NOIP 2011 年普及组解题报告

北京市八一中学　教练员　王祺磊

前言

又是一年……今年，我真心感谢我的学生们。是他们跟我一起，不分春夏秋冬，无论上课时间还是周末、假日，我们共同努力、共同奋斗，才有了今年的收获。当然，我们的缺陷还很多，但是，我们相信，经过不懈的努力，我们一定能够不断的前进。

本解题报告，前面部分为解题思路，如果您对于思路并不是很确定参看前面部分即可。对于题目的参考程序，放在最后。如果您在代码遇到困难时，也可以在后面参看。

因本人水平有限，也欢迎各位老师、选手给我提出宝贵的意见。我的邮箱：fatship@163.com。我的 blog地址为：http://fatship.blog.163.com。

今年题目已经上传至http://marcool.net。如果您需要验证自己的程序是否正确也可以登录网站，提交解题代码验证。

感谢阅读。

总体分析：

今年题目与去年相比难度提升较多。简单题目不简单，难题难得非常有水平。总体来说，基础知识点考察细致，对算法要求进一步提升。开始更多涉猎数据结构的使用。总体来说，NOIP 2011 普及组出题难度较合适，知识点考察全面细致，是一套不错的题目。

本套解题报告，只提及部分代码，完整代码还有待读者细加消化后，自行完成。解题报告亦不再累述题目内容，有需要的读者可以通过题目后面网页链接查看原题。本套题目已全部上传 marcool.net，欢迎有兴趣的读者登陆 marcool.net 消化题目。

具体分析：

第一题：数字反转

【考察重点】

1、对问题情况的全面把握，不遗漏考察点；

2、字符串的基本操作；

3、字符串函数的熟练使用；

【推荐思路】

此题，思路清晰简单。多数选手，可以很好的完成题目，但使用方法较为单一。在做点反转功能时，不妨考虑使用字符串函数 strrev 来直接完成反转。

当然使用 strrev 会遇到一些问题，比如“-”号。在这里推荐使用字符指针。

char *p;

char s[20];

定义一个地址指针，然后用 p 来代替 s。这样 p 的值就可以根据 s[0]的值考虑是 p = s，还是 p = s + 1。

然后针对 p 进行反转即可。

当然，这道题目，还需要考虑翻转后的前 0 问题。这个问题，推荐在翻转后，直接判断 p[0]的值是否为“0”。用一个 while 循环解决。但是，此 while 循环，最好不要忽视数值直接为 0 的情况。

【问题点】

测试数据并未涉及输入为“0”时的情况。很多同学还是因此逃过一劫的。

第二题：统计单词数

【考察重点】

可以说这道题目是一等奖和二等奖的分水岭。此题目的得分直接影响最终成绩。

1、对程序的细心把握；

2、对细致情况的全面分析；

3、算法合理性的基本选择能力。

【推荐思路】

从测试数据的数据量来分析。文章长度为 1000000，而单词长度为 10。即便一位一位的匹配。运算测试也就是在 10000000 左右。而实际每一个单词只有匹配一次。所以，如果控制的好的话，完全可以把运算量控制在百万级别。但即便是千万的运算次数对于我们的测评来说，还是绰绰有余的。所以，在这道题目中，不用考虑 KMP 等字符串匹配算法。

当然，这道题目在测试数据中也增加了很多陷阱。如多空格分隔。前空格分隔和后空格分隔等等。但是，最最朴素的匹配算法，倒是很简单的能够将这些问题排除掉。可以说，出题人还是主要考虑考场选手的基本功底的思路。

【问题点】

1、多空格分隔；

2、大小写不区分；

3、完整单词匹配，即匹配开始的前一个为空格，后一个为空格或者“\0”。

第三题：瑞士轮

【考察重点】

1、模拟算法的基础描述能力；

2、快排；

3、排序算法的联想能力；

4、归并排序的核心思想；

5、滚动数组的使用；

【推荐思路】

首先说，这道题目的题目类型，应该划归为模拟。按理说，只要按照题目的要求一步一步来做，就可以解决问题。但是，这道题目，在数据量上开始做了很大文章。在第一次读题时，我也武断的判断这道题的难度为中等。即在考核点上，只是考核一个快排即可。但是，很快，在我写完这道题目之后，发现，竟然只能通过 4 个点。

这时才开始思考，nlogn 的效率，是否可以完成这个任务数量最大为 200000，$\log_2 n$ 大概在 17 左右，然后乘 200000。再去乘 50。的确是 1 亿 7 千万次左右。这的确超过了，我们能够承受的速度几倍。当然，也想了一次排序后，再进行插入排序，毕竟合并果子的先例的确存在。不过，这个题目和合并果子还是有很大差距的。尝试后，也的确不能够达到目的。

当然，也是受到了网上的启发后，开始感慨，题目设计的巧妙。将结构定义为两个有序序列的合并。也就是归并法排序上来。这就提高了很大的难度。相信不少同学在这道题目丢分也是因为忽略了这个排序方法。

也就是说归并两个有序序列就成为了我们在这道题目里面的重点。题目设计为奇数项和偶数项进行比较，结果将必有一个增加，另外一个保持不变。那么其实在这道题里面我们就可以保证增加项的序列仍然保持有序，而未增加项的序列也保持有序。那么就需要我们在每一次做完比较后，将胜者保持在奇数项和败者保留在偶数项。这样就可以把一个序列的奇数项和偶数项分别当成两个序列。将他们合并到另外一个序列中去。然后再将那么序列覆盖会原有序列中。这样就能够最大效果的减少计算量。

所以，本道题目，需要先进行一次快排（而且不能单纯使用左侧数值当成标准值，否则有数据将快排降速到 n^2 效率）。每轮结束后，都进行一次归并排序，再覆盖回原来序列（当然，也可以使用滚动数组，让两个数组来回使用，提高效率。）

【问题点】

1、低估运算效率，单纯使用快排或基础排序方法；

2、快排单纯使用起始位置值为标准值划分；

3、为考虑胜者在前败者在后，无法形成归并排序序列思路。

第四题　表达式的值

【考察重点】

1、表达式计算；

2、栈的结构使用；

3、或运算和与运算的运算特点，即递推式；

【推荐思路】

在题目阐述中，对题目要求说的非常清晰，即题目本身就是一个非常标准的表达式计算。只是为了节约一些字符串处理的时间，将题目进行了简化。因为只给符号，那么数值就只可能是 0 或者 1。也就是在每一位上都会产生一个 0 的种类的变化和 1 的种类的变化。这就省去了，输入中数值的提取过程。

而在运算表达式的基础上。我们可以把两个运算符的计算变化成下面的过程。如：+号，即或的位运算。假设它两边的表达式为 a 和 b。那么 f(a + b)表示 a + b 这个表达式，最后的的运算种类数。因为在计算过程中我们需要考虑结果是 0 的种类和结果是 1 的种类。所以，可以把 f(a + b) = f0(a + b) + f1(a + b)。我们分别计算一下 f0(a + b)和 f1(a + b)。

f0(a + b) = f0(a) * f0(b)，也就是要求 a 和 b 结果均为 0 的种类数相乘。

f1(a + b) = f0(a) * f1(b) + f1(a) * f0(b) + f1(a) * f1(b)。也就是 a 或 b 有一个为 1 的种类数的和。

自然我们可以得到：

f0(a * b) = f0(a) * f1(b) + f1(a) * f0(b) + f0(a) * f0(b)

f1(a * b) = f1(a) * f1(b)

这个就是运算递推式的基础。

剩下的内容就是，表达式运算的栈操作了。标准的表达式运算需要两个栈进行操作。一个是操作符栈，一个是数值栈。而在本题中，很明显，我们需要 3 个栈：一个操作符栈，两个数值栈（一个存得 0 的种类数、一个存得 1 的种类数）。

介绍一下优先级判定数组。

个人比较喜欢的方法，设立一个数组，按照数需存放'(‘，'+’，'*’，')'这几个符号。然后再定义一个二维数组，分别代表前后的两个操作符。

如下：

前\后	（	+	*	）
（	<	<	<	=
+	<	>	<	>
*	<	>	>	>
）	>	>	>	>

这幅图，列举了按照前后顺序，操作符相遇时的优先级问题。当然，特别需要注意的是第一行的第四个等号。这个时候，其实就是一对括号中已经计算完成时的状态，只需要消掉相应一对括号即可。

另外，需要注意的是，在这个数组中，遇到小于号时，则后面优先级高于前面优先级，此时前面数据无法计算，即需要做压栈处理。只有当遇到大于号时，则栈顶优先级高于待处理优先级，则进行退栈和运算操作，即所有进行的运算均在栈顶进行。

在处理数值栈时，无比注意，只有当入栈符号非左括号时，才需入站 f0,f1 为（1,1）。而为左括号时，只需将操作符入站即可。即基本入栈情况为先数后操作符，而左括号除外。右括号只有出栈操作，不用考虑。

【问题点】

1、与、或运算的递推式；

2、栈操作的基本思路和细节。

参考程序：

1、 数字反转：

```
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

ifstream cin("reverse.in");
ofstream cout("reverse.out");

int main(int argc, char *argv[])
{
    char s[100];
    char *p;//实际操作数组部分
    cin >> s;
    p = s;
    if (p[0] == '-')
    {
       p++;//去负号，便于翻转
       cout << "-";
    }
    strrev(p);//因可能出现的负号已经去除，可安心翻转
    //去掉前 0，为排除全 0 而特别考虑保留至少 1 个
    while (p[0] == '0' && p[1] != '\0') p++;
    cout << p << endl;
    //system("PAUSE");
    return EXIT_SUCCESS;
}
```

2、统计单词数

```
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

ifstream cin("stat.in");
ofstream cout("stat.out");

char p[1000050];
char s[20];
int main(int argc, char *argv[])
{
    cin.getline(s, 20);
    cin.getline(p, 1000005);
    //将两个字符串均进行大写处理，便于匹配
    strupr(s);
    strupr(p);
    bool f;
    int l1, l2, x, q;
    int i, j;
    l1 = strlen(s);
    l2 = strlen(p);
    f = false;
    x = 0;
    q = -1;
    for (i = 0 ; i < l2 ; i++)//逐字符匹配
    {
      if (i > 0 && p[i - 1] != ' ') continue;//进行前空格检测，确认匹配从单词开头开始
      for (j = 0 ; j < l1 ; j++)
      {
        if (p[i + j] != s[j])
          break;
        //其实可以跳过当前单词的部分，但不影响实际效率，就不做介绍了
      }
      if (j == l1 && (p[i + j] == ' ' || p[i + j] == '\0'))//特别需要对\0 的判断。
      {
        if (q == -1) q = i;
        f = true;
        x++;
      }
```

```
        }
        if ( f ) cout << x << " ";
        cout << q << endl;//对 q 做了初始值为-1 的处理便于输出。
        //system("PAUSE");
        return EXIT_SUCCESS;
}
```

3、瑞士轮

```
#include <cstdlib>
#include <fstream>

using namespace std;
//结构体便于处理排序和数据操作
struct person
{
   int num;
   int s;
   int w;
}ps[2][200050], pt[200050];
//多关键字快排
void qsort(int b, int e)
{
   person t;
   int l, r, x;
   if (b >= e) return ;
   l = b;
   r = e;
   x = (b + e) / 2;//取中值为快排参数
   swap(ps[0][x], ps[0][b]);
   t = ps[0][b];
   while (l < r)
   {
      while (l < r && (ps[0][r].s < t.s || ps[0][r].s == t.s && ps[0][r].num > t.num)) r--;
      ps[0][l] = ps[0][r];
      while (l < r && (ps[0][l].s > t.s || ps[0][l].s == t.s && ps[0][l].num < t.num)) l++;
      ps[0][r] = ps[0][l];
   }
   ps[0][r] = t;
   qsort(b, r - 1);
   qsort(r + 1, e);
}

int main(int argc, char *argv[])
{
      int n, r, q, x;
      int i, j;
      int z;
      int win, lose;
      freopen("swiss.in", "r", stdin);
      freopen("swiss.out", "w", stdout);
```

```
        scanf("%d %d %d", &n, &r, &q);
        n <<= 1;
        for (i = 1 ; i <= n ; i++)
        {
            ps[0][i].num = i;
            scanf("%d", &ps[0][i].s);
        }
        for (i = 1 ; i <= n ; i++)
        {
            scanf("%d", &ps[0][i].w);
        }
        x = n / 2;
        qsort(1, n);
        z = 0;//z 为滚动数组操作，即在数组 0 和 1 之间不断滚动切换
        for (i = 0 ; i < r ; i++)
        {
            for (j = 1 ; j <= x ; j++)
            {
                if (ps[z][2 * j - 1].w > ps[z][2 * j].w)
                    ps[z][2 * j - 1].s++;
                else
                {
                    ps[z][2 * j].s++;
                 //当后者赢时，交换相应数据，维护奇数项、偶数项有序
                    swap(ps[z][2 * j], ps[z][2 * j - 1]);
                }
            }
            win = 1;
            lose = 2;
          //奇数项、偶数项进行归并排序，
            for (j = 1 ; j <= n ; j++)
            {
                if (win <= n && (ps[z][win].s > ps[z][lose].s || ps[z][win].s == ps[z][lose].s
&& ps[z][win].num < ps[z][lose].num))
                {
                    ps[1 - z][j] = ps[z][win];
                    win += 2;
                }
                else
                {
                    ps[1 - z][j] = ps[z][lose];
                    lose += 2;
                }
            }
```

```
            z = 1 - z;//切换滚动数组
        }
        printf("%d\n", ps[z][q].num);
        //system("PAUSE");
        return EXIT_SUCCESS;
}
```

4、表达式的值

```
#include <cstdlib>
#include <iostream>
#include <stack>

using namespace std;

char fu[5] = {"(+*)"};//操作符判定
char yx[5][5] = {"<<<=", "<><>", "<>>>", ">>>>"};//按照上面符号顺序，确定前后优先级

char s[100050];
int l;

stack<int> n0;//0 的种类数栈
stack<int> n1;//1 的种类数栈
stack<char> p;//操作符栈
//查找操作符的编号，便于对比优先级
int pc(char c)
{
   int i;
   for (i = 0 ; i < 4 ; i++)
      if (fu[i] == c)
         return i;
}

int main(int argc, char *argv[])
{
      int a0, a1, b0, b1;
      int t0, t1, i;
      char f;
//为了安全，先进行栈清空操作。
      while (!n0.empty())n0.pop();
      while (!n1.empty())n1.pop();
      while (!p.empty())p.pop();
      cin >> l;
      cin >> s;
//为便于计算，将表达式套入一对括号之中。
      s[l] = ')';
      l++;
      s[l] = '\0';
      i = 0;
//处理第一位的种类数并将添加的左括号入栈
```

```
            n0.push(1);
            n1.push(1);
            p.push('(');
            while (i < l)//目标即为全部操作符入栈完毕
            {
               if (yx[pc(p.top())][pc(s[i])] == '=') //当遇到()这种情况时代表括号内处理完毕
               {
                  p.pop();//去掉栈内左括号
                  i++;//去掉右括号
               }
               else
               {
                  if (yx[pc(p.top())][pc(s[i])] == '<')
                  {
                     p.push(s[i]);//操作符入栈
                     if (s[i] != '(')//不为左括号才满足先数据后操作符入栈操作
                     {
                        n0.push(1);
                        n1.push(1);
                     }
                     i++;
                  }
                  else
                  {
                     if (yx[pc(p.top())][pc(s[i])] == '>')//栈内优先级高，开始计算
                     {
                      //弹出运算符两端的运算数值和提取操作符
                        b0 = n0.top();
                        b1 = n1.top();
                        n0.pop();
                        n1.pop();
                        a0 = n0.top();
                        a1 = n1.top();
                        n0.pop();
                        n1.pop();
                        f = p.top();
                        p.pop();
                      //分别根据或运算或与运算做递推操作
                        if (f == '+')
                        {
                           t0 = (a0 * b0) % 10007;
                           t1 = (a1 * b0 % 10007 + a0 * b1 % 10007 + a1 * b1 % 10007) %
10007;
                        }
```

```
                    else
                    {
                        t0 = (a1 * b0 % 10007 + a0 * b1 % 10007 + a0 * b0 % 10007) %
10007;
                        t1 = a1 * b1 % 10007;
                    }
                   //将运算结果入栈
                    n0.push(t0);
                    n1.push(t1);
                  }
                }
              }
            }
            cout << n0.top() << endl;//栈顶即最后运算结果。
            //system("PAUSE");
            return EXIT_SUCCESS;
}
```