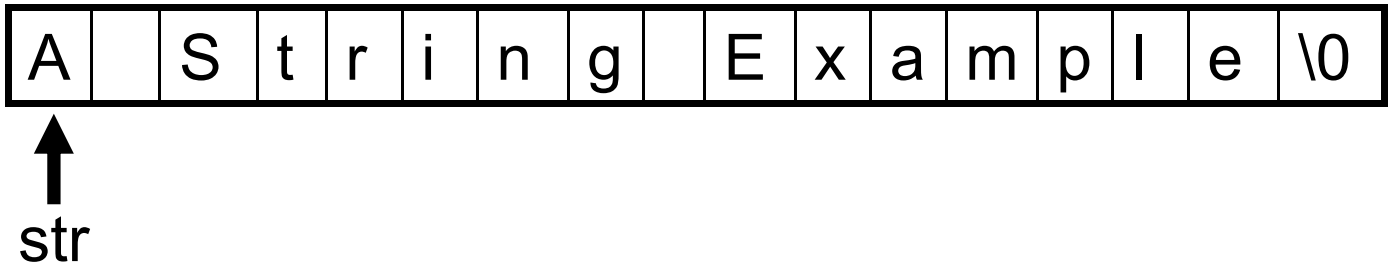
Обработка на низове

Низ е линейна последователност от символи.

Видове низове

1. **Текстов низ – последователност от букви, цифри и специални символи.**
2. **Двоичен низ – последователност от 0 и 1.**

```
char s[81];
char str[] = ″A String Example″;
```

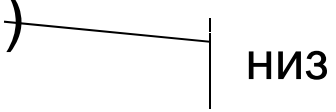


Дължина на низ

Алгоритъм

брой ← **0**
докато не е достигнат краят на низа
повтаряй
  брой ← брой **+ 1**
  преместване към следващия символ в низа

```
int length (char *s)
{
   int n = 0;
   while (*s != '\0')
   {
      n++;
      s++;
   }
   return n;
}
```

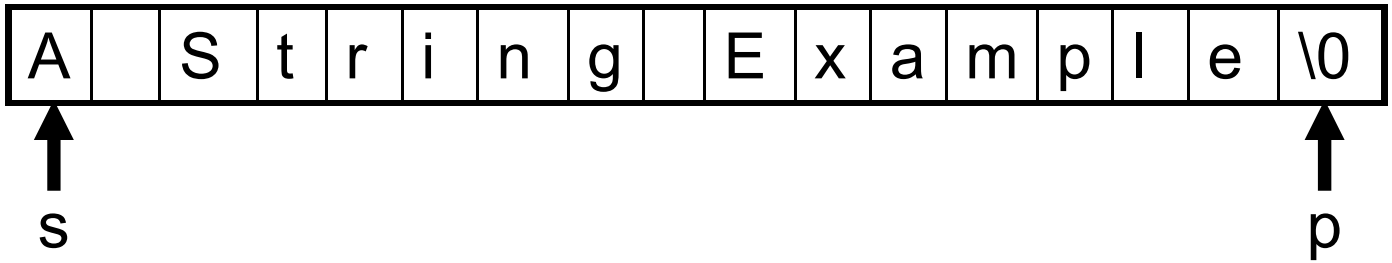


Алгоритъм

указател p ← s
докато p не е достигнал края на низа
повтаряй
  преместване на p към следващия символ в низа
брой ← p - s

```
int length (char *s)
{
   char *p = s;
   while (*p != '\0')
      p++;
   return p - s;
}
```

низ



Копиране на низ

Алгоритъм

копира символ от s2 в s1 и докато копираният символ не е достигнал края на s2
повтаряй
преместване на s1 към следващия символ в низа
преместване на s2 към следващия символ в низа

```
// Копира s2 в s1 – Вариант 1
void copy (char *s1, char *s2)
{
   while ((*s1 = *s2) != '\0')
   {
      s1++;
      s2++;
   }
}
```

```
// Копира s2 в s1 – Вариант 2
void copy (char *s1, char *s2)
{
   while ((*s1++ = *s2++) != '\0')
      ;
}
```

Сравняване на низове

Алгоритъм

докато съответстващите символ от s1 и s2 съвпадат
повтаряй
ако съвпадащият символ е край на низ
s1 и s2 съвпадат
преместване на s1 към следващия символ в низа
преместване на s2 към следващия символ в низа
разлика между несъвпадащите символи

```
//              < 0, ако s1 < s2;
// Връща = 0, ако s1==s2;
//              > 0, ако s1>s2.
int compare (char *s1, char *s2)
{
   while (*s1 == *s2)
   {
      if (*s1 == '\0')
           return 0;
      s1++;
      s2++;
   }
   return *s1 - *s2;
}
```

Търсене на шаблон

Текст

```
A STRING SEARCHING EXAMPLE CONSISTING OF ...
```

Шаблон

```
STING
```

Алгоритъм с принудително преместване

Алгоритъмът използва указател pp**, който сочи към шаблона, указател** pt**, който сочи към текста и указател** back**, който определя позицията за принудително връщане назад. Докато двата указателя** pp **и** pt **сочат към съвпадащи символи, и двата указателя напредват. При несъвпадение** pt **се връща принудително назад, като мястото** back **всеки път се премества с една позиция надясно в текста, а** pp **– към началото на шаблона. Намира позицията на първото срещане на шаблона в текста или -1 при неуспешна операция.**

Алгоритъм

M ← **дължина на шаблон**
pp ← **шаблон**
pt ← **текст**
back ← **текст**
докато не е достигнат краят на шаблона и края на текста
повтаряй
  ако символ(текст) ≠ **символ(шаблон)**
    back **напредва надясно в текста**
    pt ← back
    pp ← **шаблона**
  в противен случай
    pp **напредва**
    pt **напредва**
ако е достигнат края на шаблона
  съвпадението започва в текуща позиция(текст) - M
в противен случай
  няма съвпадение

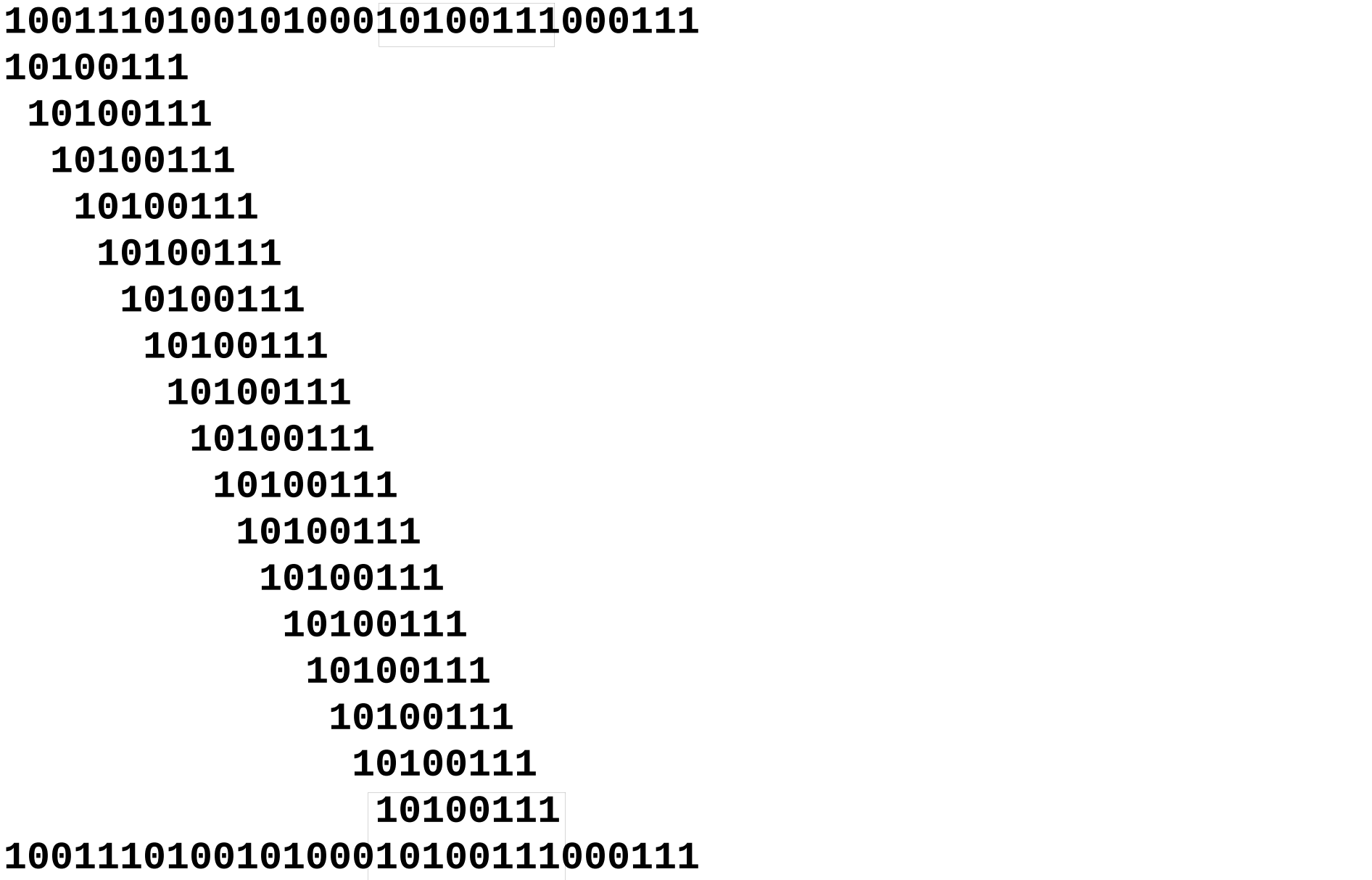
позиция

шаблон

текст

```
int search1 (char *p, char *t)
{  char *pp, *pt, *back;
   int M = strlen (p);          // дължина на шаблона
   pp = p;                      // сочи към начало на шаблона
   pt = t;                      // сочи към начало на текста
   back = t;                    // сочи към начало на текста
   while (*pp != '\0' && *pt != '\0')
      if (*pp != *pt)      // несъвпадение
      {
          back++;          // преместване надясно в текста
          pt = back;       // връщане назад в текста
          pp = p;          // преместване в началото на шаблона
      }
      else
      {
          pp++;            // двата указателя напредват
          pt++;
      }
```

```
    if (*pp == '\0')                // шаблонът е намерен
        return pt-t-M;
    else
        return -1;                  // няма съвпадение
}
```



Търсене в двоичен низ

i=16

Методът с принудително преместване за търсене на шаблон може да изисква MN **сравнения на символи.**

M – **дължина на шаблона**

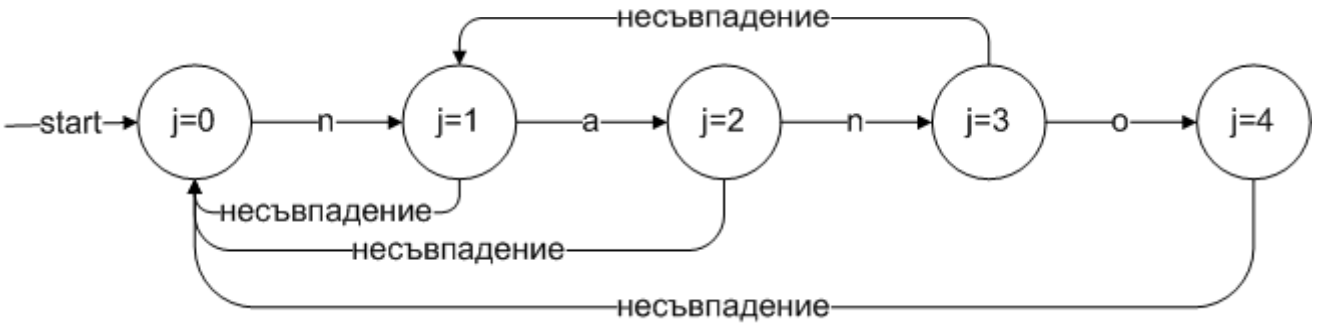
N – **дължина на текста**

Алгоритъм на Knuth-Morris-Pratt

Подобрява дължината на преместването.
Извършва се предварителна обработка.
Изчислява се функция на грешката failure**, която показва подходящото преместване, така че да се използват предварителните сравнения.**
Шаблонът p **се претърсва сам върху себе си. При несъвпадение в масив** f **се записва броят на застъпващите се символи. Приема се, че** f[0]=0**.**

Ако при търсене на шаблона p **в текста** t **се получи несъвпадение** t[i]≠p[j]**, то позицията в текста** i **остава непроменена и се установява позицията в шаблона** j **да бъде равна на** f[j]**.**

Алгоритъмът е по-бърз, тъй като не се връщаме принудително назад.



int failure[] = {0, 0, 1, 0};

```
int main()
{
        char t[] = "ancnahtnanofdnansgtna";
        char p[] = "nano";

        int i = kmpSearch(t, p);
        int i = kmpSearchFSM(t);

}
```

Алгоритъм за изчисляване на функцията на грешката

```
i ← 1
i ←  0
f(0) ← 0
m ← дължина на шаблон
докато не е достигнат краят на шаблона повтаряй
   ако p[j]=p[i]                  // съвпадение на j+1 символа
      f(i) ← j + 1
      i ←  i + 1
      j ←  j + 1
   в противен случай
      ако j > 0                   // j е точно след съвпадащите символи
         j ← f(j – 1)
   в противен случай    // няма съвпадение
      f(i) ← 0
      i ←  i + 1
```

```
void failure(char *p, int *f)
{       f[0] = 0;
        int m = strlen(p);
        int j = 0;
        int i = 1;
        while(i < m)
        {
                if(p[j]==p[i])
                {
                        f[i] = j+1;
                        i++;
                        j++;
                }
                else if(j>0)
                        j = f[j-1];
                else
                {
                        f[i] = 0;
                        i++;
                }
        }
}
```

Алгоритъм на Knuth-Morris-Pratt

```
i ← 0
i ←  0
f ← функция на грешката(p)
m ← дължина на шаблон
n ← дължина на текст
докато не е достигнат краят на текста повтаряй
   ако p[j]=t[i]                    // съвпадение
      ако j = m – 1
        съвпадение започва в i - m + 1
      i ←  i + 1
      j ←  j + 1
   в противен случай
      ако j > 0                     // напреднали сме в шаблона
         j ← f(j – 1)               // j е точно след съвпадащите символи
   в противен случай   // няма съвпадение
      i ←  i + 1
шаблонът не се съдържа в текста
```

```
int kmpSearch(char *t, char *p)
{       int n = strlen(t);
        int m = strlen(p);
        int *f = (int *)malloc(m*sizeof(int));
        failure(p, f);
        int i = 0;
        int j = 0;
        while(i<n)
        {       if(p[j]==t[i])
                {       if(j==m-1)
                                return i-m+1;
                        i++;
                        j++;
                }
                else
                        if(j>0)
                                j = f[j-1];
                        else
                                i++;
        }
        return -1;
}
```

```
int kmpSearchFSM(char *t)
{       enum fsm {j0, j1, j2, j3, j4};
        int n = strlen(t);
        enum fsm state = j0;
        for(int i =0; i < n; i++)
        {
                switch(state)
                {
                case j0: if(t[i] == 'n') state = j1; else state = j0;
                        break;
                case j1: if(t[i] == 'a‘) state = j2; else state = j0;
                        break;
                case j2: if(t[i] == 'n') state = j3; else state = j1;
                        break;
                case j3: if(t[i] == 'o') state = j4; else state = j0;
                        break;
                case j4:
                        return i - 4;
                }
        }
        return -1;
}
```

Методът на Knuth-Morris-Pratt за търсене на шаблон не изисква повече от M+N **сравнения на символи.**

M – **дължина на шаблона**

N – **дължина на текста**

Подходящ е за обработка на голям файл, който се чете от външно устройство.