精読『アルゴリズムイントロダクション 第4版』

第2回

Panot

最終更新：April 30, 2023

復習

- アルゴリズムは明確に定義された手続き。
- アルゴリズムによって効率が異なり、必要なリソースも大きく変わる。
- 実行時間を比較する際、増加が一番速い項で特徴づけられる。

今日の内容

- 分割統治、再帰、漸化式
- アルゴリズム的な漸化式の解く方法
- 確率的解析と乱択アルゴリズム

Section 1

分割統治、再帰、漸化式

基底ケースと再帰ケース

問題のサイズが十分に小さい場合は**基底ケース（base case）**で、直接的な方法で解く。
そうでない場合は**再帰ケース（recursive case）**で、このように問題を部分問題に分割し、統合して解を得る。

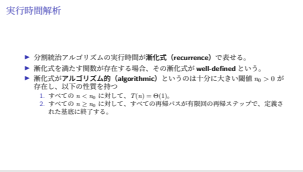
分割 問題をいくつかの同じ問題のより小さいインスタンスである部分問題に分割。

統治 部分問題を再帰的に解く。

統合 部分問題の解を組み合わせて元の問題の解を得る。

実行時間解析

- 分割統治アルゴリズムの実行時間が**漸化式（recurrence）**で表せる。
- 漸化式を満たす関数が存在する場合、その漸化式が **well-defined** という。
- 漸化式が**アルゴリズム的（algorithmic）**というのは十分に大きい閾値 $n_0 > 0$ が存在し、以下の性質を持つ
  1. すべての $n < n_0$ に対して、$T(n) = \Theta(1)$。
  2. すべての $n \geq n_0$ に対して、すべての再帰パスが有限回の再帰ステップで、定義された基底に終了する。



- アルゴリズム的とは要するに、ある定義された手続きによって条件下のあらゆる入力に対して有限時間に終了する

分割統治と漸化式

- この章は正方行列の積を計算問題として、いくつかのアルゴリズムを紹介して分析する。
- $n \times n$ の正方行列の積は直接計算した場合、実行時間 $\Theta(n^3)$ となる。
- 単純な分割統治法で 4 つの大きさ $n/2$ の部分問題にして計算したら、実行時間は漸化式 $T(n) = 8T(n/2) + \Theta(1)$ になるが、この実行時間は $\Theta(n^3)$ である。
- Strassen の分割統治法を使うと、実行時間は $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{\lg 7}) = \Theta(n^{2.81})$ である。

分割統治と漸化式

- この章は正方行列の積を計算問題として、いくつかのアルゴリズムを紹介して分析する。
- $n \times n$ の正方行列の積は直接計算した場合、実行時間 $\Theta(n^3)$ となる。
- 単純な分割統治法で 4 つの大きさ $n/2$ の部分問題にして計算したら、実行時間は漸化式 $T(n) = 8T(n/2) + \Theta(1)$ になるが、この実行時間は $\Theta(n^3)$ である。
- Strassen の分割統治法を使うと、実行時間は $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{\log 7}) = \Theta(n^{2.81})$ である。

- 積を直接計算して $\Theta(n^3)$ になる理由は、積の要素数は n^2 それぞれ計算するのに n 回の掛け算して和をとるので、n^3 回の掛け算と足し算が行われます。
- 単純の分割統治法を使っても $\Theta(n^3)$ なので、1969 年の Strassen の発見まで上界が小さくなることに誰も期待していなかった。
- ちなみに、理論上 Strassen のアルゴリズムより上界が小さいアルゴリズムがいくつか発見されていますが、実用的ではないそうです。

漸化式を解く方法

- **置き換え法（substitution method）**では、推測し数学的帰納法で証明する。
- **再帰木法（recursion-tree method）**では、漸化式を木と見立てて、各階層の総実行時間を計算し、全ての階層の総和を計算する。
- **マスター法（master method）**では、$a > 0$ かつ $b > 1$ のときの漸化式 $T(n) = aT(n/b) + f(n)$ の閉じた限界式を求める。
- **Akra-Bazzi 法**では、分割統治の漸化式の閉じた限界式を一般的に求める。

漸化式を解く方法

- **置き換え法（substitution method）**では、推測し数学的帰納法で証明する。
- **再帰木法（recursion-tree method）**では、漸化式を木と見立てて、各階層の総実行時間を計算し、全ての階層の総和を計算する。
- **マスター法（master method）**では、$a > 0$ かつ $b > 1$ のときの漸化式 $T(n) = aT(n/b) + f(n)$ の閉じた限界式を求める。
- **Akra-Bazzi 法**では、分割統治の漸化式の閉じた限界式を一般的に求める。

- 置き換え法（substitution method）は、解くよりも推測した限界を数学的帰納法で証明する方法です。事前に推測を行う必要があります。
- 再帰木法（recursion-tree method）は、個人的に一番直感的かなと思います。木を書いて解析することです。
- マスター法（master method）は、特定の形の漸化式を解く方法です。最初は怖そうな定理だと思ってましたが、意外と直感的で再帰木法からの閉じた解が得られるような感覚です。
- ２次方程式を平方完成で解くか、解の公式を使うかみたいな感覚かな？

正方行列の積

$n \times n$ の正方行列 $A = (a_{ik})$ と $B = (b_kj)$ に対して、積 $C = A \cdot B$ も $n \times n$ の正方行列で C の要素 c_{ij} は以下のように成り立つ。

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

正方行列の積

行列

$$AB = \begin{matrix} \\ ① \\ ② \end{matrix} \begin{matrix} \begin{matrix} \\ ① \end{matrix} \\ \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \end{matrix} \begin{matrix} \begin{matrix} ① & ② \end{matrix} \\ \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{matrix}$$

$$c_{11} = \underset{①\ ①}{a_{11}b_{11}} + \underset{①\ ①}{a_{12}a_{21}} \qquad (1,1)\ \text{成分}$$

$$c_{12} = \underset{①\ ②}{a_{11}b_{12}} + \underset{①\ ②}{a_{12}a_{22}} \qquad (1,2)\ \text{成分}$$

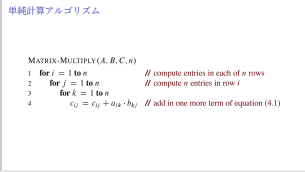
$$c_{21} = \underset{②\ ①}{a_{21}b_{11}} + \underset{②\ ①}{a_{22}a_{21}} \qquad (2,1)\ \text{成分}$$

$$c_{22} = \underset{②\ ②}{a_{21}b_{12}} + \underset{②\ ②}{a_{22}a_{22}} \qquad (2,2)\ \text{成分}$$

単純計算アルゴリズム

MATRIX-MULTIPLY(A, B, C, n)

```
1  for i = 1 to n                    // compute entries in each of n rows
2      for j = 1 to n                // compute n entries in row i
3          for k = 1 to n
4              c_ij = c_ij + a_ik · b_kj   // add in one more term of equation (4.1)
```



- まず c_{ij} を 0 に初期化します。
- 最初の 2 つの for ループは積の各要素を繰り返します。
- 一番深い for ループは積の各要素を計算です。

単純な分割統治法

行列 A と B を４分割して、以下の計算をする

$$\begin{aligned}\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \\ &= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}\end{aligned}$$

単純な分割統治法のアルゴリズム

```
MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)
 1  if n == 1
 2      // Base case.
 3      c11 = c11 + a11 · b11
 4      return
 5  // Divide.
 6  partition A, B, and C into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22;
        and C11, C12, C21, C22; respectively
 7  // Conquer.
 8  MATRIX-MULTIPLY-RECURSIVE(A11, B11, C11, n/2)
 9  MATRIX-MULTIPLY-RECURSIVE(A11, B12, C12, n/2)
10  MATRIX-MULTIPLY-RECURSIVE(A21, B11, C21, n/2)
11  MATRIX-MULTIPLY-RECURSIVE(A21, B12, C22, n/2)
12  MATRIX-MULTIPLY-RECURSIVE(A12, B21, C11, n/2)
13  MATRIX-MULTIPLY-RECURSIVE(A12, B22, C12, n/2)
14  MATRIX-MULTIPLY-RECURSIVE(A22, B21, C21, n/2)
15  MATRIX-MULTIPLY-RECURSIVE(A22, B22, C22, n/2)
```

実行時間は漸化式 $T(n) = 8T(n/2) + \Theta(1)$ になるが、この実行時間は $\Theta(n^3)$ である。

単純な分割統治法のアルゴリズム

```
MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)
1  If n = 1
2    // base case
3    c_11 = a_11 · b_11
4    return
5  // Divide
6  partition A, B, and C into n/2 × n/2 submatrices
       A_11, A_12, A_21, A_22, B_11, B_12, B_21,
       and B_22, C_11, C_12, C_21, C_22, respectively
7  // Conquer
8  MATRIX-MULTIPLY-RECURSIVE(A_11, B_11, C_11, n/2)
9  MATRIX-MULTIPLY-RECURSIVE(A_11, B_12, C_12, n/2)
10 MATRIX-MULTIPLY-RECURSIVE(A_21, B_11, C_21, n/2)
11 MATRIX-MULTIPLY-RECURSIVE(A_21, B_12, C_22, n/2)
12 MATRIX-MULTIPLY-RECURSIVE(A_12, B_21, C_11, n/2)
13 MATRIX-MULTIPLY-RECURSIVE(A_12, B_22, C_12, n/2)
14 MATRIX-MULTIPLY-RECURSIVE(A_22, B_21, C_21, n/2)
15 MATRIX-MULTIPLY-RECURSIVE(A_22, B_22, C_22, n/2)
```

実行時間は漸化式 $T(n) = 8T(n/2) + \Theta(1)$ になるが、この実行時間は $\Theta(n^3)$ である。

- 単純に分割しても計算量は変わりませんね。

Strassen のアルゴリズム

- 詳細は各自確認してください。
- 要は実行時間は $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{2.81})$。

- 詳細は細かいのでここで紹介しません。
- 実行時間の漸化式をどうやって解析するのが今回のキモです。

Strassen のアルゴリズム
- 詳細は各自確認してください。
- 要は実行時間は $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{2.81})$。

Section 2

アルゴリズム的な漸化式の解く方法

漸化式を解く方法

- **置き換え法（substitution method）**では、推測し数学的帰納法で証明する。
- **再帰木法（recursion-tree method）**では、漸化式を木と見立てて、各階層の総実行時間を計算し、全ての階層の総和を計算する。
- **マスター法（master method）**では、$a > 0$ かつ $b > 1$ のときの漸化式 $T(n) = aT(n/b) + f(n)$ の閉じた限界式を求める。
- **Akra-Bazzi 法**では、分割統治の漸化式の閉じた限界式を一般的に求める。

置き換え法（substitution method）

- 定数を変数のままにして、限界を推測する。
- 数学的帰納法を用いて限界の正しさを証明し、定数を特定する。

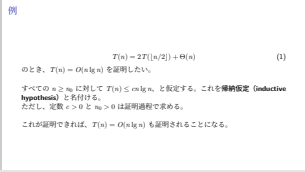
例

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n) \tag{1}$$

のとき、$T(n) = O(n \lg n)$ を証明したい。

すべての $n \geq n_0$ に対して $T(n) \leq cn \lg n$、と仮定する。これを**帰納仮定（inductive hypothesis）**と名付ける。
ただし、定数 $c > 0$ と $n_0 > 0$ は証明過程で求める。

これが証明できれば、$T(n) = O(n \lg n)$ も証明されることになる。


例

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n) \tag{1}$$

のとき、$T(n) = O(n \lg n)$ を証明したい。

すべての $n \geq n_0$ に対して $T(n) \leq cn \lg n$、と仮定する。これを**帰納仮定（inductive hypothesis）**と名付ける。
ただし、定数 $c > 0$ と $n_0 > 0$ は証明過程で求める。

これが証明できれば、$T(n) = O(n \lg n)$ も証明されることになる。

- 帰納仮定を証明したい。

例：数学的帰納法の帰納ケース（inductive case）

- $[n_0, n)$ で $T(n) \leq cn \lg n$ が成り立つなら、n でも成り立つことを証明する。

例：数学的帰納法の帰納ケース（inductive case）

- $[n_0, n)$ で $T(n) \leq cn \lg n$ が成り立つなら、n でも成り立つことを証明する。
- $n \geq 2n_0$ なら、$\lfloor n/2 \rfloor$ で成り立つ。従って、$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$。

例：数学的帰納法の帰納ケース（inductive case）

- $[n_0, n)$ で $T(n) \leq cn \lg n$ が成り立つなら、n でも成り立つことを証明する。
- $n \geq 2n_0$ なら、$\lfloor n/2 \rfloor$ で成り立つ。従って、$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$。
- （1）に代入すると、

$$
\begin{aligned}
T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\
&\leq 2(c(n/2)\lg(n/2)) + \Theta(n) \\
&= cn \lg(n/2) + \Theta(n) \\
&= cn \lg n - cn \lg 2 + \Theta(n) \\
&= cn \lg n - cn + \Theta(n) \\
&\leq cn \lg n
\end{aligned}
$$

例：数学的帰納法の帰納ケース（inductive case）

- $[n_0, n)$ で $T(n) \leq cn \lg n$ が成り立つなら、n でも成り立つことを証明する。
- $n \geq 2n_0$ なら、$\lfloor n/2 \rfloor$ で成り立つ。従って、$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$。
- （1）に代入すると、

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\ &\leq 2(c(n/2)\lg(n/2)) + \Theta(n) \\ &= cn \lg(n/2) + \Theta(n) \\ &= cn \lg n - cn \lg 2 + \Theta(n) \\ &= cn \lg n - cn + \Theta(n) \\ &\leq cn \lg n \end{aligned}$$

- これにより、$n \geq 2n_0$ のとき成り立つことを証明した。

例：数学的帰納法の帰納ケース（inductive case）

- $[n_0, n)$ で $T(n) \leq cn \lg n$ が成り立つなら、n でも成り立つことを証明する。
- $n \geq 2n_0$ なら、$\lfloor n/2 \rfloor$ で成り立つ。従って、$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$。
- (1) に代入すると、

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\ &\leq 2(cn/2)\lg(n/2)) + \Theta(n) \\ &= cn\lg(n/2) + \Theta(n) \\ &= cn\lg n - cn\lg 2 + \Theta(n) \\ &= cn\lg n - cn + \Theta(n) \\ &\leq cn\lg n \end{aligned}$$

- これにより、$n \geq 2n_0$ のとき成り立つことを証明した。

- 帰納仮定の帰納ケースを証明する。
- 整数上の数学的帰納法の帰納ケースは「$n-1$ までが成り立つなら n も成り立つ」を証明する。
- 帰納仮定に代入。
- 結果を漸化式に代入。
- 最後の行では c の選び方によって、cn が $\Theta(n)$ に隠されている無名の関数より大きくなるために不等式が成り立つ。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。
- $T(n)$ はアルゴリズム的という前提の元で $T(2)$ と $T(3)$ が定数。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。
- $T(n)$ はアルゴリズム的という前提の元で $T(2)$ と $T(3)$ が定数。
- $c = \max\{T(2), T(3)\}$ と選ぶ。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。
- $T(n)$ はアルゴリズム的という前提の元で $T(2)$ と $T(3)$ が定数。
- $c = \max\{T(2), T(3)\}$ と選ぶ。
- よって、

$$T(2) \leq c < (2 \lg 2)c$$
$$T(3) \leq c < (3 \lg 3)c$$

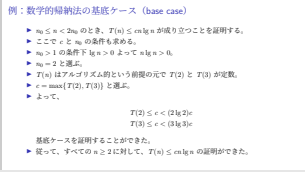
基底ケースを証明することができた。

例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。
- $T(n)$ はアルゴリズム的という前提の元で $T(2)$ と $T(3)$ が定数。
- $c = \max\{T(2), T(3)\}$ と選ぶ。
- よって、

$$T(2) \leq c < (2 \lg 2)c$$
$$T(3) \leq c < (3 \lg 3)c$$

  基底ケースを証明することができた。
- 従って、すべての $n \geq 2$ に対して、$T(n) \leq cn \lg n$ の証明ができた。


例：数学的帰納法の基底ケース（base case）

- $n_0 \leq n < 2n_0$ のとき、$T(n) \leq cn \lg n$ が成り立つことを証明する。
- ここで、c と n_0 の条件も求める。
- $n_0 > 1$ の条件下 $\lg n > 0$ よって $n \lg n > 0$。
- $n_0 = 2$ と選ぶ。
- $T(n)$ はアルゴリズム的という前提の元で $T(2)$ と $T(3)$ が定数。
- $c = \max\{T(2), T(3)\}$ と選ぶ。
- よって、

$$T(2) \leq c < (2 \lg 2)c$$
$$T(3) \leq c < (3 \lg 3)c$$

基底ケースを証明することができた。

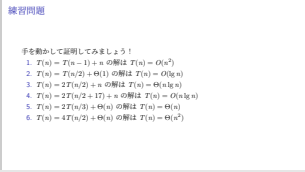
- 従って、すべての $n \geq 2$ に対して、$T(n) \leq cn \lg n$ の証明ができた。

- 帰納仮定の基底ケースを証明する。
- 帰納ケースでは、$n \geq 2n_0$ の場合を証明したが、今度 $n_0 \leq n < 2n_0$ の場合を証明。
- アルゴリズム的というのは、必ず漸化式の基底ケースに帰着して、有限時間内に終了。

練習問題

手を動かして証明してみましょう！

1. $T(n) = T(n-1) + n$ の解は $T(n) = O(n^2)$
2. $T(n) = T(n/2) + \Theta(1)$ の解は $T(n) = O(\lg n)$
3. $T(n) = 2T(n/2) + n$ の解は $T(n) = \Theta(n \lg n)$
4. $T(n) = 2T(n/2 + 17) + n$ の解は $T(n) = O(n \lg n)$
5. $T(n) = 2T(n/3) + \Theta(n)$ の解は $T(n) = \Theta(n)$
6. $T(n) = 4T(n/2) + \Theta(n)$ の解は $T(n) = \Theta(n^2)$


練習問題

手を動かして証明してみましょう！

1. $T(n) = T(n-1) + n$ の解は $T(n) = O(n^2)$
2. $T(n) = T(n/2) + \Theta(1)$ の解は $T(n) = O(\lg n)$
3. $T(n) = 2T(n/2) + n$ の解は $T(n) = \Theta(n \lg n)$
4. $T(n) = 2T(n/2 + 17) + n$ の解は $T(n) = O(n \lg n)$
5. $T(n) = 2T(n/3) + \Theta(n)$ の解は $T(n) = \Theta(n)$
6. $T(n) = 4T(n/2) + \Theta(n)$ の解は $T(n) = \Theta(n^2)$

- ここでやりません。
- このスライドを後で見て、練習したい方はどうぞ。

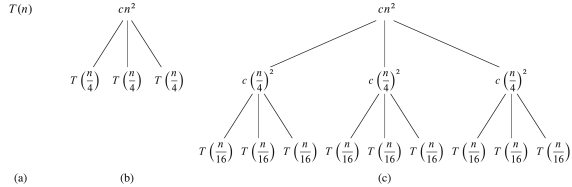
再帰木法（recursion-tree method）

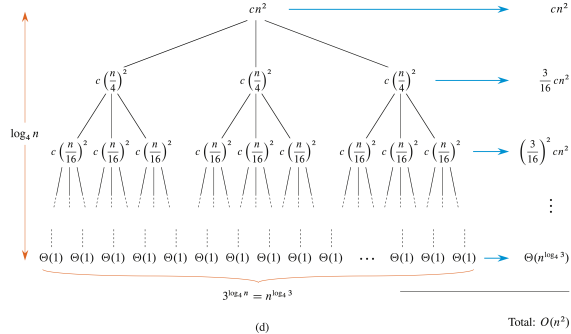
- 漸化式を木と見立てて、各階層の総実行時間を計算し、全ての階層の総和を計算する。
- この方法で直接照明してもいいし、置き換え法の推測のために使ってもいい。

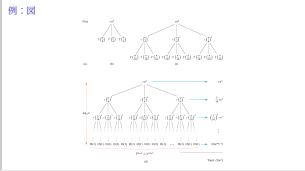
例

$T(n) = 3T(n/4) + \Theta(n^2)$ の上界を求めたい。

例：図


$T(n)$
cn^2
$T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$
$c\left(\frac{n}{4}\right)^2$
$T\left(\frac{n}{16}\right)$
(a)
(b)
(c)

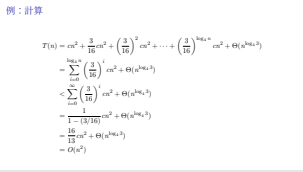

cn^2
$\frac{3}{16}cn^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$\log_4 n$
$\Theta(1)$
$\Theta(n^{\log_4 3})$
$3^{\log_4 n} = n^{\log_4 3}$
(d)
Total: $O(n^2)$



- まず $T(n)$ を根にします。
- 次は T で表されている葉っぱを再帰的に作ります。
- 節点を漸化式じゃないほうの項にして、この場合は $\Theta(n^2)$ なので、係数を c にして、書きます。その子節点は部分問題にしていきます。
- ２－３回繰り返していって、基底ケースを書く。
- 分割統治法の基底ケースは「問題の大きさは特定な大きさより小さいとき直接な方法で解く」とあるため、実行時間は定数で $\Theta(1)$ になります。
- 次は各層の実行時間の和を計算して、層の数を計算して、総和を取ります。

例：計算

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$


例：計算

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

- この場合の総和はこう計算できますね。

マスター法（master method）

- $a > 0$ かつ $b > 1$ のときの漸化式 $T(n) = aT(n/b) + f(n)$ の閉じた限界式を求める。
- この漸化式は問題の大きさ n を a 個大きさ n/b の部分問題に分割統治法を用いたアルゴリズムの実行時間を示す。$f(n)$ は分割と統合にかかるコストを表す。

Theorem (マスター定理（master theorem）)

$a > 0$ と $b > 1$ を定数として、$f(n)$ を十分に大きい実数に定義されかつ非負の関数とする。$n \in \mathbb{N}$ 上の漸化式 $T(n)$ をこのように定義する：

$$T(n) = aT(n/b) + f(n)$$

ただし $aT(n/b)$ は厳密に $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ であり、$a' \geq 0$ かつ $a'' \geq 0$ かつ $a = a' + a''$ である。このとき、$T(n)$ の漸近的特徴は以下のように得られる。

1. $f(n) = O(n^{\log_b a - \epsilon})$ となるような定数 $\epsilon > 0$ が存在したら、$T(n) = \Theta(n^{\log_b a})$ である。
2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ となるような定数 $k \geq 0$ が存在したら、$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ である。
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ となるような定数 $\epsilon > 0$ が存在し、しかもある定数 $c < 1$ と十分に大きい n に対して、$af(n/b) \leq cf(n)$ が成り立ったら、$T(n) = \Theta(f(n))$ である。

Theorem (マスター定理（master theorem）)
$a > 0$ と $b > 1$ を定数として、$f(n)$ を十分に大きい実数に定義されかつ非負の関数とする。$n \in \mathbb{N}$ 上の漸化式 $T(n)$ をこのように定義する：

$$T(n) = aT(n/b) + f(n)$$

ただし $aT(n/b)$ は厳密に $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ であり、$a' \geq 0$ かつ $a'' \geq 0$ かつ $a = a' + a''$ である。このとき、$T(n)$ の漸近的特徴は以下のように得られる。

1. $f(n) = O(n^{\log_b a - \varepsilon})$ となるような定数 $\varepsilon > 0$ が存在したら、$T(n) = \Theta(n^{\log_b a})$ である。
2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ となるような定数 $k \geq 0$ が存在したら、$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ である。
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ となるような定数 $\varepsilon > 0$ が存在し、しかもある定数 $c < 1$ と十分に大きい n に対して、$af(n/b) \leq cf(n)$ が成り立ったら、$T(n) = \Theta(f(n))$ である。

- 第一印象：こわい。
- 上の方は、このような漸化式で使えますよ、という定理の適応条件ですね。大体アルゴリズム解析で見る漸化式はこの形です。
- 次はこの 3 つの $f(n)$ の場合分けです。
- 1 は $f(n)$ の上界がある条件に満たす場合。
- 2 は $f(n)$ のタイトな限界がある条件に満たす場合。
- 3 は $f(n)$ の下界がある条件に満たす場合。
- 全ての条件は $n^{\log_b a}$ に関連してる。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。
- $n^{\log_b a}$ を**分水嶺関数（watershed function）**という。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。
- $n^{\log_b a}$ を**分水嶺関数（watershed function）**という。
- 1 の場合は $aT(n/b)$ の増加率が大きい。

マスター定理を直感的に見てみよう

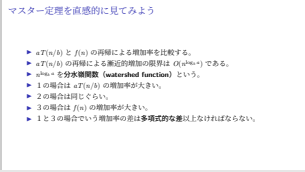
- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。
- $n^{\log_b a}$ を**分水嶺関数（watershed function）**という。
- 1 の場合は $aT(n/b)$ の増加率が大きい。
- 2 の場合は同じぐらい。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。
- $n^{\log_b a}$ を**分水嶺関数（watershed function）**という。
- 1 の場合は $aT(n/b)$ の増加率が大きい。
- 2 の場合は同じぐらい。
- 3 の場合は $f(n)$ の増加率が大きい。

マスター定理を直感的に見てみよう

- $aT(n/b)$ と $f(n)$ の再帰による増加率を比較する。
- $aT(n/b)$ の再帰による漸近的増加の限界は $O(n^{\log_b a})$ である。
- $n^{\log_b a}$ を**分水嶺関数（watershed function）**という。
- 1 の場合は $aT(n/b)$ の増加率が大きい。
- 2 の場合は同じぐらい。
- 3 の場合は $f(n)$ の増加率が大きい。
- 1 と 3 の場合でいう増加率の差は**多項式的な差**以上なければならない。



- $aT(n/b)$ という項が再帰的に足し算していけば、どれぐらいの増加率があるのか？
- $f(n)$ という項が再帰的に足し算していけば、どれぐらいの増加率があるのか？
- この２つの増加率を比較して、場合分けしている。
- 前のスライドで全ての条件は $n^{\log_b a}$ に関連していると言いましたが、これは $aT(n/b)$ という項の再帰による漸近的増加率。
- これを名付けて、軸にして $f(n)$ という項の再帰による漸近的増加率と比較します。
- １の場合は $aT(n/b)$ の増加率が大きいため、漸化式全体の漸近的限界は分水嶺関数のオーダになる。
- ２の場合は同じぐらいだから、漸化式全体の漸近的限界はそのコンビネーションです。
- ３の場合は $f(n)$ の増加率が大きいため、漸化式全体の漸近的限界は $f(n)$ のオーダになる。

マスター定理を使ってみる：マージソート

- 漸化式は $T(n) = 2T(n/2) + \Theta(n)$ である。

マスター定理を使ってみる：マージソート

- 漸化式は $T(n) = 2T(n/2) + \Theta(n)$ である。
- $a = 2$ と $b = 2$、よって分水嶺関数は $n^{\log_2 2} = n$ である。

マスター定理を使ってみる：マージソート

- 漸化式は $T(n) = 2T(n/2) + \Theta(n)$ である。
- $a = 2$ と $b = 2$、よって分水嶺関数は $n^{\log_2 2} = n$ である。
- $k = 0$ とおき $f(n) = \Theta(n) = \Theta(n^{\log_b a} \lg^k n)$、よって 2 の場合になる。

マスター定理を使ってみる：マージソート

- 漸化式は $T(n) = 2T(n/2) + \Theta(n)$ である。
- $a = 2$ と $b = 2$、よって分水嶺関数は $n^{\log_2 2} = n$ である。
- $k = 0$ とおき $f(n) = \Theta(n) = \Theta(n^{\log_b a} \lg^k n)$、よって 2 の場合になる。
- 従って、$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n) = \Theta(n \lg n)$ である。

マスター定理を使ってみる：マージソート

- 漸化式は $T(n) = 2T(n/2) + \Theta(n)$ である。
- $a = 2$ と $b = 2$、よって分水嶺関数は $n^{\log_2 2} = n$ である。
- $k = 0$ とおき $f(n) = \Theta(n) = \Theta(n^{\log_2 n} \lg^k n)$、よって 2 の場合になる。
- 従って、$T(n) = \Theta(n^{\log_2 n} \lg^{k+1} n) = \Theta(n \lg n)$ である。

- 分水嶺関数と $f(n) = \Theta(n)$ は同じオーダだから 1 と 3 の場合ではなさそう。
- じゃ、2 の場合の k を考えましょう。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。
- $a = 8$ と $b = 2$、よって分水嶺関数は $n^{\log_2 8} = n^3$ である。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。
- $a = 8$ と $b = 2$、よって分水嶺関数は $n^{\log_2 8} = n^3$ である。
- 任意の正の $\epsilon < 3$ をとれば $f(n) = O(n^{3-\epsilon})$ がなりたつ。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。
- $a = 8$ と $b = 2$、よって分水嶺関数は $n^{\log_2 8} = n^3$ である。
- 任意の正の $\epsilon < 3$ をとれば $f(n) = O(n^{3-\epsilon})$ がなりたつ。
- 分水嶺関数 n^3 は $f(n) = \Theta(1)$ より、多項式的な増加率の差を持つ。よって 1 の場合になる。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。
- $a = 8$ と $b = 2$、よって分水嶺関数は $n^{\log_2 8} = n^3$ である。
- 任意の正の $\epsilon < 3$ をとれば $f(n) = O(n^{3-\epsilon})$ がなりたつ。
- 分水嶺関数 n^3 は $f(n) = \Theta(1)$ より、多項式的な増加率の差を持つ。よって 1 の場合になる。
- 従って、$T(n) = \Theta(n^3)$ である。

マスター定理を使ってみる：行列積の単純分割統治

- 漸化式は $T(n) = 8T(n/2) + \Theta(1)$ である。
- $a = 8$ と $b = 2$、よって分水嶺関数は $n^{\log_b a} = n^3$ である。
- 任意の正の $\epsilon < 3$ をとれば $f(n) = O(n^{3-\epsilon})$ がなりたつ。
- 分水嶺関数 n^3 は $f(n) = \Theta(1)$ より、多項式的な増加率の差を持つ。よって 1 の場合になる。
- 従って、$T(n) = \Theta(n^3)$ である。

- 分水嶺関数 n^3 と $f(n) = \Theta(1)$ とを比較して、次数が 3 の差があるので、1 の場合になります。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。
- $a = 7$ と $b = 2$、よって分水嶺関数は $n^{\log_2 7} = n^{\lg 7}$。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。
- $a = 7$ と $b = 2$、よって分水嶺関数は $n^{\log_2 7} = n^{\lg 7}$。
- $\lg 7 > 2$ なので、任意の正の $\epsilon < \lg 7 - 2$ をとれば $f(n) = O(n^{\lg 7 - \epsilon})$ がなりたつ。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。
- $a = 7$ と $b = 2$、よって分水嶺関数は $n^{\log_2 7} = n^{\lg 7}$。
- $\lg 7 > 2$ なので、任意の正の $\epsilon < \lg 7 - 2$ をとれば $f(n) = O(n^{\lg 7 - \epsilon})$ がなりたつ。
- 分水嶺関数 $n^{\lg 7}$ は $f(n) = \Theta(n^2)$ より、多項式的な増加率の差を持つ。よって 1 の場合になる。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。
- $a = 7$ と $b = 2$、よって分水嶺関数は $n^{\log_2 7} = n^{\lg 7}$。
- $\lg 7 > 2$ なので、任意の正の $\epsilon < \lg 7 - 2$ をとれば $f(n) = O(n^{\lg 7 - \epsilon})$ がなりたつ。
- 分水嶺関数 $n^{\lg 7}$ は $f(n) = \Theta(n^2)$ より、多項式的な増加率の差を持つ。よって 1 の場合になる。
- 従って、$T(n) = \Theta(n^{\lg 7})$。

マスター定理を使ってみる：行列積の Strassen のアルゴリズム

- 漸化式は $T(n) = 7T(n/2) + \Theta(n^2)$ である。
- $a = 7$ と $b = 2$、よって分水嶺関数は $n^{\log_2 7} = n^{\lg 7}$。
- $\lg 7 > 2$ なので、任意の正の $\epsilon < \lg 7 - 2$ をとれば $f(n) = O(n^{\lg 7 - \epsilon})$ がなりたつ。
- 分水嶺関数 $n^{\lg 7}$ は $f(n) = \Theta(n^2)$ より、多項式的な増加率の差を持つ、よって 1 の場合になる。
- 従って、$T(n) = \Theta(n^{\lg 7})$。

- 分水嶺関数 $n^{2.81}$ と $f(n) = \Theta(n^2)$ とを比較して、次数が 0.8 以上の差があるので、1 の場合になります。

飛ばす内容

- マスター定理の証明
- Akra-Bazzi 法

Section 3

確率的解析と乱択アルゴリズム

採用問題（hiring problem）

- 採用エイジェントに依頼して、新しい助手を採用したい。
- 毎日候補を紹介してもらい、面接して、採用するかどうかを判断する。
- 面接するときに、エイジェントに少量の料金を払う。
- 採用するときに、解雇にかかるコストと大量の料金をエイジェントに払う。
- もっとも適した助手を雇いたい。

単純アルゴリズム

- 面接した候補が現在の助手より適任なら採用する。
- どんな時点でも採用される候補は今までもっとも適任であることが保証される。

```
HIRE-ASSISTANT(n)
1  best = 0            // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3      interview candidate i
4      if candidate i is better than candidate best
5          best = i
6          hire candidate i
```

費用解析

- 実行時間ではなく、費用を解析する。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。
- 総コストは $O(c_i n + c_h m)$。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。
- 総コストは $O(c_i n + c_h m)$。
- 何人採用しても面接に関する費用は固定の $c_i n$ であるので、採用のコスト $c_h m$ に着目する。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。
- 総コストは $O(c_i n + c_h m)$。
- 何人採用しても面接に関する費用は固定の $c_i n$ であるので、採用のコスト $c_h m$ に着目する。
- 最悪時の解析：候補が適任でない順に紹介されるとき、一人ずつ全員採用することになる。コストは $O(c_h n)$ になる。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。
- 総コストは $O(c_i n + c_h m)$。
- 何人採用しても面接に関する費用は固定の $c_i n$ であるので、採用のコスト $c_h m$ に着目する。
- 最悪時の解析：候補が適任でない順に紹介されるとき、一人ずつ全員採用することになる。コストは $O(c_h n)$ になる。
- ただし、どういう順番に紹介されるかわからないため、平均時を解析したい。

費用解析

- 実行時間ではなく、費用を解析する。
- 面接のコスト（安い）を c_i として、採用のコスト（高い）を c_h とする。
- 総コストは $O(c_i n + c_h m)$。
- 何人採用しても面接に関する費用は固定の $c_i n$ であるので、採用のコスト $c_h m$ に着目する。
- 最悪時の解析：候補が適任でない順に紹介されるとき、一人ずつ全員採用することになる。コストは $O(c_h n)$ になる。
- ただし、どういう順番に紹介されるかわからないため、平均時を解析したい。
- 平均時解析するとき確率的解析を行う。

指標確率変数（indicator random variables）

- 確率と期待値との間を行き来しやすいような方法。

指標確率変数（indicator random variables）

- 確率と期待値との間を行き来しやすいような方法。
- 標本空間 S と事象 A が与えられたとき、事象 A に関する指標確率変数（indicator random variables）$I\{A\}$ をこのように定義する：

$$I\{A\} = \begin{cases} 1 & A \text{ が起きるとき} \\ 0 & A \text{ が起きないとき} \end{cases}$$

指標確率変数（indicator random variables）

- 確率と期待値との間を行き来しやすいような方法。
- 標本空間 S と事象 A が与えられたとき、事象 A に関する指標確率変数（indicator random variables）$I\{A\}$ をこのように定義する：

$$I\{A\} = \begin{cases} 1 & A \text{ が起きるとき} \\ 0 & A \text{ が起きないとき} \end{cases}$$

- $X_A = I\{A\}$ のとき $\mathrm{E}[X_A] = \Pr\{A\}$ である。

採用問題の解析

- X を新しい助手を採用する回数を表す確率変数とする。$\mathrm{E}[X]$ を求めたい。

採用問題の解析

- X を新しい助手を採用する回数を表す確率変数とする。$\mathrm{E}[X]$ を求めたい。
- X_i を候補 i が採用される指標確率変数とする。つまり、

$$
\begin{aligned}
X_i &= I\{\text{候補 } i \text{ が採用される}\} \\
&= \begin{cases} 1 & \text{候補 } i \text{ が採用されるとき} \\ 0 & \text{候補 } i \text{ が採用されないとき} \end{cases}
\end{aligned}
$$

採用問題の解析

- X を新しい助手を採用する回数を表す確率変数とする。$\mathrm{E}[X]$ を求めたい。
- X_i を候補 i が採用される指標確率変数とする。つまり、

$$
\begin{aligned}
X_i &= I\{\text{候補 } i \text{ が採用される}\} \\
&= \begin{cases} 1 & \text{候補 } i \text{ が採用されるとき} \\ 0 & \text{候補 } i \text{ が採用されないとき} \end{cases}
\end{aligned}
$$

- よって、$X = X_1 + X_2 + \cdots + X_n$。

採用問題の解析

- X を新しい助手を採用する回数を表す確率変数とする。$\mathrm{E}[X]$ を求めたい。
- X_i を候補 i が採用される指標確率変数とする。つまり、

$$
\begin{aligned}
X_i &= I\{\text{候補 } i \text{ が採用される}\} \\
&= \begin{cases} 1 & \text{候補 } i \text{ が採用されるとき} \\ 0 & \text{候補 } i \text{ が採用されないとき} \end{cases}
\end{aligned}
$$

- よって、$X = X_1 + X_2 + \cdots + X_n$。
- そして、$\mathrm{E}[X_i] = \Pr\{\text{候補 } i \text{ が採用される}\}$。

採用問題の解析

- X を新しい助手を採用する回数を表す確率変数とする。$\mathrm{E}[X]$ を求めたい。
- X_i を候補 i が採用される指標確率変数とする。つまり、

$$\begin{aligned} X_i &= I\{\text{候補 } i \text{ が採用される}\} \\ &= \begin{cases} 1 & \text{候補 } i \text{ が採用されるとき} \\ 0 & \text{候補 } i \text{ が採用されないとき} \end{cases} \end{aligned}$$

- よって、$X = X_1 + X_2 + \cdots + X_n$。
- そして、$\mathrm{E}[X_i] = \Pr\{\text{候補 } i \text{ が採用される}\}$。
- 候補 i が候補 1 から $i-1$ より適任であるときのみ採用される。候補がランダムに判断されると仮定したため、候補 i が候補 1 から $i-1$ より適任である確率は $1/i$ であり、以下が成り立つ：

$$\mathrm{E}[X_i] = 1/i$$

採用回数の期待値

$\mathrm{E}[X]$ を求める。

$$\begin{aligned}\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\ &= \sum_{i=1}^{n} \mathrm{E}[X_i] \\ &= \sum_{i=1}^{n} \frac{1}{i} \\ &= \ln n + O(1)\end{aligned}$$

したがって、平均時の費用は $O(c_h \ln n)$ である。最悪時の $O(c_h n)$ より安い。

乱択アルゴリズム

- 候補が適任でない順に紹介される最悪の場合が起こらないようにしたい。
- 紹介される候補をランダムに選択する、**乱択アルゴリズム（randomized algorithm）**を採用。
- （確率的にみて）平均時の効率が保証される。

RANDOMIZED-HIRE-ASSISTANT(n)

1 randomly permute the list of candidates
2 HIRE-ASSISTANT(n)