Estruturas de Dados e Algoritmos

Folha de exercícios número 12

2000/2001

1. Este é um algoritmo clássico para a determinação de um valor aproximada de π através de um método probabilístico simples e interessante mas pouco preciso.

   Geram-se n pontos aleatórios $\langle x_i, y_i \rangle$ com $0 \leq x_i, y_i \leq 1$. Alguns destes n pontos, digamos m, "caem" dentro do quarto de círculo de raio 1 e centro na origem que está no primeiro quadrande (de facto, no quadrado $0 \leq x \leq 1$, $0 \leq y \leq 1$). Como a área deste quarto de círculo é $\pi/4$ e a área do quadrado é 1, vem a proporção aproximada

   $$\frac{\pi/4}{1} \approx \frac{\text{pontos no quarto de círculo}}{\text{pontos no quadrado}} = \frac{m}{n}$$

   Um valor aproximado de π (em princípio tanto mais aproximado quanto maior for n) é então $4m/n$.

   Implemente um programa que determine por este processo um valor aproximado de π e analise experimentalmente a precisão obtida em função de n. Que conclusão tira?

2. Considere o método de Rabin-Miller para averiguar se um inteiro é primo. Um inteiro n é primo sse, para todo o a com $2 \leq a < n$, é $a^{n-1} = 1 \pmod{n}$ (pequeno Teorema de Fermat). Neste problema vamos supor que se n não é primo, pelo menos metade dos inteiros a dentro daqueles limites satisfaz $a^{n-1} \neq 1 \pmod{n}$[1].

   (a) Calcule para $n = 6$ e 7 os valores de $a^{n-1} \pmod{n}$ para todos os valores de a satisfazendo $1 \leq a < n$.

   (b) Defina uma função com o protótipo

   ```
   int power(int a,int b,int n);
   ```

   que determine $a^b \pmod{n}$ utilizando um método de eficiência $O(\log b)$; supõe-se que n é fixo e que $a, b \leq n$.

   Sugestão: Se $b \geq 1$, divida b em duas partes sensivelmente iguais e utilize recursividade.

[1] Isto não é exactamente verdade, havendo contudo muito poucos inteiros compostos para os quais isso não é verdade; é necessário um teste um pouco mais complexo para que essa afirmação seja (sempre) verdadeira.

(c) Implemente um programa em linguagem C que imprima uma tabela com oo seguinte formato

```
  n    Primo?       %factores   %testemunhas
---------------------------------------------
  2     SIM             0            0
  3     SIM             0            0
...     ...           ...          ...
200     NÃO           ...          ...
```

onde `%factores` é a percentagem de divisores de n compreendidos entre 2 e $n-1$ e `%testemunhas` é a percentagem de valores a compreendidos entre 2 e $n-1$ que satisfaz $a^{n-1} \neq 1 \pmod{n}$.

(d) Defina uma função com o protótipo

```
boolean primo(int n, float eps);
```

que determine se n é primo com uma probabilidade de erro da resposta (só quando esta for "sim", caso contrário não há erro) inferior a `eps`. Teste o seu programa.

(e) Procure (prováveis) primos da forma $2^p - 1$ e $2^p + 1$.

(f) Compara as suas respostas às duas últimas alíneas e correspondente velocidade com as de um programa "clássico" de teste de primalidade (que terá que implementar!).

Nota Um processo de inicializar a função de pseudo-aleatórios:

```
srand(getpid());
```

Gerando um pseudo-aleatório inteiro entre `a` e `b`:

```
int aleatorio(int a,int b){
  return(a+(int) ((b-a+1.0)*rand()/(RAND_MAX+1.0)));
}
```

Gerando um "float" pseudo-aleatório entre 0 e 1

```
float aleatorio(void){
  return(((float)*rand())/(RAND_MAX))
}
```