【算法分析】

可以先看目录下的精简算法，那个简单。以下的讲解较复杂

经典的搜索题，最单纯的算法的时间复杂度为O（n!），是会严重超时的，在搜索过程中，例如出现1＋1＝3，这很明显是错误的，但计算机仍会继续搜索下去。

我们可以使用剪枝，每次搜索的时候，从最后向前判断是否有不合法的式子。这个剪枝简单且效果很好。在此基础上，我们还可以从右往左，按字母出现顺序搜索。

又如下式：

A\*\*\*?\*\*\*

＋ B\*?\*\*?\*\*

＝ C\*\*\*???\*

其中\*代表已知，？代表未知，那么，A＋B＝C要考虑到进位和不进位的两种情况，但如果（A＋B）%N与（A＋B＋1）%N都不等于C的话，那这等式一定不成立。

我们可以用一个数组记录一个数字是否被使用过，对于某一位A＋B＝C的等式，如果已得到两个数，另一个数还待搜索的时候，可以加入一个剪枝，即考虑不进位情况的可能值和进位情况的可能值，如果这两种可能的值均被使用过，那么这个搜索应被剪去。

我们还可以考虑其他的思路，例如枚举进位，解方程（可能需要枚举）等。

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  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149 | /\*  虫食算  可以采用一些比较高级的算法，比如解方程之类的，不过通常的算法是搜索，  这个程序也采用搜索的方法  搜索思路：  1.算式存储方式：A[i][j]表示第j个(从0开始计数)字符串从右往左数第i个  (从0开始计数)字母  如样例:ABCED 对应存储于这些元素中 A[4,3,2,1,0][0]  BDACE A[4,3,2,1,0][1]  EBBAA A[4,3,2,1,0][2]  如此存储，是因为搜索是一位一位进行枚举的，即每次确定 同一列的字母，  如第一次确定A[0][0],A[0][1],A[0][2]三个 字母  2.A[i][2]=(A[i][0]+A[i][1]+u)%n  对每一位枚举前两个字母可能对应的数字。如果在之前的搜索 中字母已经被  确定，那此次不需再枚举。若未被确定，则需要 选择一个数字。  当前两个字母确定下来时，第三个字母即得到确定，只需判断这个结果是否合法  (是否同一个字母对应两个数字，或者同一 个数字被多个字母使用) 其中u表示  第i－1位的进位，如果从低位向高位搜索(即从右向 左)，那么每一次的进位都  是确定的  3.剪枝:  （１）每个数字只能被一个字母使用，那么给每个数字做上标记， 避免被重复使用。  （２）如果最高位出现进位，则等式不成立  （３）如果第三个字母对应的数字已经确定，但是并不是前两个 字母计算后的结果，  那等式不成立  （４）如果前两个字母计算后的结果已经被使用，并且不是被第 三个字母使用，那  等式不成立  （５）如果第三个字母未被确定，那么这一位确定下来后，可能 会对后面的算式造  成影响，甚至直接造成矛盾，后者可以直接剪掉直接依次枚举之后的每一位  a.如果一个数字不确定，那可以通过另外两个数字计算出 这个数字，如果这个数  字被使用，那等式不成立  (注意进位，如果进位已经确定则直接计算判断即可，否则是否进位两种情况都不成立  等式才不成立)  b.如果三个数字都确定了，那需要计算一下该位等式是否 成立  \*/  #include <bits/stdc++.h>  using namespace std;  int n;  char in[3][30]; //存储读入的字符串  int A[30][3]; //存储算式  //标记每个数字是否被使用,1表示被使用，0表示未被使用  bool use[30];  //记录每个字母对应的数字，-1表示未确定  int num[30];  void Search(int,bool);  #define RETURN {num[x]=-1;use[c]=0;return;}//宏定义 还原+返回操作  //处理第p位第q个数字，前两个数字存储在a中，上一位的进位为u  void Search2(int p,int q,int a[2],bool u)  {  int x=A[p][q];  if(q==2)  {  if(p==n-1 && a[0]+a[1]+u>=n)  return; //剪枝2  int c=(a[0]+a[1]+u)%n,i,j;  if(num[x]!=-1 && num[x]!=c)  return; //剪枝3  if(use[c] && num[x]!=c)  return; //剪枝4  if(num[x]==-1)  {  num[x]=c;  use[c]=1;  int up=(a[0]+a[1]+u)/n;  for(i=p+1; i<n; i++) //剪枝5  {  int k1=num[A[i][0]],k2=num[A[i][1]],k3=num[A[i][2]];  if(k1==-1 || k2==-1 || k3==-1)//a)其中up为进位，若up==－1表示进位不确定  {  if(k1!=-1 && k2!=-1 && ((up==-1 && use[(k1+k2)%n] && use[(k1+k2+1)%n])||(up!=-1 && use[(k1+k2+up)%n])))  RETURN  else if(k1!=-1 && k3!=-1 && ((up==-1 && use[(k3+2\*n-k1)%n] && use[(k3+2\*n-1-k1)%n])||(up!=-1 && use[(k3+2\*n-up-k1)%n])))  RETURN  else if(k2!=-1 && k3!=-1 && ((up==-1 && use[(k3+2\*n-k2)%n] && use[(k3+2\*n-1-k2)%n])||(up!=-1 && use[(k3+2\*n-up-k2)%n])))  RETURN  else up=-1;//有两个以上数不确定的情况下，进位也不确定  }  else if((up==-1 && (k1+k2)%n!=k3 && (k1+k2+1)%n!=k3)||(up!=-1 && (k1+k2+up)%n!=k3))  RETURN  else if(i==n-1&&((up==-1&&(k1+k2)/n!=0&&(k1+k2+1)/n!=0)||(up!=-1&&(k1+k2+up)/n!=0)))//剪枝2  RETURN  else up=((k1+k2)>k3); //三个数都确定的情况下要计算进位  }  Search(p+1,(a[0]+a[1]+u)/n);//递归处理下一位  num[x]=-1;  use[c]=0;  }  else  Search(p+1,(a[0]+a[1]+u)/n);  return;  }  if(num[x]==-1) //如果字母未被确定  {  for(int i=n-1; i>=0; i--) //倒着取数，避免一些特殊数据  {  if(use[i]==0) //剪枝1  {  num[x]=i;  use[i]=1;  a[q]=i; //确定下来的数字作为参数传递下去，更方便  Search2(p,q+1,a,u);  num[x]=-1;  use[i]=0;  }  }  }  Else //如果字母在之前的搜索已经被确定  {  a[q]=num[x];  Search2(p,q+1,a,u);  }  }  void Search(int p,bool u) //处理第p位，上一位进位为u  {  if(p>=n) //如果出结果了  {  int i;  for(i=0; i<n-1; i++)  cout<<num[i]<<' ';  cout<<num[i];  cout<<endl;  exit(0); //输出后直接结束程序，节约返回的时间  }  int a[2];  Search2(p,0,a,u);  }  int main()  {  for(int i=0; i<30; i++) //初始时所有的字母、数字均未定  num[i]=-1,use[i]=0;  cin>>n>>in[0]>>in[1]>>in[2];  for(int i=0; i<n; i++) //将字符串按预定规则预处理至A数组中  for(int j=0; j<3; j++)  A[n-1-i][j]=in[j][i]-'A';  Search(0,0); //从最低位开始搜索，此时没有进位  //cout<<"NO"<<endl; //题目保证有且只有一组解  return 0;  } |

|  |  |
| --- | --- |
| man | 进一步改进的方法是使用随机化搜索或双向搜索，这是因为如果数据是完全逆序的或者是完全升序的，可能会导致搜索的效果不好，双向搜索就是1、n、２、n－1，…地搜索，但针对特别的数据，可能会导致搜索超时，而随机化搜索的效率一般是很稳定的。请尝试完成。 |