人工知能入門
第10回

藤田 悟

黄　潤和

前回の復習

- 探索プログラムの作成
  - 深さ優先探索
    - 再帰を使ってプログラミングする
  - 幅優先探索
    - Queueを使ってプログラミングする

今回学ぶこと

- 幅優先探索と深さ優先探索の比較
  - 2つの探索を書き換える
  - 記憶が必要なオープンリストの数
  - 解の品質
- 繰り返し深化(Iterative Deeping)
- 分枝限定法

深さ優先/幅優先探索プログラム

探索プログラムの書き換え

- 再帰から whileへの書き換え
  - 前回の幅優先探索プログラムは、while を用いて書き換えることができる。
    - 深い再帰をしないことで、実行速度やメモリ消費量的にも有利
- 幅優先探索プログラムから深さ優先探索プログラムへの書き換え
  - Queue の利用を Stack に書き換える

再帰を使った幅優先探索プログラムの擬似コード

```
void start() {
        初期化();
        search();
}
boolean search() {
        state = poll(); // Queue から状態を1個取り出す
        if(state == null) {
                return false; // Queue が空になったので終了
        }
        // 状態の下に存在する新たな状態を生成する
        for(int i = 0; i < width; i++) {
                新状態 = update(状態, i);
                if(中間状態判定(新状態)) { // 中間状態が矛盾ない場合だけ、探索を深く行う
                        if(state.depth == n) {
                                return true; // 深さnの解が見つかったので、終了
                        } else {
                                offer(新状態); // 新状態を Queue に詰め込む
                        }
                }
        }
        return search(); // 下位のノードを全て作り終わったところで、search()を再帰呼び出しする。
}
```

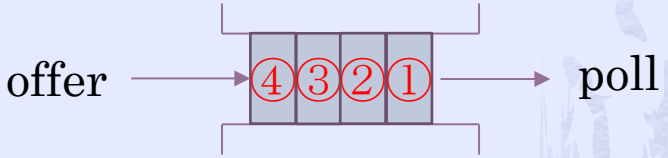
while() を使った幅優先探索プログラムの擬似コード

```
void start() {
        初期化();
        search();
}
boolean search() {
        while(true) {
                state = poll(); // Queue から状態を1個取り出す
                if(state == null) {
                        return false; // Queue が空になったので終了
                }
                // 状態の下に存在する新たな状態を生成する
                for(int i = 0; i < width; i++) {
                        新状態 = update(状態, i);
                        if(中間状態判定(新状態)) { // 中間状態が矛盾ない場合だけ、探索を深く行う
                                if(state.depth == n) {
                                        return true; // 深さnの解が見つかったので、終了
                                } else {
                                        offer(新状態); // 新状態を Queue に詰め込む
                                }
                        }
                }
        }
}
```

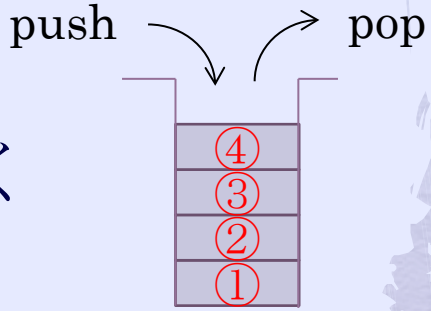
幅優先探索プログラムを深さ優先探索プログラムに書き換える

```
void start() {
        初期化();
        search();
}
boolean search() {
        while(true) {
                state = pop(); // Stack から状態を1個取り出す
                if(state == null) {
                        return false; // Queue が空になったので終了
                }
                // 状態の下に存在する新たな状態を生成する
                for(int i = 0; i < width; i++) {
                        新状態 = update(状態, i);
                        if(中間状態判定(新状態)) { // 中間状態が矛盾ない場合だけ、探索を深く行う
                                if(state.depth == n) {
                                        return true; // 深さnの解が見つかったので、終了
                                } else {
                                        push(新状態); // 新状態を Stack に詰め込む
                                }
                        }
                }
        }
}
```

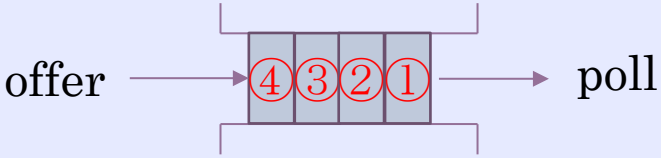
Queue と Stack

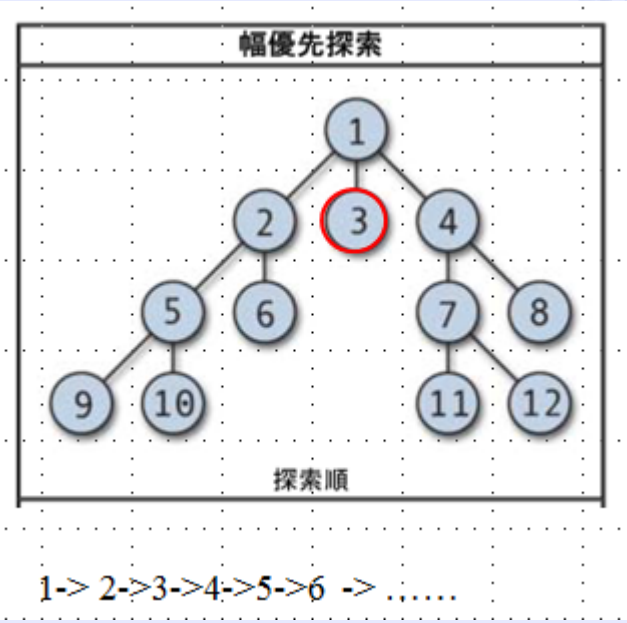


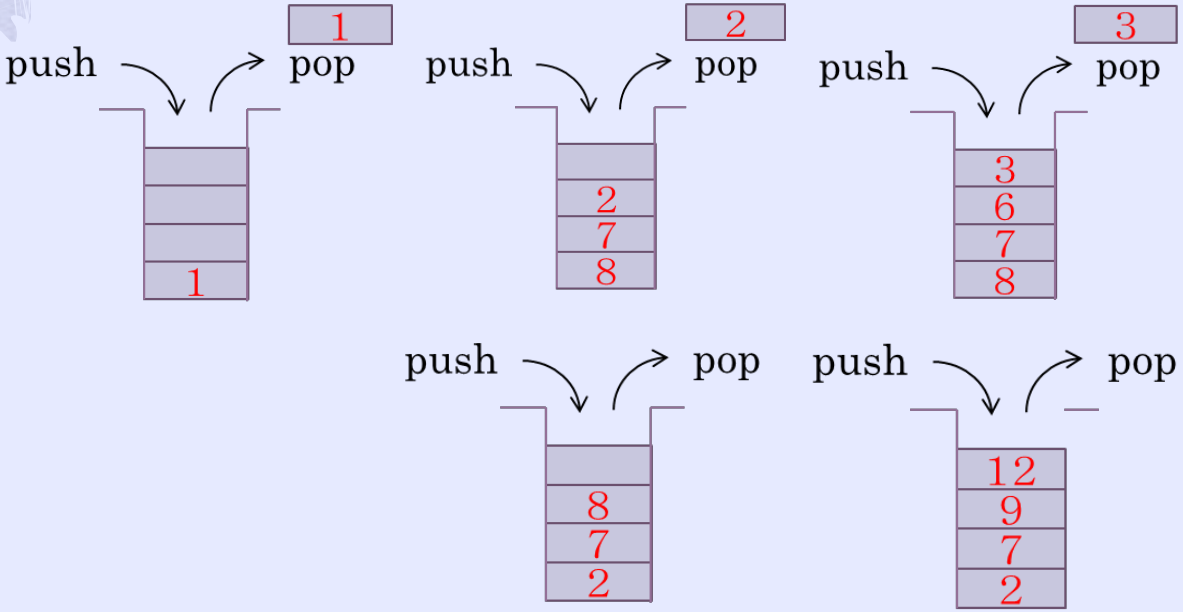
- Queue
  - 初めに積んだデータから読みだしていく
  - First In First Out
  - 初めに積んだ depth が浅いデータから順にオープンリストに格納して、下位ノードを探索する

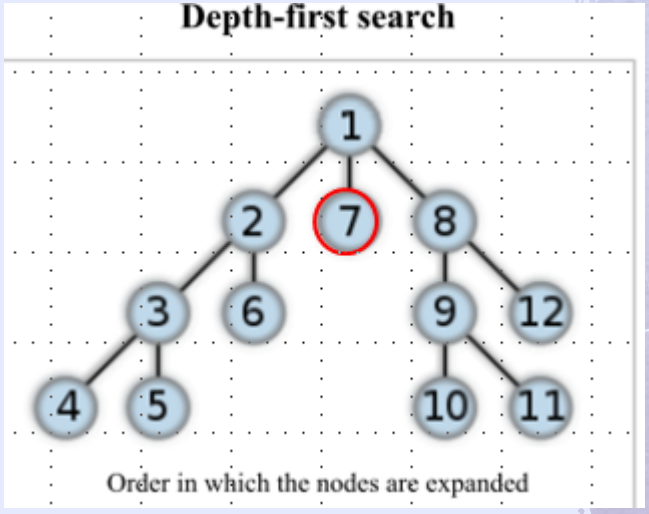


- Stack
  - 最後に積んだデータから読みだしていく
  - Last In First Out
  - 最後に積んだ depth が深いノードから順にオープンリストに格納して、下位ノードを探索する。


offer
4
3
2
1
poll

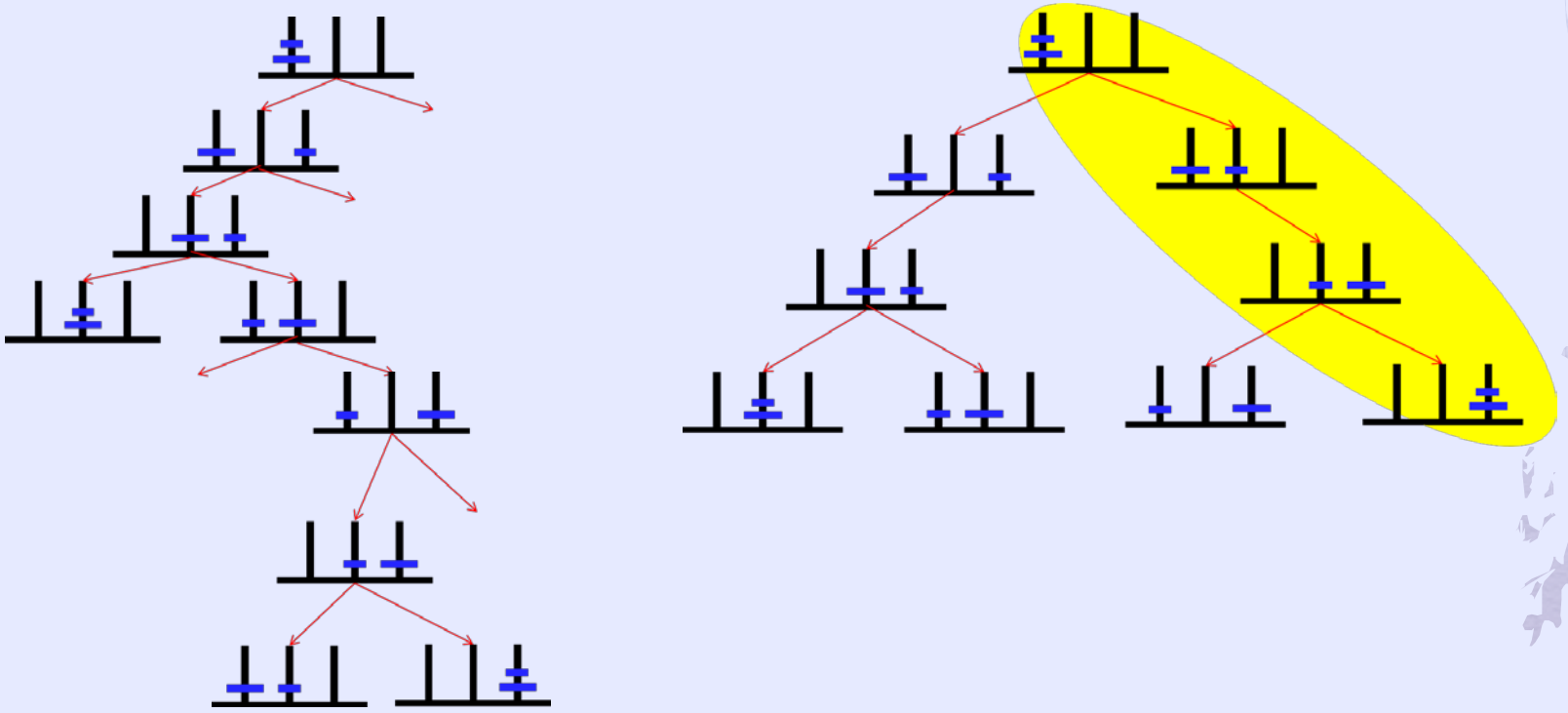

幅優先探索
1
2
3
4
5
6
7
8
9
10
11
12
探索順
1-> 2->3->4->5->6 -> .....


push
1
pop
1
push
2
pop
2
7
8
push
3
pop
3
6
7
8
push
pop
8
7
2
push
pop
12
9
7
2


Depth-first search
1
2
7
8
3
6
9
12
4
5
10
11
Order in which the nodes are expanded

深さ優先はパスが長くなるかも…

- nQueen は、いつでも深さnに解が存在する
- ハノイの塔は、深さがいくつになるか運次第
  - 深さ優先で探索すると、深さ6
  - 幅優先で探索すると深さ3



深さ優先だと永久に解が見つからないこともある

- 逆方向を探索し始めたら、永久に法政大学にたどり着けない

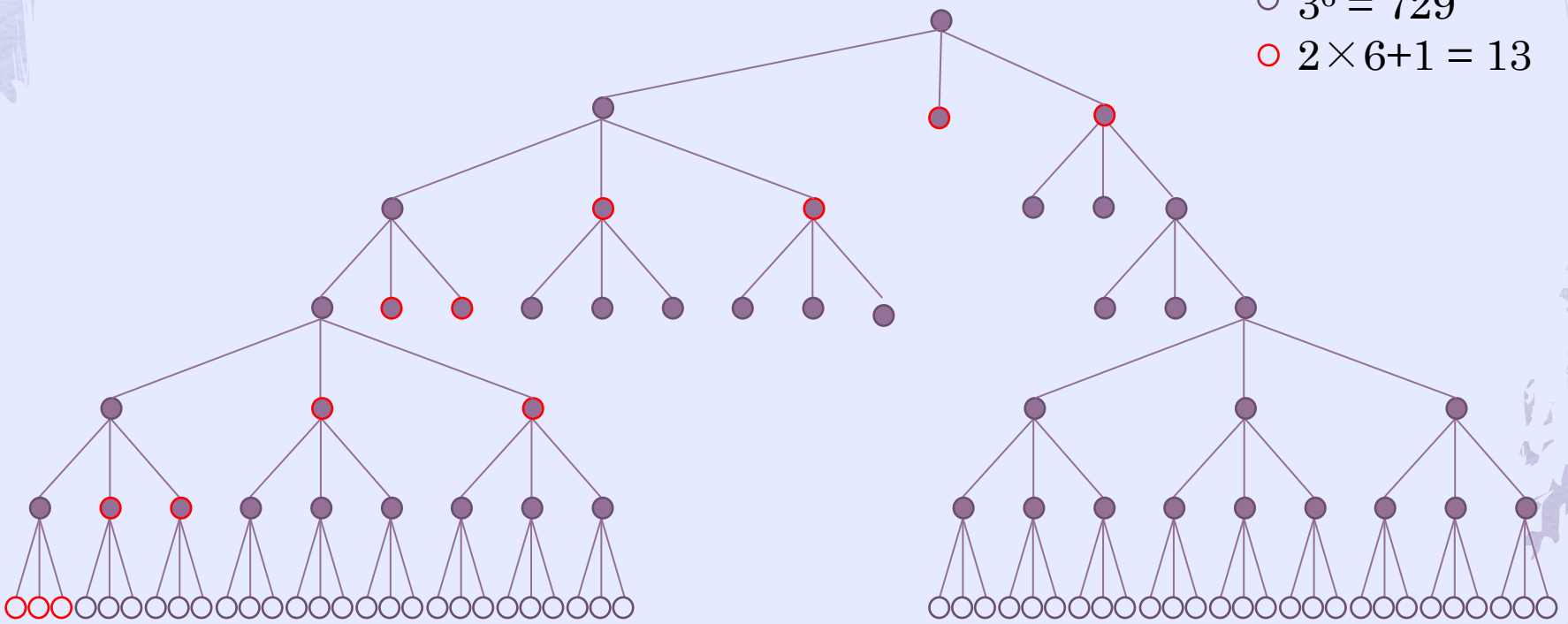
幅優先なら、解にたどり着ける

- 分岐ごとに幅優先で地図探索を行う
  - しかし…

- オープンリスト中のノード

オープンリスト数の爆発

- 分岐数b, 深さmとすると、
  - 幅優先のオープンリスト数 $= b^m$
  - 深さ優先のオープンリスト数 $= (b-1) \times m + 1$



(演習1) nQueenのQueueの大きさ

- n-Queen を幅優先探索するプログラムについて、Queueに利用する配列 (states)の大きさを100にすると、n-Queen の nはいくつまで探索可能か
- 1000の時, 10000の時に探索可能な nはいくつか

繰り返し深化(ITERATIVE DEEPING)

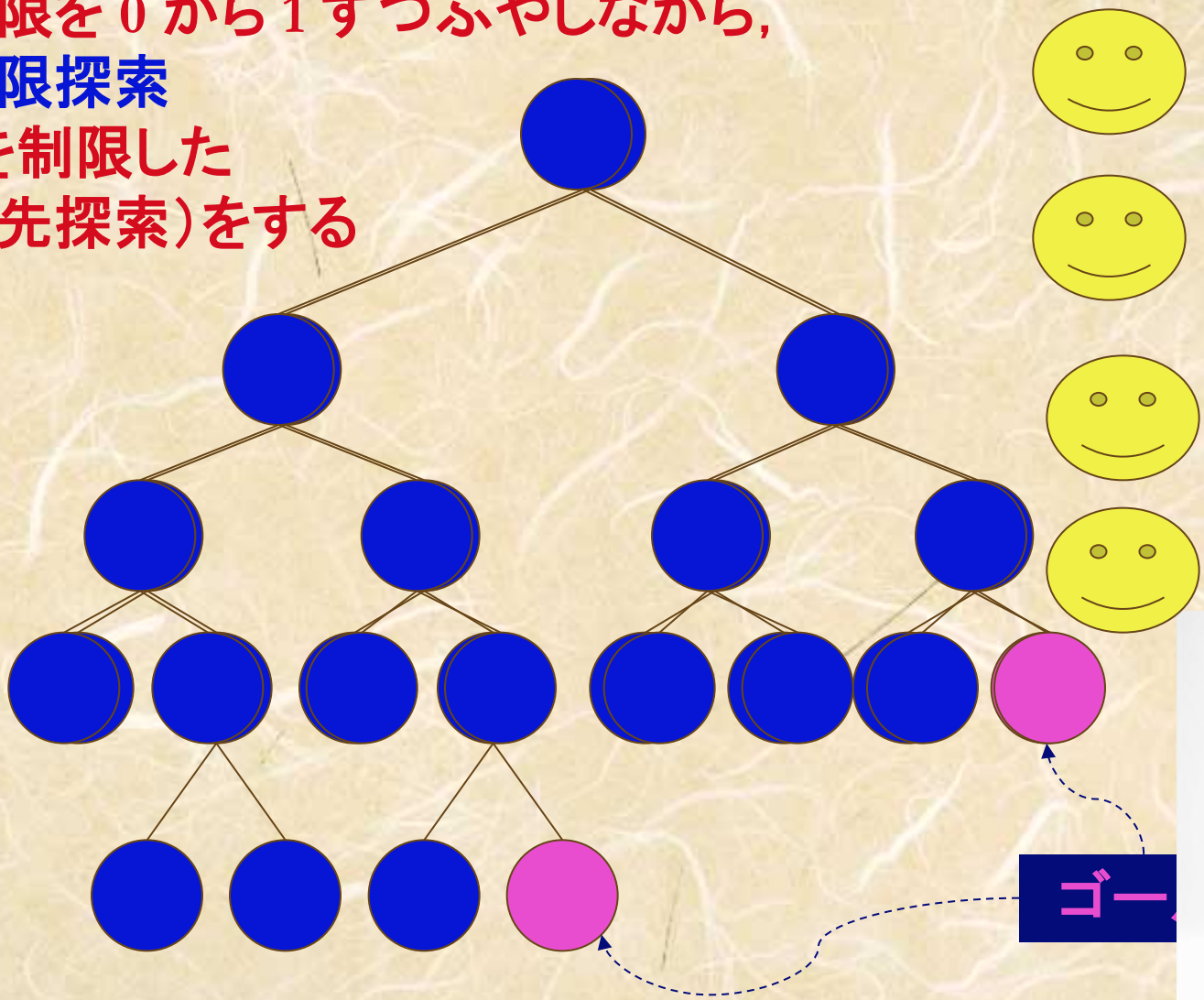
Iterative Deeping
反復深化深さ優先探索

- 解の深さが不均質で、かつ、問題サイズが大きくて、幅優先探索ができないとき、Iterative Deeping というアルゴリズムが有効

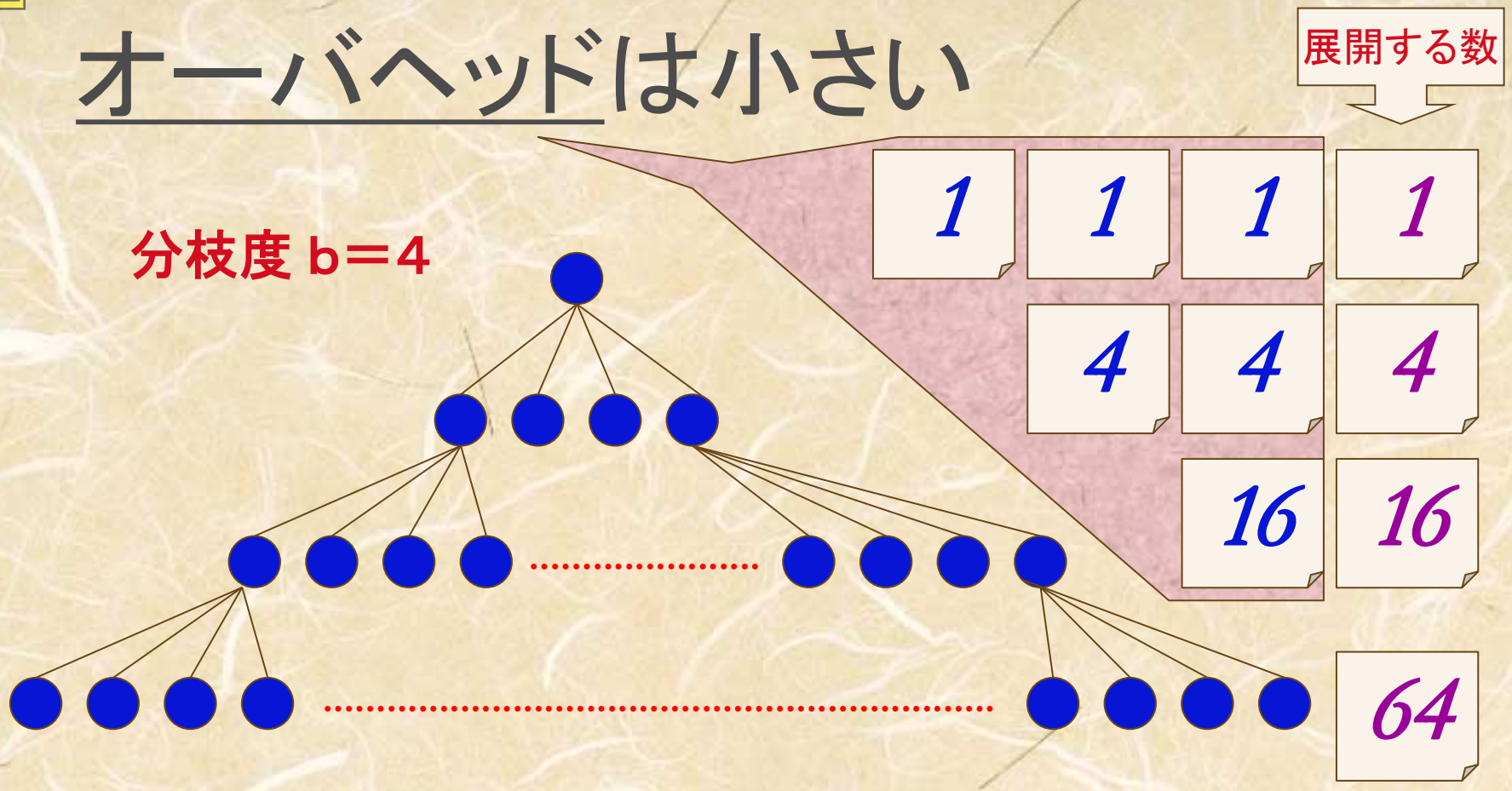
```
depth = 1
while() {
    depth までの深さ優先探索を行う
    if(解が見つかる) break;
    depth++;
}
```

反復深化探索(iterative deepening) - demo

深さ制限を 0 から 1 ずつふやしながら,
深さ制限探索
(深さを制限した
深さ優先探索)をする



オーバヘッドは小さい



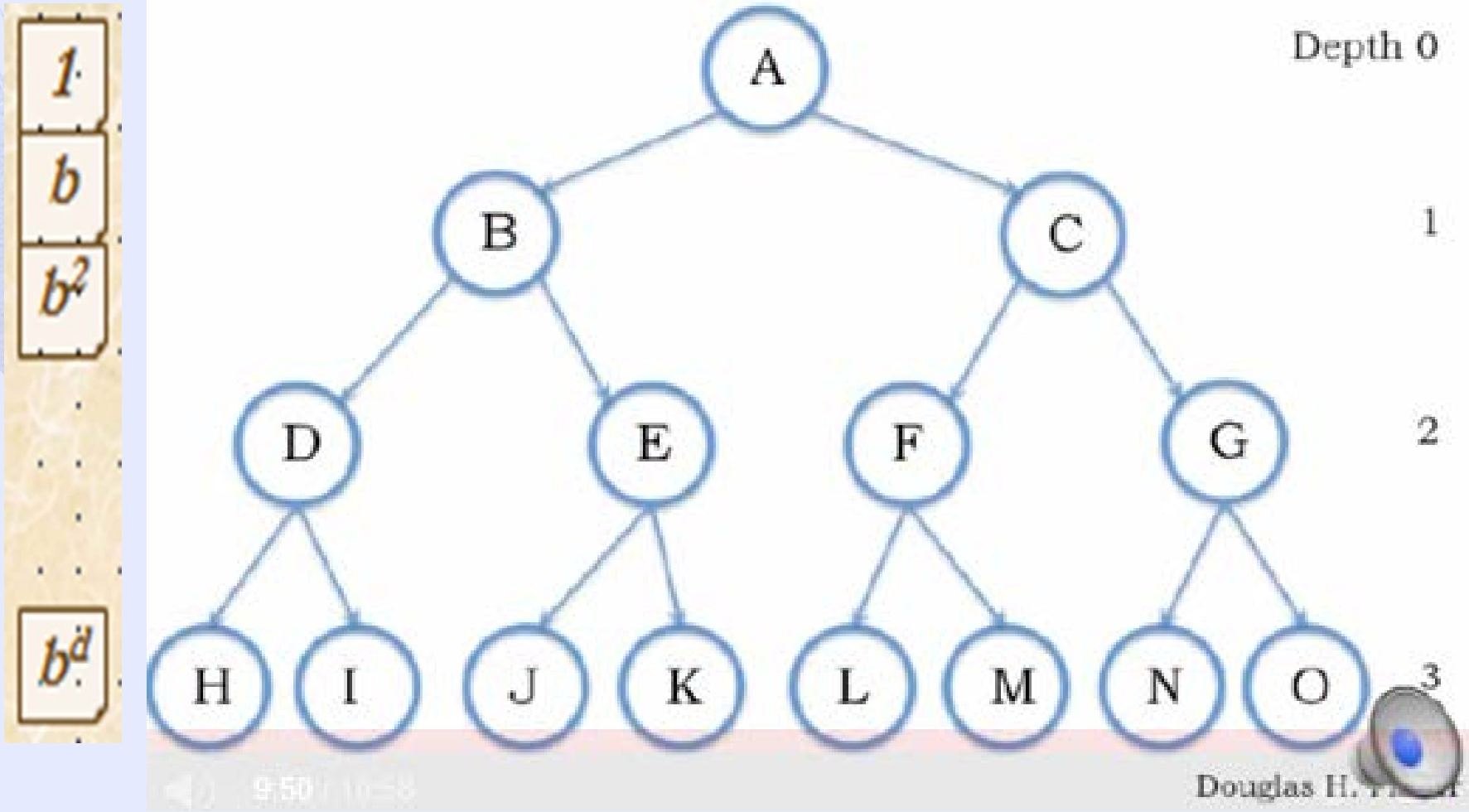
$$\text{オーバヘッド（比）} = \frac{27}{85} \approx 0.32$$

固定された b に対して，d がじゅうぶん
大きいとき，この比は

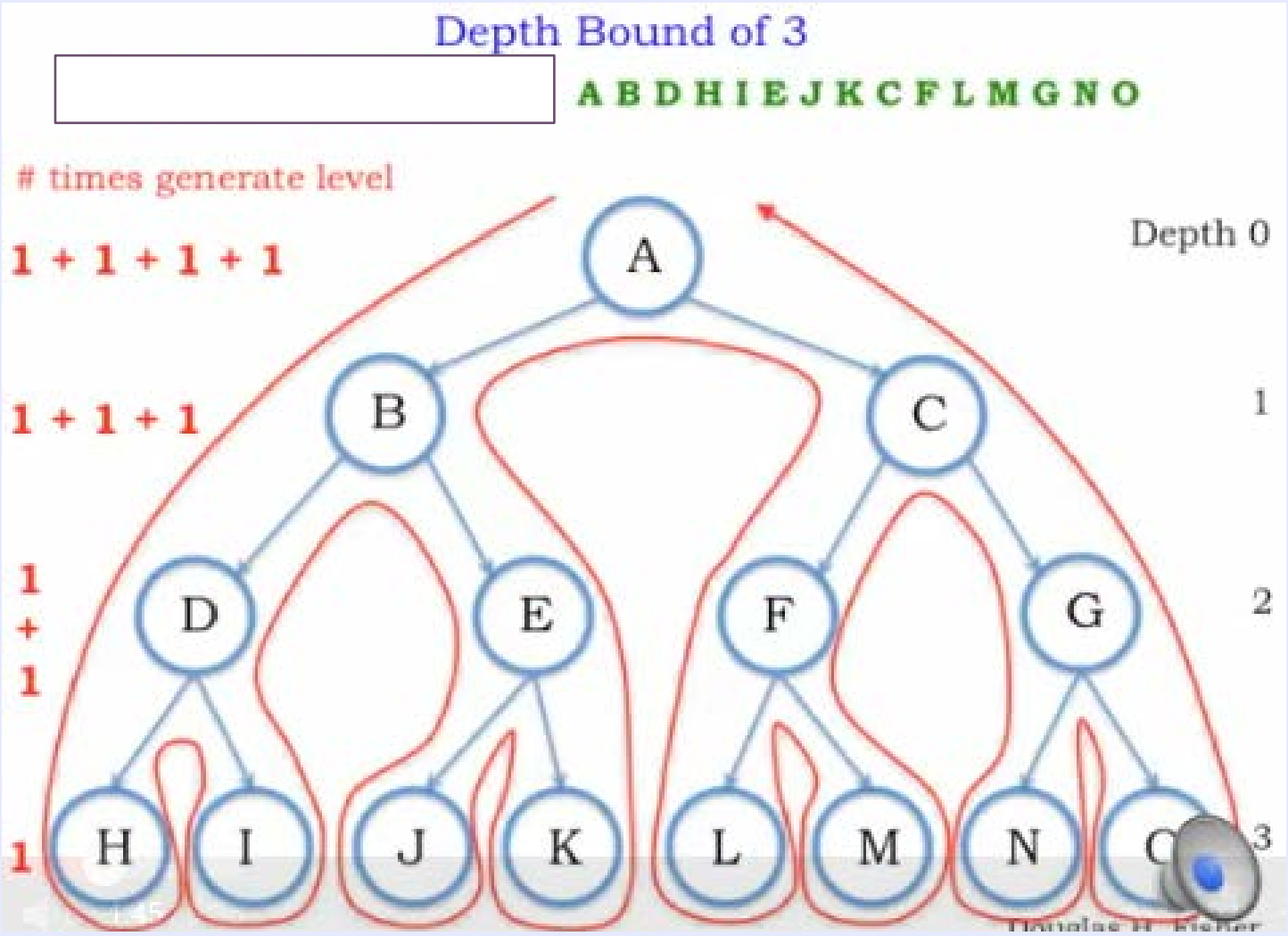
$$\frac{1}{b-1}$$

Over ALL the iterations, from depth bound 0 to 1 the order in which nodes removed from the frontier is:

A A B C



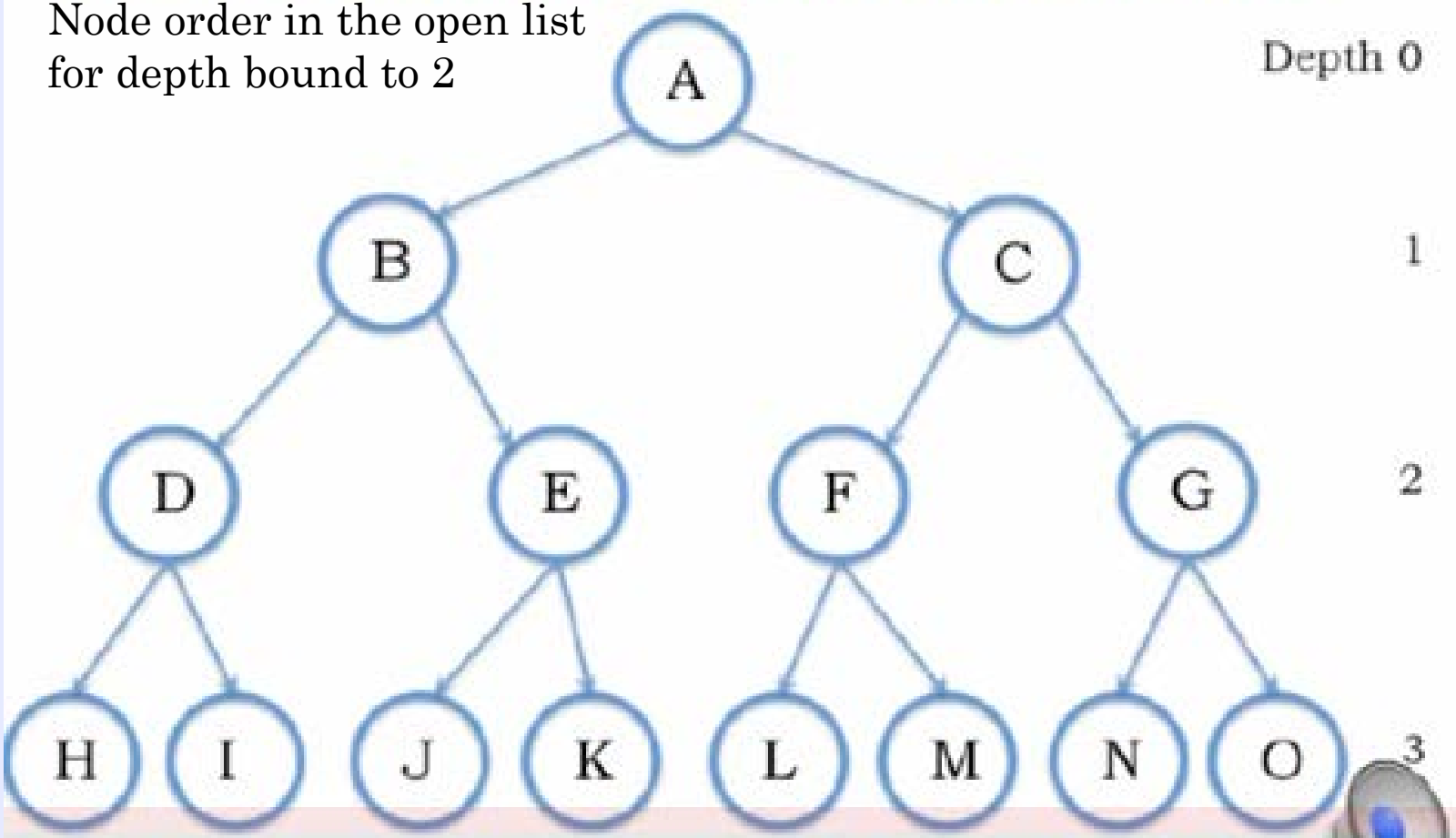
Douglas Fisher


Depth Bound of 3
A B D H I E J K C F L M G N O
times generate level
1 + 1 + 1 + 1
1 + 1 + 1
1 + 1
1
Depth 0
1
2
3
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
1
b
b2
bd

Over ALL the iterations, from depth bound 0 to 3, the order in which nodes removed from the frontier is:

A A B C ? A B D H I E J K C F L M G N O

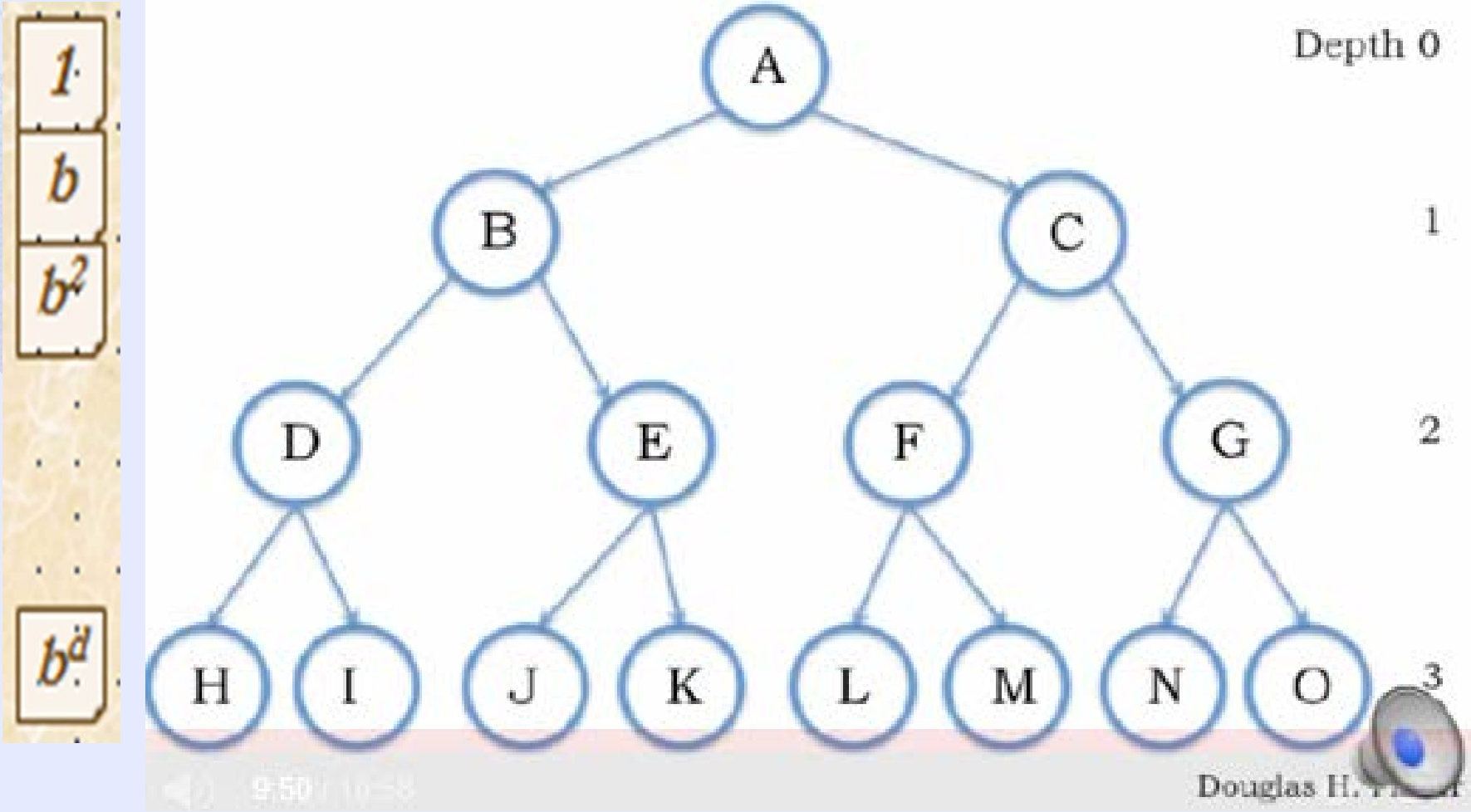
Node order in the open list for depth bound to 2



Douglas Fisher

Over ALL the iterations, from depth bound 0 to 1 the order in which nodes removed from the frontier is:

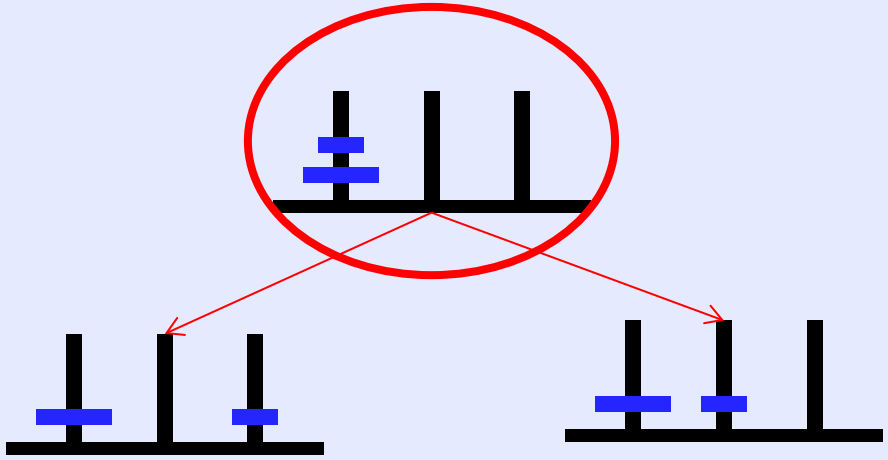
A A B C A B D E C F G A B D H I E J K C F L M G N O



Douglas Fisher

ハノイの塔だと…

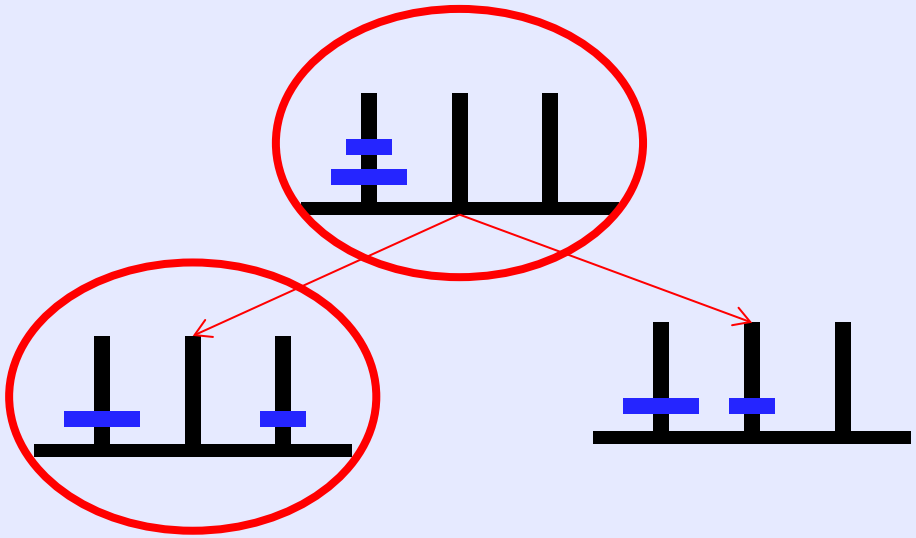
- ハノイの塔は、深さ3の探索で解を見つける



Depth bound 0

ハノイの塔だと…

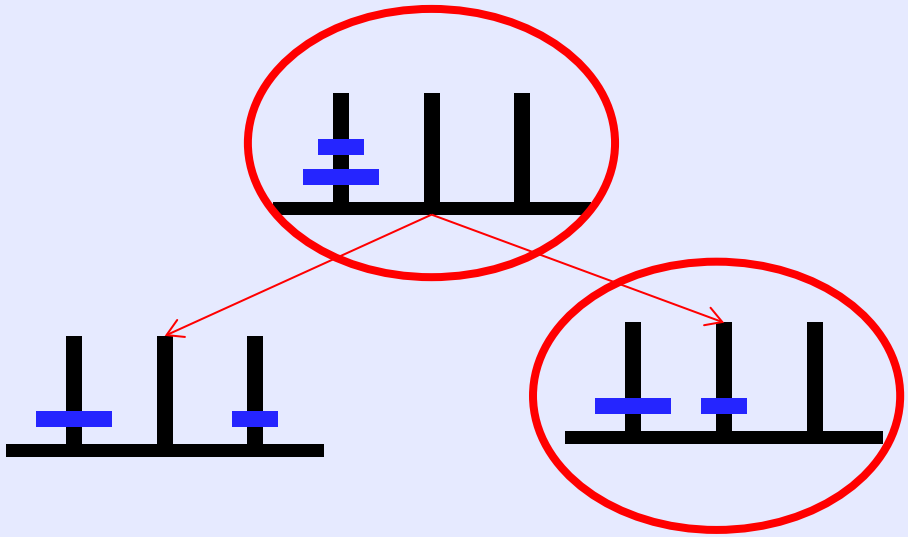
- ハノイの塔は、深さ3の探索で解を見つける



Depth bound 1

ハノイの塔だと…

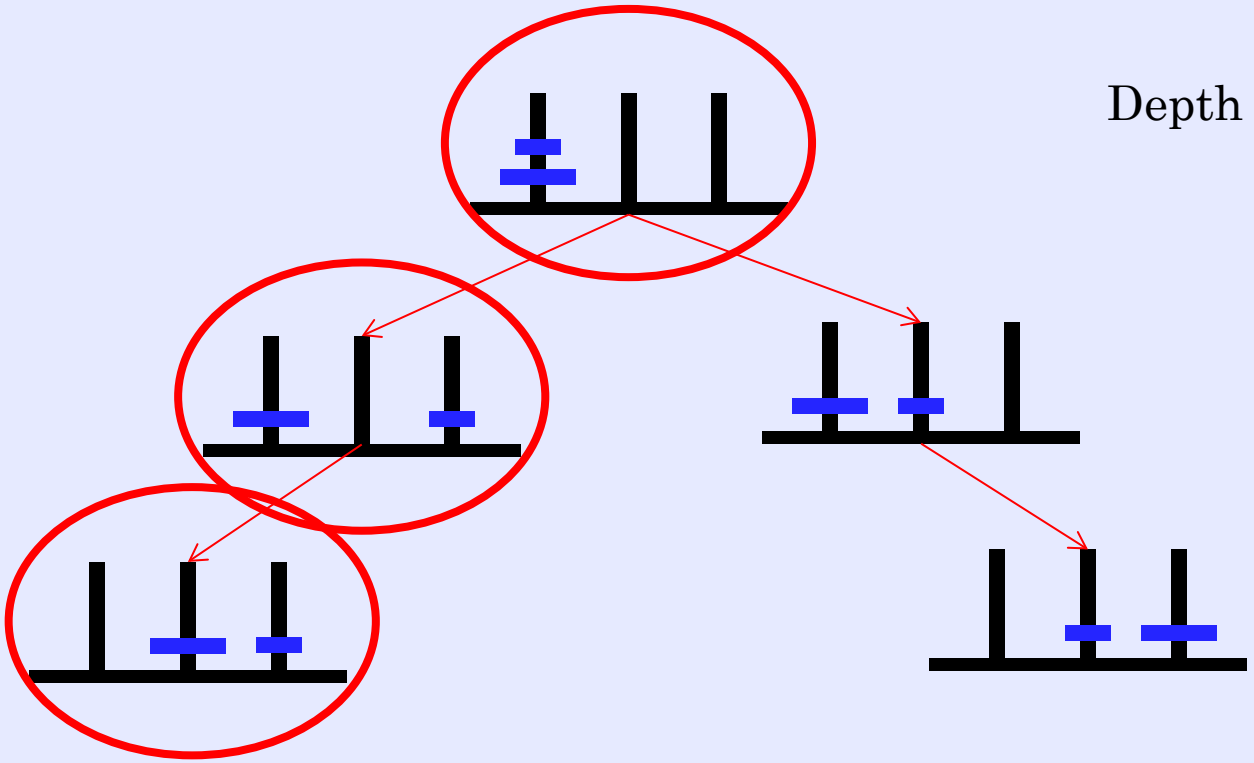
- ハノイの塔は、深さ3の探索で解を見つける



Depth bound 1

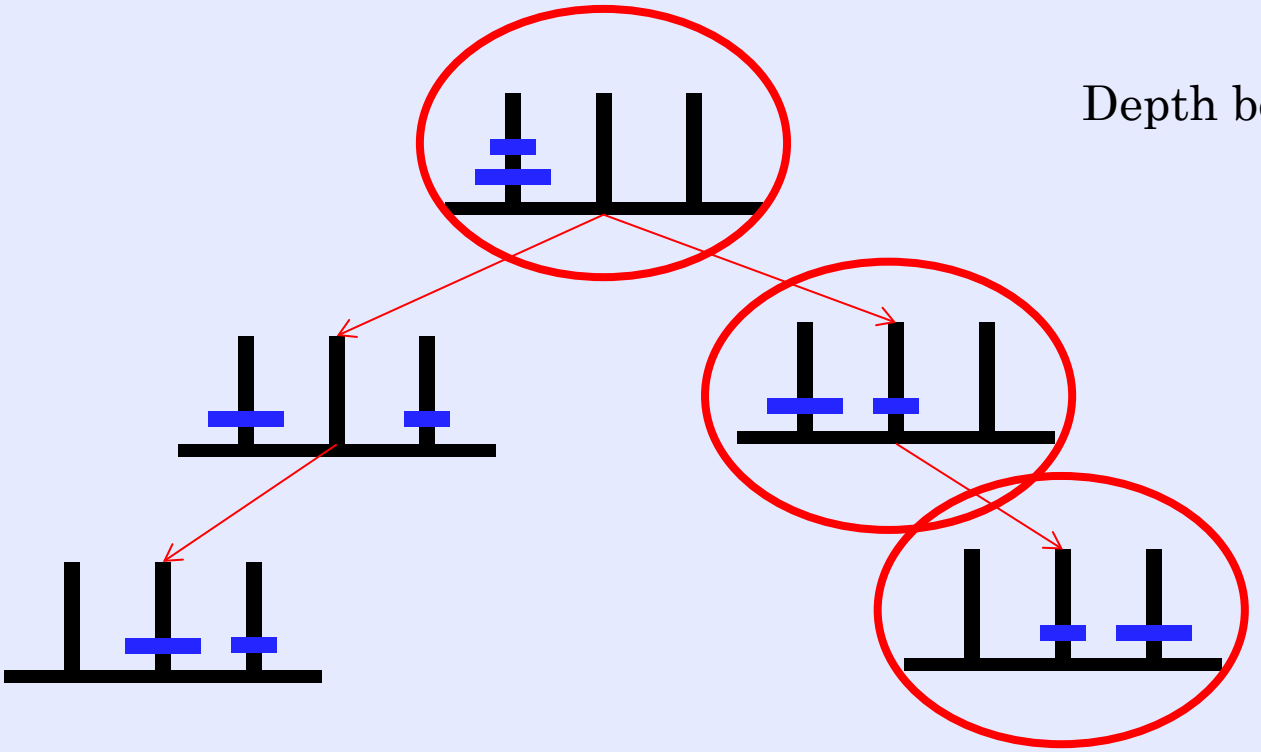
ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



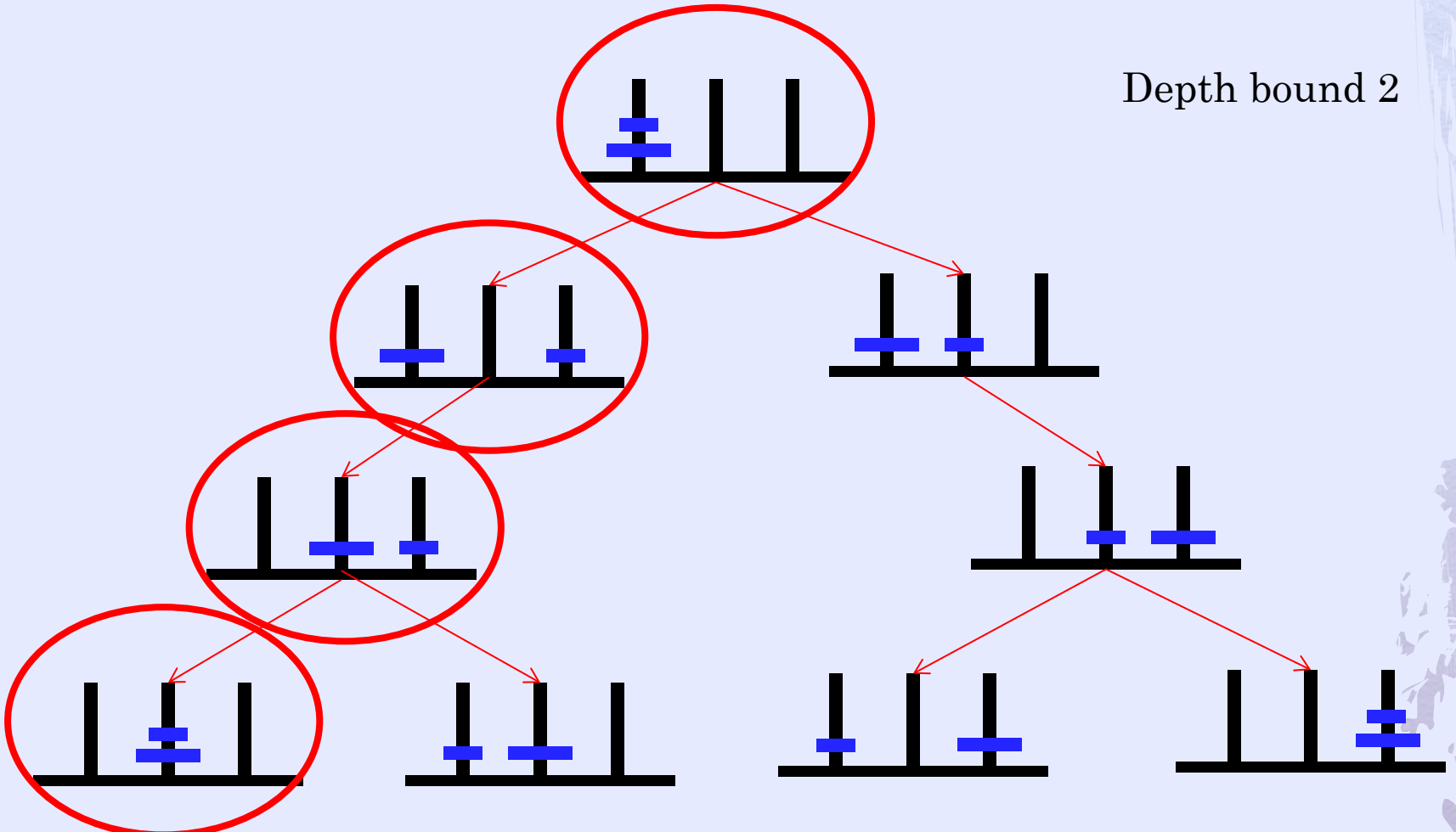
ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



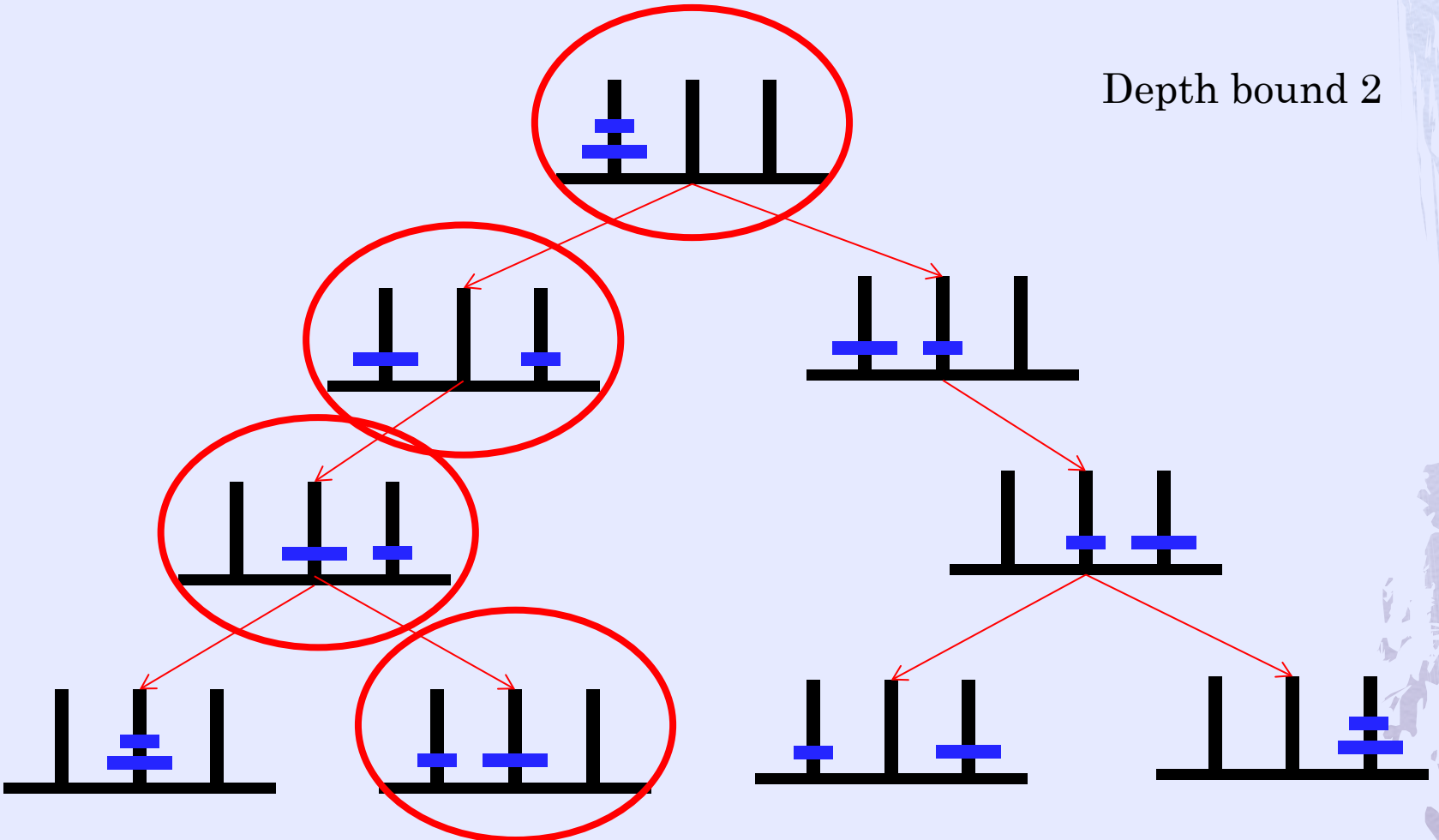
ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



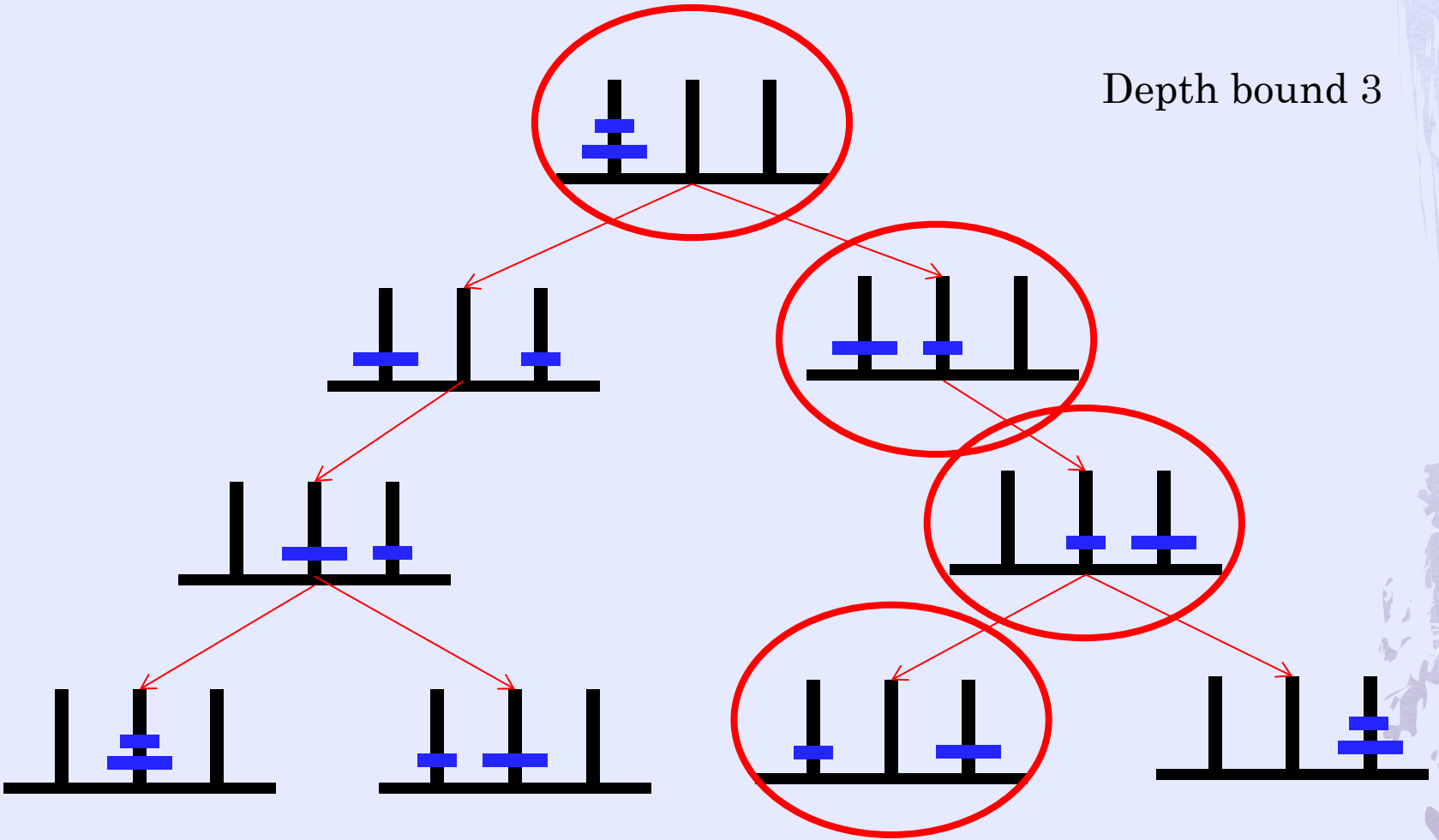
ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



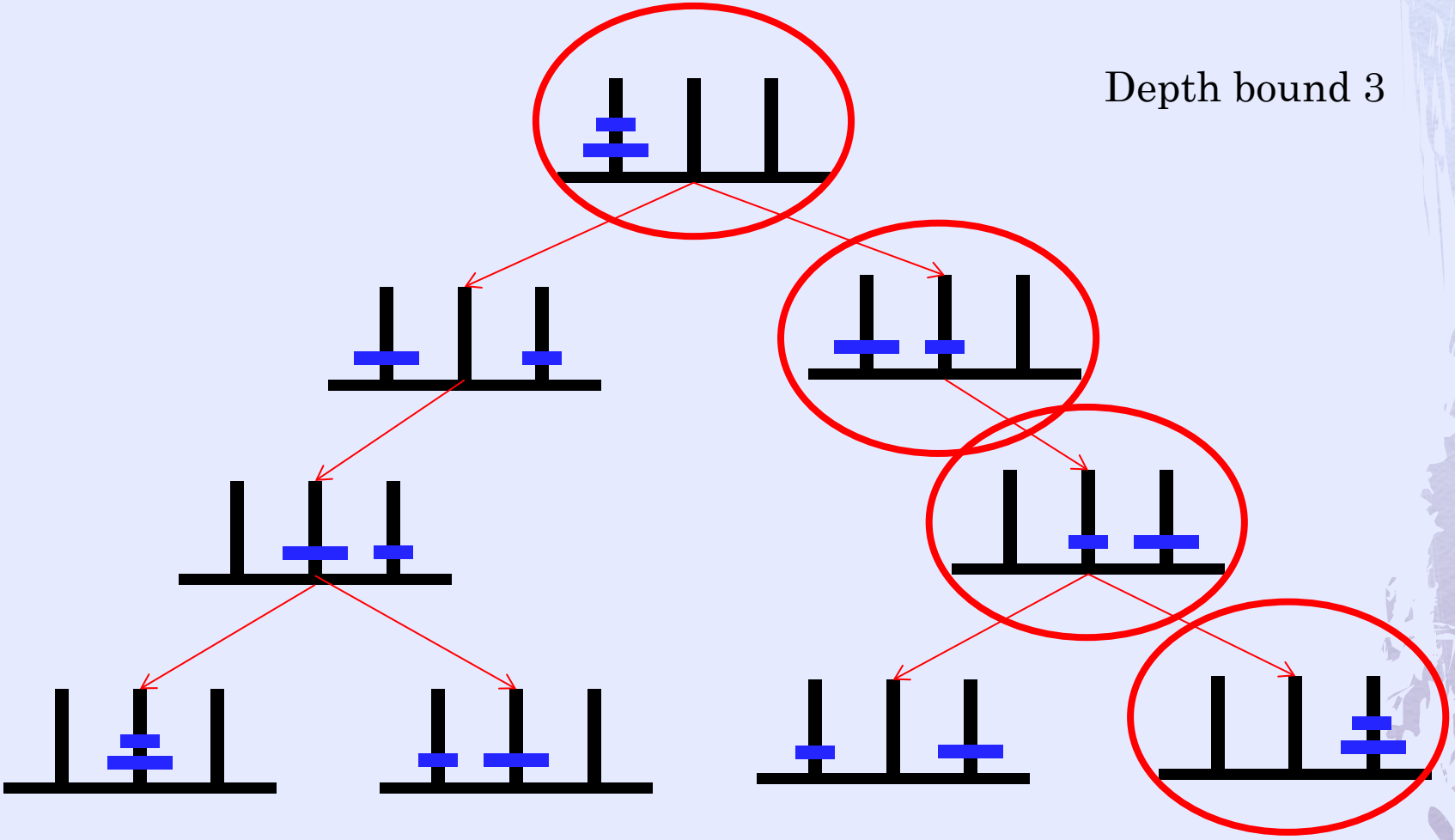
ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



ハノイの塔だと…

- ハノイの塔は、深さ3の探索で解を見つける



無駄な検索が多い？

- 中間的なオープンリストを記憶していないために、無駄な探索が多い
  - depth = 1 で 2回
  - depth = 2 で 4回
  - depth = 3 で 8回
- depth = 3 だけ探索したら、8回で済む
- Iterative deeping を利用すると、14回かかる。
  - しかし、14/8 = 1.75倍 は、それほど大きくない
  - なぜならば、depth = 1, 2の探索数は depth = 3　の探索数に比べて、十分小さいから。

Iterative Deeping による探索増加量

- 次の探索問題を考える
  - 分岐数 b
  - 深さ d
- 深さ d のノード数を $N(d)$とすると、
  - 深さ $d+1$ のノード数は、$N(d+1) = N(d) \times b$
    - $N(d) = b^d$
  - 従って、深さnまでの探索ノード数は、
    - $S(d) = \sum N(d) = \frac{b(b^d - 1)}{b-1}$
  - Iterative Deeping の探索ノード数は、
    - $D(d) = \sum S(d) = \frac{b(b^{d+1} - db + d - b)}{(b-1)^2}$
  - Iterative Deeping による探索増加率は、
    - $\frac{D(d) - S(d)}{S(d)} = \frac{D(d-1)}{S(d)} \approx \frac{1}{b-1}$
    - $b = 2$ で100%、$b = 3$ で50%、$b = 6$ で20%
    - ただし、深さnで解が見つかる位置によって、多少変化する

反復深化探索の性質

幅優先と深さ優先の利点を合わせ持つ

○ 完全性(completeness)
解があれば必ず見つける

○ 最適性(optimality)
最も浅い解を見つける

幅優先の利点

× 時間計算量(time complexity)
b^d

○ 空間計算量(space complexity)
bd

深さ優先の利点

kussharo.complex.eng.hokudai.ac.jp/~kurihara/classes/AI/blind.ppt

探索戦略の比較

	幅優先 (BFS)	深さ優先 (DFS)	反復深化 (ID)
完全	○	△	○
最適	○	×	○
時間	△ b^d	× b^m	△ b^d
空間	△ b^d	○ bm	◎ bd

b : 分枝度　d : 解の深さ　m : 探索木の最大の深さ

kussharo.complex.eng.hokudai.ac.jp/~kurihara/classes/AI/blind.ppt

(演習2)

1. 分岐数3の探索木において、深さ3のところの27個の探索ノードのうち、14番目に探索されるノードに解がある場合について、幅優先探索とIterative Deeping による探索のノード数を比較せよ。
2. 分岐数4, 深さ3の20番目に解がある場合についても、同様に比較せよ。

分枝限定法

最適解探索の解の品質

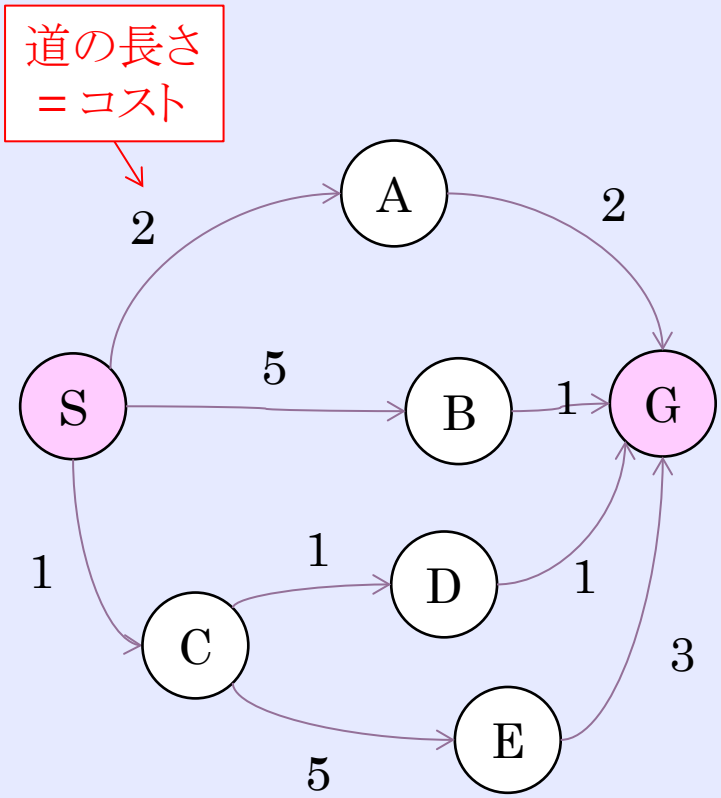
- 最適解探索を行う場合には、解に品質(あるいはコスト)が定義される
  - 品質を最大(コストを最小)にする解を求める
- 解の品質は、様々なところに定義される
  - 末端ノードに定義される場合
    - 例: 到達した場所が高級ホテルか、安宿か
  - パスに定義される場合
    - 例: 到達するまでの道のりの長さ、乗車料金、探索の深さ
  - 融合
    - 末端ノードとパスの品質の融合

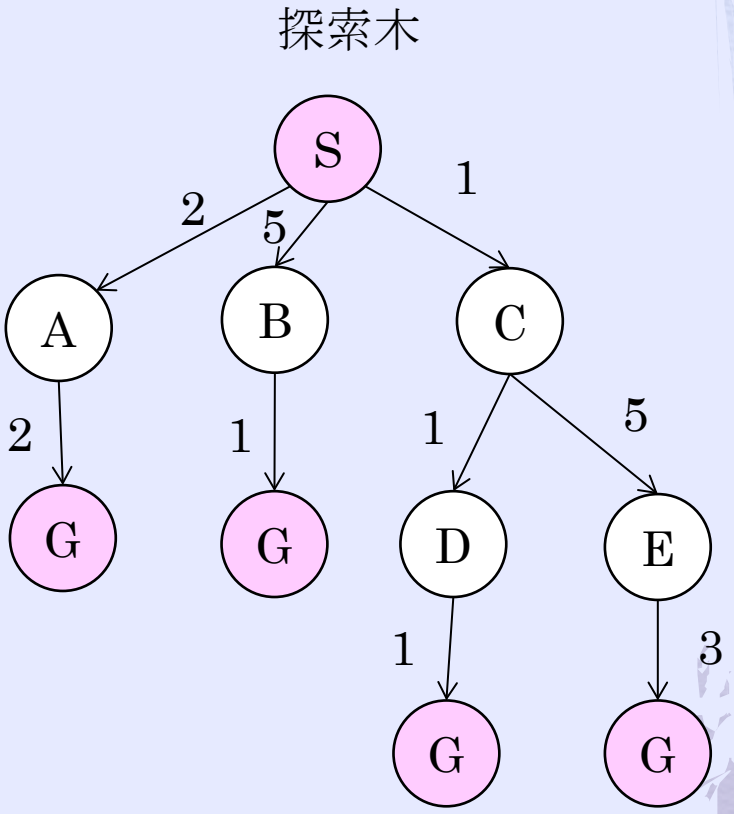
分枝限定法

- 探索パスのコストが、パスに沿って単調増加する場合、探索の途中で、その探索ノードの下には、最適解が存在しないことが明かな場合がある
  - すでに見つかった解のコスト c(g)
  - 探索途中のノードpまでのコスト c(p)
    - ノードpからたどれる解のコストは、単調増加の性質より、$c(p_g) >= c(p)$
  - よって、c(p) > c(g)の時、pの下位ノードに最適解は存在しない。
- 上記の場合、探索をノードpで打ち切る方法を分枝限定法と呼ぶ

分枝限定法の例

- 経路探索問題として、左図の問題を考える



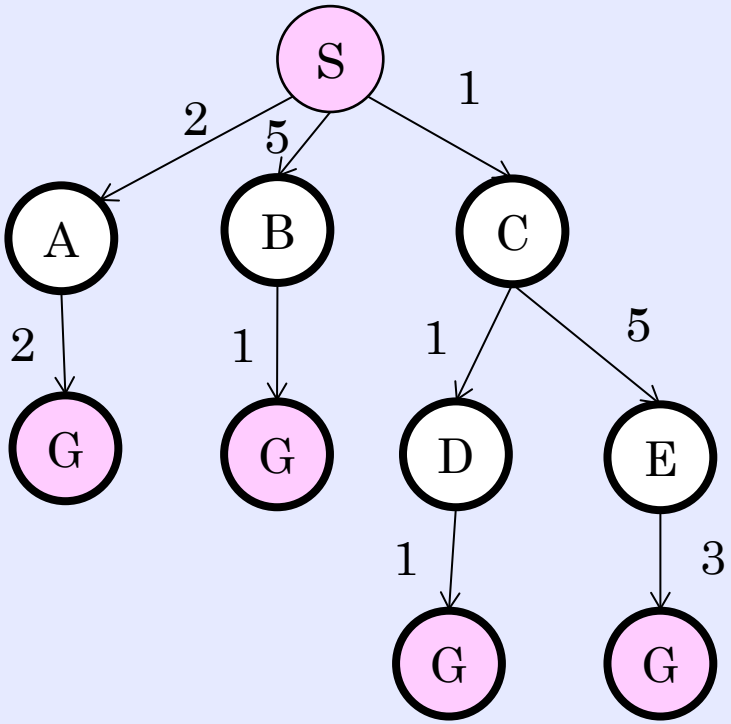


分枝限定法の例

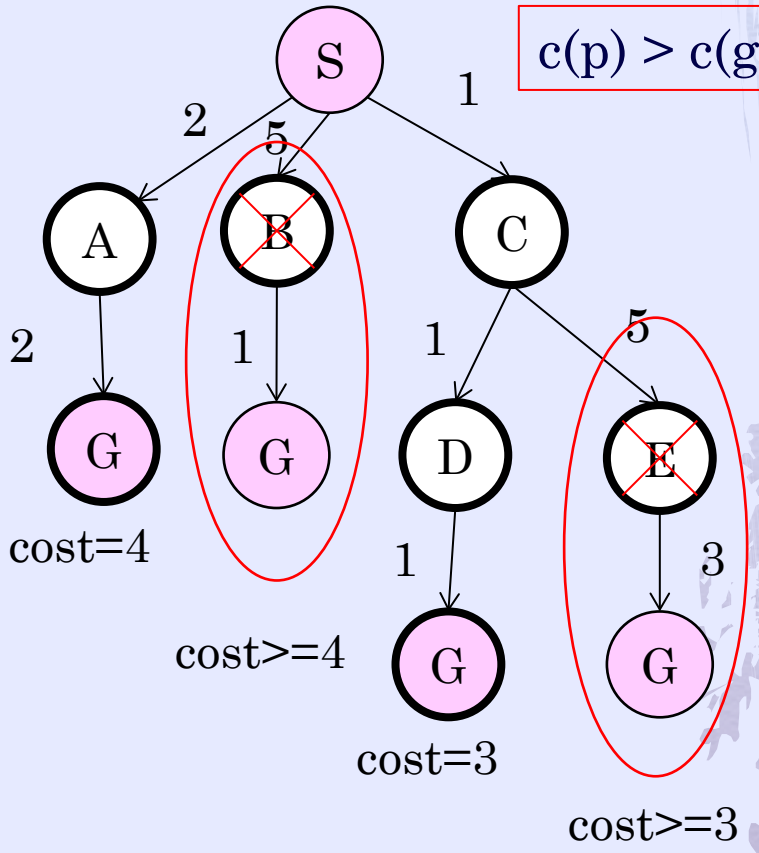
- 分枝限定法により、探索ステップ数が減少

c(p) > c(g)の時、pの下位ノードに最適解は存在しない

深さ優先探索=9ステップ



分枝限定法 = 7ステップ



(課題1)

- 次の15パズルについて、探索木を作成し、幅優先探索と Iterative Deeping の探索ノード数について、それぞれ計算せよ。なお、探索順は、上・左・下・右の順とする。

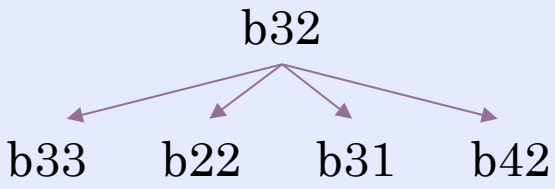
初期状態

4	1	2	3	4
3	5	6	7	8
2	9	10	■	11
1	13	14	15	12
	1	2	3	4

解

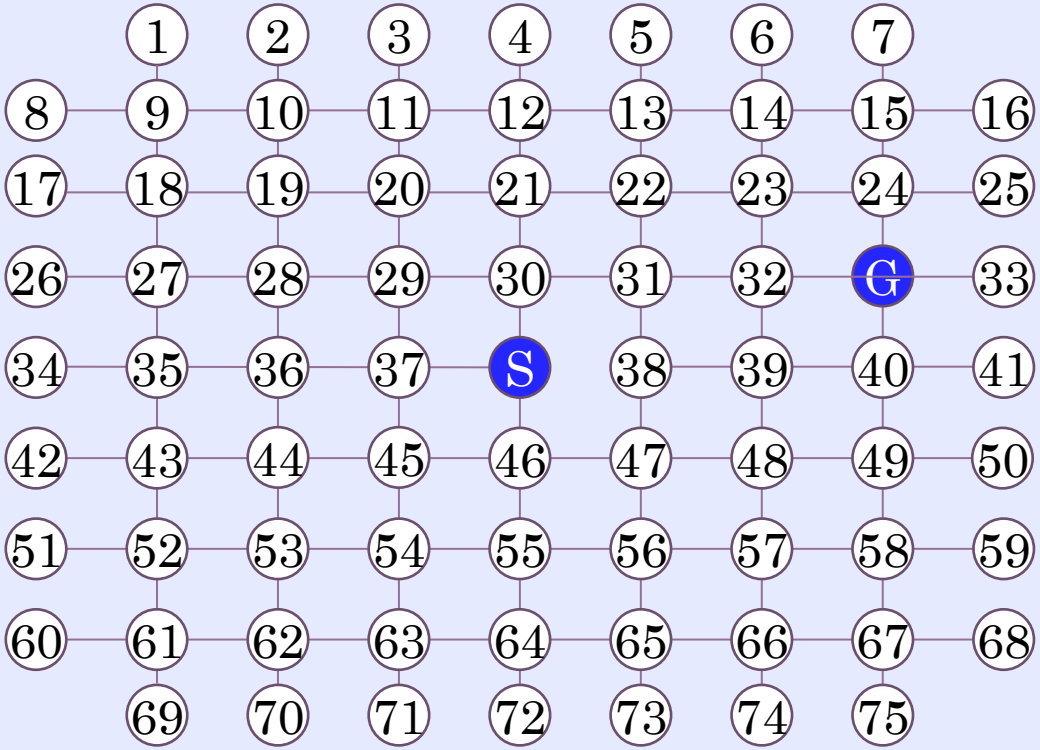
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	■

ただし、各状態は、黒のマスの位置をつかって、初期状態をb32、解の状態をb41のように表せ。



(課題2)

- 下記の道路網がある時、出発点Sから目的地Gまでの探索木を作成し、幅優先探索とIterative Deepingの探索ノード数について、それぞれ計算せよ。なお、同じノードは二度探索しないことにする。端点は行き止まりである。なお、探索順は、上・左・下・右の順とする。



(オプション課題3)

- 分枝限定法を実行するプログラムを作成し、例題の最短経路探索を実行せよ。