COMP 472 Artificial Intelligence
State Space Search part #3
Uninformed Search video #2

- Russell & Norvig – Section 3.4
- see also: https://www.javatpoint.com/ai-uninformed-search-algorithms

Today

1. State Space Representation
2. State Space Search
   a) Overview 

   b) Uninformed search
      1. Breadth-first and Depth-first
      2. Depth-limited Search
      3. Iterative Deepening
      4. Uniform Cost
   c) Informed search
      1. Intro to Heuristics
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary

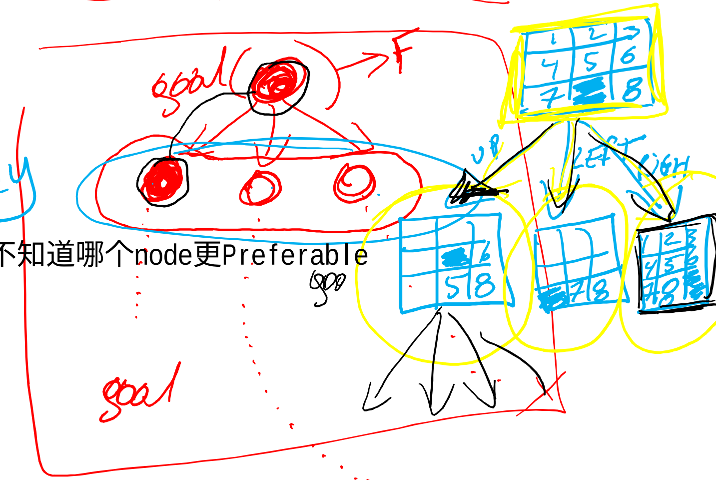
Uninformed VS Informed Search

盲目搜索　　有信息搜索

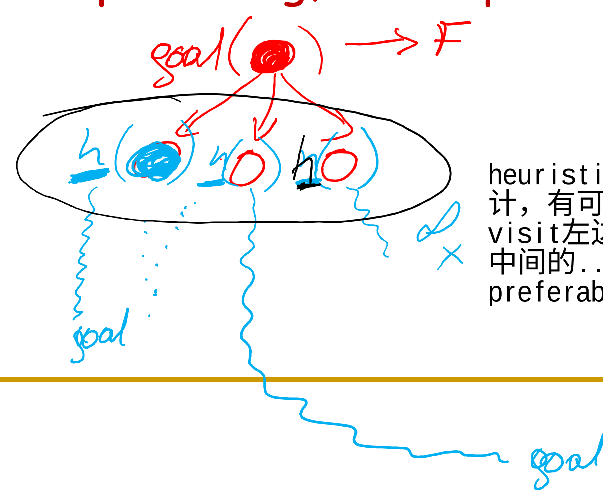
- Uninformed search
  - all nodes are equally promising, so we explore them systematically（系统的）
  - aka: systematic/blind/brute force search
  - many algorithms:
    1. Breadth-first search
    2. Depth-first seach
    3. Uniform-cost search
    4. Depth-limited search
    5. Iterative deepening search
    6. ...

352 video today

他不知道哪个node更Preferable



- Informed search (heuristic search)（探索）
  - we try to identify which nodes *seem* more promising, and explore these first
  - many algorithms:
    1. Hill climbing
    2. Gready Best-First search
    3. Algorithms A and A*
    4. ...



heuristic 只是一个estimate估计，有可能wrong，所以我们先visit左边的，如果不行再visit中间的...但他提供了哪个点更preferable

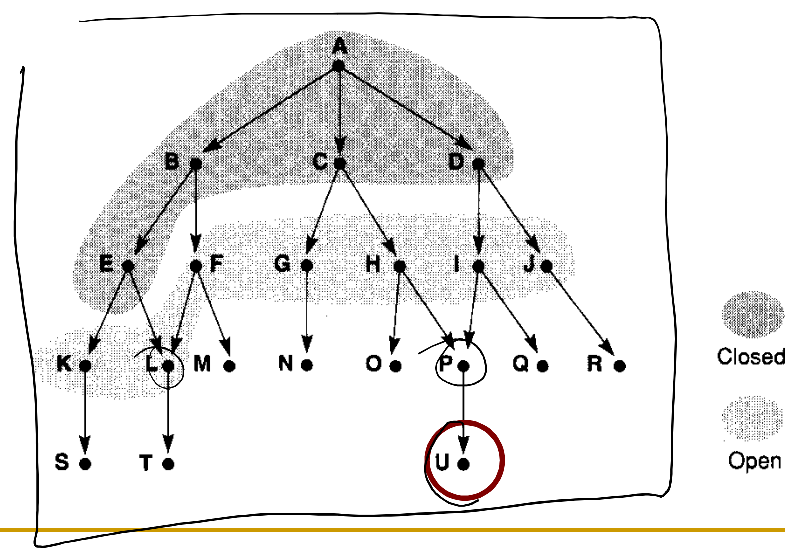
Data Structures

- Most search strategies require:
  - *open list* (aka the frontier) To-DO 也叫做to-do list
    - lists generated nodes not yet expanded
    - order of nodes controls order of search 根据insert顺序决定search顺序
  - *closed list* (aka the explored set)
    - stores all the nodes that have already been visited (to avoid cycles).
- ex: 存储所有visited点来避免cycle

Closed = [A, B, C, D, E]

Open = [F, G, H, I, J, K, L]

Order is important

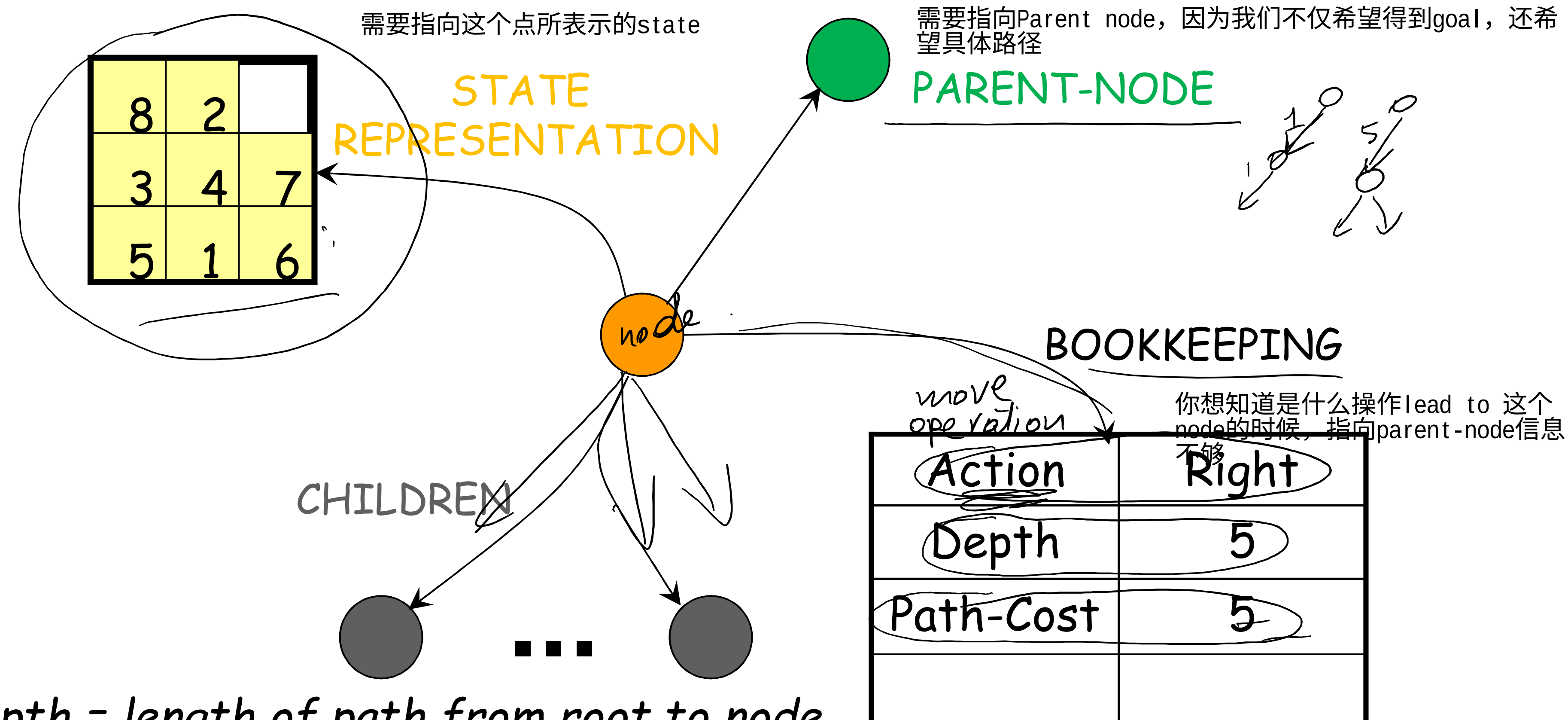


source: G. Luger (2005)

Data Structures

node结构

- state space representation: To trace back the solution path after the search, each node in the lists contain:



Depth = length of path from root to node

robotics.stanford.edu/~latombe/cs121/2003/home.htm

Generic Search Algorithm

1. Initialize the open list with the initial node s_o (top node)

   TO-DO

2. Initialize the closed list to empty

   done / visited

3. Repeat
   a) If the open list is empty, then exit with failure.
   b) Else, take the first node s from the open list.
   c) If s is a goal state, exit with success. Extract the solution path from s to s_o
   d) Else, insert s in the closed list (s has been visited /expanded)
   e) Insert the successors of s in the open list in a certain order if they are not already in the closed and/or open lists (to avoid cycles)

Notes:

- The order of the nodes in the open list depends on the search strategy

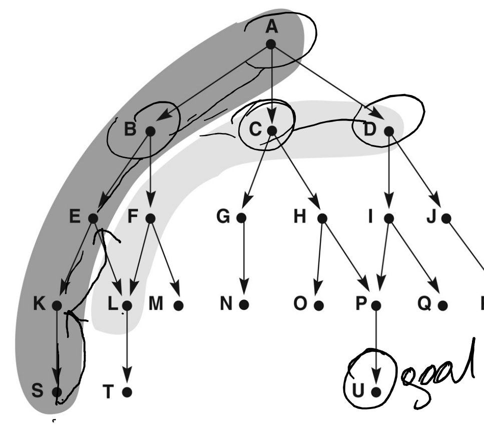
Today

1. State Space Representation
2. State Space Search
   a) Overview
   b) Uninformed search
      1. Breadth-first Search and Depth-first Search 

      2. Depth-limited Search
      3. Iterative Deepening
      4. Uniform Cost
   c) Informed search
      1. Intro to Heuristics
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary

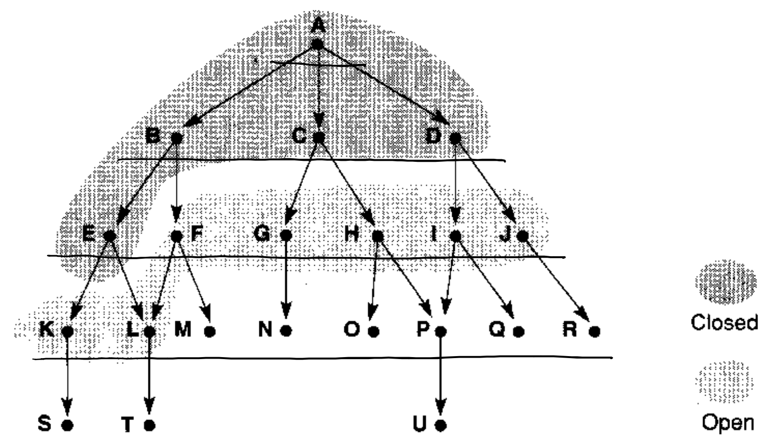
Depth-first vs Breadth-first Search

- Depth-first (DFS):
  - visit successors before siblings
  - Open list is a stack



想象stack就行了，到底了就Pop backtrack

- Breadth-first (BFS):
  - visit siblings before successors
  - ie. visit level-by-level
  - open list is a queue



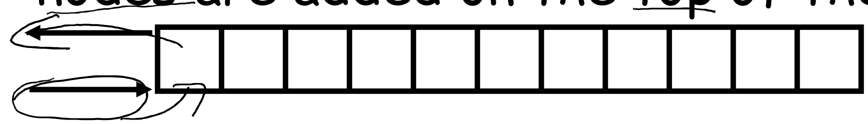
一个level一个level的看

source: G. Luger (2005)

DFS and BFS

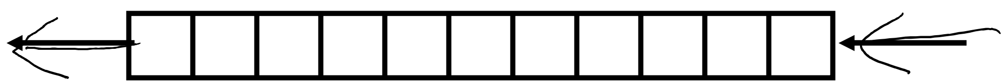
- DFS and BFS differ only in the way they order nodes in the open list:
  - DFS uses a stack:
    - nodes are added on the top of the open list.

open



  - BFS uses a queue:
    - nodes are added at the end of the list.

open



通常决定的是open list，
closed list我们只需要加入然后知道有没有
就行了

Breadth-First Search

```
begin
   open := [Start];                                          % initialize
   closed := [ ];
   while open ≠ [ ] do                                       % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS               % goal found
               else begin
                  generate children of X;
                  put X on closed;
                                                             忽略已经有的children
                  discard children of X if already on open or closed;   % loop check
                  put remaining children on right end of open           % queue
               end                    把剩下的children放在右边
      end
   return FAIL                                               % no states left
end.
```

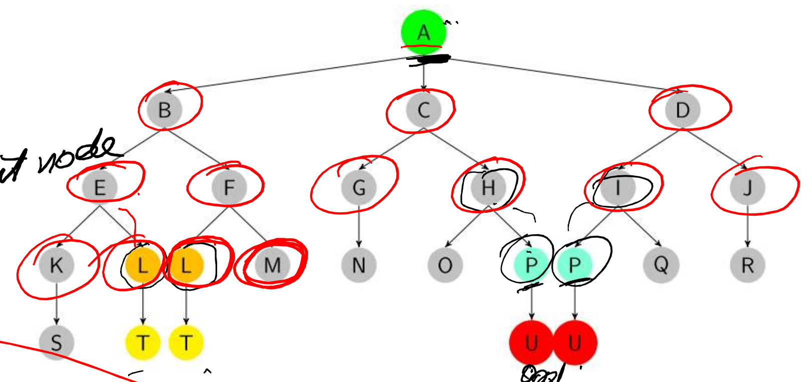
source: G. Luger (2005)

Breadth-First Search Example

- BFS: (open is a queue)

Assume U is goal state

name of node
name of parent node



1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed [A]
3. open = [C-A D-A E-B F-B] closed = [B A]
4. open = [D-A E-B F-B G-C H-C] closed = [C B A]
5. open = [E-B F-B G-C H-C I-D J-D] closed = [D C B A]
6. open = [F-B G-C H-C I-D J-D K-E L-E] closed = [E D C B A]
7. open = [G-C H-C I-D J-D K-E L-E M-F] *as L is already in open* closed = [F E D C B A]
8. and so on until either U is found or open = []

visit => run goal function

search path = A B C D E F G H I J K L M N ... U // closed list
solution path = A C H P U with a cost of 4 // extract by following the parent pointer

source: G. Luger (2005)

Depth-First Search

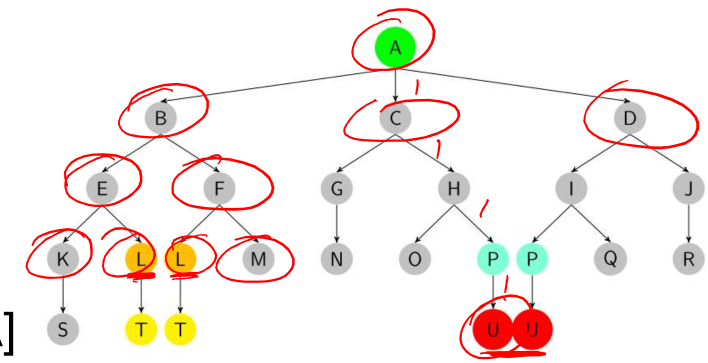
```
begin
   open := [Start];                                                    % initialize
   closed := [ ];
   while open ≠ [ ] do                                                 % states remain
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS                            % goal found
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;    % loop check
               put remaining children on left end of open              % stack
            end
      end;
   return FAIL                                                         % no states left
end.
```

source: G. Luger (2005)

Depth-First Search Example

- DFS: (open is a stack)

Assume U is goal state



1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed [A]
3. open = [E-B F-B C-A D-A] closed = [B A]
4. open = [K-E L-E F-B C-A D-A] closed = [E B A]
5. open = [S-K L-E F-B C-A D-A] closed = [K E B A]
6. open = [L-E F-B C-A D-A] closed = [S K E B A]
7. open = [T-L F-B C-A D-A] closed = [L S K E B A]
8. open = [F-B C-A D-A] closed = [T L S K E B A]
9. open = [M-F C-A D-A] *as L is already on closed* closed = [F T L S K E B A]
10. open = [C-A D-A] closed = [M F T L S K E B A]
11. open = [G-C H-C D-A] closed = [C M F T L S K E B A]

与传统DFS区别在于他把backtrack那步省略了，因为放到closed里去了

search path = A B E K S L .... U // closed list

solution path = A C H P U with a cost of 4

source: *G. Luger (2005)*

Depth-first vs. Breadth-first

- Breadth-first:
  - advantage: optimal, i.e. will always find shortest solution path 总能找到shortest path
  - disadvantage:
    - high memory requirement as we need to keep all states of a level before expanding to the next level
    - exponential space for states required B^n // B=branching factor, n =level

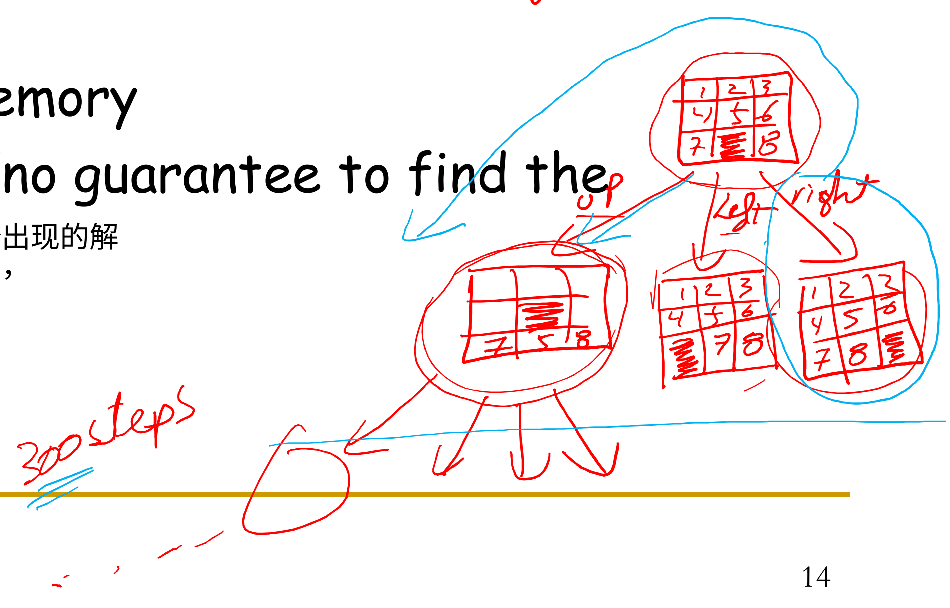
average number of successor

average # of successors

puzzle为例，假设平均successor为3.5(有的时候靠角落就只能两个)
那就是3.5^n

- Depth-first:
  - advantage: Requires less memory
  - disadvantage: Not optimal (no guarantee to find the shortest path)

他找到的是第一个出现的解
而不见得是最优解，
取决于node的顺序



DFS我们会用300STEP的这个，而不是仅用右边

Today

1. State Space Representation
2. State Space Search
   a) Overview
   b) Uninformed search
      1. Breadth-first Search and Depth-first Search
      2. Depth-limited Search
      3. Iterative Deepening
      4. Uniform Cost
   c) Informed search
      1. Intro to Heuristics
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary

Depth-Limited Search

- Compromise for DFS :
  - Do depth-first but
  - with depth cutoff k (depth at which nodes are not expanded)

    深度截止， 到了某一个depth以后就不再展开
- Three possible outcomes:
  - Solution withing your k limit :)
  - Failure (no solution) 实际上无解
  - Cutoff (no solution found within cutoff) ×

    最差的情况，实际解在depth以外
- advantage: memory efficient – it's a DFS
- disadvantage: may not find a solution if it is below the cutoff

Today

1. State Space Representation
2. State Space Search
   a) Overview
   b) Uninformed search
      1. Breadth-first search and Depth-first search
      2. Depth-limited Search
      3. Iterative Deepening

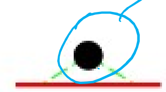
      4. Uniform Cost
   c) Informed search
      1. Intro to Heuristics
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary

Iterative Deepening

- Combination of BFS and DFS:
  - do depth-first search, but
    每次固定一个depth limit，进行DFS，如果没找到，提高Limit,再次从头DFS
  - with a maximum depth before going to next level
  - i.e. Repeats depth first search with gradually increasing depth limit~~s~~
- advantage:
  好处，memory 少，根本上来说还是DFS
  - Requires little memory (fundamentally, it's a depth first)
  - optimal: will find the shortest path (limited depth)
    还能找到最短路径，因为是一层一层遍历的
- disadvantage:
  - repeated traversal of the tree top
    多次重复遍历

Iterative Deepening: Example

Limit = 0

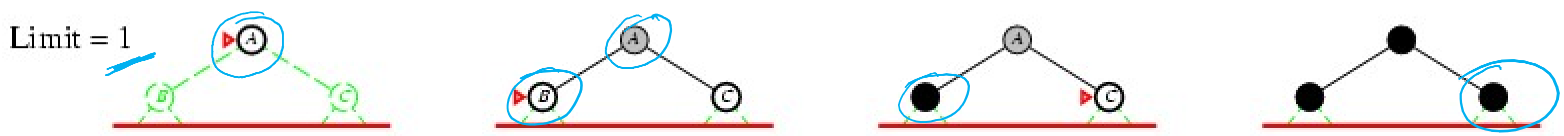


black node
⇒ node has been visited

source: Russel & Norvig (2003)

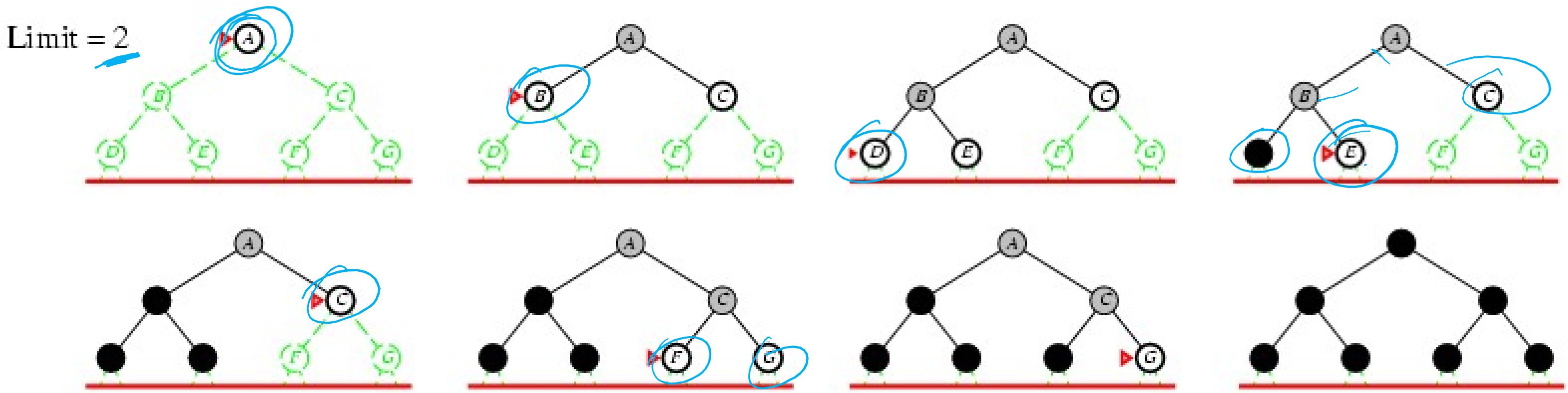
Iterative Deepening: Example

每次都要从头遍历



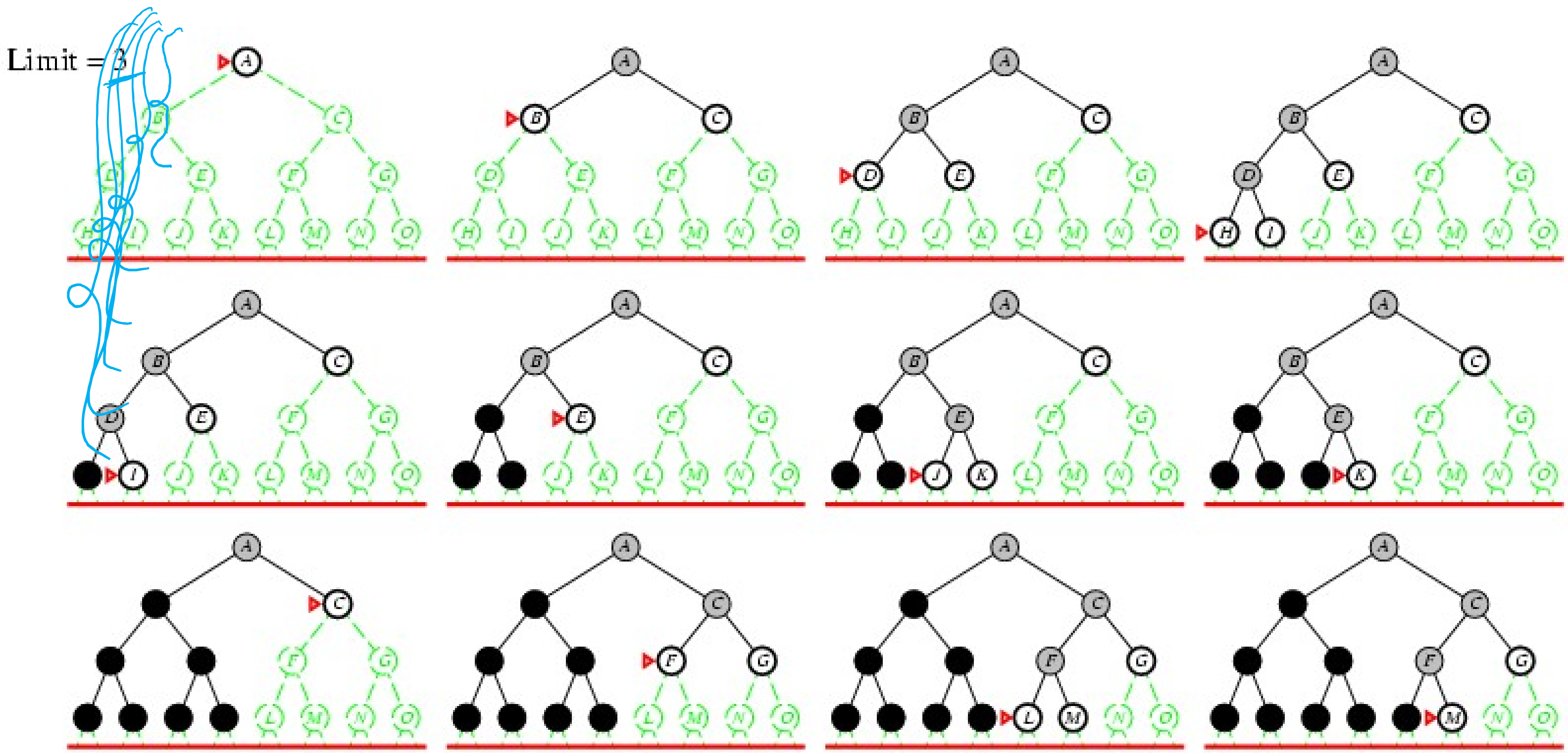
source: Russel & Norvig (2003)

Iterative Deepening: Example



source: *Russel & Norvig (2003)*

Iterative Deepening: Example



source: Russel & Norvig (2003)

Today

1. State Space Representation
2. State Space Search
   a) Overview
   b) Uninformed search
      1. Breadth-first and Depth-first
      2. Depth-limited Search
      3. Iterative Deepening
      4. Uniform Cost

   c) Informed search
      1. Intro to Heuristics
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary

Uniform Cost Search

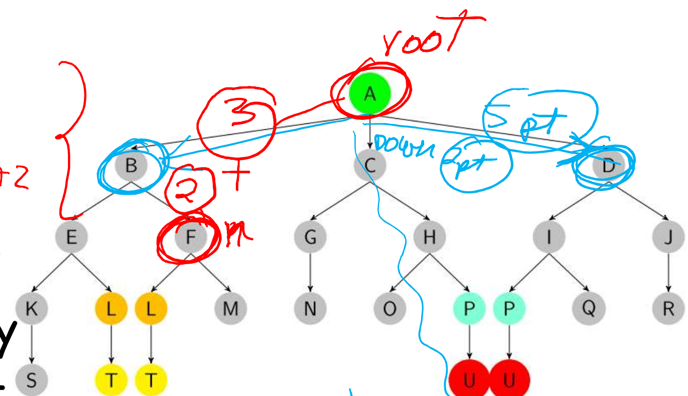
- all algorithms so far assume that all edges have the same cost
- but what if they have different costs ?
  - eg: move UP -> 2pts but move DOWN -> 1 pt
  - eg: cost(residential road) > cost(commercial road)

前三个只能追求最小operation

有的时候edge weight不一样，我们追求最小cost，用uniform cost search

- Breadth ~~Depth~~ First Search
  - uses OPEN as a priority queue sorted by the depth of nodes
  - guarantees to find the **shortest** solution path
- Uniform Cost Search UCS其实有点DFS那个味儿的
  - takes the cost of the edge into account
  - uses OPEN as a priority queue sorted by the total cost from the root to node n – *later we will call this g(n)* root到n的所有cost
  - guarantees to find the **lowest cost** solution path

g(n) 3+2



lowest cost of the solution path

Example

node
parent node
Total cost from root to n

open	closed
$S_{null\,0}$	$\emptyset$
$\{P_{S1}\ E_{S9}\ D_{S3}$ $\{P_{S1}\ D_{S3}\ E_{S9}$	$S_{null\,0}$
$\{Q_{P16}\ D_{S3}\ E_{S9}$ $\{D_{S3}\ \cancel{E_{S9}}\ Q_{P16}$ 排序	$S_{null\,0}\ P_{S1}$
$\{B_{D4}\ C_{D11}\ E_{D5}\ Q_{P16}$ $\{B_{D4}\ E_{D5}\ C_{D11}\ Q_{P16}$	$S_{null\,0}\ P_{S1}\ D_{S3}$
$\{A_{B6}\ E_{D5}\ CD_{11}\ Q_{P16}$ $\{E_{D5}\ A_{B6}\ CD_{11}\ Q_{P16}$	$S_{null\,0}\ P_{S1}\ D_{S3}\ B_{D4}$
$H_{E6}\ R_{E14}\ A_{B6}\ CD_{11}\ QP_{16}$ $HE_6\ AB_6\ CD_{11}\ RE_{14}\ QP_{16}$	ED_5

主要区别在于会sort

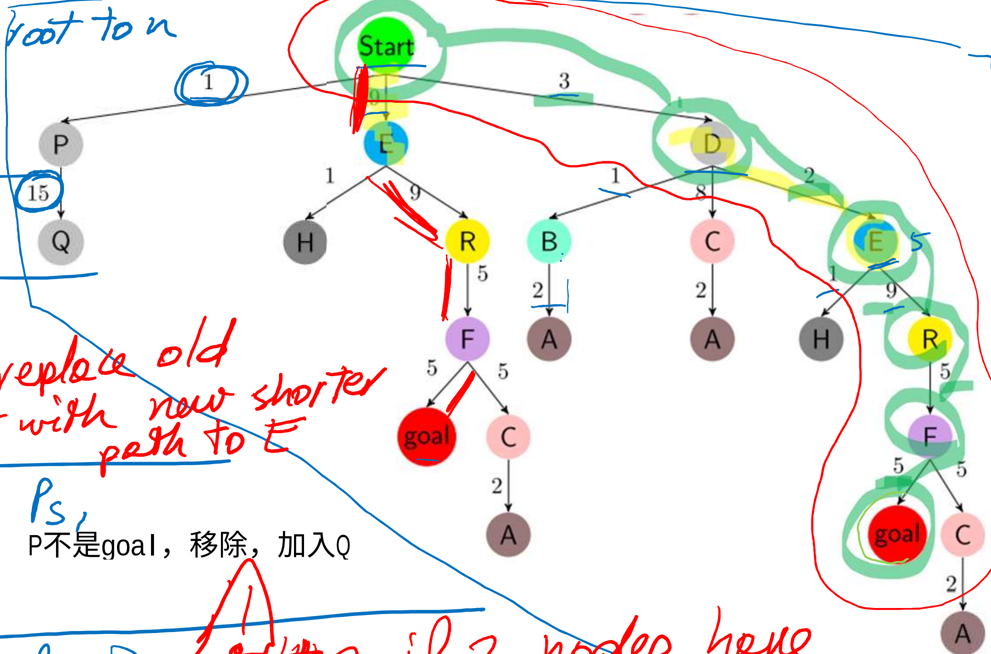
#1 replace old E with new shorter path to E

P不是goal，移除，加入Q

ED5小于ES9，替代了ES9

#2 if 2 nodes have the same cost in the open list
⇒ arbitrary choice

如果有相同cost，任意



I skipped the last steps

solution path : Goal F R E D S cost of 24

最后找到goal

Today

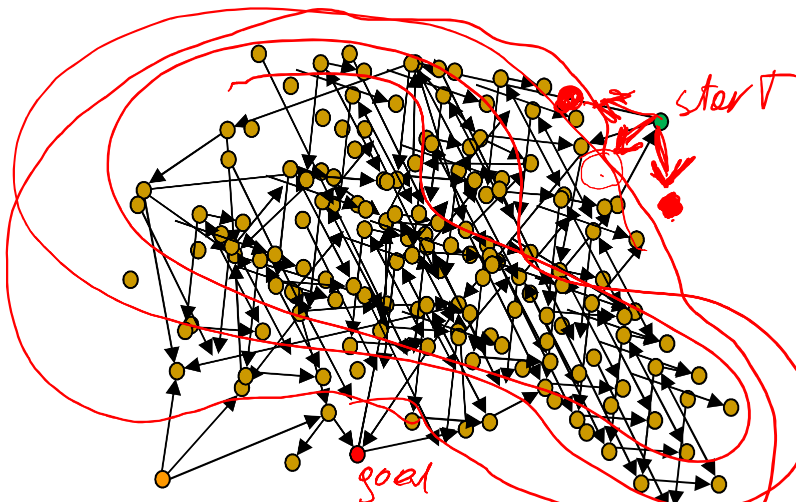
1. State Space Representation
2. State Space Search
   - a) Overview
   - b) Uninformed search
     1. Breadth-first and Depth-first
     2. Depth-limited Search
     3. Iterative Deepening
     4. Uniform Cost

     352
   - c) Informed search
     1. Intro to Heuristics
     2. Hill climbing
     3. Best-First
     4. Algorithms A & A*
     5. More on Heuristics
   - d) Summary

Problem with Uninformed Search

AI问题的state space太大了

- inefficient for most AI problems, the state space is too large!
  - e.g. state space of all possible moves in chess = 10^{120}
  - 10^{75} = nb of molecules in the universe
  - 10^{26} = nb of nanoseconds since the "big bang"



- we need a way to try the most promising nodes first

Up Next

1. State Space Representation
2. State Space Search
   a) Overview
   b) Uninformed search
      1. Breadth-first and Depth-first
      2. Depth-limited Search
      3. Iterative Deepening
      4. Uniform Cost
   c) **Informed search**
      1. **Intro to Heuristics** $h(n)$
      2. Hill climbing
      3. Best-First
      4. Algorithms A & A*
      5. More on Heuristics
   d) Summary