**第一章**

**1、使用STL算法sort()实现整型数组*a*的递增排序**

#include<iostream>

#include <algorithm>

using namespace std;

int main()

{ int a[]={2,5,4,1,3};

sort(a,a+5);

for (int i=0;i<5;i++)

printf("%d ",a[i]); //输出: 1 2 3 4 5

printf("\n");

}

**2、【问题描述】编写一个实验程序，对于一个含n(n>1)个元素的queue<int>队列容器qu，出队从队头到队尾的第k(1<=k<=n)个元素，其他队列元素不变。**

#include<iostream>

#include<queue>

using namespace std;

char solve(queue<char> &qu,int k)

{

queue<char> temp;

char e;

for(int i=0;i<k-1;i++)

{

temp.push(qu.front());

qu.pop();

}

e=qu.front();

qu.pop();

while(!qu.empty())

{

temp.push(qu.front());

qu.pop();

}

qu=temp;

return e;

}

int main()

{

queue<char> qu;

qu.push('a');

qu.push('b');

qu.push('c');

qu.push('d');

int k=3;

char e=solve(qu,k);

cout<<"出队元素是"<<e<<endl;

cout<<"出队顺序是:";

while(!qu.empty())

{

cout<<qu.front()<<" ";

qu.pop();

}

cout<<endl;

return 0;

}

**第二章**

**1、递归实现简单选择排序**

**#include<iostream>**

**using namespace std;**

**void SelectSort(int a[], int n, int i)**

**{ int j, k;**

**if (i==n-1) return; //满足递归出口条件**

**else**

**{**

**k=i; //k记录a[i..n-1]中最小元素的下标**

**for (j=i+1;j<n;j++) //在a[i..n-1]中找最小元素**

**if (a[j]<a[k])**

**k=j;**

**if (k!=i) //若最小元素不是a[i]**

**swap(a[i],a[k]); //a[i]和a[k]交换**

**SelectSort(a,n,i+1);**

**}**

**}**

**int main()**

**{**

**int a[10]={5,4,6,2,1,0,3,8,7,9};**

**SelectSort(a,10,0);**

**for(int i=0;i<10;i++)**

**cout<<a[i]<<' ';**

**}**

**2、递归实现冒泡排序**

**#include<iostream>**

**using namespace std;**

**void BubbleSort(int a[], int n,int i)**

**{ int j;**

**bool exchange;**

**if (i==n-1) return; //满足递归出口条件**

**else**

**{**

**exchange=false; //置exchange为false**

**for(j=n-1;j>i;j--)**

**if(a[j]<a[j-1]) //当相邻元素反序时**

**{**

**swap(a[j],a[j-1]);**

**exchange=true; //发生交换置exchange为true**

**}**

**if(exchange==false) //未发生交换时直接返回**

**return ;**

**else //发生交换时继续递归调用**

**BubbleSort(a,n,i+1);**

**}**

**}**

**int main()**

**{**

**int a[10]={5,4,6,2,1,0,3,8,7,9};**

**BubbleSort(a,10,0);**

**for(int i=0;i<10;i++)**

**cout<<a[i]<<' ';**

**}**

**3、 应用递归算法求解逆置单链表问题**

**【问题描述】对于不带头结点的单链表L，设计一个递归算法逆置所有结点。编写完整的实验程序，并采用相应数据进行测试。**

#include<iostream>

#include<list>

#include<malloc.h>

using namespace std;

typedef int ElemType;

typedef struct Node

{ ElemType data;

struct Node \*next;

} LinkNode;

void CreateList(LinkNode \*&L,ElemType a[],int n) //由a[0..n-1]创建单链表L

{

LinkNode \*p, \*r;

L=(LinkNode \*)malloc(sizeof(LinkNode));

L->data=a[0];

r=L; //r指向当前尾结点

for (int i=1;i<n;i++)

{

p=(LinkNode \*)malloc(sizeof(LinkNode));

p->data=a[i];

r->next=p;

r=p;

}

r->next=NULL; //尾结点next域置为空

}

void DispList(LinkNode \*L) //输出单链表L

{

LinkNode \*p=L;

while (p!=NULL)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

}

LinkNode \*Reverse(LinkNode \*L) //逆置不带头结点的单链表L

{ LinkNode \*p;

if (L==NULL || L->next==NULL)

return L;

p=Reverse(L->next);

L->next->next=L; //将L结点链接到L->next结点后面

L->next=NULL; //将L结点作为整个逆置后的尾结点

return p;

}

int main()

{

ElemType a[]={1,2,3,4,5,6};

int n=sizeof(a)/sizeof(a[0]);

LinkNode \*L;

CreateList(L,a,n);

cout<<"实验结果:"<<endl;

cout<<" 逆置前L: "<<endl;

DispList(L);

cout<<" 执行L=Reverse(L)"<<endl;

L=Reverse(L);

cout<<" 逆置后L: "<<endl;

DispList(L);

return 0;

}

**第三章**

**1、分治法进行快排**

**#include<iostream>**

**using namespace std;**

**int Partition(int a[],int s,int t) //划分算法**

**{ int i=s,j=t;**

**int temp=a[s]; //用序列的第1个记录作为基准**

**while(i!=j)**

**{**

**while(j>i&&a[j]>=temp)**

**j--; //从右向左扫描，找第1个关键字小于tmp的a[j]**

**a[i]=a[j]; //将a[j]前移到a[i]的位置**

**while(i<j&&a[i]<temp)**

**i++;//从左向右扫描，找第1个关键字大于tmp的a[i]**

**a[j]=a[i]; //将a[i]后移到a[j]的位置**

**}**

**a[i]=temp;**

**return i;**

**}**

**void QuickSort(int a[],int s,int t)**

**//对a[s..t]元素序列进行递增排序**

**{ if (s<t) //序列内至少存在2个元素的情况**

**{ int i=Partition(a,s,t);**

**QuickSort(a,s,i-1); //对左子序列递归排序**

**QuickSort(a,i+1,t); //对右子序列递归排序**

**}**

**}**

**int main()**

**{**

**int a[10]={5,4,6,2,10,0,3,8,7,9};**

**QuickSort(a,0,10);**

**for(int i=0;i<10;i++)**

**{**

**cout<<a[i]<<" ";**

**}**

**cout<<endl;**

**return 0;**

**}**

**2、分治法进行归并排序**

#include<iostream>

#include<algorithm>

#include<malloc.h>

using namespace std;

void Merge(int a[],int low,int mid,int high)

{

int \*tmpa;

int i=low,j=mid+1,k=0;

tmpa=(int \*)malloc((high-low+1)\*sizeof(int));

while(i<=mid&&j<=high)

if(a[i]<=a[j]) //将第1子表中的元素放入tmpa中

{

tmpa[k]=a[i]; i++; k++;

}

else //将第2子表中的元素放入tmpa中

{

tmpa[k]=a[j];j++;k++;

}

while (i<=mid) //将第1子表余下部分复制到tmpa

{ tmpa[k]=a[i]; i++; k++; }

while (j<=high) //将第2子表余下部分复制到tmpa

{ tmpa[k]=a[j]; j++; k++; }

for(k=0,i=low;i<=high;k++,i++) //将tmpa复制回a中

a[i]=tmpa[k];

free(tmpa); //释放tmpa所占内存空间

}

//一趟二路归并排序

void MergePass(int a[],int length,int n)

{

int i;

for(i=0;i+2\*length-1<n;i=i+2\*length) //归并length长的两相邻子表

Merge(a,i,i+length-1,i+2\*length-1);

if(i+length-1<n) //余下两个子表，后者长度小于length

Merge(a,i,i+length-1,n-1); //归并这两个子表

}

void MergeSort(int a[],int n) //二路归并算法

{ int length;

for (length=1;length<n;length=2\*length)

MergePass(a,length,n);

}

int main()

{

int a[]={2,5,1,7,10,6,9,4,3,8};

MergeSort(a,10);

for(int i=0;i<10;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

return 0;

}

**3、分治法进行折半查找**

#include<iostream>

#include<algorithm>

using namespace std;

int BinSearch(int a[],int low,int high,int k)

//拆半查找算法

{

int mid;

if(low<high)

{

mid=(low+high)/2;

if(a[mid]==k)

return mid;

if(a[mid]>k)

return BinSearch(a,low,mid-1,k);

else

return BinSearch(a,mid+1,high,k);

}

return -1;

}

int main()

{

int i;

int k=2;

int a[]={1,2,3,4,5,6,7,8,9,10};

i=BinSearch(a,0,9,k);

if(i!=-1)

{

cout<<"找到,位置是："<<i<<endl;

}

else cout<<"未找到"<<endl;

return 0;

}

**4、应用递归和分治法求解众数问题**

**【问题描述】给定含有n个元素的多重集合S，每个元素在S中出现的次数称为该元素的重数。多重集S中重数最大的元素称为众数。 例如，S={1，2，2，2，3，5}。  多重集S的众数是2，其重数为3。  对于给定的由n 个自然数组成的多重集S，编程计算S 的众数及其重数。**

**#include <iostream>**

**#include<algorithm>**

**using namespace std;**

**#define M 100**

**int a[M];**

**int num,val,n; //重数, 众数,个数**

**void find(int &l,int &r,int mid)//找中位数的最左，最右边界位置**

**{**

**l = r= mid;**

**while(a[l]==a[mid] && l>= 0) --l;**

**l++; //还原**

**while(a[r]==a[mid] && r<= n-1) ++r;**

**r--;**

**}**

**void Fun(int low,int high)**

**{**

**if(low > high) return; //左右边界交叉，结束**

**int mid = (low + high)/2; //中位数**

**int i,j;**

**find(i,j,mid);**

**if(j-i+1 > num){ //更新**

**num = j-i+1;**

**val = a[mid];**

**}**

**if(i-low > num){//分治递归**

**Fun(low,i-1);**

**}**

**if(high -j > num){**

**Fun(j+1,high);**

**}**

**}**

**int main()**

**{**

**int i;**

**cout<<"输入元素个数：\n";**

**cin>>n;**

**cout<<"输入元素：\n";**

**for(i=0;i<n;i++)**

**cin>>a[i];**

**sort(a,a+n);**

**for(i=0;i<n;i++)**

**cout<<a[i]<<",";**

**Fun(0,n-1);**

**cout<<endl<<"众数："<<val<<" 重数："<<num;**

**return 0;**

**}**

**第四章**

**1、蛮力法解决简单选择排序**

**#include<iostream>**

**using namespace std;**

**void SelectSort(int a[],int n)**

**//对a[0..n-1]元素进行递增简单选择排序**

**{ int i,j,k;**

**for (i=0;i<n-1;i++) //进行n-1趟排序**

**{ k=i; //用k记录每趟无序区中最小元素的位置**

**for (j=i+1;j<n;j++) //在a[i+1..n-1]中穷举找最小元素a[k]**

**if (a[j]<a[k])**

**k=j;**

**if (k!=i) //若a[k]不是最小元素,将a[k]与a[i]交换**

**swap(a[i],a[k]);**

**}**

**}**

**int main()**

**{**

**int a[]={5,9,4,2,1};**

**SelectSort(a,5);**

**for(int i=0;i<5;i++)**

**{**

**cout<<a[i]<<" ";**

**}**

**}**

**2、蛮力法解决冒泡排序**

**#include<iostream>**

**using namespace std;**

**void BubbleSort(int a[],int n)**

**//对a[0..n-1]按递增有序进行冒泡排序**

**{ int i,j; int tmp;**

**bool exchange;**

**for (i=0;i<n-1;i++) //进行n-1趟排序**

**{ exchange=false; //本趟排序前置exchange为false**

**for (j=n-1;j>i;j--) //无序区元素比较,找出最小元素**

**if (a[j]<a[j-1]) //当相邻元素反序时**

**{ swap(a[j],a[j-1]); //a[j]与a[j-1]进行交换**

**exchange=true; //发生交换置exchange为true**

**}**

**if (exchange==false) //本趟未发生交换时结束算法**

**return;**

**}**

**}**

**int main()**

**{**

**int a[]={5,9,4,2,1};**

**BubbleSort(a,5);**

**for(int i=0;i<5;i++)**

**{**

**cout<<a[i]<<" ";**

**}**

**}**

**3、应用蛮力法求解钱币兑换问题**

**【问题描述】某个国家仅有1分、2分和5分硬币，将钱n(n>=5)兑换成硬币有很多种兑法。编写一个实验程序计算出10分钱有多少种兑法，并列出每种兑换方式。**

#include<iostream>

using namespace std;

int main()

{

int n=10;

int x,y,z;

int num=0;

for(z=0;z<=2;z++)

{

for(y=0;y<=5;y++)

{

for(x=0;x<=10;x++)

{

if(z\*5+y\*2+x\*1==10)

{

cout<<"兑换方式"<<++num;

if(z!=0) cout<<" 5分的硬币有："<<z<<"个";

if(y!=0) cout<<" 2分的硬币有："<<y<<"个";

if(x!=0) cout<<" 1分的硬币有："<<x<<"个";

cout<<endl;

}

}

}

}

cout<<"共有"<<num<<"种兑换方式"<<endl;

return 0;

}

**第五章**

**1、回溯法求解求解0/1背包问题**

**#include <iostream>**

**using namespace std;**

**#define MAXN 20**

**//问题表示**

**int n=4; //4种物品**

**int W=6; //限制重量为6**

**int w[]={0,5,3,2,1}; //存放4个物品重量,不用下标0元素**

**int v[]={0,4,4,3,1}; //存放4个物品价值,不用下标0元素**

**//求解结果表示**

**int x[MAXN]; //存放最终解**

**int maxv; //存放最优解的总价值**

**void dfs(int i,int tw,int tv,int rw,int op[])**

**{ if (i>n) //找到一个叶子结点**

**{ if (tw==W && tv>maxv) //找到一个满足条件的更优解,保存**

**{ maxv=tv;**

**for (int j=1;j<=n;j++)**

**x[j]=op[j];**

**}**

**}**

**else //尚未找完所有物品**

**{**

**if ( tw+w[i]<=W ) //左孩子结点剪枝**

**{ op[i]=1; //选取第i个物品**

**dfs(i+1,tw+w[i],tv+v[i],rw-w[i],op);**

**}**

**if ( tw+rw-w[i]>=W ) //右孩子结点剪枝**

**{ op[i]=0; //不选取第i个物品,回溯**

**dfs(i+1,tw,tv,rw-w[i],op);**

**}**

**}**

**}**

**int main()**

**{**

**int op[MAXN];**

**dfs(1,0,0,11,op);**

**cout<<"最优值是："<<maxv<<endl;**

**for(int j=1;j<=n;j++)**

**cout<<x[j]<<" ";**

**}**

**2、应用回溯法求解组合问题**

【问题描述】编写一个实验程序，采用回溯法输出自然数1~n中任取r个数的所有组合。

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**int n,r;**

**void disp(vector<int> path)**

**{**

**for(int j=0;j<path.size();j++)**

**cout<<path[j]<<" ";**

**cout<<endl;**

**}**

**void dfs(vector<int> path,int i,int num)**

**{**

**if(num==r)**

**disp(path);**

**for(int j=i;j<=n;j++)**

**{**

**path.push\_back(j);**

**dfs(path,j+1,num+1);**

**path.pop\_back();**

**}**

**}**

**int main()**

**{**

**cin>>n>>r;**

**vector<int> path;**

**dfs(path,1,0);**

**return 0;**

**}**

**第六章**

**1、应用分枝限界法求解n皇后问题**

**【问题描述】在n×n的方格棋盘上，放置n个皇后，要求每个皇后不同行、不同列、不同左右对角线。如图1所示是6皇后问题的一个解。要求采用队列式分枝限界法求解4皇后问题的一个解，并分析对应程序运行中创建的队列结点的个数。**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int n=4;

int Count=1;

struct NodeType

{

int no;

int row;

vector<int> cols;

};

void dispnode(NodeType e)

{

if(e.row!=-1)

cout<<"编号"<<e.no<<"对应位置是："<<e.row<<","<<e.cols[e.row]<<endl;

else

cout<<"编号"<<e.no<<"对应位置是："<<e.row<<","<<"\*"<<endl;

}

bool Valid(vector <int> cols,int i,int j)

{

int k=0;

while(k<i)

{

if((cols[k]==j)||(abs(cols[k]-j)==abs(k-i))) return false;

k++;

}

return true;

}

void solve()

{

int i,j;

NodeType e,el;

queue<NodeType> qu;

e.no=Count++;

e.row=-1;

qu.push(e);

cout<<"进队:";

dispnode(e);

while(!qu.empty())

{

e=qu.front();qu.pop();

cout<<"出队";

dispnode(e);

if(e.row==n-1)

{

cout<<"产生一个解:";

for(int i=0;i<n; i++)

{

cout<<i+1<<","<<e.cols[i]+1<<" ";

}

cout<<endl;

return ;

}

else

{

for(j=0;j<n;j++)

{

i=e.row+1;

if(Valid(e.cols,i,j))

{

el.no=Count++;

el.row=i;

el.cols=e.cols;

el.cols.push\_back(j);

qu.push(el);

cout<<"进队子结点:";

dispnode(el);

}

}

}

}

}

int main()

{

solve();

return 0;

}

**第七章**

**1、贪心算法求解活动安排问题**

#include <iostream>

#include<string.h>

#include<algorithm>

using namespace std;

#define MAX 51

//问题表示

struct Action //活动的类型声明

{ int b; //活动起始时间

int e; //活动结束时间

bool operator<(const Action &s) const //重载<关系函数

{

return e<=s.e; //用于按活动结束时间递增排序

}

};

int n=11;

Action A[]={{0},{1,4},{3,5},{0,6},{5,7},{3,8},{5,9},{6,10},{8,11},

{8,12},{2,13},{12,15}}; //下标0不用

//求解结果表示

bool flag[MAX]; //标记选择的活动

int Count=0;

void solve() //求解最大兼容活动子集

{ memset(flag,0,sizeof(flag)); //初始化为false

sort(A+1,A+n+1); //A[1..n]按活动结束时间递增排序

int preend=0; //前一个兼容活动的结束时间

for (int i=1;i<=n;i++) //扫描所有活动

{ if (A[i].b>=preend) //找到一个兼容活动

{ flag[i]=true; //选择A[i]活动

preend=A[i].e; //更新preend值

}

}

}

int main()

{

solve();

cout<<"求解结果,选取的活动:"<<endl;

for (int i=1;i<=n;i++) //求v/w

if(flag[i])

{

cout<<A[i].b<<","<<A[i].e<<endl;

Count++;

}

cout<<"共有"<<Count<<"个活动";

return 0;

}

**2、应用贪心法求解删数问题**

**【问题描述】编写一个实验程序实现求解删数问题。给定共有n位的正整数d，去掉其中任意,k<=n个数字后剩下的数字按原次序排列组成一个新的正整数。对于给定的n位正整数d和正整数k，找出剩下数字组成的新数最小的删数方案。**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main()**

**{**

**string N;**

**int k,i;**

**cin >> N >> k;**

**while (k--)**

**{**

**int len=N.length();**

**for (i=0;i<len-1;i++)**

**if (N[i]>N[i+1])**

**{**

**N.erase(N.begin()+i);**

**break;**

**}**

**if (i==len-1)**

**N.erase(N.end()-1); //删除最后数字**

**}**

**while(N[0]=='0'&&N.length()>1)**

**N.erase(N.begin());**

**cout << N << endl;**

**return 0;**

**}**

**第八章**

**1、动态规划求解最大连续子序列和问题**

#include<iostream>

using namespace std;

int a[]={0,-2,11,-4,13,-5,-2};

int n=6;

int dp[20];

void maxsubsum()

{

dp[0]=0;

for(int j=1;j<=n;j++)

dp[j]=max(dp[j-1]+a[j],a[j]);

}

void dispmaxsum()

{

int maxj=1;

for(int j=1;j<=n;j++)

if(dp[j]>dp[maxj]) maxj=j;

int k;

for(k=maxj;k>=1;k--)

if(dp[k]<=0) break;

cout<<dp[maxj]<<endl;

for(int i=k+1;i<=maxj;i++)

cout<<a[i]<<" ";

}

int main()

{

maxsubsum();

dispmaxsum();

return 0;

}

**2、应用动态规划算法求解矩阵最小路径和问题**

**【问题描述】给定一个m行n列的矩阵,从左上角开始每次只能向右或者向下移动,最后到达右下角的位置,路径上的所有数字累加起来作为这条路径的路径和。编写一个实验程序求所有路径和中的最小路径和。例如,以下矩阵中的路径1→3→1→0→6→1→0是所有路径中路径和最小的，返回结果是12：**

**1 3 5 9**

**8 1 3 4**

**5 0 6 1**

**8 8 4 0**

#include<iostream>

#include<vector>

#include<stdio.h>

using namespace std;

#define MAXM 100

#define MAXN 100

//问题表示

int a[MAXM][MAXN]={{1,3,5,9},{8,1,3,4},{5,0,6,1},{8,8,4,0}};

int m=4,n=4;

//求解结果表示

int ans; //最小路径长度

int dp[MAXM][MAXN];

int pre[MAXM][MAXN];

void Minpath() //求最小和路径ans

{ int i,j;

dp[0][0]=a[0][0];

for(i=1;i<m;i++) //计算第一列的值

{ dp[i][0]=dp[i-1][0]+a[i][0];

pre[i][0]=0; //垂直路径

}

for(j=1;j<n;j++) //计算第一行的值

{ dp[0][j]=dp[0][j-1]+a[0][j];

pre[0][j]=1; //水平路径

}

for(i=1;i<m;i++) //计算其他dp值

for(j=1;j<n;j++)

{ if (dp[i][j-1]<dp[i-1][j])

{ dp[i][j]=dp[i][j-1]+a[i][j];

pre[i][j]=1;

}

else

{ dp[i][j]=dp[i-1][j]+a[i][j];

pre[i][j]=0;

}

}

ans=dp[m-1][n-1];

}

void Disppath() //输出最小和路径

{ int i=m-1,j=n-1;

vector<int> path; //存放反向最小路径

vector<int>::reverse\_iterator it;

while (true)

{ path.push\_back(a[i][j]);

if (i==0 && j==0) break;

if (pre[i][j]==1) j--; //同行

else i--; //同列

}

printf(" 最短路径: ");

for (it=path.rbegin();it!=path.rend();++it)

printf("%d ",\*it); //反向输出构成正向路径

printf("\n 最短路径和:%d\n",ans);

}

int main()

{

Minpath(); //求最小路径和

printf("求解结果\n");

Disppath(); //输出求最小路径与最小路径和

return 0;

}