

### Depth-First Search Implementation:

For all trees (containing no cycles), a depth-first search can be implemented with the following algorithm.

Use a stack data structure.

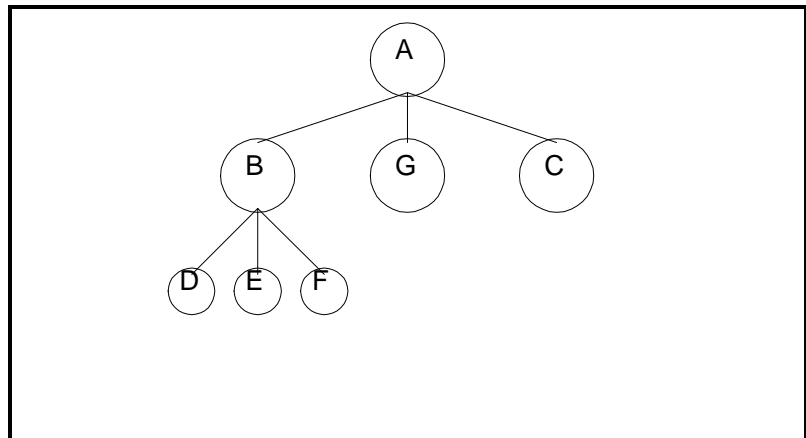
1. Push root node of tree on the stack.
2. Pop the top node off the stack. Call this node N.
3. Push children of node N onto the stack in right-to-left order (so they are popped in left-to-right order).
4. Repeat steps 2 and 3 until stack is empty or goal is reached.

In step 2, node N is "opened". It is closed when step 3 is completed for all of N's descendants.

**Be Careful:** DFS can get off track! If a subtree that does NOT contain a goal node is encountered, the entire subtree gets searched before moving on.

For trees such as this, DFS is worst possible approach.

We'd like to be smarter in our application of DFS.



If we know which children of a node were likely to lead to a goal, we'd like to search those children FIRST.

We can use heuristics to guess at which children are likely to lead to successful searches.

The better the heuristic, the better the method will work.

This approach is called **hill climbing**.

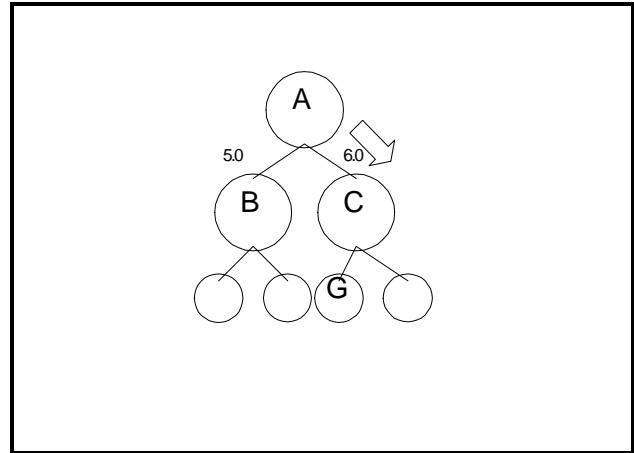
The idea is to expand the best node from the set of children of the current node in a depth-first fashion.

The example in the book uses estimated distance to goal as its heuristic.

### ***Hill Climbing Example:***

Hill climbing is similar to DFS, with the following exception:

Rather than pushing the children of node N on the stack in right-to-left order, apply the heuristic to the children. Push the worst node first, the 2nd worst node next, etc.



### Hill Climbing Algorithm:

1. Push root onto stack.
2. Pop node N from stack.
3. Evaluate heuristic for all children of N. Sort children in worst to best order.
4. Push children of N onto stack in worst to best order.
5. Repeat steps 2, 3, and 4 until goal reached or stack empty.

## Why is it called hill climbing?

When climbing, you always want to take the best foreseeable path to the top.

You don't reach the summit in a single step; you continually choose the best path.

Analogy: Squaw Peak, Camelback Mountain examples.

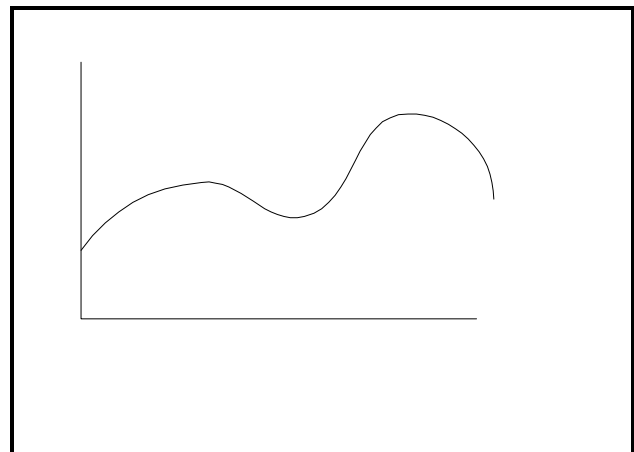
There are some obvious problems with this search method:

### 1. Foothills Problem

The path to the goal may not yield the best heuristic value at each step along the way.

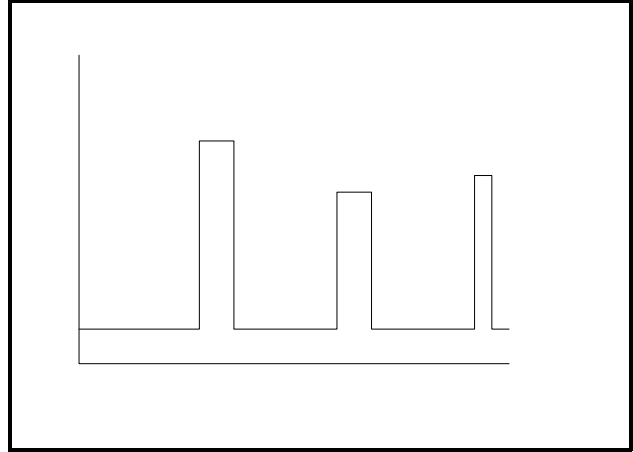
Since hill climbing tries to show improvement at each step along the way, local maximums can lure the algorithm off the best path.

Since hill climbing does not want to take steps backward (which decrease the heuristic value), a local maximum draws it to a false conclusion.



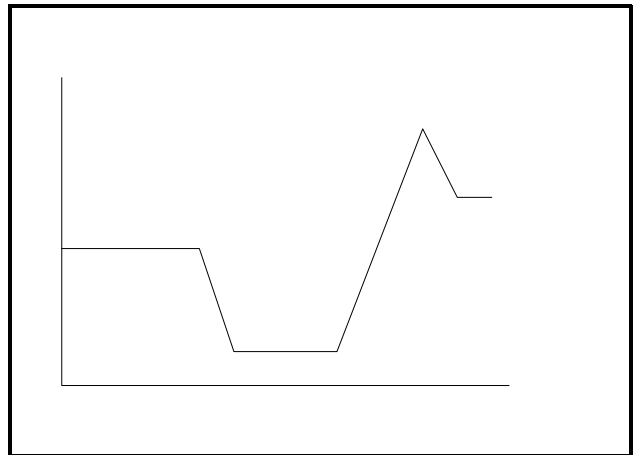
## Problem 2. Plateaus

Hill climbing looks at all possible next steps and takes the best. In a plateau, there is no local improvement, therefore no best step. The method breaks down.



## Problem 3. Ridges

Imagine a plain, a canyon, and a hill on the other side of the canyon.



### Solutions to problems:

Foothills - backtracking

Plateaus - large step size

Ridges - apply multiple moves before testing heuristic

Breadth-First Search Implementation:

When depth-first search and hill climbing are bad choices, breadth first search may be useful.

BFS looks for a goal node among all children of  $n$  before proceeding.

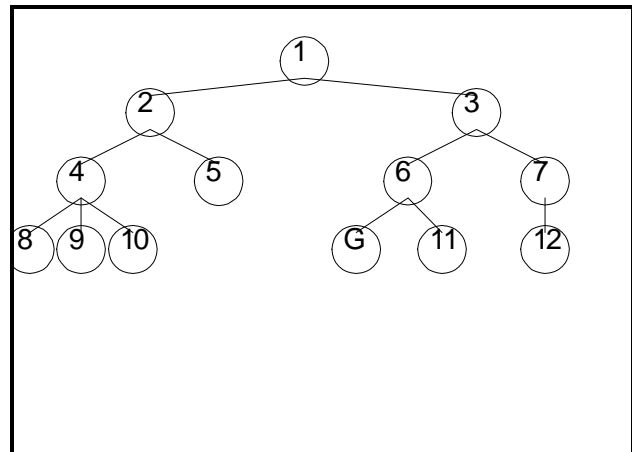
Search paths for example:

DFS      1 2 4 8 9 10 5 3 6 G

HC      1 3 6 G (given an ideal heuristic)

BFS      1 2 3 4 5 6 7 8 9 10 G

BFS does not work well when goal nodes are deeply nested in a tree. However, if the alternatives to search are all roughly equal, BFS will work just fine.



Example: What happens if node 5 is our goal?

BFS is just like DFS, except it uses a queue rather than a stack. (Winston's terminology uses a queue for both searches, adding children to the front or back as appropriate.)

BFS algorithm:

1. Enqueue the root node on the queue.
2. Dequeue the first node N from the queue.
3. Enqueue the children of N into the queue in left-to-right order. (Beam search: use heuristic to determine enqueueing order.)
4. Repeat steps 2 and 3 until goal found or queue is empty.

If the tree is very wide, BFS will not perform well. If it is narrow, BFS looks better and better.

## Beam search:

Breadth-first search with heuristics.

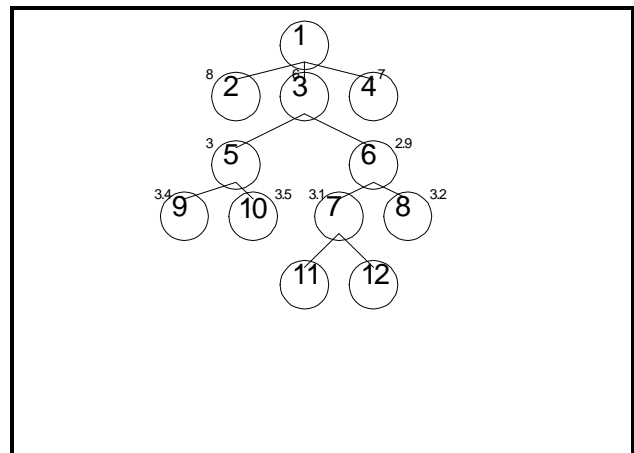
Algorithm is same as BFS, but we enqueue only the  $W$  best nodes at any level, ignoring the rest.

If branching factor =  $B$ , at each level only  $W \times B$  of the  $B^d$  nodes are enqueued.

Enqueue best nodes in best-to-worst order. The algorithm will search the best child node first.

If we search the best OPEN node next, no matter where we encountered it, we have a BEST FIRST SEARCH.

Best First Search requires a sorted data structure. Uses hill climbing with a sorted stack or beam search with a sorted queue.



We must be careful not to get caught up in search problems.

Search is seductive; it draws our attention away from the goal of achieving intelligent behavior in machines. We wish to do LESS search rather than more.

"More knowledge means less search."

Search is, nevertheless, one ingredient of discovery. Hypothesizing new paths in our search may lead us down a path of success which we did not know.

Discovery through search:

Douglas Lenat - in *AM* and *Eurisko*

Cray Blitz - discovered a win position from what experts thought was a stalemate. (Took 100 moves in proper sequence.)

## Search Graph Characteristics:

	Shallow	Deep
Narrow	BFS/DFS	BFS/Beam
Wide	HC/DFS	HC/Best  (or Beam with small branch factor)