Graphsearch

Note Title

Review: Irrevocable vs. Tentative Strategies

Implicit vs. Explicit Graphs

Implicit vs. Explicit Enumeration

Weighted Graphs vs. Uniform Cost

Irrevocable

Flail Blindly

Hill Climbing

Simulated Annealing

Tabu search

Genetic Algorithms

Swarm Optimization

Tentative

Backtrack

Improvements to Backtrack

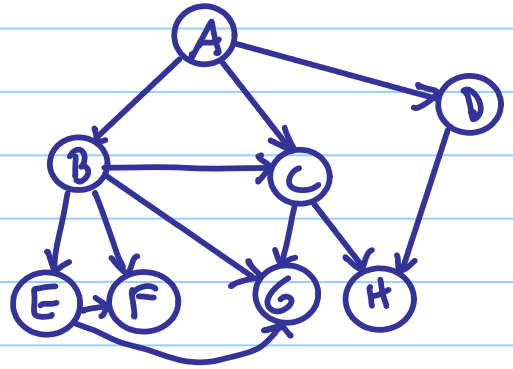
- heuristics for ordering rule selection
- symmetry
- forward/backward search

Graphsearch

Graph search:

- Maintain all nodes generated, not just current path
- each node will have <u>one</u> parent as its predecessor
  (If nodes recur w/ different parents, choose <u>one</u> that provides best path from start to current node)

Example



(F) has two parents, (B) + (E)

clearly (B) is better choice, so forget (E)→(F) edge -

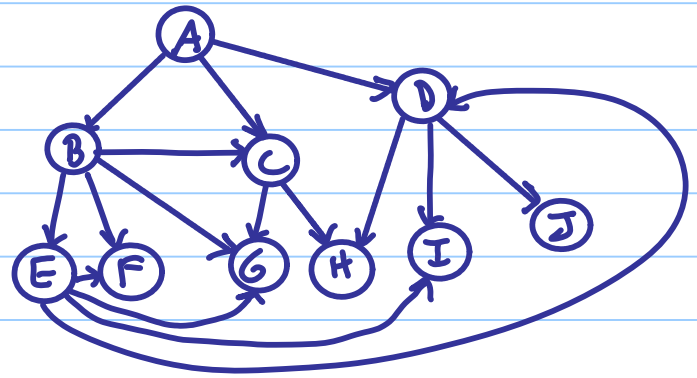
So result will be a "search tree" where each node has exactly one path back to the root

This is a subtree of the problem graph

- How to decide which node to examine next is the philosophy that provides different versions of graph search algorithms: Breadth-First Search, Depth-First Search and Best-First Search

---

Example:

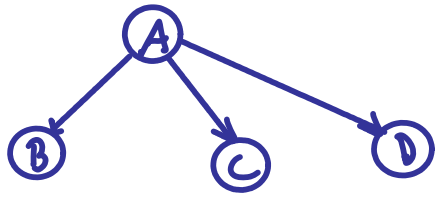


Breadth-First

Examine all nodes at level K before examining any nodes at level K+1, K+2 etc.

first, (A) start node

next examine a node, generate its offspring ("expanding the node")



Level 0

Level 1

next choose a node at level 1 and examine it

B has children C, E, F, G

C has been generated previously

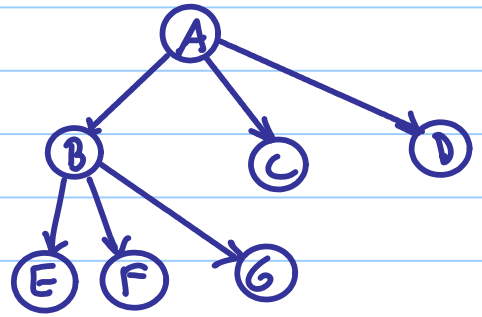
E, F, G new nodes

Is B preferable as a parent to C's current parent?

For Breadth-first search with uniform cost (i.e, all edges have same weight – we are only trying to minimize total # of moves) answer is NO.

But, if edges have potentially different weights, answer could be YES.

May need to reassign a "parent pointer" for any previously generated node.



Level 0

Level 1

Level 2

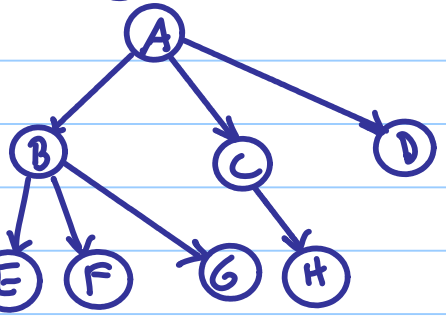
A key difference between Breadth-first and Depth-first search:

BFS – expand C or D next

DFS – expand E, F or G next

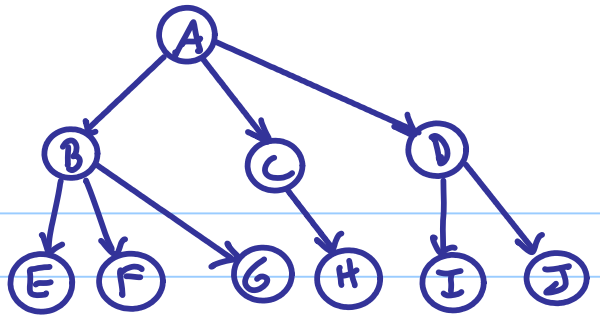
next, expand C

has children G + H. G has already been generated. No apparent advantage to reassigning G's parent



next expand D

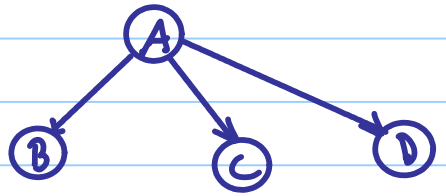
children H, I, J H already generated. No need to change H's parent



Depth-First

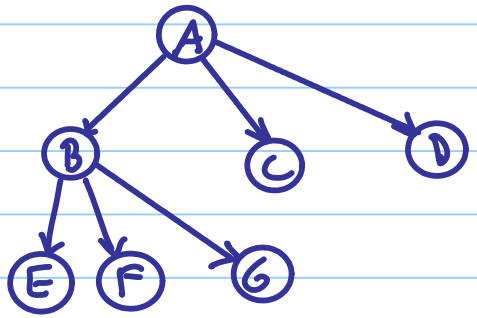
Start with (A)

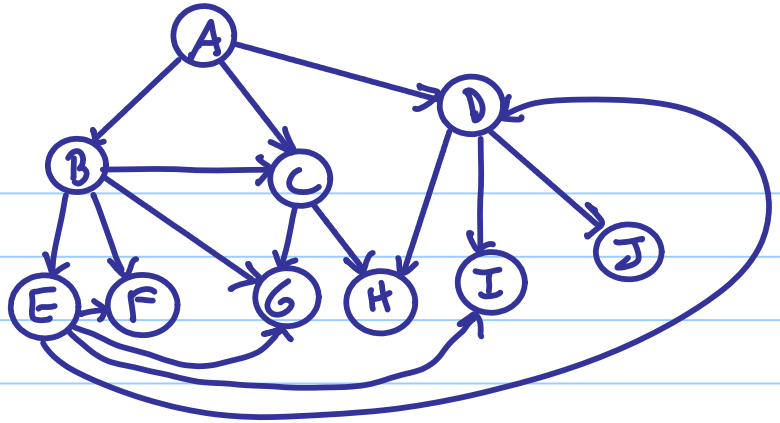
next, expand (A) — children (B), (C), (D)



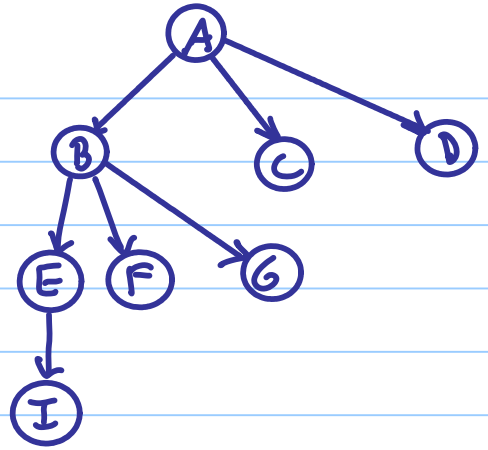
Choose to expand (B): children (C) (E) (F) (G)

↑ generated previously, keep current parent





expand (E) — children (D), (F), (G), (I)



generated previously — no need to redirect parents

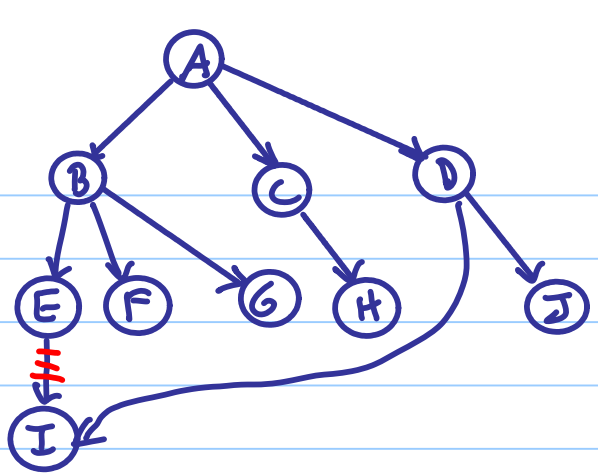
no children for (I), (F), (G)

(C) has children (G) + (H) — already seen and expanded — no need to redirect

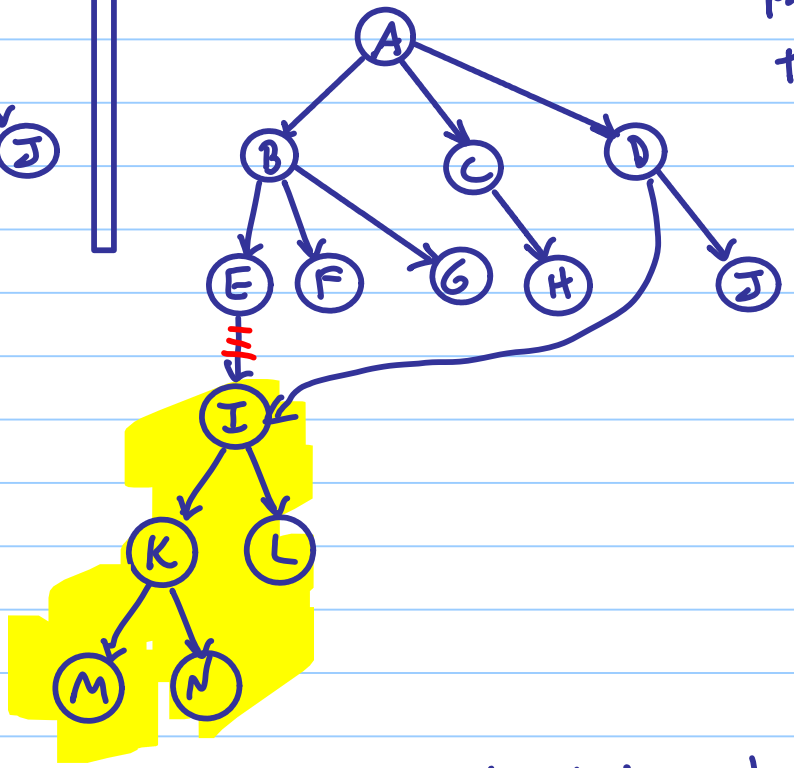
(H) has no children

(D) has (I) + (J)

↳ previously seen — (D) is selected as (I)'s new parent (shorter path)



Suppose in previous example that (I) had other offspring — by redirecting (I)'s parent from (E) to (D) - the future generations are also redirected (and the depths of those nodes have all changed)



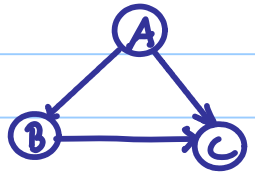
Graphsearch will be a generalized algorithm that describes both Breadth-First and Depth-First Search - the "depth" of each node is an important attribute and will need to be recalculated after reassigning parent links

When generating a node's offspring ("expanding" the node):

- some of the offspring may not have been seen before
  [automatically retained as offspring of this node]

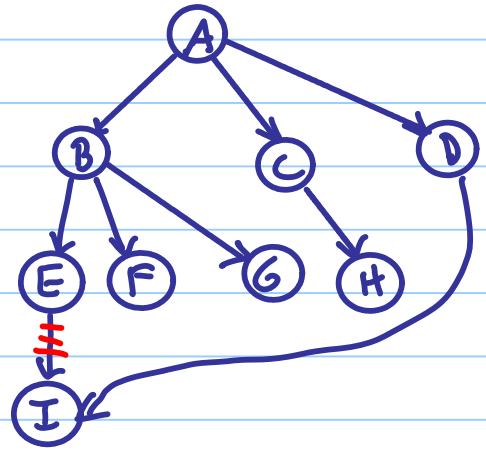
- some offspring may have been seen before, but not expanded

  like (C) in these examples —

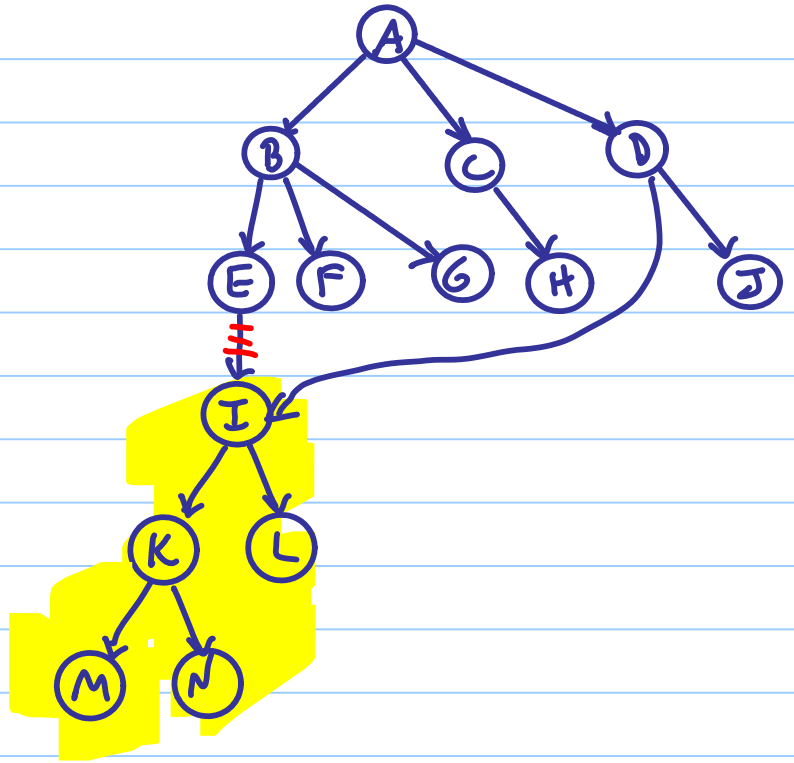


  choose to re-adjust who the parent of that node is, based on path cost from root to that node

  like (I) in third example —
  we find shorter path to (I) the second time



- some offspring may have been seen before and were expanded
  - like (I) in last example



adjust depth of all descendants of the child node.