すべての基本“全探索”

情報理工学コース3年　菊地　翔馬

全探索とは

しらみつぶしにすべての組み合わせを調べ、その中から解をさがす方法。

今回は、基本中の基本

・深さ優先探索（Depth-First Search）

・幅優先探索（Breadth-First Search)

を扱います。

全探索の前に知っておいてほしい知識

その前に、全探索のコードの中で使う考え方やデータ構造を覚えましょう。

・再帰関数

・スタック（stack)

・キュー(queue)

再帰関数

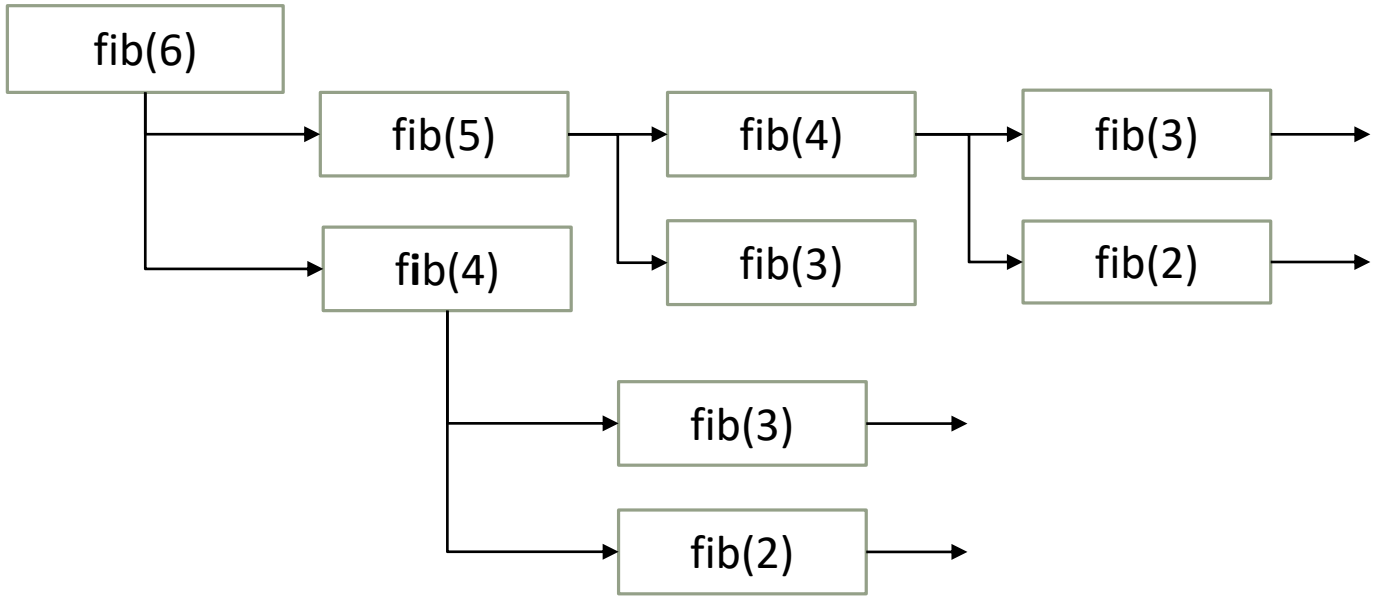
再帰関数とは、関数の中で同じ関数を呼び出すことを再帰呼び出しといい、再帰呼び出しをする関数を再帰関数といいます。

例：フィボナッチ数列を計算する関数int fib(int n)

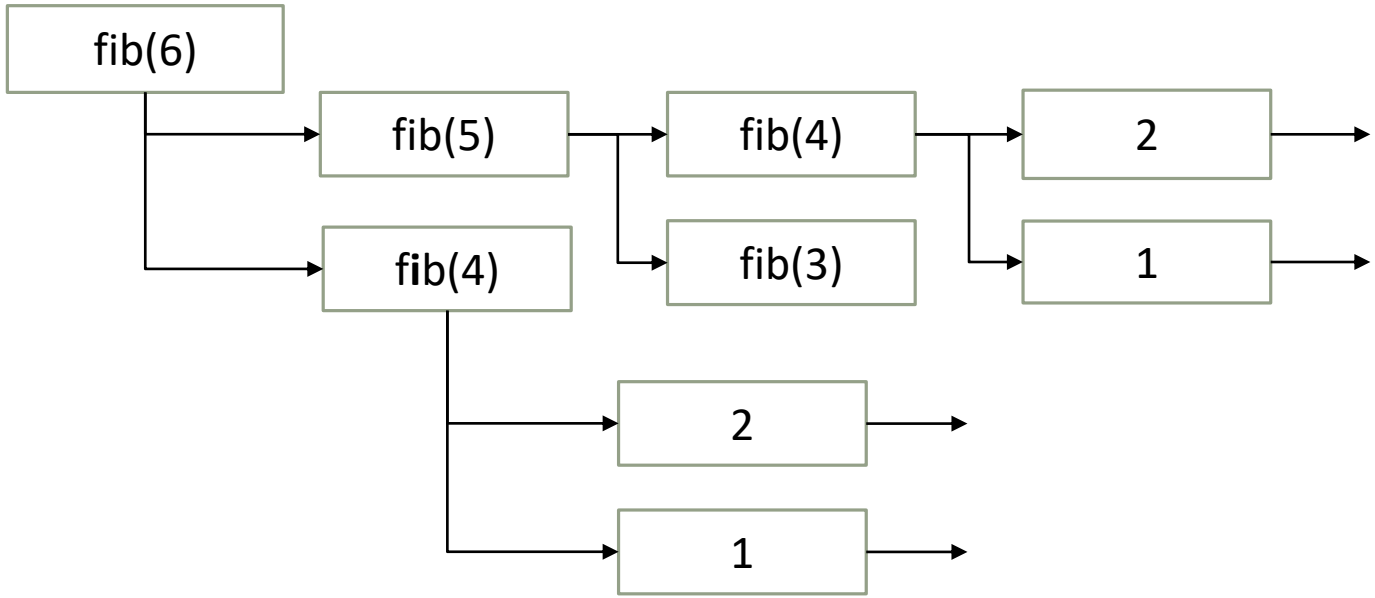
```
Int fib( int n){
        if(n <= 1) return n;            //停止条件
        return fib(n-1) + fib(n-2);    // 関数fibの中で関数fibが呼び出されている
}
```

fib(0)=0 , fib(1)=1 , fib(2)=1, fib(3)=2 , fib(4)=3 , fib(5)=5 , fib(6)=8 , …

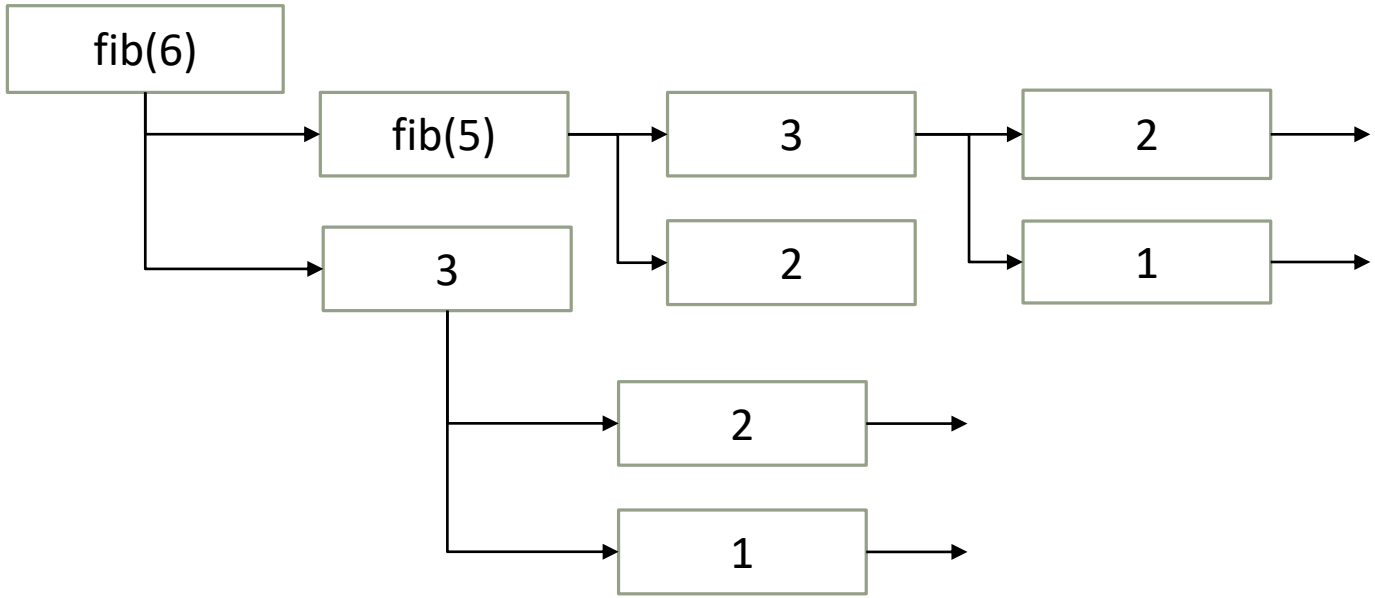
fib(6)の再帰の様子



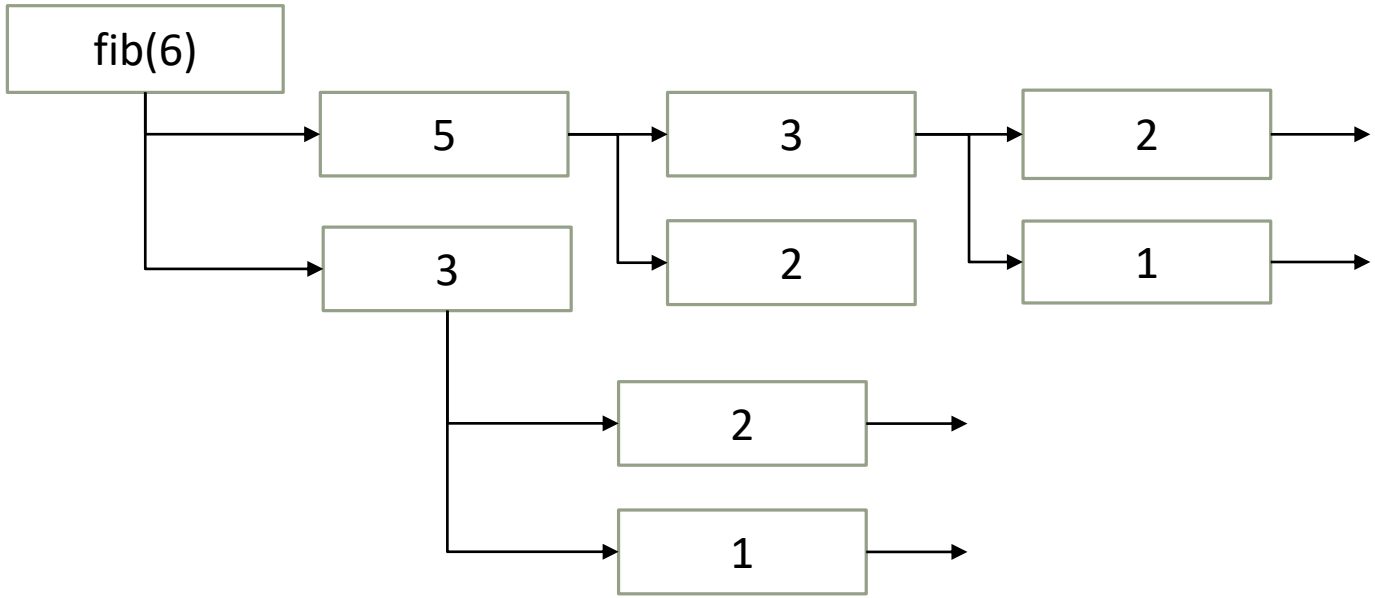
fib(6)の再帰の様子



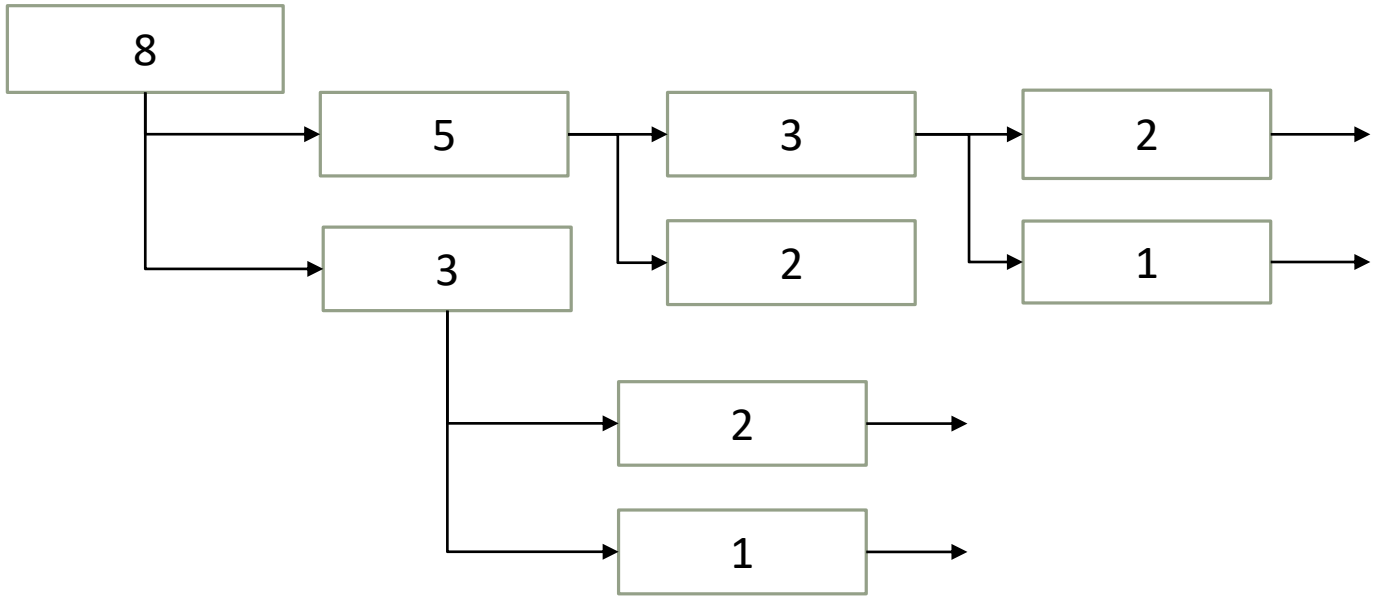
fib(6)の再帰の様子



fib(6)の再帰の様子



fib(6)の再帰の様子



スタックとキュー

次に、深さ優先探索と幅優先探索で用いられているデータ構造を紹介します。

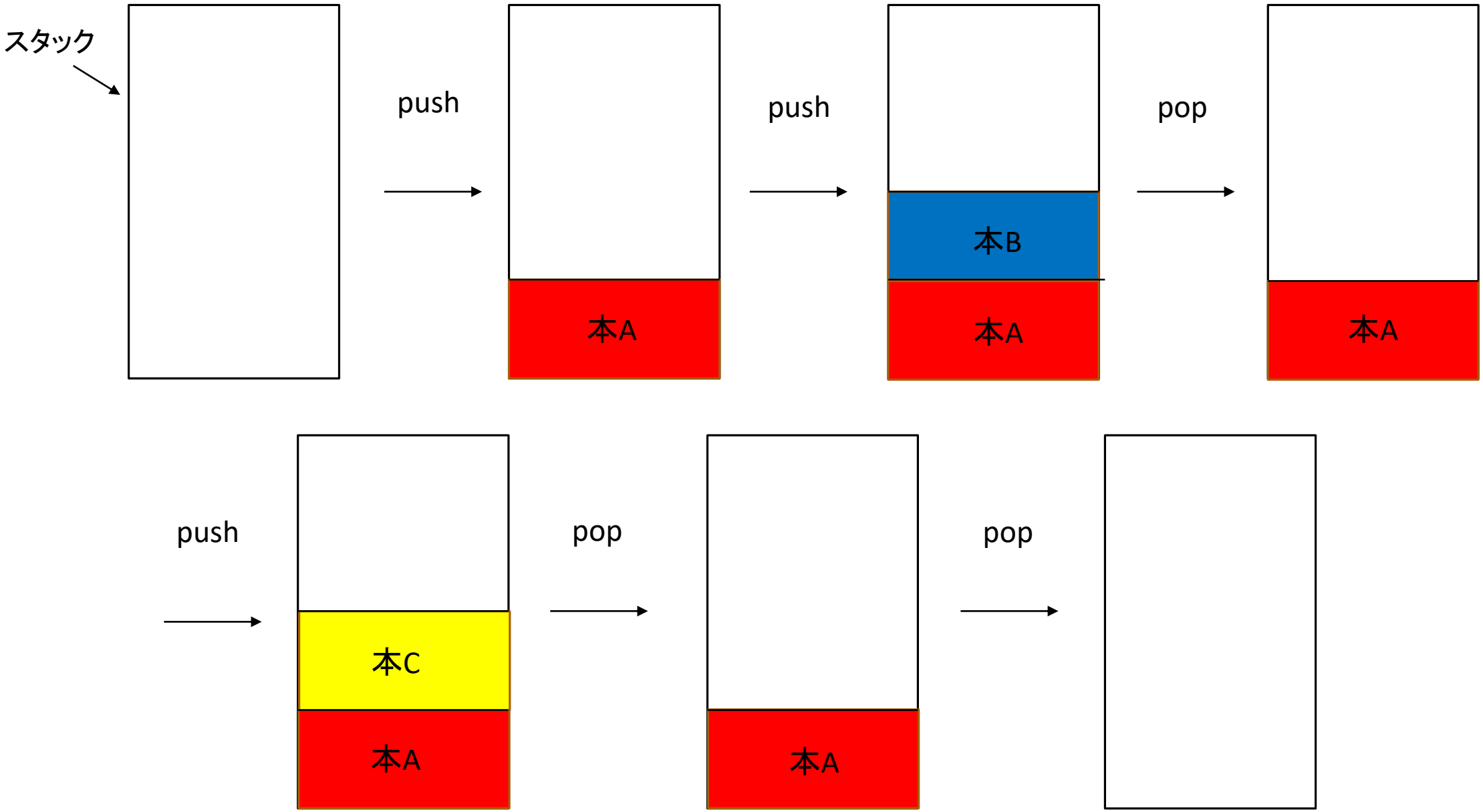
データ構造って？

そもそもデータ構造とは、計算機でデータを効率よく扱うために、データの格納の形式がすでにできているもの。中身は配列をうまく組み合わせたものなので、配列を使って同様のものを実装することは可能であるが、C++やJavaでは標準ライブラリが用意されているので、楽しく安全に使える。Cだとめんどくさい。

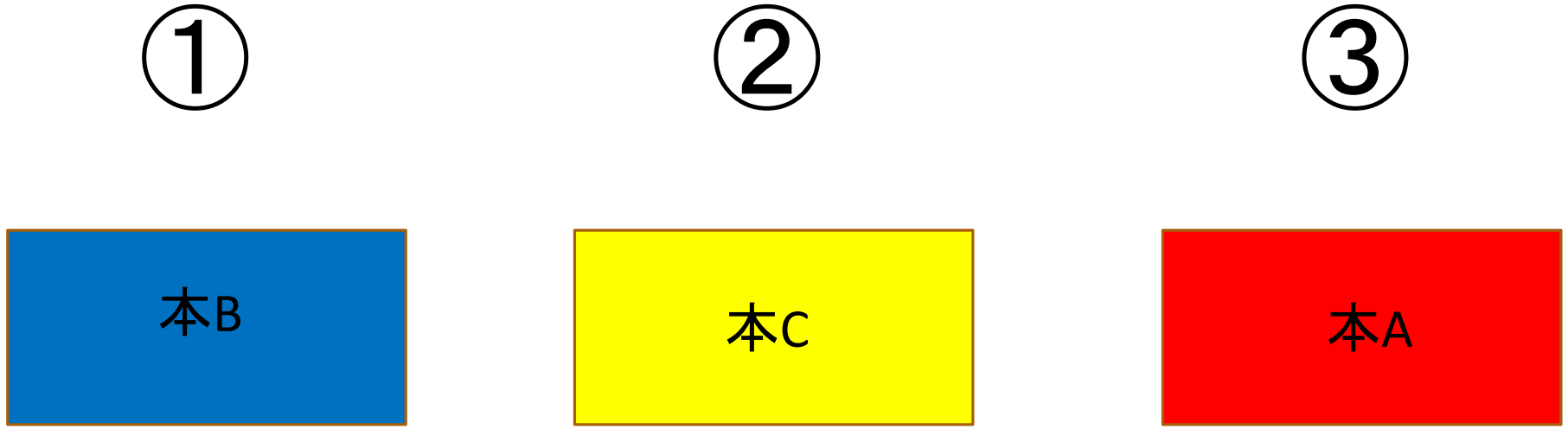
データ構造にはこれから紹介するスタックとキューや他に、リスト、ベクター、マップなどがある。

スタック（stack）

まずは、図のイメージから。

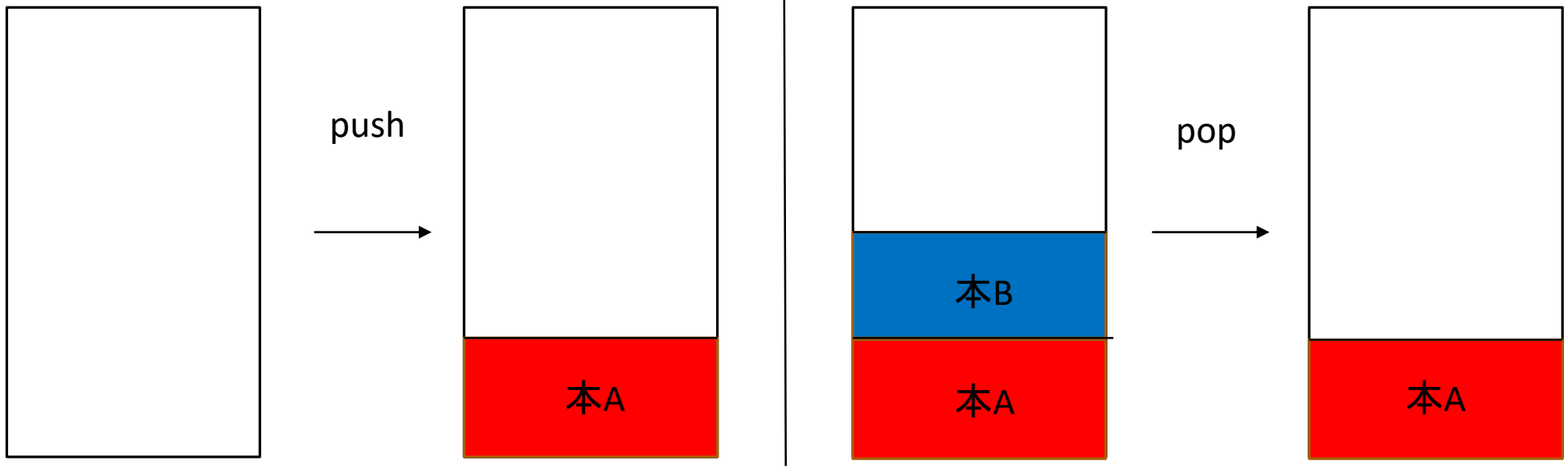

スタック
push
push
pop
本A
本B
本A
本A
push
pop
pop
本C
本A
本A

出された本の順番は



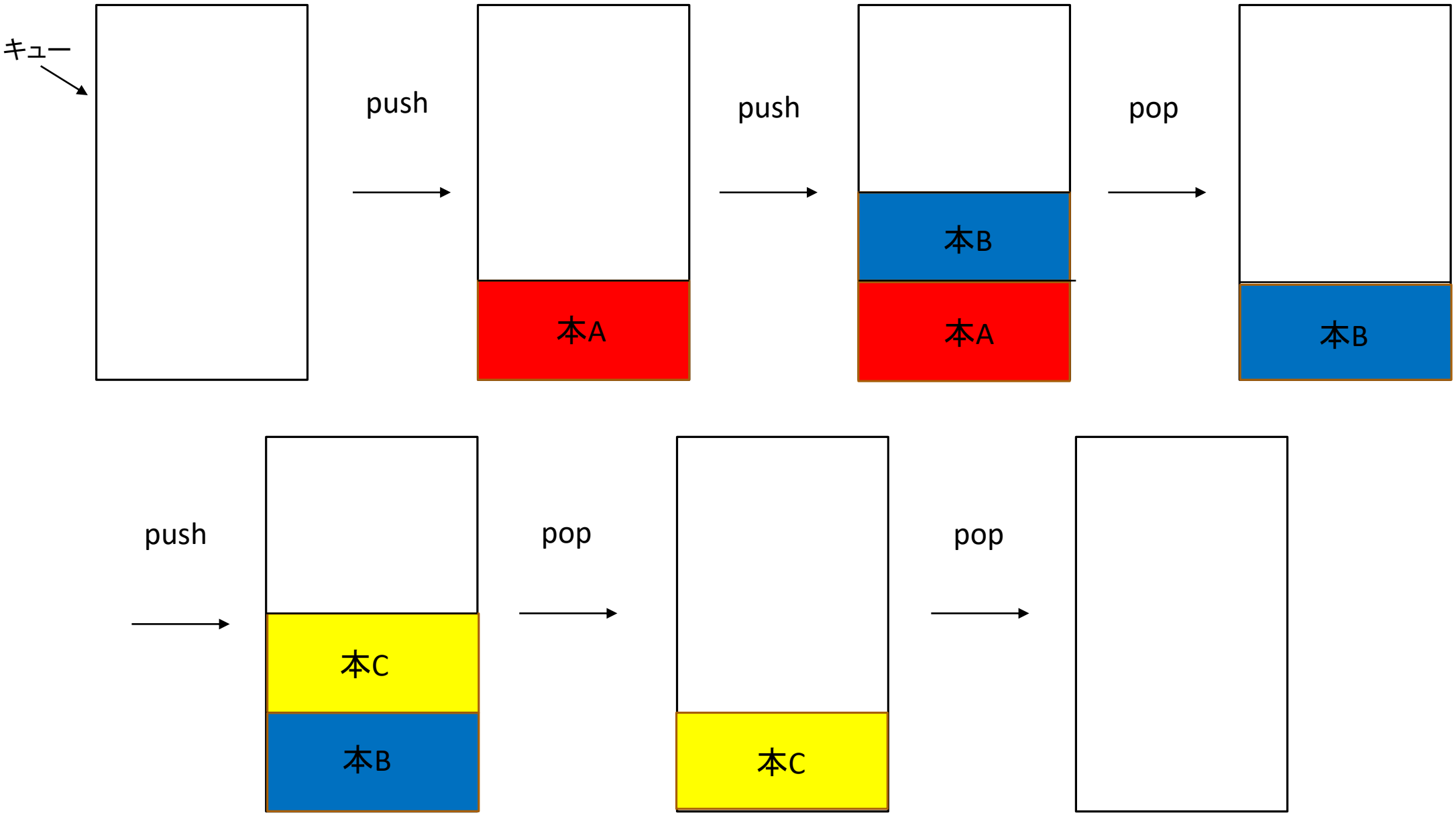
スタック（stack）

スタックは、pushとpopという二つの操作ができるデータ構造。Pushはスタックの一番上にデータを積む操作。Popは逆にスタックの一番上からデータを取り出す操作。つまり、最後に入れた要素が最初に出てくる。（これをLIFO：Last　In First　Outと呼ぶ。）

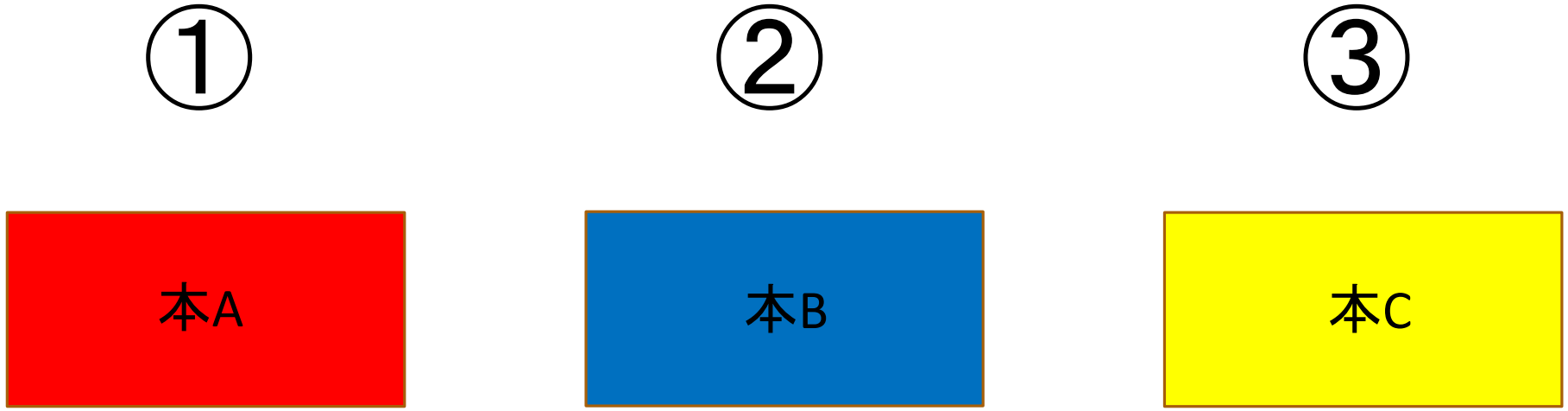


キュー（queue）

まずは、図のイメージから。

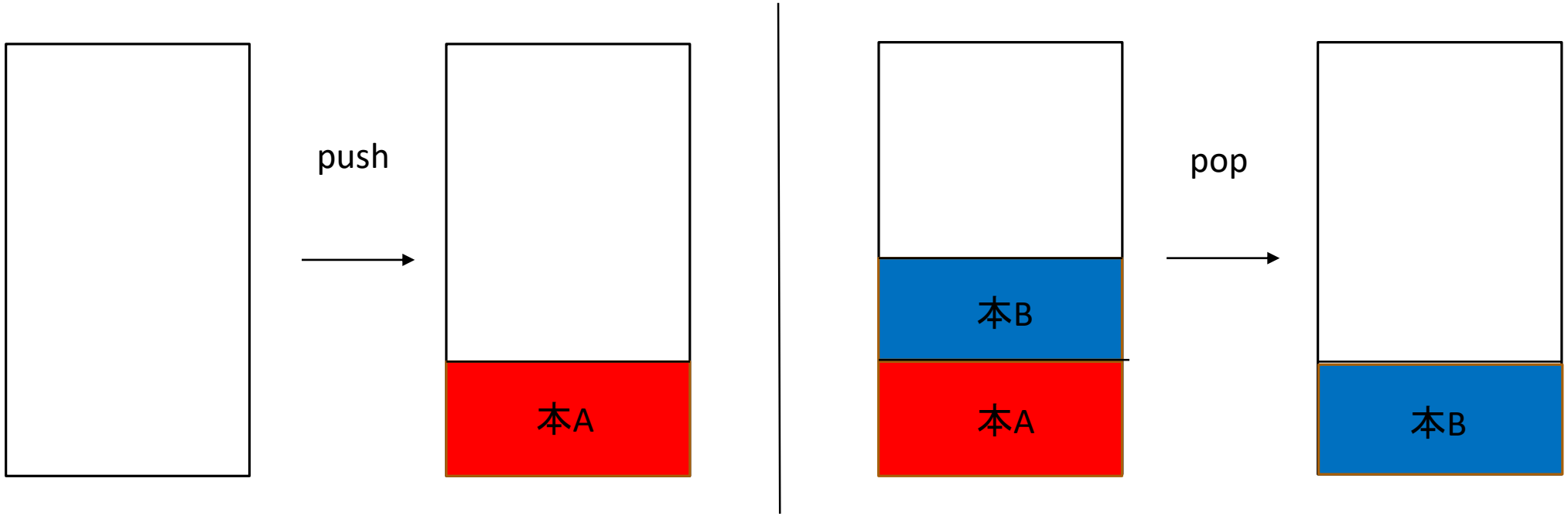

キュー
push
本A
push
本B
本A
pop
本B
push
本C
本B
pop
本C
pop

出された本の順番は



キュー（queue）

キューはスタックと同じように、pushとpopができるデータ構造。スタックと異なるのは、popが一番上から取り出すのではなく、一番下から取り出す部分。つまり最初に入れたものが最初に出てくる。（これをFIFO：First　In　First　Outと呼ぶ。）



再帰、スタック、キューの練習問題

再帰

最大公約数を求める

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_1_B&lang=jp

フィボナッチ数列

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_A

スタック

電車車両入替え

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0013&lang=jp

逆ポーランド記法

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_A&lang=jp

キュー

ラウンドロビンスケジューリング

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_B&lang=jp

深さ優先探索（Depth-First　Search）

深さ優先（DFS: Depth-First　Search）とは、グラフの探索アルゴリズムの一つです。

AOJ 0030 Sum of Integers

問題　制限時間1sec

n個の異なる数字で合計がsになるような組み合わせの数を出力せよ。ただし、各数字は0から9であるとする。（1≤n≤ 9、0≤s≤100）

例：　n = 3、s = 6の時は
1 + 2 + 3 = 6
0 + 1 + 5 = 6
0 + 2 + 4 = 6

の3通りなので、答えは3。

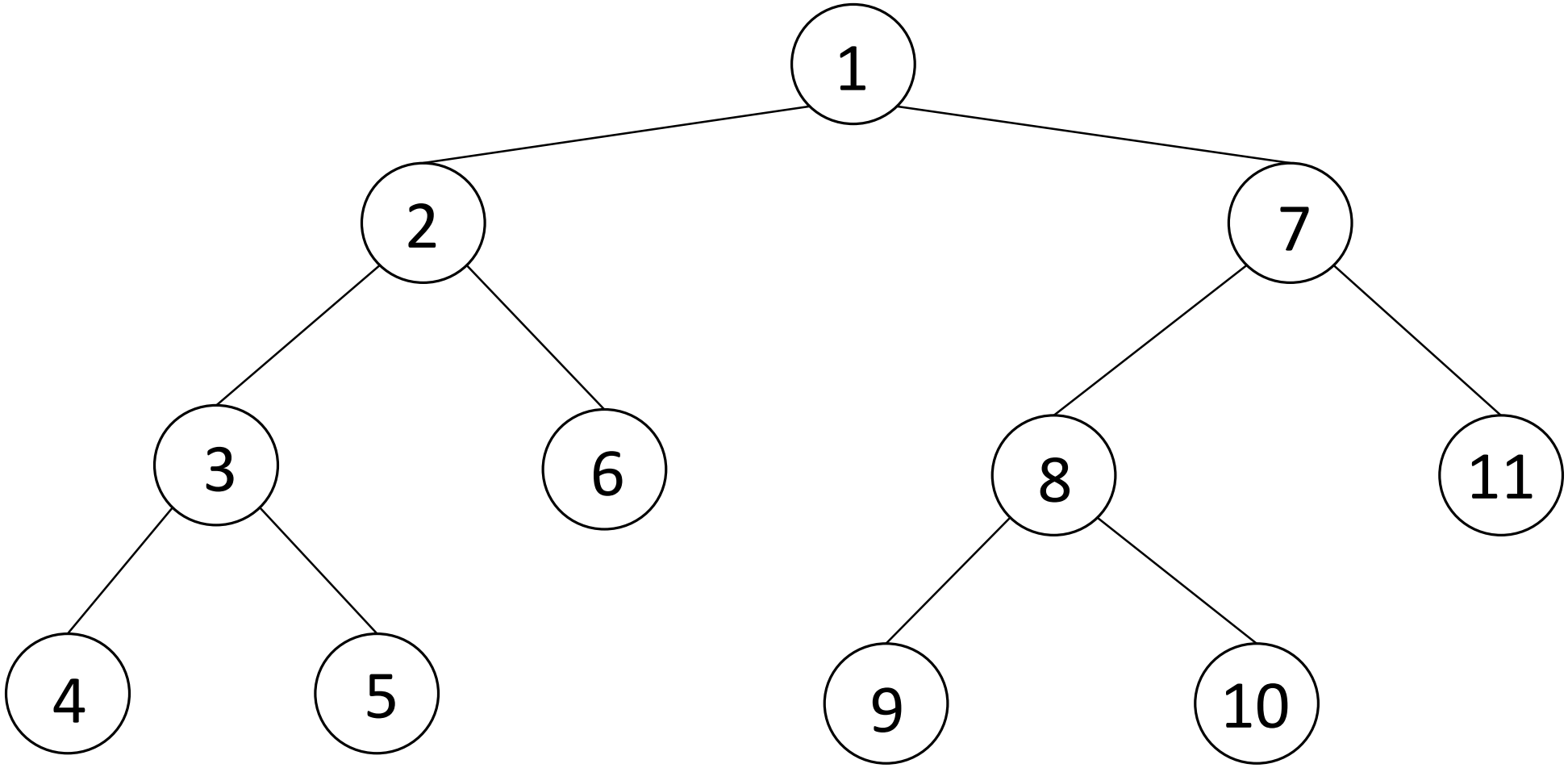
どうやって解く？

素直な解法

各数字に対して、ループ文を回して探す。

```
For i= 0 to 9
  For j = 0 to 9
    For k = 0 to 9
      …
      if i + j + k + …. == s then ans++
Return ans
```

これだと10^n回計算することになるので、最悪で10^9回計算することになる。実はこれ1秒じゃ終わらない・・・。1秒は10^7 ～10^8回くらい。



状態の遷移の順番

深さ優先探索（Depth-First　Search）

ある状態から出発し、遷移できなくなるまで状態を進めていき、遷移できなくなったら1つ前の状態に戻るということを繰り返して解を見つける。

一番はじめの状態から遷移を繰り返せば、たどり着ける全ての状態を見ることになります。したがって、全ての状態に対して操作を施したり、全状態を列挙したりできます。

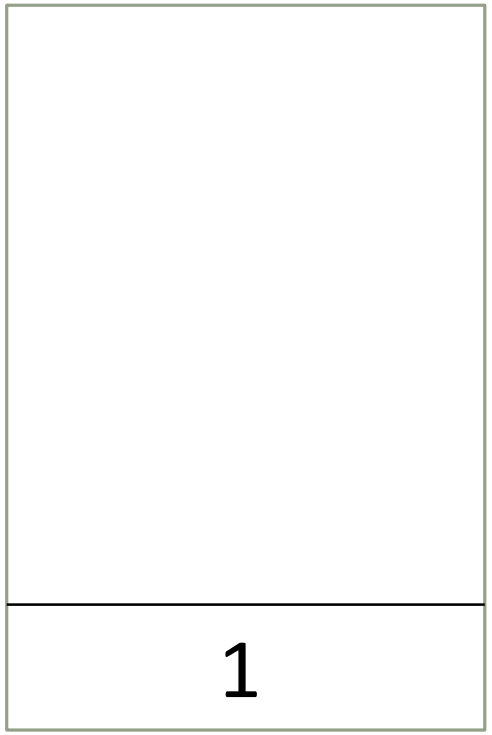
DFSはスタックを（暗に）使っている

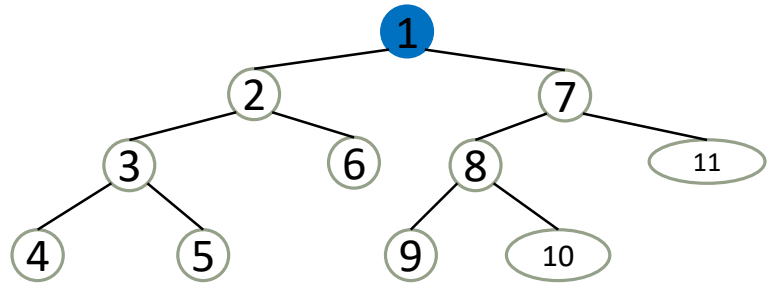
DFSを書くとき、コードの中にスタックが出てくることはないのですが、DFSでスタックを使ってる様子を図で見て見ましょう。

考え方は、あるノードを探索したら、そこからいける全てのノードをスタックに入れ、また先頭のノードを探索していきます。

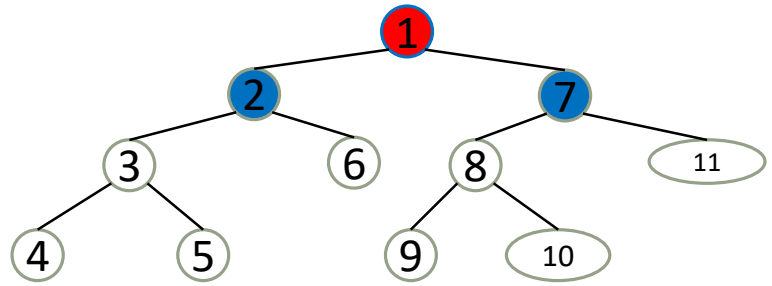
push pop

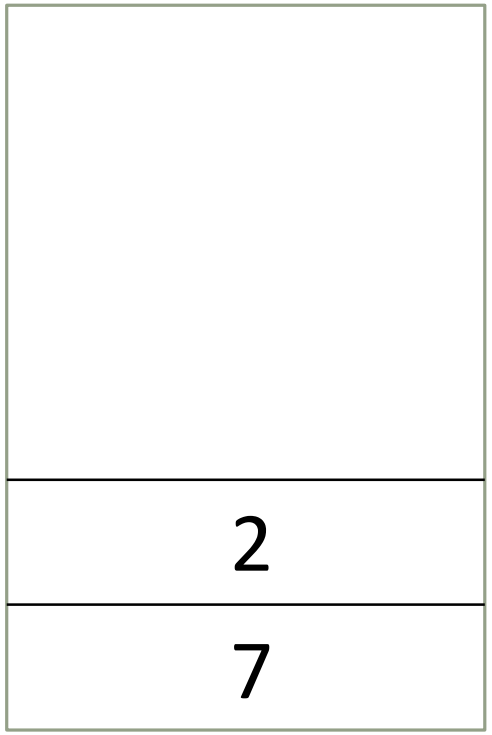
1



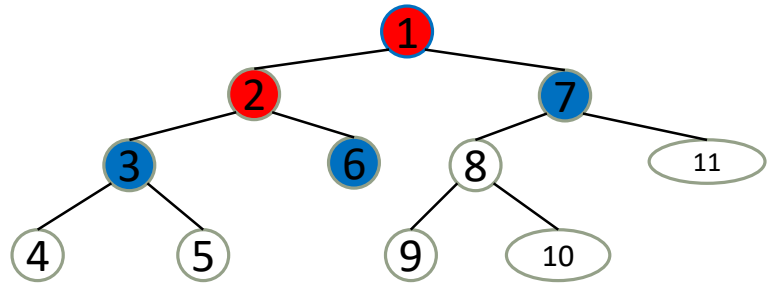


push	pop
2,7	1



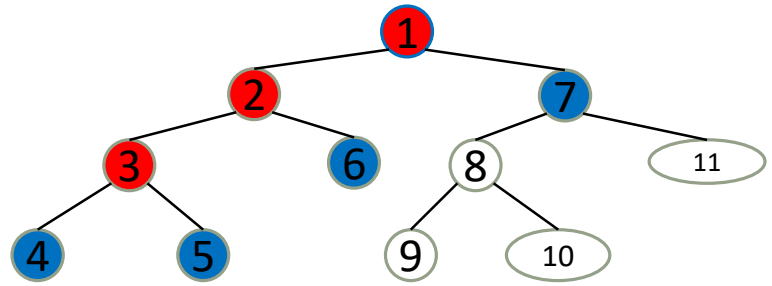


push	pop
3,6	2



3
6
7

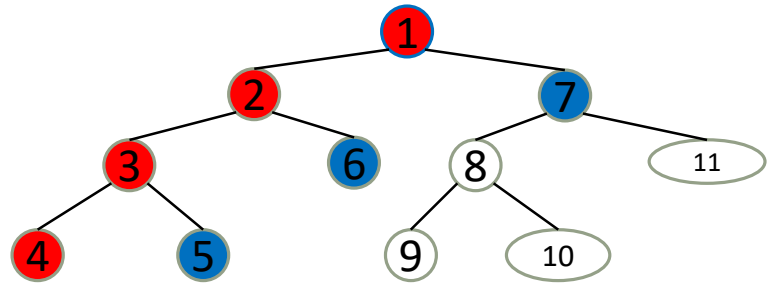
push	pop
4,5	3



4
5
6
7

push pop

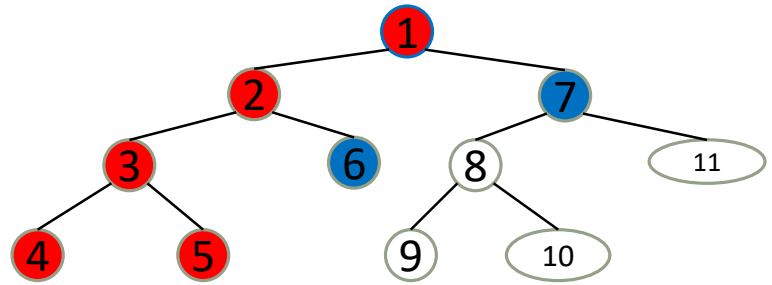
4



5
6
7

push pop

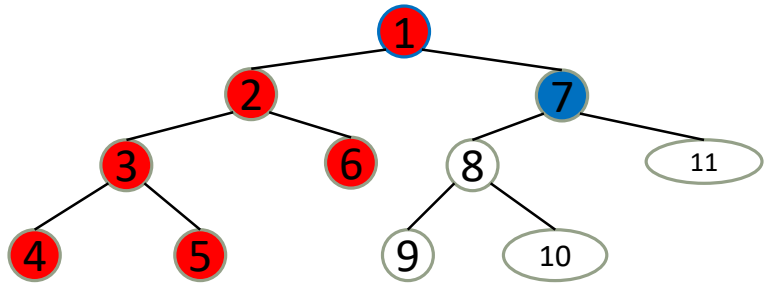
5

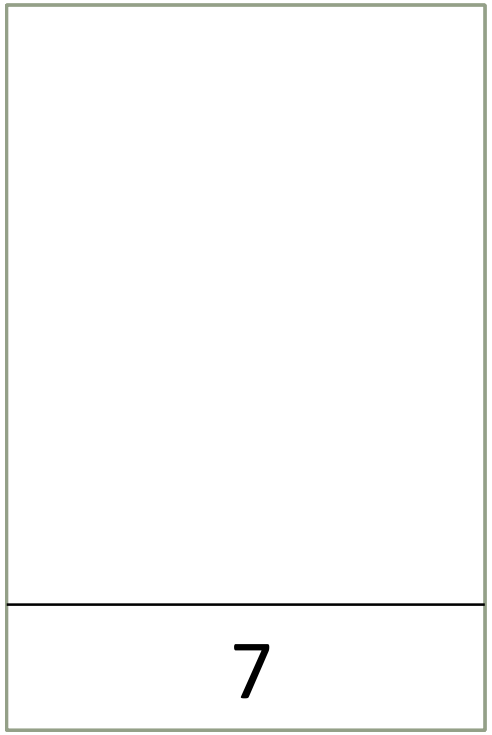


6
7

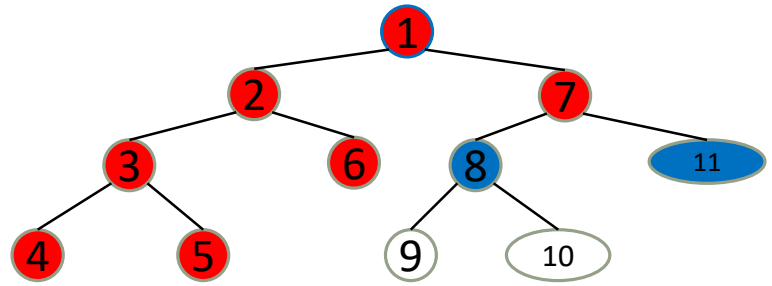
push pop

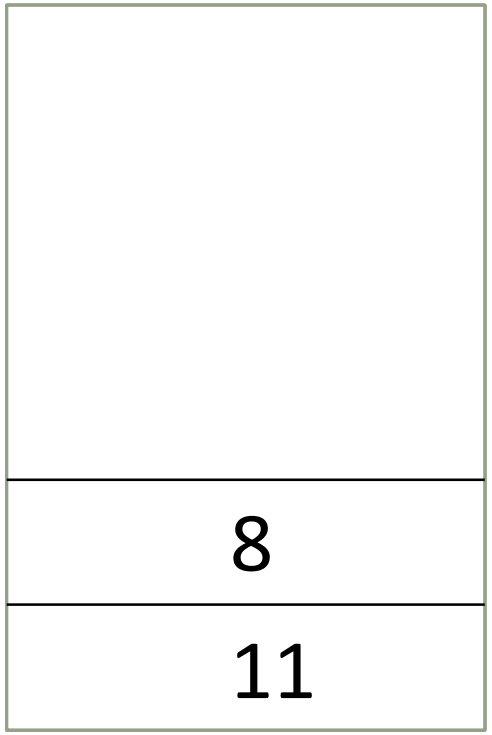
6



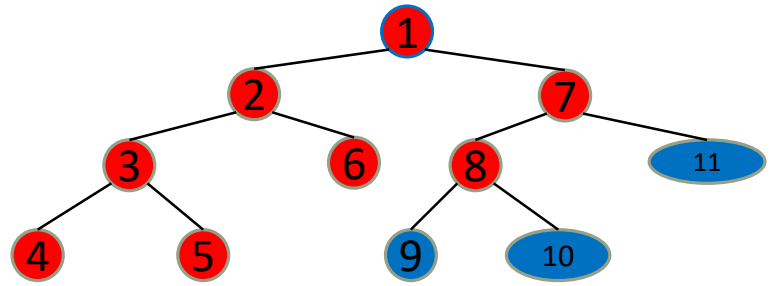


push	pop
8,11	7





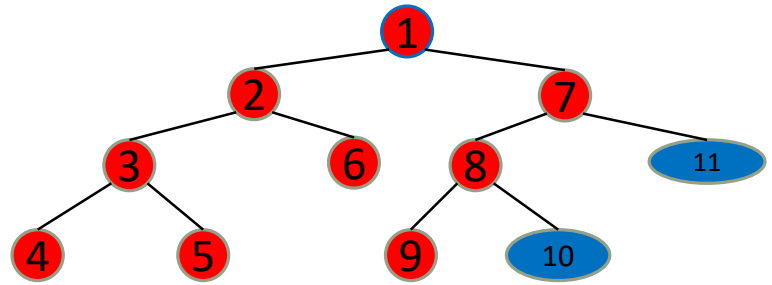
push	pop
9,10	8



9
10
11

push pop

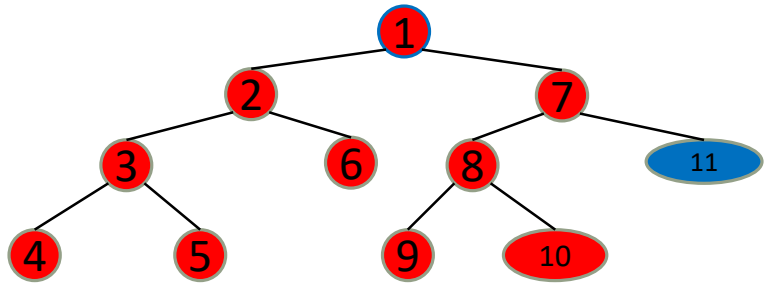
9

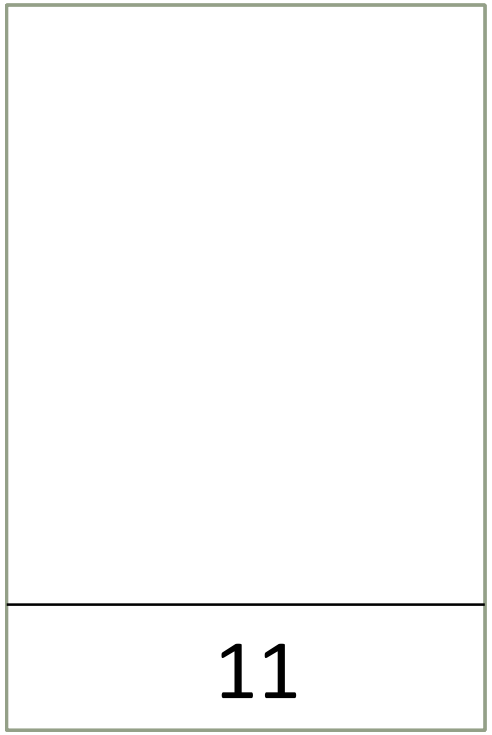


10
11

push pop

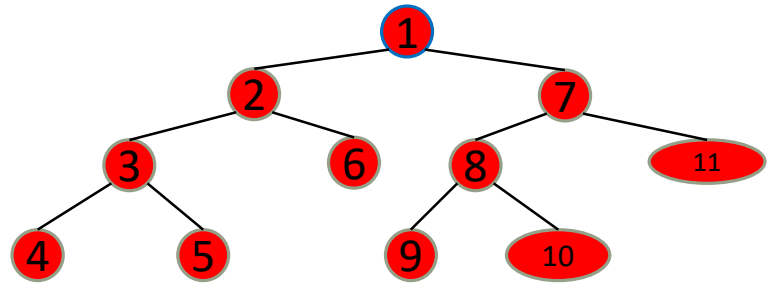
10

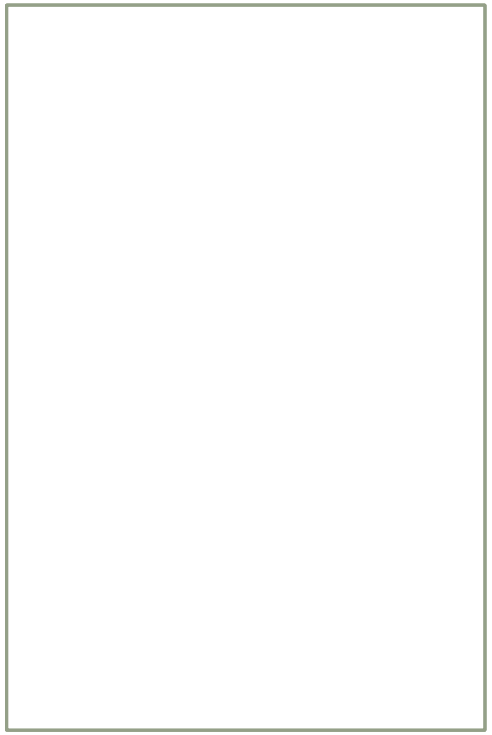




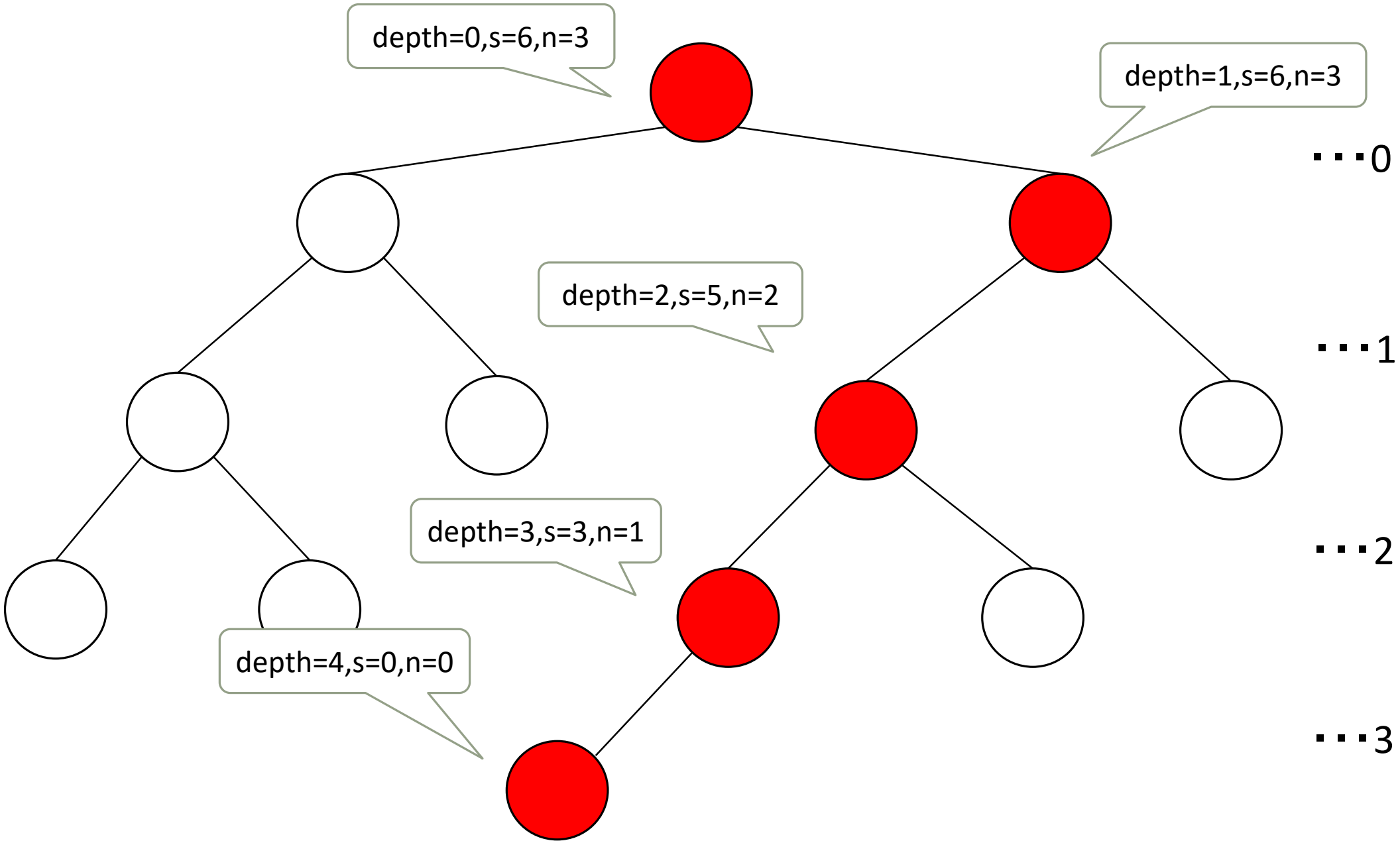
push pop

11





```cpp
#include<iostream>

using namespace std;

int ans;

void dfs(int n,int s,int depth){
    if (s == 0 and n == 0) { // 解が見つかったので終了。
        ans++;
        return;
    }
    if (n == 0) return; //全てのnについて試した終了。

    depth++; //加える数字でもある
    if (depth == 10) return; //0～9全て試したので終了。

    //選んだ場合
    dfs(n-1,s-depth,depth); // 数字を一つ選んだのでn-1、sから加えた数字を引く。
    //選ばなかった場合
    dfs(n,s,depth); //選んでないので何もしない。

    return;
}

int main(){
    while(1){
        ans = 0; //グローバル関数を初期化
        int n,s; cin >> n >> s;
        if (n + s == 0) break;
        dfs(n,s,-1);
        cout << ans << endl;
    }

return 0;
}
```


depth=0,s=6,n=3
depth=1,s=6,n=3
depth=2,s=5,n=2
depth=3,s=3,n=1
depth=4,s=0,n=0
...0
...1
...2
...3

計算量のお話

0～9の数字をそれぞれ、選ぶ、選ばないという状態があるので、状態数は$2^{n+1}-1$なので$O(2^n)$になります。
今回だと大きくても2^{10}回なので大丈夫
ここで状態数とは、さっきのグラフの〇の数のこと。

計算量のお話

深さ優先探索や幅優先探索は、全ての状態を探索するので、計算量はO(|V|)になる。|V|は頂点数。

計算量のお話

そもそも計算量とは？

→問題を解くのにどれくらいの計算をするか。

競技プログラミングではプログラム自体の速度も重要になるので必要な計算。自分で計算してプログラムが制限時間以内に間に合うかどうかを見積もる。

O(N),O(logN),O(N^2),O(2^n)とか書くことが多い。

教養の微積分学でランダウの記号をやったと思うけどそれと同じ感じ。

X→∞の時、f(x) = 3x^2 + 4x+ 5 は

f（x） = O(x^2)

である。

DFSを使って解く問題

Sum of Intager

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0030&lang=jprs

Ball

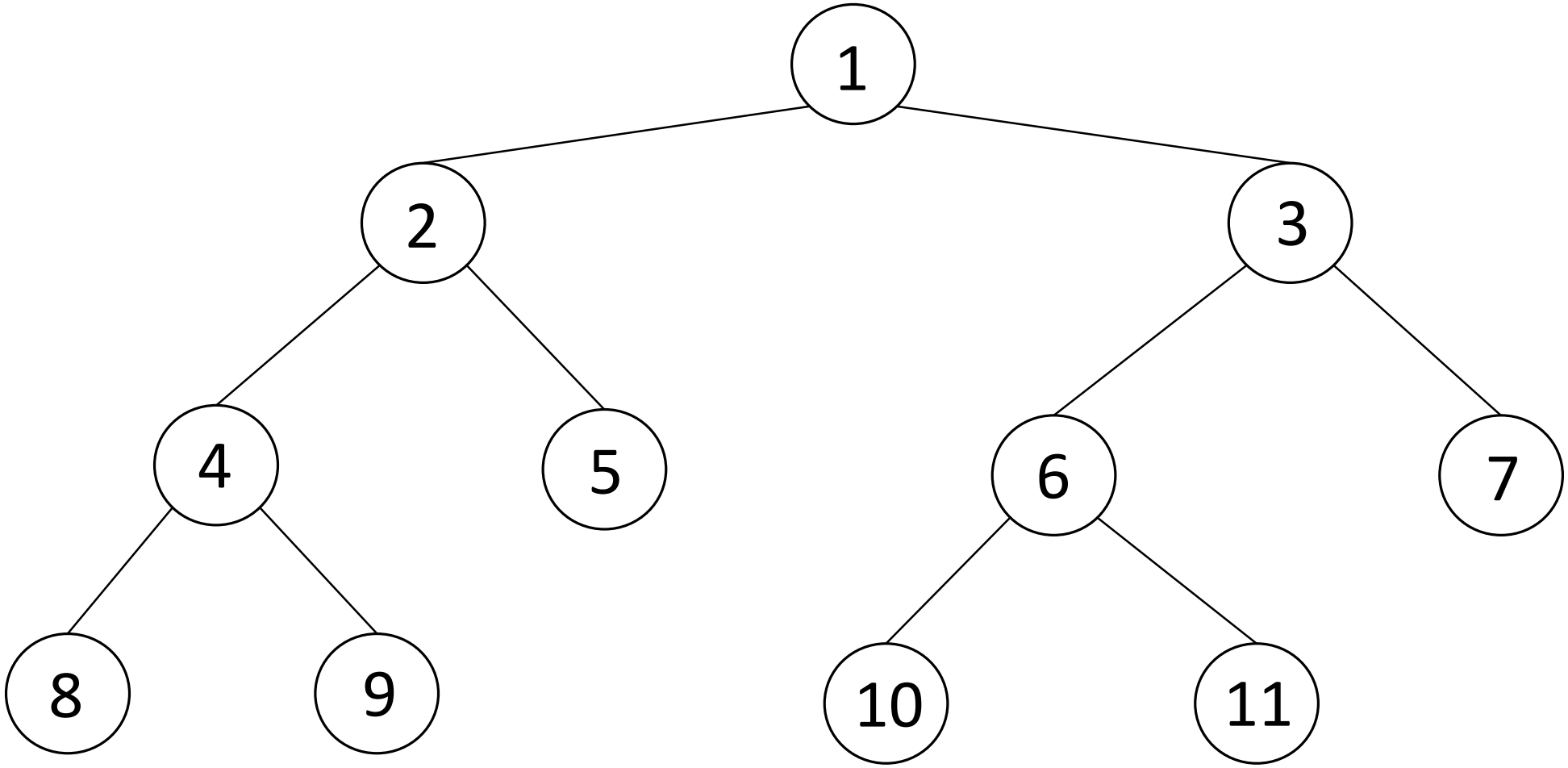
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0033

Split Up!

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1045

幅優先探索（Breadth-First　Search）

幅優先探索（BFS:Breadth-First　Search）も、グラフの探索アルゴリズムの一つです。



状態の遷移の順番

幅優先探索（Breadth-First　Search）

ある状態から出発し、近いところから状態を進めていく。

つまり、

最短1回の遷移でたどり着ける状態

↓

最短2回の遷移でたどり着ける状態

↓

最短3回の遷移でたどり着ける状態

↓

:

という順に探索します。

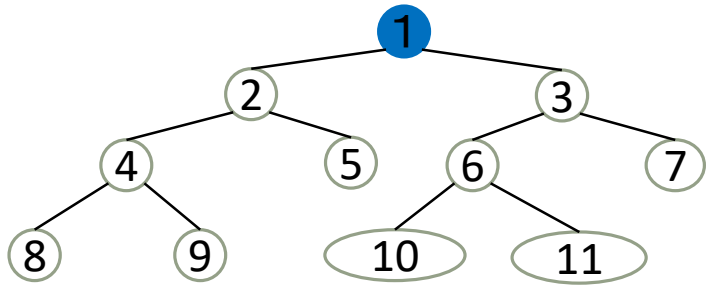
BFSはキューを使う

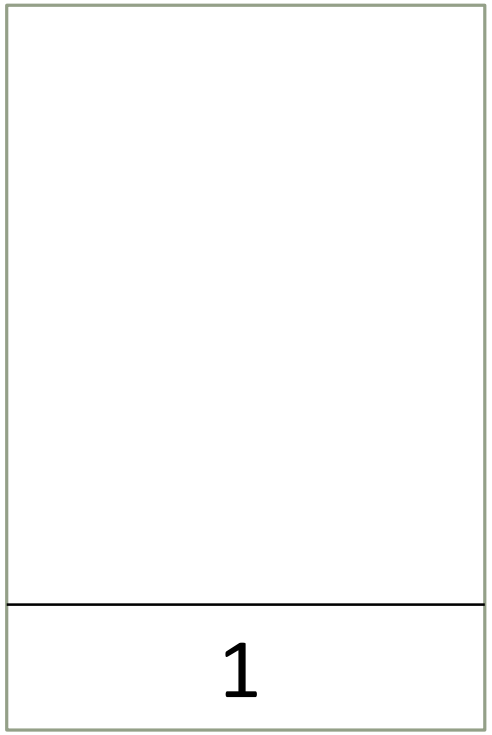
DFSでは、スタックは書かなくても良いのですが、BFSの場合はキューを書く必要があります。なのでCだとつらい。

ただ、考え方はDFSと同じで、あるノードを探索したら、そこからいける全てのノードをキューに入れ、また先頭のノードを探索していきます。

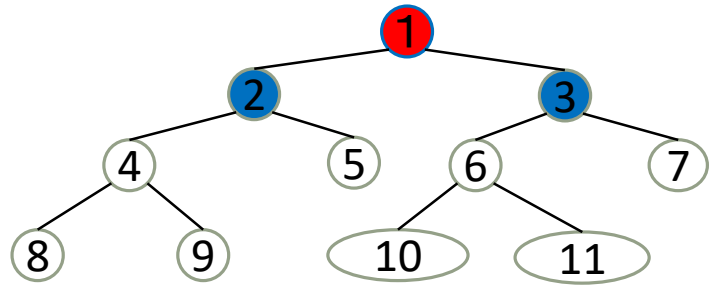
push pop

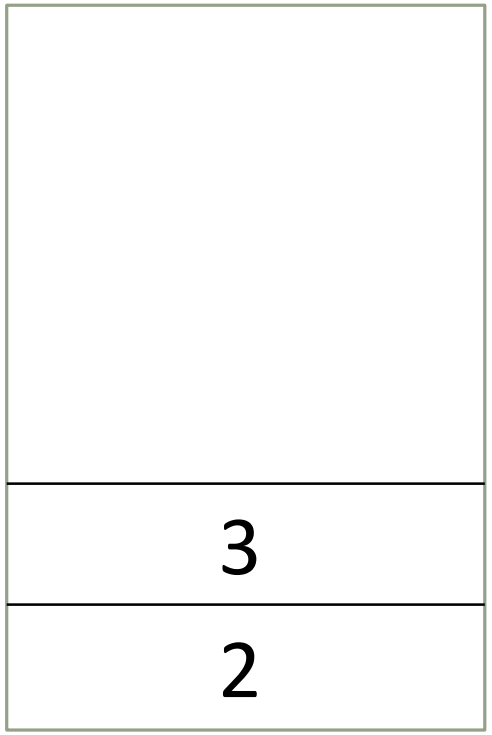
1



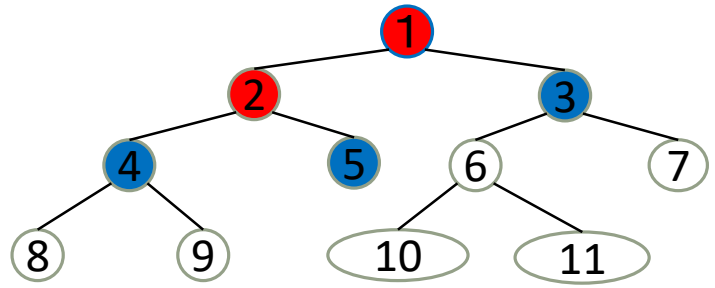


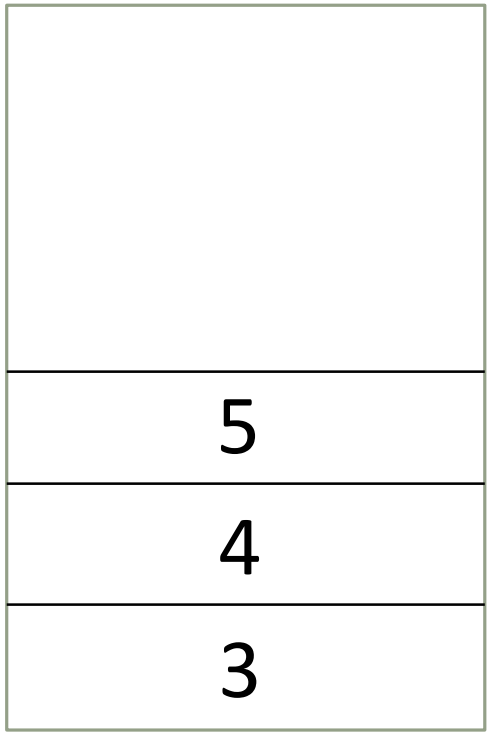
push	pop
2,3	1



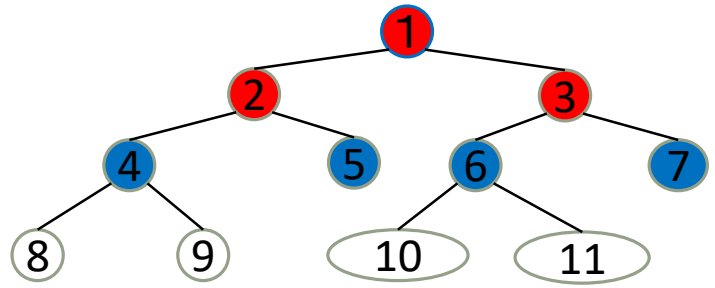


push	pop
4,5	2



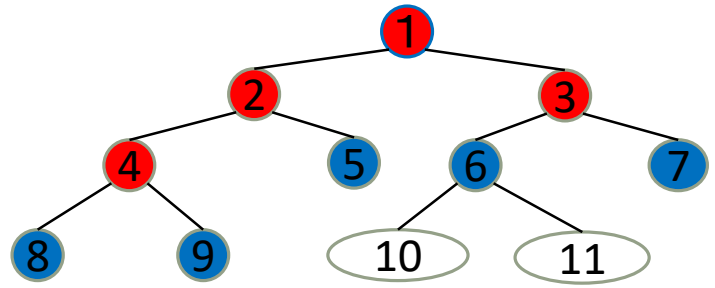


push	pop
6,7	3



7
6
5
4

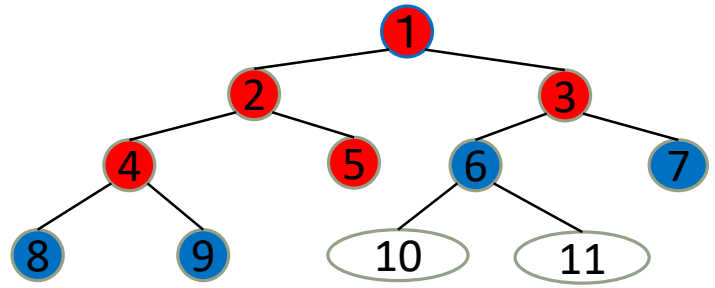
push	pop
8,9	4



9
8
7
6
5

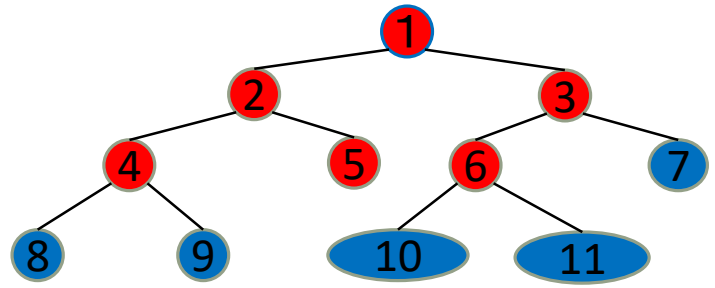
push pop

5



9
8
7
6

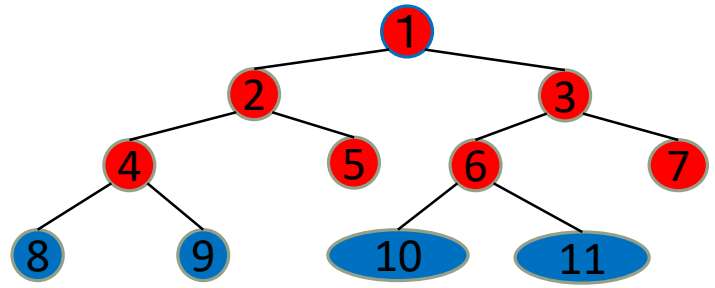
push	pop
10,11	6



11
10
9
8
7

push pop

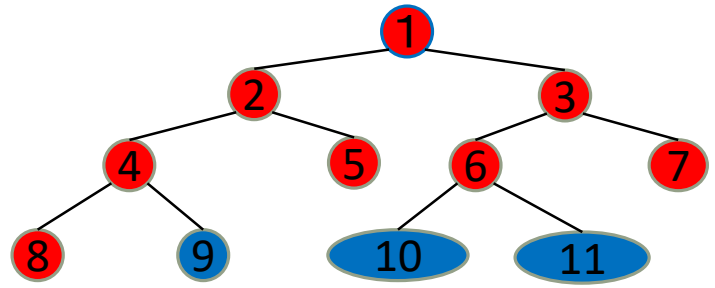
7



11
10
9
8

push pop

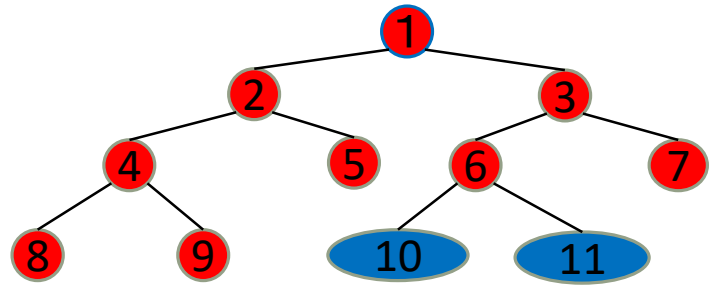
8



11
10
9

push pop

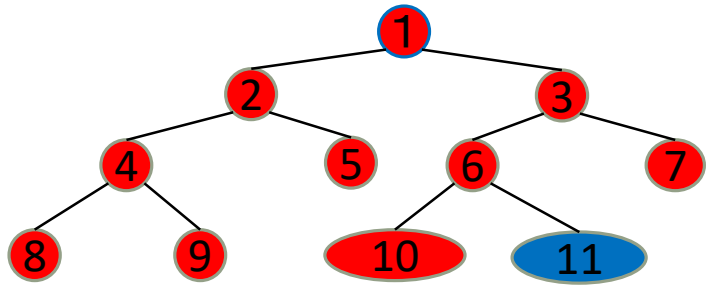
9

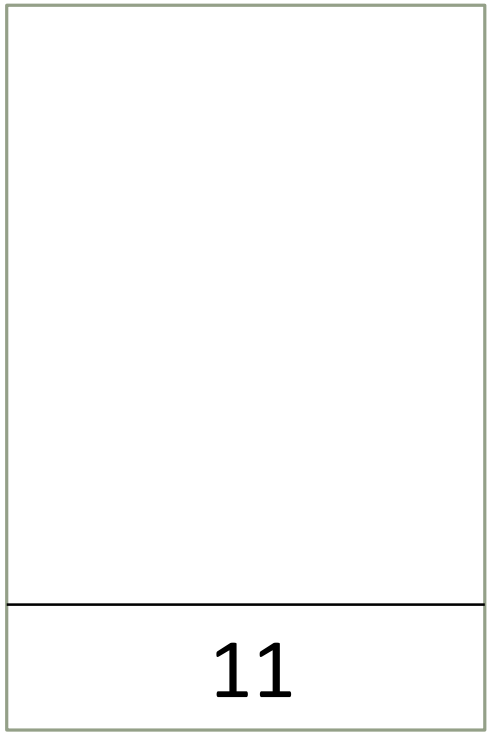


11
10

push pop

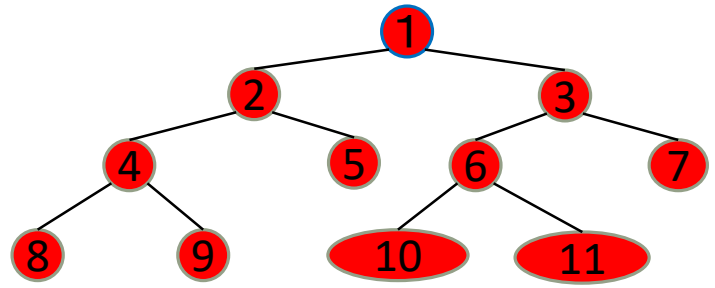
10

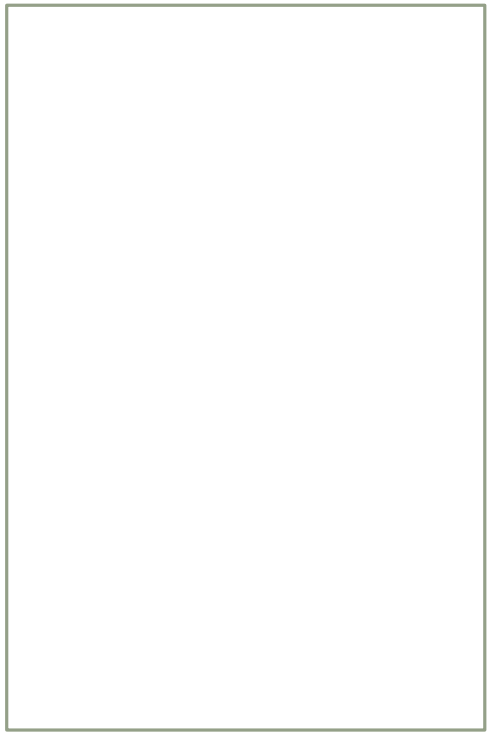




push pop

11





二次元配列のようなものも解ける！！

問題：迷路の最短路

大きさがＮ×Ｍの迷路が与えられます。迷路は壁と通路からできており、１ターンに隣接する上下左右４マスの通路へ移動することができます。スタートからゴールまで移動するのに必要な最小のターン数を求めなさい。

入力例：N=10,M=10

入力例： #,.,S,Gはそれぞれ、壁、通路、スタート、ゴール

N=10,M=10

#S######.#

......#..#

.#.##.##.#

.#.......

##.##.####

....#....#

.########.#

....#.....

.####.###.

....#...G#

```cpp
#include<iostream>
#include<vector>
#include<string>
#include<queue>
#define INF 2140000000
#define MAX_N 100
#define MAX_M 100

using namespace std;

int n,m;
vector<string> maze;

//方向ベクトル
vector<int> dx = {1,0,-1,0};
vector<int> dy = {0,1,0,-1};

int sx,sy;
int gx,gy;

vector< vector<int> > d(MAX_N,(vector<int>(MAX_M,INF)));

int bfs(void){
    queue<pair<int,int>> q;
    q.push(make_pair(sx,sy)); //一個目をキューにpush

    d[sy][sx] = 0; //スタート地点の距離を0

    while(!q.empty()) { //キューが空になるまで。
        pair<int,int> p = q.front();
        q.pop(); //先頭をpopする。

        if (p.first == gx and p.second == gy) {
            break;
        }

        for (int i = 0; i < 4; i++) {
            int nx = p.first + dx[i];
            int ny = p.second + dy[i];

            if (/*1*/0 <= nx and nx < n and 0 <= ny and ny < m and /*2*/maze[ny][nx] != '#' and /*3*/d[ny][nx] == INF) { //1.迷路の範囲外じゃない、2.壁じゃない、3.一度も訪れたことがない
                q.push(make_pair(nx,ny)); //遷移できる状態をキューにpushする
                d[ny][nx] = d[p.second][p.first] + 1;
            }

        }
    }
    return d[gy][gx];
}

int main() {
```

```
50
51 int main() {
52     cin >> n >> m;
53     for (int i = 0; i < n; i++){
54         string str; cin >> str;
55         maze.push_back(str);;
56         for (int j = 0; j < m; j++){
57             if (maze[i][j] == 'S'){
58                 sy = i; sx = j;
59             }
60             if (maze[i][j] == 'G'){
61                 gy = i; gx = j;
62             }
63         }
64     }
65
66     int ans = bfs();
67     cout << ans << endl;
68     return 0;
69 }
70
71
72
```

計算量

状態数は各マスのことなので、N×M。Loopごとに4方向のfor文を回すので4NM。

よってO(NM)になる。

BFSを使って解く問題

Seven Puzzle

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0121

Stray Twins

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0223

Mysterious Worm

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0179

Amazing Mazes

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1166&lang=jp