



NANJING UNIVERSITY

ACM-ICPC Codebook 0  
**Miscellaneous**

October 16, 2017

# Contents

<b>1</b>	<b>General</b>	<b>4</b>
1.1	Template . . . . .	4
<b>2</b>	<b>String</b>	<b>4</b>
2.1	Knuth-Morris-Pratt algorithm . . . . .	4
2.2	Trie . . . . .	5
2.3	Aho-Corasick automaton . . . . .	6
<b>3</b>	<b>Game Theory</b>	<b>8</b>
3.1	Nim game . . . . .	8
3.1.1	Bash game . . . . .	8
3.1.2	Fibonacci nim . . . . .	8
3.1.3	Wythoff's game . . . . .	8
<b>4</b>	<b>Others</b>	<b>8</b>
4.1	Fast Fourier transform . . . . .	8
4.2	2-SAT . . . . .	10



# 1 General

## 1.1 Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define rep(i, n) for (int i = 0; i < (n); i++)
5 #define Rep(i, n) for (int i = 1; i <= (n); i++)
6 #define range(x) (x).begin(), (x).end()
7 typedef long long LL;
8
9 int main(){
10
11     return 0;
12 }
```

# 2 String

## 2.1 Knuth-Morris-Pratt algorithm

Single-pattern matching.

**Usage:**

construct(p)	Construct the failure table of pattern p.
match(t, p)	Match pattern p in text t.
found(pos)	Report the pattern found at pos.

**Time complexity:**  $O(l)$ .

```
1 const int SIZE = 10005;
2 int fail[SIZE];
3 int len;
4
5 void construct(const char* p){
6     len = strlen(p);
7     fail[0] = fail[1] = 0;
8     for (int i = 1; i < len; i++) {
9         int j = fail[i];
10        while (j && p[i] != p[j]) j = fail[j];
11        fail[i+1] = p[i] == p[j] ? j+1 : 0;
12    }
13 }
```

```

14
15 inline void found(int pos){
16     // ! add codes for having found at pos
17 }
18
19 void match(const char* t, const char* p){ // must be called after construct
20     int n = strlen(t);
21     int j = 0;
22     rep (i, n){
23         while (j && p[j] != t[i]) j = fail[j];
24         if (p[j] == t[i]) j++;
25         if (j == len) found(i - len + 1);
26     }
27 }

```

## 2.2 Trie

Support insertion and search for a set of words.

- △ If duplicate word exists, only the last one is preserved.
- △ The tag must not be 0, which is considered as not being a word.

### Usage:

id(c)	Covert character to its id.
add(s, t)	Add word <i>s</i> into Trie, where <i>t</i> is the tag attached to <i>s</i> .
search(s)	Search for word <i>s</i> . Return the tag attached to <i>s</i> if found; otherwise return 0.

**Time complexity:**  $O(l|\Sigma|)$  for insertion,  $O(l)$  for search.

```

1 const int MAXN = 12000;
2 const int CHARN = 26;
3
4 inline int id(char c){
5     return c - 'a';
6 }
7
8 struct Trie{
9     int n;
10    int tr[MAXN][CHARN]; // Trie tree, 0 denotes fail
11    int tag[MAXN];
12
13    Trie(){
14        memset(tr[0], 0, sizeof(tr[0]));
15        tag[0] = 0; n = 1;
16    }

```

```

17 // tag should not be 0
18 void add(const char* s, int t){
19     int p = 0, c, len = strlen(s);
20     rep (i, len){
21         c = id(s[i]);
22         if (!tr[p][c]){
23             memset(tr[p], 0, sizeof(tr[p]));
24             tag[p] = 0;
25             tr[p][c] = n++;
26         }
27         p = tr[p][c];
28     }
29     tag[p] = t;
30 }
31
32 // returns 0 if not found
33 // AC automaton does not need this function
34 int search(const char* s){
35     int p = 0, c, len = strlen(s);
36     rep (i, len){
37         c = id(s[i]);
38         if (!tr[p][c]) return 0;
39         p = tr[p][c];
40     }
41     return tag[p];
42 }
43 };
44

```

## 2.3 Aho-Corasick automaton

Automaton for multi-pattern matching.

△ See the warnings of Trie.

△ If a word has too many suffixes, the automaton might run slow.

### Usage:

<code>add(s, t)</code>	Add word $s$ into Trie, where $t$ is the tag attached to $s$ .
<code>construct()</code>	Construct the automaton after all words added.
<code>find(text)</code>	Find words in text.
<code>found(pos, j)</code>	Report a word found in node $j$ , the last character of which is at $pos$ .

### Requirement:

2.2 Trie

**Time complexity:**  $O(l|\Sigma|)$  for insertion and construction,  $O(l)$  for finding, provided the number of suffixes of a word is constant.

```

1 struct AC : Trie{
2     int fail[MAXN];
3     int last[MAXN];
4
5     void construct(){
6         queue<int> q;
7         fail[0] = 0;
8         rep (c, CHARN){
9             if (int u = tr[0][c]){
10                 fail[u] = 0;
11                 q.push(u);
12                 last[u] = 0;
13             }
14         }
15         while (!q.empty()){
16             int r = q.front(); q.pop();
17             rep (c, CHARN){
18                 int u = tr[r][c];
19                 if (!u){
20                     tr[r][c] = tr[fail[r]][c];
21                     continue;
22                 }
23                 q.push(u);
24                 int v = fail[r];
25                 while (v && !tr[v][c]) v = fail[v];
26                 fail[u] = tr[v][c];
27                 last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
28             }
29         }
30     }
31
32     void found(int pos, int j){
33         if (j) {
34             // ! add codes for having found word with tag[j]
35             found(pos, last[j]);
36         }
37     }
38
39     void find(const char* text){ // must be called after construct()
40         int p = 0, c, len = strlen(text);
41         rep (i, len){
42             c = id(text[i]);
43             p = tr[p][c];
44             if (tag[p])
45                 found(i, p);

```

```

46         else if (last[p])
47             found(i, last[p]);
48     }
49 }
50 };

```

## 3 Game Theory

以下游戏中，不能动的算输。

### 3.1 Nim game

#### 3.1.1 Bash game

有  $n$  个石子，每人最多拿  $m$  个，最少拿 1 个。 $n \bmod (m + 1) \neq 0$  时先手必胜。

#### 3.1.2 Fibonacci nim

有  $n$  个石子，第一轮可以拿不超过  $n$  个石子。此后，每次拿的石子数不超过前一次的 2 倍。当  $n$  是斐波那契数时先手必胜。

#### 3.1.3 Wythoff's game

有 2 堆石子，分别有  $a, b$  个 ( $a \leq b$ )，每人可以从一堆中拿任意多个，或从两堆中拿相同多个。当  $a = \lfloor (b - a) \frac{\sqrt{5} + 1}{2} \rfloor$  时先手必败。

## 4 Others

### 4.1 Fast Fourier transform

△ The size of the sequence must be some power of 2.

△ When performing convolution, the size of the sequence should be doubled. To compute  $k$ , one may call `32-__builtin_clz(a+b-1)`, where  $a$  and  $b$  are the lengths of two sequences.

**Usage:**



**FFT(k)**            Initialize the structure with maximum sequence length  $2^k$ .  
**fft(a)**            Perform Fourier transform on sequence  $a$ .  
**ifft(a)**           Perform inverse Fourier transform on sequence  $a$ .  
**conv(a, b)**        Convolve sequence  $a$  with  $b$ .

**Time complexity:**  $O(n \log n)$  for **fft**, **ifft** and **conv**.

```

1  const int NMAX = 1<<20;
2  typedef complex<double> cplx;
3  const double PI = 2*acos(0.0);
4  struct FFT{
5      int rev[NMAX];
6      cplx omega[NMAX], oinv[NMAX];
7      int K, N;
8
9      FFT(int k){
10         K = k; N = 1 << k;
11         rep (i, N){
12             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
13             omega[i] = polar(1.0, 2.0 * PI / N * i);
14             oinv[i] = conj(omega[i]);
15         }
16     }
17
18     void dft(cplx* a, cplx* w){
19         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
20         for (int l = 2; l <= N; l *= 2){
21             int m = l/2;
22             for (cplx* p = a; p != a + N; p += l)
23                 rep (k, m){
24                     cplx t = w[N/l*k] * p[k+m];
25                     p[k+m] = p[k] - t; p[k] += t;
26                 }
27         }
28     }
29
30     void fft(cplx* a){dft(a, omega);}
31     void ifft(cplx* a){
32         dft(a, oinv);
33         rep (i, N) a[i] /= N;
34     }
35
36     void conv(cplx* a, cplx* b){
37         fft(a); fft(b);
38         rep (i, N) a[i] *= b[i];
39         ifft(a);
40     }
41 };
  
```

## 4.2 2-SAT

### Usage:

<code>init(n)</code>	Initialize the structure with at most $n$ Boolean variables.
<code>add_clause(x, xval, y, yval)</code>	Add clause: $x = xval$ or $y = yval$ .
<code>solve()</code>	Solve the 2-SAT problem. Return false if no solution.
<code>value(i)</code>	Return the value of $i$ -th variable in some solution, if exists.

**Time complexity:**  $O(m + n)$ .

```

1  const int MAXN = 100005;
2  struct twoSAT{
3      int n;
4      vector<int> G[MAXN*2];
5      bool mark[MAXN*2];
6      int S[MAXN*2], c;
7
8      void init(int n){
9          this->n = n;
10         for (int i=0; i<n*2; i++) G[i].clear();
11         memset(mark, 0, sizeof(mark));
12     }
13
14     bool dfs(int x){
15         if (mark[x^1]) return false;
16         if (mark[x]) return true;
17         mark[x] = true;
18         S[c++] = x;
19         for (int i=0; i<G[x].size(); i++)
20             if (!dfs(G[x][i])) return false;
21         return true;
22     }
23
24     void add_clause(int x, bool xval, int y, bool yval){
25         x = x * 2 + xval;
26         y = y * 2 + yval;
27         G[x^1].push_back(y);
28         G[y^1].push_back(x);
29     }
30
31     bool solve() {
32         for (int i=0; i<n*2; i+=2){
33             if (!mark[i] && !mark[i+1]){
34                 c = 0;
35                 if (!dfs(i)){
36                     while (c > 0) mark[S[--c]] = false;

```

```
37         if (!dfs(i+1)) return false;
38     }
39 }
40 }
41 return true;
42 }
43
44 inline bool value(unsigned i){return mark[2*i+1];}
45 };
```