**贪婪钱币找零**

for(i=0；i<4；i++) {

b[i] = s/a[i]；

s- = b[i]\*a[i]；}；

若 s=57，则找钱方案为：

b[0]=2；b[1]=1；b[2]=1；b[3]=2

**哈密尔顿回路**

void main() {

int n; //顶点个数

int x[n]; //哈密尔顿图回路上的顶点编号

boolean c[n][n]; //图G的0/1邻接矩阵

hamilton(int n, int x[], boolean c[][])

printf(“哈密尔顿图回路为：”x[]);

void hamilton(int n, int x[], boolean c[][]) {

int i, k;

boolean \*s=new boolean[n] //定义数组s记录顶点状态，若顶点*i*在回路中，则s[*i*]=1，否则为0

for (i=0; i<n; i++) { x[i]=-1; s[i]=0; } //初始化

k=1; s[0]=1; x[0]=0; //从0号顶点开始搜索

while( k>=0 ) { //*k*用于控制对顶点的依次搜索

**x[k] = x[k] + 1 ;**  //设置顶点*k*的搜索起点

while( x[k]<n ) {

if (!s[x[k]] && c[x[k]][x[*k-1*]])==1 )

break; //搜索到一个顶点

else x[*k*]=x[*k*]+1; //否则，搜索下一个顶点编号

}

if( (x[k]<n) && (k!=n-1)) //搜索成功且*k*<*n*-1

{ s[x[k]] = 1; k = k + 1; } //向前推进一个顶点

else if ((x[k]<n)&&(k==n-1) && ( c[x[*k*]][x[0]]==1)) break; //是最后的顶点，完成搜索

else { x[k] = -1 ; k = k - 1; s[x[k]] = 0; } //搜索失败，回溯到前一个顶点

}

delete s; //删除数组s，搜索到的顶点序列存储在X[]中

}

**Dijkstra**

for (i=0; i<n; i++) { //初始化

Dist[i] = Max\_dist; s[i]=0; p[i]=-1;}

if (!(pnode=N[u].next) //如果源点u与其它顶点不相邻，则返回

return;

while(pnode) { // pnode指针指向了源点u的邻接表

Dist[pnode->v\_num] = pnode->length; //距离设置为边上的权

Pre[pnode->v\_num] = u; //前驱结点是u

pnode = pnode->next ;}

Dist[u]=0; s[u]=1; //顶点u加入s中

for (i=1; i<n; i++) { //计算到其余n-1个顶点的最短距离

temp = Max\_dist; t=u;

for (j=0; j<n; j++) { //在Dist[j]中找最小的距离，顶点记为t，顶点j不在s中

if (!s[*j*] && Dist[j]<temp) {

t=j; temp = Dist[j]; }

if (t==u) break; //找不到的情况下，跳出循环，说明存在顶点从顶点u不可达

s[t] = 1; //将顶点t并入集合s

pnode = N[t].next; //更新与t相邻接的所有顶点到u的距离

while (pnode) {

if (!s[pnode->v\_num] && Dist[pnode->v\_num] > Dist[t] + pnode->length) {

Dist[pnode->v\_num] = Dist[*t*] + pnode->length ;

Pre[pnode->v\_num] = t; //路径经过t后变得更短 }

pnode = pnode->next;

}

}delete s;

时O(n2) 空O(n)

**回溯法r组合问题**

#define MAXN 100

int a[MAXN]；

void comb\_back(int m，int r) { //求从*m*个数字中找*r*个数字的全部组合，找的一个，输出一个

int i，j；

i=0；a[i]=1；

do{

if( a[i]-i<=m-r+l ) { //若还可继续向前试探，则修改侯选组合

if(i==r-1 ) { //若已经找到一个组合，则输出数组a

for(j=0；j<r；j++) printf(“%4d”，a[j])；

printf(“﹨n”)；

a[i]++ ； //在当前解的基础，直接构成下一个组合

continue ； //跳出本次循环，进入下一次循环，继续判断所构成的候选解是否满足条件

}

i++； //继续向前试探

a[i]=a[i-1]+1； //保证满足条件a[*i*+1]>a[*i*]

}

else { //回溯

if(i==0) return； //已找完了全部

a[--i]++； //继续找下一个组合，

}

}while(1)；

}

void main() {

comb\_back(7，4)；

}

**分治策略求最大最小值**

int mid , x, y, u, w；

if((high – low)<=1) {

if (A[high] > A[low] {

amax = A[high];

amin = **A[low]** ;

}

else {

amax = **A[low]** ;

amin = A[high] ;

} }

else｛

mid = ( low + high ) /2 ;

maxmin(A, x , y, low , mid );

maxmin(A, u, w, mid+1 , high );

amax = max ( x , u );

amin = min ( y , w ); ｝

}// maxmin

**最长公共子序列**

for(i=0；i<=m；i++) c[*i*][0]=0 ； //最长公共子序列的长度初始化

for(j=1；j<=n；j++) c[0][j] = 0； //最长公共子序列的长度初始化

for(i＝1；i<=m；i++)

for(j＝1；j<=n；j++)

if(a[i-1]==b[j-l]) c[i][j] = c[i-1][j-1] +l；

else if(c[i-1][j] >= c[i][j-1])

c[i][j] = c[i-1][j]；

else c[i][j] = c[i][j-1] ；

return c[*m*][*n*] ； //返回最长公共子序列的长度

}

char \*build-lcs(char s[],char \*a,char \*b) { //构造最长公共子序列函数

int k，i=strlen(a)，j=strlen(b)，c[N][N]；

k = lcs\_len(a，b， c )；

s[k] = ‘\0’；

while(k>0)

if(c[i][j]==c[i-1][j]) i--；

else if(c[i][j]==c[i][j-1]) j-- ；

else { s[--k] = a[i-1]；

i--；j--；}

return s ；}