# Estruturas de dados compostas

Vimos “estruturas lineares simples”:

- Vetores, listas ligadas, strings

Consideramos agora “estruturas compostas”:

- Vetores de vetores (em particular, matrizes), vetores de listas ligadas, vetores de strings, listas ligadas de vetores de strings, etc.

# Matrizes

▷ Estrutura de dados já bem conhecida; alguns detalhes

Alocação dinâmica de matrizes:

```
/* prog3.16.c */
int **malloc2d(int r, int c)
  { int i;
    int **t = malloc(r * sizeof(int *));
    for (i = 0; i < r; i++)
      t[i] = malloc(c * sizeof(int));
    return t;
  }
```

# Representação de textos

```
/* prog3.17b.c */
#include <stdio.h>
#include <string.h>

#define Nmax 1000
#define Mmax 10000

char buf[Mmax]; int M = 0;
```

# Representação de textos

```
/* prog3.17b.c */
[...]
int main()
  { int i, N;
    char *a[Nmax];
    for (N = 0; N < Nmax; N++)
      { a[N] = &buf[M];
        if (scanf("%s", a[N]) == EOF) break;
        M += strlen(a[N])+1;
      }
    for (i = 0; i < N; i++) printf("%s\n", a[i]);
    return 0;
  }
```

# Representação de textos

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ cat prog3.17b.in
   chegou a hora     da
verdade

      nada como

escrever um programa de
 mac122
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

# Representação de textos

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ prog3.17b < prog3.17b.in
chegou
a
hora
da
verdade
nada
como
escrever
um
programa
de
mac122
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

# Alternativa

```
/* prog3.17c.c */
#include <stdio.h>
#include <string.h>

#define Nmax 1000
#define Mmax 10000

char buf[Mmax]; int M = 0;
```

# Alternativa

```
/* prog3.17c.c */
[...]
int main()
  { int i, N;
    char *a[Nmax];
    for (N = 0; N < Nmax; N++)
      { a[N] = &buf[M];
        if (fgets(a[N], Mmax - M, stdin) == NULL) break;
        M += strlen(a[N])+1;
      }
    for (i = 0; i < N; i++) fputs(a[i], stdout);
    return 0;
  }
```

# Exercícios

**Exercício 1.** *Como é a saída de* `prog3.17c < prog3.17b.in`*?*

**Exercício 2.** *Escreva um programa como o* `prog3.17c.c`*, mas que não impõe nenhuma restrição (artificial) no comprimento das linhas. (Lembre-se de* `getline.h`, `getline.c`*.)* Sugestão: *use um vetor de strings.*

**Exercício 3.** *Escreva um programa como o* `prog3.17c.c`*, mas que não impõe uma restrição (artificial) no número de linhas no texto. As linhas do texto devem ser organizadas em uma lista ligada.*

**Exercício 4.** *Escreva um programa como o* `prog3.17c.c`*, mas que não impõe nenhuma restrição (artificial) no comprimento das linhas nem no número de linhas no texto. (Combinação dos dois exercícios anteriores.)*

# Um exemplo com ordenação

```
/* prog3.17a.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Nmax 1000
#define Mmax 10000

#define exch(A, B) { char *t = A; A = B; B = t; }
#define compexch(A, B) if (strcmp(B, A)<0) exch(A, B)

char buf[Mmax]; int M = 0;
```

# Um exemplo com ordenação

```
/* prog3.17a.c */
[...]
void sort(char *a[], int l, int r)
  { int i, j;
    for (i = l+1; i <= r; i++)
      for (j = i; j > l; j--)
        compexch(a[j-1], a[j]);
  }
```

# Um exemplo com ordenação

```
/* prog3.17a.c */
[...]
int main()
  { int i, N;
    char *a[Nmax];
    for (N = 0; N < Nmax; N++)
      {
        a[N] = &buf[M];
        if (scanf("%s", a[N]) == EOF) break;
        M += strlen(a[N])+1;
      }
    sort(a, 0, N - 1);
    for (i = 0; i < N; i++) printf("%s\n", a[i]);
    return 0;
  }
```

# Um exemplo com ordenação

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ prog3.17a < prog3.17b.in
a
chegou
como
da
de
escrever
hora
mac122
nada
programa
um
verdade
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

# Exercício

**Exercício 5.** *Escreva um programa que recebe um texto como entrada e produz como saída o texto com as linhas rearranjadas em ordem alfabética.*

# Pontos próximos, novamente

- Já consideramos o problema da contagem de pontos próximos: dados $N$ e $d$, geramos $N$ pontos ao acaso no quadrado unitário, e queremos saber quantos pares ficam a uma distância menor que $d$.
- O algoritmo elementar que vimos (implementado em `prog3.8.c`) tem complexidade $\Theta(N^2)$, que é proibitivo para valores grandes de $N$.

# Pontos próximos, novamente

```
/* 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

- Há um algoritmo de complexidade $O(d^2N^2)$ bastante simples.
- Dividimos o quadrado unitário em quadrados menores, todos de mesmo tamanho, de forma que pontos a uma distância menor que $d$ necessariamente pertencem ao mesmo quadrado ou a quadrados adjacentes. Em média, cada ponto é comparado com $O(d^2N)$ outros.

# Pontos próximos, bis

```
/* prog3.20_alt.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "Point.h"

typedef struct node* link;
struct node { point p;  link next; };

link **grid; int G; float d; int cnt = 0;

float randFloat()
  { return 1.0*rand()/RAND_MAX; }
```

# Pontos próximos, bis

```
[...]
link **malloc2d(int r, int c)
  { int i;
    link **t = malloc(r * sizeof(link *));
    for (i = 0; i < r; i++)
      t[i] = malloc(c * sizeof(link));
    return t;
  }
```

# Pontos próximos, bis

```
[...]
void gridinsert(float x, float y)
  { int i, j; link s;
    int X = x*G + 1; int Y = y*G + 1;
    link t = malloc(sizeof *t);
    t->p.x = x; t->p.y = y;
    for (i = X-1; i <= X+1; i++)
      for (j = Y-1; j <= Y+1; j++)
        for (s = grid[i][j]; s != NULL; s = s->next)
          if (distance(s->p, t->p) < d) cnt++;
    t->next = grid[X][Y]; grid[X][Y] = t;
  }
```

# Pontos próximos, bis

```
[...]
int main(int argc, char *argv[])
 { int i, j, N = atoi(argv[1]);
   d = atof(argv[2]); G = 1/d;
   grid = malloc2d(G+2, G+2);
   for (i = 0; i < G+2; i++)
     for (j = 0; j < G+2; j++)
       grid[i][j] = NULL;
   for (i = 0; i < N; i++)
     gridinsert(randFloat(), randFloat());
   printf("%d edges shorter than %f\n", cnt, d);
   return 0;
 }
```

# Pontos próximos, bis

**Exercício 6.** *Execute* `prog3.8` *e* `prog3.20_alt` *para comparar seus tempos de execução.*