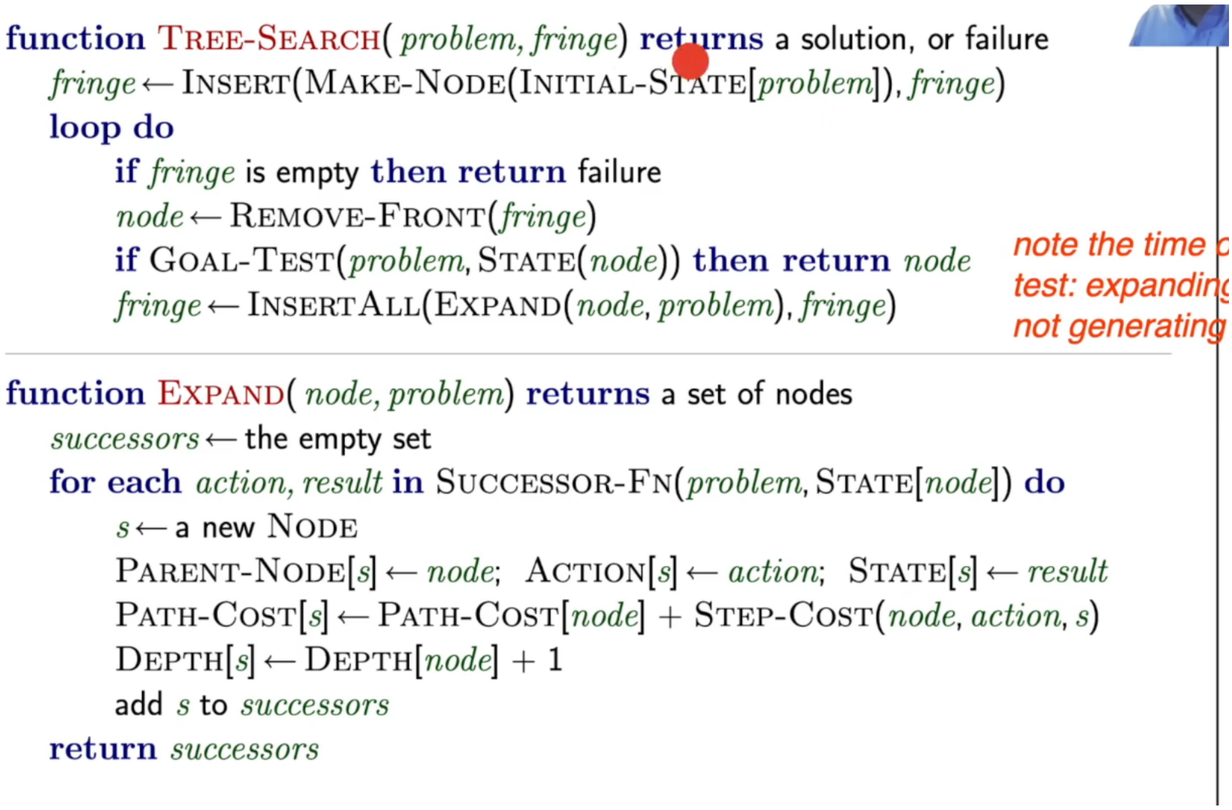
goal test：检测是否是目标

path cost：计算路径和（积）

solution：策略，一系列动作（到goal才算）

> 离散的状态，没有中间状态，确定的转移，能看到所有状态

树搜索算法



```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

note the time c
test: expandin
not generating

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

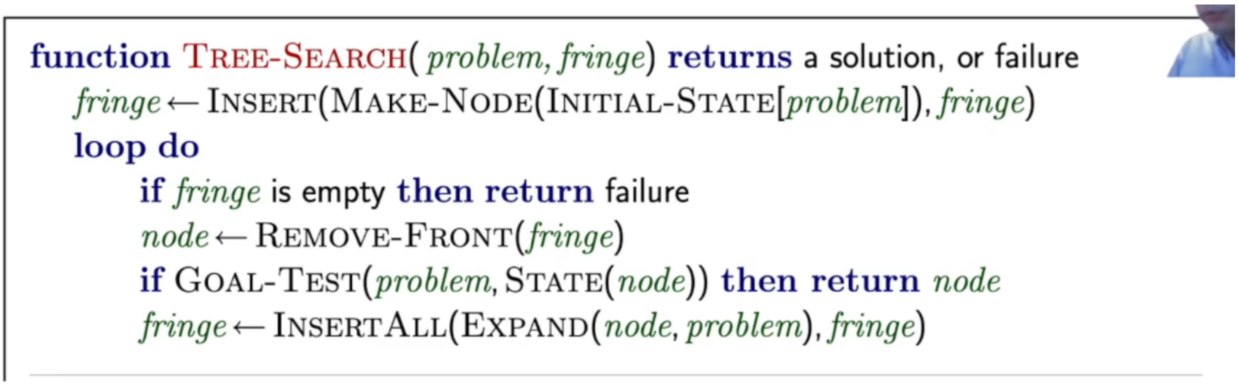
> 把根节点放进空间
>
> 返回时所有都返回结束，找到solution

Expand

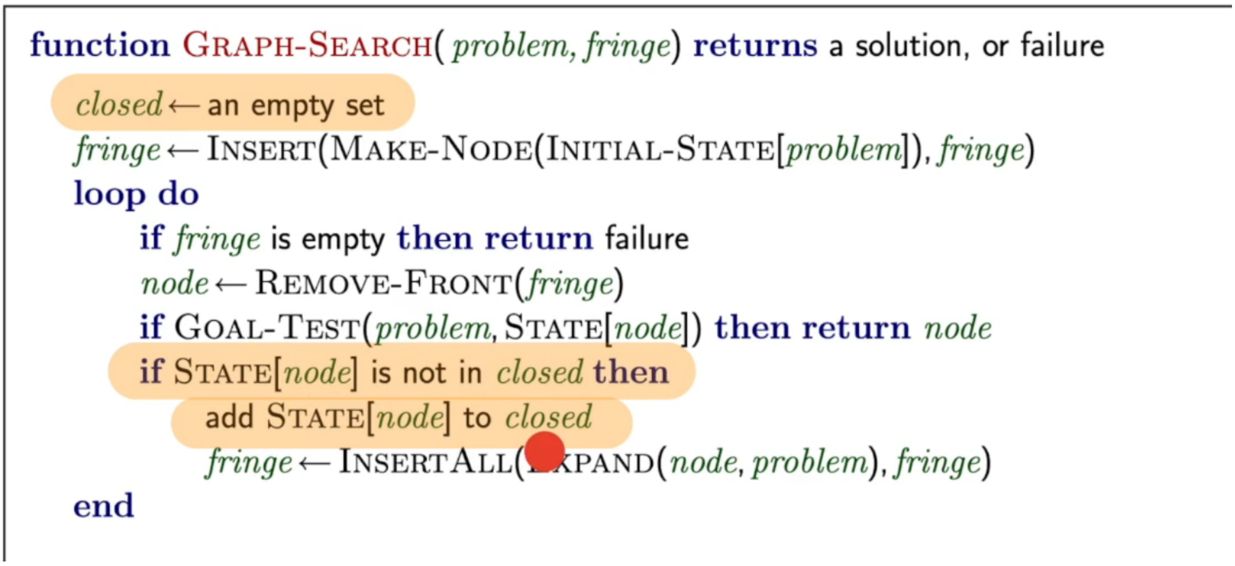
> 抓一个出去，展开子节点，再扔回去，直到找到goal
>
> 展开node的子节点，Insertall入空间
>
> 后继初始化为空
>
> 对每个node考虑所有可做的action，
>
> 每个action新建一个Node数据结构，分别记录3个值
>
> 计算cost，深度

- 只在往出拿的时候进行goal test，回放的时候不行！

图搜索算法



```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```



```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

> 唯一不同：需要检查重复节点，用close set
>
> 可以两个不同node（）数据结构放同一个state（问题状态）

搜索策略

- 完整性：能找到
- 最优性：最小cost
- 时空复杂度：分叉数b、深度d、状态空间深度m

uninformed搜索策略

> 没有告诉其他信息，只能用最定义基本

1.宽度优先搜索

> 一层一层来
>
> 用FIFO queue（先入先出）

- 完整性：ok
- 时间&空间：O(b^(d+1))

> 只保留最后一层，空间复杂度还是这么多，因为最后一层太多了

2.深度优先搜索

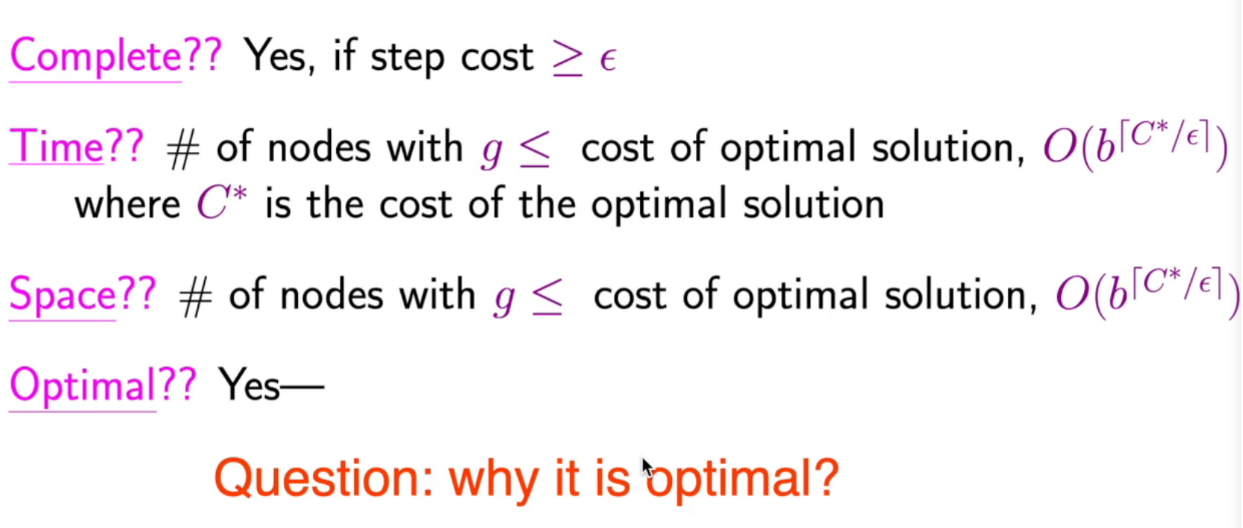
> LIFO queue
>
> 节省空间

- 完整性：no（可能重复）
- 时间O（b^m）
- 空间O（bm）

3.uninform-cost search

> cost优先，优先队列
>
> 碰到目标也不会终止，每次延申cost最小路径
>
> 解释了：一定要拿的时候再goal test

- 完整性:ok
- 时空：



Complete?? Yes, if step cost $\geq \epsilon$

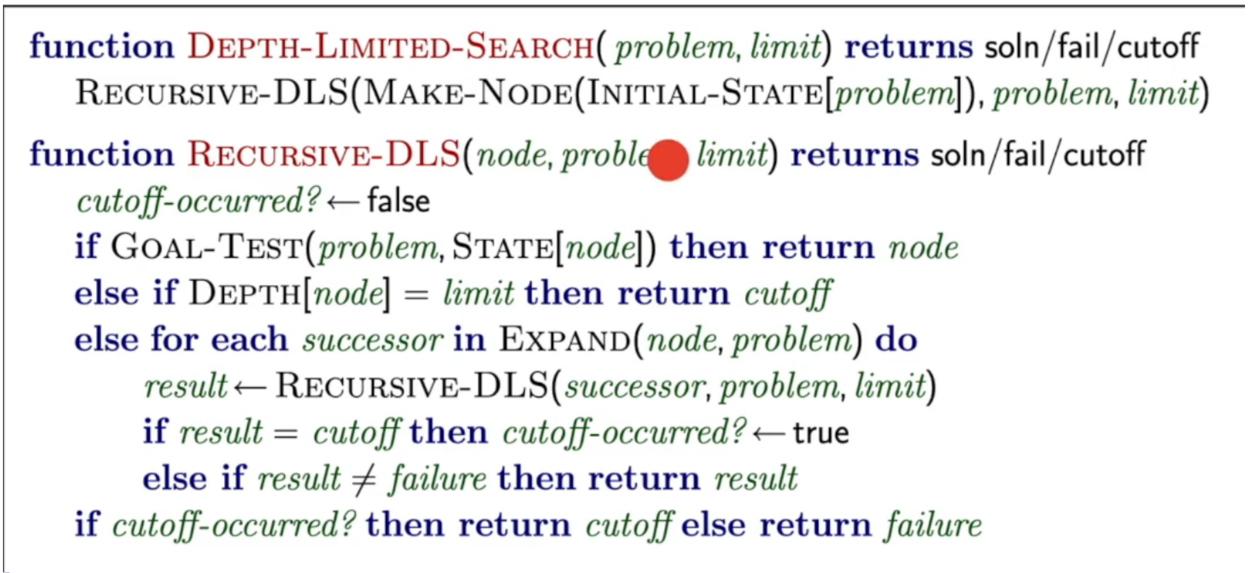
Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—

Question: why it is optimal?

4.深度有限优先搜索



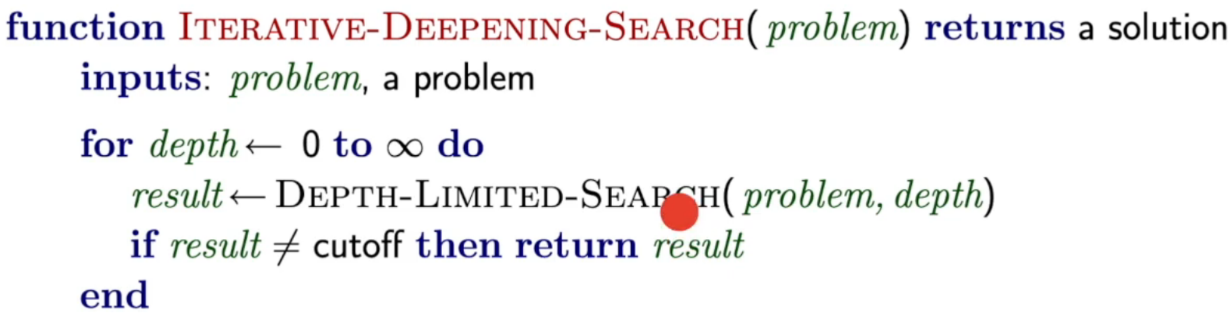
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result $\neq$ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

> 深度到达limit返回cutoff，终止标志

> return值有三种
>
> 递归时深度需要+1（上图错了）

逐步假定有限搜索深度

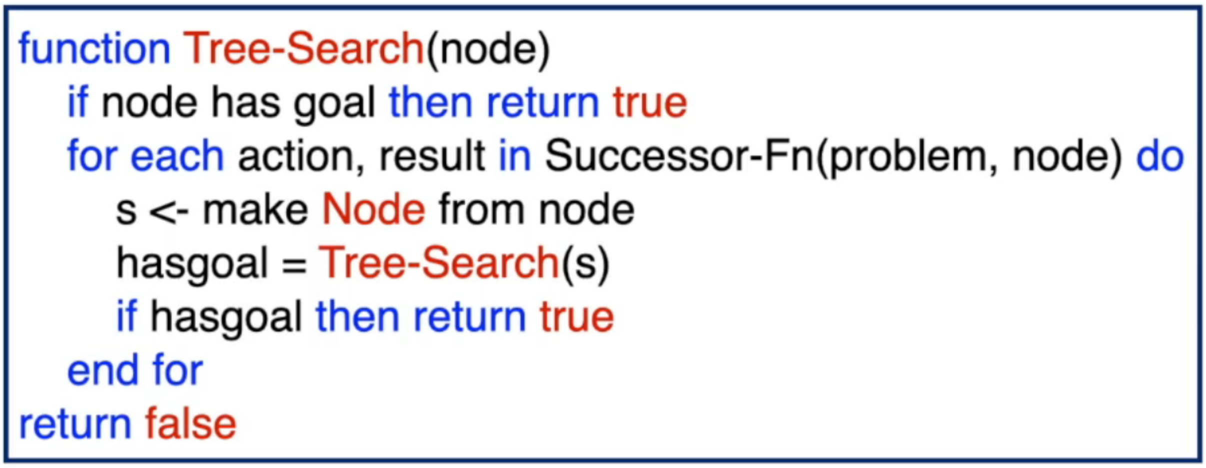
> 有点浪费重复，但并不多，树是指数增长的



```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

用递归

> 树很深时容易栈溢出



```
function Tree-Search(node)
    if node has goal then return true
    for each action, result in Successor-Fn(problem, node) do
        s <- make Node from node
        hasgoal = Tree-Search(s)
        if hasgoal then return true
    end for
return false
```

用队列

> 在堆中申请栈空间
>
> 可以调大java虚拟机的内存

informed搜索策略

> 人类给一个偏好

heuristic function：对问题拍脑袋规则，启发式