**best first search**

(nao percebi mt bem)

choose node with best value of best-first search evaluation function

**breadth-first search (bfs)**

nodes at lowest depth expanded first

goes always horizontal, if no nodes in that depth left, go down 1 depth

takes long time and space

only good for small problems

A picture containing clock

Description automatically generated

**Dijkstra’s algorithm/Uniform Cost Search**

always expand border node with lowest cost (basically greedy)

**Depth-First Search**

expand one of deepest nodes on tree

llittle memory required, good for problems with multiple solutions

cannot be used in infinite-depth trees

sometimes limit depth is defined, becoming **search with limited depth**

A picture containing group, bunch

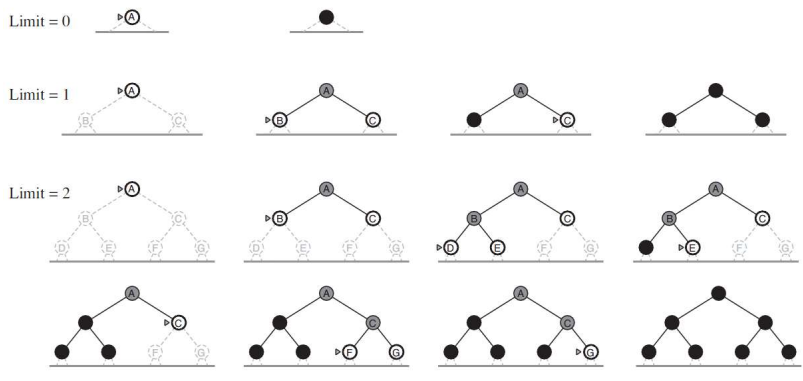
Description automatically generated

**iterative deepening search**

perform **limited depth search**, iteratively, always increasing depth limit

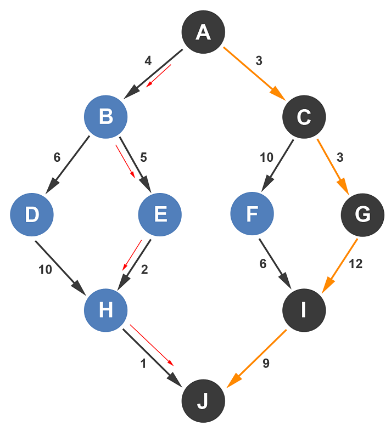
normally best strategy for problems with big search space where depth of solution not known

nodes as bottom level (depth d) generated once, next-to-bottom level generated twice, etc



**bidirectional search**

run two search algorithms, one from initial state and one from goal state



**greedy search**

expand node with lowest cost of heuristic function

**A\* algorithm**

combines greedy search with uniform search

uses function: f(n) = g(n) + h(n)

* g(n) -> total cost, so far, to reach state n
* h(n) -> estimated cost to reach objective (CANNOT overestimate cost)
* normally for h(n) there is a table with the values

for all current avaibable nodes, choose one with lowest f(n) and expand the children, then do same thing again

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated

**weighted A\* search**

if we allow A\* search to use heuristics that may overestimate, we risk missing optimal solution but can reduce drastically nodes expanded

f(n) = g(n) + W \* h(n)

Diagram

Description automatically generated

**minimax algorithm**

generate tree to end states (limited depth first search)

apply utility function to these states

calculate utility values back to root, one layer at a time

choose movement with highest value

Radar chart

Description automatically generated with medium confidence

Assuming that MAX is the first to play, the selected move would be A->D with expected value of 8, then we assume the oponent would play D->J and then we would play J->V

**MinimaxCutoff**

identical to Minimax but:

terminal-test replaced by cutoff

utility is replaced by evaluation (evaluates the reached position)

alpha-beta (α, β) cuts:

alpha-beta cuts do not affect final result

good ordering improves cutting efficiency

α is best value (for Max) found so far in current path

if V worse than α, Max should cut off the branch

β is defined same way but for Min

Diagram

Description automatically generated with medium confidence

**negamax**

relies on the zero-sum property of a two-player game

relies on the fact that: max(a,b)= -min(-a,-b)

for example below:

for each level, always choose highest value possible, and then multiply it by -1 and save that number

for example:

E-F-G: 9, -6, 0 -> highest is 9 -> B becomes 9 \* -1 = -9

Diagram

Description automatically generated

**monte carlo tree search**

works well for games with large branching factor

good when difficult to define appropriate evaluation function

selection: start from root, select successive childnode until leaf node “L” is reached (leaf node is any node that has potential child from which no simulation has yet been initiated. Should be a way of expanding towards the most promising moves

expansion: unless “L” ends the game (win/loss/draw), create child nodes and choose node “C” from one of them. Child nodes are any valid moves from game position at “L”

simulation: complete one random playout at node “C”. Playout may be as simple as choosing uniform random moves until game is decided

backpropagation: use result of playout to update information in nodes on path from “C” to “R”

**ExpectiMiniMax**

search tree must include probability nodes

decision is made based on expected value

Diagram

Description automatically generated