Graphsearch, continued

Note Title

OPEN – a list of all nodes that have been generated, but not expanded

Initially, OPEN = { startNode }

CLOSED – a list of all nodes that have been (generated and) expanded

Initially CLOSED = { }

DEPTH-FIRST SEARCH

New nodes are added to front of OPEN. (OPEN is a stack)

BREADTH-FIRST SEARCH

New nodes are added to rear of OPEN. (OPEN is a queue)

Graphsearch Will build a search tree of the program graph, with root of tree at start node

$OPEN \leftarrow \{ start \}$ → OPEN is a list of all nodes that have been seen but not expanded (leaf nodes)

$CLOSED \leftarrow \{ \}$ → CLOSED is a list of nodes that have been expanded (interior nodes)

while $OPEN \neq \{ \}$ will iterate until entire graph has been explored OR a goal node is reached

- $s \leftarrow$ first($OPEN$)
- $OPEN \leftarrow$ rest($OPEN$)
- $CLOSED \leftarrow CLOSED + \{ s \}$

take first node from OPEN and expand it (generate all successors and add it to CLOSED

- **if** goal(s), **exit**
- **for each** $r \in$ ApplicableRules(s)
  - $s' \leftarrow$ ApplyRule(r,s)
  - **if** $s' \notin \{ OPEN \cup CLOSED \}$
    - parent(s') $\leftarrow s$
    - depth(s') $\leftarrow$ depth(s) + 1
    - $OPEN \leftarrow$ Insert(s',$OPEN$)

if a successor state has not already been seen (ie, is not already on OPEN or CLOSED) put it on OPEN. Its parent is s. Its depth is one greater than depth of its parent (s).

  - **else if** $s' \in OPEN$ Where to insert?
    - parent(s') $\leftarrow$ **arg min** { depth(s), depth(parent(s')) }
    - depth(s') $\leftarrow$ depth(parent(s')) + 1

successor state has been seen but not expanded - Figure out if s is a better choice for parent than its current parent Recompute depth appropriately.

  - **else if** $s' \in CLOSED$
    - parent(s') $\leftarrow$ **arg min** { depth(s), depth(parent(s')) }
    - **for each** $d \in$ descendants(s')
      - depth(d) $\leftarrow$ depth(parent(d)) + 1

successor state has been seen and expanded already — and so all its descendants need depth recomputation if the parent is reassigned.

if goal(s),

path is $\{ s \rightarrow$ parent(s) $\rightarrow$ parent(parent(s)) $\rightarrow \ldots \rightarrow start \}$

Example (Depth-First)

start

2	8	3
1	6	4
7		5

goal

1	2	3
8		4
7	6	5

OPEN : (1)

CLOSED : — (empty)

OPEN : 2 18 □

CLOSED : (1)

OPEN : 3 18 □

CLOSED : (1) (2)

.... order of elements on CLOSED not important

OPEN : 4 8 18 □

CLOSED : (1) (2) (3)

OPEN : 5 8 18 □

CLOSED : 1 2 3 4

OPEN : 6 7 8 18 □

CLOSED : 1 2 3 4 5

...

NOTE: Applicable Rules can check depth bound so no successors to 6 & 7 are generated.

Breadth-First

Items are put at rear of OPEN

OPEN: 1
CLOSED

OPEN: 2 3 4
CLOSED: 1

OPEN: 3 4 5
CLOSED: 1 2

OPEN: 4 5 6 7 8
CLOSED: 1 2 3

...

IDEA: Why can't we find a more intelligent way of guessing the order to try the children nodes?

- instead of "insert at rear" or "insert at front"

  try "insert most promising nodes at front; less promising at rear ...."

"Best First Search" [use a heuristic to evaluate states — low value ⇒ good
high value ⇒ bad

use priority queue to insert states on OPEN using this heuristic —

(requires good heuristic)