第 3 章 线性结构的扩展

1. 选择题

（1）C （2）B （3）A （4）DAB （5）CCC （6）C

2. 判断题

（1）X （2）√ （3）√ （4）X （5）X （6）X （7）X

3. 简答题

1．KMP 算法较朴素的模式匹配算法有哪些改进?

KMP 算法主要优点是主串指针不回溯。当主串很大不能一次读入内存且经常发生部分匹配时，KMP 算法的优点更为突出。

2．设字符串 S=‘aabaabaabaac'，P=‘aabaac'。

（1）给出 S 和 P 的 next 值和 nextval 值；

（2）若 S 作主串，P 作模式串，试给出利用 KMP 算法的匹配过程。

【解答】

（1）S 的 next 与 nextval 值分别为 012123456789 和 002002002009，p 的 next 与 nextval 值分别为 012123 和 002003。

（2）利用 BF 算法的匹配过程：

第一趟匹配：aabaabaabaac
aabaac(i=6,j=6)

第二趟匹配：aabaabaabaac
aa(i=3,j=2)

第三趟匹配：aabaabaabaac
a(i=3,j=1)

第四趟匹配：aabaabaabaac
aabaac(i=9,j=6)

第五趟匹配：aabaabaabaac
aa(i=6,j=2)

第六趟匹配：aabaabaabaac
a(i=6,j=1)

第七趟匹配：aabaabaabaac
(成功) aabaac(i=13,j=7)

利用 KMP 算法的匹配过程：

第一趟匹配：aabaabaabaac
aabaac(i=6,j=6)

第二趟匹配：aabaabaabaac
(aa)baac

第三趟匹配：aabaabaabaac
(成功) (aa)baac

3. 假设按行优先存储整数数组 A[9][3][5][8]时，第一个元素的字节地址是 100，每个整数占 4 个字节。问下列元素的存储地址是什么？

(1) a_{0000} (2)a_{1111} (3)a_{3125} (4)a_{8247}

【解答】(1) LOC(a_{0000})= 100

(2) LOC(a_{1111})=100+(3*5*8*1+5*8*1+8*1+1)*4=776

(3) LOC(a_{3125})=100+(3*5*8*3+5*8*1+8*2+5) *4=1784

(4) LOC(a_{8247})= 100+(3*5*8*8+5*8*2+8*4+7) *4=4816

4．假设一个准对角矩阵：

$$\begin{pmatrix} a_{11} & a_{12} & & & & & & \\ a_{21} & a_{22} & & & & & & \\ & & a_{33} & a_{34} & & & & \\ & & a_{43} & a_{44} & & & & \\ & & & & \cdots & & & \\ & & & & a_{ij} & & & \\ & & & & & & a_{2m-1,2m-1} & a_{2m-1,2m} \\ & & & & & & a_{2m,2m-1} & a_{2m,2m} \end{pmatrix}$$

按以下方式存储于一维数组 B[4m]中（m 为一个整数）：

0	1	2	3	4	5	6	…	k	…	4m-1	4m
a_{11}	a_{12}	a_{21}	a_{22}	a_{33}	a_{34}	a_{43}	…	a_{ij}	…	$a_{2m-1,2m}$	$a_{2m,2m-1}$	$a_{2m,2m}$

写出下标转换函数 k=f(i,j)。

【解答】

由题目可知，每一行有两个非 0 元素。

当 i 为奇数时，第 i 行的元素为：$a_{i,i}$、$a_{i,(i+1)}$，此时 k=2*(i-1)+j-i=i+j-2

当 i 为偶数时，第 i 行的元素为：$a_{i,(i-1)}$、$a_{i,i}$，此时 k=2*(i-1)+j-I+1=i+j-1

综上所述，k=i+j-i%2-1。

5. 设有 n×n 的带宽为 3 的带状矩阵 A，将其 3 条对角线上的元素存于数组 B[3][n]中，使得元素 B[u][v]=a_{ij}，试推导出从（i,j）到 (u,v)的下标变换公式。

【解答】

u=j-i+1

v=j-1

6．现有如下的稀疏矩阵 A（如图所示），要求画出以下各种表示方法。

（1）三元组表表示法

（2）十字链表法。

$$\begin{pmatrix} 0 & 0 & 0 & 22 & 0 & -15 \\ 0 & 13 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{pmatrix}$$

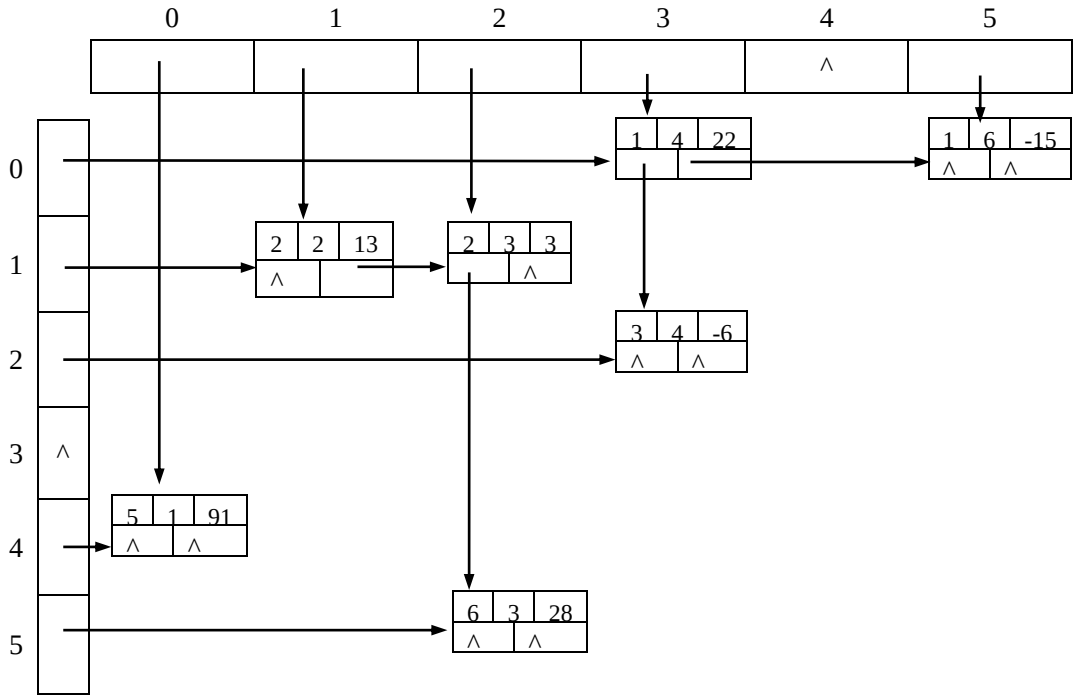
【解答】

（1）三元组表表示法:

	i	j	v
1	1	4	22
2	1	6	-15
3	2	2	13
4	2	3	3

5	3	4	-6
6	5	1	91
7	6	3	28

（2）十字链表法：

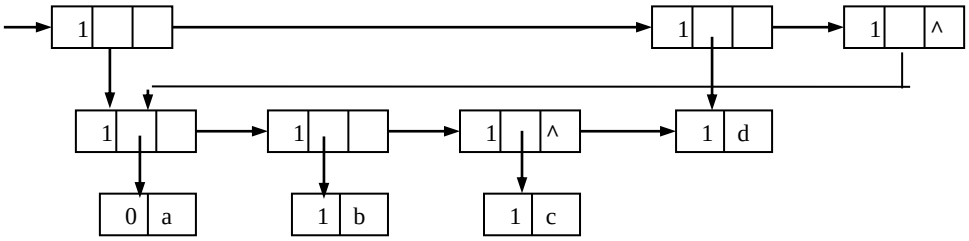


7．画出下列广义表的头尾表示存储结构示意图。

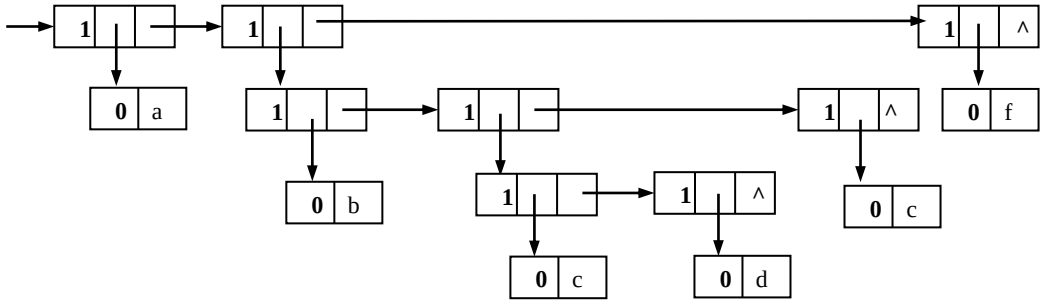
（1）A=((a,b,c),d,(a,b,c))

（2）B=(a,(b,(c,d),e),f)

（1）



（2）



2. 算法设计题

1．利用 C 的库函数 strlen,strcpy 和 strcat 写一个算法 void StrInsert(char *S,char *T,int i) ，将串 T 插入到 S 的第 i 个位置上。若 i 大于 S 的长度，则插入不执行。

```
void strinsert(char *S,char *t, int i)
    {if(i>strlen(S))    printf("error");
     strcopy(S+i,r);
     S[i]='\0';
     strcat(S,T);
     strcat(S,r);
    }
```

2．利用 C 的库函数 strlen,strcpy（或 strncpy）写一个算法 void StrDelete(char *S,int i,int m) ，删除串 S 中从位置 i 开始的连续的 m 个字符。若 i≥strlen(S)，则没有字符被删除；若 i+m≥strlen(S)，则将 S 中从位置 i 开始直至末尾的字符均被删去。

```
void StrDelete(char *S, int m)
    {if(i>=strlen(S)) return;
     if(i+m>=strlen(S)) S[i]='\0';
     else {for(k=i+m;S[k]!='\0';k++)
                S[k-m]=S[k];
           S[k]='\0';
          }
     }
```

3．采用顺序结构存储串，编写一个函数，求串 s 和串 t 的一个最长的公共子串。

```
void maxsubstr(SeqString*s, SeqString *t, SeqString *r)      /*求 S 和 T 的最长公共子串*/
   {int i,j,k,num,maxnum=0,index=0;
    i=0;
    while(i<s->curlen)
       {j=0;
        while(j<t->curlen)
            {if(s->data[i]==t->data[j])
                 {num=1;
                  for(k=1;s->data[i+k]==t->data[j+k];k++)
                          num=num+1;
                  if(num>maxnum)
                     {index=i;
                      maxnum=num;
                     }
                  j+=num;
                  i=0;
                 }
             else    j++;
            }
        i++;
       }
    for(i=index,j=0;i<index+maxnum;i++,j++)
         r->data[j]=s->data[i];
```

```
}
```

4．采用顺序存储结构存储串，编写一个函数，计算一个子串在一个字符串中出现的次数，如果该子串不出现，则次数为 0。

```
int count(SeqStirng *s,SeqString *t)
  {int i=0,c=0,j;
   while (i<StrLength(s))
      {j=StrIndex_KMP(s,t,i,next);
       if (j!=0) c++;
       i=i+StrLengrh(t);
      }
   return(c);
  }
```

5．假设稀疏矩阵 A 和 B（具有相同的大小 m*n）都采用三元组表存储，编写一个算法计算 C=A+B，要求 C 也采用三元组表存储。

```
SPMatrix *Matrix_Add (SPMatrix *A,SPMatrix *B)
  {SPMatrix    *C;
   C->mu=A->mu;
   C->nu=A->nu;
   C->tu=0;
   pa=1 ;
   pb=1 ;
   pc=1;
   while (pa<=A->tu && pb<=B->tu)
     {if ((A->data[pa].i==B->data[pb].i)&&(A->data[pa].j==B->data[pb].j))/*行号、列号相等时*/
        {C->data[pc].i=A->data[pc].i;
         C->data[pc].j=A->data[pc].j;
         C->data[pc].v=A->data[pa].v + B->data[pb].v;
         C->tu++ ;pc++ ; pa++ ; pb++ ;
        }
      else if ((A->data[pa].i < B->data[pb].i)||(A->data[pa].i==B->data[pb].i
                    &&A->data[pa].j< B->data[pb].j))          /*行号、列号不相等时*/
        {C->data[pc].i=A->data[pa].i;
         C->data[pc].j=A->data[pa].j;
         C->data[pc].v=A->data[pa].v;
         C->tu++ ; pc++ ;pa++;
        }
       else {C->data[pc].i=B->data[pb].i;
             C->data[pc].j=B->data[pb].j;
             C->data[pc].v=B->data[pb].v;
             C->tu++;
             pc++;
             pb++;
            }
     }
   while( pa<=A->tu )                                  /* A 中有剩余元素时*/
     {C->data[pc].i=A->data[pa].i ;
```

```
        C->data[pc].j=A->data[pa].j ;
        C->data[pc].v=A->data[pa].v ;
        pc++;
        pa++;
       }
    while ( pb<=B->tu )                                   /*B 中有剩余元素时*/
       {C->data[pc].i=B->data[pb].i;
        C->data[pc].j=B->data[pb].j;
        C->data[pc].v=B->data[pb].v;
        pc++;
        pb++;
       }
    return(c);
  }
```

6．假设稀疏矩阵 A 和 B（分别为 m*n 和 n*1 矩阵）采用三元组表存储，编写一个算法计算 C=A*B，要求 C 也是采用稀疏矩阵的三元组表存储。

```
SPMatrix * Matrix_Mul (SPMatrix *A, SPMatrix *B)
/*稀疏矩阵 A(m1×n1)和 B(m2×n2) 用三元组表存储，求 A×B */
  {SPMatrix    *C;                                      /*乘积矩阵的指针*/
   int p,q,i,j,k,r;
   datatype temp[n+1];
   int num[B–>mu+1], rpot[B–>mu+1];
   if(A–>nu!=B–>mu)    return NULL;                     /*A 的列与 B 的行不相等*/
   C=malloc(sizeof(SPMatrix));                          /*申请 C 矩阵的存储空间*/
   C–>mu=A–>mu;
   C–>nu=B–>nu;
   if(A–>tu*B–>tu==0)
     {C–>tu=0;
      return C; }
      for(i=1;i<=B–>mu;i++) num[i]=0;                   /*求矩阵 B 中每一行非零元素的个数*/
          for (k=1;k<=B–>tu;k++)
              {i=B–>data[k].i;
               num[i]++;
              }
      rpot[1]=1;                          /*求矩阵 B 中每一行第一个非零元素在 B.data 中的位置*/
      for (i=2;i<=B–>mu;i++)
          rpot[i]=rpot[i-1]+num[i-1];
      r=0;                                              /*当前 C 中非零元素的个数*/
      p=1;                                              /*指示 A.data 中当前非零元素的位置*/
      for(i=1;i<=A–>mu;i++)
          {for(j=1;j<=B–>nu;j++)    temp[j]=0;          /*cij 的累加器初始化*/
           while(A–>data[p].i==i )                      /*求第 i 行的*/
              {k=A–>data[p].j;                          /*A 中当前非零元的列号*/
               if(k<B–>mu)    t=rpot[k+1];
               else      t=B–>tu+1;   /*确定 B 中第 k 行的非零元素在 B.data 中的下限位置*/
               for (q=rpot[k]; q<t; q++;)               /*B 中第 k 行的每一个非零元素*/
```

```
                {j=B–>data[q].j;
                 temp[j]+=A–>data[p].v * B–>data[q].v;
                }
              p++;
             }
          for(j=1;j<=B–>nu;j++)
              if(temp[j])
               {r++;
                C->data[r]={i,j,temp[j]};
               }
        }
      C–>tu=r;
      return C;
  }
```

7．假设稀疏矩阵只存放其非 0 元素的行号、列号和数值，以一维数组顺次存放，以行号为-1 作为结束标志。例如如下图所示的稀疏矩阵 M：

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

则存在一维数组 D 中：

D[0]=1， D[1]=1，D[2]=1，D[3]=1，D[4]=5
D[5]=10，D[6]=3，D[7]=9，D[8]=5，D[9]= -1

现有两个如上方法存储的稀疏矩阵 A 和 B，它们均为 m 行 n 列，分别存放在数组 A 和 B 中，编写求矩阵加法 C=A+B 的算法，C 亦放在数组 C 中。

【提示】注意当 A、B 中元素的行和列都相同时才能相加。

```
void add (int A[],int B[],int C[])
  {pa=0;
   pb=0;
   pc=0;
   while(A[pa+2] && B[pb+2])                          /*当 A、B 都未结束时*/
       {if(A[pa]==B[pb]&&A[pa+1]==B[pb+1])            /*当行号、列号都相同时*/
           {C[pc]=A[pa]; C[pc+1]=A[pc+1];
            C[pc+2]=A[pa+2]+B[pb+2] ;
            pa+=3;
            pb+=3;
            pc+=3;
           }
        else if(A[pa]<B[pb]||(A[pa]==B[pb]&&A[pa+1]<B[pb+1]))  /*当行号、列号不等时*/
           {C[pc]=A[pa];
            C[pc+1]=A[pc+1];
            C[pc+2]=A[pc+2];
            pa+=3;
            pc+=3;
           }
```

```
            else{C[pc]=B[pb];
                 C[pc+1]=B[pb+1];
                 C[pc+2]=B[pb+2];
                 pb+=3;
                 pc+=3;
                }
        }
      while (A[pa+2]!=0)                                   /*A 中有剩余元素*/
        {C[pc]=A[pa];
         C[pc+1]=A[pa+1];
         C[pc+2]=A[pc+2];
         pa+=3;
         pc+=3;
        }
      while (B[pa+2]!=0)                                   /*B 中有剩余元素*/
        {C[pc]=B[pb];
         C[pc+1]=B[pb+1];
         C[pc+2]=B[pb+2];
         pb+=3;
         pc+=3;
        }
    }
```

8．已知 A 和 B 为两个 n*n 阶的对称矩阵，输入时，对称矩阵只输入下三角形元素，按压缩存储方法存入一维数组 A 和 B 中，编写一个计算对称矩阵 A 和 B 的乘积的算法。

【提示】注意对对称矩阵采用压缩存储时，元素的表示方法；乘积矩阵仍然采用压缩存储的方法。

```
void Mul (int A[], int B[], int C[], int n)
 /*计算 A 和 B 的乘积 C，其中 A、B、C 均为压缩存储的 n 阶对称矩阵*/
   {for(i=0;i<n;i++)
       for(j=0;j<n;j++)
          {mi=max(i,j);
           mj=min(i,j);
           x=mi*(mi-1)/2 + mj-1;                    /*计算矩阵元素 C[i][j]压缩后的存放地址*/
           C[x]=0;
           for (k=0; k<n; k++)
                {u1=max(i,k) ;
                 vi=min(i,k);
                 u2=max(k,j) ;
                 v2=min(k,j);
                 w1=u1*(u1-1)/2 +v1-1;              /*计算 A[i][k]的存放地址*/
                 w2=u2*(u2-1)/2 +v2-1;              /*计算 B[k][j]的存放地址*/
                 C[x]+=A[w1]*B[w2];
                }
          }
    }
```