



# MACHINE INTELLIGENCE UNINFORMED SEARCH STRATEGIES

---

**K.S.Srinivas**

Department of Computer Science and Engineering

# MACHINE INTELLIGENCE

---

## UNINFORMED SEARCH STRATEGIES

**Srinivas K S.**

Associate Professor, Department of Computer Science

# MACHINE INTELLIGENCE

## Uninformed Search Strategy

---



- Uninformed search strategies use only the information available in the problem definition
- The term means that the strategies have no additional information about states beyond that provided in the problem definition.
- All they can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the order in which nodes are expanded
- We will be discussing 5 types of uninformed search
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

## MACHINE INTELLIGENCE

### Breadth-First Search

---

- Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- This is achieved very simply by using a FIFO queue for the frontier.
- let us see the pseudo code and simultaneously explore a problem

# MACHINE INTELLIGENCE

## Problem +algo

**function** BREADTH-FIRST-SEARCH(problem) **returns** a solution, or failure

node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0

**if** problem.GOAL-TEST(node.STATE) **then return** SOLUTION(node)

frontier  $\leftarrow$  a FIFO queue with node as the only element

explored  $\leftarrow$  an empty set

**loop do**

**if** EMPTY?(frontier) **then return** failure

node  $\leftarrow$  POP(frontier) /\* chooses the shallowest node in frontier \*/

add node.STATE to explored

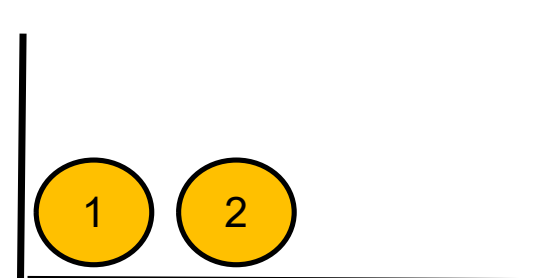
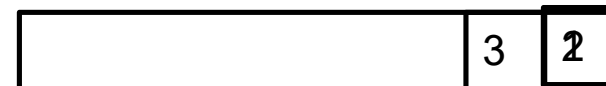
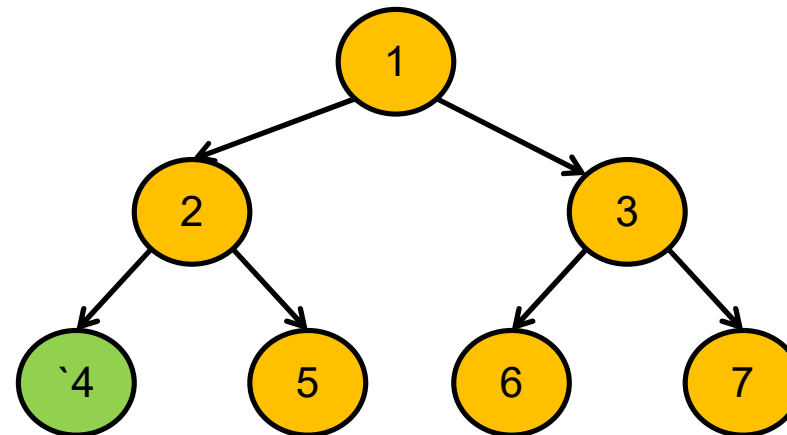
**for** each action in problem.ACTIONS(node.STATE) **do**

child  $\leftarrow$  CHILD-NODE(problem,node,action)

**if** child.STATE is not in explored or frontier **then**

**if** problem.GOAL-TEST(child.STATE) **then return** SOLUTION(child)

frontier  $\leftarrow$  INSERT(child,frontier)



# MACHINE INTELLIGENCE

## How good is it???

### COMPLETENESS



we can easily say its complete.If the shallowest goal node is at some finite depth  $d$ , breadth-first search will eventually find it after generating all shallower nodes with finite branching factor  $b$ .

### TIME COMPLEXITY



for a uniform tree where every state  $b$  has  $b$  successor.  
root would generate  $b$  nodes at first level each of which generates  $b$  more nodes.In the worst case scenario for a tree of depth  $d$  the number of nodes generated would be  $b+b^2+b^3+\dots+b^d=O(b^d)$

### SPACE COMPLEXITY



For breadth-first graph every node generated remains in memory.  
There will be  $O(b^{d-1})$  in the explored set and  $O(b^d)$  in the queue.

### OPTIMALITY



the shallowest goal node is not necessarily the optimal one  
technically, breadth-first search is optimal if the path cost is a non decreasing function of the depth of the node. The most common such scenario is that all actions have the same cost.

## MACHINE INTELLIGENCE

### Uniform Cost Search

---



- when all the step cost are equal bfs is optimal because it always expands the shallowest unexplored node
- uniform-cost search is an extension bf when the cost is not uniform.
- uniform-cost search expands the node  $n$  with the lowest path cost  $g(n)$
- This is done by storing the frontier as a priority queue ordered by  $g$ .
- let's see the pseudo code and then explore a problem

# MACHINE INTELLIGENCE

## Pseudo-code

---

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered-by PATH-COST, with node as the only element
explored ← an empty set
loop do
  if EMPTY?(frontier) then return failure
  node ← POP(frontier) /* chooses the lowest-cost node in frontier */
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  add node.STATE to explored
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem,node,action)
    if child.STATE is not in explored or frontier then frontier ← INSERT(child,frontier)
  else if child.STATE is in frontier with higher PATH-COST then replace that frontier node with child
```

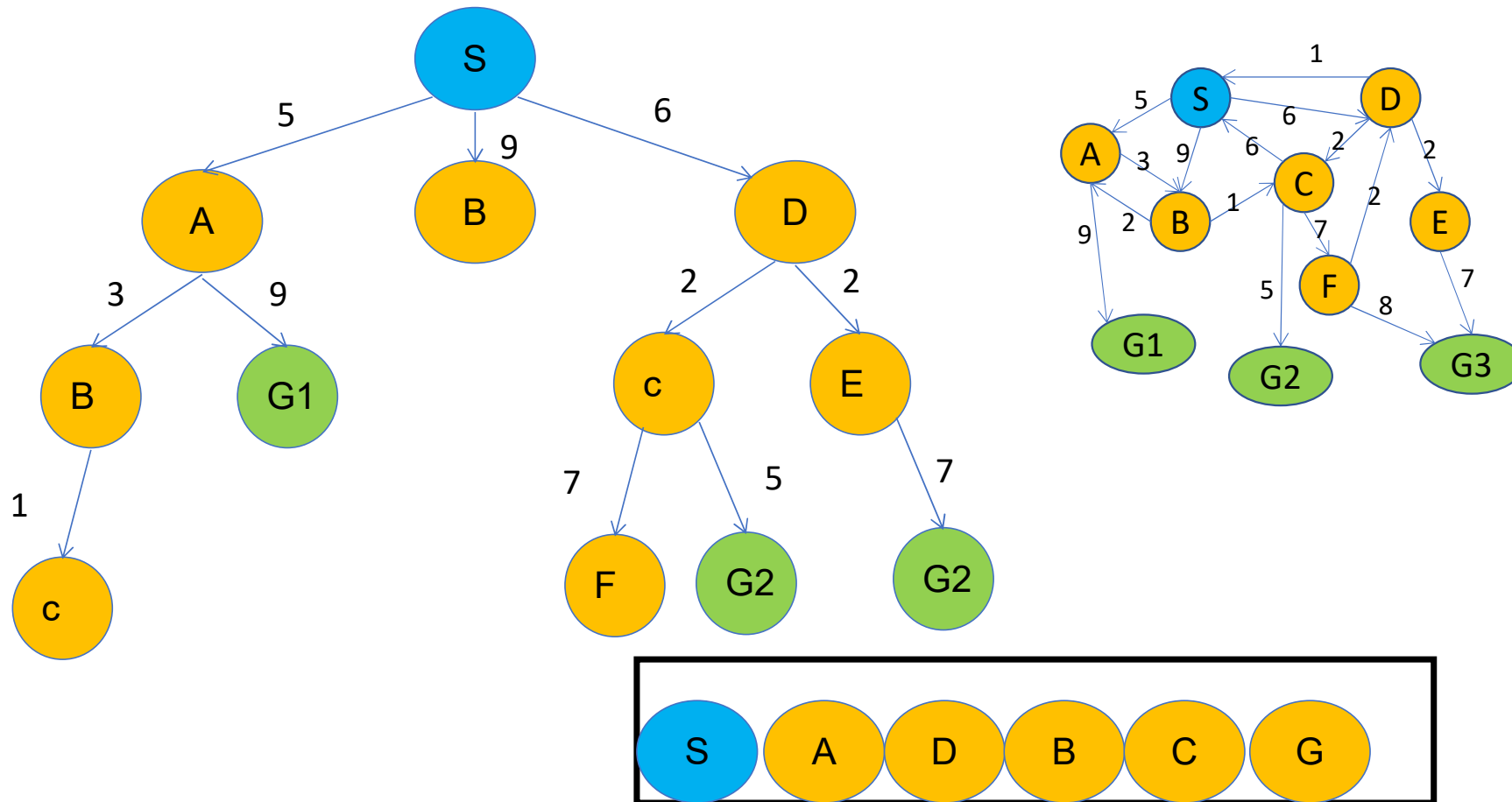




## Problem



~~10  
 11  
 12  
 13  
 14  
 15  
 16  
 17  
 18  
 19  
 20  
 21  
 22  
 23  
 24  
 25  
 26  
 27  
 28  
 29  
 30  
 31  
 32  
 33  
 34  
 35  
 36  
 37  
 38  
 39  
 40  
 41  
 42  
 43  
 44  
 45  
 46  
 47  
 48  
 49  
 50  
 51  
 52  
 53  
 54  
 55  
 56  
 57  
 58  
 59  
 60  
 61  
 62  
 63  
 64  
 65  
 66  
 67  
 68  
 69  
 70  
 71  
 72  
 73  
 74  
 75  
 76  
 77  
 78  
 79  
 80  
 81  
 82  
 83  
 84  
 85  
 86  
 87  
 88  
 89  
 90  
 91  
 92  
 93  
 94  
 95  
 96  
 97  
 98  
 99  
 100  
 101  
 102  
 103  
 104  
 105  
 106  
 107  
 108  
 109  
 110  
 111  
 112  
 113  
 114  
 115  
 116  
 117  
 118  
 119  
 120  
 121  
 122  
 123  
 124  
 125  
 126  
 127  
 128  
 129  
 130  
 131  
 132  
 133  
 134  
 135  
 136  
 137  
 138  
 139  
 140  
 141  
 142  
 143  
 144  
 145  
 146  
 147  
 148  
 149  
 150  
 151  
 152  
 153  
 154  
 155  
 156  
 157  
 158  
 159  
 160  
 161  
 162  
 163  
 164  
 165  
 166  
 167  
 168  
 169  
 170  
 171  
 172  
 173  
 174  
 175  
 176  
 177  
 178  
 179  
 180  
 181  
 182  
 183  
 184  
 185  
 186  
 187  
 188  
 189  
 190  
 191  
 192  
 193  
 194  
 195  
 196  
 197  
 198  
 199  
 200  
 201  
 202  
 203  
 204  
 205  
 206  
 207  
 208  
 209  
 210  
 211  
 212  
 213  
 214  
 215  
 216  
 217  
 218  
 219  
 220  
 221  
 222  
 223  
 224  
 225  
 226  
 227  
 228  
 229  
 230  
 231  
 232  
 233  
 234  
 235  
 236  
 237  
 238  
 239  
 240  
 241  
 242  
 243  
 244  
 245  
 246  
 247  
 248  
 249  
 250  
 251  
 252  
 253  
 254  
 255  
 256  
 257  
 258  
 259  
 260  
 261  
 262  
 263  
 264  
 265  
 266  
 267  
 268  
 269  
 270  
 271  
 272  
 273  
 274  
 275  
 276  
 277  
 278  
 279  
 280  
 281  
 282  
 283  
 284  
 285  
 286  
 287  
 288  
 289  
 290  
 291  
 292  
 293  
 294  
 295  
 296  
 297  
 298  
 299  
 300  
 301  
 302  
 303  
 304  
 305  
 306  
 307  
 308  
 309  
 310  
 311  
 312  
 313  
 314  
 315  
 316  
 317  
 318  
 319  
 320  
 321  
 322  
 323  
 324  
 325  
 326  
 327  
 328  
 329  
 330  
 331  
 332  
 333  
 334  
 335  
 336  
 337  
 338  
 339  
 340  
 341  
 342  
 343  
 344  
 345  
 346  
 347  
 348  
 349  
 350  
 351  
 352  
 353  
 354  
 355  
 356  
 357  
 358  
 359  
 360  
 361  
 362  
 363  
 364  
 365  
 366  
 367  
 368  
 369  
 370  
 371  
 372  
 373  
 374  
 375  
 376  
 377  
 378  
 379  
 380  
 381  
 382  
 383  
 384  
 385  
 386  
 387  
 388  
 389  
 390  
 391  
 392  
 393  
 394  
 395  
 396  
 397  
 398  
 399  
 400  
 401  
 402  
 403  
 404  
 405  
 406  
 407  
 408  
 409  
 410  
 411  
 412  
 413  
 414  
 415  
 416  
 417  
 418  
 419  
 420  
 421  
 422  
 423  
 424  
 425  
 426  
 427  
 428  
 429  
 430  
 431  
 432  
 433  
 434  
 435  
 436  
 437  
 438  
 439  
 440  
 441  
 442  
 443  
 444  
 445  
 446  
 447  
 448  
 449  
 450  
 451  
 452  
 453  
 454  
 455  
 456  
 457  
 458  
 459  
 460  
 461  
 462  
 463  
 464  
 465  
 466  
 467  
 468  
 469  
 470  
 471  
 472  
 473  
 474  
 475  
 476  
 477  
 478  
 479  
 480  
 481  
 482  
 483  
 484  
 485  
 486  
 487  
 488  
 489  
 490  
 491  
 492  
 493  
 494  
 495  
 496  
 497  
 498  
 499  
 500  
 501  
 502  
 503  
 504  
 505  
 506  
 507  
 508  
 509  
 510  
 511  
 512  
 513  
 514  
 515  
 516  
 517  
 518  
 519  
 520  
 521  
 522  
 523  
 524  
 525  
 526  
 527  
 528  
 529  
 530  
 531  
 5~~



## MACHINE INTELLIGENCE

### Analyzing

---



- It is easy to see that uniform-cost search is optimal in general. First, we observe that whenever uniform-cost search selects a node  $n$  for expansion, the optimal path to that node has been found.
- Uniform-cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost actions. Completeness is guaranteed provided the cost of every step exceeds some small positive constant  $\epsilon$ .
- Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of  $b$  and  $d$ . Instead, let  $C^*$  be the cost of the optimal solution, and assume that every action costs at least  $\epsilon$ . Then the algorithm's worst-case time and space complexity is  $O(b^{1+\lceil C^*/\epsilon \rceil})$ , which can be much greater than  $b^d$ .

# MACHINE INTELLIGENCE

## How good is it??

COMPLETENESS

yes its  
complete

TIME COMPLEXITY

$O(b^{1+[C_{\square}/\epsilon]}),$

SPACE COMPLEXITY

$O(b^{1+[C_{\square}/\epsilon]}),$

OPTIMALITY

yes it finds  
an optimal  
solution

## MACHINE INTELLIGENCE

### Depth-First Search

---

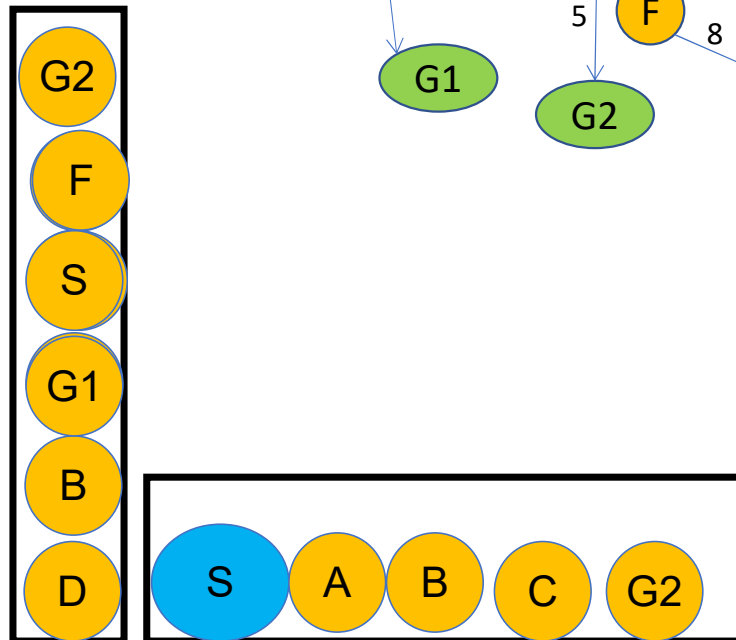
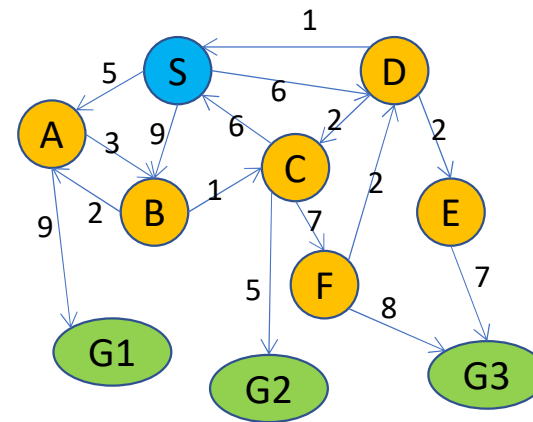
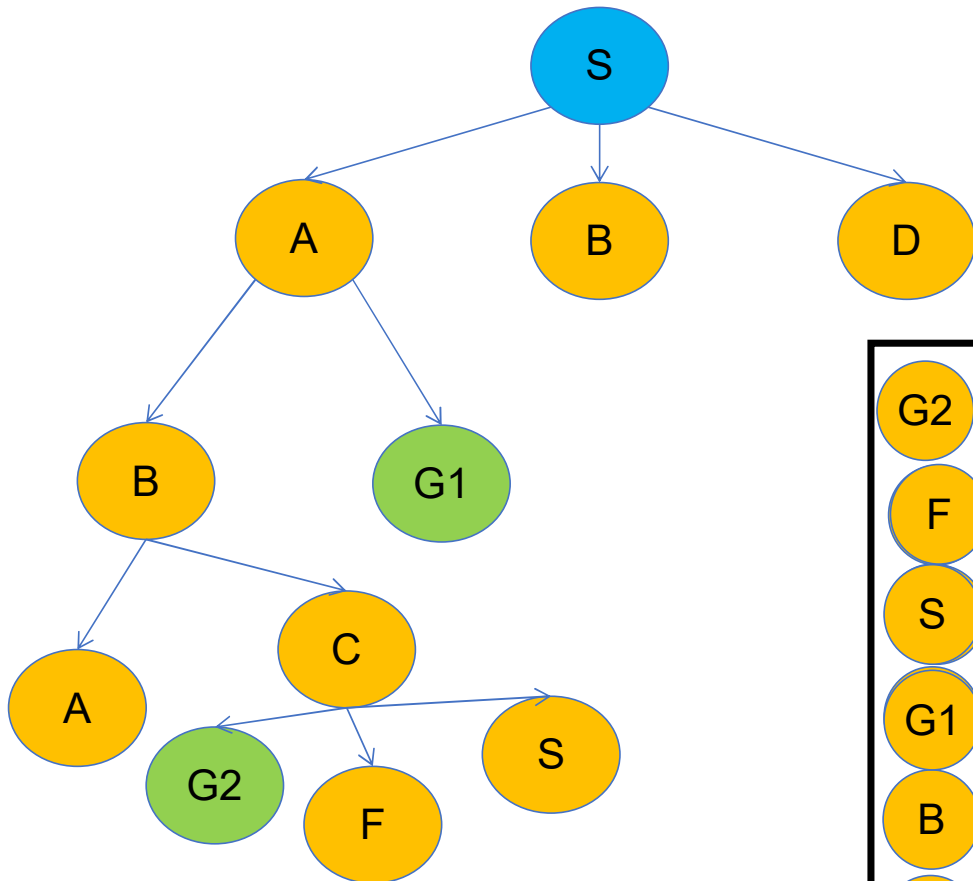


- Depth-first search always expands the deepest node in the current frontier of the search tree.
- depth-first search uses a LIFO queue.
- A LIFO queue means that the most recently generated node is chosen for expansion .
- it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn.
- lets see how it actually does the job for us.

# MACHINE INTELLIGENCE

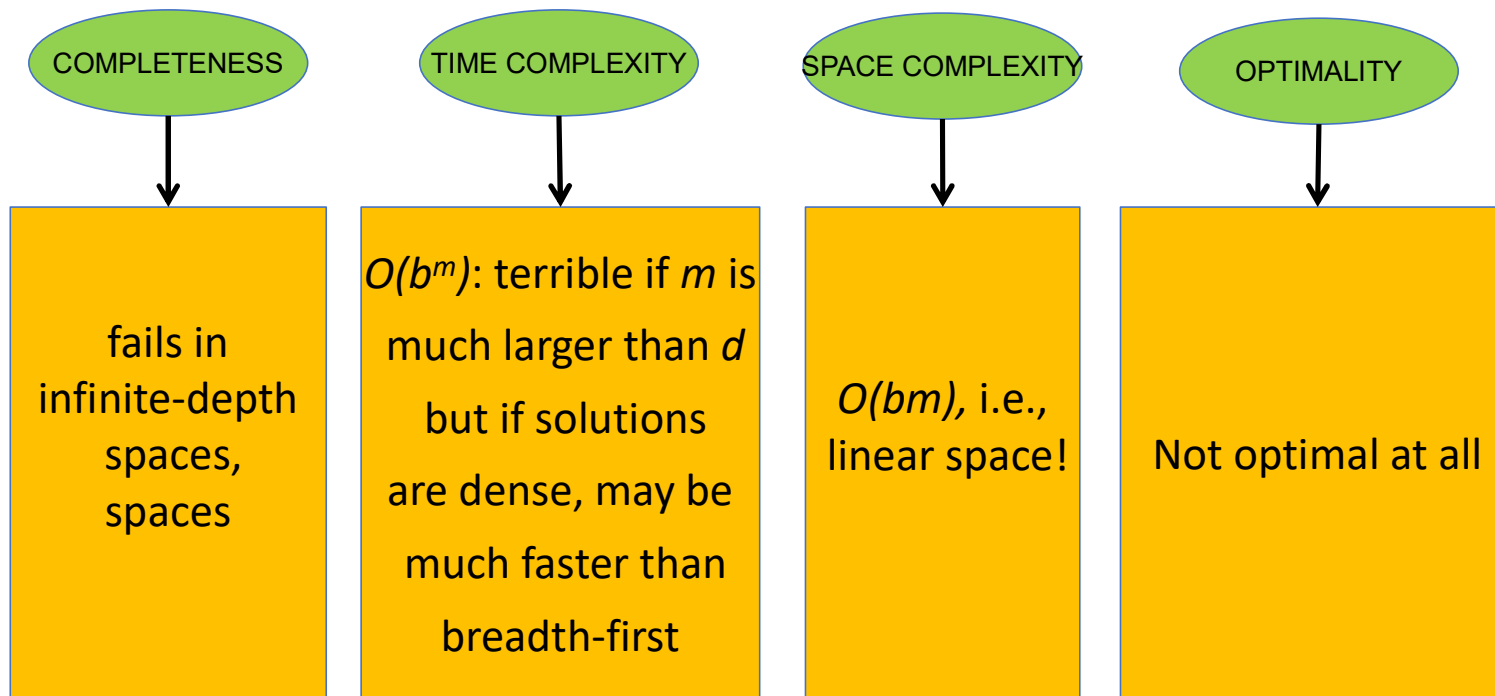
## Problem

Find the shortest path from S to G2. The graph is a directed graph with nodes S, A, B, C, D, E, F, G1, G2, G3. The edges and their weights are: S to A (5), S to B (3), S to D (1); A to B (2), A to G1 (9); B to A (9), B to C (1), B to G2 (2); C to B (1), C to F (7), C to G2 (5); D to S (1), D to C (2), D to E (2); E to D (2), E to G3 (7); F to C (7), F to G3 (8); G1 to A (9), G1 to B (2); G2 to C (5), G2 to F (8); G3 to E (7), G3 to F (8).



# MACHINE INTELLIGENCE

## How good is it???



# MACHINE INTELLIGENCE

## Depth-limit Search

---

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit  $l$ .
- That is, nodes at depth  $l$  are treated as if they have no successors. This approach is called depth-limited search.
- let us see the pseudo code for depth-limit search



# MACHINE INTELLIGENCE

## Pseudo-code

---



```
function DEPTH-LIMITED-SEARCH(problem,limit) returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE),problem,limit)

function RECURSIVE-DLS(node,problem,limit) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit =0 then return cutoff
    else cutoff occurred?←false
    for each action in problem.ACTIONS(node.STATE)
        do child ←CHILD-NODE(problem,node,action)
        result ←RECURSIVE-DLS(child,problem,limit – 1)
        if result = cutoff then cutoff occurred?←true
        else if result != failure then return result
    if cutoff occurred? then return cutoff else return failure
```



# MACHINE INTELLIGENCE

## Iterative deepening Search

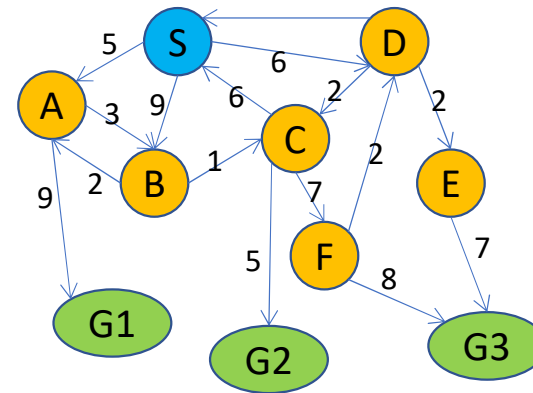
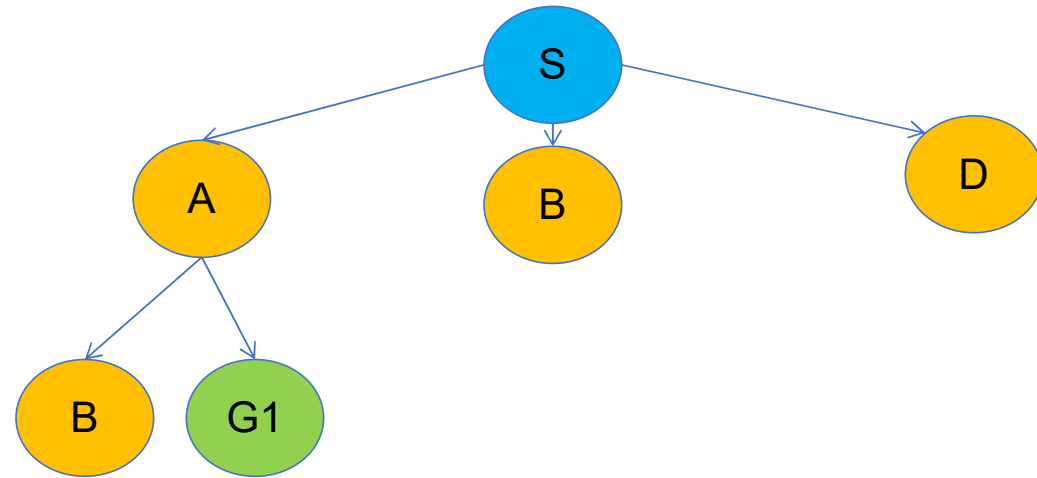
- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or fail-  
ure  
  inputs: problem, a problem  
  for depth  $\leftarrow$  0 to  $\infty$  do  
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)  
    if result  $\neq$  cutoff then return result
```

# MACHINE INTELLIGENCE

## Problem

Find the shortest path from S to G1, G2, and G3. The graph is as follows:



limit l = 2

Number of nodes generated in a depth-limited search to depth  $d$  with branching factor  $b$ :

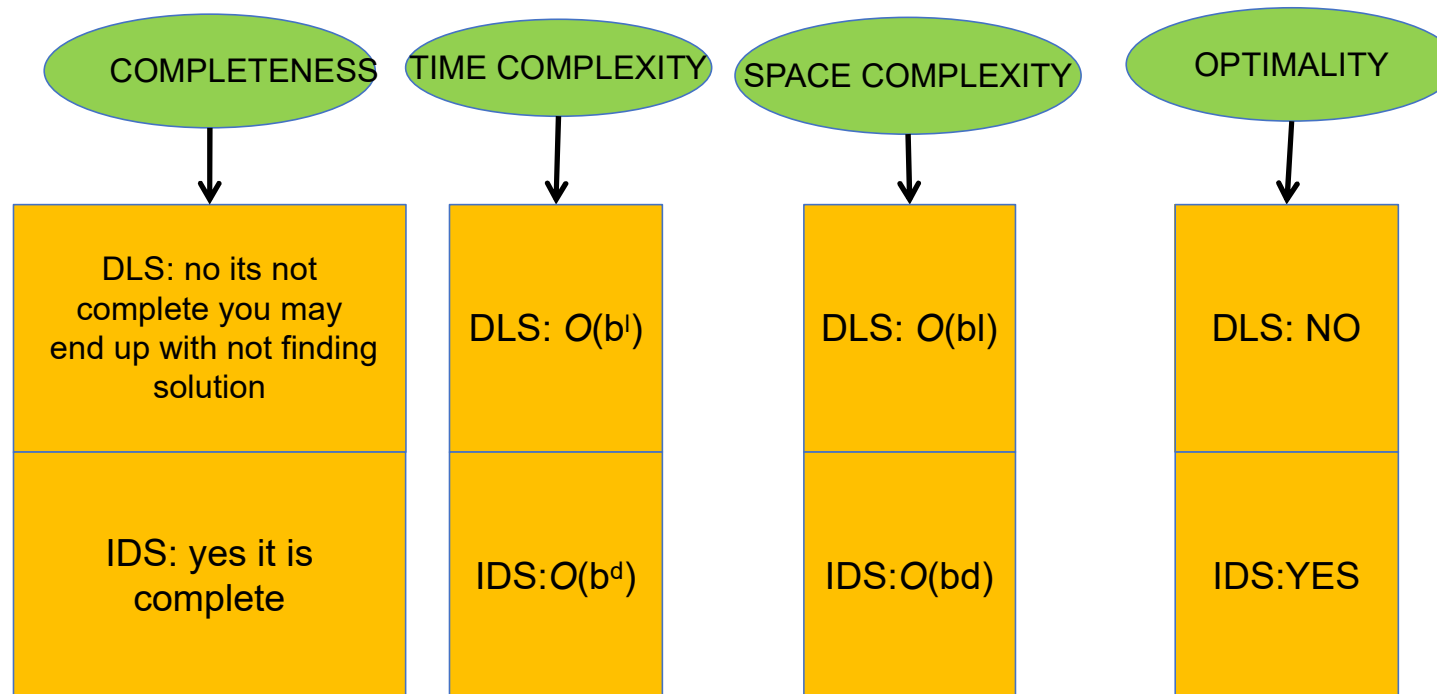
$$N_{DLS} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$$

Number of nodes generated in an iterative deepening search to depth  $d$  with branching factor  $b$ :

$$N_{IDS} = (d+1)b^0 + d b^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

# MACHINE INTELLIGENCE

## How good is it??



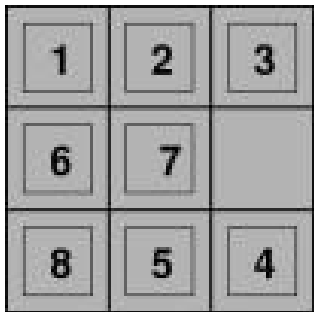
# MACHINE INTELLIGENCE

## Applications

- Now let us look at some of the real world example that our search algorithms include.



## 8-puzzle



Can you try to formalize this search problem by defining the 5 components we discussed earlier?

STATES

Locations of tiles

START STATE

Given position of tiles

GOAL STATE

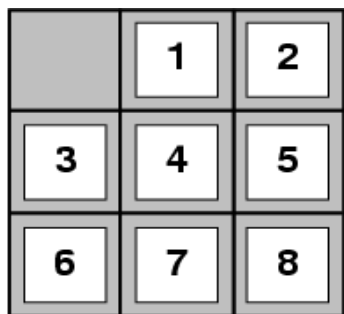
GIVEN

ACTIONS

move blank left, right, up, down

COST

1 per move



Goal State

- MAZE PROBLEM



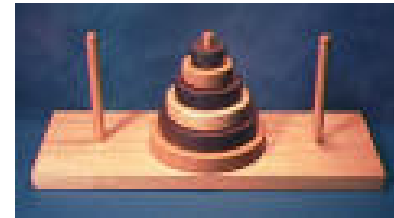
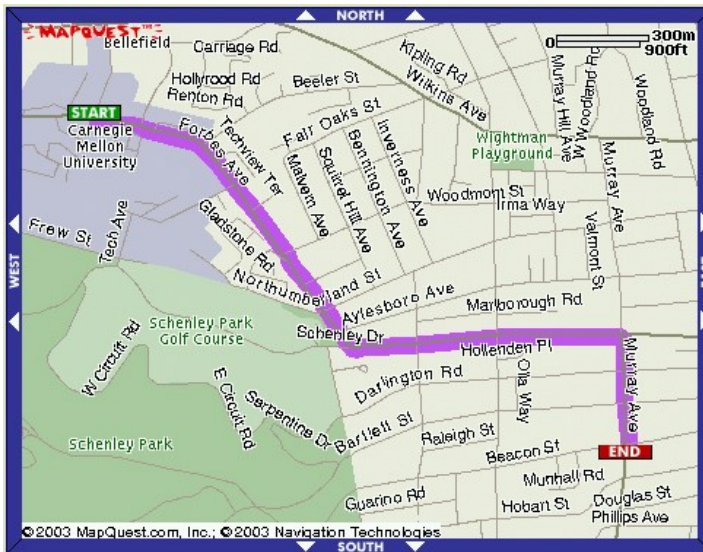
can you guess what kind of search algorithm it may be using?

DFS

# MACHINE INTELLIGENCE

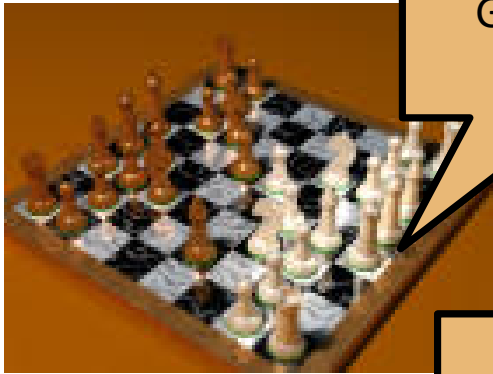
## Applications

some more applications

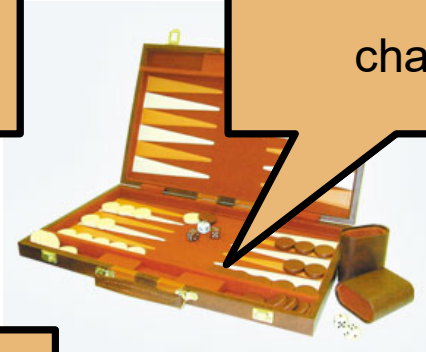


# MACHINE INTELLIGENCE

## Our definition excludes

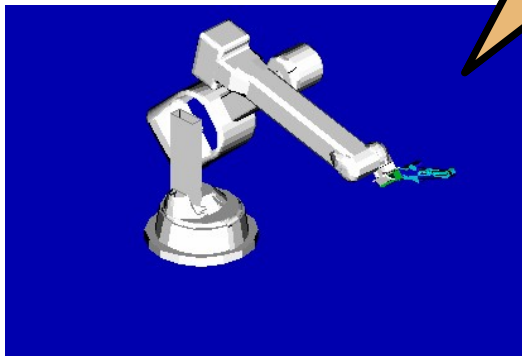


Game against  
adversary



chance

Continuum  
(infinite number)  
of states



All of the above, plus  
distributed team control





THANK YOU

---

**K.S.Srinivas**  
**srinivasks@pes.edu**

+91 80 2672 1983 Extn 701