**回溯法**

**子集树**

**非递归**

void LoopBackTrace(data x[], int n)//x记录解矢量，n是解矢量的长度

{

int i = 1, t = 0;//i描述第i步

data a[];

while (i >0) {

设置第i步的子节点a[]

for (;(t<子节点数) && i<n+1;t++) {//t是第t个子节点

if (constrain(x, i, a[t])) {

设置第i步现场;

x[i]=a[t];

if (i == n) {

if (x[1~n]是一个解)

输出或记录;

}

i++;

设置第i步的子节点a[]

t = -1;

}

}

i--;

t=m[i];

恢复第i步的现场;

t++;

}

}

**递归**

void RecursSubsetBackTrace(data x[], int n, int i)

{

data a[];

if (i>n) {

if (x[1~n]是解) {

输出或保存;

}

} else {

设定子节点a[]

for (int t = 0; t <子节点数目; t++) {

if (constrain(x, i, a[t])) {

设置第i步现场;

x[i]=a[t];

RecursiveBackTrace(x, n, i+1);

恢复第i步现场;

}

}

}

}

**排列树**

初始化x[]的一个基本解

void RecursPermBackTrace(data x[], int n, int i)

{

if (i>n) {

if (x[1~n]是解) {

输出或保存;

}

} else {

for (int t = i; t <=n; t++) {

swap(x[i], x[t]);

if (constrain(x, i)) {

设置第i步现场;

RecursPermBackTrace(x, n, i+1);

恢复第i步现场;

}

swap(x[t], x[i]);

}

}

}

分支界限法

void BranchBound()

{

std::queue<Nodetype> nodequeue;//node.x[1],x[2],…,x[n]记录解矢量

产生根节点并插入队列;

while (!nodeque.empty()) {

取出队首节点

for(int t=0;t<子节点数;t++) {

产生子节点;

if (子节点满足约束和界限) {

if (子节点是非叶节点(层次<n))//n解矢量的长度

插入到队列

}

}

}

}