Data Structures

Linear Structure

Teacher： Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 金远平, 数据结构
3. http://inside.mines.edu/~dmehta/
4. 殷人昆, 数据结构

Linear List

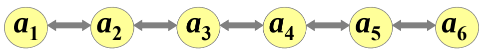
Definition

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

// a_i : element , a finite set $0 \leq i < n$

// empty

// n : length of linear list



- ✓ The first element a_0 has an unique successor
- ✓ The last element a_{n-1} has an unique precursor
- ✓ The other elements a_i have unique successors and precursors
- ✓ Assume : each element has the same data type

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**
- Now we will use the C++ class to define an ADT

Operations (操作) 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 LinearList {
// 对象: L = ( a0, a1, …, an-1) 或 ( ), ai∈T/type, 0 ≤ i < n
public:
    LinearList ( );              // 构造函数, 创建一个空表
    int Length( );               // 返回该实例的长度
    void LeftToRight( );         // 从左到右遍历全部元素
    float Retrieve( int i );     // 返回第i个元素的值
    void Store( int i, float v );   // 将v的值赋予第i个元素
    void Insert( int i, float v );  // 将v作为第i个元素插入
    float Delete( int i );       // 删除第i个元素并返回其值
};
```

❑**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)
{
//将新元素x插入到表中第i (1≤i≤n+1) 个表项位置

    if (n == maxSize) return false;        //表满
    if (i < 1 || i > n+1) return false;    //参数i不合理
    for (int j = n; j >= i; j - -)         //定位,依次后移
        data[j] = data[j-1];
                                //插入(第 i 表项在data[i-1]处)
    data[i-1] = x;
    n++;

   return true;                 //插入成功
};
```


Analysis

- Insert into *ith* 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 , $1 \leqslant i \leqslant n+1$

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

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

Remove

```
//通过引用型参数 x 返回被删元素
template <typename T>
bool Remove (T data[], int i, T & x)
{
     //从表中删除第 i (1≤i≤n) 个表项
     if (n == 0) return false;          //表空
     if (i < 1 || i > n) return false;  //参数i不合理

     x = data[i-1];
     for (int j = i; j <= n-1; j++)  //定位,依次前移,填补
         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 :

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

  - when p_i =1/n, and $1 \leqslant i \leqslant n-1$

Search

```
typedef  int  T;   //
int search(T data[], int Size, T & x)
{
   //在表中顺序搜索与给定值 x 匹配的表项
  //找到则函数返回该表项是第几个元素,
    //顺序搜索
    for (int i = 1; i <= Size; i++)
        if ( data[i-1] == x ) return i;
                                        //表项序号和表项位置差1

    return 0;   //搜索失败
};
```

Analysis

- **Average Comparing Number**

  Success:

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

when p_i **=1/n （等概率）**

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

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

Unsuccess： ACN = n

Data Structures

Polynomial

Teacher： Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 金远平, 数据结构
3. http://inside.mines.edu/~dmehta/
4. 殷人昆, 数据结构

Polynomial ADT

```
class Polynomial {
    //  p(x)=a_0 x^e0+,…,+ a_n x^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
}
```

王伟, 计算机工程系, 东南大学

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.

王伟, 计算机工程系, 东南大学

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]; //
}
```

王伟, 计算机工程系, 东南大学

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. 金远平, 数据结构
3. http://inside.mines.edu/~dmehta/
4. 殷人昆, 数据结构

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);
};
```

王伟, 计算机工程系, 东南大学

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;
};
```

王伟, 计算机工程系, 东南大学

✓ 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	0	0	0	0	0
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
[6]	4	0	91
[7]	5	2	28

Transposing (转置) a Matrix

✓ 2-dimensional (二维) 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**
  - **Locate elements in the correct column order**

	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

Further improvement :

- ✓ **If use some more space to store *some knowledge* about the matrix**
- ✓ **Can do much better : doing it in O(cols + terms)**

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]	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


```
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. 金远平, 数据结构
3. http://inside.mines.edu/~dmehta/
4. 殷人昆, 数据结构

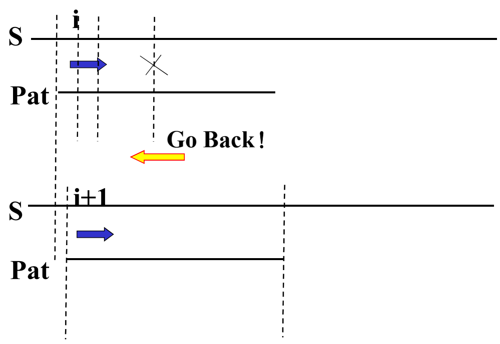
String ADT

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

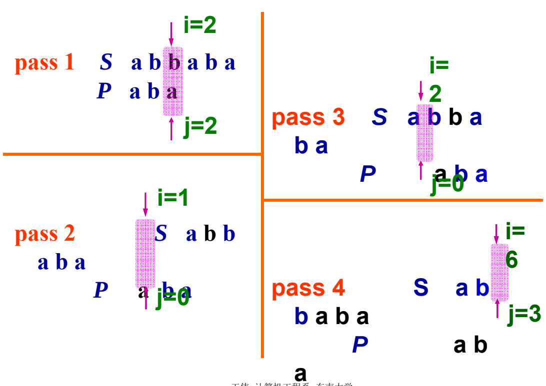
```
class String
{  // a finite set of zero or more characters
 public:
    String (char *init, int m );
    // initialize *this to string init of length m
```

```
    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 at
position i
      // return –1 if pat is either empty or not a substring of *this
private:
   char* str;
};
```


String Pattern Matching : Simple Algorithm

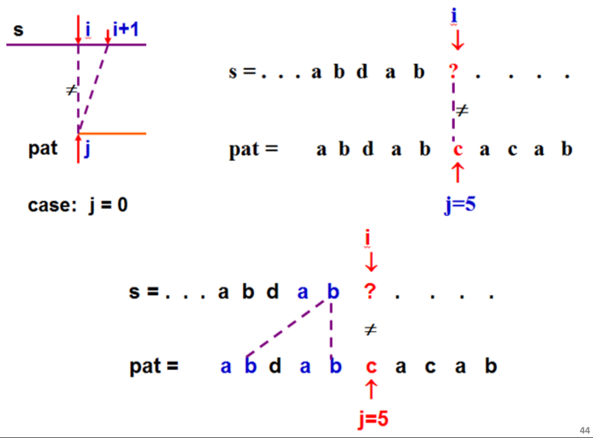


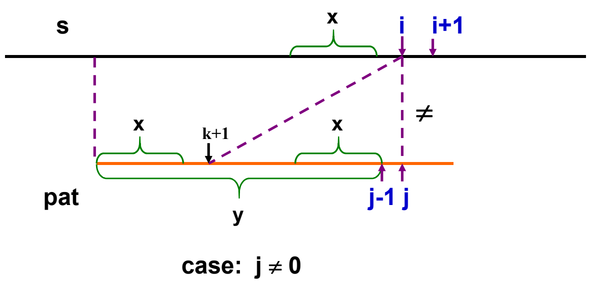
Simple Algorithm : B-F Algorithm



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





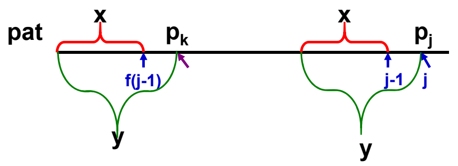
Failure Function

$$f(j) = \begin{cases} \text{largest } k < j, \text{ such that } p_0p_1\cdots p_k = p_{j-k}p_{j-k+1}\cdots p_j & \text{if such } k \geq 0 \text{ exists} \\ -1 & \text{, otherwise} \end{cases}$$

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

Compute $f(j)$

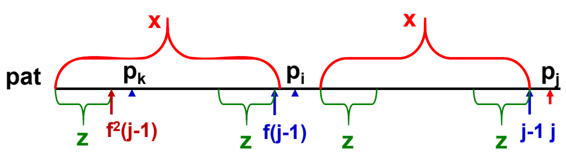
- $f(0) = -1$
- if have $f(j-1)$, by the following observation, compute $f(j)$



if $p_k = p_j$, then `f(j) = f(j-1)+1`

else …

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



if $pk = pj$, `f(j) = f(f(j-1))+1 = f²(j-1)+1`

else `f¹(j) = f(j)`

……

`fᵐ(j) = f(fᵐ⁻¹(j))`

```
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;
  }
}
```

王伟, 计算机工程系, 东南大学