[[九度OJ] 1080 进制转换（大数运算以及如何提升计算效率）](http://blog.csdn.net/on_1y/article/details/7885433)

分类： [九度OJ](http://blog.csdn.net/on_1y/article/category/1206917) [算法](http://blog.csdn.net/on_1y/article/category/1128018)2012-08-20 16:25 461人阅读 [评论](http://blog.csdn.net/on_1y/article/details/7885433#comments)(6) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/on_1y/article/details/7885433#report)

**题目描述：**

将M进制的数X转换为N进制的数输出。

**输入：**

输入的第一行包括两个整数：M和N(2<=M,N<=36)。  
下面的一行输入一个数X，X是M进制的数，现在要求你将M进制的数X转换成N进制的数输出。

**输出：**

输出X的N进制表示的数。

**样例输入：**

16 10

F

**样例输出：**

15

**提示：**

输入时字母部分为大写，输出时为小写，并且有大数据。

-----------------------------------华丽的分割线----------------------------------

题目分析：

题目其实就是实现2~36任意进制转换，之所以是2~36是因为字符'0'~'9','a'~'z'一共有36个，最多可以表示到36进制，当然也可以引入更多的字符来表示更大基底的进制，不过就算法的描述而言，更大的基底没有更多算法上的变化。

-----------------------------------华丽的分割线----------------------------------

算法分析：

限于数学水平和信息搜集能力，我没有找到更好的算法，处理进制转换，基本的算法就是作除法，例如：

将m进制的数s转换为n进制:

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. **while** (m)
2. {
3. r[i++] = m%n;
4. m /= n;
5. }

当然这只是一个简单的描述，代码不能直接这样写，原因在于ｓ是ｍ进制数，不能直接用十进制的方法来除法，并且这里的ｓ是一个大数,就算只是10进制到２进制的转换，ｓ是一个大数时，即便使用unsigned long long数据类型也是不够用的。这里涉及的问题有：

1. s的表示问题

s是ｍ进制数，ｍ最大可以达到３６，里面会出现'ａ'~'ｚ'的字符，这些字符是不能直接参与运算的，需要以适当的方式转换成可以计算的模式

２.ｍ进制除法

通常我们作十进制除法时，是从左向右依次作除法，例如：

http://my.csdn.net/uploads/201208/20/1345433512_2636.jpg

如果是30进制，294就的2/12=0,此时需要用29/12，这时候需要注意要用(2\*30+9)/12，就是上一步除法的余数乘以m加上下一步的数再作除法。

从上面的过程可以看出，解决这个问题需要二步：

1 以适当方式表示出m进制数

2 手工模拟除法

-----------------------------------华丽的分割线----------------------------------

实现：

算法很明确，就是模拟除法，但是实现起来效率差别很大，九度OJ现在开放了查看别人代码的功能，每次查看别人代码要支付九度豆，我觉得这对于学习别人的优秀代码有很大好处，下面就看看同一个算法，不同的实现，效率的差别。以下由慢到快给出几个实现。

实现1：

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. #include <stdio.h>
2. #include <string.h>
3. #define LEN 2000
5. **char** s[LEN];
6. **char** d[] = {'0','1','2','3','4','5','6','7','8','9','A','B', \
7. 'C','D','E','F','G','H','I','J','K','L','M','N','O','P', \
8. 'Q','R','S','T','U','V','W','X','Y','Z'};
9. **int** gap;
10. **void** add (**char** s[], **char** t[], **char** r[])
11. {
12. **int** i,j,k;
13. **long** **long** c,sum;
14. **char** p[LEN];
15. **int** del='0';
17. memset(p,'\0',LEN);
18. i = strlen(s)-1;
19. j = strlen(t)-1;
20. k = 0;
21. c = 0;
23. **while** ( i>=0 && j>=0 )
24. {
25. sum = c;
26. **if**(s[i]>'9')
27. del = 'A'-10;
28. **else**
29. del = '0';
30. sum += s[i]-del;
32. **if**(t[j]>'9')
33. del = 'A'-10;
34. **else**
35. del = '0';
36. sum += t[j]-del;
37. p[k++] = d[sum%gap];
38. c = sum/gap;
39. --i;--j;
40. }
41. **while** ( i>=0 )
42. {
43. sum = c;
44. **if**(s[i]>'9')
45. del = 'A'-10;
46. **else**
47. del = '0';
48. sum += s[i]-del;
49. p[k++] = d[sum%gap];
50. c = sum/gap;
51. --i;
52. }
53. **while** ( j>=0 )
54. {
55. sum = c;
56. **if**(t[j]>'9')
57. del = 'A'-10;
58. **else**
59. del = '0';
60. sum += t[j]-del;
61. p[k++] = d[sum%gap];
62. c = sum/gap;
63. --j;
64. }
65. **while**(c)
66. {
67. p[k++] = d[c % gap];
68. c /= gap;
69. }
70. i = 0;
71. **while** ( k>=1 )
72. r[i++] = p[--k];
73. r[i] = '\0';
74. }
76. **void** multi(**char** s[], **int** n, **char** r[])
77. {
78. **int** slen = strlen(s);
79. **int** i,j,c;
80. **char** t[LEN];
81. **int** del;
83. memset(t,'\0',LEN);
84. j = c = 0;
85. **for**(i=slen-1; i>=0; i--)
86. {
87. del = '0';
88. **if**(s[i]>'9')
89. del = 'A'-10;
90. t[j++] = d[(n \* (s[i]-del) + c)%gap];
91. c = (n \* (s[i]-del) + c)/gap;
92. }
93. **while**(c)
94. {
95. t[j++] = d[c % gap];
96. c /= gap;
97. }
98. i = 0;
99. j--;
100. **while**(j>=0)
101. r[i++] = t[j--];
102. r[i] = '\0';
103. }
105. **void** solve(**int** m, **int** n, **char** s[])
106. {
107. **int** slen = strlen(s);
108. **int** i;
109. **char** r[LEN] = "0";
110. **char** e[LEN] = "1";
111. **char** t[LEN];
112. memset(t,'\0',LEN);
113. gap = n;
115. **for** (i=slen-1; i>=0; i--)
116. {
117. **if** (s[i] > '9')
118. multi(e,s[i]-'A'+10,t);
119. **else**
120. multi(e,s[i]-'0',t);
121. add(r,t,r);
122. **if** (i>=1)
123. multi(e,m,e);
124. }
125. slen = strlen(r);
126. i = 0;
127. **while**(i<slen)
128. {
129. **if**(r[i] >= 'A')
130. r[i] += 32;
131. ++i;
132. }
133. i = 0;
134. **while** (slen !=1 && r[i] == '0')
135. ++i;
136. strcpy(s,r+i);
137. }
139. **int** main(**void**)
140. {
141. **int** m, n;
142. **while** ( scanf( "%d %d",&m,&n ) != EOF )
143. {
144. scanf ("%s", s);
145. **if** (s[0] != '-')
146. {
147. solve (m,n,s);
148. printf ( "%s\n", s);
149. }
150. **else**
151. {
152. solve (m,n,s+1);
153. printf ( "%s\n", s);
154. }
155. }
157. **return** 0;
158. }
159. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
160. Problem: 1080
161. User: minix
162. Language: C
163. Result: Accepted
164. Time:270 ms
165. Memory:912 kb
166. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

不得不说，实现1这段冗长难看的代码是我写的。。。之所以自己去写是因为我觉得不管我会不会，都要努力尝试自己写出来，再改进到不能改进，然后再去看别人的代码。实现的思路很简单。根据公式：

s = a(n)\*m^n + a(n-1)\*m^(n-1+...+a(2)\*m^2+a(1)\*m+a(0)

函数multi负责每次的乘法，add负责每次的加法，只是在计算乘法和加法的过程中，每一步的计算在进位时都是按%n计算的，就是边计算边进制转换。

AC了之后，我就想着怎么去改进，一点点地看哪儿有改进的地方。下面是：

实现2：

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. #include <stdio.h>
2. #include <string.h>
3. #define LEN 1000
5. **char** d[] = {'0','1','2','3','4','5','6','7','8','9','A','B', \
6. 'C','D','E','F','G','H','I','J','K','L','M','N','O','P', \
7. 'Q','R','S','T','U','V','W','X','Y','Z'};
8. **int** gap;
9. **void** add (**char** s[], **char** t[], **char** r[])
10. {
11. **int** i,j,k;
12. **int** c,sum;
13. **char** p[LEN];
14. **int** del='0';
16. i = strlen(s)-1;
17. j = strlen(t)-1;
18. k = 0;
19. c = 0;
21. **while** ( i>=0 && j>=0 )
22. {
23. sum = c;
24. **if**(s[i]>'9')
25. del = 'A'-10;
26. **else**
27. del = '0';
28. sum += s[i]-del;
30. **if**(t[j]>'9')
31. del = 'A'-10;
32. **else**
33. del = '0';
34. sum += t[j]-del;
35. p[k++] = d[sum%gap];
36. c = sum/gap;
37. --i;--j;
38. }
39. **while** ( i>=0 )
40. {
41. sum = c;
42. **if**(s[i]>'9')
43. del = 'A'-10;
44. **else**
45. del = '0';
46. sum += s[i]-del;
47. p[k++] = d[sum%gap];
48. c = sum/gap;
49. --i;
50. }
51. **while** ( j>=0 )
52. {
53. sum = c;
54. **if**(t[j]>'9')
55. del = 'A'-10;
56. **else**
57. del = '0';
58. sum += t[j]-del;
59. p[k++] = d[sum%gap];
60. c = sum/gap;
61. --j;
62. }
63. **while**(c)
64. {
65. p[k++] = d[c % gap];
66. c /= gap;
67. }
68. i = 0;
69. **while** ( k>=1 )
70. r[i++] = p[--k];
71. r[i] = '\0';
72. }
74. **void** multi(**char** s[], **int** n, **char** r[])
75. {
76. **int** slen = strlen(s);
77. **int** i,j,c;
78. **char** t[LEN];
79. **int** del;
80. **int** sum;

83. j = c = 0;
84. **for**(i=slen-1; i>=0; i--)
85. {
86. del = '0';
87. **if**(s[i]>'9')
88. del = 'A'-10;
89. sum = n \* (s[i]-del) + c;
90. t[j++] = d[sum%gap];
91. c = sum/gap;
92. }
93. **while**(c)
94. {
95. t[j++] = d[c % gap];
96. c /= gap;
97. }
98. i = 0;
99. j--;
100. **while**(j>=0)
101. r[i++] = t[j--];
102. r[i] = '\0';
103. }
105. **void** solve(**int** m, **int** n, **char** s[])
106. {
107. **int** slen = strlen(s);
108. **int** i;
109. **char** r[LEN] = "0";
110. **char** e[LEN] = "1";
111. **char** t[LEN];
113. **char** ns[2];ns[1]='\0';
114. **for** (i=0; i<slen; i++)
115. {
116. multi(r,m,r);
117. ns[0] = s[i];
118. add (r,ns,r);
119. }
120. i = 0;
121. **while**(r[i])
122. {
123. **if**(r[i] >= 'A')
124. r[i] += 32;
125. ++i;
126. }
127. i = 0;
128. **while** (slen !=1 && r[i] == '0')
129. ++i;
130. strcpy(s,r+i);
131. }
133. **int** main(**void**)
134. {
135. **int** m;
136. **char** s[LEN];
137. **while** ( scanf( "%d%d%s",&m,&gap,s ) != EOF )
138. {
139. **if** (s[0] != '-')
140. {
141. solve (m,gap,s);
142. puts(s);
143. }
144. **else**
145. {
146. solve (m,gap,s+1);
147. puts(s);
148. }
149. }
151. **return** 0;
152. }
153. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
154. Problem: 1080
155. User: minix
156. Language: C
157. Result: Accepted
158. Time:110 ms
159. Memory:908 kb
160. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

这次改进主要是改了两方面

一方面是细节上，去掉了一些不必要的初始化，例如memset，有时候根本没有必要调用它来浪费时间。把重复计算的地方去掉，将计算结果放到临时变量里面。

一方面是算法上，改变了计算的方式：

s = a(n)\*m^n + a(n-1)\*m^(n-1+...+a(2)\*m^2+a(1)\*m+a(0)

计算这个公式时，实现1采用从a(0)开始累加，实现2则是从a(n)开始，把s写成

s = (.....((a(n)\*m+a(n-1))\*m+a(n-2))\*m+a(n-3)......)\*m + a(0)

使用这种计算方式，可以减少乘法运算的次数，从而提升效率

下面是用九度豆从九度OJ上看来的代码（感谢九度OJ以及kinwolfofsky以及鹰之殇，如果觉得贴这不合适，通知我，我会及时删除）

实现3

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. #include<stdio.h>
2. **char** s[1000],t;
3. **int** X[1000],Y[1000];
4. **inline** **int** ctoi(**char** c){**if**(c<='9')**return** c-'0';**else** **return** c-'A'+10;}
5. **inline** **char** itoc(**int** i){**if**(i<10)**return** i+'0';**else** **return** i-10+'a';}
6. **int** main(){
7. **int** M,N,i,j,l;
8. **while**(~scanf("%d%d%s",&M,&N,s)){
9. i=-1;**while**(s[++i])X[i]=ctoi(s[i]);
10. l=i--;j=0;**while**(j<i){t=X[j];X[j++]=X[i];X[i--]=t;}
11. j=0;
12. **while**(l>0){
13. **for**(i=l-1;i>0;--i){X[i-1]+=(X[i]%N)\*M;X[i]/=N;}
14. Y[j++]=X[0]%N;X[0]/=N;
15. **if**(!X[l-1])--l;
16. }
17. Y[j]=0;**while**(j&&Y[j]==0)--j;
18. **if**(j==0&&Y[0]==0)putchar('0');**else**{++j;**while**(j--)putchar(itoc(Y[j]));}
19. puts("");
20. }
21. }
22. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
23. Problem: 1080
24. User: kingwolfofsky
25. Language: C
26. Result: Accepted
27. Time:70 ms
28. Memory:920 kb
29. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

选择这这个代码是因为这是70ms级里面代码长度最短的，虽然我以前觉得一行代码只写一行，但是看了这个代码后，觉得确实比较美观~

代码的思路是：

1 将m进制数s从字符串类型转化为整数数组类型，每一个字串对应一个整数数组元素，这样方便计算

2 手工模拟除法，x[]/n: 每一个数组元素x[i]首先把自身%n的余数\*m再加到下一个数组元素x[i-1]中，然后自身再/n，一直到把所有数组元素处理完，这样得到一个余数，存储到y[]中，此时即完成 x[] = x[] / n；此后作循环处理，直到x[]为0；这和我们平时作的除法其实是一样的，仔细想想，找个例子模拟一下数据就明白了。  
实现4

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
4. **int** main()
5. {
6. **int** M,N,\*Y,i,j,k,\*m,n,s,t;
7. **char** \*X;
8. X=(**char** \*)malloc(1000\***sizeof**(**char**));
9. Y=(**int** \*)malloc(1000\***sizeof**(**int**));
10. m=(**int** \*)malloc(1000\***sizeof**(**int**));
11. **while**(scanf("%d %d",&M,&N)!=EOF)
12. {
13. memset(X,'\0',1000\***sizeof**(**char**));
14. memset(Y,0,1000\***sizeof**(**int**));
15. memset(m,0,1000\***sizeof**(**int**));
16. scanf("%s",X);
17. n=strlen(X);
18. **for**(i=0;i<n;i++)
19. {
20. **if**(X[i]<='9'&&X[i]>='0') Y[i]=X[i]-'0';
21. **else** Y[i]=X[i]-'A'+10;
22. }
23. k=1;
24. **for**(i=0;i<n;i++)
25. {
26. s=0;
27. **for**(j=0;j<k;j++)
28. {
29. **if**(j==0)
30. {
31. t=(m[j]\*M+Y[i]+s)/100000;
32. m[j]=(m[j]\*M+Y[i]+s)%100000;
33. s=t;
34. }
35. **else**
36. {
37. t=(m[j]\*M+s)/100000;
38. m[j]=(m[j]\*M+s)%100000;
39. s=t;
40. }
41. }
42. **if**(t!=0) m[k++]=s;
43. }
44. **for**(i=0;m[0]>=N||(m[0]<N&&m[k-1]!=0&&k!=1);i++)
45. {
46. s=0;
47. **for**(j=k-1;j>=0;j--)
48. {
49. t=(m[j]+s\*100000)%N;
50. m[j]=(m[j]+s\*100000)/N;
51. s=t;
52. }
53. Y[i]=s;
54. **if**(m[k-1]==0) k--;
55. }
56. Y[i]=m[0];
57. n=i;
58. **for**(i=0;i<=n;i++)
59. {
60. **if**(Y[n-i]<10) X[i]=Y[n-i]+'0';
61. **else** X[i]=Y[n-i]-10+'a';
62. }
63. X[i]='\0';
64. printf("%s\n",X);
65. }
66. free(X);
67. free(Y);
68. free(m);
69. **return** 0;
70. }
71. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
72. Problem: 1080
73. User: 鹰之殇
74. Language: C
75. Result: Accepted
76. Time:30 ms
77. Memory:908 kb
78. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

选择这个代码是因为这是九度OJ里面用时最少的，而且代码量比别的30ms级的少。这个算法的基本思想是这样的：

1. 把m进制数x[]由字符串转为整数数组y[]，这是为了方便计算，和实现3道理一样

2. 之所以这个代码运行的快，就在第2步,没有像实现3那样直接循环使用y[]/n，求余数，而是采用了将数据进行分割与积累的方式，将y[]转变成m[]

3. 将m[]循环执行m[]/n得到多个余数

 举例说明一下：

将10进制数123456789转变成10进制数,其中分割数据的基数为100

1. x[] = "123456789"-->y[] = {1,2,3,4,5,6,7,8,9}

2.y[] = {1,2,3,4,5,6,7,8,9} ->m[] = {89,67,45,23,1}

如果采用实现3的方法，y[]的每次求余数时，y[]第一个数组元素都要参与除法，假如x[]有1000位，第一次求余数的除法次数就是1000次，别外每次除法的余数还要\*m再与下一个数组元素相加，可以说乘法次数和除法次数都很大。

如果采用实现4的方法，在上例中m[]的数组元素个数是5个，如果基数的选择变大，数组元素还会更少，这样，就大大减少了求余数时所使用的乘法和除法的计算次数。如果减少的次数大于y[]->m[]的开销，那么这种优化就是可取的，事实证明，这样的改进确实减少了时间。

需要注意的是m[] = {89,67,45,23,1},而不是被转换成{12,34,56,78,9}。这其实是不一样的。z[]需要参与后面的除法运算，因此高位的余数要扩大确定的倍数以后再加到低位，例如，分割的基数为1000时：

如果采用高位取满（123），低位不足（78），数据是12345678，分割成{123,456,78}，那么倒数第二次求得的余数扩大的倍数并不是1000，而是100，因为78只有两位，这样，如果低位不足，就要根据低位的数据来决定上一次余数扩大多少倍。

如果采用低位取满(678)，高位不足(12),数据是12345678，分割成{678，456，12}，就不会出现余数扩大位数不一致的问题

而理解实现4的关键就在于如何将y[]->m[],我曾经也想过这个问题，但是由于代码能力数学能力不足，想的也不是那么清楚，所以并没有实现。下面是这段关键代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. k=1;
2. **for**(i=0;i<n;i++)
3. {
4. s=0;
5. **for**(j=0;j<k;j++)
6. {
7. **if**(j==0)
8. {
9. t=(m[j]\*M+Y[i]+s)/100000;
10. m[j]=(m[j]\*M+Y[i]+s)%100000;
11. s=t;
12. }
13. **else**
14. {
15. t=(m[j]\*M+s)/100000;
16. m[j]=(m[j]\*M+s)%100000;
17. s=t;
18. }
19. }
20. **if**(t!=0) m[k++]=s;
21. }

还以刚才的例子 说明

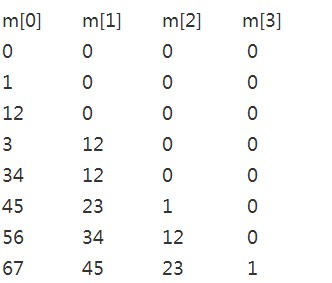
将10进制数1234567转变成10进制数,其中分割数据的基数为100(就是将上述代码中的100000改为100）

1. x[] = "1234567"-->y[] = {1,2,3,4,5,6,7}

2.y[] = {1,2,3,4,5,6,7} ->m[] = {67,45,23,1}

上述代码实现的就是2的过程

外层循环for (i=0; i<n; i++) 控制每次新加入一个数组元素，内层循环for(j=k; j>=0; j--)实现m[]中数据不断向右推移，所以m[]中的数据其实是这样变化的：



从上面的数据变化应该会明白整个过程是如何动作的，数据首先在m[0]中积累，积累到超过分割基数时，m[]中的数据就要向右移动，变量s其实就是每次向右移动的那个数字。s的最后一个值就是向最高位存储的数值。如果s不为0说明有进位。这样m[]中的元素个数就增加，k即表示m[]中元素的个数。

故事的最后，提供一个我后来想明白思路后重新写的：

实现5：

**[cpp]** [view plaincopy](http://blog.csdn.net/on_1y/article/details/7885433)

1. #include <stdio.h>
2. #include <string.h>
4. #define XMAX 100000000000000
5. #define LEN 1000
7. **char** itoc (**int** i ) { **return** (i>=10)?(i-10+'a'):(i+'0'); }
9. **int** main (**void**)
10. {
11. **int** m,n,len,i,j,k,x[LEN];
12. **char** t,str[LEN];
13. unsigned **long** **long** s,sum,y[LEN];
15. **while** (scanf ("%d%d%s",&m,&n,str) != EOF)
16. {
17. len = 0;
18. **while** (str[len]) {
19. x[len] = (str[len]>'9')?(str[len]-'A'+10):(str[len]-'0');
20. ++len;
21. }
23. k = 1;
24. y[0] = 0;
25. **for** (i=0; i<len; i++)
26. {
27. s = 0;
28. **for** (j=0; j<k; j++)
29. {
30. sum = (j==0)?(y[j]\*m+x[i]+s):(y[j]\*m+s);
31. s = sum / XMAX;
32. y[j] = sum % XMAX;
33. }
34. **if** (s) y[k++] = s;
35. }
37. len = k;
38. k = 0;
39. j = len - 1;
40. **while** (j>=0)
41. {
42. s = 0;
43. **for** (i=j; i>=0; i--)
44. {
45. sum = s\*XMAX + y[i];
46. s = sum % n;
47. y[i] = sum / n;
48. }
49. str[k++] = itoc(s);
50. **if** (y[j]==0 || j==0) j--;
51. }
52. **while** (y[0]) { str[k++] = itoc(y[0]%n); y[0] /= n; }
53. str[k] = '\0';
54. i = 0; j=k-1;
55. **while** (i<j) { t=str[i];str[i]=str[j];str[j]=t; ++i; --j; }
57. puts(str);
58. }
59. **return** 0;
60. }
61. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
62. Problem: 1080
63. User: minix
64. Language: C
65. Result: Accepted
66. Time:10 ms
67. Memory:908 kb
68. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

运行时间减少到了10ms，其实就是扩大了那个分割基数以及减少了没必要的初始化还有减少了重复计算。

其实每次我一想这个题的时候都会去想，同一个问题，不管是多小的问题，例如这个任意进制转化问题，世界上最快的算法是什么，可是，我还是不太会找~~~也不知道怎么设计一个算法~~