第一章 绪论

1.（第18页，第（5）题）

确定下列各程序段的程序步，确定划线语句的执行次数，计算它们的渐近时间复杂度。

(1) i=1; k=0;

```
do {
   k=k+10*i; i++;
 } while(i<=n-1)
```

划线语句的执行次数为 n-1 。

(2) i=1; x=0;

```
do{
    x++; i=2*i;
} while （i<n）;
```

划线语句的执行次数为$\lceil \log_2 n \rceil$。

(3)

```
for(int i=1;i<=n;i++)
   for(int j=1;j<=i;j++)
      for (int k=1;k<=j;k++)
         x++;
```

划线语句的执行次数为$n(n+1)(n+2)/6$ 。

(4)

```
x=n;y=0;
while(x>=(y+1)*(y+1)) y++;
```

划线语句的执行次数为$\lfloor \sqrt{n} \rfloor$。

第二章 线性表

1. 第37页 习题(2).2

在类LinearList 中增加一个成员函数，将顺序表逆置，实现该函数并分析算法的时间复杂度。不利用类SeqList 提供的操作直接实现。

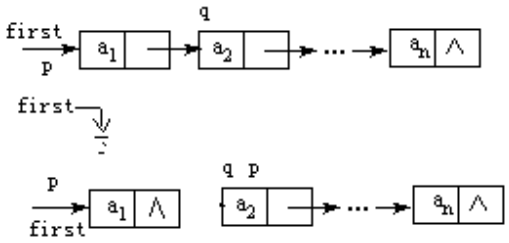
```
template <class T>
void SeqList<T>::Invert()
{
  T e;
  for (int i=1;i<=length/2;i++){
    e=elements[i-1];
    elements[i-1]=elements[length-i];
    elements[length-i]=e;
  }
}
```

2. 第37页习题(5)

在类SingleList中增加一个成员函数，将单链表逆置运算，直接实现该函数并分析其时间复杂度。

```
template <class T>
void SingleList<T>::invert()
{
    Node<T> *p=first,*q;
    first=NULL;
    while (p){
```

```
        q=p->link; p->link=first;
       first=p;  p=q;
    }
}
```



3.（第 37 页，第 7 题） 单链表中结点按元素值递增链接，在类 SingleList 中增加一个成员函数，直接实现删除结点值在 a 至 b 之间的结点(a≤b)。

```
template <class T>
void SingleList<T>::DeleteAb(T a,T b)//第 37 页，习题 (7)
{
Node<T> *p=first,*q=first;
    while (p && p->data<b){
  if (p->data<=a){
     q=p;p=p->link;
  }
  else if (q==first){
          q=p->link;
          delete p;
          p=first=q;
      }
      else{
          q->link=p->link;
          delete p;
          p=q->link;
      }
  }
}
```

4.（第 37 页，第 8 题） 在类 CircularList 中增加一个成员函数，在不增加新结点的情况下，直接实现两个链表合并为一个链表的算法，并分析其时间复杂度。

```
template<class T>
void Merge(CircularList<T> &a,
     CircularList<T> &b)
{
    Node<T> *p=b.first;
    while (p->link!=b.first) p=p->link;
    p->link=a.first->link;
    a.first->link=b.first->link;
    b.first->link=b.first;
```

```
    b.length=0;
}
template<class T>
void Merge1(CircularList<T> &a,
     CircularList<T> &b)
{
    Node<T> *p=b.first->link;
    b.first->data=b.first->link->data;
    b.first->link=a.first->link;
    a.first->link=p->link;
    p->link=p;
    b.first=p;
}
```

第三章 栈与队列

1. 第50页 习题(1)

设A、B、C、D、E五个元素依次进栈(进栈后可立即出栈)，问能否得到下列序列。若能得到，则给出相应的push和pop序列；若不能，则说明理由。

1) A,B,C,D,E　2) A,C,E,B,D

3) C,A,B,D,E　4) E,D,C,B,A

答：2）和3）不能。对2）中的E，B，D而言，E最先出栈，则表明，此时B和D均在栈中，由于，B先于D进栈，所以应有D先出栈。同理3)。

2. 第50页 习题（9）

利用栈可以检查表达式中括号是否配对，试编写算法实现之。

```
bool match(char a[],int n)
{
int top=-1;
for (int i=0;i<n;i++)
  if (a[i]=='(') top++;
      else if (a[i]==')')
          if (top>-1) top--;
          else return true;
      if (top>-1) return true;
 return false;
}
```

3. 第50页 习题(10)

声明并实现链式队列类LinkedQueue。

```
template <class T>
class LinkedStack: public Stack<T>
{
public:
    LinkedStack();
    ~LinkedStack();
    virtual bool IsEmpty() const;
    virtual bool IsFull() const;
    virtual bool GetTop(T& x);
    virtual bool Push(const T x);
```

```
    virtual bool Pop();
    virtual void SetNull();
private:
    Node<T> *top;
};

template <class T>
LinkedStack<T>::LinkedStack()
{
    top=NULL;
}

template <class T>
LinkedStack<T>::~LinkedStack()
{
    Node<T> *q;
    while (top){
        q=top->link;
        delete top;
        top=q;
    }
}

template <class T>
bool LinkedStack<T>::IsEmpty() const
{
    return !top;
}

template <class T>
bool LinkedStack<T>::IsFull() const
{
    return false;
}

template <class T>
bool LinkedStack<T>::GetTop(T &x)
{
    if (IsEmpty()){
        cout<<"Empty stack"<<endl;
        return false;
    }
    x=top->data;
    return true;
}

template <class T>
```

```
bool LinkedStack<T>::Push(const T x)
{
    Node <T> *q=new Node<T>;
    q->data=x;
    q->link=top;
    top=q;
    return true;
}

template <class T>
bool LinkedStack<T>::Pop()
{
    Node<T> *q=top;
    if (IsEmpty()){
        cout<<"Empty stack"<<endl;
        return false;
    }
    top=top->link;
    delete q;
    return true;
}

template <class T>
void LinkedStack<T>::SetNull()
{
    Node<T> *q;
    while (top){
        q=top->link;
        delete top;
        top=q;
    }
}
```

第四章 数组与字符串

1. 给出三维数组元素 A[i][j][k]的存储地址 loc(A[i][j][k])。

答：设有三维数组声明为 $A[n_1][n_2][n_3]$，每个元素占 k 个存储单元，则

loc(A[i][j][k])=loc(A[0][0][0])+k*(i*n_2*n_3+j*n_3+k)

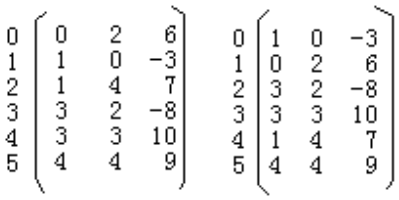
2.（第 68 页，第 5 题）给出下列稀疏矩阵的

$$\begin{bmatrix} 0 & 0 & 6 & 0 & 0 \\ -3 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & 10 & 0 \\ 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

题图4-5

顺序三元组的行优先和列优先表示。

答：



(a)行三元组　　(b)列三元组

3.（第 68 页，第 6 题）

对题图 4-5 的稀疏矩阵执行矩阵转置时数组 num[]和 k[]的值。

答：

col	0	1	2	3	4
num[col]	1	0	2	1	2
k[col]	0	1	1	3	4

4. （第 69 页，第 15 题（2））

在顺序串类 String 中增加一个成员函数，实现统计该串中出现的字符、字符个数和每个字符出现的次数。

```
void String::count(char ch[],int &n, int num[])
{
    n=0;
    for (int i=0;i<length;i++){
    int j=0;
    while (j<n && str[i]!=ch[j]) j++;
    if (j==n) {
        ch[j]=str[i];
        num[j]=1;
          n++;
    }
  else num[j]++;
}
}
```

第五章 递归

1. 设计一个递归算法，实现对一个有序表的顺序搜索。

```
template<class T>
int SeqList<T>::Search4(const T& x) const
{
   elements[length]=1000;
   return Sch4(x,0);
}
template<class T>
int SeqList<T>::Sch4(const T& x,int i) const
```

```
    {
       if (x<elements[i]) return 0;
       else if (x==elements[i]) return ++i;
            else return Sch4(x,i+1);
    }
```

补充题：

（2）求顺序搜索有序表失败时的平均搜索长度。设有序表为（$a_1, a_2, \ldots, a_n$），通常可以假定待查元素位于 a_1之前，a_1与 a_2之间，a_2与 a_3之间,…,a_{n-1}与 a_n之间以及 a_n之后的共 n+1 个位置处的概率是相等的。

答：

$$=\sum_{i=1}^{n+1} i \cdot p_i = \frac{1}{n+1} \sum_{i=1}^{n+1} i = \frac{n+2}{2}$$

第六章 树

1．第 107 页，第（2）题

对于三个结点 A，B 和 C，可分别组成多少不同的无序树、有序树和二叉树？

答：（1）无序树：9 棵

（2）有序树：12 棵

（3）二叉树：30 棵

2、（1）k 的 i－1 次方

（2）（i＋k－2）/k 取整

（3）（i－1）k＋m＋1

（4）（i－1）%k 不等于 0

2．第 107 页，第（4）题

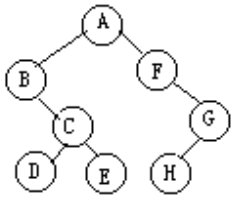
设对一棵二叉树进行中序遍历和后序遍历的结果分别为：

（1）中序遍历 B D C E A F H G

（2）后序遍历 D E C B H G F A

画出该二叉树。

答：



3．第 107 页，第（6）题的第 6 小题

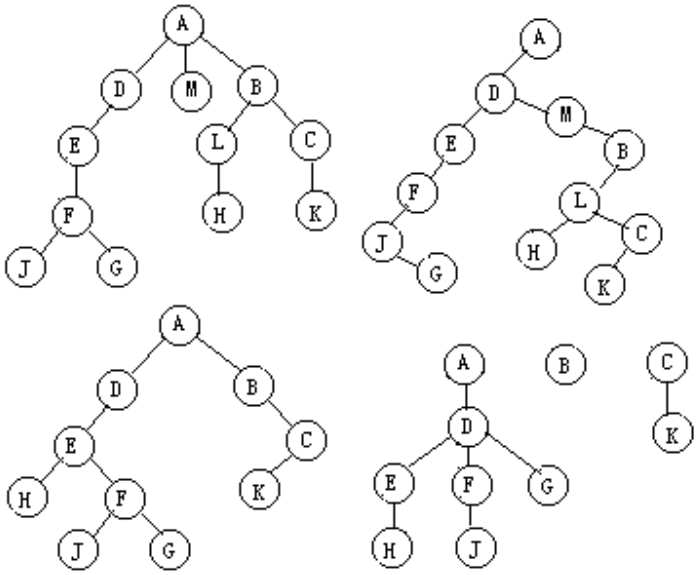
设计算法，交换一棵二叉树中每个结点的左、右子树。

```
template <class T>
void BTree<T>::Exch(BTNode<T> *p)
```

```
    {
        if (p){
            BTNode<T> *q=p->lchild;
            p->lchild=p->rchild;
            p->rchild=q;
            Exch(p->lchild);
            Exch(p->rchild);
        }
    }
```

4. 第 107 页，第 10 题

将图题 6-24 中的树转换成二叉树，并将图 6-23 中的二叉树转换成森林。



5. 第 107 页，第 11 题

给出对图 6-24 中的树的先序遍历和后序遍历的结果。

答：先序：A，D，E，F，J，G，M，B，L，H，C，K

后序：J，G，F，E，D，M，H，L，K，C，B，A

6. 第 107 页，第 12 题

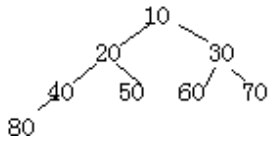
分别以下列数据为输入，构造最小堆。

(1) 10，20，30，40，50，60，70，80

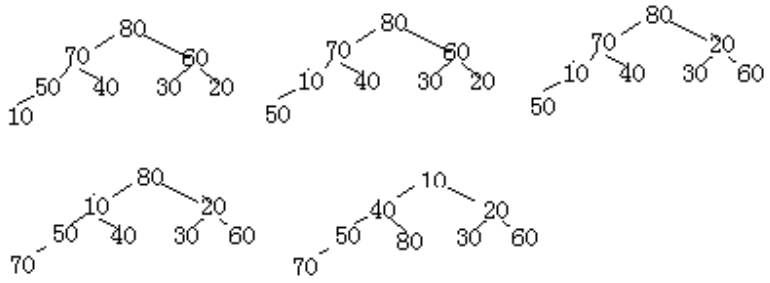
(2) 80，70，60，50，40，30，20，10

(3) 80，10，70，20，60，30，50，40

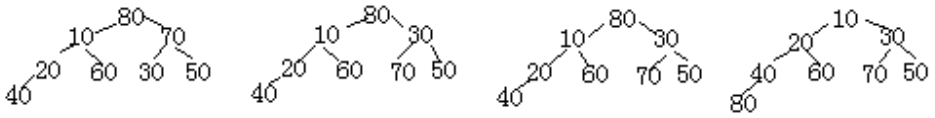
（1）



（2）

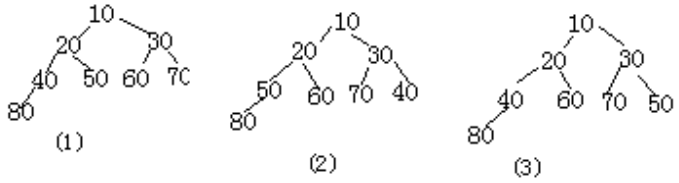


（3）



7. 第 107 页，第 13 题

分别以上题的数据为输入，从空的优先权队列开始，依此插入这些元素，求结果优先权队列的状态。



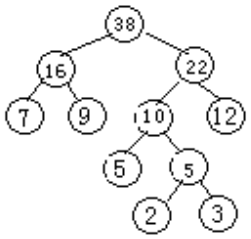
8. 第 108 页，第 14 题第（1）、（2）小题

设 S={A,B,C,D,E,F}，W={2,3,5,7,9,12}，对字符集合进行哈夫曼编码，W 为各字符的频率。

（1）画出哈夫曼树

（2）计算带权路径长度

S={A,B,C,D,E,F}，W={2,3,5,7,9,12}，

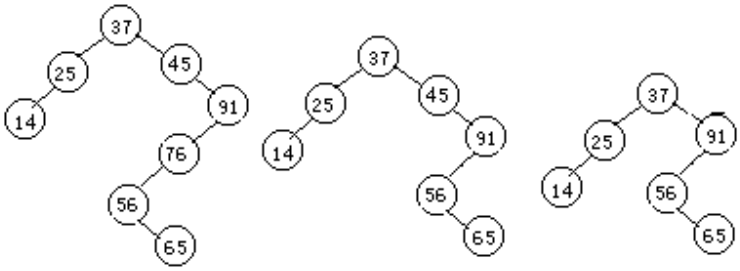


带权路径长度为：91

第七章 集合与搜索树

1．第 137 页，第（5），

建立 37，45，91，25，14，76，56，65 为输入时的二叉搜索树，再从该树上依此删除 76，45，则树形分别如何？



2. 第137页，第（6）

试写一个判定任意给定的二叉树是否二叉搜索树算法。

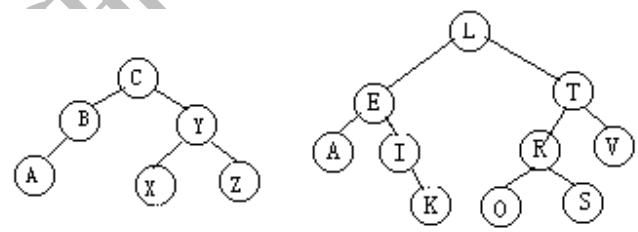
```
int k=-∞; bool fail=false;
template <class T>
void BTree<T>::IsBiTree(BTNode<T> *p,int &k,bool &fail)
{
    if (p&& !fail){
       IsBiTree(p->lchild,k,fail);
       if(k< p->element) k=p->element;
       else  fail=true;
      IsBiTree(p->rchild,k,fail);
    }
}
```

3. 第137页，第（8）

以下列序列为输入，从空树开始构造AVL搜索树。

(1)A，Z，B，Y，C，X

(2)A,V,L,T,R,E,I,S,O,K



4. 第137页，第（12）

5阶B-树的高度为2时，树中元素个数最少为多少？

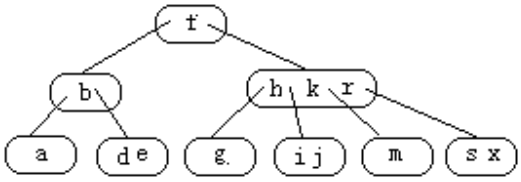
答： 5

5. 第137页，第（13）题

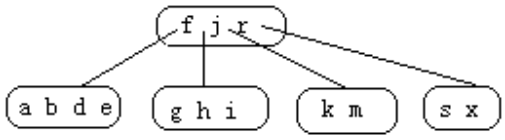
从空树开始，以关键字序列：a,g,f,b,k,d,h,m,j,e,s,i,r,x ,建立

(1) 4阶B-树;

(2) 5阶B-树。



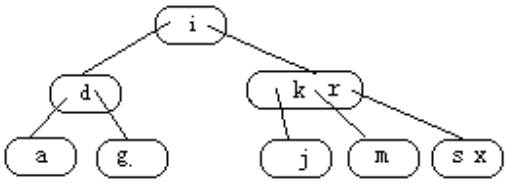
(1) 4阶B-树



(2) 5阶B-树

6. 第137页，第（14）题

从上题的4阶B-树上依次删除a,e,f,h。



第八章 散列与跳表

1. 第154页，第（3）题

设散列表ht[11]，散列函数h(key)=key % 11。采用线性探查法解决冲突，试用关键字值序列：70，25，80，35，60，45，50，55 建立散列表。

Key	70	25	80	35	60	45	50	55
h(Key)	4	3	3	2	5	1	6	0

0	1	2	3	4	5	6	7	8	9	10
55	45	35	25	70	80	60	50			

2. 第154页，第（6）题

给出用拉链方法解决冲突的散列表搜索操作的C++函数实现。

```
template<class T>
bool LinkedHashTable<T >::Search(const K &k,E&e)const
{
     int i=k % M,j;
```

```
        HNode<T>* p=ht[i];//元素结点类型 Hnode<T>
        while (p){
          if(p->element==k)return true;
          p=p->nextsynonym;
        }
        return false;
    }
```

3. 第154页，第（3）题

设散列表ht[11]，散列函数h(key)=key % 11。采用二次探查法解决冲突，试用关键字值序列：70，25，80，35，60，45，50，55 建立散列表。

Key	70	25	80	35	60	45	50	55
h(Key)	4	3	3	2	5	1	6	0

0	1	2	3	4	5	6	7	8	9	10
45	35	80	25	70	60					50

4. 第154页，第（4）题

对上题的例子，若采用双散列法，试以散列函数h_1(key)=key % 11，h_2(key)= key % 9+1 建立散列表。

Key	70	25	80	35	60	45	50	55
h1(key)	4	3	3	2	5	1	6	0
h2(key)	8	8	9	9	7	1	6	2

0	1	2	3	4	5	6	7	8	9	10
55	80	35	25	70	60	45	50			

第九章 图

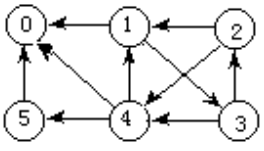
1. 第188页，第（1）题

对下面的有向图求

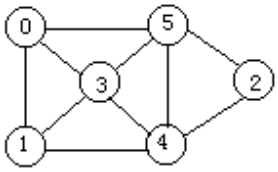
(1) 每个顶点的入度和出度；

(2) 强连通分量

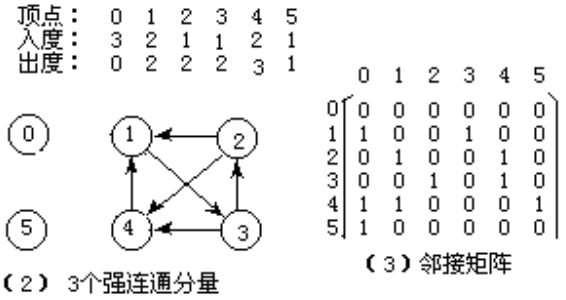
(3) 邻接矩阵



图题 9-1

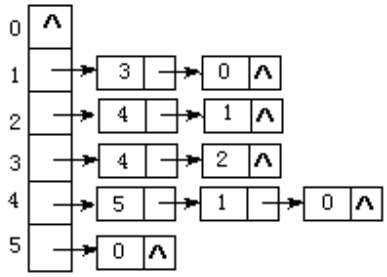


图题 9-2



2. 第188页，第（2）题

当以边〈1，0〉，〈1，3〉，〈2，1〉，〈2，4〉，〈3，2〉，〈3，4〉，〈4，0〉，〈4，1〉，〈4，5〉，〈5，0〉的次序从只有 6 个顶点没有边的图开始，通过依此插入这些边，建立邻接表结构。画出该邻接表。



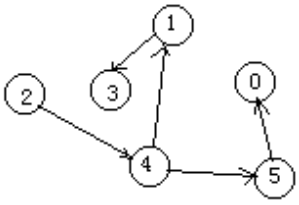
3. 第188页，第（4）题

已知有向图的邻接表，试写出计算各顶点的入度的算法。

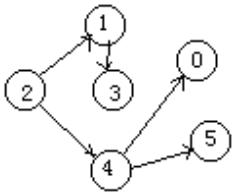
```
template<class T>
void LinkedGraph<T >::Degree()
{
     int *d=new int[n];int i;
     ENode<T>* p;
     for ( i=0;i<n;i++) d[i]=0;
     for ( i=0;i<n;i++){
      p=a[i];
      while (p){
        d[p->adjvex]++;
        p=p->nextarc;
      }
     }
   for ( i=0;i<n;i++) cout<<"d["<<i<<"]="<<d[i];
}
```

4. 第188页，第（6）题

在题 2 所建立的邻接表上进行以顶点 2 为起始顶点的深度优先搜索和广度优先搜索。分别画出遍历所得到的深度优先搜索和广度优先搜索的生成森林（或生成树）。



(a)深度优先搜索



(b)广度优先搜索

5. 第188页，第（8）题

分别设计算法，在邻接矩阵上实现有向图的深度优先搜索.

```
template<class T>
void MGraph<T >::DFS(int v,bool visited[])
{

    visited[v]=true;
    cout<<" "<<v;
    for ( int j=0;j<n;j++)
      if (a[v][j]&&!visited[j])
         DFS(j,visited);
}
```

6. 第188页，第（10）题

设某项工程由图题 9-3 所示的工序组成。若各工序以流水方式进行（即串行进行）。

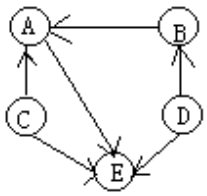
(1) 画出给工程的 AOV 网络；

(2) 给出该工程的全部合理的工程流程。

工序	紧前工序
A	B，C
B	D
C	——
D	——
E	A，C，D

注：紧前工序是指若有工序A和B，工序B必须在工序A完工后才能开工，则工序A是工序B的紧前工序。

图题 9-3



(a) AOV网络

C D B A E

D B C A E

D C B A E

(b) 工程流程

7. 第188页，第（11）题

设有边集合：〈0，1〉，〈1，2〉，〈4，1〉，〈4，5〉，〈5，3〉，〈2，3〉

(1) 求此图的所有可能的拓扑序列；

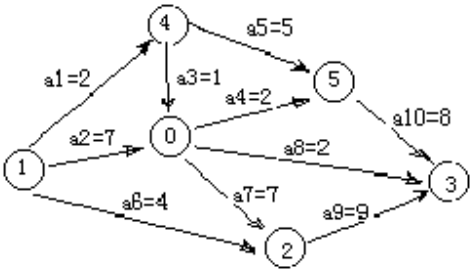
(2) 若以此顺序通过加入边的方式建立邻接表，则在该邻接表上执行拓扑排序算法（TopoSort），则得到的拓扑序列是其中哪一种？

0 4 1 2 5 3
0 4 1 5 2 3
0 4 5 1 2 3
4 0 1 2 5 3
4 0 1 5 2 3
4 0 5 1 2 3
4 5 0 1 2 3

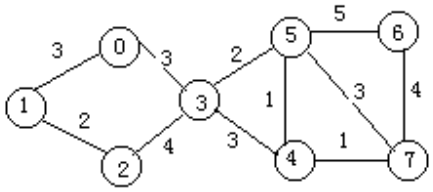
(a)所有可能的拓扑序列

4 5 0 1 2 3

(b)按邻接表的拓扑序



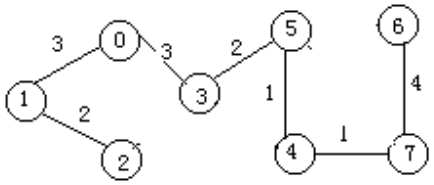
图题 9-4



图题 9-5

8. 第188页，第（13）题

使用普里姆算法以 1 为源点，画出图题 9-5 的无向图的最小代价生成树。



图题 9-5 的最小代价生成树

9. 第188页，第（16）题

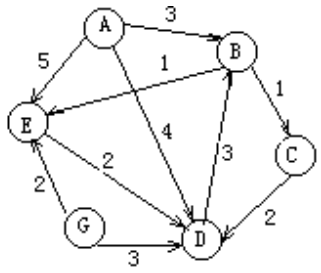
用迪杰斯特拉算法求图题9-6的有向图中以顶点A为源点的单源最短路径。写出一维数组d和 path在执行该算法的过程中各步的状态。

源点	终点	最短路径	路径长度
A	B	(A,B)	3
	C	(A,B,C)	4
	D	(A,D)	4
	E	(A,B,E)	4
	G	__	∞

S	d[A] path[A]	d[B] path[B]	d[C] path[C]	d[D] path[D]	d[E] path[E]	d[G] path[G]
A	0,-1	3,A	∞,-1	4,A	5,A	∞,-1
B	0,-1	3,A	4,B	4,A	4,B	∞,-1
C	0,-1	3,A	4,B	4,A	4,B	∞,-1
D	0,-1	3,A	4,B	4,A	4,B	∞,-1
E	0,-1	3,A	4,B	4,A	4,B	∞,-1

10. 第188页，第（17）题

用弗洛伊德算法求图题9-6的有向图的所有顶点之间的最短路径。写出二维数组d和path在执行该算法的过程中各步的状态。



图题 9-6

d

0	3	∞	4	5	∞
∞	0	1	∞	1	∞
∞	∞	0	2	∞	∞
∞	3	∞	0	∞	∞
∞	∞	∞	2	0	∞
∞	∞	∞	3	2	0

path

-1	A	-1	A	A	-1
-1	-1	B	-1	B	-1
-1	-1	-1	C	-1	-1
-1	D	-1	-1	-1	-1
-1	-1	-1	E	-1	-1
-1	-1	-1	G	G	-1

（1）初始状态

dA

0	3	∞	5	4	∞
∞	0	1	∞	1	∞
∞	∞	0	2	∞	∞
∞	3	∞	0	∞	∞
∞	∞	∞	2	0	∞
∞	∞	∞	3	2	0

pathA

-1	A	-1	A	A	-1
-1	-1	B	-1	B	-1
-1	-1	-1	C	-1	-1
-1	D	-1	-1	-1	-1
-1	-1	-1	E	-1	-1
-1	-1	-1	G	G	-1

dB

0	3	4	4	4	∞
∞	0	1	∞	1	∞
∞	∞	0	2	∞	∞
∞	3	4	0	4	∞
∞	∞	∞	2	0	∞
∞	∞	∞	3	2	0

pathB

-1	A	B	A	B	-1
-1	-1	B	-1	B	-1
-1	-1	-1	C	-1	-1
-1	D	B	-1	B	-1
-1	-1	-1	E	-1	-1
-1	-1	-1	G	G	-1

dC

0	3	4	4	4	∞
∞	0	1	3	1	∞
∞	∞	0	2	∞	∞
∞	3	4	0	4	∞
∞	∞	∞	2	0	∞
∞	∞	∞	3	2	0

pathC

-1	A	B	A	B	-1
-1	-1	B	C	B	-1
-1	-1	-1	C	-1	-1
-1	D	B	-1	B	-1
-1	-1	-1	E	-1	-1
-1	-1	-1	G	G	-1

dD

0	3	4	4	4	∞
∞	0	1	3	1	∞
∞	5	0	2	6	∞
∞	3	4	0	4	∞
∞	5	6	2	0	∞
∞	6	7	3	2	0

pathD

-1	A	B	A	B	-1
-1	-1	B	C	B	-1
-1	D	-1	C	B	-1
-1	D	B	-1	B	-1
-1	D	B	E	-1	-1
-1	D	B	G	G	-1

dE dG

0	3	4	4	4	∞
∞	0	1	3	1	∞
∞	5	0	2	6	∞
∞	3	4	0	4	∞
∞	5	6	2	0	∞
∞	6	7	3	2	0

pathE pathG

-1	A	B	A	B	-1
-1	-1	B	C	B	-1
-1	D	-1	C	B	-1
-1	D	B	-1	B	-1
-1	D	B	E	-1	-1
-1	D	B	G	G	-1

21. 第200页，第（1）题

元素序列(61,87,12,03,08,70,97,75,53,26)按下列算法排序，给出各趟排序结果。

(1) 简单选择排序；

(2) 冒泡排序；

(3) 直接插入排序；

(4) 快速排序；

(5) 两路合并排序；

(1) 简单选择排序

初始序列（61 87 12 03 08 70 97 75 53 26

第 1 趟 （03）87 12 61 08 70 97 75 53 26

第 2 趟 （03 08）12 61 87 70 97 75 53 26

第 3 趟 （03 08 12）61 87 70 97 75 53 26

第 4 趟 （03 08 12 26）87 70 97 75 53 61

第5趟 （03 08 12 26 53）70 97 75 87 61
第6趟 （03 08 12 26 53 61）97 75 87 70
第7趟 （03 08 12 26 53 61 70）75 87 97
第8趟 （03 08 12 26 53 61 70 75）87 97
第9趟 （03 08 12 26 53 61 70 75 87）97

（2）冒泡排序

初始序列（61 87 12 03 08 70 97 75 53 26
第1趟 61 12 03 08 70 87 75 53 26 97
第2趟 12 03 08 61 70 75 53 26 87 97
第3趟 03 08 12 61 70 53 26 75 87 97
第4趟 03 08 12 61 53 26 70 75 87 97
第5趟 03 08 12 53 26 61 70 75 87 97
第6趟 03 08 12 26 53 61 70 75 87 97
第7趟 03 08 12 26 53 61 70 75 87 97
第8趟 03 08 12 26 53 61 70 75 87 97
第9趟 03 08 12 26 53 61 70 75 87 97

（3）直接插入排序

初始序列（61）87 12 03 08 70 97 75 53 26
第1趟 （61 87）12 03 08 70 97 75 53 26
第2趟 （12 61 87）03 08 70 97 75 53 26
第3趟 （03 12 61 87）08 70 97 75 53 26
第4趟 （03 08 12 61 87）70 97 75 53 26
第5趟 （03 08 12 61 70 87） 97 75 53 26
第6趟 （03 08 12 61 70 87 97）75 53 26
第7趟 （03 08 12 61 70 75 87 97）53 26
第8趟 （03 08 12 53 61 70 75 87 97）26
第9趟 （03 08 12 26 53 61 70 75 87 97）

22. 第200页，第（3）题

在带表头结点的单链表上实现简单选择排序。

```
template <class T>
void SingleList<T>::select_sort()
{ Node<T> *p,*r,*small;
  p=first->link;
  if(p)
    for(; p->link; p=p->link)
    { for(small=p,r=p->link; r; r=r->link)
        if(small->data>r->data) small=r;
      if(small!=p) swap(p->data, small->data);
    }
};
```

直接插入排序：

```
template <class T>        //用带表头节点的单链表存储
void SingleList<T>::DirInsert()
```

```
{  Node<T> *head,*q,*p1,*p2;
   if(first->link)
   { head=first->link->link;  //head 是待插入元素的链表头
     first->link->link=NULL;  //first 是只有一个节点的有序链表
     while(head)
     {  q=head;       head=head->link;//从待插入链中取下一个节点
        p1=first;     p2=first->link; //p1 是 p2 的直接前驱节点
        while(p2&&p2->data<q->data)  //从前往后找插入位置
          { p1=p2;   p2=p2->link; }
        if(p2)  //q 插入在 p1 和 p2 之间
          { q->link=p2;      p1->link=q; }
        else    //q 插入在 p2 之后，即有序链的最后
          { q->link=NULL;   p1->link=q; }
     }
   }
}
```

10．4

```
template <class T>
void sort(T a[],int n)
{  int i,j,k=0,m=-1,s,t;
   T x;
   while (k<m)
   { s=k;
     for (i=k;i<m;i++)  //从上向下沉底
      if (a[i]>a[i+1])
     { x=a[i]; a[i]=a[i+1]; a[i+1]=x; s=i; }
    m=s;t=m;
    for (j=m;j>k;j--) //从下向上冒泡
      if (a[j]<a[j-1])
      { x=a[j]; a[j]=a[j-1]; a[j-1]=x; t=j; }
    k=t;
   }
}
```

10．6

证明：快速排序算法的基本思想时，每次把一个集合划分成大于和小于某个基准值的两个子集合。最坏情况是两个子集合中总有一个是空集合，即将一个集合分成了一个子集合和一个作为基准值的元素。即最坏情况下划分的自己和比原集合的元素只少一个。故要划分 n-1 次才能使得子集合中的元素少于 2，而每次划分子集合都要扫描整个集合，所以时间复杂度是 $O(n^2)$。

10．7

```
template <class T>  //方法一
void sort(T a[],int n)
{
  int i=0,j=n-1;
  T x=a[0];
  while (i<j)
  {
```

```
    while (a[j]>=0&&j>i) j--; //找负数
    if (j>i) a[i++]=a[j];
    while (a[i]<0&&i<j) i++;  //找正数    if (i<j) a[j--]=a[i];
   }
  a[j]=x;
}
```

10．8

```
template <class T>
void Merge(T A[],T Temp[],int i1,int j1,int i2,int j2,int &k)
{ if (i1>j1&&i2>j2) cout << "wrong ! \n";
  else if (i1>j1) while (i2<=j2) Temp[k++]=A[i2++];
     else if (i2>j2) while (i1<=j1) Temp[k++]=A[i1++];
      else if (A[i1]<A[i2])
        { Temp[k++]=A[i1];
          Merge(A,Temp,i1+1,j1,i2,j2,k);
        }
        else
         {  Temp[k++]=A[i2];
            Merge(A,Temp,i1,j1,i2+1,j2,k);
         }
}

template <class T>  //p.196
void MergeSort(T A[],int n)
{  T *Temp=new T [n];
   int i1,i2,j1,j2,i,k,size=1;
   while (size<n)
   {    i1=0; k=0;
    while (i1+size<n)
    { i2=i1+size; j1=i2-1;
      if (j1+size>n-1) j2=n-1; else j2=j1+size;
      Merge(A,Temp,i1,j1,i2,j2,k);
      i1=j2+1;
    }
    for (i=0;i<n;i++) A[i]=Temp[i];
    size*=2;
   }
   delete [] Temp;
}
```