Search:

or why the thing you want is always in the last place you look

General Introduction

- Search Concepts

  State - allowable locations or situations

  Operators - procedures that generate successor states

  Goal state - needs goal test criteria

  Path - sequence of states

  Cost - of paths and states

  Total cost = search cost + path cost (tradeoff)

- General Algorithm

  Initialise data structure (*agenda)*

  Loop if no states left to search -> return fail

  select node for expansion

  if node = goal -> return success

  apply operator to node

  and add successors to *agenda*.

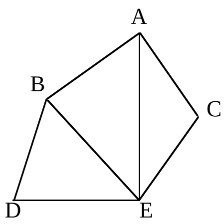
Types of Search

- Blind Search (Uninformed)
  - Use no domain knowledge
    - Breadth First
    - Depth First
    - Depth First Limited
    - Iterative Deepening
- Heuristic Search (Informed)
  - Employ domain knowledge
    - Hill Climbing
    - Best First
      - uses h(n): cost to goal estimate
    - Beam Search
      - Uses h(n) to prioritise search
    - A* Search
      - uses f(n) = h(n) + g(n): full cost estimate

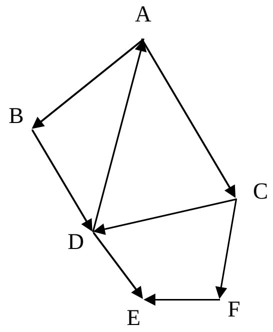
Criteria for Evaluation

- Completeness
  - guarantees to find a solution?
- Time complexity
  - how long to find solution?
- Space complexity
  - how much memory is needed?
- Optimality
  - best solution found?

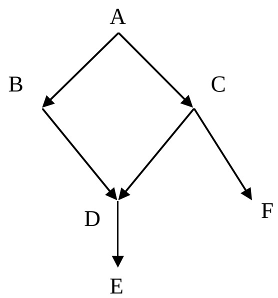
Graphs and Trees



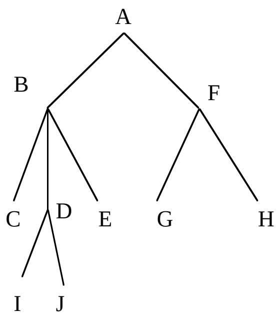
A (labelled) graph



A directed graph



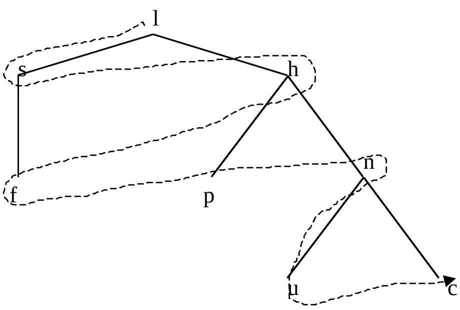
A directed acyclic graph



A rooted tree

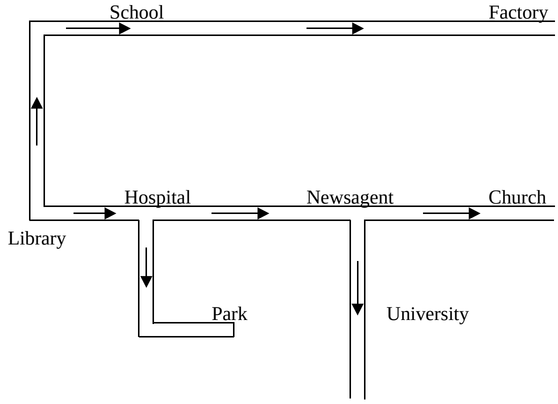
Breadth First Search

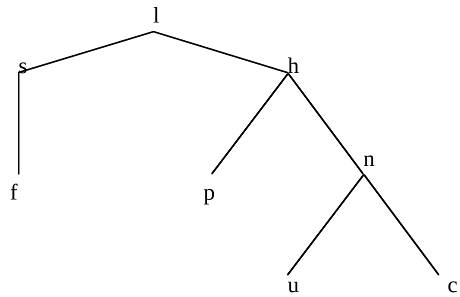
Nodes are explored in the order they are created
=> agenda is a *queue*



1. Start with *queue* = [initial state] and *found* = FALSE
2. While *queue* not empty and not *found* do:
   (a) remove first node N from *queue*
   (b) if N is goal state then *found* = TRUE
   (c) find all successor nodes of N and add them to the *queue*

Finding routes on a map



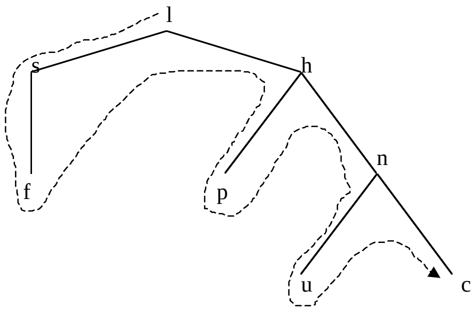


Time and Memory

Depth	Nodes	Time	Memory
0	1	1 ms	100 bytes
2	111	.1 s	11 Kb
4	11,111	11 s	1 Mb
6	10^6	18 min	111 Mb
8	10^8	31 hrs	11 Gb
10	10^10	128 days	1 Tb
12	10^12	35 yrs	111 Tb
14	10^14	3500 yrs	11,111 Tb

Depth First Search

Nodes are explored as they are created
=> agenda is a *stack*



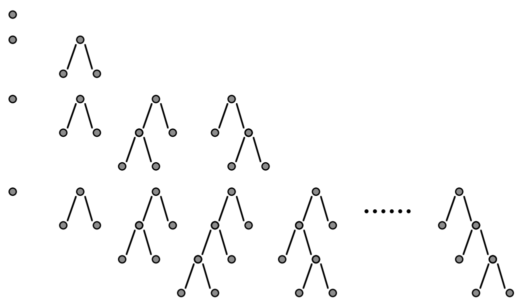
1. Start with *queue* = [initial state] and *found* = FALSE
2. While *queue* not empty and not *found* do:
   (a) remove the first node N from *agenda*.
   (b) if N not in *visited* then:
      i. add N to *visited*
      ii. if N is a goal then *found* = TRUE
      iii. put N's successors on the front of the *stack*

Depth First Search 2

- Evaluation
  - modest memory requirements
  - not optimal (but may sometimes be better than breadth first if more than one solution).
  - not complete - infinite loops
- Depth First Limited
  - "backs-up" if nodes further from initial state than a specified distance are reached.
  - complete but not optimal (as long as depth is large enough)

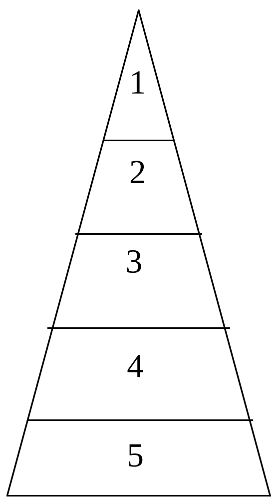
Iterative Deepening

- Method
  - sidesteps problems with depth choice by trying all depths.
- Evalution
  - optimal and complete (combines best of breadth and depth first).
  - preferred method when search space large and depth of solution is not known



Iterative Deepening 2

How much extra time is needed to search to depth *d* with ID compared to DF search directly to depth *d*?



Level	Total DF	Total ID
20	20	20
400	420	440
8000	8420	8860
160,000	168,420	177,280
3200000	3368420	3545700

Iterative Deepening 3

Ratio of time is: $\frac{b + 1}{b - 1}$

So for different branching factors we get:

b	Ratio
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Bidirectional Search

- Meet in the middle
  - search forwards from *start* and backwards from *goal*
  - helps time complexity
- Issues to be addressed
  - what does "search backwards mean?
  - predecessors difficult to calculate?
  - what if many goal states?
  - efficient checks between halves!
  - what is best search for each half?

Comparing Search Strategies

Criterion	**Breadth First**	**Uniform Cost**	**Depth First**	**Depth Limited**	**Iterative Deep'**	**Bidire ctional**
Time	b^d	b^d	b^m	b^l	b^d	b^d/2
Space	b^d	b^d	bm	bl	bd	b^d/2
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if l >= d	Yes	Yes