Vetores

- Coleção indexada de objetos do mesmo tipo, armazenada de forma contígua

Crivo de Eratosthenes

```
/* prog3.5.c */
#include <stdio.h>
#define N 10000
int main()
  { int i, j, a[N];
    for (i = 2; i < N; i++) a[i] = 1;
    for (i = 2; i*i < N; i++)
      if (a[i])
        for (j = i; i*j < N; j++) a[i*j] = 0;
    for (i = 2; i < N; i++)
      if (a[i]) printf("%4d ", i);
    printf("\n");
    return 0;
  }
```

Exercício

Exercício 1. *Podemos trocar a condição* `i*j < N` *por* `j < N/i` *no Programa 3.5?*

▷ **Detalhe:** talvez você queira definir o tipo `bool`, para variáveis booleanas. Basta fazer

```
typedef enum {FALSE, TRUE} bool;
```

Outro exemplo

```
typedef enum {Jan = 1, Feb, Mar, [...] Dec} month;
```

Alocação dinâmica de memória

▷ `malloc()` (do `stdlib`)

```
int *a;
[...]
a = (int *)malloc(N*sizeof(int));
if (a == NULL) {
  printf("Insufficient memory.\n");
  return 1;
}
[...]
```

Crivo de Eratóstenes II

```
/* prog3.5b.c */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{ int i, j, N = atoi(argv[1]);
  int *a;

  a = (int *)malloc(N*sizeof(int));
  if (a == NULL) {
    printf("Insufficient memory.\n");
    return 1;
  }
```

Crivo de Eratóstenes II

```
/* prog3.5b.c */
  [...]
  for (i = 2; i < N; i++) a[i] = 1;
  for (i = 2; i*i < N; i++)
    if (a[i])
      for (j = i; i*j < N; j++) a[i*j] = 0;
  for (i = 2; i < N; i++)
    if (a[i]) printf("%4d ", i);
  printf("\n");
  return 0;
}
```

Exercício

Exercício 2. *Podemos trocar a condição* `i*i < N` *por* `i < N` *no Programa 3.5b?*

▷ Experimente executar o programa para verificar sua resposta. O resultado pode ser inesperado.

Complexidade do Crivo de Eratóstenes

▷ Quantas vezes a atribuição `a[i*j]=0` é executada?

◦ Algo como

$$\frac{1}{2}N + \frac{1}{3}N + \frac{1}{5}N + \frac{1}{7}N + \frac{1}{11}N + \cdots + ?$$

Melhor:

$$\sum_{p} \frac{1}{p} N,$$

onde a soma se estende sobre todo primo p tal que $p^2 < N$.

Números harmônicos

n-ésimo número harmônico ($n \geq 0$):

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{1 \leq k \leq n} \frac{1}{k}.$$

Temos

$$\log n < H_n < \log n + 1, \quad \text{para } n > 1$$

(basta comparar com $\int_1^n dx/x$).

Somas de inversos de primos

Estamos interessados em

$$M_n = \sum_{p \le n} \frac{1}{p}.$$

Claramente $M_n \le H_n$. Concluímos que o algoritmo de Eratóstenes faz $\le N(\log \sqrt{N} + 1) \le N(\log N + 1)$ atribuições `a[i*j]=0`. O algoritmo de Eratóstenes tem complexidade de tempo $O(N \log N)$. [Na verdade, sabe-se que $M_n = \log\log n + c + o(1)$, onde c é uma constante de valor $0.261497\ldots$. Assim, a complexidade é $O(N \log\log N)$.]

Pontos próximos, bis

```
/* prog3.8.c */
#include <stdio.h>
#include <stdlib.h>
#include "Point2.h"
int main(int argc, char *argv[])
 { float d = atof(argv[2]);
   int i, j, cnt = 0, N = atoi(argv[1]);
   point *a = malloc(N*sizeof(*a));
   for (i = 0; i < N; i++) a[i] = randPoint();
   for (i = 0; i < N; i++)
     for (j = i+1; j < N; j++)
       if (distance(a[i], a[j]) < d) cnt++;
   printf("%d edges shorter than %f\n", cnt, d);
   return 0;
 }
```

Pontos próximos, bis

```
[...]
point *a;
a = (point *)malloc(N*sizeof(point));
[...]
```

Discussão do Programa 3.8

▷ O Programa 3.8 tem complexidade $\Theta(N^2)$. Isto implica que ele não é viável para valores grandes de N. Faça alguns experimentos para verificar isso.

Exercício 3. *Modifique o algoritmo do Programa 3.8 para que ele seja mais rápido, especialmente para valores pequenos de* d *(digamos,* $d = 0.1$ *ou* $d = 0.01$*).*

Exercícios

Exercício 4. *Escreva um programa que gera* N *pontos aleatórios uniformemente distribuídos no quadrado unitário e determina a menor distância entre dois destes pontos.*

Exercício 5. *Escreva um programa que gera* N *pontos aleatórios uniformemente distribuídos no quadrado unitário e determina a menor área de um triângulo determinado por três destes pontos.*

▷ Procure escrever programas que podem resolver os problemas acima para valores grandes de N.

Leitura recomendada

`http://www.ime.usp.br/~pf/algoritmos/aulas/pont.html`

`http://www.ime.usp.br/~pf/algoritmos/aulas/aloca.html`

Listas ligadas

Definição 6. *Uma lista ligada é uma coleção de itens com cada item em uma célula, que também contém um ponteiro para uma célula.*

- ▷ **Em geral:** uma lista ligada é uma coleção de itens em uma seqüência linear.
- ▷ **Ponteiro especial:** `NULL` (às vezes: Λ)
- ▷ **Indicação de fim de uma lista linear:** o ponteiro da última célula contém `NULL`, um ponteiro para uma célula especial, ou um ponteiro para a primeira célula da lista (*lista circular*)

As células em C

```
typedef struct node *link;
struct node { Item item; link next; };
```

Para alocar memória para uma célula:

```
link x = malloc(sizeof *x);
```

O problema de Josephus

- Veja

  `http://mathworld.wolfram.com/JosephusProblem.html`