KYOTO UNIVERSITY

アルゴリズムとデータ構造⑤
～ 順序統計量・動的計画法 ～

鹿島久嗣

DEPARTMENT OF INTELLIGENCE SCIENCE AND TECHNOLOGY

順序統計量

順序統計量：
小さい方からk番目の要素は線形時間で発見可能

- 順序統計量：小さい方からk番目の要素
- 自明なやり方：ソートを使えば$O(n \log n)$
- 工夫すれば$O(n)$で可能：
  - 平均的に$O(n)$で見つける方法　（前回）
  - **最悪ケースで$O(n)$で見つける方法**　（今回）

  の二つのやり方を紹介する

平均$O(n)$の順序統計量アルゴリズム：クイックソートと同じ考え方で可能だが、最悪ケースで$O(n^2)$

- $q \leftarrow \mathrm{Partition}(A, p, r)$の結果：

配列$A[p, r]$を枢軸以下の要素と大きい要素に分け qの前後に分割

1. $k \leq q$ であれば、求める要素は$A[p\colon q]$にある
2. $k > q$であれば、求める要素は$A[q+1\colon r]$にある

– 再帰的にPartitionを呼ぶことで範囲を限定していく

- 平均的には問題サイズは半々になっていくので$O(n)$

$$T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n)$$

- 最悪ケースでは問題サイズは定数しか減らないので$O(n^2)$
- 問題サイズが確実に比率で減っていくようにしたい

最悪$O(n)$の順序統計量アルゴリズム：うまく「だいたい真ん中」をとってくる

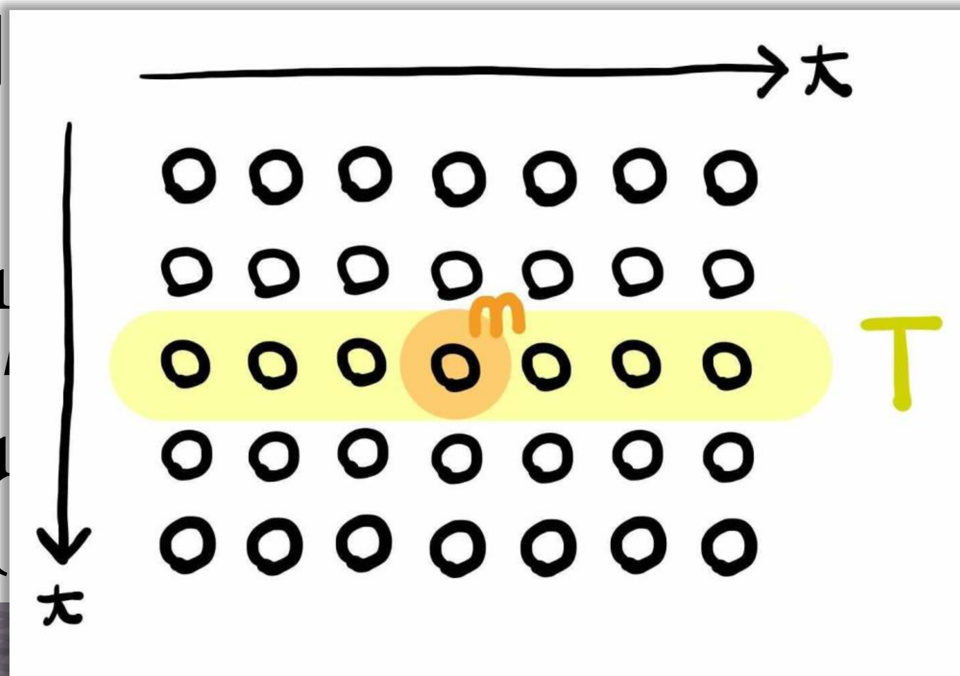
- $\mathrm{Order}(A,k)$：Aの中からk番目に小さい要素を見つける

1. Aを5個ずつのグループに分け、各グループをソートして中央値（3番目の値）を見つけ、これらを集めてTとする（定数個の要素のソートは定数時間でできることに注意）
2. Tの中央値mをみつける $\mathrm{Order}(T,\lfloor n/10 \rfloor)$
3. Aをmより小さいもの（S_1）、同じもの（S_2）、大きいもの（S_3）に分割する
4. (i) $k \le |S_1|$ならば$\mathrm{Order}(S_1,k)$を実行
   (ii) $|S_1| < k \le |S_1| + |S_2|$ ならば mは目的の要素
   (iii) $k > |S_1| + |S_2|$ならば
   $\mathrm{Order}\big(S_3, k-(|S_1|+|S_2|)\big)$を実行する。

最悪$O(n)$の順序統計量アルゴリズム：うまく「だいたい真ん中」をとってくる

- $\mathrm{Order}(A, k)$：Aの中からk番目に小さい要素を見つける

1. Aを5個ずつのグループに分け、各グループをソートして中央値（3番目の値）を見つけ、これらを集めてTとする（定数個の要素のソートは定数時間でできることに注意）
2. Tの中央値mをみつける $\mathrm{Order}(T, \lfloor n/10 \rfloor)$
3. Aをmより小 S_2）、大きいもの（S_3）
4. (i) $k \le |S_1$
   (ii) $|S_1| <$ 目的の要素
   (iii) $k > |S_1$
   Order(する。

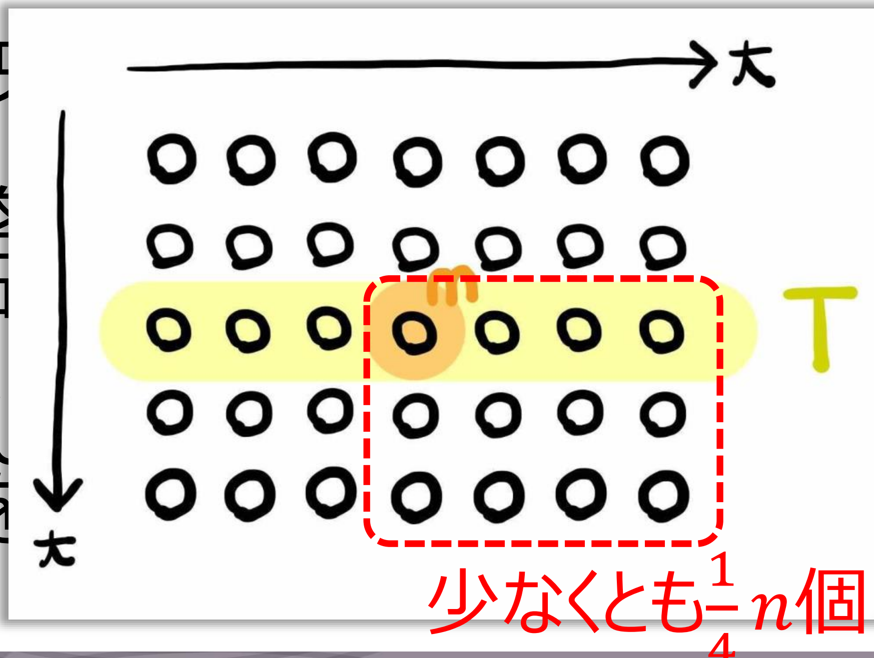


最悪$O(n)$の順序統計量アルゴリズム：計算量の漸化式

- $T(n) = O(n) + T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(\left\lceil \frac{3}{4}n \right\rceil\right) = O(n)$
  - ステップ4の分岐で (i)$\mathrm{Order}(S_1, k)$ が選ばれたとする
  - 中央値m以上の要素が少なくとも$\frac{1}{4}n$個ある
  - したがって中央値より小さい要素数は最大$\frac{3}{4}n$個
- 直観的には「各グループの中央値を集めた中の中央値は概ね全体の中央値になっている」
  - 全体を分割した小グループのそれぞれの中央値をあつめてその中央値をとると、おおむね全体の中央値が取れるはず

最悪$O(n)$の順序統計量アルゴリズム：計算量の漸化式

- $T(n) = O(n) + T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(\left\lceil \frac{3}{4}n \right\rceil\right) = O(n)$
  - –ステップ4の分岐で (i)$\mathrm{Order}(S_1, k)$ が選ばれたとする
  - –中央値m以上の要素が少なくとも$\frac{1}{4}n$個ある
  - –したがって中央 … 最大$\frac{3}{4}n$個
- 直観的には「各 … た中の中央値は概ね全体の中 …
  - –全体を分割し … の中央値をあつめてその中央値を … 央値が取れるはず



最悪$O(n)$の順序統計量アルゴリズム：計算量の導出

- 定理：$s_1 + s_2 + \cdots + s_d < 1$として
$T(n)$
$$= \begin{cases} c & (n \le n_0) \\ T(s_1 n) + T(s_2 n) + \cdots + T(s_d n) + c'n & (n > n_0) \end{cases}$$
とするとき、$T(n) \le \dfrac{cn}{1-(s_1+s_2+\cdots+s_d)} = O(n)$

- 今回のケースでは$s_1 = \dfrac{1}{5}, s_2 = \dfrac{3}{4}$であり、☝の定理を使うと
$T(n) = O(n)$

動的計画法

動的計画法：
問題を再帰的に分割しボトムアップに解く

- 動的計画法と分割統治法はともに問題を再帰的に分割
  – 分割統治法：トップダウン
  – 動的計画法：ボトムアップ
- 動的計画法の流れ
  1. 問題の構造を再帰的に捉える
  2. 解を再帰的に構成する
  3. ボトムアップで解を計算する

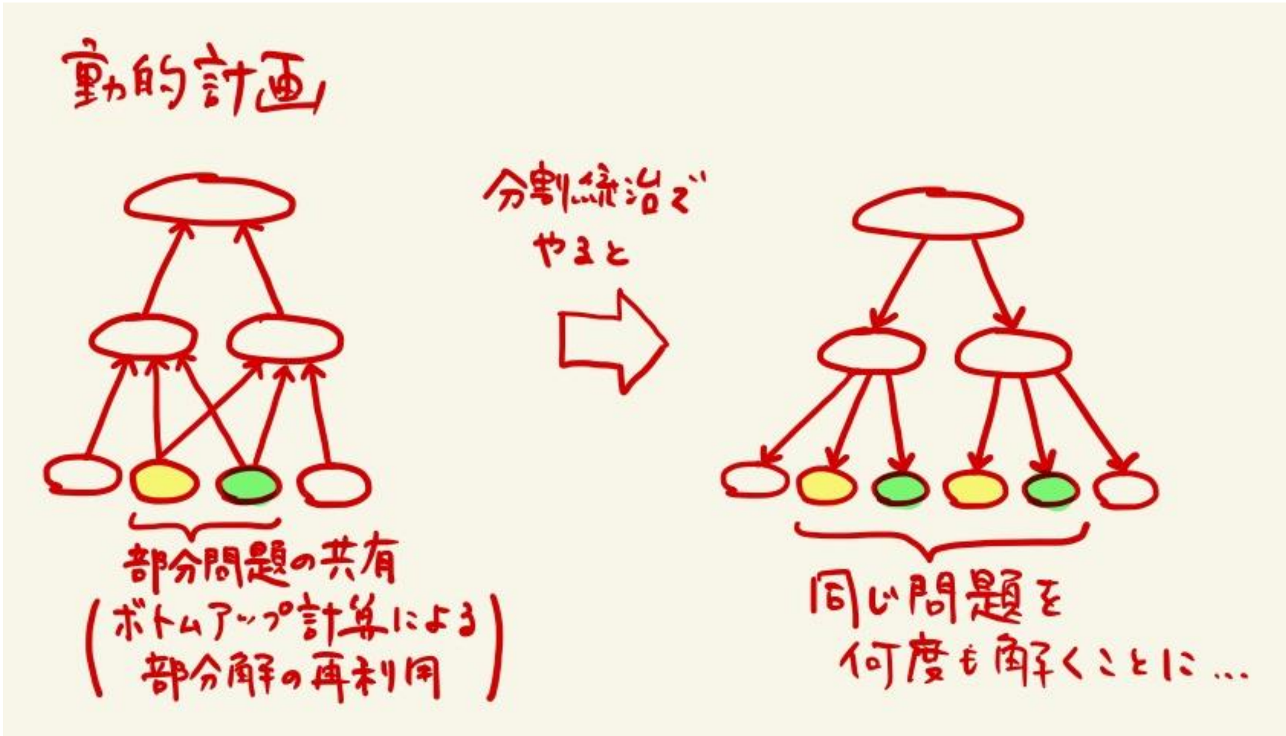
分割統治法と同じ

動的計画法のポイント：
解の使いまわしによる効率化

- 分割された問題が重複している場合に差が生じる
  - トップダウンでは同じ問題を何度も解くことになる
  - ボトムアップでは解の使いまわしが可能
  - 両者に指数的な差が生じうる
- 逆にいえば、部分問題が重複していることが動的計画法のカギ

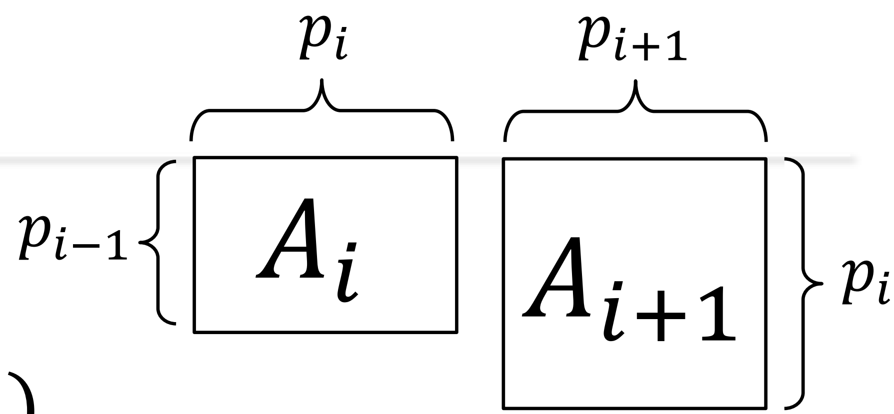
動的計画法のポイント：解の使いまわしによる効率化

- 分割された問題が重複している場合に差が生じる



- 部分問題が重複していることが動的計画法のカギ

動的計画法の例：複数の行列積の計算



- 入力： 行列 $A_1, A_2, \ldots, A_n$
- 出力： 積 $A_1 A_2 \cdots A_n (= A_{1,\ldots,n})$
- 掛け算できる前提
  - $A_i A_{i+1}$の計算は$O(p_{i-1}\, p_i\, p_{i+1})$かかる
- $A_1 : 10 \times 100, A_2 : 100 \times 5, A_3 : 5 \times 50$とすると：
  - $\left((A_1 A_2) A_3\right) = 7500$
  - $\left(A_1 (A_2 A_3)\right) = 75000$

解の構成における観察：全体の最適解は部分最適解の組合せで作られている

- $A_1 A_2 \cdots A_n (= A_{1,\ldots,n})$ の計算において、k番目の分割が最後にくるとする
  - つまり $A_{1,\ldots,k}$と$A_{k+1,\ldots,n}$を別々に計算して最後に統合する
    - ちなみに、最後の統合コストは$O(p_0\ p_k\ p_n)$
- これが最適解であるなら、$A_{1,\ldots,k}$ と $A_{k+1,\ldots,n}$の計算コストもそれぞれ最小のはず（部分問題の最適解のはず）
  - 理由：そうでなければ、それぞれをコスト最小のものに置き換えるだけで、全体のコストがもっと下がるはず
- つまり、全体最適な解は、部分最適な解でつくられている

最小コストについて成り立つ再帰式：部分行列積の最適解を組合わせて大きな最適解をつくる

- $A_{i,\ldots,j}$を計算する最小のコストを $m[i,j]$ とする
- $m[i,j]$は再帰的に表現できる：

$$m[i,j] = \begin{cases} 0 & (i = j) \\ \min\limits_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1}\, p_k\, p_j & (i \ne j) \end{cases}$$

どこで分割するのが最良か？

前半の最小計算コスト

後半の最小計算コスト

統合（掛け算）のコスト

動的計画法の計算量：ボトムアップ計算により多項式時間で解が求まる

- 再帰式の適用により最小の$m[1, n]$を求める
- 「ボトムアップ」で計算：
  1. $i = j$の場合について計算（全部ゼロ； $m[i, i] = 0$）
  2. $i = j - 1$の場合について計算（ $m[i, i + 1]$ ）
  3. $i = j - 2$の場合について計算（ $m[i, i + 2]$ ）
  4. …
  - 上記がnステップ、それぞれで$O(n)$個の再帰式評価、それぞれの評価に$O(n)$ 必要なので、全部で$O(n^3)$の計算量
- バックトラック：各再帰式での最良のkを記憶しておくことで実際の掛け算の順番を得る

動的計画法でやらないと…：
指数的な計算量になる

- 解きたい再帰式：

$$m[i,j] = \begin{cases} 0 & (i = j) \\ \min\limits_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1}\, p_k\, p_j & (i \ne j) \end{cases}$$

- トップダウン計算（分割統治）だと指数的な計算量になる

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k) + c) = 2 \sum_{k=1}^{n-1} T(k) + cn = O(2^n)$$

最長共通部分系列問題：
２つの系列に共通に含まれる最長の部分系列をみつける

- 系列$X = (x_1, x_2, \ldots, x_m)$の部分系列(subsequence)とは$X$からいくつかの要素を取り除いたもの

- 2つの系列XとYに対して、系列Zが両方の部分系列のときこれを共通部分系列とよぶ

- 最長共通部分系列(LCS; Longest Common Sequence)：
  - 入力：2つの系列
    $$X = (x_1, x_2, \ldots, x_m), Y = (y_1, y_2, \ldots, y_n)$$
  - 出力： XとYのLCS $Z = (z_1, z_2, \ldots, z_k)$ をひとつ

最長共通部分系列の性質：LCSも再帰的な構造をもつ

- $Z = (z_1, z_2, \dots, z_k)$ を XとYの任意のLCSとすると

① $x_m = y_n$ のとき、$x_m = y_n = z_k$で、
また、$Z_{k-1} = (z_1, z_2, \dots, z_{k-1})$は$X_{m-1}$と$Y_{n-1}$のLCS

- つまり、最後の文字が一致していれば、
  $x_m = y_n = z_k$ としてLCSを1つ伸ばせる

② $x_m \neq y_n$ かつ $z_k \neq x_m$ならば、ZはX_{m-1}とYのLCS

③ $x_m \neq y_n$ かつ $z_k \neq y_n$ならば、ZはXとY_{n-1}のLCS

- つまり、一致していなければ、単にスキップ

LCSの長さについて成り立つ再帰式：動的計画法により$O(mn)$で計算できる

- X_iとY_jとのLCSの長さを$c[i,j]$とする

- $c[i,j] = \max$

$$
\begin{cases}
0 & (i = 0 \text{ または } j = 0) \\
c[i-1, j-1] + 1 & (x_i = y_j) \leftarrow ① \\
\max\{c[i, j-1], c[i-1, j]\} & (x_i \neq y_j) \leftarrow ②③
\end{cases}
$$

- $c[m,n]$が$O(mn)$で求まる

解の構成：
実際の解はバックトラックで求まる

- LCSはバックトラックで構成できる
  - 注：LCSは複数ありうる
    （最適な経路は複数ありうる）
- 例：
  - $X = (a, b, c, b, d, a, b)$
  - $Y = (b, d, c, a, b, a)$
  - $Z = (b, c, b, a)$