Computing Science 204 (B1)
de Bruijn Sequences
Tuesday, August 8

Let

$$\Sigma = \{0,\, 1,\, \cdots,\, p-1\}$$

be an alphabet of p letters and consider all words of length n from Σ. A (p, n) *de Bruijn sequence* is a sequence

$$a_0\, a_1\, \cdots\, a_{L-1} \qquad (*)$$

with each a_i in Σ such that every word w of length n from Σ is realized as

$$a_i\, a_{i+1}\, \cdots\, a_{i+n-1}$$

for exactly one i, where addition in the subscripts is moldulo L, that is, we wrap around when we come to the end of the sequence $(*)$. De Bruijn sequences are important in coding theory, they are implemented by shift registers and are sometimes called *shift register sequences.*

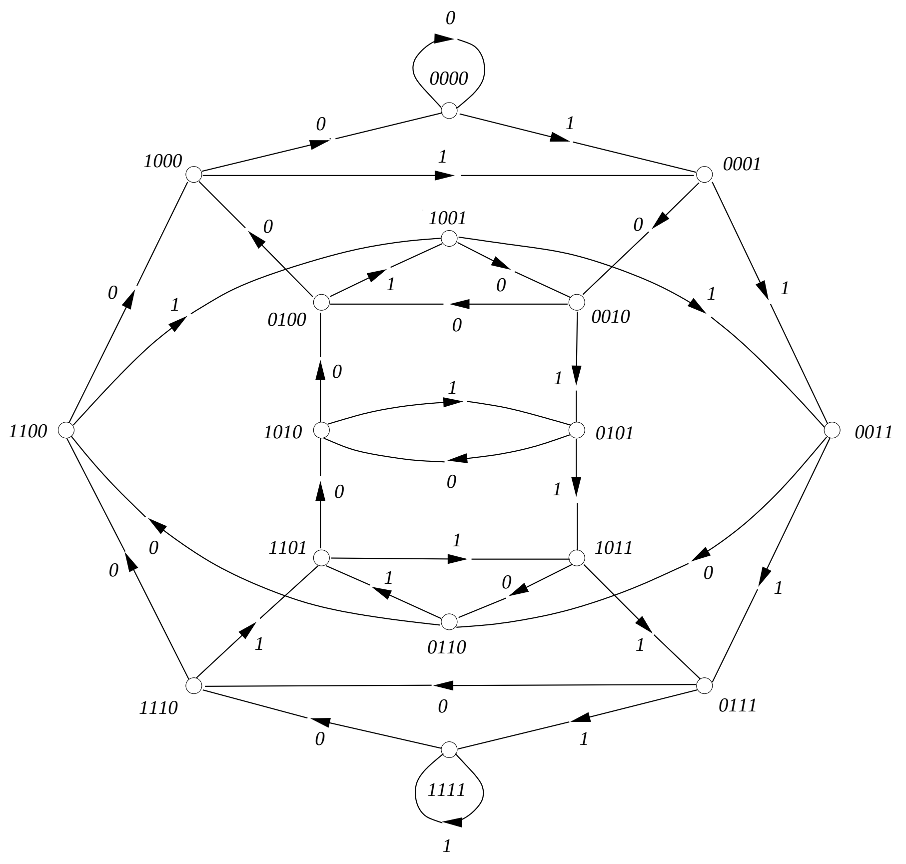
The example we did in class had $p = 2$ and $n = 4$, that is, the alphabet was $\Sigma = \{0, 1\}$ and we were looking for a $(2, 4)$ de Bruijn sequence, that is, one that contained all possible strings of 0's and 1's of length 4. We found the sequence

$$1\,1\,1\,1\;\;0\,1\,1\,0\;\;0\,1\,0\,1\;\;0\,0\,0\,0$$

by explicitly constructing the de Bruijn digraph, and this sequence contains all possible strings of length 4, provided that we wrap around.

We can find a $(2, 5)$ de Bruijn sequence, that is, a sequence of 0's and 1's which contains all possible sequences of length 5, again, allowing wrap around in the same manner.

More generally, the *de Bruijn digraph* has vertex set consisting of all 2^{n-1} words of length $n-1$ over the alphabet $\Sigma = \{0, 1\}$, and its arc set is the set of all ordered pairs of the form $(a_1 a_2 \cdots a_{n-1}, a_2 a_3 \cdots a_n)$. Thus, there is an arc joining the two vertices A and B if and only if the right–most $n-2$ bits in A are the same as the left–most $n-2$ bits of B, and the arc is labeled with the right–most bit of the head of the arc AB.



By following an Eulerian circuit in this digraph, that is, a circuit that traverses each edge exactly once, we can generate a $(2, n)$ de Bruijn sequence by listing the label of each arc as we traverse the Eulerian circuit. For $n = 5$ the de Bruijn digraph is shown above, and we traverse the Eulerian circuit given by

$$\begin{aligned}
&1001 \to 0010 \to 0100 \to 1000 \to 0000 \to 0000 \to 0001 \to 0011 \to \\
&0111 \to 1111 \to 1111 \to 1110 \to 1101 \to 1011 \to 0111 \to 1110 \to \\
&1100 \to 1001 \to 0011 \to 0110 \to 1101 \to 1010 \to 0101 \to 1011 \to \\
&0110 \to 1100 \to 1000 \to 0001 \to 0010 \to 0101 \to 1010 \to 0100 \to \\
&1001
\end{aligned}$$

This generates the $(2, 5)$ de Bruijn sequence

$$0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,0\,0\,1\,0\,1\,0\,0\,1$$

Given a positive integer n, there is an explicit method for generating a $(2, n)$ de Bruijn sequence, namely, start with n 0's and add a 1 to the sequence provided it does not repeat a previous string of length n, otherwise add a 0. This algorithm is very easily implemented to generate any $(2, n)$ de Bruijn sequence and does not require that the de Bruijn digraph be constructed.

In order to generate a $(2, n)$ de Bruijn sequence without constructing the digraph, we first implement the Knuth-Morris-Pratt pattern–matching algorithm This algorithm takes as its input a pattern to be searched for in a string of text, and outputs a boolean value `true` or `false` depending on whether the pattern occurs in the text string or not.

The Knuth-Morris-Pratt algorithm is implemented below.

```
#include<stdio.h>

char *p,*t;
int m,n,*fail;

read_data()
{
   int i,j;
   printf("Enter the number (m) of characters in the pattern P:\n");
   scanf("%d",&m);
   p = (char *)malloc((m+1)*sizeof(char));
   printf("Enter the pattern P to be found:\n");
   for (i = 0; i < m; i++)
       scanf(" %c",&p[i]);
   printf("Enter the number (n) of characters in the text string T:\n");
   scanf("%d",&n);
   t = (char *)malloc((n+1)*sizeof(char));
   printf("Enter the text string T to be searched:\n");
   for (i = 0; i < n; i++)
       scanf(" %c",&t[i]);
}
```

```
kmp_setup()
{
   int k, r;
   fail = (int *)malloc((m+1)*sizeof(int));
   k = 0; r = -1; fail[0] = -1;
   do
   {
       if ((r == -1) || (r >= 0 && p[k] == p[r]))
       {
           k++;
           r++;
           if (p[r] != p[k])
               fail[k] = r;
           else
               fail[k] = fail[r];
       }
       else
           r = fail[r];
   }
   while (r < m && k < m);
}

int kmp_match()
{
   int j,k;
   j = 0;
   k = 0;
   while (j < n && k < m)
   {
       while ((k >= 0) && (t[j] != p[k]))
           k = fail[k];
       j++;
       k++;
   }
   if (k == m)
       return(j - m + 1);
   else
       return(j);
}

main()
{
   int index;
   read_data();
   kmp_setup();
   index = kmp_match();
   if (index > n - 1)
       printf("No match was found");
   else
       printf("The match occurs at %d\n",index);
   printf("\n");
}
```

Now the Knuth-Morris-Pratt algorithm is used to implement the program below for generating a $(2, n)$ deBruijn sequence for any positive integer n.

```
#include<stdio.h>
int m,n;

get_word_size()
{
   printf("Enter the word size m:\n");
   scanf("%d",&m);
   n = pow2(m);
}

get_fail(p,f)
int *p,*f;
{
   int i,j;
   i = 0; j = -1; f[0] = -1;
   do
   {
       if ((j == -1) || (j >= 0 && p[i] == p[j]))
       {
           i++;
           j++;
           if (p[j] != p[i])
               f[i] = j;
           else
               f[i] = f[j];
       }
       else
           j = f[j];
   }while (i < m && j < m);
}

int match_pattern(t,p,f,k)
int *t,*p,*f,k;
{
   int i,j,index;
   i = 0;
   j = 0;
   while (i < m && j < k)
   {
       while ((i >= 0) && (t[j] != p[i]))
           i = f[i];
       i++;
       j++;
   }
   if (i == m)
       index = j - m + 1;
   else
       index = j;
   return index;
}
```

```
build_deBruijn(t,f)
int *t,*f;
{
   int i,j,k,*p;
/* uses get_fail() and  match_pattern() to get deBruijn sequence */
   t = (int *)malloc(n*sizeof(int));
   p = (int *)malloc(m*sizeof(int));
   f = (int *)malloc(m*sizeof(int));
   for (k = 0; k < n; k++)
       t[k] = 0;
   for (k = 0; k < n-m; k++)
   {
       p[m-1] = 1;
       for (j = 0; j < m-1; j++)
           p[j] = t[k+j+1];
       get_fail(p,f);
       if (match_pattern(t,p,f,m+k) > m+k-1)
           t[m+k] = 1;
       else
           t[m+k] = 0;
   }
   printf("A (2,%d) deBruijn sequence is: \n",m);
   for (k = 0; k < n; k++)
       printf("%d",t[k]);
   printf("\n");
}

int pow2(i)
int i;
{
   int j, pow2;
   pow2 = 1;
   for (j = 0; j < i; j++)
       pow2 = 2*pow2;
   return pow2;
}

main()
{
   int *text,*fail;
   get_word_size();
   build_deBruijn(text,fail);
}
```