プロコン塾（2015/12/21）
深さ、幅優先探索の復習

制御情報3年 宮川大樹

今日の予定

- 深さ、幅優先探索の復習

- 再帰での深さ優先探索の考え方と実装を解説

【復習】 深さ、幅優先探索とは？

◆共通

- 木やグラフを探索するためのアルゴリズムである

- 全探索に分類される

  ―全探索とは？

  力まかせ探索（Brute-force Search）とも呼ばれる、もっとも単純な解を探索する方法。
  問題がとり得る全ての解を、しらみつぶしに調べていく。
  問題の規模によっては組み合わせ爆発を起こす。

【復習その2】深さ優先探索とは？

- Depth-first Search またはバックトラック法

- 探索方法

  探索対象となる木の最初のノードから、目的のノードが見つかるか子のないノードに行き着くまで、深く伸びていく

- 特徴

  ✓解がかなり早い段階で見つかることもある。しかし最短経路とは限らない

【復習その3】幅優先探索とは？

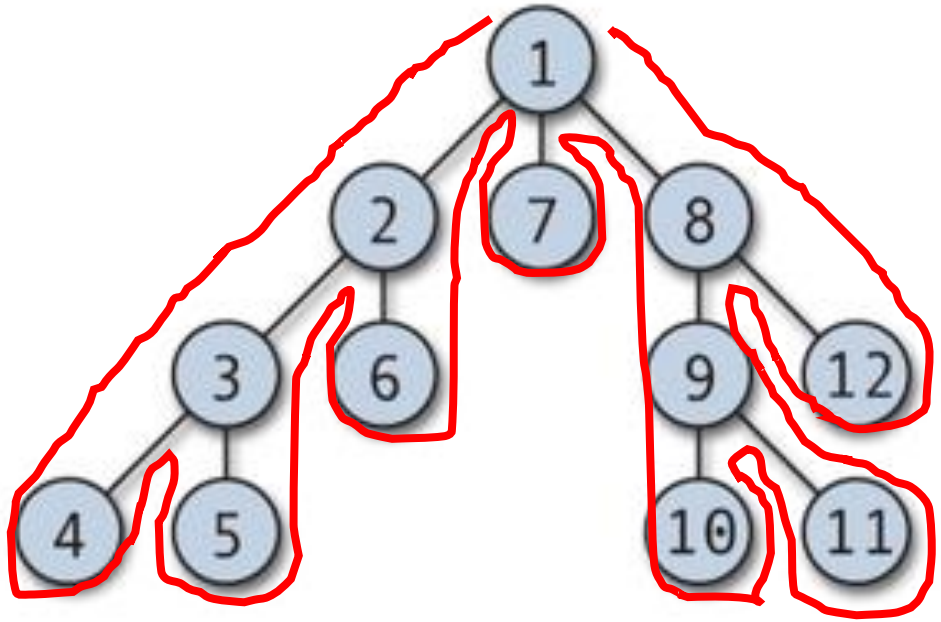
- Breadth-first Search とも呼ばれる

- 探索方法
  グラフの全てのノードを網羅的に展開・検査する

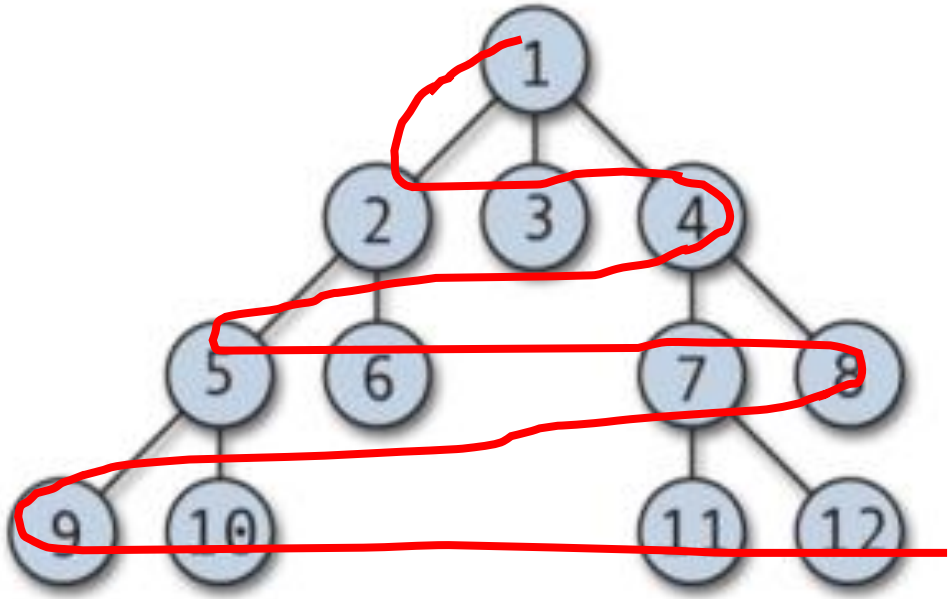
- 特徴
  ✓見つかった解は必ず最短経路である

走査法比較

深さ優先探索



幅優先探索



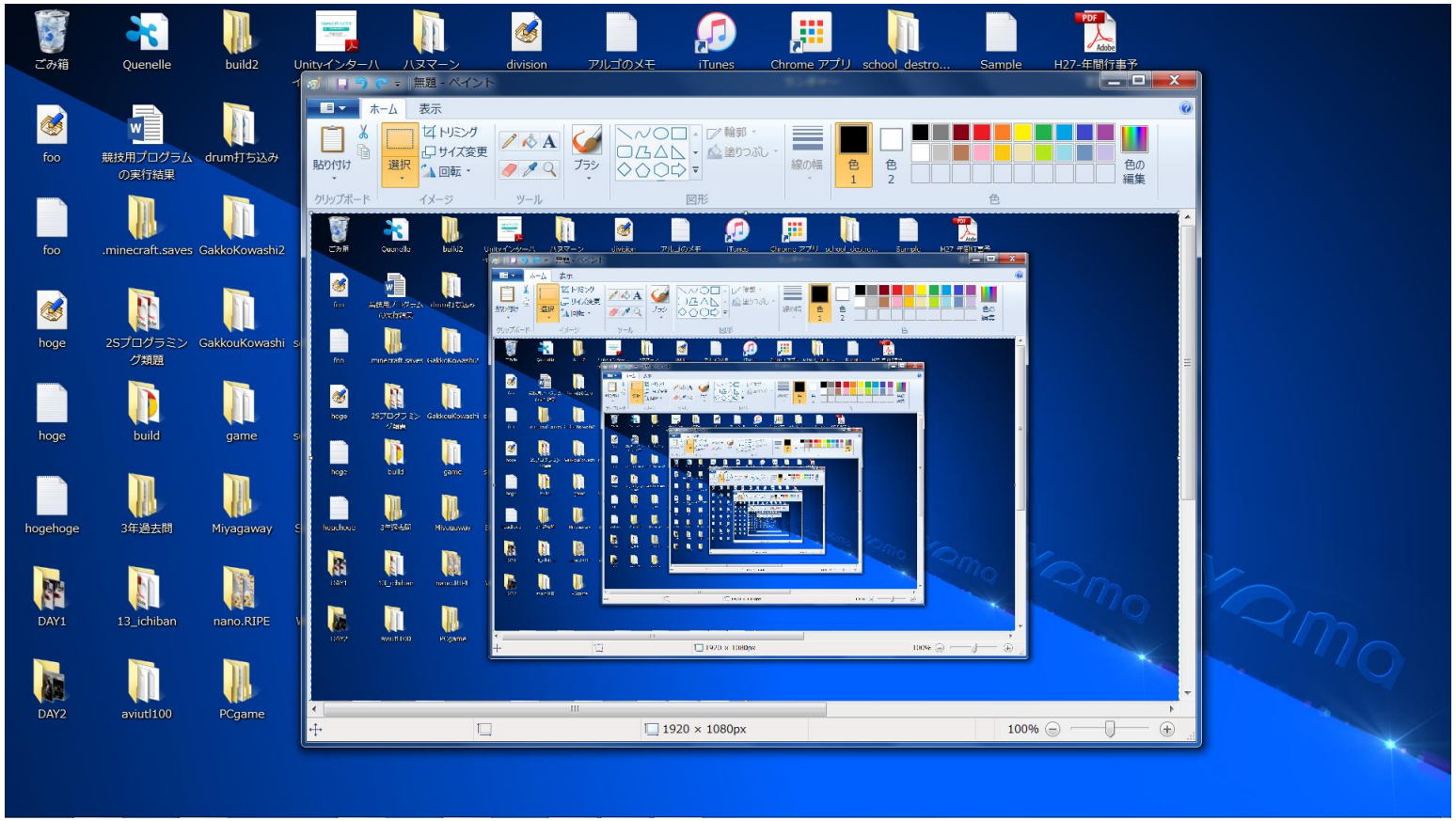
再帰による深さ優先探索

- 深さ優先探索は再帰関数を使って実装することもできる
  その他にスタックやfor文も使える

- 私的によいところ
  メソッド一つのため簡潔

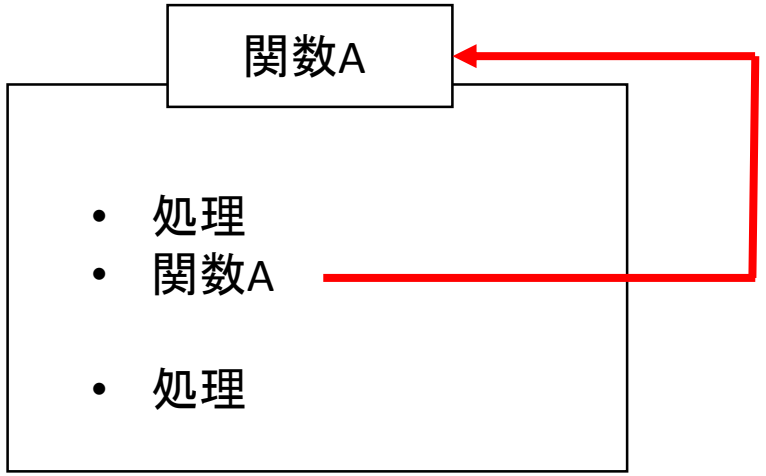
再帰とは

- ある事象が自分自身を含んでいること



再帰関数とは

- 自分自身を呼び出している関数。



上の例だと、関数A内で関数Aを呼び出している。

再帰関数の書き方（テンプレ）

```
E fanction() {
        if(条件) return;                    //←無限ループを防ぐ
        else return 再帰を使った処理;//←主にやって欲しい処理
}
```

- 詳しくは漸化式（数学）を勉強しよう

例:階乗

```
int factorial(int n) {
        if(n == 0) return 1;
        else return n * factorial(n-1);
}
```

漸化式

$$\begin{cases} a_n = 1 & (n = 0) \\ a_n = n * a_{n-1} & (n > 0) \end{cases}$$

今回解く問題 ： お小遣い使い切り問題

◆状況

- あなたはお小遣いをN円もらったので、買い物に出かけている
- 財布にはもらったお小遣いのみ入っていることとする
- 商品がM個あり、それぞれ$P_0, P_1, P_2, \cdots P_{M-1}$の値段である
- いくつかの商品を選んで買った場合、お小遣いを使いきって買い物することができるか（また使いきれた場合何円の商品を買ったか）を求める
- 同じ商品は複数買えないこととする

問題説明続き

◆渡されるデータの書式

M
P_0 P_1 P_2 …… P_M-2 P_M-1　　←改行区切り
N

◆出力の形式

Yes　　←使いきれた場合

No　　←使い切れなかった場合

解答の例その１

- 入力

  ４

  １　２　４　７

  １３

- 出力

  Yes

解答の例その２

- 入力

  ４

  １　２　４　７

  １５

- 出力

  No

問題の考え方

- それぞれの商品に対して、買うか買わないかの2択とする。それによって2分木を構成することができる

- 買う場合はお小遣いから代金を引く。残金が0であればそれを解とする

再帰関数を実装してみた

```
public static boolean recursion(int i, int rest) {        //①
    if(i == data.item_prices.length) {                    //②
        return rest == 0;
    }
    else {                                                //③
        return recursion(i+1, rest) || recursion(i+1, rest-data.item_prices[i]);
    }
}
```

言語化すると

recursion（i：添え字用，rest：残金）　{　　//①

　　基底部：葉ノードを参照していれば、残金の有無を返す

　　//②

　　一般項：　　//③

　　参照している商品を買わなかった場合の関数呼び出し

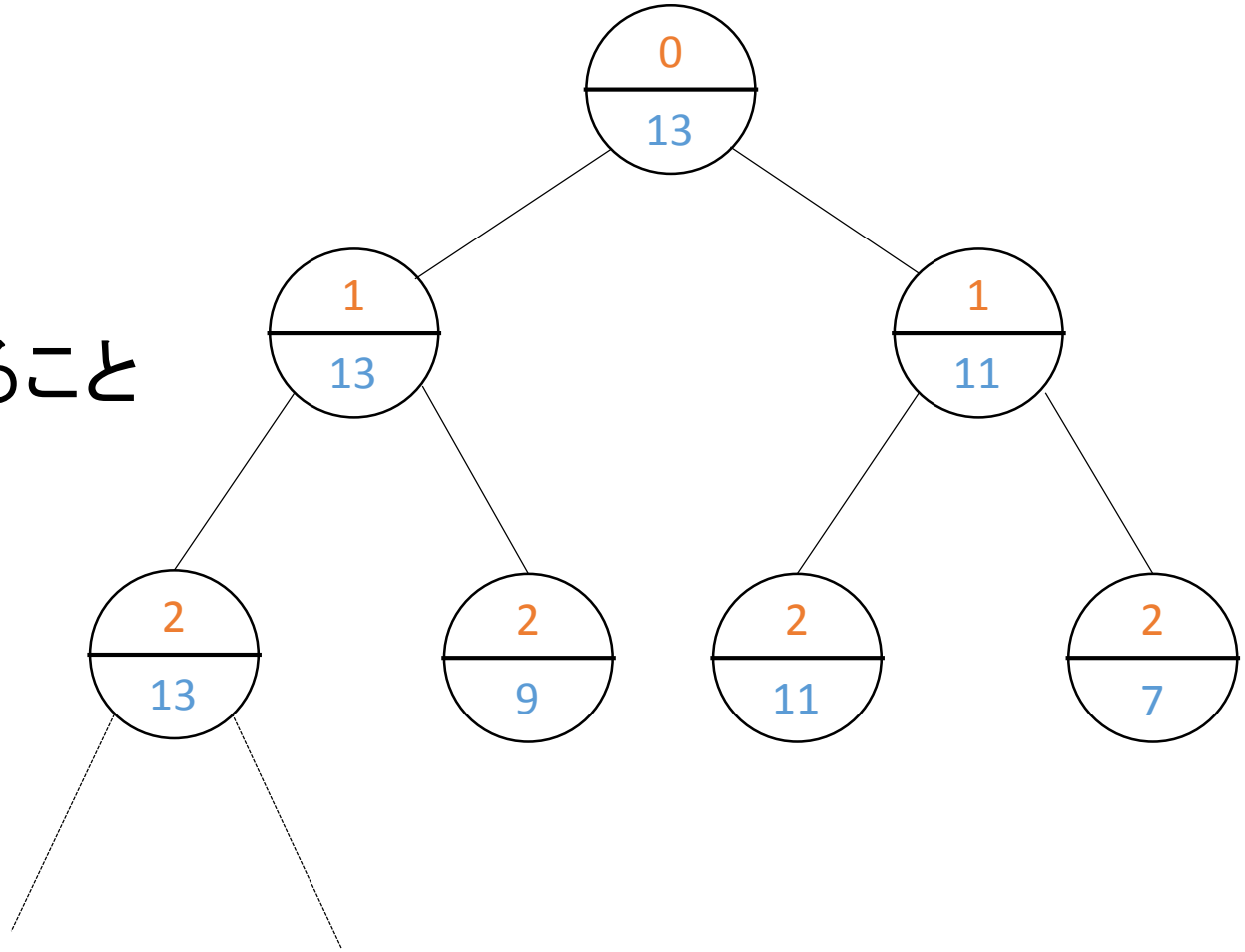
　　かつ

　　参照している商品を買った場合の関数呼び出し

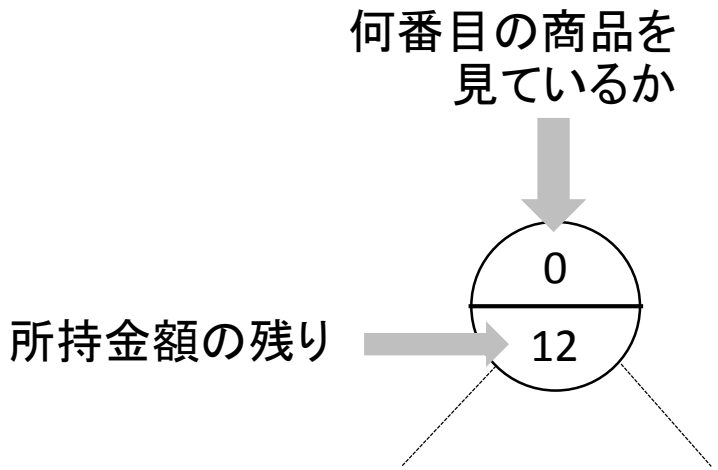
}

再帰の使い方

- 解答の例その1の問題を使用
- 再帰といえど木構造を意識することが重要
- ノード展開は右のような感じ



再帰の使い方：①の解説



- メソッドの構成

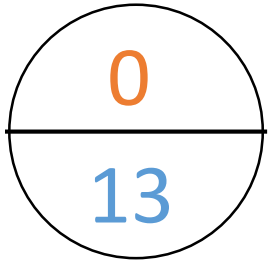
  引数は、

  ✓何番目の商品に注目しているか

  ✓現在の残金

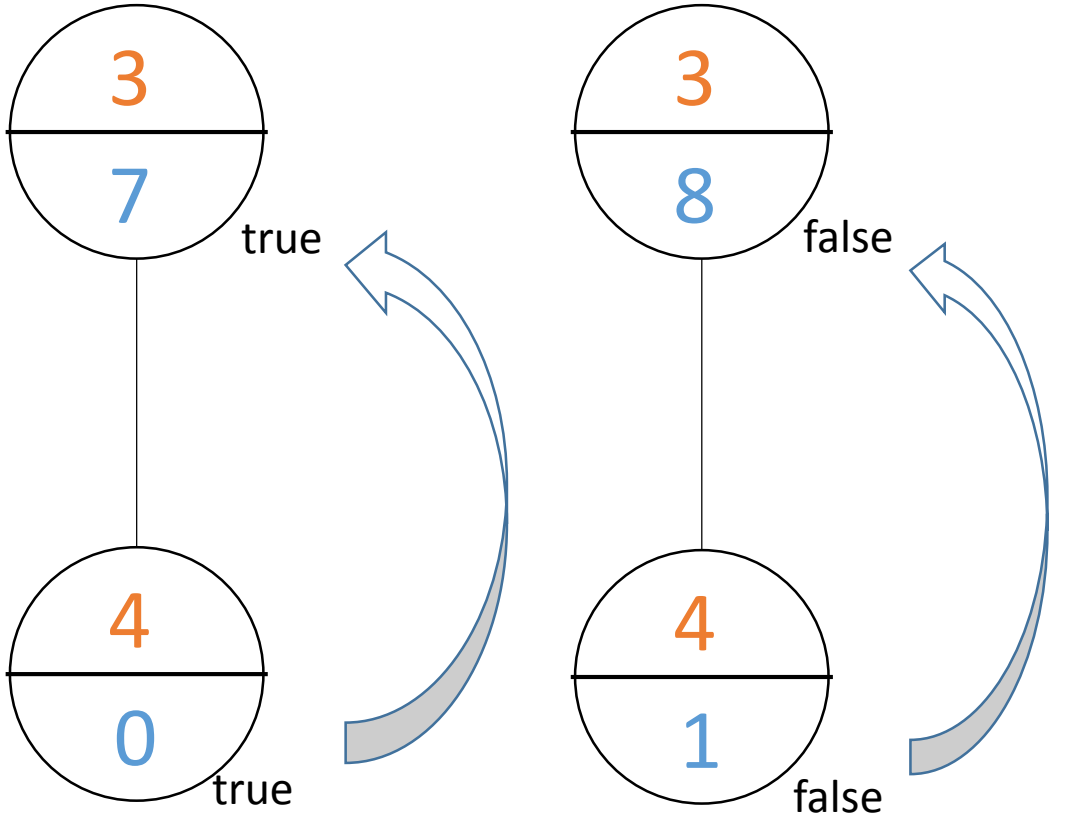
  とする。下の図だと、

  0番目の商品に注目し、現在の残金は13である



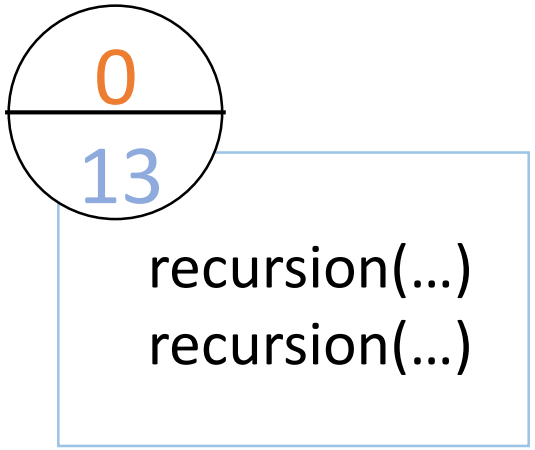
再帰の使い方：②の解説

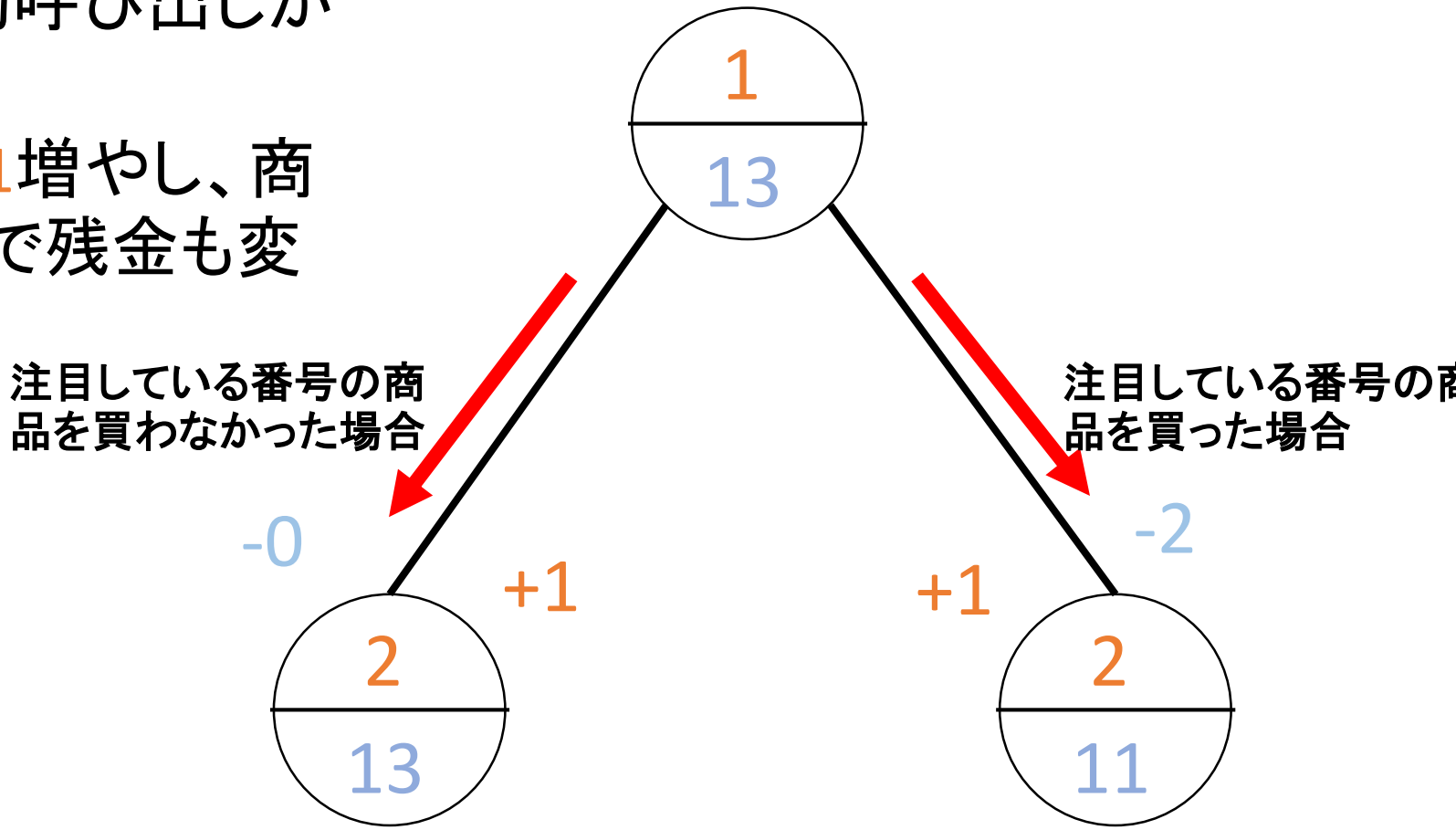
- 注目する番号が商品数（要素数）を超えたら基底部となる
- この場合、残金が0ならtrueを、そうでなければfalseを返す
- 上の階層に位置するメソッドにもその結果が受け継がれる



再帰の使い方：③の解説

- メソッドの中では再帰的呼び出しが２回行われている
- 呼び出すときは番号を1増やし、商品を買うか買わないかで残金も変動させる

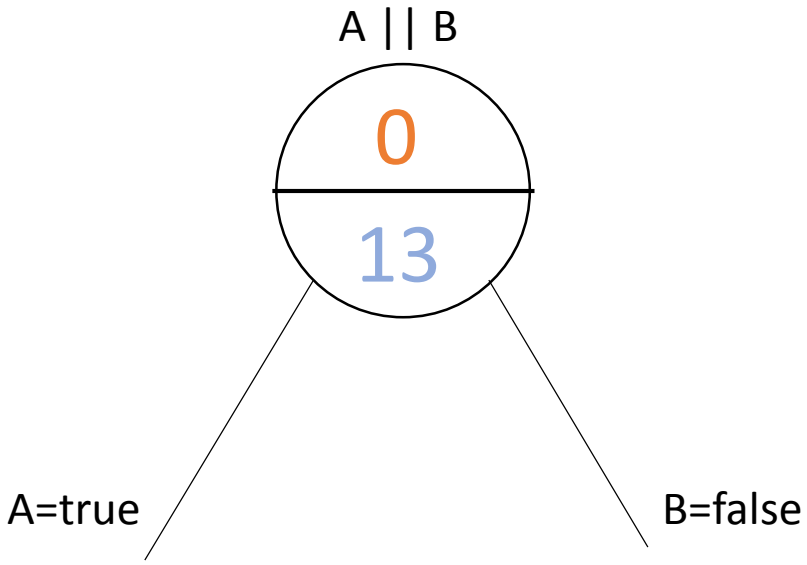




再帰の使い方：③の解説

- 末尾のノード以外は一つ階層がしたの子ノードたちから真偽を受け取る

- それをOR演算する（どちらか一方でもtrueならそれを残す）



コードの構成

◆Mainクラス

- Data data
- void main(String[] args)
- boolean recursion(int i, int rest)

◆Dataクラス

- int[] item_prices
- int pocket_money
- Data()
- String toString()

```
import java.util.Scanner;
class Main {
    //解答する問題のデータ
  static Data data;
    //再帰関数　漸化式の形で表す
  public static boolean recursion(int i, int rest) {
      //基底
    if(i == data.item_prices.length) {
     return rest == 0;
    }
      //一般項
    else {
            return recursion(i+1, rest) ||
            recursion(i+1, rest-data.item_prices[i]);
    }
```

```
public static void main(String[]args) {
    data = new Data();
    System.out.println(data.toString());
System.out.printf("%s%n",recursion(0,data.pocket_money)?"¥nYes":"¥nNo");
  }
}
```

```
class Data {
                        //商品の値段を持つ配列
  public final int[] item_prices;
                        //使える最大のお金
  public final int pocket_money;
                        //コンストラクタにて標準入力から与えられたデータを取得
  public Data() {
    Scanner stdIn = new Scanner(System.in);

    int[] a = new int[stdIn.nextInt()];

    for(int i=0;i<a.length;i++) {
      a[i] = stdIn.nextInt();
    }
    item_prices = a.clone();

    pocket_money = stdIn.nextInt();
  }
```

```
@Override
 public String toString() {
   String t = "";

   t += "number=" + item_prices.length + "¥n";

   t += "prices=";
   for(int i : item_prices)
     t += i + " ";
   t += "¥n";

   t += "pocket_money=" + pocket_money;

   return t;
  }
}
```

参考

- **Wikipedia大先生**
- **「再帰関数」～マンガでプログラミング用語解説**
  http://codezine.jp/article/detail/7265