数理工学セミナー (離散数理) Day7

harsaka

金 2

2019/11/29

Abstract

- 前回 (僕は寝坊していました、ごめんなさい) あたりからアルゴリズムを抽象的に捉える運動が始まりました. 例として, 演算のダブリング (doubling) が行列累乗などにも拡張できてうれしいという話が前回範囲に含まれていました.
- 今回はまず, これまで扱った加算や GCD のような演算が定義できる『多項式』というクラスについて考えます.
- それから環・半環の性質の話をします.
- 最後に半環上の行列演算の応用例としてソーシャルネットワークと最短経路問題に触れます.

目次

目次

(1 元) 多項式について

定義 A1

多項式とは, $\sum_{k=0}^{n} a_k x^k$ の形で表される式のことである.

(1 元) 多項式について

定義 A1

多項式とは, $\sum_{k=0}^{n} a_k x^k$ の形で表される式のことである.

例:$4x^4 + 7x^3 - x^2 + 27x - 3$

(1 元) 多項式について

定義 A1

多項式とは, $\sum_{k=0}^{n} a_k x^k$ の形で表される式のことである.

例:$4x^4 + 7x^3 - x^2 + 27x - 3$
→係数のベクトル (この例だと {4, 7, -1, 27, -3}) で多項式を表現できる！

多項式について

ホーナー法

$\sum_{k=0}^{n} a_k x^k = (\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_2)x + a_1)x + a_0$

多項式の具体的な計算において, x の 2 乗以上の累乗を行わなくてよい.
n 次多項式の場合, n 回の乗算と n 回以下の加算 (係数が 0 の個数を m とすれば $n - m$ 回だが係数が 0 かどうかを判定するより 0 を加算する方が簡単) により計算できる.

polynomial_value 関数

ホーナー法に基づくと多項式評価関数は次のように実装される (リスト 8-1)

```
template <InputIterator I, Semiring R>
R polynomial_value(I first, I last, R x) {
    if (first == last) return R(0);
    R sum(*first);
    while (++first != last) {
        sum *= x;
        sum += *first;
    }
    return sum;
}
```

係数データについて, array や vector では $a[3]$ のようにアクセスしたが, ここではイテレータを用いて値を参照している.

I の要件

前項の polynomial_value 関数で用いた引数の型について考える.
$first$ や $last$ はループ処理のためのイテレータであり, それらの値 ($*first$ のようにアクセス可能) は係数に型 R の単位元を掛けたものと解釈できる.
I の型は不等 (!=) の評価が可能でありポインタとしての機能を期待されている一方で I の値の型は R の型に等しく, 係数の型が実数だからといって変数 x の型 R が実数である必要はない.

I の要件

例)$n \times n$ 行列 X を変数とする多項式 $3X^3 - X + 10I$(I:単位行列) について考える. 係数データは {3, 0, -1, 10} だが, この場合の polynomial_value 関数の実装においては例えば $*first = 3I$ となることが要請されていることが分かる.

R の要件

問題 8.1:R の要件

- 値渡し, 代入が可能.
- 加減算'+' が可能 (負の値も足すことができる).
- 乗算'×' が可能.
- 単位元 1_R が存在.(係数 1 のときの計算に必要)
- 零元 0_R が存在.(係数 0 のときの計算に必要)
- $1x = x1 = x, 0x = x0 = 0$
- $x + y = y + x$ (和の交換法則)
- $x(yz) = (xy)z$ (積の結合法則)
- 以下の分配法則が成立する.

$$\begin{cases} a \times (b + c) = a \times b + a \times c \\ (a + b) \times c = a \times c + b \times c \end{cases}$$

多項式の加減算, 乗算

定義 A2

n 次多項式 $\sum_{k=0}^{n} a_k x^k$ を $a(x,\ n)$ で表すとする.

定理 B1:多項式の加減算

$c(x, \max(n,\ m)) = a(x,\ n) \pm b(x,\ m)$
$\Leftrightarrow c_k = a_k \pm b_k\ (k = 0, \cdots \max(n, m))$
ただし, $k \in (\min(n,\ m), \max(n,\ m)]$ において次数の低い方の係数は 0 とみなす.

定理 B2:多項式の乗算

$c(x, n+m) = a(x,\ n) \times b(x,\ m) \Rightarrow c_k = \sum_{k=i+j} a_i b_j\ (k = 0, \cdots n+m)$

多項式の除算

定義 8.1

多項式 a の次数を $\deg(a)$ と表す.

定義 8.2

以下のような多項式 q, r が存在するとき, 多項式 a を多項式 b で割ることができ, 商は q で余りは r となる.

$$a = bq + r \ \wedge \ \deg(r) < \deg(b)$$

問題 8.2

(1)

$$\begin{aligned} p(x) := q(x) \cdot (x - x_0) + r \Rightarrow p(x_0) &= q(x_0) \cdot (x_0 - x_0) + r \\ &= q(x_0) \cdot 0_R + r \\ &= r \end{aligned}$$

(2) 対偶を示す.

$$\begin{aligned} p(x) \neq q(x) \cdot (x - x_0) \Rightarrow p(x_0) &\neq q(x_0) \cdot (x_0 - x_0) \\ &= q(x_0) \cdot 0_R \\ &= 0 \end{aligned}$$

問題 8.3

(1)$4x^2 - 24x + 35$
(2)$x^2 + 2x + 1$
(3)$x - 1$
(1), (2) はプログラムにより求めたが愚直に除算しても等価な式を得られる (定数倍に対し自由度があることに注意). (3) については以下の通り実際に除算を繰り返すことで求められる.

$$nx^{n+1} - (n+1)x^n + 1 = (nx - (n+1))(x^n - nx + (n-1)) - n(x-1)$$
$$(x^n - nx + (n-1)) = (\sum_{k=1}^{n-1} x^k - n + 1)(x-1)$$

従って $(x-1)$ が答えとなる. 多項式の GCD は共通因数の積ということになる.

目次

定義 8.3

環 (ring) とは以下が定義される集合であり,

- 演算: $x + y,\ -x,\ xy$
- 定数: $0_R,\ 1_R$

以下の公理が成り立つ.

- $x + (y + z) = (x + y) + z$
- $x + 0 = 0 + x = x$
- $x + -x = -x + x = 0$
- $x + y = y + x$
- $x(yz) = (xy)z$
- $1 \neq 0$
- $1x = x1 = x,\ 0x = x0 = 0$
- $x(y + z) = xy + xz,\ (y + z)x = yx + zx$

環について

環の例

- 整数
- 実数を要素とする正方行列
- ガウス整数
- 整数係数多項式

$xy = yx$ が成立する環を可換環, 成立しない環を非可換環という.
例えば, 多項式環は非可換環である (変数に正方行列を乗せた多項式を考えると明らか).

定義 8.4

以下のような元 x^{-1} が存在するとき, 環の元 x は可逆であるという.

$$xx^{-1} = x^{-1}x = 1$$

定義 8.5

環の可逆元はその環の単元 (unit) と呼ばれる.

問題 8.4:ガウス整数の環の単元を全て求めよ.

$\pm 1,\ \pm i.$
(略称)
$\mathbb{Z}[\sqrt{-1}]$ の元は 0 を除いて絶対値が 1 以上であるから, 元が単元ならば絶対値は 1 である. したがってこの 4 つの元しか単元の候補とならないがこれらは全て単元であることが確かめられる. □

問題 8.5:1_R が単元であることと, 単元の逆元が単元であることを示せ.

- 単位元の公理より, $1_R \times 1_R = 1_R$ であるから定義 8.4 より 1 は可逆であり, $1^{-1} = 1$ である.
- x を単元とし x の逆元を y とすると, $xy = yx = 1$ が成立するので $yx = xy = 1$ より定義 8.4 より y も単元であり $y^{-1} = x$ である.

定義 8.6

以下の条件を満たす環の元 x を零因子 (zero devisor) と呼ぶ.

- $x \neq 0$
- $\exists y \neq 0,\ xy = 0$

定義 8.7

零因子を持たない可換環を整域 (integral domain) と呼ぶ.

問題 8.6:零因子が単元でないことを示せ

(証明)
ある元 x が零因子でかつ単元であると仮定すると, ある元 $y \neq 0$ について $xy = 0$ であり, ある元 z について $xz = zx = 1$ が成立する.

$$\begin{aligned} zx = 1 &\Rightarrow (zx)y = 1y \\ &\Rightarrow z(xy) = y \\ &\Rightarrow z0 = y \\ &\Rightarrow y = 0 \end{aligned}$$

これは $y \neq 0$ に矛盾. □

定義 8.8

半環 (semiring) とは以下が定義される集合であり,

- 演算: $x + y,\ xy$
- 定数: $0_R,\ 1_R$

以下の公理が成り立つ.

- $x + (y + z) = (x + y) + z$
- $x + 0 = 0 + x = x$
- $x + y = y + x$
- $x(yz) = (xy)z$
- $1 \neq 0$
- $1x = x1 = x$
- $0x = x0 = 0$
- $x(y + z) = xy + xz,\ (y + z)x = yx + zx$

半環, 弱半環

- 半環は環からマイナス (加法の逆演算) を除いた代数構造である. 例えば自然数の集合は半環である.
- 加法の単位元 0 および乗法の単位元 1 とそれらに対応する公理を削除した半環を弱半環 (weak semiring) と呼ぶことがある.
- 非負整数を要素とする行列における行列乗算は弱半環上でも逐次計算することは可能である (高速な行列累乗には乗法の単位元が必要).

目次

boolean 半環上の行列累乗

問題 8.7:bool 値の隣接行列から連結成分を導出.

A:B, C, D, F
B:A, C, D, F
C:A, B, D, F
D:A, B, C, F
E:G
F:A, B, C, D
G:E

boolean 半環への修正

問題 8.7:行列累乗の変更点

ソースコードにおいて, 実数行列の行列累乗を matrix_pow 関数で与えているが, そこで用いている matrix_mul 関数の更新式を以下の通り変更することでこの問題を解くための bool_pow 関数を作ることができる.

$$\mathrm{ret}[i][j] = \mathrm{ret}[i][j] + (\mathrm{l}[i][k] \times \mathrm{r}[k][j]);$$

$$\downarrow$$

$$\mathrm{ret}[i][j] = \mathrm{ret}[i][j] \mid (\mathrm{l}[i][k] \ \&\ \mathrm{r}[k][j]);$$

ソースコードの transitive_closure 関数あたりを参照.

tropical 半環上の行列累乗

問題 8.8:有向辺の重みの隣接リストから全点対最短距離の導出.

	A	B	C	D	E	F	G
A	0	6	8	3	8	7	11
B	23	0	16	26	2	10	5
C	7	13	0	10	15	14	18
D	12	18	5	0	11	4	12
E	35	12	28	38	0	22	3
F	13	17	6	16	7	0	8
G	32	9	25	35	11	19	0

tropical 半環への修正

問題 8.8:行列累乗の変更

単位元は『対角成分が 0, その他の要素が ∞ であるような正方行列』.
更新式は行列乗算のものを以下のように変更する.

$$\mathrm{ret}[i][j] = \min(\mathrm{ret}[i][j], \mathrm{l}[i][k] + \mathrm{r}[k][j]);$$

ソースコードの shortest_distance 関数あたりを参照. 発想は Warshall-Floyd と同様に, ある点 k を経由した方がパス (i, j) の長さを短縮できるならば短縮するというもの.
悲しいことにこのアルゴリズムは $O(n^3 logn)$ であり Warshall-Floyd より複雑なのに Warshall-Floyd より遅い. フライゴンか？

Warshall-Floyd

全点対最短路問題を解くアルゴリズム. おおまかには以下の通り.
d[s][t] : s から t への暫定的な最短距離としたとき, d[i][j] を $i->k->j$ の順で回ることで最短距離を更新できるならば更新するという操作を繰り返す.
単純ながら強力で O(n^3) で動作する (n:頂点数).

問題 8.9:問題 8.8 の経路復元

有向辺 $(s,\ t)$ が最短経路に使われたかどうかを調べるとよい.

```
vector<int> restore_path(int from, int to) {
    auto d = matrix_8_8;
    int n = d.size();
    auto g = shortest_distance(d, n - 1);
    vector<int> p;
    int cur = from;
    while (cur != to) {
        for (int i = 0; i < n; i++) {
            if (i != cur &&
            d[cur][i] + g[i][to] == g[cur][to]) {
                cur = i;
                p.push_back(i);
                break;
            }
        }
    }
    return p;
```

おしまい

- Thank you!
- 多項式除算 (ソースコードにおける remainder 関数) の実装が今のところめちゃくちゃなので良い実装を思いついた方は教えてください.
- 各ステップで最高次の係数を最小公倍数で合わせたら良い感じに整数係数の解が求まるとは思うのでとりあえずは気が向き次第それを実装します.