

--

В продолжение статьи «Циклический инкремент паролей» хотелось бы затронуть тему распределения диапазонов паролей из заданного множества символов. Во время развития распределения мощностей между различными машинами задача является наиболее актуальной, например, описанный ниже метод уже был применен в системе распределенного перебора BruteNet. Применяемый язык C++.

Назовем заданное множество символов, на котором ведется перебор, **charset**. На уровне программы это будет обычный массив байт (будем считать для ясности, что любой символ можно представить в одном байте), таким образом, если перебор ведется на множестве 'abc', то:

```
charset[]='abc';
```

На момент описания внутреннего представления каждой строки мы можем абстрагироваться от ее непосредственного представления, храня каждую строку как набор позиций, начиная с единицы, текущего символа в массиве **charset**. Такое представление действительно является удобным: строка 'bab', хранимая в виде '\x02\x01\x02' дает возможность перейти к следующей допустимой строке 'bac' простой инкрементацией последнего байта. Примем очевидным факт того, что начиная со строки 'a' и переходя подобным образом к следующей строке мы обойдем последовательно все участвующее в переборе строки. Такое поведение наталкивает на мысль о «численно-подобном» поведении строк. Действительно, прибавление единицы почти в точности выполняется по правилам сложения столбиком, однако при переходе за предел длины **charset** мы ставим текущий символ не в ноль, а в единицу, делая перенос на разряд влево. Таким образом, следующая строка после 'bac' будет 'bba'. Если бы мы научились выполнять арифметические операции на такими «числами» нам бы, очевидно, не составило бы и труда выполнить изначальную задачу деления диапазона. Но это и не представляется проблемой, ведь тема длинной целочисленной арифметики в пределах нашей задачи не является трудоемкой. Итак, установив связь между строками и числами нам остается только понять как их хранить в виде непосредственно чисел; параллельно возникает и обратная задача – преобразование такого числа в строку.

### Длинная арифметика и деление диапазона

Заметим, что отображение каждой строки на множество целых чисел больше нуля является взаимно-однозначным, более того будем считать значением строки `abc...def` длиной `n` число равное:

$$f+e*power+d*power^2+...+c*power^{(n-3)}+b*power^{(n-2)}+a*power^{(n-1)}$$

Причем очевидно, что каждый символ строки принимает значение из диапазона `[1..power]`, где **power** – количество символов в **charset**. Будем считать **power** степенью системы исчисления нашей арифметики. Для того чтобы строка была корректной для привычных операций столбиком, необходимо, ее изменить так, чтобы соответствующее ей число не поменяло значения и каждый символ находился в промежутке от `[0..power-1]`. Для этого достаточно пройти по строке в цикле с младшего разряда к большему и, если, текущее значение больше либо равно **power** вычесть **power** и прибавить единицу к следующему разряду.

Затроним момент реализации. Для простоты будем считать, что количество символов лежит в константе `PASS_SIZE`:

```
BYTE power;  
BYTE word[PASS_SIZE];
```

Массив `word` будет хранить в себе число с которым мы и будем проводить все арифметические операции. В итоге функция преобразующая строку в необходимый нам вид будет выглядеть так:

```
arithmatic ariphmetic::operator=(char * rhs)  
{ memset(word,0,PASS_SIZE);  
  memcpy(&word[PASS_SIZE-strlen(rhs)],rhs,strlen(rhs));  
  
  for(i=PASS_SIZE-1;word[i]>0;i--)  
    if(word[i]>=power)  
    {  
      word[i]-=power;  
      word[i-1]++;  
    }  
}
```

где `rhs` – наша строка. Обратная операция – вывод числа из нашей арифметики в строку:

```
int ariphmetic::ToWord(char * result)  
{ int i,v;  
  char buff[PASS_SIZE];  
  memcpy(buff,word,PASS_SIZE);  
  for(j=0;( j<PASS_SIZE) && (buff[j]==0));j++);  
  if(j==PASS_SIZE) return 0;  
  v=0;  
  for(i=PASS_SIZE-1;i>j;i--)  
  {  
    int c=buff[i]+v;  
    buff[i]=c;  
    if(c<=0)  
    {  
      buff[i]+=power;  
      v=-1;  
    }  
    else  
    {  
      v=0;  
    }  
  }  
  buff[i]+=v;  
  if(buff[i]==0)  
  {  
    j++;  
  }  
  memcpy(result,&buff[j],[PASS_SIZE-j]);  
}
```

Сначала мы определяем с какой позиции в массиве начинается слово и заносим это значение в переменную `j`. Далее проделываем в цикле операцию обратную той, что использовалось в предыдущей функции: если текущая элемент меньше или равен нулю,

прибавляем к нему **power** и вычитаем единицу из следующего разряда.

Предположим нам нужно узнать какая строка будет, к примеру, через 5 000 000 инкрементаций. Благодаря настоящим выкладкам это можно сделать естественнее и быстрее чем циклом и сложением с единицей на каждом итерационном шаге. Ответ на задачу о представлении целого числа в нашей арифметике поможет в дальнейшем в разделении диапазона перебора на равное количество частей. Здесь все очень просто:

```
void ariphmetic::Dec2Word(unsigned int rhs)
{ memset(&word[0],0,PASS_SIZE);
  int i=PASS_SIZE-1;
  do
  {
    if(i==0) break;
    word[i--]=rhs%power;
    rhs=rhs/power;
  }
  while(rhs>0);
}
```

Нет смысла приводить в данном контексте алгоритм деления или умножения двух чисел, потому как это скорее тема длинной арифметики, я приведу лишь пример сложения двух таких чисел:

```
void ariphmetic::Add(const ariphmetic & rhs)
{
  int i;
  int sum,v=0;

  for(i=PASS_SIZE-1;i>0;i--)
  {
    sum=word[i]+rhs.word[i]+v;
    if(sum<power)
    {
      word[i]=sum;
      v=0;
    }
    else
    {
      word[i]=sum-power;
      v=1;
    }
  }
}
```

После реализации всех необходимых арифметических операций, код распределения диапазона на N частей будет выглядеть так:

```
void ReDistribute(const ariphmetic & first, const ariphmetic & last)
{
  int i;
  ariphmetic temp;
  temp=(last-first)/(N);
  for(i=0;i<N-1;i++)
  {
    cl[i]->first=temp*i+first;
  }
}
```

```
cl[i]->last=cl[i]->first+temp-1;  
}  
cl[i]->first=temp*i+first;  
cl[i]->last=last;  
};
```

где, *cl* – массив частей, *last*, *first* – соответственно нижняя и верхняя границы диапазона. Перед вызовом необходимо указать границы:

```
first='\x01';  
last='\x03\x03\x03';
```

## Заключение

Хотелось бы отметить действительную привлекательность предложенного метода, который, при должной реализации, является не только быстрым и удобным, сколько универсальным. Затронутая и применяемая тема длинной арифметики описана мной в заметке «Длинная и быстрая арифметика». Полноценную же библиотеку для языка C++, входящую в состав проекта BruteNet, можно скачать с моего сайта. Все мысли, идеи и пожелания можете отправить на [zaco@yandex.ru](mailto:zaco@yandex.ru) или <http://zaco.itdefence.ru>.