Array

Yu-Tai Ching
Department of Computer Science
National Chiao Tung University

Array in C++

- A set of consecutive memory storing identical data type object,
- int $A[100]$; float $B[30]$; Object $C[200]$;
- Direct access, given index i, $A[i]$ gives the $i+1$st integer stored in array A. Random Access Machine (RAM)
- A, the address where array A begin, an `int` is 4 bytes, we find $A[i]$ at address $A+i$, the value is $*(A+i)$.
- It takes constant time to compute the address, then direct access the memory location.
- We are talking about detail implementation, not the abstract data type.

Array, ADT

- A set of pairs $< index, value >$, a *correspondence* or a *mapping*
- The operations
  - Retrieve the value for a given index,
  - store a value to the array at a given index.

class $GeneralArray$ {
public:
    $GeneralArray$(int j, $RangeListlist$,
        float $initValue$ = defaultValue);
    float $Retrieve(index)$;
    void $Store(index\ \ i$, float x);
};

- Array can be used to implement other data types,
- *Ordered* or *linear list* (store those objects that you can define“linear order”). Ordering is an important information for efficient computation.
- examples are
  - Days of the week, (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday).
  - Value of card deck (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King).
  - Floor of a building (Basement, Lobby, mezzanine, first, second).
  - Or an empty list ().

Ordered List

- Can be written in the form $(a_0, a_1, \ldots, a_{n-1})$
- operations performed on list (ADT definitions needs data objects and operations (methods))
  - Find the length n of the list.
  - Read list from left to right (or from right to left).
  - Retrieve the ith element, $0 \leq i < n$.
  - Insert a new element at the position i, $0 \leq i < n$, causing elements numbered i, $i+1$, . . . , $n-1$ to become numbered $i+1$, $i+2$, . . . , n.
  - Delete the element at position i, $0 \leq i < n$, causing elements numbered $i+1$, . . . , $n-1$ to become numbered i, $i+1$, . . . , $n-2$.

Ordered List- Implementation Consideration

- Use an array (C++ array).
  - Find the length, $\Theta(1)$.
  - read the list, $\Theta(n)$.
  - Retrieve the ith element, $\Theta(1)$.
  - Insert a new element, suppose we know where to insert, first move data around then insert.
  - Delete an element, suppose we know what to delete, delete it and move data around.

Polynominal

- A problem requires ordered list.
- Building ADT for the representation and manipulation of polynominals in a single variable.
- Two such polynominals are $a(x) = 3x^2 + 2x - 4$ and $b(x) = x^8 - 10x^5 - 3x^3 + 1$
- There are 3 terms in $a(x)$, $3x^2$, $2x$, and -4.
- *Coefficients* of the 3 terms are 3, 2, and -4; the *exponents* are 2, 1, 0.
- The *degree* is the largest exponent among the nonzero terms.
- A term can be representated as a pair (coeficient,exponent), such as (3,2) , a method that we have chance only display the nonzero terms.

Polynominal

- Operations applied to two polynominals.
- Sum of two polynominals, product of two polynominals
- $a(x) = \sum a_i x^i$ and $b(x) = \sum b_i x^i$; then
- $a(x) + b(x) = \sum (a_i + b_i) x^i$,
- $a(x)b(x) = \sum (a_i x^i \cdot \sum (b_j x^j))$
- also substraction and division

In C++ class

class $Polynominal$ {
public:

$Polynominal()$;

$Polynominal\ Add(Polynominal\ poly)$;

$Polynominal\ Mult(Polynominal\ poly)$;

$float\ Eval(float\ f)$;

};

What should the private data look like? (detail implementation!)

Polynominal Represnetation

- First approach

  private:
      int $Degree$;
      float $coef[MaxDegree + 1]$;

- $MaxDegree$ represents the largest-degree polynominal, $n \leq MaxDegree$.
- $coef[i]$ stores a_{n-i}.
- A polynominal a, $a.degree = n$, $a.coef[i] = a_{n-i}$, $0 \leq i \leq n$.
- Space $MaxDegree$ is allocated, regarless the number of terms (not a function of input size).

Polynominal Representation

- Second Approach, space needed depend on the degree.

  private:

      int $degree$;

      float $*coef$;

- and we need the constructor

  $Polynominal :: Polynominal$ (int d)

  {

      $degree = d$;

      $coef$ = new float $[degree + 1]$;

  }

Polynomial Representation

- A problem with the second approach
- Zero terms also take space,
- $a(n) = n^{1000} + 1$;
- It take 1001 memory spaces but there are just only two nonzero terms.

Polynominal Representation, The Third Approach

class Polynominal; // forward declaration

class $Term$ {
friend $Polynominal$;
private:
    float $coef$;
    int exp;
};

The private data members of $Polynominal$ are private:
    $Term *termArray$;
    int $capacity$;
    int $terms$;

- Third approach fixes the problem with the first and the second approaches.
- Memory required depends on the problem size (number of nonzero terms).
- Addition, $Polynominal\ a$, b, c; $c = a + b$;
- if ($a.termArray[i].exp$ == $b.termArray[j].exp$)
- $c.termArray[k].exp$ = ($a.termArray[i].exp$ + $b.termArray[j].exp$);
- Another example is shown in Figure 2.1. An old implementation using an array.

- Using two arrays, `float * coef` and `int * exp`.
- A polynomial `a`, `a.start` and `a.finish` point to start and finish locations where `a` stored in arrays.
- Figure 2.1.
- A garbage collection problem.
  - compare `new` and `delete` operations for different approaches (C++ and the old array approaches)

Sparse Matrices

Matrix

- mathematical object arises in many physical science problems.
- Study the way to represent matrices so that the operations to be performed on them can be carried out efficiently.
- A matrix: m rows and n columns, an $m \times n$ matrix, there are mn elements,
- $m = n$, square matrix.

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

- A matrix can be stored in a C++ 2D array, $a[m][n]$, element $a[i][j]$ can be efficiently access.
- The second matrix contains many zeros, a *sparse* matrix.
- In practice, many matrices are large but sparse, develope a sparse matrix ADT.

class $SparseMatrix$
{
public:
    $SparseMatrix$ (int r, int c, int t);
    $SparseMatrix\ Transpose()$;
    $SparseMatrix\ Add(SparseMatrix\ b);$
    $SparseMatrix\ Multiply(SparseMatrix\ b);$
};

Sparse Matrix Representation

- Try not to store zero terms.
- A nonzero term can be represented by a triple $< row, col, value >$.
- use an array of triples to store the non-zero terms.
- To make operations easier, the non-zero terms are stored according
  - sorted according to row index,
  - sorted according to column index in the same row.

class $SparseMatrix$;

class $MatrixTerm$ {
friend class $SparseMatrix$;
private:
    int row, col, $value$;
};

and in class $SparseMatrix$;

private:
    int $rows$, $cols$, $terms$, $capacity$;
    $MatrixTerm *smArray$;

Left, Sparse Matrix of (b), Right, its transpose

	row	col	$value$		row	col	$value$
$smArray$[0]	0	0	15	$smArray$[0]	0	0	15
$smArray$[1]	0	3	22	$smArray$[1]	0	4	91
$smArray$[2]	0	5	-15	$smArray$[2]	1	1	11
$smArray$[3]	1	1	11	$smArray$[3]	2	1	3
$smArray$[4]	1	2	3	$smArray$[4]	2	5	28
$smArray$[5]	2	3	-6	$smArray$[5]	3	0	22
$smArray$[6]	4	0	91	$smArray$[6]	3	2	-6
$smArray$[7]	5	2	28	$smArray$[7]	5	0	-15

Transposing a Matrix

Change row and column, $A = B^T$, $A[i][j] = B[j][i]$; Direct change row and column in sparse matrix representation, row index is not sorted.

	row	col	$value$		row	col	$value$
$smArray$[0]	0	0	15	$smArray$[0]	0	0	15
$smArray$[1]	0	3	22	$smArray$[1]	3	0	22
$smArray$[2]	0	5	-15	$smArray$[2]	5	0	-15
$smArray$[3]	1	1	11	$smArray$[3]	1	1	11
$smArray$[4]	1	2	3	$smArray$[4]	2	1	3
$smArray$[5]	2	3	-6	$smArray$[5]	3	2	-6
$smArray$[6]	4	0	91	$smArray$[6]	0	4	91
$smArray$[7]	5	2	28	$smArray$[7]	2	5	28

- Try to fix the unsorted problem.
  - observation: row are originally ordered,
  - find all elements in column 0 and store them in row 0, find all elements in column 1, ...
  - to process column i, scan the whole list once,
  - $cols$, $rows$, and $terms$ denote number of columns, number of rows, and number of terms.
  - total time required is $O(terms \cdot cols)$.
  - in the worst case, $terms$ = $cols \cdot rows$, i.e., we have $O(rows \cdot cols^2)$, worse than a straightforward implementation that takes $\Theta(rows \cdot cols)$ time.

$FastTranspose$

- First determine the number of elements in each column.
- We can then calculate the starting positon for each row in the transposed matrix.

	[0]	[1]	[2]	[3]	[4]	[5]
$rowSize$	2	1	2	2	0	1
$rowStart$	0	2	3	5	7	7

- Total time $O(cols + terms)$.

Representation of Arrays

- Multidimensional arrays are stored in a 1-D array. (Actually, any data type objects are stored in a 1-D array, i.e., memory.) And recall that we need random access.
- An array declared as $a[u_1][u_2], \cdots, [u_n]$, the number of elements $\Pi_{i-1}^{n} u_i$.
- Two way to store array $a[m][n]$, row major and column major. Row major, store array row by row, Column major, column by column.

- 1-D array $a[m]$, a the starting address of array a (address of $a[0]$), denoted α, $a[i]$ stored at $\alpha + i$.
- 2-D array $a[m][n]$, a address of $a[0][0]$, $a[i][j]$ is stored at $\alpha + i \cdot n + j$.
- What if a sparse matrix
  - upper triangle
  - band
- Given a sparse matrix $A[i][j]$, we store only the nonzero terms in a 1D array B, s.t. for a given (i, j) we return $A[i][j]$ by calculating the address we store $A[i][j]$ in B and retrieve it in B.

String Abstract Data Type

- C-string using array, C provides functions to precess string of char.
- A string ADT

```
class String
{
public:
  String (char *init, int m);
  int length();
  String Concat (String t);
  String Substr(int, i, int j);
  int Find(String pat);
}
```

String Matching

- Given two strings, s and pat, we usually say that s is the text and pat is the pattern.
- Determine whether pat in s by using the function $Find$, $Find$ returns -1 if pat is not in s or returns the position in s where pat starts.
- Naive appraoch, compareing pat with string starting $s[i]$ for every $0 \leq i \leq |s| - |pat|$.
- if matched, return i, otherwise, move to $i + 1$.
- Worst case will be $|s| \cdot |pat|$.
- Can we do better?

Knuth-Morris-Pratt Algorithm

- An $O(|pat| + |s|)$ time algorithm, and it is “optimal”.
- Basic idea, a mismatch also tells us something.
- Calculate information from the pattern, pat, so that when a mismatch occurs, we don’t have to start at the next position in s.

- The pattern $pat = abcabcacab$
- The string $s = s_0 s_1 \ldots s_{m-1}$.
- Suppose that we are determining whether or not there is a match beginning s_i.
- If $s_i \neq a$ then we proceed by comparing s_{i+1} and a.

s	=	-	a	?	?	?	.				
pat	=		a	b	c	a	b	c	a	c	a	b

- If $s_i = a$, we compare b against s_{i+1}
- if $b \neq s_{i+1}$, since that first question mark can be anything but not b,
- we proceed by comparing a and s_{i+1}.

$$\begin{array}{lclllllllll} s & = & \text{-} & \text{a} & \text{b} & ? & ? & . \\ pat & = & & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} & \text{c} & \text{a} & \text{c} & \text{a} & \text{b} \end{array}$$

- Suppose that we have $s_i s_{i+1} = ab$, but $c \neq s_{i+2}$,
- We proceed by comparing a and s_{i+2}
- because we know that s_{i+1} is b, it cannot be a, so we don't have to compare a and s_{i+1}. And s_{i+2} is not c but can be any other, so we compare a and s_{i+2}.

s	=	-	a	b	c	a	?	?				
pat	=		a	b	c	a	b	c	a	c	a	b

- Suppose that we match $abca$ and $s_i s_{i+1} s_{i+2} s_{i+3}$ but b does not match s_{i+4},
- Since we know s_{i+3} is a, we can proceed by comparing s_{i+4} and b, the 2nd character in pat.

Failure Function

If $p = p_0 p_1 \ldots p_{n-1}$ is a pattern, then the failure function, f, is defined as

$$f(j) = \begin{cases} \text{largest } k < j \text{ s.t. } p_0 \ldots p_k = p_{j-k} \ldots p_j & \text{if such a } k \geq 0 \text{ exist} \\ -1 & \text{otherwise.} \end{cases}$$

$pat = abcabcacab$

j	0	1	2	3	4	5	6	7	8	9
pat	a	b	c	a	b	c	a	c	a	b
f	-1	-1	-1	0	1	2	3	-1	0	1

Matching Algorithm

If a partial match is found s.t. $s_{i-j} \ldots s_{i-1}$ = $p_0 \ldots p_{j-1}$ and $s_i \neq p_j$ then matching may be resumed by comparing s_i and $p_{f(j-1)+1}$ if $j \neq 0$. If $j = 0$, then we may continue by comparing s_{i+1} and p_0.

j	0	1	2	3	4	5	6	7	8	9
pat	a	b	c	a	b	c	a	c	a	b
f	-1	-1	-1	0	1	2	3	-1	0	1

a	b	c	a	b	c	a	b	c	a	b	c	c	b	c	a	b	c	a	b	c	c
a	b	c	a	b	c	a	c														
			a	b	c	a	b	c	a	c											
			a	b	c	a	b	c	a	c											
						a	b	c	a	b	c	a									
									a	b	c	a									
													a								

Matching Algorithm

- Time complexity $O(length(S))$.

- the correctness of the algorithm depends on the failure function can be computed in $O(length(P))$.
- restate the failure function

$$f(j) = \begin{cases} -1 & \text{if } j = 0 \\ f^m(j-1)+1 & \text{where } m \text{ is the least integer} \\ & k \text{ for which } p_{f^k(j-1)+1} = p_j \\ -1 & \text{if there is no } k \text{ satisfying the above} \end{cases}$$

- $f^1(j) = f(j)$ and $f^m(j) = f(f^{m-1}(j))$
- Program 2.17 (next page), time complexity $O(length(P))$

j	0	1	2	3	4	5	6	7	8	9
pat	a	b	c	a	b	c	a	c	a	b
f	-1	-1	-1	0	1	2	3	-1	0	1

- If j = 6, $p_j = a$,
- $m = 1$, $f(5) + 1 = 3$, $p_3 = a = p_j$, then $f(6) = 3$

- If $j = 5$, $p_5 = c$,
- $m = 1$, $f(4) + 1 = 2$, $p_2 = c = p_5$, $f(5) = 2$.
-
- If $j = 9$, $p_9 = b$,
- $m = 1$, $f(8) + 1 = 1$, $p_1 = b = p_9$, $f(9) = 1$.
-
- If $j = 7$, $p_7 = c$,
- $m = 1$, $f(6) + 1 = 4$, $p_4 = b \neq p_7$,
- $m = 2$, $f(3) + 1 = 1$, $p_1 = b \neq p_7$,
- $m = 3$, $f(0) + 1 = 0$, $p_0 = a \neq p_7$,
- thus $f(7) = -1$.

```
void String::FailureFunction()
{
  int lengthP=Length();
  f[0]=-1;
  for (int j=1;j<legnthP; j++)
  {
        int i=f[j-1];
        while ((*(str+j)!=*(str+i+1)) && (i>=0))
          i=f[i];
        if (*(str+j)==*(str+i+1))
          f[j]=i+1;
        else f[j]=-1;
  }
}
```