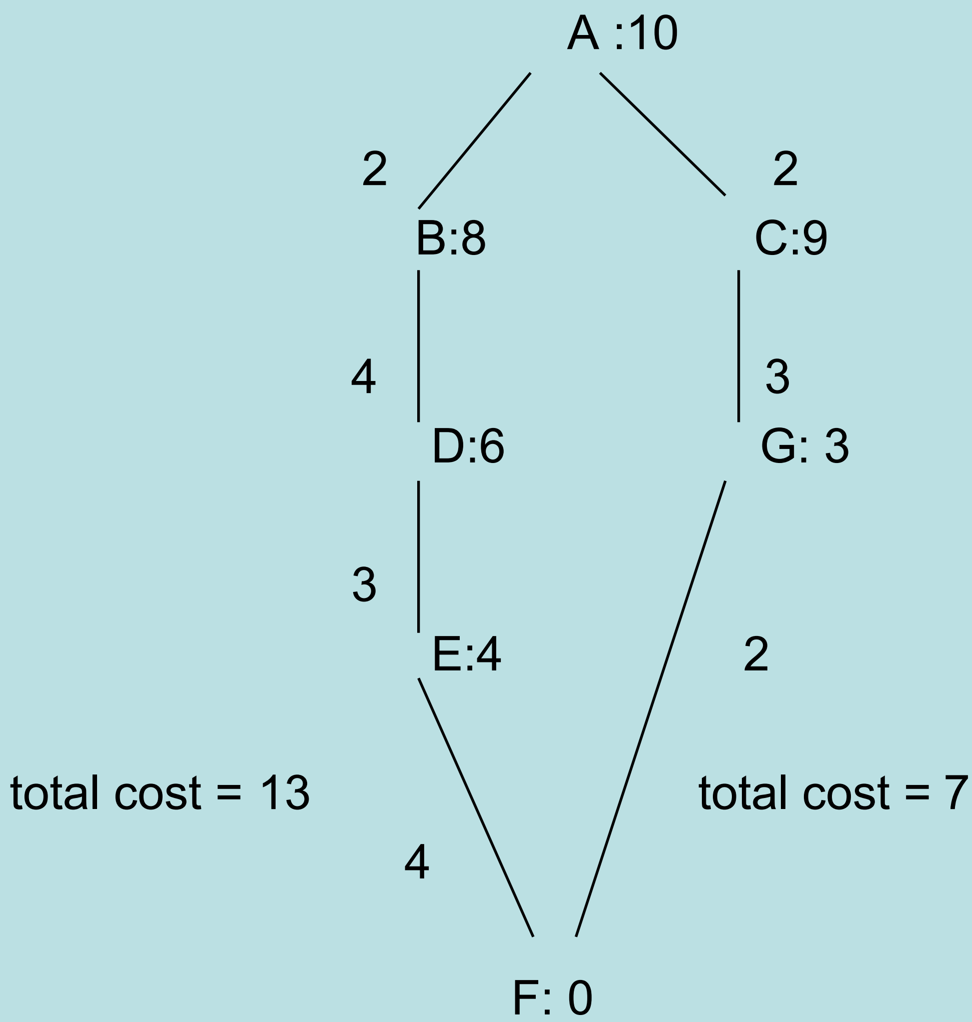
- Order nodes searched: Library, hospital, park, newsagent, university.



A* algorithm

- Extension of best-first search (takes total path length into account)
- A*: Score based on predicted total path "cost", so (weighed) sum of
  - actual cost/distance from initial to current node,
  - predicted cost/distance to target node.
- In Breadth First search (if the cost of traversing a link is the same), the solution with the lowest cost will be found first. However it may take time. In Best First search a solution can be found quickly, yet it may not be a very good one. The A* algorithm finds a cheap solution quickly.

A* algorithm, an example



best first : A,B,D,E,F

A* : A,B,C,G,F’

Summary

- General search methods can be used to solve complex problems.
- Problems are formulated in terms of initial and target state, and the primitive actions that take you from one state to next.
- May need to use *heuristic* search for complex problems, as search space can be too large.