「アルゴリズム」資料 7
深さ優先探索と幅優先探索

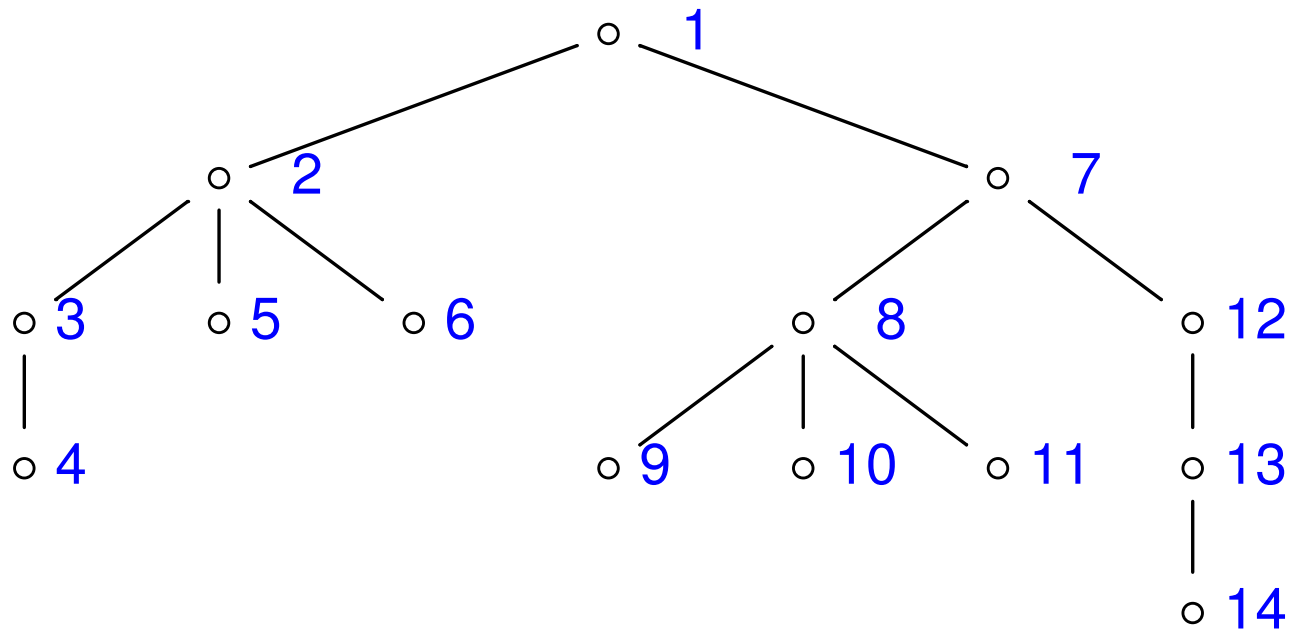
奈良女子大学理学部情報科学科
鴨浩靖

2009 年 12 月 1 日 初版
2011 年 11 月 14 日 第二版
2012 年 11 月 13 日 第三版
2020 年 10 月 20 日 第四版

深さ優先探索

深いほうへ進めるだけ進み、進めなくなったら戻るアルゴリズム。DFS と略す。

例：



深さ優先探索の実装—再帰呼び出し

```
再帰的に探す (状態)
{
    if (見つかった)
        見つかった場合の処理;
    if (まだ続ける必要がある) {
        次の状態の各々について {
            再帰的に探す (次の状態);
        }
    }
}
探す ()
{
    再帰的に探す (初期状態);
}
```

深さ優先探索の実装—スタックを使用

```
探す ()
{
    初期状態をスタックに積む;
    while (まだ続ける必要がある && スタックが空でない) {
        スタックから状態を降ろす;
        if (見つかった)
            見つかった場合の処理;
        次の状態をすべてスタックに積む;
    }
}
```

深さ優先探索の長所と短所

長所

- 実装が簡単。
- うまく実装すれば、メモリ効率が良い。

短所

- 深いパスがあると、そこに時間を取られる。特に、無限パスがある止まらなくなる。

深さ優先探索の実装—再帰呼び出し メモリの節約

次の状態から元の状態へ戻る操作が可能な場合

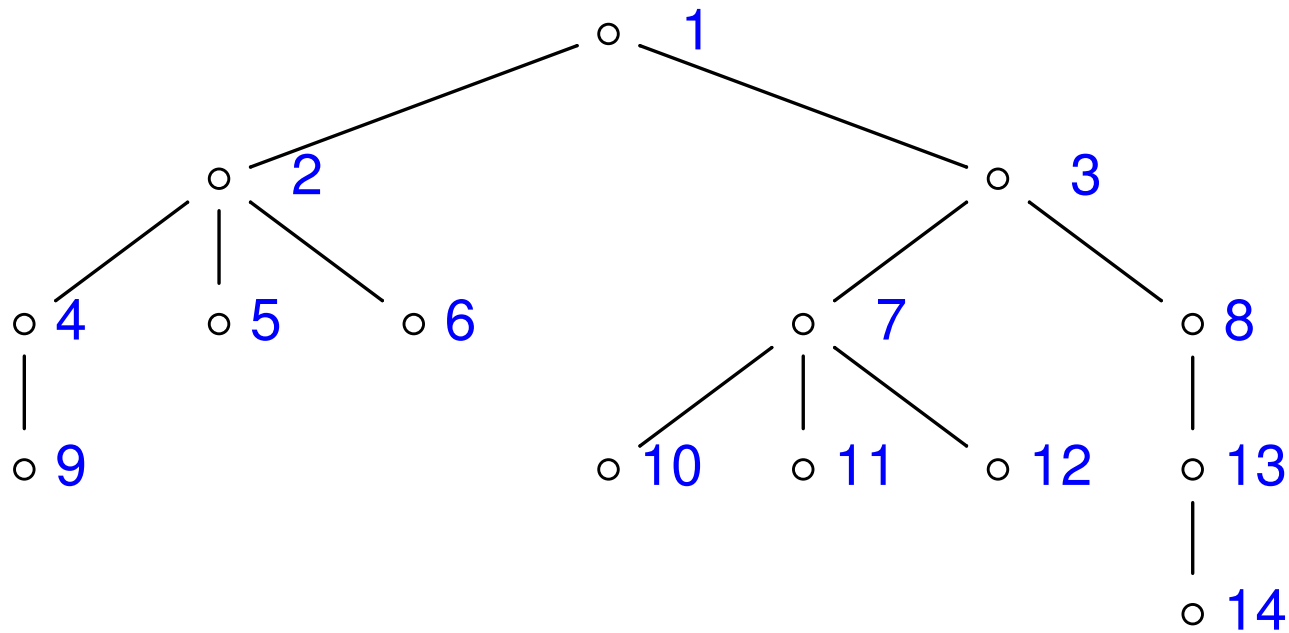
```
再帰的に探す (状態)
{
    if (見つかった)
        見つかった場合の処理;
    if (まだ続ける必要がある) {
        次の状態の各々について {
            状態を進める; 再帰的に探す (状態); 状態を戻す;
        }
    }
}
```

いつでも可能ではないが、可能なときは効果が大きい。

幅優先探索

同じ深さを順に調べ、それがつきたら次の深さに進むアルゴリズム。BFS と略す。

例：



幅優先探索の実装—キューを使用

```
探す ()
{
    初期状態をキューに入れる;
    while (まだ続ける必要がある && キューが空でない) {
        キューから状態を取り出す;
        if (見つかった) {
            見つかった場合の処理;
        }
        次の状態をすべてキューに入れる;
    }
}
```

幅優先探索の長所と短所

長所

- 実装が簡単。
- 深いパスがあっても OK。

短所

- 一般にメモリ効率が悪い。特に、大きな分岐があるとメモリを大量消費しやすい。

応用例—15 パズル

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

1〜15 を書いた正方形の板と穴 (★ で表示) が 4×4 にバラバラに配置されている。穴に隣の板を滑らせて入れることを繰り返して、初期状態（左図）にするパズル。

盤の配置を状態として BFS は簡単に実装できる。ただし、メモリを節約する工夫は必要。
盤の配置を状態として DFS だと、素朴な実装だと無限パスが生じるので、何らかの回避策が必要。
詳細は白版で