UEM/CTC – Departamento de Informática
Curso: Ciência da Computação
Professor: Flávio Rogério Uber

Estrutura de Dados
(6884/3 e 4)

Alocação

Estática
- int a;
- float b;

Dinâmica
- malloc
- char *p1;
- p1 = malloc(1);

- typedef struct {
  int dia,mes,ano;
  } data;
- data *d;
- d = (data *)malloc(sizeof(data));
- d->dia=1;
- d->mes=5;
- d->ano=2019;

- free(d);

Tipos Abstrados de Dados (TAD)

- Coleção bem definida de dados a serem armazenados e um grupo de operadores que podem ser aplicados para a manipulação desses dados
- Características:
  - Os operadores do TAD implementam regras bem definidas para a manipulação dos valores armazenados
  - Os valores armazenados devem ser manipulados exclusivamente pelos operadores do TAD

Tipos Abstratos de Dados (TAD)

Mundo Real	Coleção Bem Definida de Dados	Grupo de Operadores
 pessoa	· a idade da pessoa	· nasce (idade recebe o valor zero) · aniversário (idade aumenta em 1)
 fila de espera	· nome de cada pessoa e sua posição na fila	· sai da fila (o primeiro) · entra na fila (no fim)
 cadastro de funcionários	· o nome, cargo e o salário de cada funcionário	· entra no cadastro · sai do cadastro · altera o cargo · altera o salário
 Pilha de Cartas	· informações que identificam a carta (nipe, valor), e sua posição na pilha de cartas	· põe uma carta na pilha (no topo da pilha) · retira uma carta da pilha (a do topo)

Fonte: http://www2.dc.ufscar.br/~bsi/materiais/ed/u2.html

Tipos Abstratos de Dados (TAD)

Usuário de TAD – Acessa apenas as operações através de uma interface

Por não conhecer os detalhes da implementação é que se diz “ABSTRATO”

Em linguagens OO a implementação é feita através de classes

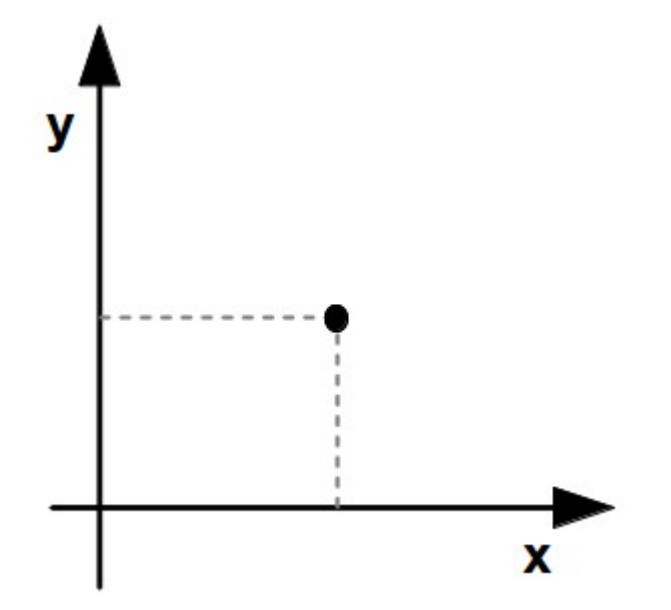
Em linguagens estruturadas é feita através da definição de tipos (em C – **typedef**)

Tipos Abstratos de Dados (TAD)

- O TAD é colocado em uma unidade sintática separada
- Qualquer programa pode usar esta unidade
- Alterações no TAD não afetam os programas que utilizam a unidade

Tipos Abstratos de Dados (TAD)

- Representação de um ponto usando TAD



- Possíveis Operações
  - Cria: cria(x,y)
  - Libera: libera(ponto P)
  - Acessa: acessa(ponto P)
  - Atribui: atribui(ponto P, x, y)
  - Calcular a distância entre dois pontos: distancia (ponto P1, ponto P2)

Tipos Abstratos de Dados (TAD)

- Criaremos um arquivo ponto.h para especificar a interface para o “ponto” em si e suas operações.
- No arquivo que fará uso do TAD usaremos a instrução #include “ponto.h”
  - No include usamos aspas para bibliotecas criadas pelo usuário e < > para bibliotecas nativas da linguagem
- Criaremos um arquivo ponto.c onde faremos a implementação da interface criada

Tipos Abstratos de Dados (TAD)

Arquivo ponto.h (interface)

```
#ifndef PONTO
#define PONTO

typedef struct ponto Ponto;

//FUNCOES

Ponto *cria (float x, float y);

void libera (Ponto* p);

void acessa (Ponto* p, float* x, float* y);

void atribui (Ponto* p, float x, float y);

float distancia (Ponto* p1, Ponto* p2);



#endif // PONTO_H_INCLUDED
```

Tipos Abstratos de Dados (TAD)

Arquivo ponto.c (implementação do TAD – definido pelo ponto.h)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ponto.h"

struct ponto{
    float x;
    float y;
};

Ponto *cria (float x, float y)
{
    Ponto *p = (Ponto*) malloc(sizeof(Ponto));

    if (p == NULL)
    {
        printf("Memória Insuficiente!\n");
        exit(1);
    }

    p->x = x;
    p->y = y;

    return p;
}
```

```
void acessa (Ponto *p, float *x, float *y)
{
    *x = p->x;
    *y = p->y;
}

void libera (Ponto *p)
{
   free(p);
}

void atribui (Ponto *p, float x, float y)
{
   p->x = x;
   p->y = y;
}


float distancia (Ponto* p1, Ponto* p2)
{
   float dx = p2->x - p1->x;
   float dy = p2->y - p1->y;
   return sqrt(dx*dx + dy*dy);
}
```

Tipos Abstratos de Dados (TAD)

Arquivo aplic.c (aplicação que faz uso do TAD)

```
# include <stdio.h>
# include "ponto.h"

int main ()
{
    float x,y;

    Ponto *a = cria(2.0, 1.0);

    acessa(a, &x, &y);
    printf("Ponto p: %f ** %f\n", x, y);

    atribui(a,3.2,4.1);
    acessa(a,&x, &y);
    printf("Ponto p: %f ** %f\n", x, y);



    return 0;

}
```

Tipos Abstratos de Dados (TAD)

- Em linha de comando usamos as instruções:
  gcc -o aplic.o -c aplic.c – *compila a aplicação criada mas sem gerar o executável. Apenas gera o arquivo objeto.*
  gcc -o ponto.o -c ponto.c – *compila a implementação da biblioteca gerando também um arquivo objeto.*
  gcc aplic.o ponto.o -o aplic <-lm> – *gera um executável (aplic) com a junção dos dois arquivos objetos.*

A Vantagem deste formato é a possibilidade de não precisar compilar a biblioteca a cada alteração da aplicação. Tendo o seu arquivo objeto, basta compilar com a aplicação em futuras implementações.

Tipos Abstratos de Dados

- Vantagens:
  - É possível programar sem pensar em detalhes de implementação
  - É mais seguro: a aplicação não altera o TAD
  - Maior independência da aplicação: alterar o TAD não causa problema na aplicação (desde que mantida a interface)
  - Potencial de reutilização maior

Tempo de Execução

Soma dos elementos de um vetor com **n** elementos:

```
#include<stdio.h>


int main( ) {
int i,vet[5],soma=0;

// preenche o vetor
for (i=0; i<5; i++){
    printf("Digite o %io. elemento:",i+1);
    Scanf(“%d”, &vet[i]);
}


for(i=0; i<5; i++){
    soma = soma + vet[i];
}

printf("soma dos elementos: %i", soma);
}
```

--→ 1 unidade de tempo
--→ 1 unidade de tempo } 5 vezes (tam vetor)

--→ 1 unidade de tempo } 5 vezes (tam vetor)

--→ 1 unidade de tempo

Tempo total: 2n + n + 1 = 3n + 1 → O(n)

Tempo de Execução

```
#include<stdio.h>


int main( ) {
int i,j,mat[5][5],soma=0;

// preenche o vetor
for (i=0; i<5; i++){
    for (j=0; j<5; j++){
        printf("Digite o elemento [%i,%i]:",i+1,j+1);
        Scanf(“%d”,&mat[i][j]);
    }
}


for(i=0; i<5; i++){
    soma = soma + mat[0][i];
}

printf("soma dos elementos da primeira linha: %i", soma);
}
```

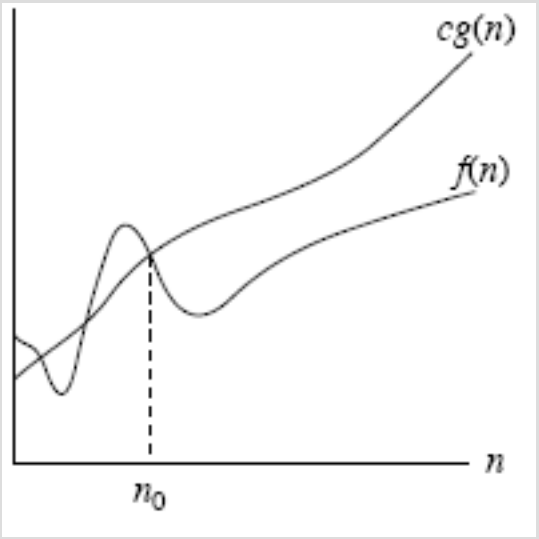
--→ 1 un de tempo
--→ 1 un de tempo 5x5 vezes (tam vetor^2)

--→ 1 un de tempo 5 vezes (tam vetor)

--→ 1 un de tempo

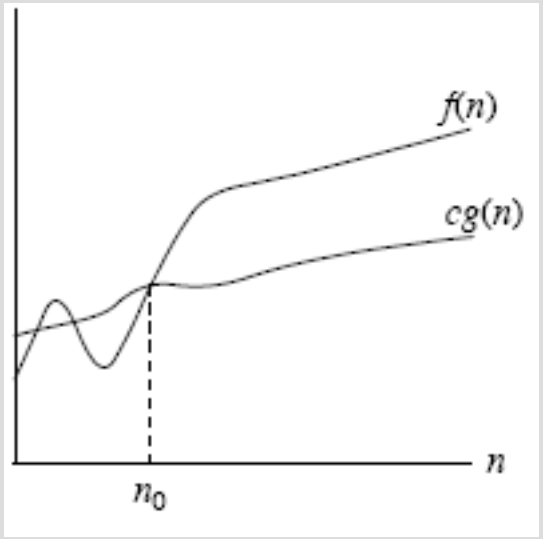
Tempo total: $n^2 + n + 1$ → $O(n^2)$

Notação Assintótica



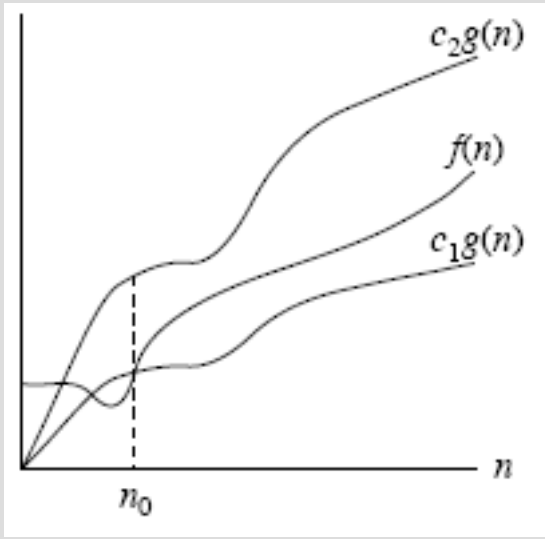
Notação O

f(n) <= c. g(n)



Notação Ω

f(n) >= c.g(n)



Notação Θ

c1.g(n) <= f(n) <= c2.g(n)

Tempo de Execução

Considere 5 algoritmos com as complexidades de tempo.
Suponhamos que uma operação leve 1 ms.

n	$f_1(n) = n$	$f_2(n) = n \log n$	$f_3(n) = n^2$	$f_4(n) = n^3$	$f_5(n) = 2^n$
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10^{137} séculos