Data Structures

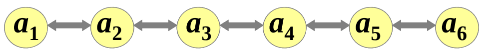
Linear Structure

Linear List

Definition

$$L = \begin{cases} (a_0, a_1, \ldots, a_{n-1}) & n \geq 1 \\ () & n = 0 \end{cases}$$

≤



- ✓ a_0
- ✓ a_{n-1}
- ✓ a_i
- ✓

Arrays

A set of pairs: <index, value>

correspondence or mapping

Two operations:

- Retrieve
- Store

Array can be used to implement other abstract data types

The simplest one might be: Ordered or linear list

Operations (Ӝ) on linear list, including

1 **Find** the length *n* of the list

2 **Read** the list from left to right (or right to left)

3 **Retrieve** the *ith* element, $0 \le i < n$

4 **Store** a new value into the *ith* position, $0 \le i < n$

5 **Insert** a new element at the position *i*, $0 \le i < n$

i,i+1,…,n–1 to i+1,i+2,…,n

6 **Delete** the element at position *i*, $0 \le i < n$

i+1,i+2,…,n–1 to I,i+1,…,n–2

Linear List ADT or GeneralArray

```
class            {                      ∈        ≤
public:
                ;                        ü     ἠ  ˣ  Ω
    int         ;                              Ҥ
    void              ;             ю  Ἀ ᾶ   ʐ ắ   ẇ
    float          int  ;                 Ω ẇ      ᵍ
    void       int  float   ;               ᵍ   J   Ω ẇ
    void       int  float   ;             Ӝ ɿ   Ω ẇ      Ậ
    float        int   ;            ó     Ω ẇ          ẽ ᵍ
};
```

❑Generally specified as a C++ (template) class

How to represent ordered list efficiently?

- Sequential mapping
  - Use array : a_i ←→ index i
- Complexity
  - Random access any element, T(n) = O(1)

```
float  Retrieve(int i);
// if (i∈IndexSet) return the float associated with i in the
//  array;else throw an exception.

void Store(int i, float x);
// if (i∈IndexSet) replace the old value associated with i
// by x;  else throw an exception.
```

Operations Insert and Delete

```
void  Insert(int i, float x);
// insert x as the indexth element, elements
// with theIndex >= index have their index increased by 1

void Delete(int i);
// remove and return the indexth element,
// elements with higher index  have their index reduced by 1
```

Insert

```
template <typename T>
bool Insert (T data[], int i, T x)
{
//     ẇ   x   Ậ Ậ   ά   i (1≤i≤n+1) Ω      Ɱ

    if (n == maxSize) return false;        //
    if (i < 1 || i > n+1) return false;    // ɜ  i Ή Ő
    for (int j = n; j >= i; j - -)         //  Ɱ , ˛
         data[j] = data[j - 1];
                                     //  Ậ (  i        data[i - 1]   )
    data[i - 1] = x;
    n++;

   return true;                        //  Ậ  ſ
};
```

Analysis

- Insert into *i*th position, need move backward from *data*[*i* - 1] to *data* [*n* - 1]

$$n - 1 - (i - 1) + 1 = n - i + 1$$

Average Moving Number

- when p_i =1/n, and for all position , 1 *i*   +

$$\text{AMN} = \frac{1}{n+1}\sum_{i=1}^{n+1}(n - i + 1) = \frac{1}{n+1}(n + \cdots + 1 + 0)$$

$$= \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

Remove

```
//            ɜ   x        ó ẇ
template <typename T>
bool Remove (T data[], int i, T & x)
{
    //ю   ά ό      i (1≤i≤n) Ω
    if (n == 0) return false;        //
    if (i < 1 || i > n) return false;  // ɜ   iዛ ŏ

    x = data[i-1];
    for (int j = i; j <= n-1; j++)  //  Ϻ ̦ ̗   Ϙ̣   ,
        data[j-1] = data[j];
    n--;

    return true;
};
```

Analysis

- If removed the *ith* term, need to move forward from *i+1th* to *nth*

  $n-(i+1)+1 = n-i$

- **AMN :**

$$AMN = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

  - when p_i =1/n, and 1 i -

Search

```
typedef int T; //
int search(T data[], int Size, T & x)
{
  //    ά         ɪ       g x ╚
 //  Ѧ ȅ ΰ                ἰ Ω ẇ
   //
   for (int i = 1; i <= Size; i++)
      if ( data[i-1] == x ) return i;
                                    //      ᾶ Ϥ     Ϻ    1

   return 0;  //
};
```

Analysis

Average Comparing Number

Success:

$$ACN=\sum_{i=1}^{n} p_i \times c_i$$

when p_i **=1/n ( )**

$$ACN = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}(1+2+\cdots+n) =$$

$$= \frac{1}{n} * \frac{(1+n)*n}{2} = \frac{1+n}{2}$$

Unsuccess : ACN = n

Data Structures

Polynomial

Polynomial ADT

class Polynomial {

// $p(x)=a_0x^{e0}+,\ldots,+ a_nx^{en}$

// a set of ordered pairs of $<e_i, a_i>$

// where a_i **is a nonzero float coefficient**

// and e_i **is a non-negative exponent**

public:

Polynomial ( );

// Construct the polynomial $p(x)=0$

```
    void AddTerm (Exponent e, Coefficient c);
    // add the term <e,c> to *this, so that it can be initialized

    Polynomial Add (Polynomial poly);
    // return the sum of the polynomials *this and poly

    Polynomial Mult (Polynomial poly);
    // return the product of the polynomials *this and poly

    float Eval ( float f);
    // evaluate polynomial *this at f and return the result
}
```

Polynomial Representation 1

```
private:
    int degree;            // degree ≤ MaxDegree
    float coef[MaxDegree+1];
    a.degree = ?           // n
    a.coef[i] = ?          // a_{n-i} , 0 ≤i≤ n

    // Simple algorithms for many operations
```

When a.degree << MaxDegree, representation 1 is very poor in memory use.

Polynomial Representation 2

To improve, define variable sized data member as:

```
private:
    int degree;
    float *coef;  //

Polynomial::Polynomial(int d)
 {
    int degree=d;
    coef= new float[degree+1];   //
 }
```

Polynomial Representation 3

```
class Polynomial; // forward declaration
class Term {
   friend Polynomial;
  private:
     float coef;  // coefficient
     int exp;      // exponent
};

class Polynomial {
  public:
     // ……
  private:
    Term *termArray;
    int capacity;     // size of termArray
    int terms;          // number of nonzero terms
};
```

Addition

Use presentation 3 to obtain C = A +B

$A(x)=3x^2+ 2x+4$

$B(x)=x^4+ 10x^3+ 3x^2+1$

Idea:

- ✓ Because the exponents are in descending order, can adds A(x) and B(x) term by term to C(x)
- ✓ The terms of C are entered into its *termArray* by calling function NewTerm
- ✓ If the space in *termArray* is not enough, its capacity is doubled

```
Polynomial  Polynomial::Add (Polynomial b)
{   // return the sum of the polynomials *this and b
  Polynomial c;
  int aPos=0, bPos=0;
  while (( aPos < terms) && (b < b.terms))
     if (termArray[aPos].exp==b.termArray[bPos].exp) {
        float t = termArray[aPos].coef + termArray[bPos].coef
        if ( t )  c.NewTerm (t, termArray[aPos].exp);
        aPos++;  bPos++;
     }
     else if (termArray[aPos].exp < b.termArray[bPos].exp) {
         c.NewTerm (b.termArray[bPos].coef, b.termArray[bPos].exp);
         bPos++;
     }
```

```
  else {
     c.NewTerm (termArray[aPos].coef, termArray[aPos].exp);
     aPos++;
  }
} // end of while
// add in the remaining terms of *this
 for ( ; aPos < terms; aPos++ )
     c.NewTerm(termArray[aPos].coef, termArray[aPos].exp );
// add in the remaining terms of b
 for ( ; bPos < b.terms; bPos++ )
 c.NewTerm(b.termArray[bPos].coef, b.termArray[bPos].exp);
 return c;
}
```

```
void Polynomial::NewTerm(const float theCoeff, const int theExp)
{ // add a new term to the end of termArray
   if (terms == capacity)
   { // double capacity of termArray
      capacity *= 2;
      term *temp = new term[capacity]; // new array
      copy(termArray, termAarry + terms, temp);
      delete [ ] termArray;  // deallocate old memory
      termArray = temp;
   }
  termArray[terms].coef = theCoeff;
  termArray[terms++].exp = theExp;
}
```

Data Structures

Matrix

Teacher :   Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. http://inside.mines.edu/~dmehta/
4. ,

Representation

A natural way

- ✓ a[m][n]
- ✓ access element by a[i][j], easy operations
- ✓ But for sparse matrix, wasteful of both memory and time

Alternative way

- ✓ store nonzero elements explicitly
- ✓ 0 as default

Sparse Matrix ADT

```
class SparseMatrix
{ // a set of <row, column, value>, where row, column are
  // non-negative integers and form a unique combination;
  //  value is also an integer.
 public:
    SparseMatrix ( int r, int c, int t);
    // creates a r×c SparseMatrix with a capacity of t nonzero
    // terms
    SparseMatrix Transpose ( );
    // return the SparseMatrix obtained by transposing *this
    SparseMatrix  Add ( SparseMatrix b);
    SparseMatrix  Multiply ( SparseMatrix b);
};
```

Sparse Matrix Representation

- ✓ Triple *<row, col, value>*
- ✓ Sorted in ascending order by *<row, col>*

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

✓ Need also

the **number** of rows

the **number** of columns

the **number** of nonzero elements

✓ in class SparseMatrix

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

Triple representation

	0	1	2	3	4	5
0	15	0	0	22	0	-15
1	0	11	3	0	0	0
2	0	0	0	-6	0	0
3	0	0	0	0	0	0
4	91	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
5	0	0	28	0	0	0

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

Transposing ( ) a Matrix

✓ 2-dimensional ( ) representation

✓ if an element is at position *[i][j]* in the original matrix

✓ then it is at position *[j][i]* in the transposed matrix

```
for (int j=0; j < columns; j++)
    for (int i=0; i < rows; i++)
        B[j][i] = A[i][j];
```

T(n)=O(cols* rows)

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

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

First try the transpose :

for (each row i)

✓take element *(i, j, value)*

✓store it in *(j, i, value)*

Improvement: **for (all elements in col j)**
store *(i, j, value)* of the original matrix
as *(j, i, value)* of the transpose

- **Since the rows are in order**
  - so7̄6(i)-./T039TT121 1 Tf4/T039TT121 1 Tf4/T0395.6(77̄6(i)-.56800be>1.1(0.Tc.0028Tw[34)831.002h[34)8s 0 0 .8cn4.118 0 TDn w

FastTranspose Algorithm

Step1: get Acol value

Acol is the number of elements in each column of *this

Step2: Brow = Acol

Brow is the number of elements in each row of B

Step3: obtain Bstart

Bstart is the starting point in B of each of its rows

Step4: move the elements of *this one by one into their right position in B

	[0]	[1]	[2]	[3]	[4]	[5]
RowSize=	2	1	2	2	0	1
RowStart=	0	2	3	5	7	7

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

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

```
SparseMatrix SparseMatrix::FastTranspos ( )
{ // return the transpose of *this in O(terms+cols) time
  SparseMatrix b(cols, rows, terms);
  if (terms > 0)
  { // nonzero matrix
    int *rowSize = new int[cols];
    int *rowStart = new int[cols];
    // compute rowSize[i] = number of terms in row i of b
    fill(rowSize, rowSize + cols, 0);  // initialze
    for (i=0; i<terms; i++ )  rowSize[smArray[i].col]++;
```

```
// rowStart[i] = starting position of row i in b
  rowStart[0] = 0;
  for (i=1;i<cols;i++) rowStart[i]=rowStart[i-1]+rowSize[i-1];
  for (i=0; i<terms; i++)
  {       // copy from *this to b
          int j = rowStart[smArray[i].col];
          b.smArray[j].row = smArray[i].col;
          b.smArray[j].col = smArray[i].row;
          b.smArray[j].value = smArray[i].value;
          rowStart[smArray[i].col]++;
   }  // end of for
  delete [ ] rowSize;  delete [ ] rowStart;
 }  // end of if
 return b;
}
```

Data Structures

Strings

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. http://inside.mines.edu/~dmehta/
4. ,

String ADT

- A string $S = s_0, s_1, \ldots, s_{n-1}$
  - where $s_i \in$ char , $0 \le i < n$, n is the length

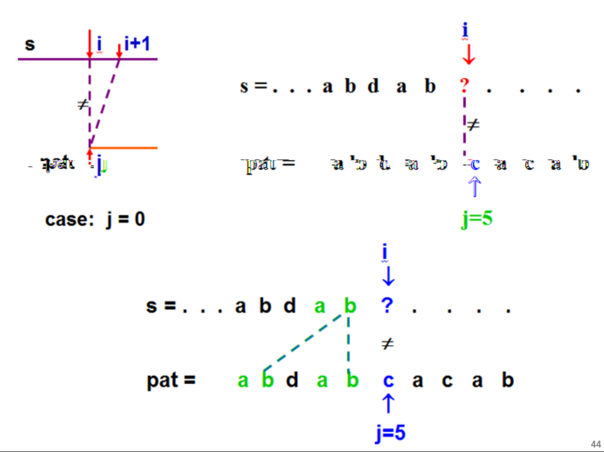
```
class String
{   // a finite set of zero or more characters
 public:
    String (char *init, int m );
    // initialize *this to string init of length m
```

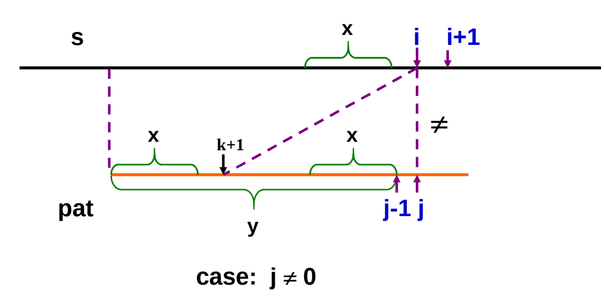
```
bool operator == (String t );
                    // if *this equals t, return true else false
bool operator ! (  );
                   // if *this is empty return true else false
int Length ( );
                  // return the number of chars in *this
 String Concat (String t);
 String Substr (int  i, int  j);
 int Find (String pat);
  // return i such that pat matches the substring of *this that begins  g2015.4170.491 -1.448TD.0029Tc[(St)5.2(r5.3(368f)-16.0T0 (positio)]T-1.42.2545.3(n)34of)-8( of)-17)28(char–1t,)-7 r)1477.4.0
```

String Pattern Matching: KMP Algorithm

- KMP : Knuth-Morris-Pratt
- This is optimal for B-F algorithm
  - *avoid rescanning* ?
  - O(LengthP + LengthS) ?
  - in the worst it is necessary to look at characters in the pattern and string at least once

- Determine where to continue the search and avoid moving backwards in the string





```
void String::Failurefunction( )
{           // compute the failure function of the pattern *this
  int LengthP= Length( );
  f [0]= -1;
  for (int j=1; j< LengthP; j++)        // compute  f[j]
  {  int i=f [j-1];
     while ( (str[j]!=str[i+1]) && (i>=0)) i=f[i]; // try for m
     if ( str[j]==str[i+1]) f[j]=i+1;   // f^m(j-1)+1
     else f[j]= -1;
  }
}
```