計算機科学第一

11 月 17 日

今週の目標

- Scala における変数、破壊的代入、配列、for ループなどを理解する。

今日の課題 (提出締切は今週金曜日 23 時 59 分 59 秒)

1. 再帰的な構造を持つ行列をベクトルに掛ける計算を配列を直接書き換えることによって実現する。

今日のワークフロー

1. GitHub 上の `titech-is-cs115/assignment4` にある課題を終わらせて提出する。

1 変数

Scala では可変変数 (mutable variable) は `var` で宣言します。

```
var x = 1
var y = x
x = 3
```

このとき `x` は `Int` 型の 3, `y` は `Int` 型の 1 になっています。型は省略しても構いませんが右辺の型と異なるものを指定したい場合は

```
var x:BigInt = 1
```

のようにします。値に名前を付けるには `val` をつかって

```
val z = 1
```

のように宣言します。その後で

```
z = 3
```

のように変更しようとしてもエラーになります。

2 配列

今まで Scala ではリストを主なデータ構造として扱ってきましたが C 言語のような配列も使うことができます。`import scala.collection.mutable._` としてから

```
val x = ArraySeq(1,2,3)
```

とすると 1 と 2 と 3 からなる長さ 3 の `Int` 型の配列が作られ `x` という名前になります。

```
x(0)
```

とすると最初の成分である 1 が返ってきます。

```
x(0) = 100
```

とすると配列 `x` の最初の成分を 100 に書き換えます。これは `x` を `val` で宣言していても許されます。`x` 自体に再代入することはできませんが、`x` が含んでいるデータを書き換えることはできます。

```
val y = ArraySeq("a","b","c")
```

とすると `String` 型の配列が作られて `y` という名前になります。

```
val z = ArraySeq("a",10)
```

とすると `Int` 型と `String` 型の共通の親クラスである `Any` 型の配列が作られますが今回は説明しません。こういう使い方は考えなくてよいです。

長さを指定して配列を生成したい場合は

```
val w = new ArraySeq[Int](100)
```

とします。長さ 100 の `Int` 型の配列が作られます。この場合型を明示することに気をつけてください。この配列は初期化されていないので (null が各成分に入っています) 代入してから使ってください。

配列はリストと違いランダムアクセス (任意の i 番目の要素へのアクセス) が定数オーダーでできます。しかしリストのように先頭に要素を加える操作ができません。このような配列とリストの違いは「アルゴリズムとデータ構造」で扱ったと思います。その他の大きな違いとして Scala においては `ArraySeq` は変更できます (mutable) が `List` は変更できません (immutable)。変更できるデータ構造は他にもたくさんあります `http://www.scala-lang.org/api/current/index.html#scala.collection.mutable.package`。最初におまじないのように書いた `import scala.collection.mutable._` というのは Scala のライブラリの中からデータ構造 (collection) で変更可能 (mutable) なものの定義を全て (_) import するという意味です。それぞれの型の特徴を知るには Scala の公式のドキュメントの一部の Guides and Overviews の中の `http://docs.scala-lang.org/overviews/collections/concrete-mutable-collection-classes.html` を読むとよいでしょう。

3 for ループ

Scala でも `for` ループを使うことができます。

```
val n = 8
for(i <- 0 until n) {
  println(i)
}
```

とすると 0 から 7 が表示されます。他にもデータ構造を使って

```
val a = ArraySeq(1,10,100)
for(i <- a) {
  println(i)
}
```

とすると 1, 10, 100 が表示されます。`ArraySeq(1,10,100)` の部分は `List(1,10,100)` など他のデータ構造でも構いません。

4 今日の課題

行列 A_{2^k} を

$$A_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$A_{2^k} := \begin{bmatrix} A_{2^{k-1}} & A_{2^{k-1}} \\ A_{2^{k-1}} & -A_{2^{k-1}} \end{bmatrix}$$

で定義します。この行列の再帰的な構造を利用して入力 x について $O(n \log n)$ 時間で $A_n x$ を計算してください (x は長さ n のベクトル、n は 2 の冪とします)。関数 `trans(x: ArraySeq[Int], n: Int): ArraySeq[Int]` は長さ n の配列 `x` に対して A_n`x` を計算して返す関数とします (n は 2 の冪と仮定してよいです)。この計算の中では配列 `x` を破壊的に書き換えることとし、新しい配列を生成しないでください (特に `ArraySeq` 型のメンバ関数を使わなければ大丈夫です)。ヒントとしては補助関数 `trans_sub(x: ArraySeq[Int], rs: Int, re: Int): ArraySeq[Int]` としてベクトル `x` のインデックスが `rs` から `re` の範囲に $A_{\mathtt{re}-\mathtt{rs}+1}$ を適用する (`x` を破壊的に書き換える) ものを用意するとよいです。補助関数の中では関係

$$A_{2^k} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{2^{k-1}} x_1 + A_{2^{k-1}} x_2 \\ A_{2^{k-1}} x_1 - A_{2^{k-1}} x_2 \end{bmatrix}$$

を利用して再帰的に計算してください。ただしこの補助関数も破壊的に `x` を書き換えるので `trans_sub(x, rs, (rs+re)/2)` などと再帰呼出した返り値は受け取らなくてよいです。再帰呼出しの後に for ループで最終的な計算結果を導出してください。

テストの方法については特に指定しませんが、再帰的な関係や $A_{2^k}^2 = 2^k I$ となること (I は単位行列)、A_{2^k} の (i, j) 成分が $(-1)^{\langle b_i, b_j \rangle}$ となることが利用できるかもしれません。ここで b_i は $i \in \{0, 1, \ldots, 2^k - 1\}$ の二進数表現、$\langle b_i, b_j \rangle$ は b_i と b_j の内積 (popcount(i & j)) とします。