|  |  |
| --- | --- |
| 说明: D:\第一部出版本全部内容\封面和人物设计\1.png | 拿到题目之后，我们马上可以想到用枚举法──枚举删边的先后顺序。但边数最大可以达到50，枚举的复杂将会有50！。因此枚举算法马上被排除了。  对最优化问题的求解，我们往往可以使用动态规划来解决。这道题是不是可以使用动态规划呢？ |

考虑样例，我们可以先删除任何一条边，例如删除边1，如图13.9所示。

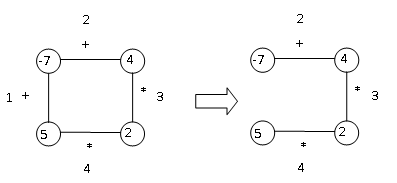


图13.9

将边1删除后，多边形变成了一条线如图13.10所示。



图13.10

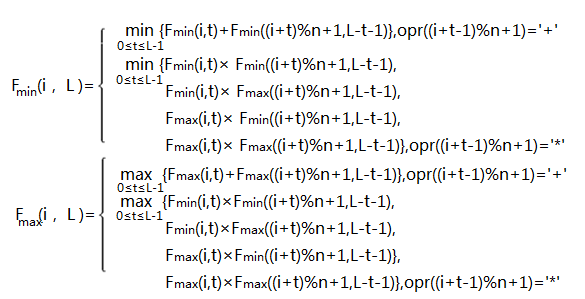
在这条“线”当中继续删边，并且每次删边都使被删边两旁的点按边上的操作符合并如图13.11所示。这样进行了n－1次删边操作后，“线”变成了一个点。问题变成了如何安排删边的顺序，使最后的点上的数尽可能的大。



图13.11

这类似于合并魔法石问题，但问题关键在于乘法运算同号得正，异号得负，如果a，b都是负数，那么a×b有可能得到一个很大的正数。因此需要同时保存子问题的最大值和最小值。

设以节点i为首按顺时针长度为L的链的计算结果最小值为Fmin（i，L），最大值为Fmax（i，L），联结第i个节点和其顺时针方向的下一个节点（i % n）＋1的边上的运算符记为opr（i），则可以得到以下公式：



其中(i＋t)% n＋1是节点i顺时针方向的第t＋1个节点的编号，opr（i）为联结第i个节点和其顺时针方向的下一个节点(i % n)＋1的边上的运算符。

边界条件为：

Fmin(i，0)＝V(i) ；其中V(i)为节点i上标的数字

Fmax(i，0)＝V(i)

由此，我们可以递推求出所有的F(i，n)，i＝1，2，…，n，其中的最大值就是所求。

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 | //多边形魔法阵  #include <bits/stdc++.h>  using namespace std;  #define NMAX 51  int n;  int arr[NMAX];  char opr[NMAX];  int Fmax[NMAX][NMAX],  Fmin[NMAX][NMAX];  bool bmax[NMAX][NMAX],  bmin[NMAX][NMAX];  int calcmax(int i, int l);  int calcmin(int i, int l);  int main(void)  {  cin >> n;  for(int i = 0; i < n; i++)  cin >> arr[i] >> opr[i];  int ans = 0x80000000, t; //初始化为最小值  for(int i = 0; i < n; i++) //循环每个结点开始链  if((t = calcmax(i, n)) > ans)  ans = t;  cout << ans << '\n';  return 0;  }  //求下标为i开始，长度为l的链最大值  int calcmax(int i, int l)  {  if(l <= 1)  return arr[i];  if(bmax[i][l]) //另用数组记录是否已经计算，防止结果为0导致多次重算  return Fmax[i][l]; //如果已计算就返回  int ret = 0x80000000; //初始化为最小值  for(int t = 0; t < l - 1; t++)  {  int tmp;  if(opr[(i + t) % n] == '+')  {  //加号只用计算两边最大值  tmp = calcmax(i, t + 1) +  calcmax((i + t + 1) % n, l - t - 1);  if(tmp > ret)  ret = tmp;  }  else  {  //乘法对两边最大、最小值求解，统计最大值  tmp = calcmax(i, t + 1) \*  calcmax((i + t + 1) % n, l - t - 1);  if(tmp > ret)  ret = tmp;  tmp = calcmax(i,t+1)\*calcmin((i+t+1)%n,l-t-1);  if(tmp > ret)  ret = tmp;  tmp = calcmin(i,t+1)\*calcmax((i+t+1)%n,l-t-1);  if(tmp > ret)  ret = tmp;  tmp = calcmin(i,t+1)\*calcmin((i+t+1)%n,l-t-1);  if(tmp > ret)  ret = tmp;  }  }  bmax[i][l] = true; //设置已经计算  Fmax[i][l] = ret; //保存结果  return ret;  }  //求下标为i开始，长度为l的链最小值（总体同上）  int calcmin(int i, int l)  {  if(l <= 1)  return arr[i];  if(bmin[i][l])  return Fmin[i][l];  int ret = 0x7FFFFFFF;  for(int t = 0; t < l - 1; t++)  {  int tmp;  if(opr[(i + t) % n] == '+')  {  tmp=calcmin(i,t+1)+calcmin((i+t+1)%n,l-t-1);  if(tmp < ret)  ret = tmp;  }  else  {  tmp = calcmin(i,t+1)\*calcmin((i+t+1)%n,l-t-1);  if(tmp < ret)  ret = tmp;  tmp=calcmax(i,t+1)\*calcmin((i+t+1)%n,l-t-1);  if(tmp < ret)  ret = tmp;  tmp=calcmin(i,t+1)\*calcmax((i+t+1)%n,l-t-1);  if(tmp < ret)  ret = tmp;  tmp=calcmax(i,t+1)\*calcmax((i+t+1)%n,l-t-1);  if(tmp < ret)  ret = tmp;  }  }  bmin[i][l] = true;  Fmin[i][l] = ret;  return ret;  } |