#### 递归算法4★

能不能再继续优化呢，答案是肯定的。

|  |  |
| --- | --- |
| **说明: D:\第一部出版本全部内容\封面和人物设计\6.png** | 我记得本套书的第一部中曾讲过一个猴子选大王的优化算法，大意是使用一个动态数组链表连接圈内的猴子，如果某只猴子出圈，即将这只猴子从链表中断开，不知道这种优化算法能不能用到N皇后问题上。 |

观察程序可以发现，在给第x行选择一个位置i时，需要与前面x－1个值进行判断，但实际上已经确定的x－1个值是无须判断的，因此需要有一个数据结构能动态地删除选过的值，又能在递归调用结束后恢复断点前的操作。不难想到可以用如图2.12中的链表来实现。

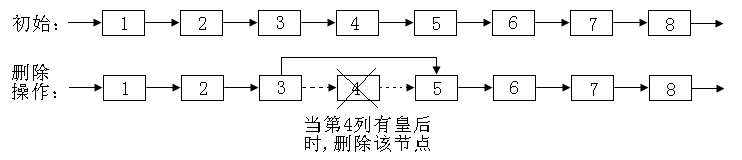


图2.12

具体实现使用基于数组的链表int next[]，其中0表示头结点，next[0]表示第一个可用位置，next[i]表示位置i的下一个可用位置。如果next[i]＝0，说明i是最后一个可用位置，这样减少了待选个数，也无须使用列判断数组y[]了。next数组初始化如表2.1所示。

表2.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标i | 0 | 1 | 2 | … | n－1 | n |
| next[i] | 1 | 2 | 3 | … | n | 0 |

为了方便进行删除操作，用一个变量pre记录i的前一个可用位置。一旦选择了i，就把i这个结点删掉。即执行next[pre]＝next[i]，在下一层调用完成后，要进行恢复操作，则执行next[pre]＝i，如图2.13所示。

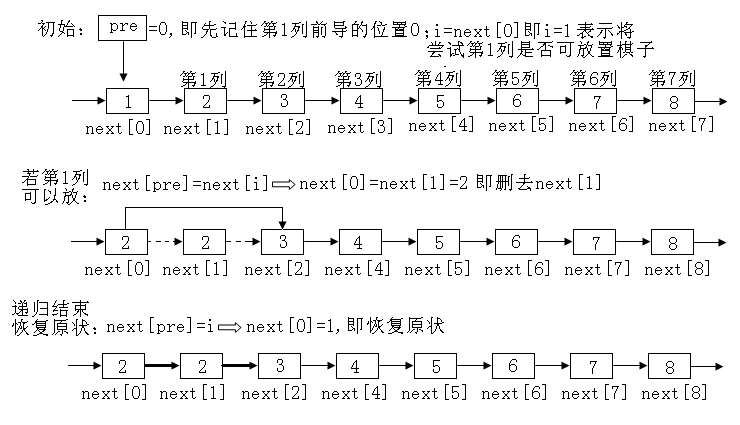


图2.13

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | //N皇后问题 — 优化三  #include <bits/stdc++.h>  using namespace std;  #define odd(num) (num&1) //采用判断最低二进制位的方法判断是否奇数  int a[20+1],n,total;  int x1[40+1];  int \_x2[100],\*x2; //将通过x2指向\_x2[]的方式模拟数组负下标  int Next[20+1]; //Next[]为数组链表  void place(int x,int y)  {  int i,pre;  if(x==n+1) //如果n个棋子已摆好，则方案数加1  total++;  else  {  i=Next[0]; //初始时i=1  pre=0;  while((i>0)&&(i<=y))  {  if((!x1[x+i])) //正斜线无冲突  if((!x2[x-i])) //反斜线无冲突  {  x1[x+i]=1;  x2[x-i]=1; //如x-i<0,则会出现负下标  Next[pre]=Next[i];  a[x]=i;  if((x==1) && odd(n) && (a[1]==(n+1)/2))//当第一行棋子在中间时  place(2,n/2-1);  else  place(x+1,n);  x1[x+i]=0; //恢复  x2[x-i]=0; //恢复  Next[pre]=i; //恢复  }  pre=i; //下一结点的前导  i=Next[i]; //下一结点位置  }  }  }  int main()  {  scanf("%d",&n);  x2=&\_x2[50]; //指针指向数组中间，模拟负下标,x2[i]中的i值可为负值  for(int i=0; i<=n-1; ++i) //链表初始化  Next[i]=i+1;  place(1,(n+1)/2);  printf("%d\n",total\*2);  return 0;  } |