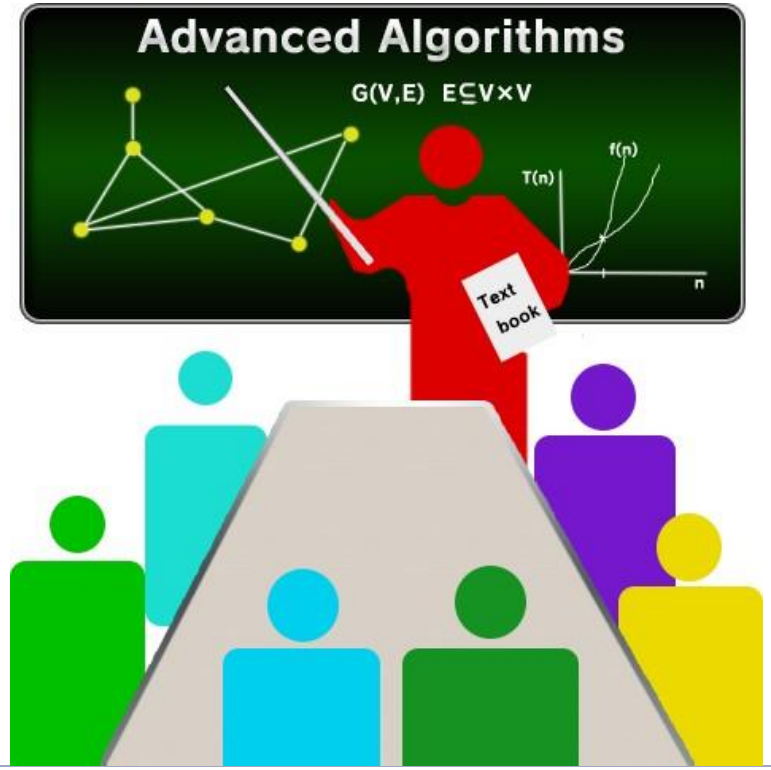

アルゴリズム特論 [AA201X]
Advanced Algorithms

Lecture12 Random number generator

Exercise 12-13のために

・コンピュータにおける乱数の発生方法
（乱数生成器のアルゴリズム）

・挙動が乱数によって決定されるアルゴリズム
（乱択アルゴリズム）

アルゴリズムで作る乱数

- 乱数列　⇒　次に現れる値が予測できない数列
- 乱数　　⇒　でたらめ（ランダム）に現れる数

- アルゴリズムで作られた乱数は、
  「本当の」乱数ではない

- アルゴリズムで作る

⇒ アルゴリズムとは「ある処理を実現するための一定の手続き」⇒ そこには**何か規則性がある**

⇒　コンピュータで作成される乱数は、
乱数っぽくみえる「**擬似乱数**」である
(Pseudo random number)

コンピュータで使われる乱数

- あるプログラムを検証するために必要なデータ列をランダムに用意する
- ゲームプログラムで、処理の流れを決定するためにサイコロを振る感覚で乱数を使う

あるアルゴリズム（規則性）で作られた乱数の性質が悪い（数の出現頻度に偏りがある、規則性が目で分かるくらい繰り返して同じ値が出る等）とマズい

原始的な乱数生成の考え方

- **線形合同法（Linear Congruential Method）による漸化式計算**

$$x_{i+1} = (Ax_i) \mathrm{mod}. M$$

- **乱数の初期値（種，seed） x0 があって、与えられた値 A と M に対して計算を繰り返すことで、次に出現する乱数の値を求める**

- x0 から始めて、x1, x2, x3, … と順々に求めていく。
- 故に、規則的な乱数（？）が生成される
- 最大周期は M-1

- いかに**各値の出現頻度を均等**にし、**より長い周期の乱数列を生み出して**、**本当の乱数のように見せかけるか**が乱数発生アルゴリズムでは重要

原始的な乱数生成の考え方

- ハンドアウトの実装の流れ

next_rnd()　：オリジナルの線形合同法

↓ 計算機だと正しく動かないので最低限のオーバーフロー対策処理 ↓

next_rnd1()　：改良版１

↓ 出現する値があまりにも偏っていて非実用的。偏りを減らしたいが，再びオーバーフロー問題に直面するため、線形合同法の式変形をさらに進めることで解決（Schrage's Algorithm と言うそうだ）　↓

next_rnd2()：改良版２

↓next_rnd1()の問題点は解決した。ただし、32bit型整数の範囲だと乱数列の周期が2^31-1=2147483647で頭打ち。さらに長い周期を生成することを求めて Subtractive method を実装 ↓

next_rnd3()：改良版３ (next_rnd2に追加操作を加える)

周期が 2^55-1の乱数列が作れるようになった

乱数の利用法

- このアルゴリズムを実装した関数で乱数を生成すると、

0 から 2147483647 までの整数値からランダムに（とはいっても漸化式に従って…だが）１つの値が抽出される。

・乱数を生成するアルゴリズム（関数）を設計するはエンジニアサイドの仕事。
・乱数をどう利用するかはユーザーサイドの仕事。
⇒　自分の希望する範囲の乱数がほしいときは、自分で変換処理を作る

例：0 から 100までの整数値が欲しい
⇒ 0～2147483647 から 0～99 までの範囲に移すような写像を定義する必要がある。
　value = my_GetRand()%100;　// value は 0 ～99の値をとる

例：0 から 1までの小数値が欲しい
⇒ 0～2147483647 から 0.0～1.0 までの範囲に移すような写像を定義する必要がある。
　value = my_GetRand() / 2147483647;　//value は 0.0 ～ 1.0 の値をとる

#define RAND_MAX　2147483647　等をプログラムに書いておくと便利かな？