アルゴリズムとデータ構造

- 第14回講義トピック:動的計画法(Dynamic Programming)
  - 動的計画法と最適化原理
    - Dynamic Programming and
    - Bellman's Principle of Optimality
  - 動的計画法の例
    - Example: Matrix Chain Multiplication
    - Example: Longest Common Subsequence

Bellman's Principle of Optimality

- An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Quiz 1

- 最適化原理を日本語に訳せ：
  最適解は最初の一歩を踏み出した時、解の残りの部分も最適解でなければならない。

- もっと簡単に説明すると
  問題解決のためにスタートS（状態、場所など）からゴールGまでの最適解（パス）SGが与えられたとする。
  Pが最適パスSGに含まれるならば、パスSPとPGはそれぞれ部分問題SPとPGの最適パスである。

Dynamic Programming (DP)

- 動的計画法は、「分割統治」方法の一種である。
  - 大きい問題(original problem)を、小さい部分問題(sub-problem)に分割する。
  - 部分問題も十分小さくない場合、それを更に分割し、問題のサイズを**段階的に引き下げる**。
  - それぞれの小さい問題を解決し、その**結果を保存する**。
  - 小さい問題の解を元に、大きい問題を解決する。
  - 問題のサイズを段階的に引き上げる。
- 動的計画法の効果
  - 小さく分割することで、問題が解きやすくなる。
  - 小さい問題の解をセーブし、繰り返して求めないので、計算効率が向上できる。

二項係数(binomial coefficients)の例

- nの集合からkの組み合わせ取り出す二項係数を考える。ただし、*k* と*n* は任意の正の整数である。
  (n,k) = n!/{k!(n-k)!}

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} n \\ n \end{bmatrix} = 1 \qquad \begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

二項係数(binomial coefficients)の例

- 再帰によるアルゴリズム(中間結果を保存しない場合)

Recursive-Binomial(n, k)

```
if either k = 0 or k = n
then output value 1
else
    u ← Recursive-Binomial(n-1, k)
    v ← Recursive-Binomial(n-1, k-1)
    output value u + v
```

＊このアルゴリズムはフィボナッチの再帰計算と同じように、計算に無駄が多く、非効率的である。

二項係数(binomial coefficients)の例

- 計算の中間結果を保存するDatabase(n,k)を取り入れる

Dynamic-Binomial(n, k)

```
if Database(n, k) is already defined
    then output Database(n, k)
    else
        if either k = 1 or k = n
            then Database(n, k) ← 1
            else
                u ← Dynamic-Binomial(n-1, k)
                v ← Dynamic-Binomial(n-1, k-1)
                Database(n, k) ← u + v
        output Database(n, k)
```

中間結果を保存することで、計算コストを抑えることができる。

行列チェーン乗算（matrix-chian multiplication）の例

- 行列のチェーン < $A_1,A_2,...,A_n$ > が与えられたとして、積($A_1*A_2*...*A_n$)を求める問題。
- p1xp2の行列とp2xp3の行列の積を求めるためには1xp2xp3回の乗算が必要である。
- 1からnまで順番に計算するとコストが大きくなる可能性があるので、コストが一番小さい計算順番を決めたい。
- 計算順番は、括弧で決める。例えば、$A_1*A_2*A_3$は以下のような順番で計算できる:

  順番1:　$(A_1*A_2)*A_3$　　順番2:　$A_1*(A_2*A_3)$

Quiz 2

- 行列A_1, A_2, A_3のサイズをそれぞれ30x35, 35x15, 15x5として、前のページにある方法1と方法2の計算量を求めよ。

方法1: $(A_1*A_2)*A_3$ 30x35x15+30x15x5 = 15750+2250

方法2: $A_1*(A_2*A_3)$ 30x35x5+35x15x5 = 5250+2625

最適化原理による行列チェーン乗算の分割統治

- A_1*A_2*…*A_nは、以下のように分割することができる:

  (A_1*A_2*…*A_i)*(A_{i+1}*A_{i+2}*…A_n)
- 一般的に、以下のような部分問題を考える:

  $A_{i..j}$=(A_i*A_{i+1}*…*A_j)

  i=jの場合、$A_{i..i}$ =A_iなので、コストは0となる。

  i<jの場合、上記の部分問題を更に分割することができる。

最適化原理による行列チェーン乗算の分割統治

- $A_{i..j}=(A_i*A_{i+1}*...*A_j)$をさらに$(A_{i..k})*(A_{k+1..j})$に分割したとする。
- $A_{i..j}$を計算するためのコストは、
  - $R_1=(A_{i..k})$を計算するコスト
  - $R_2=(A_{k+1..j})$を計算するコスト
  - R_1*R_2の計算コスト

  の和となる。
- 従って、ある部分問題を解決するコストは、それより更に小さい部分問題のコストから再帰的に計算できる。

最適化原理による行列チェーン乗算の分割統治

- 行列A_iのサイズをp_{i-1}xp_iとし、$A_{i..j}$の最小計算コストをm[i,j]とする。問題全体のコストはm[1,n]である。
- 最適化原理から、m[i,j]は以下のように再帰的に計算できる

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

行列チェーン乗算問題の擬似コード

Matrix-Chain-Order(p)

```
n ← length[p]-1
for i ← 1 to n
        do m[i, i] ← 0
for l ← 2 to n
        do for i ← 1 to n-l+1
                do j ← i+l-1
                    m[i, j] ← ∞
                    for k ← i to j-1
                        do q ← m[i,k] + m[k+1, j] + p_{i-1}p_kp_j
                            m[i,j] = min(m[i,j], q)
```

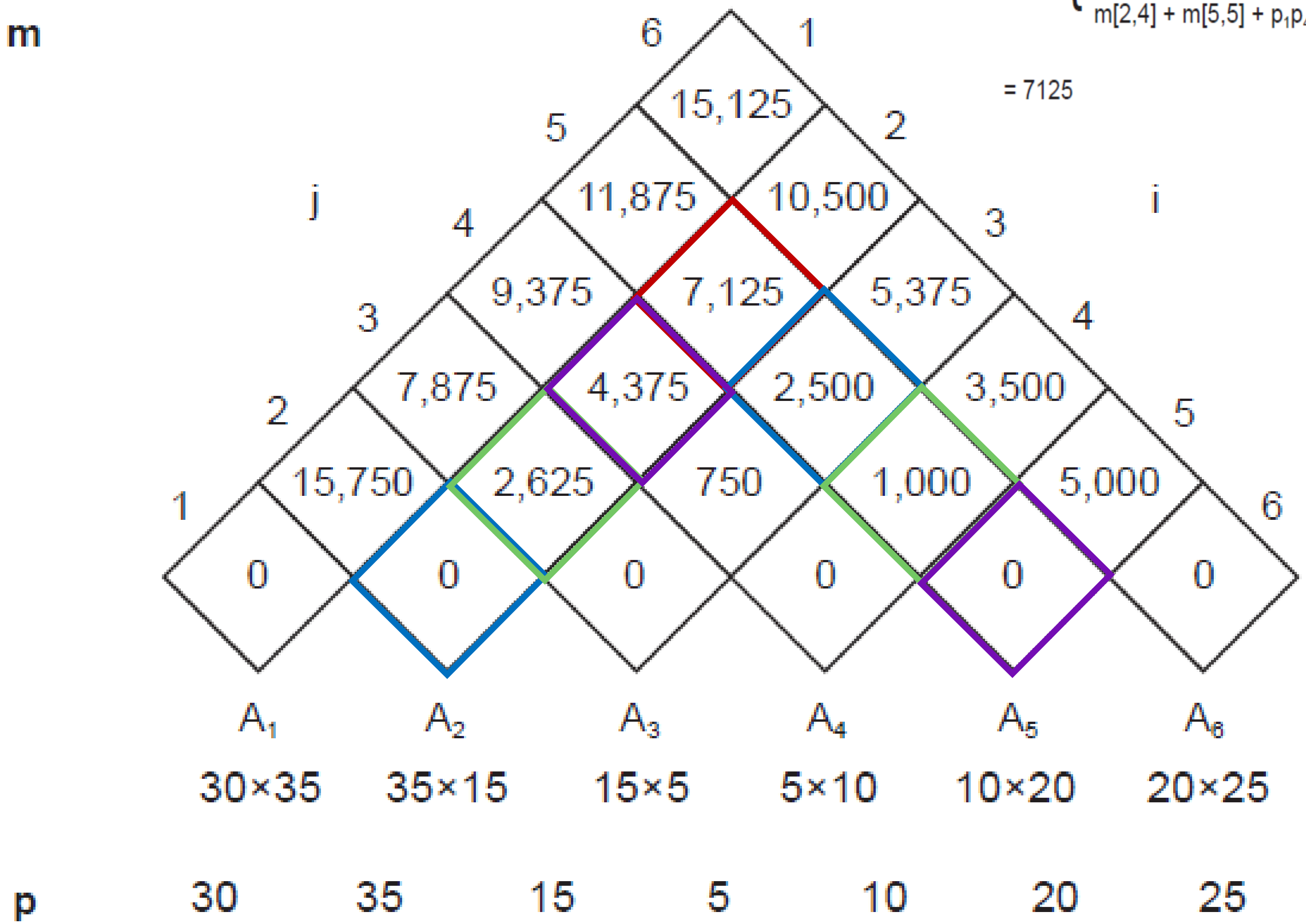
l: マトリクスチェーンのサイズを表し、隣り合う同士から徐々に距離を広げる。
j-i = l-1

m[i,k], m[k+1,j]はすでに計算済みである。

行列チェーン計算のピラミット

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1p_2p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ m[2,3] + m[4,5] + p_1p_3p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ m[2,4] + m[5,5] + p_1p_4p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

= 7125



質問：A1*A2*…A6を計算する最小コストm[1,6]は上記のプログラムで計算できるが、最適な計算順は、どのように求められるのでしょうか？

Longest Common Subsequence (LCS問題)

- LCS問題は、与えられた2つの文字列X = <x_1,x_2,…,x_m>とY= <y_1,y_2,…,y_n>に含まれる最も長い共通部分列を求める問題である。
- 例えば、
  - X = <A, B, C, B, D, A, B>
  - Y = <B, D, C, A, B, A>

  で与えられた場合、<B, C, A> はXとYの共通列であるが、最長のものではない。最長のものは、<B, C, B, A>である。
  ＊各自でご確認ください

最適化原理によるLCS問題の分割統治

- まず、両方の最後の文字を調べる。
- $x_m = y_n$なら、$X_{m-1} = <x_1,x_2,\ldots,x_{m-1}>$ と$Y_{n-1} = <y_1,y_2,\ldots,y_{n-1}>$のLCSを求め、その結果プラス1が全体の結果となる。
- $x_m \neq y_n$なら、
  - X_{m-1}とYのLCS
  - XとY$_{n-1}$のLCS

  を求める、その中でより長い方が全体の結果なる。
- 以上のことを再帰的に行う。

最適化原理によるLCS問題の分割統治

- c[i,j]を文字列$X_i = \langle x_1, x_2, \ldots, x_i \rangle$ と$Y_j = \langle y_1, y_2, \ldots, y_j \rangle$のLCSの長さとする。
- c[i,j]は以下のように再帰的に計算でいる:

$$c[i, j] = \begin{cases} 0 & \text{if } = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

- c[i,j]全部でmxn通りの組み合わせしかないので、動的計画法を使ってLCS問題をbottom-up的に解決することができる。

LCSの擬似コード

文字列の長さを徐々に増やしていく

LCS-Length(X, Y)

m ← length[X]
n ← length[Y]
for i ← 1 to m
  do c[i,0] ← 0
for j ← 0 to n
  do c[0,j] ← 0
for i ← 1 to m
  do for j ← 1 to n
    do if $x_i = y_j$
      then c[i,j] ← c[i-1, j-1] + 1
      else if c[i-1,j] ≧ c[i, j-1]
        then c[i,j] ← c[i-1,j]
        else c[i,j] ← c[i,j-1]

c[i-1,j], c[i,j-1]及びc[i-1,j-1]はすでに計算済みである。