Chapter 4 Linear Lists Using Direct Access

Yonghui Wu
School of Computer Science, Fudan University
yhwu@fudan.edu.cn
Wechat: 13817360465

Chapter 4 Linear Lists Using Direct Access

- 4.1 Application of Arrays(1)Calculation of Dates
- 4.2 Application of Arrays(2)Calculation of High Precision Numbers
- 4.3 Application of Arrays(3)Representation and Computation of Polynomials
- 4.4 Application of Arrays(4) Calculation of Numerical Matrices
- 4.5 Character Strings(1)Storage Structure of Character Strings
- 4.6 Character Strings(2)Pattern Matching of Character Strings

- Linear lists accessed directly
  - an element can be accessed directly without visiting its predecessor or successor.
- Array is one of these kinds of data structure.

4.1 Application of Arrays(1): Calculation of Dates

4.1.1 Calendar

- **Source: ACM Shanghai 2004 Preliminary**
- **IDs for Online Judge: POJ 2080, ZOJ 2420**

- A calendar is a system for measuring time, from hours and minutes, to months and days, and finally to years and centuries. The terms of hour, day, month, year and century are all units of time measurements of a calendar system.
- According to the Gregorian calendar, which is the civil calendar in use today, years evenly divisible by 4 are leap years, with the exception of centurial years that are not evenly divisible by 400. Therefore, the years 1700, 1800, 1900 and 2100 are not leap years, but 1600, 2000, and 2400 are leap years.
- Given the number of days that have elapsed since January 1, 2000 A.D, your mission is to find the date and the day of the week.

- **Input**
- The input consists of lines each containing a positive integer, which is the number of days that have elapsed since January 1, 2000 A.D. The last line contains an integer −1, which should not be processed.
- You may assume that the resulting date won’t be after the year 9999.
- **Output**
- For each test case, output one line containing the date and the day of the week in the format of "YYYY-MM-DD DayOfWeek", where "DayOfWeek" must be one of "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" and "Saturday".

Analysis

- Two functions are designed (***bottom-up***):
  - *days_of_year*(*year*): Calculate the number of days in *year*.
    - If *year* is a leap year, the number of days in *year* is 366;
    - otherwise the number of days in *year* is 365.
  - *days_of_month*(*month*, *year*): Calculate the number of days in *month*, *year*.
    - If *month* ==2 and *year* is a leap year, the number of days is 29; else the number of days is 28.
    - If *month* ==1, 3, 5, 7, 8, 10, or 12, the number of days is 31.
    - If *month* == 4, 6, 9, or 11, the number of days is 30.

- January 1, 2000 (Saturday) as the benchmark.
- *year*, *month* and *day* are variables;
- *wstr* is a string array storing the day of the week,
  - *wstr*[0..6]={"Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}. Initially *year*=2000, *month*=1, and *day*=1.
- *n* is the number of days that have elapsed since January 1, 2000 A.D..

Steps finding the date and the day of the week

- Step 1: calculate the day of the week:
  - January 1, 2000 (Saturday) is the benchmark,
  - *wstr*[0..6]={"Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}.
  - *wstr*[*n* % 7] is the day of the week.

Steps finding the date and the day of the week

- Step 2: calculate *year*:
  - while $n \geq days\_of\_year(year)$), repeat statements $n -= days\_of\_year(year)$; and $++year$.
  - When the loop ends, *year* is calculated, and *n* is the number of days in *year*.

Steps finding the date and the day of the week

- Step 3 is to calculate *month* and *day*:
  - while $n \geq days\_of\_month(month, year)$, repeat statements n-= $days\_of\_month(month, year)$; ++*month*.
  - When the loop ends, *month* is calculated. And statement *day* += *n* is to calculate *day*.

4.2 Application of Arrays(2): Calculation of High Precision Numbers

- In programming languages, range and precision for integer and real are limited.
- Two fundamental problems:
  - Representation of high precision numbers;
  - Fundamental calculations of high precision numbers;

Representation of high precision numbers

- A high precision number can be represented by an array:
  - numbers are separated by decimal digits,
  - each decimal digit is sequentially stored into an array.

Representation of high precision numbers

- In the program,
  - a string is used to store a number data, and a character stores a digit.
  - the string is conversed to the corresponding decimal number and stored in an array.

For a long positive integer, the program segment

- int *a*[100]={0}; // Array *a* is used to store a long positive integer, one digit is stored in one element. Initial values are 0.
- int *n*; // *n* is the number of digits for the long integer
- string *s*; // String *s* is used to receive the integer
- cin>>*s*; // Input the integer into *s*
- *n*=*s*.length( ); // Calculate the number of digits
- for (*i*=0; *i*<*n*; *i*++) // Array *a* stores the integer from right to left, and one element stores one digit.
- *a*[*i*]=s[*n*-*i*-1]-'0';

Fundamental calculations of high precision numbers

- Fundamental calculations of high precision numbers are ‘+’, ‘-‘, ‘*’, and ‘/’.

Addition and Subtraction of High Precision Numbers

- Rules for addition and subtraction of high precision numbers are the same as rules of arithmetic addition and subtraction.
- In programs,
  - addition of high precision numbers needs to carry,
  - subtraction of high precision numbers needs to borrow.

The program segment for addition of *x* and *y*

- Suppose
  - *x* and *y* are two non-negative high precision integers
  - *n*1 is the number of digits of *x*, and *n*2 is the number of digits of *y*.
  - *x* and *y* are stored in array *a* and array *b* in above format.
- for (*i*=0; *i*<(*n*1>*n*2 ? *n*1 : *n*2); *i*++){ //Addition of two integers whose numbers of digits are *n*1 and *n*2 respectively
- *a*[*i*]=*a*[*i*]+*b*[*i*]; // Bitwise addition
- if (*a*[*i*]>9) { // Carry
- *a*[*i*]=*a*[*i*]-10;
- *a*[*i*+1]++;
- }
- }

The program segment for subtraction of *x* and *y*

- Suppose
  - *x* and *y* are two non-negative high precision integers (*x*>*y*),
  - *n* is the number of digits of *x*.
  - *x* and *y* are stored in array *a* and array *b* in above format.
  - If *x*<*y*, then *a* and *b* exchange each other, and take a negative after the subtraction.

```
for (i=0; i<n; i++) {
                    if (a[i]>=b[i])
                        a[i]= a[i]- b[i];
                    else                    // Borrow
                   { a[i]= a[i]+10-b[i];
                        a[i+1]--;
                    }
          }
```

Multiplication and Division of High Precision Numbers

- For multiplication of high precision numbers, firstly the number of digits of the product must be determined. Suppose a and b are two positive high precision integers. LA is the number of digits for a, and LB is the number of digits for b. The number of digits for the product of a and b is at least $LA+LB-1$. And the upper limit of the number of digits is $LA+LB$.
- The algorithm for multiplication of two positive high precision integers are as follow: Firstly calculate the product of each digit of the multiplicand and each digit of the multiplier, where the product of $a[i]$ and $b[j]$ is accumulated into array $c[i+j]$. Then the carry process is done in array c.

- for (*i*=0; *i*<=*LA*-1, *i*++) // The product of each digit of multiplicand *a* and multiplier *b* is accumulated to corresponding digits of array *c*
- for (*j*=0; *j*<=*LB*-1; *j*++)
- `    c[i+j] += a[i]*b[j];`
- for (*i*=0; *i*<*LA*+*LB*; *i*++) // Carry
- if(*c*[*i*] >= 10)
- `  {`
- `    c[i+1] += c[i]/10;`
- `    c[i] %=10;`
- `  }`

4.2.1 Adding Reversed Numbers

- **Source: ACM Central Europe 1998**
- **IDs for Online Judge: POJ 1504, ZOJ 2001, UVA 713**

- The Antique Comedians of Malidinesia prefer comedies to tragedies. Unfortunately, most of the ancient plays are tragedies. Therefore the dramatic advisor of ACM has decided to transfigure some tragedies into comedies. Obviously, this work is very hard because the basic sense of the play must be kept intact, although all the things change to their opposites. For example the numbers: if any number appears in the tragedy, it must be converted to its reversed form before being accepted into the comedy play.
- Reversed number is a number written in arabic numerals but the order of digits is reversed. The first digit becomes last and vice versa. For example, if the main hero had 1245 strawberries in the tragedy, he has 5421 of them now. Note that all the leading zeros are omitted. That means if the number ends with a zero, the zero is lost by reversing (e.g. 1200 gives 21). Also note that the reversed number never has any trailing zeros.
- ACM needs to calculate with reversed numbers. Your task is to add two reversed numbers and output their reversed sum. Of course, the result is not unique because any particular number is a reversed form of several numbers (e.g. 21 could be 12, 120 or 1200 before reversing). Thus we must assume that no zeros were lost by reversing (e.g. assume that the original number was 12).

- **Input**
- The input consists of N cases. The first line of the input contains only positive integer N. Then follow the cases. Each case consists of exactly one line with two positive integers separated by space. These are the reversed numbers you are to add.
- **Output**
- For each case, print exactly one line containing only one integer - the reversed sum of two reversed numbers. Omit any leading zeros in the output.

Analysis

- Suppose
- *Num*[0][0] stores the length of the first addend, and the first addend is stoted in *Num*[0][1..*Num*[0][0]];
- *Num*[1][0] stores the length of the second addend, and the second addend is stored in *Num*[1][1..*Num*[1][0]];
- *Num*[2][0] stores the length of the sum; and the sum is stored in *Num*[2][1..*Num*[2][0]].

- Strings for the first addend and the second addend are input and zeros which the two numbers end with are deleted. The two addends are stored in *Num*[0] and *Num*[1]. Then they are changed into reversed numbers.
- Two reversed numbers *Num*[0] and *Num*[1] are added. Then their reversed sum *Num*[2] are output. And any leading zeros should be omited.

4.3 Application of Arrays(3): Representation and Computation of Polynomials

Representation and computation of polynomials is one of applications of linear lists accessed directly. A polynomial of one indeterminate is as follow.

$$P_n(x)=a_0+a_1x+a_2x^2+\cdots\cdots a_nx^n=\sum_{i=0}^{n}a_ix^i$$

Two storage methods for polynomials of one indeterminate

- ① Numeric array *a* is used to store a polynomial of one indeterminate.
- All elements' coefficients are stored in a array *a*[0..*n*] in exponents' ascending order (*n* is the highest degree).
- The index for *a* shows the number of exponent for the current element.
  - For example, if the *i*th element is empty, that is, in the polynomial the *i*th element's coefficient a_i=0, then the corresponding array element *a*[*i*]=0.
- Obviously, the length of array *a* lies on the highest degree of the polynomial.

Two storage methods for polynomials of one indeterminate

- ② Structure array *a* is used to store a polynomial of one indeterminate. Indexes for array *a* are serial numbers of elements. An array element is a structure containing its coefficient *a*[*i*].*coef* and exponent *a*[*i*].*exp*. Obviously the length of array *a* is the length of the polynomial.

Based on above data structures, computations of polynomials are introduced. For example,

$$\sum_{i=0}^{k1} a_i x^i + \sum_{i=0}^{k2} a_i x^i = \sum_{i=0}^{\max\{k1,k2\}} (a_i + b_i) x^i$$

$$\sum_{i=0}^{k1} a_i x^i * \sum_{j=0}^{k2} b_j x^j = \sum_{i=0}^{k1} (a_i x^i * \sum_{j=0}^{k2} (b_j x^j))$$

Similarly, subtraction and division of two polynomials and other polynomials' computations can also be implemented. If storage method ① is used, the storage of memory will be larger and algorithms will be simple. If storage method ② is used, the memory consumption will be reduced but the algorithms will be complex.

4.3.1 Polynomial Showdown

- **Source: ACM Mid-Central USA 1996**
- **IDs for Online Judge: POJ 1555, ZOJ 1720, UVA 392**

- Given the coefficients of a polynomial from degree 8 down to 0, you are to format the polynomial in a readable format with unnecessary characters removed. For instance, given the coefficients 0, 0, 0, 1, 22, -333, 0, 1, and -1, you should generate an output line which displays *x*^5 + 22*x*^4 - 333*x*^3 + *x* - 1.

- The formatting rules which must be adhered to are as follows:
- 1. Terms must appear in decreasing order of degree.
- 2. Exponents should appear after a caret "^".
- 3. The constant term appears as only the constant.
- 4. Only terms with nonzero coefficients should appear, unless all terms have zero coefficients in which case the constant term should appear.
- 5. The only spaces should be a single space on either side of the binary + and - operators.
- 6. If the leading term is positive then no sign should precede it; a negative leading term should be preceded by a minus sign, as in -7*x*^2 + 30*x* + 66.
- 7. Negated terms should appear as a subtracted unnegated term (with the exception of a negative leading term which should appear as described above). That is, rather than *x*^2 + -3*x*, the output should be *x*^2 - 3*x*.
- 8. The constants 1 and -1 should appear only as the constant term. That is, rather than -1*x*^3 + 1*x*^2 + 3*x*^1 - 1, the output should appear as-*x*^3 + *x*^2 + 3*x* - 1.

- **Input**
- The input will contain one or more lines of coefficients delimited by one or more spaces. There are nine coefficients per line, each coefficient being an integer with a magnitude of less than 1000.
- **Output**
- The output should contain the formatted polynomials, one per line.

Analysis

- Coefficient a_{n-i-1} whose exponent is n-i-1 is stored into array element $a[i]$. Array element $a[n-1]$ is the constant term. Initially, based on exponents' order from high to low, coefficients are inputted into $a[0..n-1]$.
- Non-zero term $a[i]$ ($a[i]\neq 0$, $i=0..n-1$) is analyzed from exponents' order from high to low. There are two cases: $a[i]$ is the first term or is not the first term:

- *a*[*i*] is the first term of the polynomial:
  - Coefficient: If *a*[*i*] == -1 and it is not a constant term (*i* < *n*-1), then output '-' directly; otherwise, if *a*[*i*]]≠1 or *a*[*i*] is a constant term (*i*==*n*-1), then output coefficient *a*[*i*].
  - Power: If the exponent is 1 (*i*==*n*-2), then output 'x' directly; else if it is not a constant term (*i*< *n*-1), output "x^"(*n*- *i*-1).
  - Reserve the mark of the first term of the polynomial.

- $a[i]$ is not the first term of the polynomial:
  - Sign: Output ($a[i] < 0$? '-' : '+');
  - Coefficient: If $a[i] \neq 1$ or -1, or $a[i]$ is a constant term, output the absolute value of $a[i]$;
  - Power: If the exponent is 1 ($i == n-2$), then output 'x' directly; else if it is not a constant term ($i < n-1$), output "x^"($n- i-1$).

- After dealing with the polynomial, if the mark of the first term of the polynomial is not changed, all coefficients are 0. Then output 0.

4.4 Application of Arrays(4): Calculation of Numerical Matrices

- Normally two-dimensional arrays are used to represent numerical matrices.
- Suppose the numbers of row and column of a matrix are *m* and *n* respectively. A two-dimensional array *a* is used represent a matrix, where *a*[*i*-1][*j*-1] represents the element in row *i* and column *j* in the matrix.

4.4.1 Error Correction

- **Source: Ulm Local Contest 1998**
- **IDs for Online Judge: POJ 2260, ZOJ 1949**

- A boolean matrix has the *parity property* when each row and each column has an even sum, i.e. contains an even number of bits which are set. Here's a 4 x 4 matrix which has the parity property:
- 1 0 1 0
- 0 0 0 0
- 1 1 1 1
- 0 1 0 1
- The sums of the rows are 2, 0, 4 and 2. The sums of the columns are 2, 2, 2 and 2.
- Your job is to write a program that reads in a matrix and checks if it has the parity property. If not, your program should check if the parity property can be established by changing only one bit. If this is not possible either, the matrix should be classified as *corrupt*.

- **Input**
- The input file will contain one or more test cases. The first line of each test case contains one integer n (n<100), representing the size of the matrix. On the next n lines, there will be n integers per line. No other integers than 0 and 1 will occur in the matrix. Input will be terminated by a value of 0 for n.
- **Output**
- For each matrix in the input file, print one line. If the matrix already has the parity property, print "OK". If the parity property can be established by changing one bit, print "Change bit (i,j)" where i is the row and j the column of the bit to be changed. Otherwise, print "Corrupt".

Analysis

- Suppose there is matrix a,
  - the sum of numbers in the i-th row is $row[i]$
  - the sum of numbers in the j-th column is $col[j]$

- matrix *a* is input,
- sums of all numbers in every row and all numbers in every columns are calculated,
- and are stored in array *row* and *col* respectively.

- The number of rows *cr* and the number of columns *cc* whose sum is odd are calculated.
- The last row number *i* and the last column number *j* whose the sum is odd are stored.
  - if (*row*[*k*]&1), then { *cr*++; *i*=*k*; };
  - if (*col*[*k*]&1), then { *cc*++; *j*=*k*; };
  - (0≤*k*≤*n*-1).

- Finally determine the parity property of matrix *a*:
  - If sums of all elements in every row and all elements in every columns are even (*cc*==0 and *cr*==0), then matrix *a* has the parity property, and "OK" is outputted.
  - If there is only one row and one column which has an odd sum (*cc*==1 and *cr*==1), then the bit in (*i*, *j*) makes sums odd, and the bit is changed to make matrix *a* has the parity property. Because row and column in array *a* are numbered from 0, (*i*+1, *j*+1) is outputted.
  - Otherwise “Corrupt” is output.

4.5 Character Strings(1): Storage Structure of Character Strings

A character string is a sequence of characters. The length of a string is the number of characters in the sequence. If the length is zero, the string is called an empty string. Character string s="$a_0a_1\ldots a_{n-1}$", where s is the name of the string, $a_0a_1\ldots a_{n-1}$ is the value of the string, and a_i ($0\le i\le n-1$) is a character in the string. The length of the string is n. '\0' is the end mark for a string and isn't regarded as a character in the string. Obviously, a character string is a linear list whose element is a character.

4.5.1 TEX Quotes

- **Source: ACM East Central North America 1994**
- **IDs for Online Judge: POJ 1488, UVA 272**

- TEX is a typesetting language developed by Donald Knuth. It takes source text together with a few typesetting instructions and produces, one hopes, a beautiful document. Beautiful documents use double-left-quote and double-right-quote to delimit quotations, rather than the mundane " which is what is provided by most keyboards. Keyboards typically do not have an oriented double-quote, but they do have a left-single-quote ` and a right-single-quote '. Check your keyboard now to locate the left-single-quote key ` (sometimes called the "backquote key") and the right-single-quote key ' (sometimes called the "apostrophe" or just "quote"). Be careful not to confuse the left-single-quote ` with the "backslash" key \. TEX lets the user type two left-single-quotes `` to create a left-double-quote and two right-single-quotes '' to create a right-double-quote. Most typists, however, are accustomed to delimiting their quotations with the un-oriented double-quote ".

- If the source contained
- "To be or not to be," quoth the bard, "that is the question."
- then the typeset document produced by TEX would not contain the desired form: "To be or not to be," quoth the bard, "that is the question." In order to produce the desired form, the source file must contain the sequence:
- ``To be or not to be," quoth the bard, ``that is the question."

- You are to write a program which converts text containing double-quote (") characters into text that is identical except that double-quotes have been replaced by the two-character sequences required by TEX for delimiting quotations with oriented double-quotes. The double-quote (") characters should be replaced appropriately by either `` if the " opens a quotation and by " if the " closes a quotation. Notice that the question of nested quotations does not arise: The first " must be replaced by ``, the next by ", the next by ``, the next by ", the next by ``, the next by ", and so on.

- **Input**
- Input will consist of several lines of text containing an even number of double-quote (") characters. Input is ended with an end-of-file character.
- **Output**
- The text must be output exactly as it was input except that:
- the first " in each pair is replaced by two ` characters: `` and
- the second " in each pair is replaced by two ' characters: ''.

Analysis

- Substitution forms for each pair of double-quotes appear alternately.
  - the first " is replaced by two ` characters,
  - the second " is replaced by two ' characters.

- Suppose $p[0]$ is the first substitution form"``" for the first ", and $p[1]$ is the second substitution form""" for the second ".
- Initially k=0.
- Then the string is scanned.
  - If the current character is not a double-quote, then output it;
  - else replace it by $p[k]$, and change the substitution form ($k = !k$) .