



NANJING UNIVERSITY

ACM-ICPC Codebook 0
Miscellaneous

July 29, 2018

Contents

1	General	4
1.1	vimrc	4
1.2	bashrc	4
1.3	runbash	4
1.4	Template	5
2	String	6
2.1	Knuth-Morris-Pratt algorithm	6
2.2	Trie	7
2.3	Aho-Corasick automaton	8
2.4	Manacher	10
3	Game Theory	11
3.1	Nim games	11
3.1.1	Bash game	11
3.1.2	Fibonacci nim	11
3.1.3	Wythoff's game	11
4	Others	12
4.1	Fast Fourier transform	12
4.2	2-SAT	13

1 General

1.1 vimrc

```
1 set nocompatible
2 syntax on
3 colorscheme slate
4 set number
5 set cursorline
6 set shiftwidth=2
7 set softtabstop=2
8 set tabstop=2
9 set expandtab
10 set magic
11 set smartindent
12 set backspace=indent,eol,start
13 set cmdheight=1
14 set laststatus=2
15 set statusline=\ %<%F[%1*%M%*%n%R%H]%=\ %y\ %0(%{&fileformat}\ %{&encoding}\ %c
    :%l/%L%\
16 set whichwrap=b,s,<,>,[,]
```

1.2 bashrc

```
1 mkdir -p ~/.trash
2 alias rm=trash
3 trash()
4 {
5     mv $@ ~/.trash/
6 }
7
8 cleartrash()
9 {
10     \rm -rvf ~/.trash
11     mkdir -p ~/.trash
12 }
```

1.3 runbash

```
1 if [ $# -ge 1 ]; then
2     fn=$1
3     echo ${fn} > .run.log
```

```

4 else
5     fn=`cat .run.log`
6 fi
7
8 # cat $fn.cpp | xsel -ib
9
10 if g++ $fn.cpp -std=c++11 -D__LOCAL_DEBUG__ -Wall -O2 -g -o $fn; then
11     echo "*****_Compilation_Success!_*****_[$fn]"
12     if [ $# -ge 2 ]; then
13         time -f "\n%U_user,%S_system,%e_real" ./$fn < $2
14     else
15         time -f "\n%U_user,%S_system,%e_real" ./$fn
16     fi
17 # cat $fn.cpp | xsel -ib
18 else
19     echo "*****_Compilation_Failed!_*****_[$fn]"
20 fi

```

1.4 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #ifdef __LOCAL_DEBUG__
5 # define _debug(fmt, ...) fprintf(stderr, "\033[94m%s:_ " fmt "\n\033[0m", \
6     __func__, ##__VA_ARGS__)
7 #else
8 # define _debug(...) ((void) 0)
9 #endif
10 #define rep(i, n) for (int i=0; i<(n); i++)
11 #define Rep(i, n) for (int i=1; i<=(n); i++)
12 #define range(x) (x).begin(), (x).end()
13 typedef long long LL;
14 typedef unsigned long long ULL;
15
16 template <unsigned p>
17 struct Zp{
18     unsigned x;
19     Zp(unsigned x):x(x){}
20     operator unsigned(){return x;}
21     Zp operator ^ (ULL e) {
22         Zp b=x, r=1;
23         while (e) {
24             if (e&1) r=r*b;
25             b=b*b;

```

```

26         e>>=1;
27     }
28     return r;
29 }
30 Zp operator + (Zp rhs) {return (x+rhs)%p;}
31 Zp operator - (Zp rhs) {return (x+p-rhs)%p;}
32 Zp operator * (Zp rhs) {return x*rhs%p;}
33 Zp operator / (Zp rhs) {return Zp(x)*(rhs^(p-2));}
34 };
35
36 typedef Zp<1000000007> zp;
37
38 zp operator"" _ (ULL n){return n;}

```

2 String

2.1 Knuth-Morris-Pratt algorithm

Single-pattern matching.

Usage:

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

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) { return c - 'a'; }
5
6  struct Trie {
7      int n;
8      int tr[MAXN][CHARN]; // Trie tree, 0 denotes fail
9      int tag[MAXN];
10
11      Trie() {
12          memset(tr[0], 0, sizeof(tr[0]));
13          tag[0] = 0;
14          n = 1;
15      }
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[n], 0, sizeof(tr[n]));
24         tag[n] = 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 };

```

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:

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

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();
17         q.pop();
18         rep(c, CHARN) {
19             int u = tr[r][c];
20             if (!u) {
21                 tr[r][c] = tr[fail[r]][c];
22                 continue;
23             }
24             q.push(u);
25             int v = fail[r];
26             while (v && !tr[v][c]) v = fail[v];
27             fail[u] = tr[v][c];
28             last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
29         }
30     }
31 }
32
33 void found(int pos, int j) {
34     if (j) {
35         // ! add codes for having found word with tag[j]
36         found(pos, last[j]);
37     }
38 }
39
40 void find(const char* text) { // must be called after construct()
41     int p = 0, c, len = strlen(text);
42     rep(i, len) {
43         c = id(text[i]);
44         p = tr[p][c];
45         if (tag[p])
46             found(i, p);
47         else if (last[p])
48             found(i, last[p]);
49     }
```

```

50     }
51 };

```

2.4 Manacher

Find maximum palindrome radii for all centers.

Usage:

`init(str)` Run this algorithm on `str`.
`maxpar(l, r)` Query maximal palindrome central region between $[l, r)$.

Time complexity: Linear in length of string.

```

1  struct Manacher {
2      int Len;
3      vector<int> lc;
4      string s;
5
6      void work() {
7          lc[1] = 1;
8          int k = 1;
9
10         for (int i = 2; i <= Len; i++) {
11             int p = k + lc[k] - 1;
12             if (i <= p) {
13                 lc[i] = min(lc[2 * k - i], p - i + 1);
14             } else {
15                 lc[i] = 1;
16             }
17             while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
18             if (i + lc[i] > k + lc[k]) k = i;
19         }
20     }
21
22     void init(const char *tt) {
23         int len = strlen(tt);
24         s.resize(len * 2 + 10);
25         lc.resize(len * 2 + 10);
26         s[0] = '*';
27         s[1] = '#';
28         for (int i = 0; i < len; i++) {
29             s[i * 2 + 2] = tt[i];
30             s[i * 2 + 1] = '#';
31         }
32         s[len * 2 + 1] = '#';
33         s[len * 2 + 2] = '\0';

```

```
34     Len = len * 2 + 2;
35     work();
36 }
37
38 pair<int, int> maxpal(int l, int r) {
39     int center = l + r + 1;
40     int rad = lc[center] / 2;
41     int rmid = (l + r + 1) / 2;
42     int rl = rmid - rad, rr = rmid + rad - 1;
43     if ((r ^ 1) & 1) {
44     } else rr++;
45     return {max(l, rl), min(r, rr)};
46 }
47 };
```

3 Game Theory

3.1 Nim games

以下游戏中，不能动的算输。

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:

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

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

```

1  const int NMAX = 1<<20;
2
3  typedef complex<double> cplx;
4
5  inline cplx operator * (cplx a, cplx b) {
6      double ra = a.real(), rb = b.real(),
7          ia = a.imag(), ib = b.imag();
8      return cplx(ra*ia-rb*ib, ra*ib+rb*ia);
9  }
10
11 const double PI = 2*acos(0.0);
12 struct FFT{
13     int rev[NMAX];
14     cplx omega[NMAX], oinv[NMAX];
15     int K, N;
16
17     FFT(int k){
18         K = k; N = 1 << k;
19         rep (i, N){
20             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
21             omega[i] = polar(1.0, 2.0 * PI / N * i);
22             oinv[i] = conj(omega[i]);
23         }
24     }
25
26     void dft(cplx* a, cplx* w){
27         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
28         for (int l = 2; l <= N; l *= 2){
29             int m = l/2;
30             for (cplx* p = a; p != a + N; p += l)

```

```

31         rep (k, m){
32             cplx t = w[N/l*k] * p[k+m];
33             p[k+m] = p[k] - t; p[k] += t;
34         }
35     }
36 }
37
38 void fft(cplx* a){dft(a, omega);}
39 void ifft(cplx* a){
40     dft(a, oinv);
41     rep (i, N) a[i] /= N;
42 }
43
44 void conv(cplx* a, cplx* b){
45     fft(a); fft(b);
46     rep (i, N) a[i] *= b[i];
47     ifft(a);
48 }
49
50 void convr(cplx* a, cplx* b) {
51     rep (i, N) b[i].imag(a[i]);
52     fft(b);
53     rep (i, N) {
54         cplx lv = b[i], rv = conj(b[N-1-i]);
55         a[i] = (lv * lv + rv * rv) * cplx(0, -0.25);
56     }
57     ifft(a);
58 }
59 };

```

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 };
```