ECSE-322

14 January 2008

Lecture 5

Arrays and Vectors

Tutorial Today 16:35

Problem Set today

Abstract Data Type

Data Structures

(Byte - addressable machine

Array → $\boxed{A\ B\ C\ D}$

Arrays and Vectors

- Ways of arranging collections of data of the same type
  - e.g. integers, real numbers, etc..
  - Each element is unique and located by a location (its *key*)
    - a_{ij}, b_k,..
  - The collection of elements is an *array*
  - If one index is used to locate an item (e.g. b_k), it is a *linear array* or *vector*

Arrays

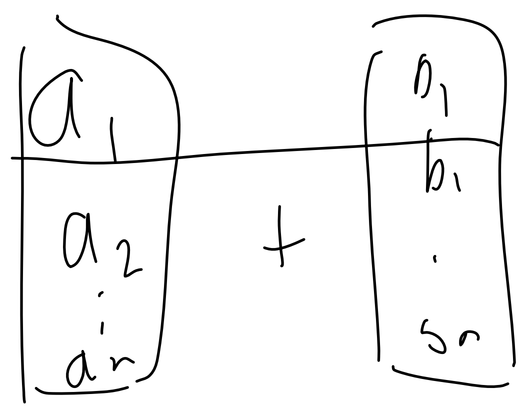
- All elements are of the same type
  - *Why?*

Arrays

- All elements are of the same type
  - *Why?*
  - *Because of the operations..*
    - Comparison .. $a_i := a_k$
    - Assignment (Store) $a_i := z$
    - Retrieval $z := a_i$

Arrays

- A linear array is a simple storage structure and is very common..
  - E.g. file storage on a magnetic tape
    representation of a vector...
  - Useful for repetitive operations in which several arrays are accessed one component at a time..



Arrays

- Add two vectors:

repeat n times:

$c_i := a_i + b_i;$

Pascal program

```
Program Addvectr (input, output);
{Adds two N-long vectors}
const N = 5;
var   A, B, C : array [1..N] of integer;
      I: integer;
procedure readvec;
begin ...... end;
procedure writvec;
begin ...... end;
begin
      readvec;
      for I:=1 to N do begin
            C[I] := A[I] + B[I];
            end;
      writvec;
end.
```

Complexity

- Important questions on any algorithm:
  - 1. How much space does it need?
  - 2. How much time is taken?
- For the sum example
  - Space = 3N integer locations

$O(N^2)$

```
for I:=1 to N do begin
        C[I] := A[I] + B[I];
        end;
```

Algorithm needs O(N) space
i.e. if N is doubled the amount of space doubles..

Complexity

- Time?
  - For the example, each storage space is visited exactly once and one addition is performed per loop:

```
for I:=1 to N do begin
        C[I] := A[I] + B[I];
        end;
```

One operation per loop

Algorithm needs O(N) time
i.e. if the number of items doubles, so does the time taken

Vectors

- Consider a polynomial:

$$x^0$$

$$p_N(x) \quad a_0 + a_1 x + a_2 x^2 + \ldots + a_N x^N$$

How can we store this?

Use an array:

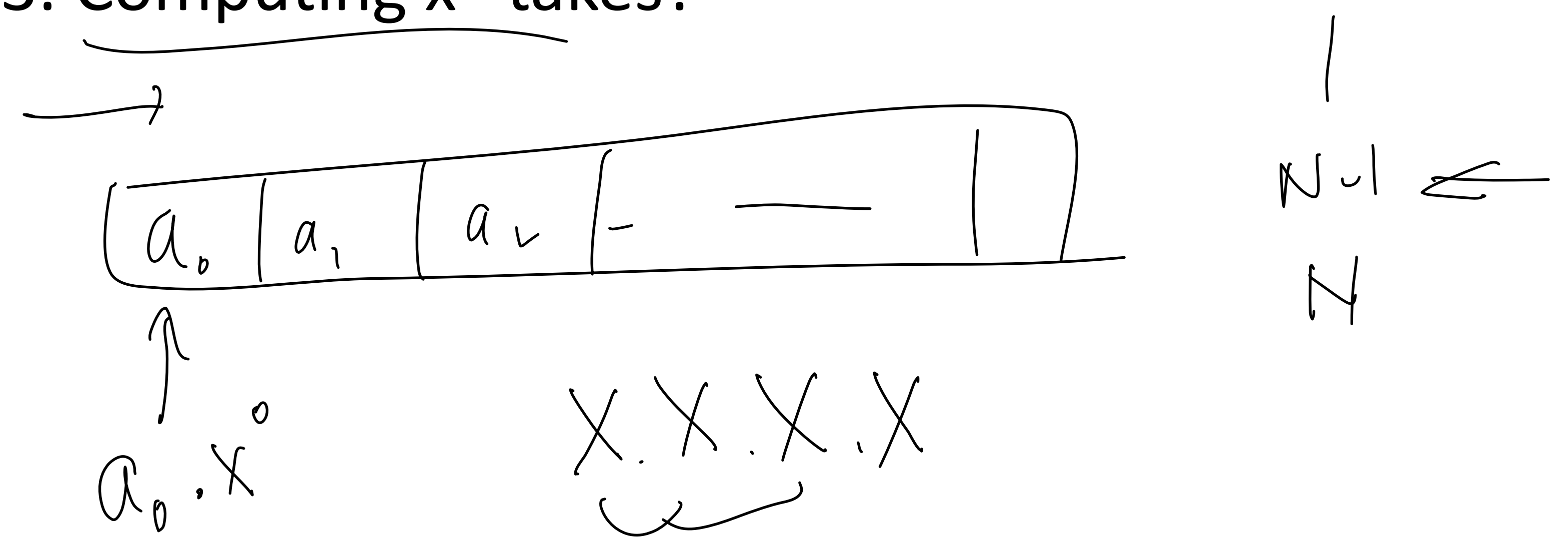
a_0	a_1	a_2	a_3	…	…	a_{N-1}	a_N

Thus the array P contains the coefficients of each power of x

Array length? = N+1 elements

Vectors

- What does it cost to compute the polynomial?
  - 1. Linear traversal (N+1 operations)
  - 2. Each element multiplied by x^N (N = position in array) (N multiplies)
  - 3. Computing x^N takes?



Vectors

- X^k:
  - k-1 multiplications plus the one by a_k
  - Thus the number of multiplications is

    m = 0+1+2+3+…+N

  The total is?

$$a_0 + a_1 x + a_2 x^2 \cdots a_n x^n$$

$$\frac{N(N+1)}{2} \qquad \frac{N^2}{2} + \frac{N}{2}$$

Vectors

- X^k:
  - Total multiplications is:

    $N(N+1)/2$

  - *The work is proportional to* N^2
  - The process runs in $O(N^2)$ time

- Can we do better?

Vectors

- Rewrite the polynomial:

$$p_N(x) \quad a_0 + x(a_1 + a_2 x + \ldots + a_N x^{N-1})$$

This is:

$$p_N(x) = a_0 + x\, p_{N-1}(x)$$ ← A recursive formula!

Note

$$p_0(x) = a_N$$

In Pascal:

```
program Polyvect(input, output);
        {Evaluates polynomial recursively, 0 <x < 1. }
        const       N=4;
        var         a: array [0..N] of real;
                    x, y : real;
                    j, k: integer;
        procedure readvect;
                    begin .... end;
        procedure writvect;
                    begin .... end;

function poly(x: real; k: integer) : real;
{ Recursively computes poly(x,k) }
begin
if k = 0 then begin
            poly := a[N];
            end
else begin
            poly := a[N - k] + x*poly(x, k - 1);
            end;
end;

begin       {Main program starts}
readvect;
x := 0.1;
y := poly(x, N);
writvect;
end.
```

Complexity

- Cost

  O(N) work!

- What is the memory cost?
- Can we save on memory?

```
function poly(x: real; k: integer) : real;
{ Recursively computes poly(x,k) }
begin
if k = 0 then begin
                poly := a[N];
                end
else begin
                poly := a[N - k] + x*poly(x, k - 1);
                end;
end;
```

One add and multiply per value of k

Restructure - again!

Start with

$$p_N(x) = a_0 + x(a_1 + a_2x + \ldots + a_Nx^{N-1})$$

And factor again...

$$p_N(x) = a_0 + x(a_1 + x(a_2 + \ldots + a_Nx^{N-2}))$$

And again...

The *nested product* form

$$p_N(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \ldots + a_Nx))\ldots))$$

The Pascal program:

```
Function polyn (x: real) : real;
{nested multiply to compute poly(x,N).}
var    j : integer;
       x : real;
begin
z := a[N];
j :=N-1;
while j >= 0 do begin
       z := a[j] + x * z;
       j :=j - 1;
       end;
polyn := z
end;
```

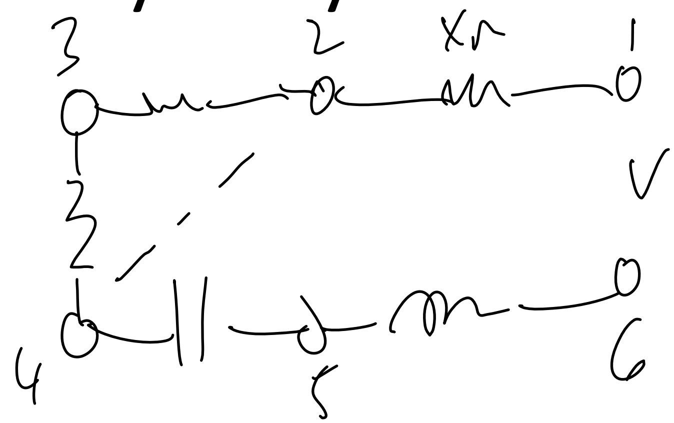
Loop is executed N times

1 multiply and add per loop

The process is O(N) in time and O(N) in space

Sparse Vector Storage

- What if most of the coefficients in the polynomial are zero?
  - The array storage would waste space and most of the calculation time would be multiplying and adding zero…
- In engineering this is very likely to be the case
  - e.g. circuits..



Sparse Vector Storage

0 2 0 . . . 3 . 1

499 1000

Consider

$$p(x)=2x+3x^{499}+x^{1000}$$

In a simple array this would require 1001 entries - 998 would be zero!

Instead, use a more complex structure:

a:	2	1
	3	499
	1	1000

coefficient

exponent (also the index)

Sparse Vector Storage

- In C this would be:

```
typedef struct { double val; /*element value*/
                 int idx;/*element index*/
               }        element;
```

Sparse Vector Storage

a	b
0	0
1	1

- Take 2 polynomials:

$$p1(x) = 13.775x^{28} - 4.006x^{337} + 8.633x^{852}$$

$$p2(x) = 3.576x^{131} - 0.106x^{337} + l.03x^{852} - 5.664x^{945}$$

Storage:

a:

13.775	28
-4.006	337
8.622	852

b:

3.576	131
-0.106	337
1.03	852
-5.664	945

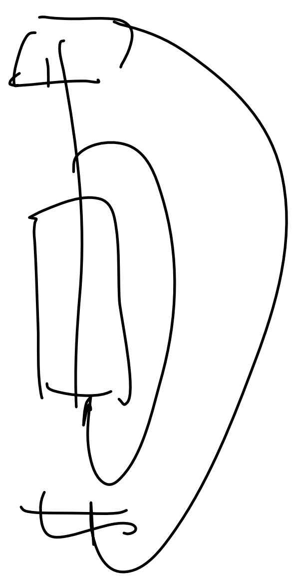
Now add them!

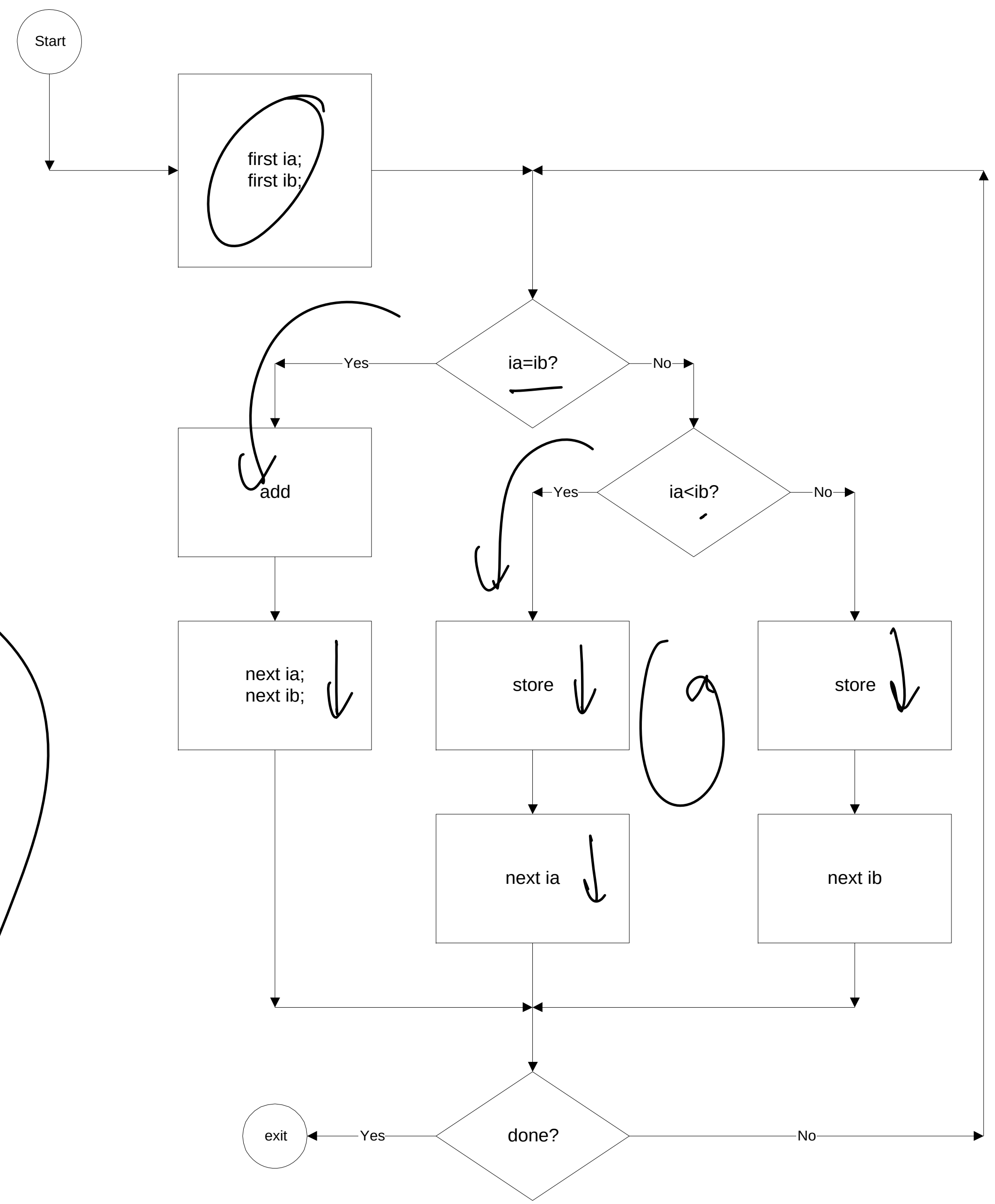
Sparse Vector Storage

- Addition process:

  1. Get the first value in each array
  2. Are the powers of x equal?
  3. If they are perform an add and get the next locations in the arrays
  4. If they are not equal, is the power of x in a less than that in b?
  5. If it is then store the value and get the next location in the array a. Go to step 2.
  6. So b is less than a, store the value and get the next location in array b. Go to step 2.

O(N)





Start
first ia;
first ib;
ia=ib?
Yes
No
add
next ia;
next ib;
ia<ib?
Yes
No
store
next ia
store
next ib
done?
Yes
No
exit

Sparse Vector Storage

- Advantages of this scheme
  - Reduces the storage requirement
- Disadvantages
  - More instructions to execute
- What is the time complexity of this algorithm?

O(N)

Hashing

- There is a need to store data which, for most of the domain, is zero.
- There is a requirement to minimize the space taken by the non-zero elements.
- There is a requirement to minimize the time to find a data item (if it exists) (see the previous algorithm)

Hashing

- With the previous structure how do you answer the question:

  "Does the coefficient of x to the 131 exist?

- We need a method of directly accessing the storage location for the coefficient of 131...