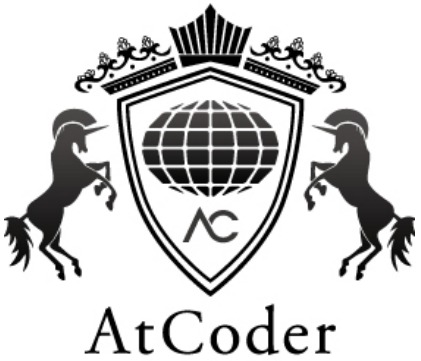
実践・最強最速のアルゴリズム勉強会
第三回　講義資料



AtCoder株式会社 代表取締役
高橋 直大

はじめに！

AtCoder

- 本講義では、ソースコードを扱います。
- 前面の資料だけでは見えづらいかもしれないので、手元で閲覧できるようにしましょう。
- URLはこちらから
  - http://www.slideshare.net/chokudai/wap-atcoder2
  - URLが打ちづらい場合は、Twitter: @chokudaiの最新発言から飛べるようにしておきます。
    - フォローもしてね！！！

目次

1. 勉強会の流れ
2. 計算量の概念
3. メモ化再帰
4. 動的計画法
5. 本日のまとめ

勉強会の流れ

1. 勉強会の日程
2. 1日の流れ

勉強会の日程

AtCoder

1日目　シミュレーションと全探索

2日目　色々な全探索

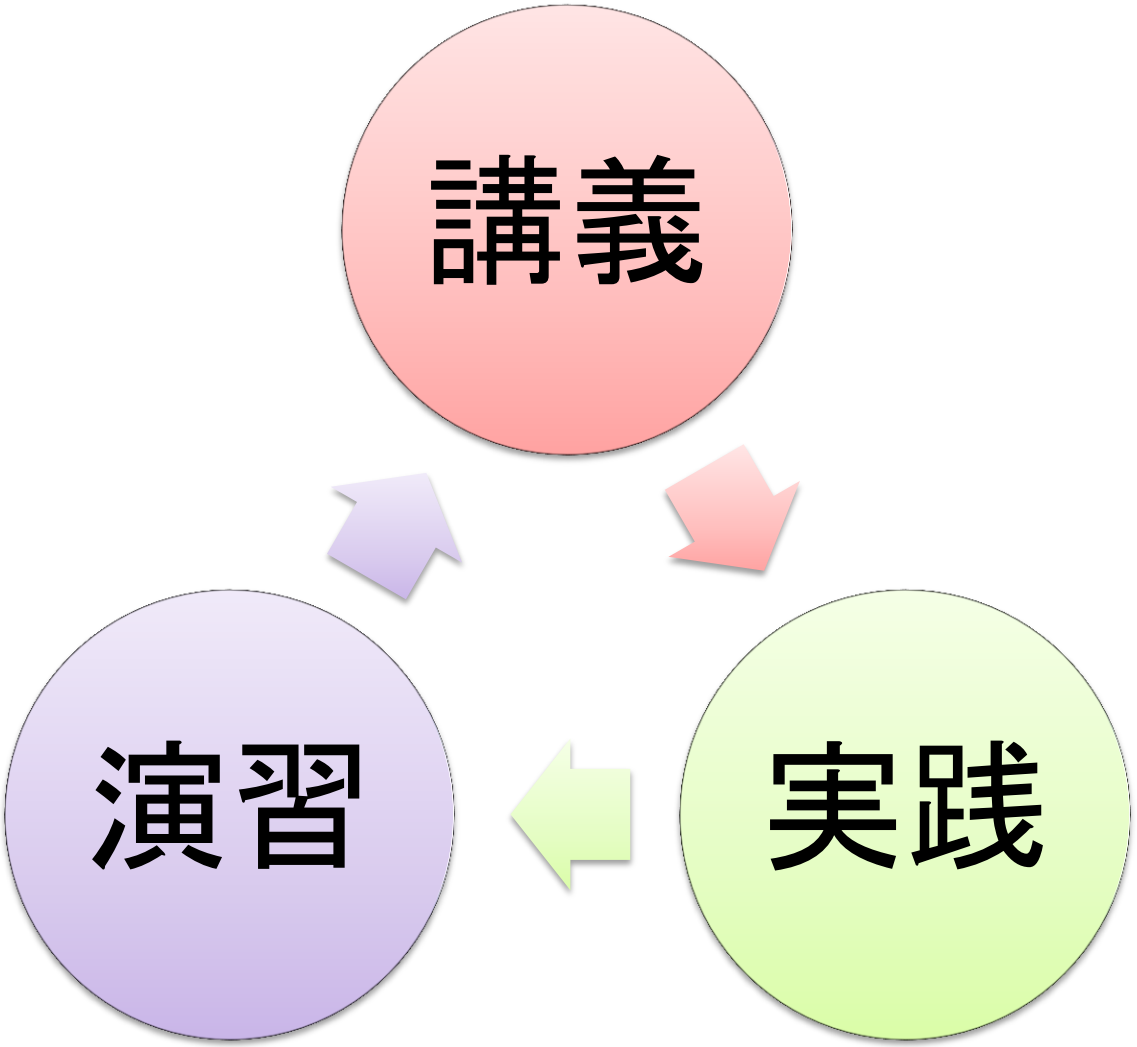
3日目　動的計画法とメモ化再帰

4日目　動的計画法と計算量を減らす工夫

5日目　難問に挑戦！

１日の流れ

AtCoder



１日の流れ

AtCoder

講義

- 基礎的なアルゴリズムを学ぶ
- 必要な知識を補う

実践

- 実際に問題例を見る
- コードでの表現方法を覚える

演習

- 自分で問題を解いてみる！
- コードを書いて理解を深める

１日の流れ

AtCoder

- 演習について
  - 実力差があると思うので、暇な時間が出来る人、ついていけない人、出ると思います。
    - どうしようもないので、早い人は支援に回ってもらえると嬉しいです！
  - 解らないことがあったら、#WAP_AtCoderでTwitterに投稿！
    - 多分早く終わった人が質問回答してくれます。
    - 具体例はこんな感じ
      - 「今やってる問題どれですか！」
      - 「この解答のWAが取れません！　http:// ~~」
      - 「コンパイルエラー出るよーなんでー＞＜　http:// ~~」
      - とりあえずコードが書けてたら、間違ってても提出してURLを貼りつけよう
  - もちろん、手を上げて質問してくれてもOK!
    - 回りきれる範囲では聞きに行きます。

今日の流れ

1. 前回までの復習
2. 今回やること

前回までの復習

AtCoder

- 全探索
  - 枝分かれの数が定数の場合 （第一回）
    - N重のforループによる全探索
  - 枝分かれの回数が不定の場合 （第二回）
    - 深さ優先探索
    - 幅優先探索
    - Bitを利用した深さ優先探索
      - 枝分かれが二股の時限定

今回やること

AtCoder

- 計算量の概念
  - 計算時間を見積もれるようにしよう！
- メモ化再帰
  - 深さ優先探索に対して、簡単に計算量を削減できる方法を覚えよう！
- 動的計画法
  - メモ化再帰の逆向き、動的計画法を使いこなそう！

今日の講義

1. 計算量の概念
2. メモ化再帰
3. 動的計画法
4. 本日のまとめ

計算量の概念

1. 計算時間を予測する必要性
2. 計算量とは？
3. 計算量を使用した実行時間の予測

計算時間を予測する必要性

AtCoder

- プログラムを書いた！　実行！
  - 答えが返ってこない・・・。
  - TLEになってしまった！
    - こんなことって、ありませんか？
- なぜ実行が終わらないのか？
    - バグ？
    - 正しくプログラムが組めていても、これだけ時間がかかってしまうアルゴリズムになっている？
  - 計算時間が予測出来ていないと、そもそもどうして実行時間がかかってしまっているのかが解らない

計算時間を予測する必要性

AtCoder

- もしアルゴリズムが間違っていた場合
  - プログラムを最初から書き直さないといけなくなる
  - 出来れば、プログラムを書く前に計算時間を予測したい

- とにかく、プログラムを書く前、プログラムを動かす前に、計算時間が解っていた方が、色々便利！

計算量とは？

AtCoder

- forループだけで出来るソースコードの場合
  - 例えば上記のようなコードでは、最も深いループの場所が、N * M * L回実行される。

```
int N = 100, M = 10, L = 1000;

for(int i=0;i<N;i++){
    for(int j=0;j<M;j++){
        for(int k=0;k<L;k++){
            //適当な処理
        }
    }
}
```

計算量とは？

AtCoder

- 計算量とは？
  - 大体の計算時間を予測するために使う概念
    - 本来は、「nやmが無限に大きくなった時に、どのように計算時間が増えていくか」を見るためのもの
  - 先ほどの様な処理は、O(NML)と表せる。
  - このNMLの最大の値を具体的に計算することにより、計算時間を予測することが出来る
    - 今回の場合は、100 * 10 * 1000 = 1,000,000

計算量とは？

AtCoder

- ループの回数と計算時間
  - 高速な言語の場合、おおよそ、1億回で1,2秒程度
    - 中の処理が本当に軽い処理の場合、10倍くらい早くなることも。

- 計算量の違いと、Nの変化による中身の変化

n	1	3	10	1000	10^8	10^16
O(logn)	1	1	3	10	30	60
O(√n)	1	1.7	3	31	10000	10^8
O(n)	1	3	10	1000	10^8	10^16
O(n^2)	1	9	100	10^6	10^16	10^32
O(2^n)	1	8	1000	10^300	莫大	莫大

計算量とは？

AtCoder

- ループの回数と計算時間
  - 高速な言語の場合、おおよそ、1億回で1,2秒程度
    - 中の処理が本当に軽い処理の場合、10倍くらい早くなることも。

- 計算量の違いと、Nの変化による計算時間

n	1	3	10	1000	10^8	10^16
O(logn)	一瞬	一瞬	一瞬	一瞬	一瞬	一瞬
O(√n)	一瞬	一瞬	一瞬	一瞬	0.001秒	1秒
O(n)	一瞬	一瞬	一瞬	一瞬	1秒	3年
O(n^2)	一瞬	一瞬	一瞬	0.01秒	3年	地球爆発
O(2^n)	一瞬	一瞬	一瞬	地球爆発	地球爆発	地球爆発

計算量とは？　実践

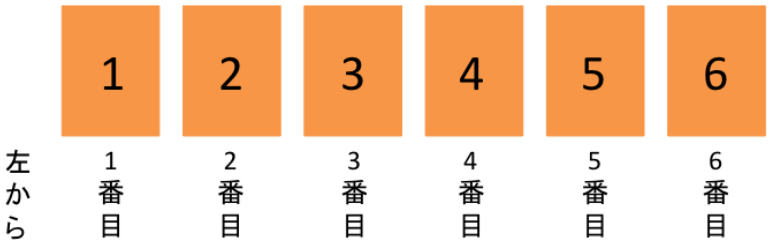
AtCoder

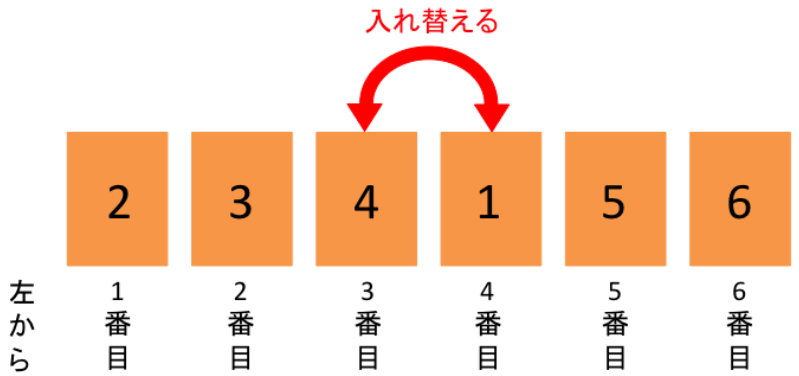
- ABC004 C問題　入れ替え
  - http://abc004.contest.atcoder.jp/tasks/abc004_3

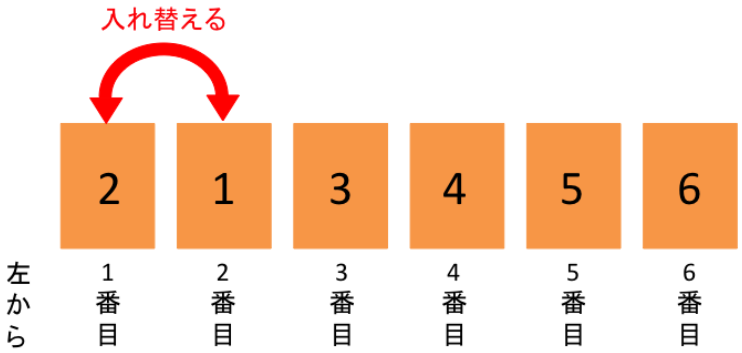
- 問題概要
  - 指定された順番でカードを入れ替える
  - n回入れ替えた時に、どのようなカードの並びになるか？
    - Nは1,000,000,000回まで！　←多い！

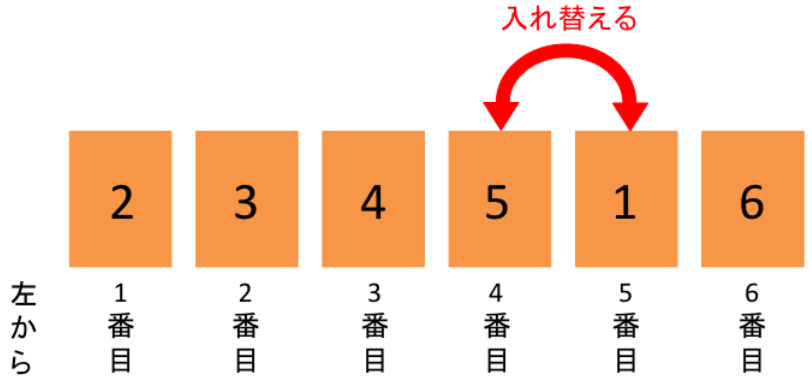
繰り返しのあるシミュレーション問題　おまけ

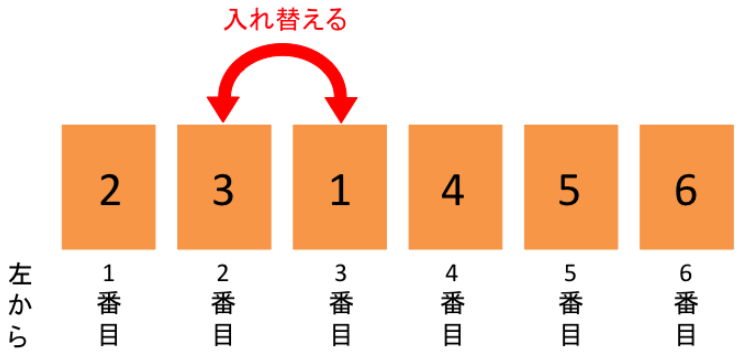
AtCoder

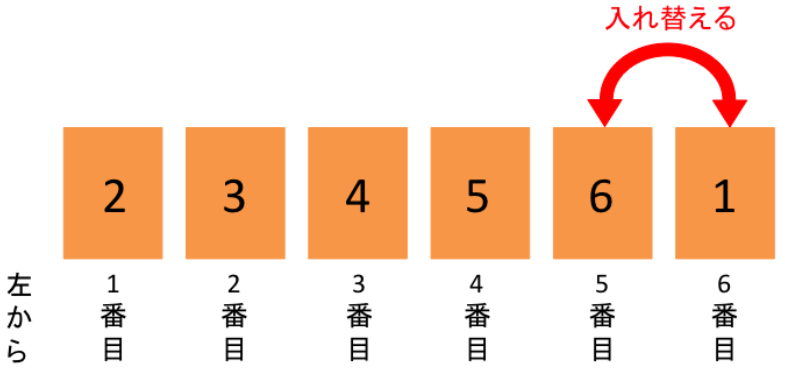












計算量とは？　実践

AtCoder

- 単純に、言われた回数を入れ替えるだけだと？
  - N回のループを回す
  - 計算量はO(N)になる
    - Nは最大1,000,000,000
    - 大体10秒くらい？
      - ちょっと間に合わなさそう

- 計算時間が予測出来れば、非常に便利！
  - 例えば、Nの最大値が10,000,000だったら、上記の解法で解けそう、などが解る。
    - 高精度ではないので、100,000,000だと微妙

計算量とは？

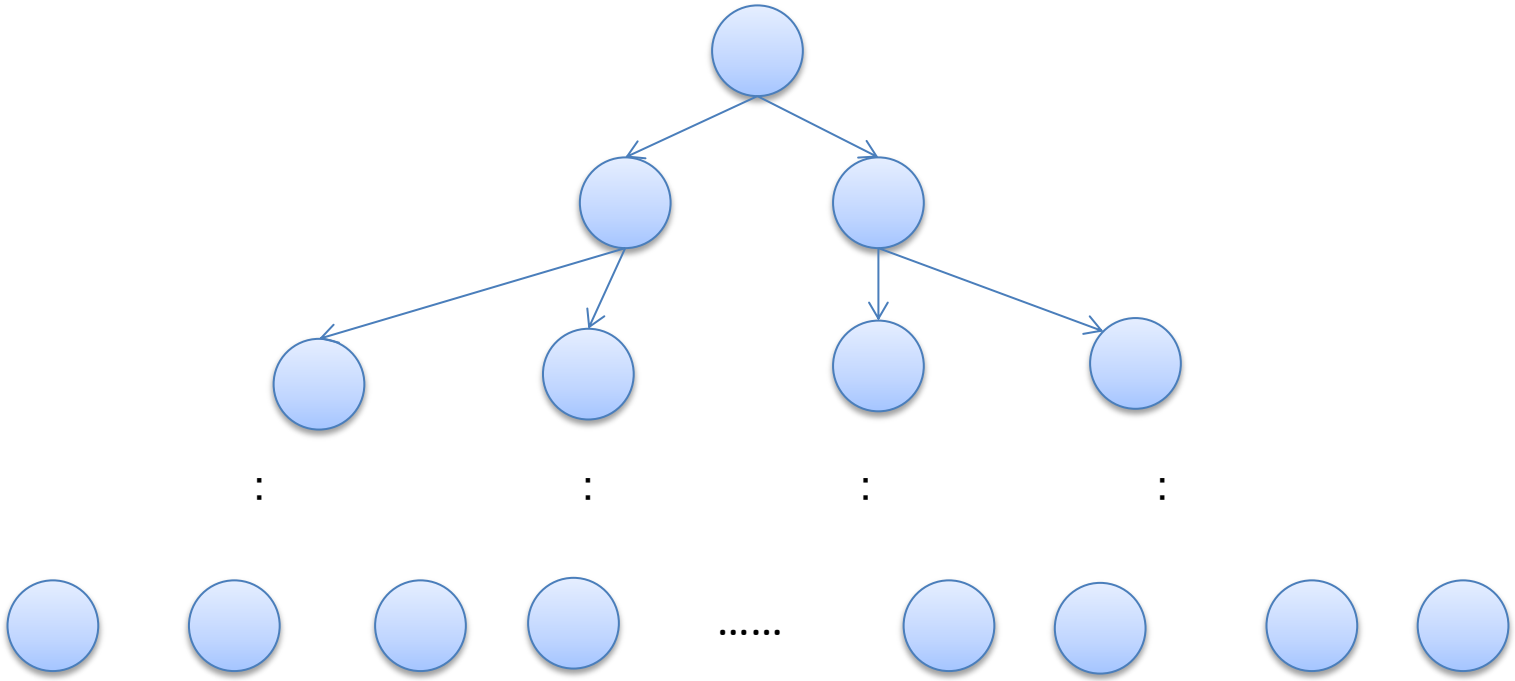
AtCoder

- forループ以外の探索の計算量
  - これは見積もりが物凄く難しい！
- 一応、幾つか見積もる方法はないわけではない。

計算量とは？

AtCoder

- 分岐の数が一定などで、一番下の状態の数が予測可能な時
  - 計算量は、O(最終状態数 * 最終状態までの深さ)くらい



計算量とは？

AtCoder

- 凄く大雑把な見積もりをする場合
  - 分岐が最大P個、深さが最大Aなどの場合
  - 計算量はO(P * A^P)とかで見積もれる
    - これは大きすぎる見積もりとなる場合が多い

- 見積もりが不可能な場合は、大抵は全探索出来ない程度に大きい場合が多い
  - 後で説明する、メモ化再帰などを使った方が、計算量が落ちる上、見積もりが簡単になる

メモ化再帰

1.深さ優先探索の限界
2.メモ化再帰

深さ優先探索の限界

AtCoder

- 深さ優先探索には、限界がある
  - 分岐の数などが多い場合に、実行時間が非常に長くなってしまう
- これを解決するのがメモ化再帰！

メモ化再帰

AtCoder

- メモ化再帰って？
  - 深さ優先探索の計算結果をメモ、再利用することによって、不必要な探索をしないようにするアルゴリズム
    - ここで言う「不必要な探索」とは、すでに探索を行っていて、同じ結果が得られることが容易に予想できる探索のこと
  - 不必要な探索って、例えばどんなのだろう？

メモ化再帰

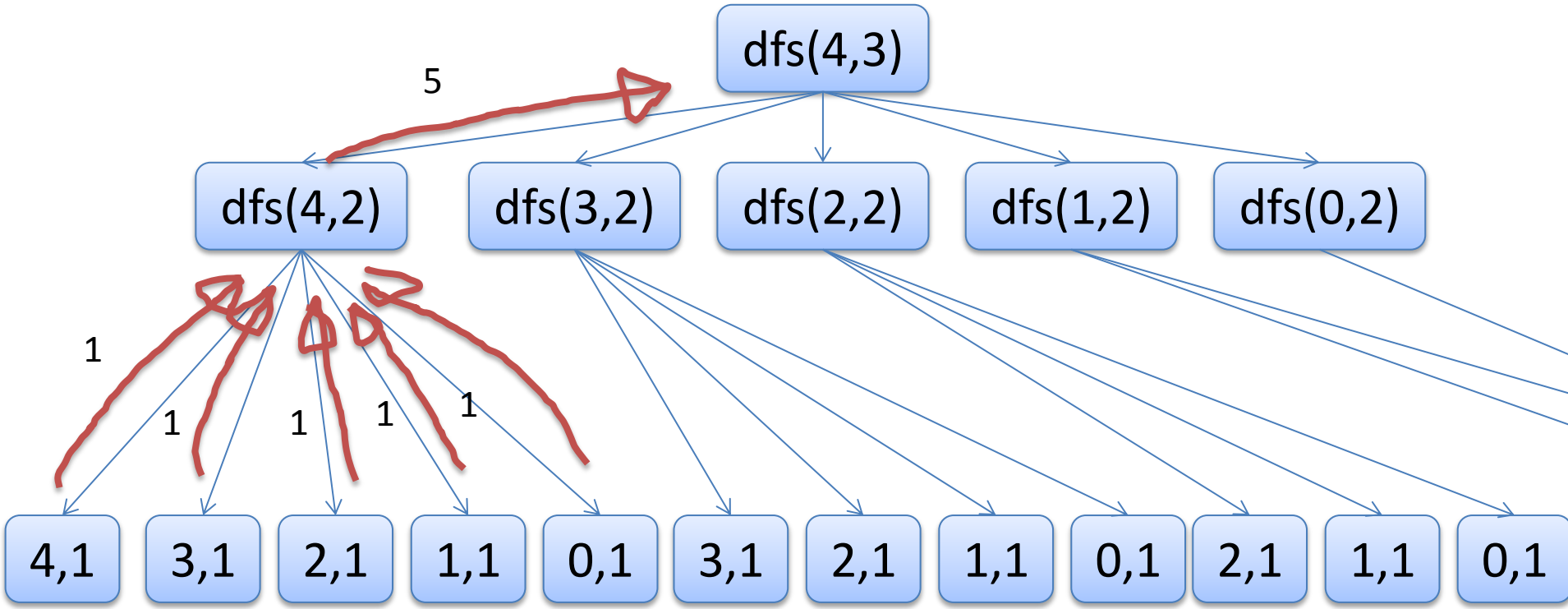
AtCoder

- 問題例
  - N人のりんごをM人で分けます。分け方が何通りあるか答えなさい。
  - 深さ優先探索では、全てのパターンを列挙した。

メモ化再帰

AtCoder

- 例：4つのりんごを3人で分ける。



メモ化再帰

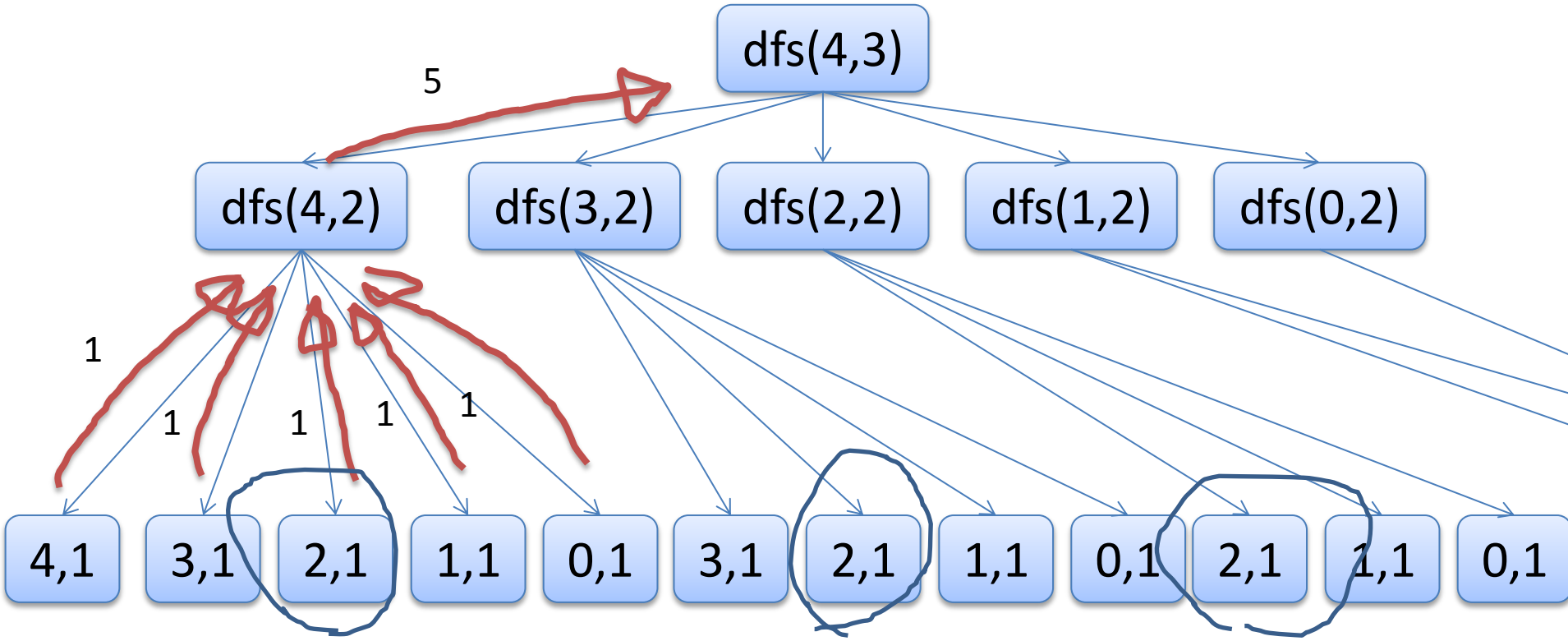
AtCoder

- 深さ優先探索の限界
  - 答えを1個ずつ列挙している
    - 答えが1億以上となる問題の場合、1億個以上のパターンを列挙している
    - つまり、ループ回数は1億を超えてしまう
      - よって、制限時間に間に合わない
  - メモ化再帰を使えば、この問題を解消できる

メモ化再帰

AtCoder

- 例：4つのりんごを3人で分ける。



メモ化再帰

AtCoder

- 図のように、dfs(2,1)を何度も呼び出している
  - 何度呼び出されても、dfs(2,1)が返す値は変わらない
    - なぜなら、dfs(N,M)は、「N人のリンゴをM人で分ける時の組み合わせ」を必ず返す関数として作成しているから。
  - であれば、dfs(2,1)の値をメモしておけば、dfs(2,1)の値は、2回目から計算する必要がない。

- これは、他のdfs(A, B)に対しても言える

- 実装例
  - 前回の深さ優先探索のソースコード

```
int dfs(int m, int n){
    if(n==1) return 1;
    int ret = 0;
    for(int i=0;i<=m;i++){
        ret += dfs(m-i, n-1);
    }
    return ret;
}
```

メモ化再帰

AtCoder

- 実装例
  - メモ化再帰のソースコード

```
int[][] dp;
int dfs(int m, int n){
    if(dp[m][n] != 0) return dp[m][n];
    if(n==1) return dp[m][n] = 1;
    int ret = 0;
    for(int i=0;i<=m;i++){
        ret += dfs(m-i,n-1);
    }
    return dp[m][n] = ret;
}
```

AtCoder

- 実装例
  - メモ化再帰のソースコード

答えをメモをするための配列を予め用意しておく

```
int[][] dp;
int dfs(int m, int n){
    if(dp[m][n] != 0) return dp[m][n];
    if(n==1) return dp[m][n] = 1;
    int ret = 0;
    for(int i=0;i<=m;i++){
        ret += dfs(m-i,n-1);
    }
    return dp[m][n] = ret;
}
```

- 実装例
  - メモ化再帰のソースコード

```
int[][] dp;
int dfs(int m, int n){
    if(dp[m][n] != 0) return dp[m][n];
    if(n==1) return dp[m][n] = 1;
    int ret = 0;
    for(int i=0;i<=m;i++){
        ret += dfs(m-i,n-1);
    }
    return dp[m][n] = ret;
}
```

答えを返す時に、
必ずメモ用の配列に
答えを入力しながら
返すようにする

メモ化再帰

AtCoder

- 実装例
  - メモ化再帰のソースコード

```
int[][] dp;
int dfs(int m, int n){
    if(dp[m][n] != 0) return dp[m][n];
    if(n==1) return dp[m][n] = 1;
    int ret = 0;
    for(int i=0;i<=m;i++){
        ret += dfs(m-i,n-1);
    }
    return dp[m][n] = ret;
}
```

答えが既に求まっていればその答えをそのまま返す。
今回は0を初期値として利用したが、毎回0で良いわけではない。その数になり得ない数を入れる。
そうした数が存在しない時は既に求めたかのチェック用の配列を別に用意する。

メモ化再帰

AtCoder

- メモ化再帰で何をメモすれば良いか？
  - その関数で求めるべきもの！
  - 最小値・最大値を求める問題
    - その引数に対する最小値・最大値をメモする
  - 組み合わせの個数を求める問題
    - その引数に対する組み合わせの個数をメモする
  - 他にも何パターンか存在するが、基本的には、求めたいものを返す関数を作って、その結果をそのままメモすれば良い！

メモ化再帰　実践

AtCoder

- 天下一プログラマーコンテスト2012決勝 A問題
  - http://tenka1-2012-final.contest.atcoder.jp/tasks/tenka1_2012_final_a
    - 難しい方は解けないので、部分点だけ解きましょう。

- 問題概要
  - 整数n (1 <= n <= 10^5)が与えられる
  - フィボナッチ数列に出てくる数字の和だけでnを作りたい。
  - 使う数字の数を出来るだけ減らしたい時、使う必要のある数字の数の最小数を求めなさい。
    - 同じ数を2回使っても、2つとカウントします。

メモ化再帰　実践

AtCoder

- フィボナッチ数って？
  - F(n) = F(n-1) + F(n-2)となる数列
    - 簡単に言うと、1つ前の数字と2つ前の数字を足した数字が次の数字になるよ、って数列
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233… みたいな感じ。
    - 最初の2つだけは、0,1と決まっている。あとは計算で求める

- 具体例
  - N = 10だと、5 + 5で表せるので、答えは2
  - N = 11でも、8 + 3で表せるので、答えは2

メモ化再帰　実践

AtCoder

- まずはk番目のフィボナッチ数を求める
  - せっかくなので、これもメモ化再帰で求めてみる
- 適当にフィボナッチ関数を、深さ優先探索で書くとこんな感じ
  - 0や1の2つ目が出てきても仕方ないので、1,2から始める

```
int getFib(int k){
    if(k==0) return 1;
    if(k==1) return 2;
    return getFib(k-1) + getFib(k-2);
}
```

メモ化再帰　実践

AtCoder

- さっきのコードをメモ化再帰にしてみる

```
int[] fibdp;
int getFib(int k){
    if(k==0) return 1;
    if(k==1) return 2;
    if(fibdp[k] != 0) return fibdp[k];
    return fibdp[k] = getFib(k-1) + getFib(k-2);
}
```

メモ化再帰　実践

AtCoder

- さっきのコードをメモ化再帰にしてみる

答えをメモするための配列

```
int[] fibdp;
int getFib(int k){
    if(k==0) return 1;
    if(k==1) return 2;
    if(fibdp[k] != 0) return fibdp[k];
    return fibdp[k] = getFib(k-1) + getFib(k-2);
}
```

メモ化再帰　実践

AtCoder

- さっきのコードをメモ化再帰にしてみる

```
int[] fibdp;
int getFib(int k){
    if(k==0) return 1;
    if(k==1) return 2;
    if(fibdp[k] != 0) return fibdp[k];
    return fibdp[k] = getFib(k-1) + getFib(k-2);
}
```

返す答えを配列にメモする

メモ化再帰　実践

AtCoder

- さっきのコードをメモ化再帰にしてみる

```
int[] fibdp;
int getFib(int k){
    if(k==0) return 1;
    if(k==1) return 2;
    if(fibdp[k] != 0) return fibdp[k];
    return fibdp[k] = getFib(k-1) + getFib(k-2);
}
```

答えがメモ済みであれば返す！

メモ化再帰

AtCoder

- フィボナッチ数を求める計算量は？
  - k番目のフィボナッチ数を求めるために探索をする回数は、各々1回しか存在しない。
  - よって、フィボナッチ数の数の分の計算量しかかからない。
    - 2回同じものを呼び出した場合、既に求まっているので、配列に入った答えを返すだけ。

- ちなみに、メモ化再帰なしだと、帰ってくる数値分の探索はおおよそしている、と推定できる。
  - すぐに非常に大きな数になってしまう

メモ化再帰　実践

AtCoder

- 次に、求めたい答えを探索する。
  - ある数字kを作るために必要な手数を、深さ優先探索で求める再帰関数dfsを作る！

```
int dfs(int k){
    if(k==0) return 0;
    //適当に大きい数を入れておく
    int ret = 999999;
    for(int i=0; i<fibdp.length;i++){
        //もし、i番目のフィボナッチ数が、目的の数を超えていたらbreak
        if(k < getFib(i)) break;
        //k-(i番目のフィボナッチ数)に対して探索を行う
        ret = Math.min(ret, dfs(k - getFib(i)) + 1);
    }
    return ret;
}
```

メモ化再帰　実践

AtCoder

- この深さ優先探索の計算量は？
  - よくわからない！
  - けど莫大になりそう！
    - という感覚だけ持っていれば良いです
  - 計算量が良くわからない時は、大体凄く多いです。

メモ化再帰　実践

AtCoder

- これをメモ化再帰に改造する

```
int[] dp;
int dfs(int k){
    if(k==0) return 0;
    if(dp[k] != 0) return dp[k];
    int ret = 999999; //適当に大きい数を入れておく
    for(int i=0; i<fibdp.length;i++){
        //もし、i番目のフィボナッチ数が、目的の数を超えていたらbreak。
        if(k < getFib(i)) break;
        //k-(i番目のフィボナッチ数)に対して探索を行う
        ret = Math.min(ret, dfs(k - getFib(i)) + 1);
    }
    return dp[k] = ret;
}
```

メモ化再帰　実践

AtCoder

- これをメモ化再帰に改造する
  - 変更した箇所はたったこれだけ！

```
int[] dp;
int dfs(int k){
    if(k==0) return 0;
    if(dp[k] != 0) return dp[k];
    int ret = 999999; //適当に大きい数を入れておく
    for(int i=0; i<fibdp.length;i++){
        //もし、i番目のフィボナッチ数が、目的の数を超えていたらbreak。
        if(k < getFib(i)) break;
        //k-(i番目のフィボナッチ数)に対して探索を行う
        ret = Math.min(ret, dfs(k - getFib(i)) + 1);
    }
    return dp[k] = ret;
}
```

メモ化再帰　実践

AtCoder

- メモ化再帰にしたときの計算量は？
  - N個の数に対して、探索する回数は1回だけ
  - 100000以下のフィボナッチ数はせいぜい20個くらい
  - よって、計算回数は、20*N程度で済む

  - これなら計算が間に合う！

メモ化再帰　実践

AtCoder

- ソースコード
  - http://tenka1-2012-final.contest.atcoder.jp/submissions/148406

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int n = cin.nextInt();
    if(n > 100000) return; //TLE回避
    fibdp = new int[100]; //適当に大きい数
    dp = new int[n+1]; //nより大きい数
    System.out.println(dfs(n));
}
```

メモ化再帰　演習

AtCoder

- 第2回早稲田大学プログラミングコンテスト B問題
  - http://wupc2nd.contest.atcoder.jp/tasks/wupc_02

- 問題概要
  - 水たまりのあるマスと、ないマスが存在する
    - .X.XXXXXX.のような、文字列で情報が与えられる
      - Xは水たまりのある場所
      - .は水たまりのない場所
  - 出来る行動は、1～3マス右に進む
  - 一番左から一番右のマスに移動する時、水たまりに入らなければならない最小値を出力せよ。

メモ化再帰　演習

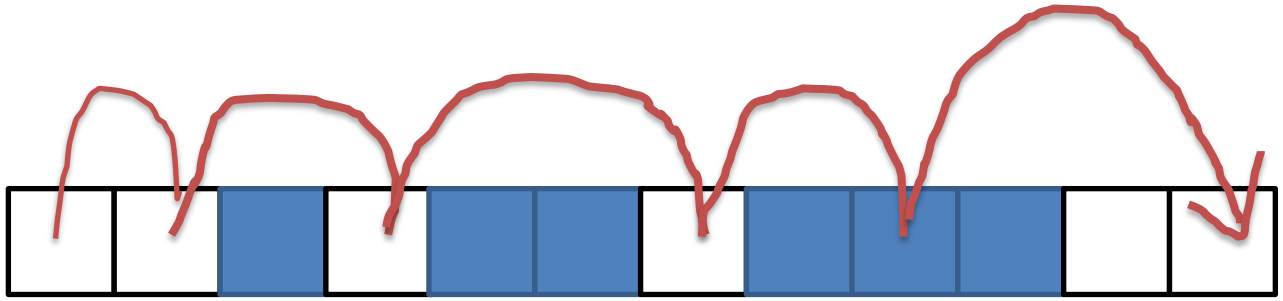
AtCoder

- 問題のイメージ
  - 例えば以下のような水たまりが与えられる

メモ化再帰　演習

AtCoder

- 問題のイメージ
  - 例えば以下のような水たまりが与えられる
  - 以下のように移動すれば、1回の水たまりでゴールすることが出来る



メモ化再帰　演習

AtCoder

- 探索をすると・・・？
  - 区間は100以下
  - この区間に対して、1,2,3個の3つの選択肢の移動が考えられる。
  - 大きく見積もって、O(3^N)
  - 適当に深さの平均がN/2になりそうであるという予測を立てても、探索回数が3^(N/2)くらいになるので、莫大な数になってしまう
  - メモ化再帰にする必要があります！

メモ化再帰　演習

AtCoder

- 注意点
  - 今回は、答えとして、0が帰ってくることがあり得る
  - 0で初期化してしまうと、正常にメモすることが出来ない。
  - 配列の使ってない部分は、予め-1などで初期化しておこう

メモ化再帰　演習

AtCoder

- 出来た人は以下の問題にもチャレンジ！
  - Recruit Programming Contest 模擬練習会 B問題
    - http://recruit-programing-contest-practice.contest.atcoder.jp/tasks/recruite_2013_pre_b
  - Typical DP Contest C問題
    - http://tdpc.contest.atcoder.jp/tasks/tdpc_tournament

メモ化再帰　演習　ヒント

AtCoder

- 作るべき関数
  - あと残り何マスの時に、あと何回の水たまりに入るとゴール出来るか？
    - » 今いるマスが何マスの時に、でも良い。
    - » 解説はそっちで実装します。
  - 今いる場所が水たまりである場合、それも含めた関数にする方が組みやすい

メモ化再帰　演習　解説

AtCoder

- kマス目にいるときに、残りの水たまりに入らないといけない回数を求める関数int dfs(int k)を作る！

```
String S;
int N;

int dfs(int k){
    if(k >= N - 1) return 0;
    int ret = 99999999;
    int temp = 0;
    //その地点が水たまりなら、回数を1増やす
    if(S.charAt(k) == 'X') temp = 1;
    //1から3の3種類の探索を行う
    for(int i=1;i<=3;i++){
        ret = Math.min(ret, dfs(k+i) + temp);
    }
    return ret;
}
```

メモ化再帰　演習　解説

AtCoder

- これをメモ化再帰に改造する！

```
String S;
int N;
int[] dp;

int dfs(int k){
    if(k >= N - 1) return 0;
    if(dp[k] != -1) return dp[k];
    int ret = 99999999;
    int temp = 0;
    //その地点が水たまりなら、回数を1増やす
    if(S.charAt(k) == 'X') temp = 1;
    //1から3の3種類の探索を行う
    for(int i=1;i<=3;i++){
        ret = Math.min(ret, dfs(k+i) + temp);
    }
    return dp[k] = ret;
}
```

メモ化再帰　演習　解説

AtCoder

- これをメモ化再帰に改造する！
  - 変わった箇所は以下の箇所

```
String S;
int N;
int[] dp;

int dfs(int k){
    if(k >= N - 1) return 0;
    if(dp[k] != -1) return dp[k];
    int ret = 99999999;
    int temp = 0;
    //その地点が水たまりなら、回数を1増やす
    if(S.charAt(k) == 'X') temp = 1;
    //1から3の3種類の探索を行う
    for(int i=1;i<=3;i++){
        ret = Math.min(ret, dfs(k+i) + temp);
    }
    return dp[k] = ret;
}
```

- ソースコード
  - http://wupc2nd.contest.atcoder.jp/submissions/148408

```
void run()
{
    Scanner cin = new Scanner(System.in);
    N = cin.nextInt();
    S = cin.next();
    dp = new int[N];
    //予め-1で初期化する
    for(int i=0;i<N;i++) dp[i] = -1;
    //探索で求めた答えを出力
    System.out.println(dfs(0));
}
```

メモ化再帰　演習　解説

AtCoder

- メモ化再帰の注意点
  - 元々は深さ優先探索なので、メモ化が上手くいってなくても、小さいケースでは正しい答えを出してしまう。
    - よって、大きいケースでTLEするコードになってしまっていても、なかなか気づけない
  - 提出前に、必ず大きな入力のテストをするようにしよう！

休憩！

動的計画法

1.動的計画法って？
2.メモ化再帰と深さ優先探索の関係
3.動的計画法

動的計画法って？

AtCoder

- これまでの結果を全て纏めておくことにより、同じ探索を2度しないアルゴリズム
  - 略称はDP
  - といっても、イメージは付きづらい？
- 具体例を見たほうが早い！

動的計画法って？

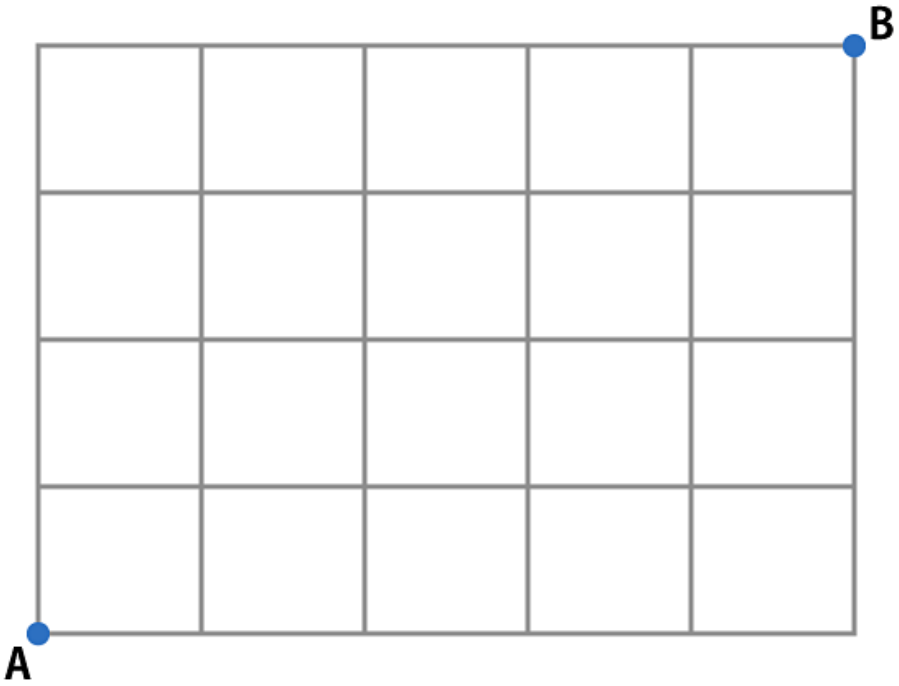
AtCoder

- フィボナッチ数列を求めるアルゴリズム
  - f(0) = 1, f(1) = 1とする
  - f(2) = f(0) + f(1) = 2
  - f(3) = f(1) + f(2) = 3
  - ….
- 前の状態を予め計算しておくことで、次の値をすぐに求めることが可能になる。

動的計画法って？

AtCoder

- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする



動的計画法って？

AtCoder

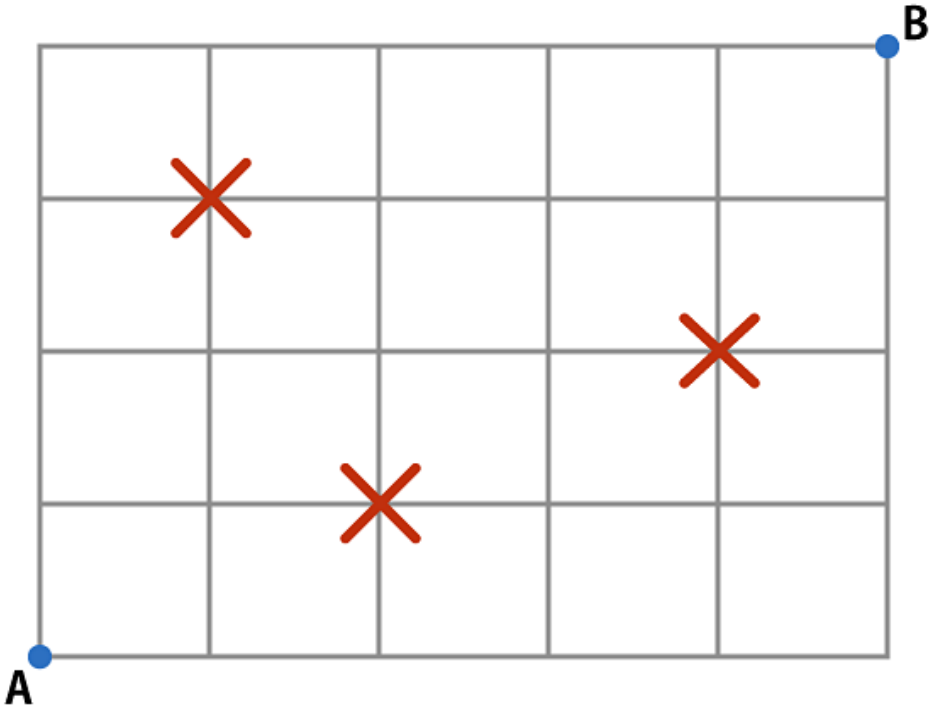
- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする

- たてに4回、よこに5回で9回の移動
  - 9回から4回を選べば良いので、9C4回
    - 数学的に求まる

動的計画法って？

AtCoder

- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
    - 通れない道があるので、数学的に求まらない！



動的計画法って？

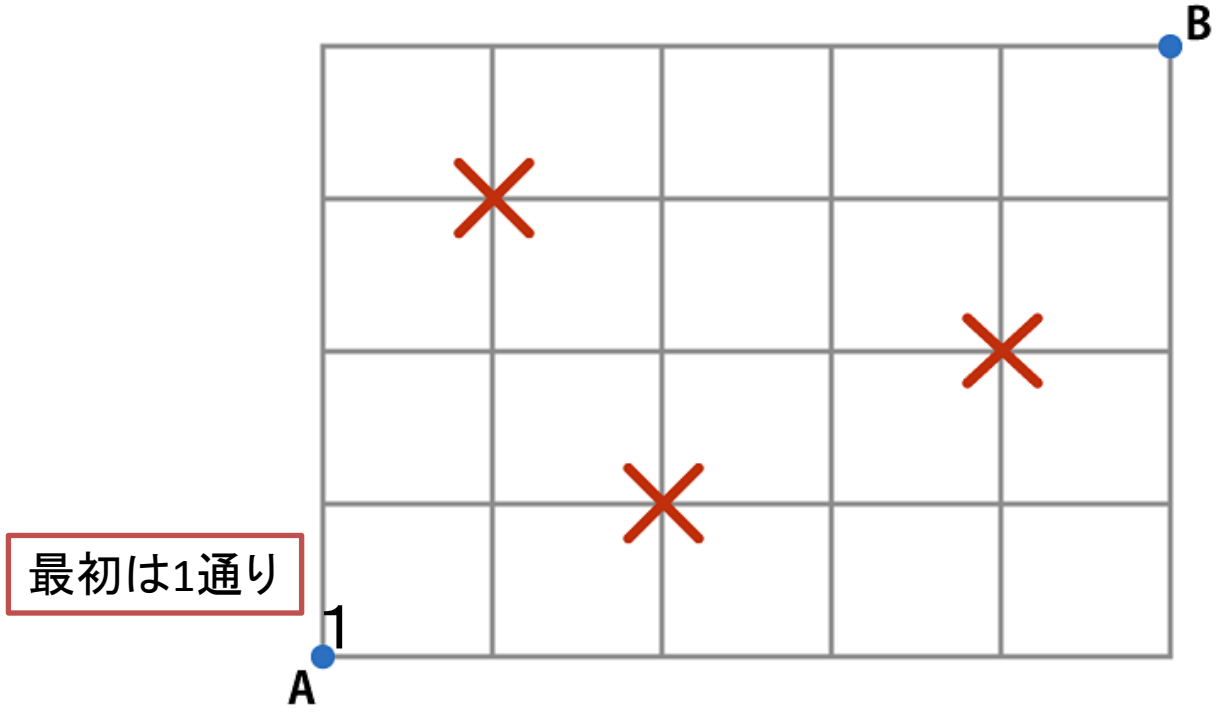
AtCoder

- 動的計画法の考え方
  - 「その場所にいくまで」の情報をメモする
    - ○○まで何通りあるか
    - ○○までの最大値はいくつか
    - ○○までの最短距離はいくつか
    - ○○まで到達可能か
  - その情報から、次の場所を埋めていく

動的計画法って？

AtCoder

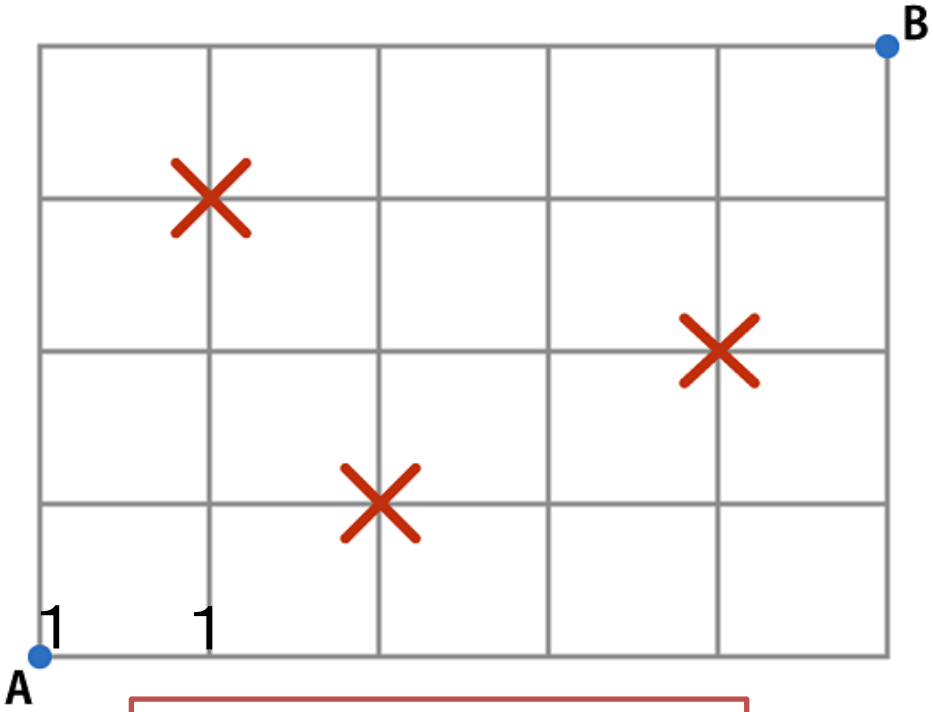
- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく



動的計画法って？

AtCoder

- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく

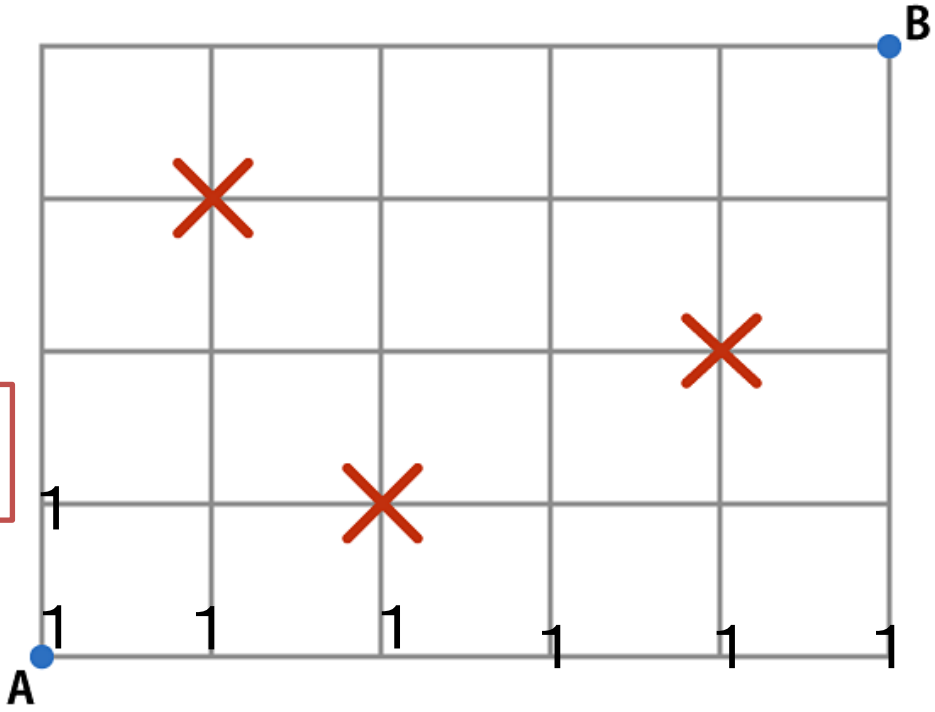


このマスにくるパターンも、
右に移動する1パターンだけ

動的計画法って？

AtCoder

- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく

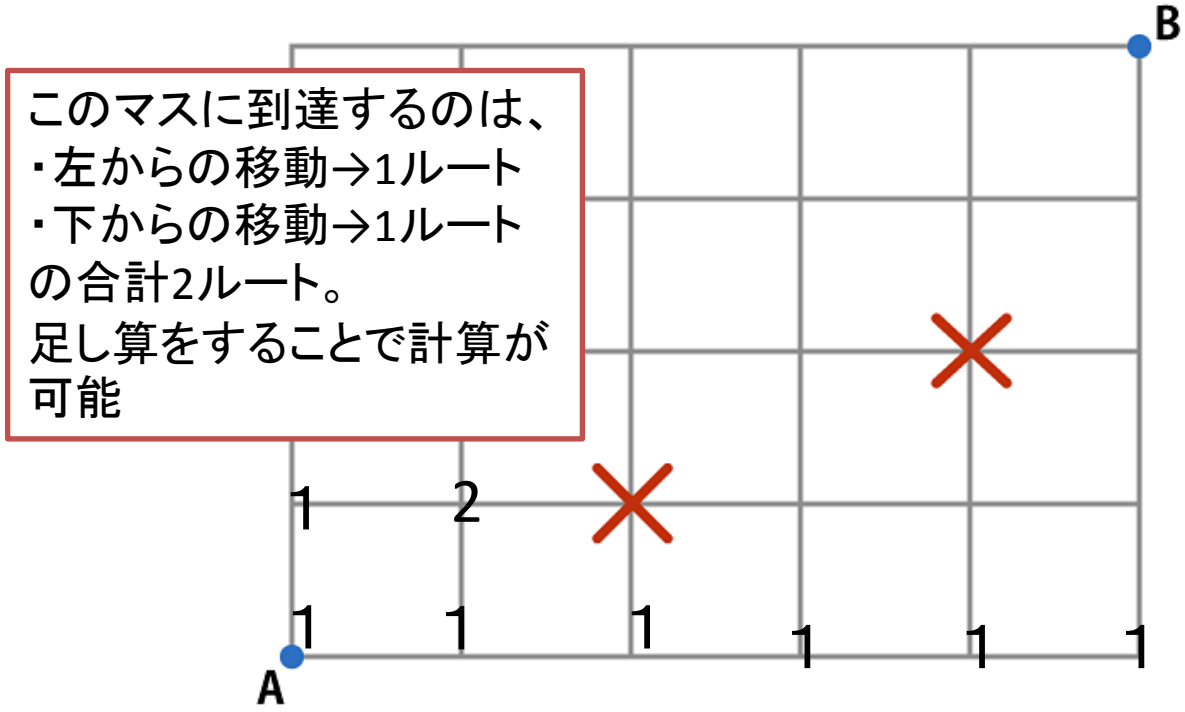


同様に、順番にここまで埋める

動的計画法って？

AtCoder

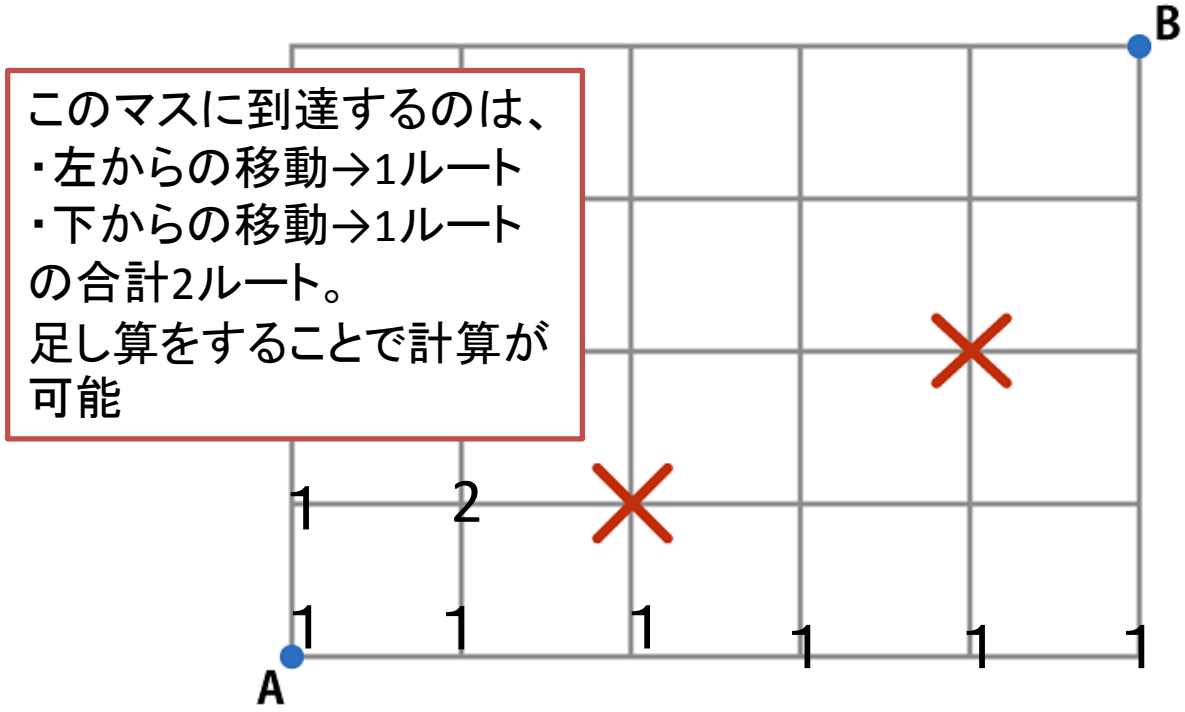
- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく



動的計画法って？

AtCoder

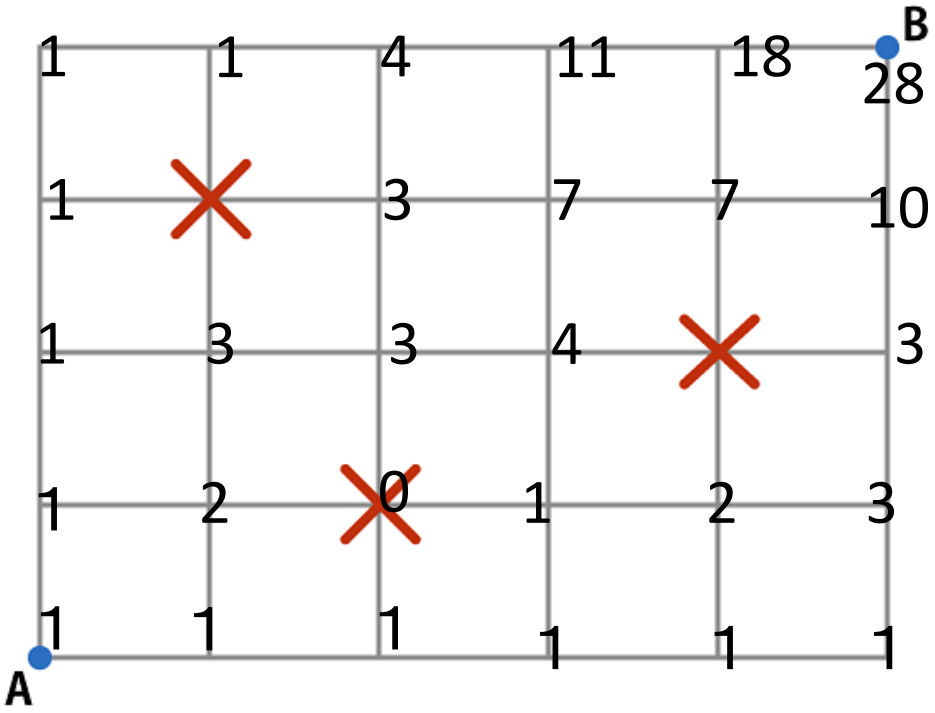
- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく



動的計画法って？

AtCoder

- AからBに行く方法は何通り？
  - ただし遠回りをするのは禁止とする
  - 左下から順番に埋めていく



最後まで埋めると
答えが解る

動的計画法って？

AtCoder

- ソースコード

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) continue;
        if(i!=0) dp[i][j] += dp[i-1][j];
        if(j!=0) dp[i][j] += dp[i][j-1];
    }
}
```

動的計画法って？

AtCoder

- ソースコード

初期値に1を入れる

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) continue;
        if(i!=0) dp[i][j] += dp[i-1][j];
        if(j!=0) dp[i][j] += dp[i][j-1];
    }
}
```

動的計画法って？

AtCoder

- ソースコード

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) continue;
        if(i!=0) dp[i][j] += dp[i-1][j];
        if(j!=0) dp[i][j] += dp[i][j-1];
    }
}
```

侵入不可能マスなら
continue

動的計画法って？

AtCoder

- ソースコード

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) continue;
        if(i!=0) dp[i][j] += dp[i-1][j];
        if(j!=0) dp[i][j] += dp[i][j-1];
    }
}
```

下から増やす処理、
左から増やす処理を
順番に行う

動的計画法って？

AtCoder

- ソースコード2

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) dp[i][j] = 0;
        if(i!=H-1) dp[i+1][j] += dp[i][j];
        if(j!=W-1) dp[i][j+1] += dp[i][j];
    }
}
```

動的計画法って？

AtCoder

- ソースコード2

```
int[][] dp = new int[H][W];
dp[0][0] = 1;
for(int i=0;i<H;i++){
    for(int j=0;j<W;j++){
        if(ng[i][j]) dp[i][j] = 0;
        if(i!=H-1) dp[i+1][j] += dp[i][j];
        if(j!=W-1) dp[i][j+1] += dp[i][j];
    }
}
```

上へ増やす処理、
右に増やす処理を
順番に行う

動的計画法って？

AtCoder

- ソースコード1とソースコード2の違い
  - 1は、下や左から、数値を「もらう」動的計画法
    - 「もらうDP」などと呼ばれている
  - 2は、上や右に、数値を「くばる」動的計画法
    - 「配るDP」などと呼ばれている
  - 大抵の場合はどっちで実装しても良い
    - 書き易い方を書きましょう

動的計画法って？

AtCoder

- 動的計画法のポイントはここ！
  - 今までの計算結果をメモしておく
  - 同じ計算結果は纏めてしまう
    - 組み合わせの個数であれば足し算。最大値や最小値であれば、最大値や最小値だけ使う
  - これらを守って実装すれば、メモ化再帰より実装は簡単

動的計画法とメモ化再帰の関係性

AtCoder

- 先に2つ上げた2つの問題は、メモ化再帰でも解くことが可能
  - フィボナッチ数列は先に挙げた通り
  - メモ化再帰は、残り縦移動回数と横移動回数を引数とした再帰関数 int dfs(int n, int m)を作ってあげれば良い。

- 動的計画法は、「ここまでの計算結果」をメモする。
- メモ化再帰は、「ここから先の計算結果」をメモする。
- やってることは逆なので、向きを変えると同じになる
  - スタートからゴールに動的計画法を行うのは、ゴールからスタートにメモ化再帰するのと同じ

動的計画法　実践

AtCoder

- 第2回早稲田大学プログラミングコンテスト B問題
  - http://wupc2nd.contest.atcoder.jp/tasks/wupc_02
  - 先ほど解いた水たまりの問題

動的計画法　実践

AtCoder

- 実装方針
  - k番目のマスに辿り着いた時の、水たまりを通る最小の数を計算すれば良い
  - これを、動的計画法で計算する

動的計画法　実践

AtCoder

- 具体的な実装(貰うDPの場合)

動的計画法　実践

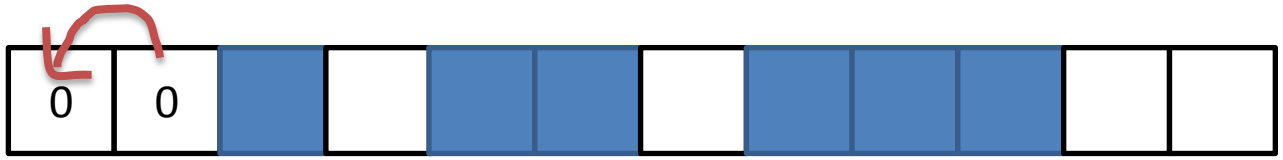
AtCoder

- 具体的な実装(貰うDPの場合)
  - まず、初期値には0が入る

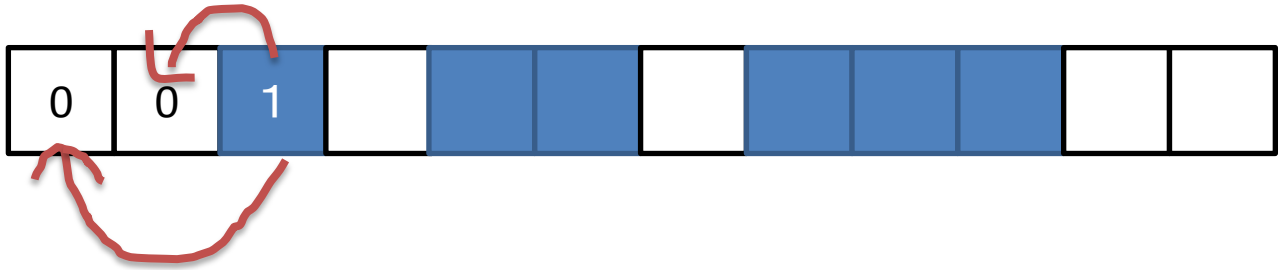
動的計画法　実践

AtCoder

- 具体的な実装(貰うDPの場合)
  - まず、初期値には0が入る
  - 次のマスは、そこに辿り着くための全てのマスの最小値を取る



- 具体的な実装(貰うDPの場合)
  - まず、初期値には0が入る
  - 次のマスは、そこに辿り着くための全てのマスの最小値を取る
  - 水たまりのマスは、その値にさらに1を足す



動的計画法　実践

AtCoder

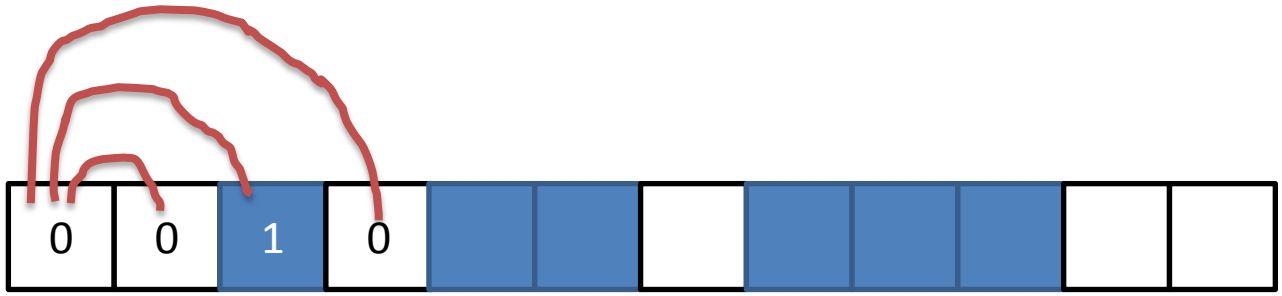
- 具体的な実装(貰うDPの場合)
  - まず、初期値には0が入る
  - 次のマスは、そこに辿り着くための全てのマスの最小値を取る
  - 水たまりのマスは、その値にさらに1を足す
  - 以下を繰り返して最後のマスまで計算する

動的計画法　実践

AtCoder

- 具体的な実装(配るDPの場合)
  - 初期値に0を入れる

- 具体的な実装(配るDPの場合)
  - 初期値に0を入れる
  - 次のマスに対し、解候補を入れてあげる
    - 水たまりがあったら1を足す
    - 最小値をメモするので、前入っている値の方が小さかったら変化させない
      - 事前には999などの大きい値を入れておく



動的計画法　実践

AtCoder

- ソースコード(貰うDP)
  - http://wupc2nd.contest.atcoder.jp/submissions/148412

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();
    String S = cin.next();

    int[] dp = new int[N];
    for(int i=1;i<N;i++){
        dp[i] = 999999; //適当に大きい数
        for(int j=1;j<=3;j++){
            if(i<j) break; //はみ出たらbreak
            //最小値を後ろから貰い、更新
            dp[i] = Math.min(dp[i], dp[i-j]);
        }
        //もし水たまりなら、答えを1足す
        if(S.charAt(i) == 'x') dp[i]++;
    }
    System.out.println(dp[N - 1]);
}
```

動的計画法　実践

AtCoder

- ソースコード(配るDP)
  - http://wupc2nd.contest.atcoder.jp/submissions/148410

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();
    String S = cin.next();

    int[] dp = new int[N];
    for(int i=1;i<N;i++){
        dp[i] = 999999; //適当に大きい数
        for(int j=1;j<=3;j++){
            if(i<j) break; //はみ出たらbreak
            //最小値を後ろから貰い、更新
            dp[i] = Math.min(dp[i], dp[i-j]);
        }
        //もし水たまりなら、答えを1足す
        if(S.charAt(i) == 'X') dp[i]++;
    }
    System.out.println(dp[N - 1]);
}
```

動的計画法　演習

AtCoder

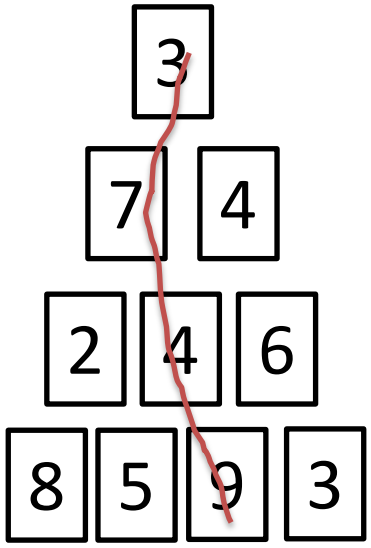
- 早稲田大学プログラミングコンテスト 2012 D問題
  - http://wupc2012.contest.atcoder.jp/tasks/wupc2012_4

- 問題概要
  - 整数Nが与えられる
  - 高さNの三角形型の数字のデータが与えられる
  - 上から順番に下に移動しながら数字を辿っていく
  - 数字の和を最大にしたい時、その値を求めなさい。

動的計画法　演習

AtCoder

- 問題概要
  - 下記の様なデータが与えられる
  - 上から順番に下りながら数字を拾っていく
  - 取れ得る数字の和の最大値を出力しなさい



動的計画法　演習

AtCoder

- 出来た人は以下の問題にも挑戦！
- Typical DP Contest A問題 コンテスト
  - http://tdpc.contest.atcoder.jp/tasks/tdpc_contest
- Typical DP Contest B問題 ゲーム
  - http://tdpc.contest.atcoder.jp/tasks/tdpc_game
- AtCoder Regular Contest 005 D問題 連射王高橋君
  - http://arc005.contest.atcoder.jp/tasks/arc005_4
    - 激ムズ注意！

動的計画法　演習　ヒント

AtCoder

- 動的計画法がイメージつかないのであれば、メモ化再帰でもOK！
- 「この場所に辿り着くまでにいくつの数字を獲得できたか」をメモするのが動的計画法
- 「この場所から残りいくつの数字を獲得できるか」をメモするのがメモ化再帰
  - 好きな方を組もう！

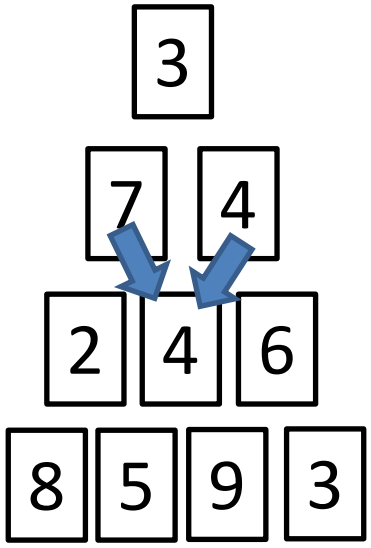
動的計画法　演習　解説

AtCoder

- 今見ている数字に対して、ここまでの数字の和の最大値が得られれば良い
  - それを念頭に入れて、動的計画法を組めば良い
    - 今回は貰うDPが一番書き易そう？

AtCoder

- 動的計画法（貰うDP）のイメージ
  - 中央の4の値を知るには、上の7,4の2つの経路の最大値を取ってきて、それに4を足せば良い。



AtCoder

- ソースコード(貰うDP)
  - http://wupc2012.contest.atcoder.jp/submissions/148415

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();

    //数値を入力する
    int[][] a = new int[N][N];
    for(int i=0;i<N;i++){
        for(int j=0;j<=i;j++){
            a[i][j] = cin.nextInt();
        }
    }
```

AtCoder

- ソースコード(貰うDP)

```
    int[][] dp = new int[N][N];
    int ret = 0;
    for(int i=0;i<N;i++){
        for(int j=0;j<=i;j++){
            //左上の場所から最大値を貰う
            if(i!=0 && j!=0){
                dp[i][j] = Math.max(dp[i][j], dp[i-1][j-1]);
            }
            //上の場所から最大値貰う
            if(i!=0) dp[i][j] = Math.max(dp[i][j], dp[i-1][j]);
            //今の数字を足す
            dp[i][j] += a[i][j];
            //答えの更新
            ret = Math.max(ret, dp[i][j]);
        }
    }
    System.out.println(ret);
}
```

動的計画法　演習

AtCoder

- もちろん、メモ化再帰や配るDPで書いてもOK!
  - 書き易いものを使いましょう！

本日のまとめ

本日のまとめ

AtCoder

- 計算量
  - 大体の計算時間が予測出来るようになった！
- メモ化再帰
  - 深さ優先探索を、一瞬で高速化出来る魔法のツール
  - 探索の計算量も見積もりやすくなった
- 動的計画法
  - メモ化再帰と対になる概念
  - こちらも計算時間を大幅に短縮できる！