Arrays

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Arrays

Definition

An array is a collection of items stored at contiguous memory locations.

The type can be anything: integer, real, character, string or a constructed type.

In this last case, we say that we are dealing with an array of records.

Access to an element : indexing operation
for any i in [1, n] : T(i) gives the value of the element having i as index in the array T.

Sorted array:
- set with an order relation.
- for all i in [1, n-1] : T(i) <= T(i+1)

Arrays

Static array Vs Dynamic array

Static array	**Dynamic array**
Space allocation is done at the beginning of a treatment.	Space allocation is done during execution
In technical terms, we say that the space is known at compile time.	The release of the space is then possible during the execution

Arrays

Abstract Machine

ELEMENT (T [i, j, ...]) :
Access to element T[i, j, ...] of array T.

ASS_ELEMENT (T [i, j, ...], Val) :
Assign value Val to element T[i, j, ...].

ALLOC_ARRAY (T) :
Array allocation. The array address is in T.

FREE_ARRAY(T) :
Free the array of address T.

Arrays

Algorithms

4 categories

Traversal : Access by value, Access by position

Update : Insert, Remove

Algorithms on several arrays : Intersection, Union, Merge, Split,...

Sorting arrays

Arrays

Algorithm : Traversal : linear search

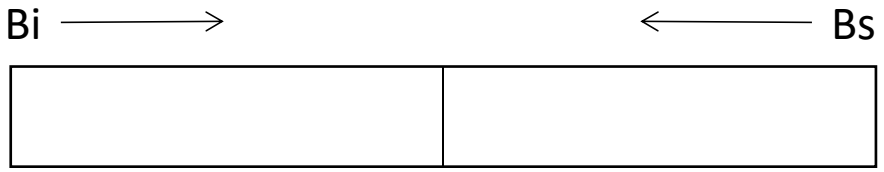
Not sorted array

```
I := 1 ;
Found := FALSE ;
WHILE ( I <= N ) AND NOT Found
   IF ELEMENT ( V [ I ] ) = Val
      Found := TRUE
   ELSE
      I := I + 1
   ENDIF
ENDWHILE ;
```

Sorted array

```
I := 1 ;
Found := FALSE ;
Stop := FALSE ;
WHILE ( I <= N ) AND NOT Found AND NOT Stop
   IF ELEMENT ( V [ I ] ) = Val
      Found := TRUE
   ELSE
      IF ELEMENT ( V [ I ] ) > Val
         Stop := TRUE
      ELSE
         I := I + 1
      ENDIF
   ENDIF
ENDWHILE ;
```

Arrays



Algorithm : Traversal : Binary search

```
Bi := 1 ;
Bs := N ;
Found := FALSE ;
WHILE ( Bi <= Bs ) AND NOT Found
   Middle := ( Bi + Bs ) / 2 ;
   IF ELEMENT ( V [ Middle ] ) = Val
      Found := TRUE
   ELSE
      IF Val < ELEMENT ( V [ Middle ] )
         Bs := Middle - 1
      ELSE
         Bi := Middle + 1
      ENDIF
   ENDIF
ENDWHILE ;
```

Maximal number of comparisons = Log2(N)

Arrays

Algorithm : Update

Insert an element after or before a given value

Insert an element at a specific position

Insert an element into a sorted array

Deleting an array element (sorted or not)

Logical delete in a sorted or not sorted array (using a bit Isvalid)

Arrays

Maximal number of shifts = N

Algorithm : Inserting a given value into a sorted array T[1..M] holding N elements

```
IF N < M
  {Searching for a position}
   I := 1 ;
   Found := FALSE ;
   WH ( I <= N ) AND NOT Found
      IF ELEMENT ( V [ I ] ) >= Val
         Found := TRUE
      ELSE
         I := I + 1
      ENDIF
   EWH ;
  { Insert    }
   IF NOT Found
      ASS_ELEMENT ( V [ I ] , Val )
   ELSE
      FOR J := N + 1 , I , - 1
         ASS_ELEMENT ( V [ J ] , ELEMENT ( V [ J - 1 ] ) )
      EFOR ;
      ASS_ELEMENT ( V [ I ] , Val )
   ENDIF ;
   N := N + 1
ELSE  {N<M}
   WRITE ( 'Overflow' )
ENDIF
```

Arrays

V1 : Array[1..N1] of Integer
V2 : Array[1..N2] of Integer
V3 : ARRAY[1..N1+N2] of Integer

Algorithm : Merging 2 sorted arrays V1[1..N1] and V2[1..N2]

```
I := 1 ;      J := 1 ;      K := 0 ;
WHILE ( I <= N1 ) AND ( J <= N2 )
   K := K + 1 ;
   IF ELEMENT ( V1 [ I ] ) < ELEMENT ( V2 [ J ] )
      ASS_ELEMENT ( V3 [ K ] , ELEMENT ( V1 [ I ] ) ) ;
      I := I + 1
   ELSE
      ASS_ELEMENT ( V3 [ K ] , ELEMENT ( V2 [ J ] ) ) ;
      J := J + 1
   ENDIF ;
ENDWHILE ;
```

```
WHILE I <= N1
   K := K + 1 ;
   ASS_ELEMENT ( V3 [ K ] , ELEMENT ( V1 [ I ] ) ) ;
   I := I + 1
ENDWHILE ;
WHILE J <= N2
   K := K + 1 ;
   ASS_ELEMENT ( V3 [ K ] , ELEMENT ( V2 [ J ] ) ) ;
   J := J + 1
ENDWHILE
```

Maximal number of assignments = N1 + N2

Arrays

Algorithm : Bubble sort

```
FOR I := 1 , N - 1
   FOR J := N , I + 1 , - 1
      IF ELEMENT ( V [ J - 1 ] ) > ELEMENT ( V [ J ] )
         Temp := ELEMENT ( V [ J - 1 ] ) ;
         ASS_ELEMENT ( V [ J - 1 ] , ELEMENT ( V [ J ] ) ) ;
         ASS_ELEMENT ( V [ J ] , Temp )
      ENDIF
   ENDFOR
ENDFOR
```

Maximal number of permutations = N (N-1)/2

Arrays

Algorithm : Merge sort

```
Step := 1 ;
WH Step <= N
   FOR I := 1 , N - Step , 2 * Step
      CALL Merge ( I , Step , I + Step , MIN ( Step , N - ( I + Step ) + 1 ) , V2 , N2 ) ;
      CALL Copy ( V2 , N2 , I , I + Step + MIN ( Step , N - ( I + Step ) + 1 ) - 1 ) ;
   EFOR ;
   WRITE ( 'Step=' , Step ) ;
   WRITE ( 'V1=' , V1 ) ;
   Step := 2 * Step
EWH
```

V1 V2 V3 V4 V5 V6 V7 V8 V9

[V1] [V2] [V3] [V4] [V5] [V6] [V7] [V8] [V9]

[V1 V2] [V3 V4] [V5 V6] [V7 V8] [V9]

[V1 V2 V3 V4] [V5 V6 V7 V8] [V9]

[V1 V2 V3 V4 V5 V6 V7 V8] [V9]

[V1 V2 V3 V4 V5 V6 V7 V8 V9]

Maximal number of assignments = N Log2(N)

Arrays

Memory representation : n-dimensional array

A typical declaration : $A(a_1:b_1; a_2:b_2; .....a_n:b_n)$

Order of arrangement of sub-arrays:

$A(a_1, *, *, ..., *)$, $A(a_1+1, *, *, ..., *)$, $A(a_1+2, *, *, ..., *)$,
....., $A(i, *, *, ..., *)$, ... $A(b_1, *, *, ..., *)$

Within each sub-array A(i, * , * , ..., *), the following order of sub-sub-array is considered:

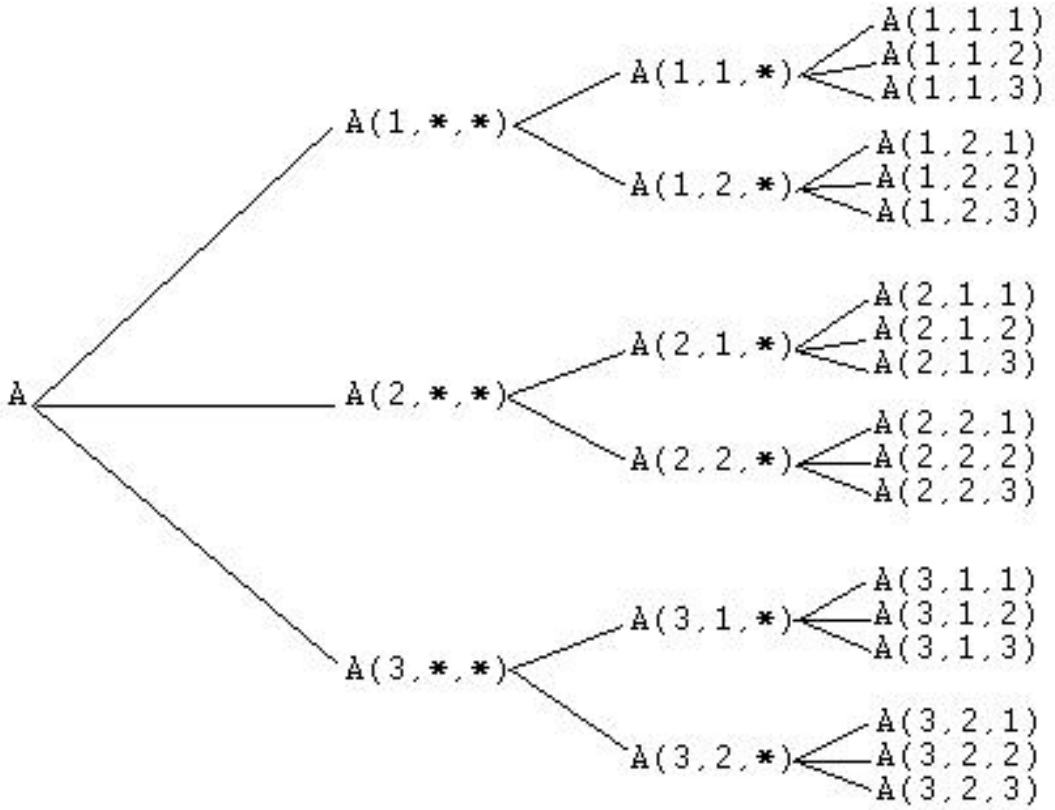
A(i, a2, *, *, ..., *), A(i, a2+1, *, *, ..., *), A(i, a2+2, *, *, ..., *),
....., A(i, b2, *, *, ..., *)

Etc.

Arrays

Memory representation : n-dimensional array

A(1..3, 1..2, 1..3)



Arrays

Memory representation : n-dimensional array

Address of item A(i_1, i_2 .., i_n) ?

Let $d_i = b_i - a_i + 1$

Address of A(i_1, *, *, ...) : $AD_1 = Base + (i_1-a_1)\, d_2d_3...d_n$

Address of A(i_1, i_2, *, ...) : $AD_2 = AD_1 + (i_2-a_2)d_3d_4...d_n$

Address of A(i_1, i_2, ...i_n) :

$AD_n = Base + (i_1-a_1)d_2d_3..d_n + (i_2-a_2)d_3d_4...d_n + ...+(i_{n-1}-a_{n-1})d_n + (i_n-a_n)$

Arrays

Memory representation : n-dimensional array

Address of $A(i_1, i_2, ...i_n)$:

$AD_n = Base + (i_1-a_1)d_2d_3..d_n + (i_2-a_2)d_3d_4...d_n + ...+(i_{n-1}-a_{n-1})d_n + (i_n-a_n)$

Constant part : $Base -(a_1 \prod_{i=2,n} d_i + a_2 \prod_{i=3,n} d_i + ... + a_{n-1}d_n + a_n)$

Variable part : $i_1 \prod_{i=2,n} d_i + i_2 \prod_{i=3,n} d_i + ... + i_{n-1}d_n + i_n$

If a1 = a2 = ... = an = 0
the address of A(i1, i2, ...in) is

$i1d2d3..dn + i2d3d4...dn + ...+in\text{-}1dn + in = \sum_{j=1,n} (i_j . \prod_{i=j+1,n} d_i)$

Arrays

Implementation (Dynamic / C)

```
#include <stdio.h>
#include <stdlib.h>

/** -Implementation- **\: ARRAY OF INTEGERS**/

/** Arrays **/

typedef int Typeelem_V10i ;
typedef Typeelem_V10i * Typevect_V10i ;

Typeelem_V10i Element_V10i ( Typevect_V10i V , int I1  )
 {
  return  *(V +  (I1-1)  ) ;
 }

void Ass_element_V10i ( Typevect_V10i V  , int I1 ,
Typeelem_V10i Val )
 {
  *(V +  (I1-1)  ) = Val ;
 }
```

```
/** Variables of main program **/
Typevect_V10i V;

int main(int argc, char *argv[])
  {
   V = malloc(10 * sizeof(int));
   system("PAUSE");
   return 0;
}
```

Arrays

Implementation (Dynamic / PASCAL)

```
PROGRAM My_program;

 { -Implementation- : ARRAY OF INTEGERS}

 { Arrays }
 TYPE
   Typeelem_V10I = INTEGER;
   Typetab_V10I = ARRAY[1..10] OF Typeelem_V10I;
   Typevect_V10I = ^ Typetab_V10I;

 FUNCTION Element_V10I ( V:Typevect_V10I; I1  : INTEGER ) : Typeelem_V10I ;
   BEGIN
     Element_V10I :=  V^[I1 ];
   END;

 PROCEDURE Ass_element_V10I ( V :Typevect_V10I; I1  :INTEGER; Val : Typeelem_V10I );
   BEGIN
      V^[I1 ] := Val;
   END;
```

```
 { Declaration part of variables }
 VAR
   V : Typevect_V10I;

 { Body of main program }
 BEGIN
   NEW(V);
READLN;
 END.
```