

# CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

4 de setembro de 2020

## Resumo

TAD conjunto: uma implementação possível

- Apresentar uma possível implementação para o TAD conjunto
- Discutir eficiência dos algoritmos

# Implementações do TAD conjunto

- Vimos como usar o TAD conjunto sem conhecer a implementação
- Agora vamos conhecer as implementações!
- Existem muitas possibilidades, veremos uma
  - Uma outra é apresentada no livro, estudem e vejam as diferenças!
  - Tentem implementar a outra sem olhar no livro
- O importante é manter os protótipos e garantir que a semântica dos procedimentos e funções seja conforme foi especificado

# Implementação 1

- O TAD conjunto usa um vetor cujos conteúdos estão ordenados em ordem crescente a partir da posição 1 do vetor
- usaremos o `tipo_vetor` definido na aula de registros
- Reservamos as posições 0 e `MAX+1` para as sentinelas
- O campo `proximo` servirá para as funções `iniciar_proximo` e `iniciar_proximo`

```
1 const max = 200;
2
3 type
4   conjunto = record
5     tam: longint;
6     proximo: longint;
7     v: array [0..MAX+1] of longint;
8   end;
```

- Implementaremos algumas funções e procedimentos
- Basicamente as que são interessantes ou didáticas
- No livro todas elas estão implementadas
- Vários algoritmos já foram estudados na parte de vetores e não serão apresentadas aqui

# Inicializar conjunto

- Todo tipo abstrato de dados deve ser inicializado para garantir consistência e correto funcionamento das funções e procedimentos
- Esta procedure tem custo constante

```
1 procedure inicializar_conjunto (var c: conjunto);  
2 // Cria um conjunto vazio. Deve ser chamado antes de qualquer  
   operacao no conjunto.  
3 // Custo: constante.  
4 begin  
5     c.tam:= 0;  
6 end;
```

- Esta função tem custo constante

```
1 function conjunto_vazio (c: conjunto): boolean;  
2 // Retorna true se o conjunto c eh vazio e false caso contrario.  
3 // Custo: constante.  
4 begin  
5     conjunto_vazio:= c.tam = 0;  
6 end;
```

# Retornar a cardinalidade

- Esta função tem custo constante

```
1 function cardinalidade (c: conjunto): longint;  
2 // Retorna a cardinalidade do conjunto c.  
3 // Custo: constante.  
4 begin  
5     cardinalidade:= c.tam;  
6 end;
```

- Implementa a busca binária, pois o vetor está ordenado
- Um TAD pode ter funções e procedimentos que não são públicos, ou seja, são internos e de interesse exclusivo do programador
- Vejam no livro para maiores detalhes
- Esta função tem custo  $\log(n)$

- Usam inserção, remoção e cópia conforme mostrado na seção de vetores
- Como o vetor está ordenado, todas tem custo linear

# Iniciar e incrementar próximo

- Retorna *false* quando o contador chegou no fim do vetor e devolve o elemento usando-se um parâmetro por referência
- Os algoritmos que a utilizam devem saber quando o contador chegou ao final e devem chamar o iniciador do contador quando precisarem percorrer o conjunto novamente
- Estas função tem custo constante

```
1 procedure iniciar_proximo (var c: conjunto);
2 // Inicializa o contador que sera usado na funcao incrementar_proximo.
3 // Custo: constante.
4 begin
5     c.proximo:= 1;
6 end;
```

```
1 function incrementar_proximo (var c: conjunto; var x: longint): boolean;
2 // Incrementa o contador e retorna x por referencia. Retorna false se acabou conjunto
3 begin
4     if c.proximo <= c.tam then
5     begin
6         x:= c.v[c.proximo];
7         c.proximo:= c.proximo + 1;
8         incrementar_proximo:= true;
9     end
10    else
11        incrementar_proximo:= false;
12 end;
```

# Retirar um elemento a cardinalidade

- Útil para se manipular conjuntos, ela permite a retirada de um elemento qualquer.
- Optamos por retirar o último elemento, mantendo o conjunto ordenado de modo simples.
- Pensada para ser usada em combinação com o teste de conjunto vazio
- Esta função tem custo constante

```
1 function retirar_um_elemento (var c: conjunto): longint;  
2 // Escolhe um elemento qualquer do conjunto para ser removido e o  
   remove  
3 // Custo: constante, pois optamos por devolver o ultimo elemento.  
4 begin  
5     retirar_um_elemento:= c.v[c.tam];  
6     c.tam:= c.tam - 1;  
7 end;
```

# Teste de igualdade

- Esta função tem custo linear

```
1 function sao_iguais (c1, c2: conjunto): boolean;  
2 // Retorna true se o conjunto c1 = c2 e false caso contrario.  
3 // Custo: dada a ordenacao, linear.  
4 var i: longint;  
5 begin  
6     if c1.tam <> c2.tam then  
7         sao_iguais:= false  
8     else  
9         begin  
10            i:= 1;  
11            while (i <= c1.tam) and (c1.v[i] = c2.v[i]) do  
12                i:= i + 1;  
13            if i <= c1.tam then  
14                sao_iguais:= false  
15            else  
16                sao_iguais:= true;  
17        end;  
18 end;
```

- Esta função tem custo linear e é muito eficiente, usa o *merge*

```
1 function uniao (c1, c2: conjunto): conjunto;
2 var i,j,k,l: longint; uni: conjunto;
3 begin
4   inicializar_conjunto (uni);
5   i:= 1; j:= 1; k:= 0;
6   while (i <= c1.tam) and (j <= c2.tam) do
7     begin
8       k:= k + 1;
9       if c1.v[i] < c2.v[j] then // avanca apontador do primeiro vetor
10        begin
11          uni.v[k]:= c1.v[i]; i:= i + 1;
12        end
13       else if c1.v[i] > c2.v[j] then // avanca apontador do segundo vetor
14        begin
15          uni.v[k]:= c2.v[j]; j:= j + 1;
16        end
17       else // descarta um dos repetidos e avanca os dois apontadores
18        begin
19          uni.v[k]:= c1.v[i];
20          i:= i + 1; j:= j + 1;
21        end;
22     end; (* while *)
23     copia_restante_do_vetor (uni, c1, i, k);
24     copia_restante_do_vetor (uni, c2, j, k);
25     uni.tam:= k;
26     uniao:= uni;
27 end;
```

- Usa dois apontadores para os vetores de entrada, como os vetores estão ordenados, comparamos os elementos apontados por  $i$  e  $j$
- O menor deles é copiado no vetor união e seu respectivo apontador é incrementado
- Se os elementos são iguais, apenas um deles é copiado, mas os dois apontadores são incrementados

- Esta função tem custo linear e é muito eficiente

```
1 function intersecao (c1, c2: conjunto): conjunto;
2 // Obtem a intersecao dos conjuntos c1 e c2.
3 // Custo: como estao ordenador, proporcional ao tamanho do vetor c1.
4 // Possivel modificacao: o custo pode ser proporcional ao tamanho do menor conjunto.
5 var i,j,k: longint; intersec: conjunto;
6 begin
7   inicializar_conjunto (intersec);
8   i:= 1;
9   j:= 1;
10  k:= 0;
11  while (i <= c1.tam) and (j <= c2.tam) do
12    if c1.v[i] < c2.v[j] then
13      i:= i + 1
14    else
15      if c1.v[i] > c2.v[j] then
16        j:= j + 1
17      else // elemento esta nos dois conjuntos
18        begin
19          k:= k + 1;
20          intersec.v[k]:= c1.v[i];
21          i:= i + 1;
22          j:= j + 1;
23        end;
24    intersec.tam:= k;
25    intersecao:= intersec;
26 end;
```

- Inserimos uma das cópias no vetor intersecção quando eles são iguais
- Neste caso ambos os apontadores são incrementados
- Quando um é menor do que o outro significa que ocorrem em um conjunto somente e apenas um dos apontadores é incrementado.

- Esta função tem custo linear e é muito eficiente

```
1 function diferenca (c1, c2: conjunto): conjunto;
2 // Obtem a diferenca dos conjuntos c1 e c2 (c1 - c2).
3 // Custo: linear por causa da ordenacao.
4 var i,j,k: longint; dif: conjunto;
5 begin
6   inicializar_conjunto (dif);
7   i:= 1; j:= 1; k:= 0;
8   while (i <= c1.tam) and (j <= c2.tam) do
9     if c1.v[i] < c2.v[j] then
10      begin
11        k:= k + 1;
12        dif.v[k]:= c1.v[i];
13        i:= i + 1;
14      end
15     else if c1.v[i] > c2.v[j] then
16       j:= j + 1
17     else // sao iguais
18      begin
19        i:= i + 1;
20        j:= j + 1;
21      end;
22     // se ainda tem elementos em c1, copia
23     for j:= i to c1.tam do
24      begin
25        k:= k + 1;
26        dif.v[k]:= c1.v[j];
27      end;
28     dif.tam:= k;
29   diferenca:= dif;
30 end;
```

- Usa dois apontadores para os vetores de entrada
- O primeiro vetor é percorrido na busca de elementos que sejam menores do que os do segundo
- São os elementos que ocorrem apenas no primeiro e devem ser copiados no vetor diferença
- Os outros são descartados
- Caso o segundo vetor termine antes os elementos do primeiro vetor são copiados para o destino.

- Esta função tem custo linear e é muito eficiente

```
1 function contido (c1, c2: conjunto): boolean;  
2 // Retorna true se o conjunto c1 esta contido no conjunto c2 e false caso contrario.  
3 // Custo: proporcional ao tamanho do conjunto c1.  
4 var i, j: longint; ok: boolean;  
5 begin  
6     if c1.tam > c2.tam then  
7         contido:= false  
8     else  
9         begin  
10            contido:= false; ok:= true; i:= 1; j:= 1;  
11            while (i <= c1.tam) and (j <= c2.tam ) and ok do  
12                if c1.v[i] < c2.v[j] then  
13                    ok:= false  
14                else if c1.v[i] > c2.v[j] then  
15                    j:= j + 1  
16                else  
17                    begin  
18                        i:= i + 1; j:= j + 1;  
19                    end;  
20                if ok and (i > c1.tam) then  
21                    contido:= true  
22            end;  
23 end;
```

- O objetivo é encontrar todos os elementos do primeiro no segundo
- Uma falha ocorre quando um elemento do primeiro vetor é menor do que o do segundo, significando que ele não está presente nesse
- Outra falha é quando chegamos no final do segundo vetor antes de chegar no final do primeiro.

- 1 Sem olhar a solução do livro:
  - Implemente todos estes algoritmos usando um vetor no qual os conteúdos não estão ordenados
  - Calcule o custo deles no pior caso
  - Faça uma tabela comparativa com os custos das duas implementações
- 2 Faça a implementação do TAD Pilha e compare com a solução do livro, seção 12.2

- este material está no livro no capítulo 12, seção 12.1

- Slides feitos em  $\text{\LaTeX}$  usando beamer
- Licença

*Creative Commons* Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>