Geimoire'l Standard Code Library*

Shanghai Jiao Tong University

Dated: 2017 年 9 月 24 日

^{*}https://github.com/kzoacn/Grimoire

目录

1	代数	C	5
	1.1	$O(n^2 \log n)$ 求线性递推数列第 n 项	5
	1.2	闪电数论变换与魔力 CRT	6
	1.3	多项式求逆	7
	1.4	多项式除法	8
	1.5	多项式取指数取对数	ę
2	数论		11
	2.1	大整数相乘取模	11
	2.2	线段下整点	11
	2.3	中国剩余定理	11
3	图论		13
	3.1	图论基础	13
	3.2	闪电二分图匹配	13
	3.3	一般图匹配	15
	3.4	一般最大权匹配	17
	3.5	无向图最小割	21
	3.6	最大带权带花树	22
	3.7	必经点 dominator tree	27
4	字符	F 串	31
	4.1	Manacher	31
	4.2	数组版后缀自动机	31
	4.3	指针版后缀自动机	32
	4.4	广义后缀自动机	34

4		Į	目录
5	技巧		37
	5.1	无敌的读入优化	37
	5.2	真正释放 STL 内存	38
	5.3	梅森旋转算法	38
	5.4	32-bit/64-bit 随机素数	38
	5.5	NTT 素数及其原根	38

Chapter 1

代数

$O(n^2 \log n)$ 求线性递推数列第 n 项

```
Given a_0, a_1, \cdots, a_{m-1}

a_n = c_0 * a_{n-m} + \cdots + c_{m-1} * a_0

a_0 is the nth element, \cdots, a_{m-1} is the n+m-1th element
```

```
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
       long long v[M] = \{1 \% p\}, u[M << 1], msk = !!n;
2
       for(long long i(n); i > 1; i >>= 1) {
3
           msk <<= 1;
       for(long long x(0); msk; msk >>= 1, x <<= 1) {
6
7
           fill_n(u, m << 1, 0);
           int b(!!(n & msk));
8
           x \mid = b;
9
           if(x < m) {
               u[x] = 1 \% p;
11
12
               for(int i(0); i < m; i++) {</pre>
13
                    for(int j(0), t(i + b); j < m; j++, t++) {
14
                        u[t] = (u[t] + v[i] * v[j]) % p;
15
                    }
16
               }
               for(int i((m << 1) - 1); i >= m; i--) {
18
                    for(int j(0), t(i - m); j < m; j++, t++) {
19
                        u[t] = (u[t] + c[j] * u[i]) % p;
20
                    }
21
               }
22
           }
23
           copy(u, u + m, v);
24
25
       //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
26
       for(int i(m); i < 2 * m; i++) {</pre>
27
           a[i] = 0;
28
```

6 CHAPTER 1. 代数

```
for(int j(0); j < m; j++) {
29
                a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
30
31
32
       for(int j(0); j < m; j++) {
33
           b[i] = 0;
34
            for(int i(0); i < m; i++) {</pre>
35
                b[j] = (b[j] + v[i] * a[i + j]) % p;
36
            }
37
38
       for(int j(0); j < m; j++) {
39
40
           a[j] = b[j];
       }
41
42 | }
```

闪电数论变换与魔力 CRT

```
| \text{define meminit}(A, 1, r) \text{ memset}(A + (1), 0, \text{sizeof}(*A) * ((r) - (1))) |
  #define memcopy(B, A, 1, r) memcpy(B, A + (1), sizeof(*A) * ((r) - (1)))
2
  void DFT(int *a, int n, int f) { //f=1 逆 DFT
       for (register int i = 0, j = 0; i < n; i++) {
           if (i > j) std::swap(a[i], a[j]);
           for (register int t = n >> 1; (j ^= t) < t; t >>= 1);
6
7
      for (register int i = 2; i <= n; i <<= 1) {
8
           static int exp[MAXN];
9
           exp[0] = 1; exp[1] = fpm(PRT, (MOD - 1) / i, MOD);
           if (f == 1) \exp[1] = fpm(\exp[1], MOD - 2, MOD);
11
           for (register int k = 2; k < (i >> 1); k++) {
12
               \exp[k] = 111 * \exp[k - 1] * \exp[1] % MOD;
13
14
           for (register int j = 0; j < n; j += i) {
15
               for (register int k = 0; k < (i >> 1); k++) {
16
                   register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
                   register long long B = 111 * pB * exp[k];
18
                   pB = (pA - B) \% MOD;
19
                   pA = (pA + B) \% MOD;
20
               }
21
           }
22
23
       if (f == 1) {
24
           register int rev = fpm(n, MOD - 2, MOD);
25
           for (register int i = 0; i < n; i++) {
26
               a[i] = 111 * a[i] * rev % MOD;
27
               if (a[i] < 0) { a[i] += MOD; }</pre>
28
```

1.3. 多项式求逆 7

```
}
29
      }
30
31 | }
  // 在不写高精度的情况下合并 FFT 所得结果对 MOD 取模后的答案
32
33 // 值得注意的是,这个东西不能最后再合并,而是应该每做一次多项式乘法就 CRT 一次
  int CRT(int *a) {
      static int x[3];
35
      for (int i = 0; i < 3; i++) {
36
          x[i] = a[i];
37
          for (int j = 0; j < i; j++) {
38
              int t = (x[i] - x[j] + FFT[i] \rightarrow MOD) \% FFT[i] \rightarrow MOD;
39
              if (t < 0) t += FFT[i] -> MOD;
40
              x[i] = 1LL * t * inv[j][i] % FFT[i] -> MOD;
41
          }
42
      }
43
      int sum = 1, ret = x[0] % MOD;
      for (int i = 1; i < 3; i ++) {
45
          sum = 1LL * sum * FFT[i - 1] \rightarrow MOD % MOD;
46
          ret += 1LL * x[i] * sum % MOD;
47
          if(ret >= MOD) ret -= MOD;
48
49
      return ret;
50
51 }
  for (int i = 0; i < 3; i++) // inv 数组的预处理过程, inverse(x, p) 表示求 x 在 p 下逆元
52
      for (int j = 0; j < 3; j++)
53
          inv[i][j] = inverse(FFT[i] -> MOD, FFT[j] -> MOD);
```

多项式求逆

Given polynomial a and n, b is the polynomial such that $a * b \equiv 1 \pmod{x^n}$

```
void getInv(int *a, int *b, int n) {
      static int tmp[MAXN];
2
      b[0] = fpm(a[0], MOD - 2, MOD);
3
      for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
4
5
           for (; M \le 3 * (c - 1); M \le 1);
           meminit(b, c, M);
6
           meminit(tmp, c, M);
7
           memcopy(tmp, a, 0, c);
8
           DFT(tmp, M, 0);
9
10
           DFT(b, M, 0);
           for (int i = 0; i < M; i++) {
11
               b[i] = 111 * b[i] * (211 - 111 * tmp[i] * b[i] % MOD + MOD) % MOD;
12
13
           DFT(b, M, 1);
14
           meminit(b, c, M);
15
```

8 CHAPTER 1. 代数

```
16 }
17 }
```

多项式除法

d is quotient and r is remainder

```
void divide(int n, int m, int *a, int *b, int *d, int *r) { // n \setminus m 分别为多项式 A (被除数)
     \rightarrow 和 B (除数) 的指数 + 1
      static int M, tA[MAXN], tB[MAXN], inv[MAXN], tD[MAXN];
2
      for (; n > 0 \&\& a[n - 1] == 0; n--);
3
      for (; m > 0 \&\& b[m - 1] == 0; m--);
       for (int i = 0; i < n; i++) tA[i] = a[n - i - 1];
5
       for (int i = 0; i < m; i++) tB[i] = b[m - i - 1];
6
       for (M = 1; M \le n - m + 1; M \le 1);
7
      if (m < M) meminit(tB, m, M);</pre>
8
       getInv(tB, inv, M);
9
      for (M = 1; M \le 2 * (n - m + 1); M \le 1);
10
      meminit(inv, n - m + 1, M);
11
      meminit(tA, n - m + 1, M);
12
      DFT(inv, M, 0);
13
      DFT(tA, M, 0);
14
       for (int i = 0; i < M; i++) {
15
           d[i] = 111 * inv[i] * tA[i] % MOD;
16
17
      DFT(d, M, 1);
18
19
       std::reverse(d, d + n - m + 1);
      for (M = 1; M \le n; M \le 1);
20
      memcopy(tB, b, 0, m);
21
       if (m < M) meminit(tB, m, M);</pre>
22
      memcopy(tD, d, 0, n - m + 1);
23
      meminit(tD, n - m + 1, M);
24
      DFT(tD, M, 0);
      DFT(tB, M, 0);
26
      for (int i = 0; i < M; i++) {
27
           r[i] = 111 * tD[i] * tB[i] % MOD;
28
29
      DFT(r, M, 1);
30
      meminit(r, n, M);
31
       for (int i = 0; i < n; i++) {
32
           r[i] = (a[i] - r[i] + MOD) % MOD;
33
34
35 | }
```

1.5. 多项式取指数取对数 9

多项式取指数取对数

Given polynomial a and n, b is the polynomial such that $b \equiv e^a \pmod{x^n}$ or $b \equiv \ln a \pmod{x^n}$

```
void getDiff(int *a, int *b, int n) { // 多项式取微分
1
      for (int i = 0; i + 1 < n; i++) {
2
          b[i] = 111 * (i + 1) * a[i + 1] % MOD;
3
      b[n - 1] = 0;
5
6
  |}
  void getInt(int *a, int *b, int n) { // 多项式取积分,积分常数为 0
7
8
      static int inv[MAXN];
      inv[1] = 1;
9
      for (int i = 2; i < n; i++) {
           inv[i] = 111 * (MOD - MOD / i) * inv[MOD % i] % MOD;
11
12
      b[0] = 0;
13
      for (int i = 1; i < n; i++) {
14
          b[i] = 111 * a[i - 1] * inv[i] % MOD;
15
16
  }
17
  void getLn(int *a, int *b, int n) {
18
      static int inv[MAXN], d[MAXN];
19
      int M = 1;
20
      for (; M <= 2 * (n - 1); M <<= 1);
21
22
      getInv(a, inv, n);
      getDiff(a, d, n);
23
      meminit(d, n, M);
      meminit(inv, n, M);
25
      DFT(d, M, 0); DFT(inv, M, 0);
26
      for (int i = 0; i < M; i++) {
27
           d[i] = 111 * d[i] * inv[i] % MOD;
28
29
      DFT(d, M, 1);
30
      getInt(d, b, n);
31
  }
32
33
  void getExp(int *a, int *b, int n) {
      static int ln[MAXN], tmp[MAXN];
34
      b[0] = 1;
35
      for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
36
           for (; M <= 2 * (c - 1); M <<= 1);
37
          int bound = std::min(c, n);
38
          memcopy(tmp, a, 0, bound);
39
          meminit(tmp, bound, M);
40
          meminit(b, c, M);
41
          getLn(b, ln, c);
42
          meminit(ln, c, M);
43
```

10 CHAPTER 1. 代数

```
DFT(b, M, 0);
44
           DFT(tmp, M, 0);
45
           DFT(ln, M, 0);
46
           for (int i = 0; i < M; i++) {
47
               b[i] = 111 * b[i] * (111 - ln[i] + tmp[i] + MOD) % MOD;
48
           }
49
          DFT(b, M, 1);
50
           meminit(b, c, M);
51
       }
52
53 }
```

Chapter 2

数论

大整数相乘取模

```
// x 与 y 须非负
long long mult(long long x, long long y, long long MODN) {
    long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) % MODN;
    return t < 0 ? t + MODN : t;
}
```

线段下整点

```
solve for \sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor, \, n,m,a,b>0
```

```
LL solve(LL n,LL a,LL b,LL m){
    if(b==0) return n*(a/m);
    if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
    if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
    return solve((a+b*n)/m,(a+b*n)%m,m,b);
}
```

中国剩余定理

first is remainder, second is module

```
inline void fix(LL &x, LL y) {
      x = (x \% y + y) \% y;
2
 }
3
 bool solve(int n, std::pair<LL, LL> a[],
                    std::pair<LL, LL> &ans) {
5
      ans = std::make_pair(1, 1);
6
7
      for (int i = 0; i < n; ++i) {
          LL num, y;
8
9
          euclid(ans.second, a[i].second, num, y);
```

12 CHAPTER 2. 数论

```
LL divisor = std::__gcd(ans.second, a[i].second);
10
          if ((a[i].first - ans.first) % divisor) {
11
               return false;
12
          }
13
          num *= (a[i].first - ans.first) / divisor;
14
          fix(num, a[i].second);
15
          ans.first += ans.second * num;
16
          ans.second *= a[i].second / divisor;
17
          fix(ans.first, ans.second);
18
19
      return true;
20
21 }
```

Chapter 3

图论

图论基础

```
struct Graph { // Remember to call .init()!
       int e, nxt[M], v[M], adj[N], n;
2
3
      bool base;
       __inline void init(bool _base, int _n = 0) {
           assert(n < N);</pre>
           n = _n; base = _base;
6
           e = 0; memset(adj + base, -1, sizeof(*adj) * n);
7
8
      __inline int new_node() {
9
           adj[n + base] = -1;
10
           assert(n + base + 1 < N);
11
           return n++ + base;
12
13
       __inline void ins(int u0, int v0) { // directional
14
           assert(u0 < n + base && v0 < n + base);
15
           v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
16
           assert(e < M);</pre>
17
18
       __inline void bi_ins(int u0, int v0) { // bi-directional
19
           ins(u0, v0); ins(v0, u0);
20
      }
21
22 | };
```

闪电二分图匹配

```
int matchx[N], matchy[N], level[N];
vector<int> edge[N];
bool dfs(int x) {
   for (int i = 0; i < (int)edge[x].size(); ++i) {
     int y = edge[x][i];</pre>
```

```
int w = matchy[y];
6
           if (w == -1 \mid | level[x] + 1 == level[w] && dfs(w)) {
7
                matchx[x] = y;
8
9
                matchy[y] = x;
                return true;
10
           }
11
12
       level[x] = -1;
13
       return false;
14
15 }
  int solve() {
16
       memset(matchx, -1, sizeof(*matchx) * n);
17
       memset(matchy, -1, sizeof(*matchy) * m);
18
       for (int ans = 0; ; ) {
19
           std::vector<int> q;
20
           for (int i = 0; i < n; ++i) {
21
                if (matchx[i] == -1) {
22
                    level[i] = 0;
23
                    q.push_back(i);
24
25
                } else {
                    level[i] = -1;
26
                }
27
           }
28
           for (int head = 0; head < (int)q.size(); ++head) {</pre>
29
                int x = q[head];
30
                for (int i = 0; i < (int)edge[x].size(); ++i) {</pre>
31
                    int y = edge[x][i];
32
                    int w = matchy[y];
33
                    if (w != -1 \&\& level[w] < 0) {
34
                         level[w] = level[x] + 1;
35
                         q.push_back(w);
36
                    }
                }
38
           }
39
           int delta = 0;
40
           for (int i = 0; i < n; ++i) {
41
                if (matchx[i] == -1 && dfs(i)) {
42
                    delta++;
43
                }
44
           }
           if (delta == 0) {
46
                return ans;
47
           } else {
48
                ans += delta;
49
           }
50
       }
51
```

3.3. 一般图匹配 15

52 | }

一般图匹配

```
// 0-base, match[u] is linked to u
  vector<int> lnk[MAXN];
int match[MAXN], Queue[MAXN], pred[MAXN], base[MAXN], head, tail, sta, fin, nbase;
4 bool inQ[MAXN], inB[MAXN];
5 inline void push(int u) {
      Queue[tail++] = u; inQ[u] = 1;
6
  }
7
  inline int pop() {
      return Queue[head++];
9
  }
10
  inline int FindCA(int u, int v) {
11
      static bool inP[MAXN];
12
      fill(inP, inP + n, false);
13
      while (1) {
14
           u = base[u]; inP[u] = 1;
           if(u == sta) break;
16
           u = pred[match[u]];
17
18
      while (1) {
19
20
           v = base[v];
           if (inP[v]) break;
21
           v = pred[match[v]];
23
      return v;
24
25 }
  inline void RT(int u) {
26
      int v;
27
      while (base[u] != nbase) {
28
           v = match[u];
29
           inB[base[u]] = inB[base[v]] = 1;
           u = pred[v];
31
           if (base[u] != nbase) pred[u] = v;
32
      }
33
  }
34
  inline void BC(int u, int v) {
35
      nbase = FindCA(u, v);
36
      fill(inB, inB + n, 0);
37
      RT(u); RT(v);
38
      if (base[u] != nbase) pred[u] = v;
39
      if (base[v] != nbase) pred[v] = u;
40
      for (int i = 0; i < n; ++i)
41
```

```
if (inB[base[i]]) {
42
               base[i] = nbase;
43
               if (!inQ[i]) push(i);
44
           }
45
  }
46
  bool FindAP(int u) {
47
       bool found = false;
48
       for (int i = 0; i < n; ++i) {
49
           pred[i] = -1; base[i] = i; inQ[i] = 0;
50
51
       sta = u; fin = -1; head = tail = 0; push(sta);
52
       while (head < tail) {</pre>
53
           int u = pop();
54
           for (int i = (int)lnk[u].size() - 1; i >= 0; --i) {
55
               int v = lnk[u][i];
56
               if (base[u] != base[v] && match[u] != v) {
57
                    if (v == sta \mid \mid match[v] >= 0 && pred[match[v]] >= 0) BC(u, v);
58
                    else if (pred[v] == -1) {
59
                        pred[v] = u;
60
                        if (match[v] >= 0) push(match[v]);
61
                        else {
62
                             fin = v;
63
                             return true;
64
                        }
65
                    }
66
               }
67
           }
68
69
       return found;
70
71
  |}
  inline void AP() {
72
       int u = fin, v, w;
73
       while (u >= 0) {
74
           v = pred[u]; w = match[v];
75
           match[v] = u; match[u] = v;
76
77
           u = w;
78
       }
  }
79
  inline int FindMax() {
80
       for (int i = 0; i < n; ++i) match[i] = -1;
81
       for (int i = 0; i < n; ++i)
82
           if (match[i] == -1 && FindAP(i)) AP();
83
       int ans = 0;
84
       for (int i = 0; i < n; ++i) {
85
           ans += (match[i] !=-1);
86
       }
87
```

3.4. 一般最大权匹配 17

```
88    return ans;
89 }
```

一般最大权匹配

```
//maximum weight blossom, change g[u][v].w to INF - g[u][v].w when minimum weight blossom

    → is needed

2 //type of ans is long long
  //replace all int to long long if weight of edge is long long
  struct WeightGraph {
5
      static const int INF = INT_MAX;
7
      static const int MAXN = 400;
       struct edge{
8
           int u, v, w;
9
           edge() {}
10
           edge(int u, int v, int w): u(u), v(v), w(w) {}
11
      };
       int n, n_x;
13
       edge g[MAXN * 2 + 1][MAXN * 2 + 1];
14
       int lab[MAXN * 2 + 1];
15
       int match [MAXN * 2 + 1], slack [MAXN * 2 + 1], st [MAXN * 2 + 1], pa [MAXN * 2 + 1];
16
       int flower_from[MAXN * 2 + 1] [MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
17
      vector<int> flower[MAXN * 2 + 1];
18
      queue<int> q;
19
       inline int e_delta(const edge &e){ // does not work inside blossoms
20
           return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
21
22
       inline void update_slack(int u, int x){
23
           if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))</pre>
               slack[x] = u;
25
26
       inline void set_slack(int x){
27
           slack[x] = 0;
28
           for(int u = 1;u <= n; ++u)
29
               if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
30
                   update_slack(u, x);
31
32
       void q_push(int x){
33
           if(x \le n)q.push(x);
34
           else for(size_t i = 0;i < flower[x].size(); i++)</pre>
35
               q_push(flower[x][i]);
36
      }
37
       inline void set_st(int x, int b){
38
           st[x]=b;
39
```

```
if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)</pre>
40
                        set_st(flower[x][i], b);
41
42
       inline int get_pr(int b, int xr){
43
           int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
44
           if(pr \% 2 == 1){
45
               reverse(flower[b].begin() + 1, flower[b].end());
               return (int)flower[b].size() - pr;
47
           } else return pr;
48
49
      inline void set_match(int u, int v){
50
51
           match[u]=g[u][v].v;
           if(u > n){
52
               edge e=g[u][v];
53
               int xr = flower_from[u][e.u], pr=get_pr(u, xr);
               for(int i = 0; i < pr; ++i)
55
                    set_match(flower[u][i], flower[u][i ^ 1]);
56
               set_match(xr, v);
57
               rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
58
           }
59
60
       inline void augment(int u, int v){
61
           for(; ; ){
62
               int xnv=st[match[u]];
63
               set_match(u, v);
64
               if(!xnv)return;
               set_match(xnv, st[pa[xnv]]);
66
               u=st[pa[xnv]], v=xnv;
67
           }
68
69
       inline int get_lca(int u, int v){
70
           static int t=0;
71
           for(++t; u || v; swap(u, v)){
72
               if(u == 0)continue;
73
               if(vis[u] == t)return u;
74
               vis[u] = t;
75
               u = st[match[u]];
76
               if(u) u = st[pa[u]];
77
           }
78
           return 0;
80
       inline void add_blossom(int u, int lca, int v){
81
           int b = n + 1;
82
           while(b <= n_x && st[b]) ++b;</pre>
83
           if(b > n_x) ++n_x;
84
           lab[b] = 0, S[b] = 0;
85
```

3.4. 一般最大权匹配 19

```
match[b] = match[lca];
86
            flower[b].clear();
87
            flower[b].push_back(lca);
88
            for(int x = u, y; x != lca; x = st[pa[y]]) {
89
                flower[b].push_back(x),
90
                flower[b].push_back(y = st[match[x]]),
91
                q_push(y);
92
93
           reverse(flower[b].begin() + 1, flower[b].end());
94
            for(int x = v, y; x != lca; x = st[pa[y]]) {
95
                flower[b].push_back(x),
96
97
                flower[b].push_back(y = st[match[x]]),
                q_push(y);
98
            }
99
            set_st(b, b);
100
            for(int x = 1; x \le n_x; ++x) g[b][x].w = g[x][b].w = 0;
101
           for(int x = 1; x \le n; ++x) flower_from[b][x] = 0;
102
            for(size_t i = 0 ; i < flower[b].size(); ++i){</pre>
103
                int xs = flower[b][i];
104
                for(int x = 1; x \le n_x; ++x)
105
                    if(g[b][x].w == 0 \mid \mid e_delta(g[xs][x]) < e_delta(g[b][x]))
106
                         g[b][x] = g[xs][x], g[x][b] = g[x][xs];
                for(int x = 1; x \le n; ++x)
108
                    if(flower_from[xs][x]) flower_from[b][x] = xs;
109
            }
110
            set_slack(b);
111
112
       inline void expand_blossom(int b){ // S[b] == 1
113
            for(size_t i = 0; i < flower[b].size(); ++i)</pre>
114
                set_st(flower[b][i], flower[b][i]);
115
            int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
116
            for(int i = 0; i < pr; i += 2){
                int xs = flower[b][i], xns = flower[b][i + 1];
118
                pa[xs] = g[xns][xs].u;
119
                S[xs] = 1, S[xns] = 0;
                slack[xs] = 0, set_slack(xns);
121
                q_push(xns);
122
123
            S[xr] = 1, pa[xr] = pa[b];
124
            for(size_t i = pr + 1;i < flower[b].size(); ++i){</pre>
                int xs = flower[b][i];
126
                S[xs] = -1, set_slack(xs);
127
            }
128
            st[b] = 0;
129
130
       inline bool on_found_edge(const edge &e){
131
```

```
int u = st[e.u], v = st[e.v];
132
            if(S[v] == -1){
133
                pa[v] = e.u, S[v] = 1;
134
                int nu = st[match[v]];
135
                slack[v] = slack[nu] = 0;
136
                S[nu] = 0, q_push(nu);
137
            else if(S[v] == 0){
138
                int lca = get_lca(u, v);
139
                if(!lca) return augment(u, v), augment(v, u), true;
140
                else add_blossom(u, lca, v);
141
           }
142
           return false;
143
       }
144
       inline bool matching(){
145
           memset(S + 1, -1, sizeof(int) * n_x);
146
           memset(slack + 1, 0, sizeof(int) * n_x);
147
           q = queue<int>();
148
            for(int x = 1; x \le n_x; ++x)
149
                if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
150
            if(q.empty())return false;
151
            for(;;){
152
                while(q.size()){
153
                    int u = q.front();q.pop();
154
                    if(S[st[u]] == 1)continue;
155
                    for(int v = 1; v \le n; ++v)
156
                         if(g[u][v].w > 0 && st[u] != st[v]){
                             if(e_delta(g[u][v]) == 0){
                                  if(on_found_edge(g[u][v]))return true;
                             }else update_slack(u, st[v]);
160
                         }
161
                }
162
                int d = INF;
                for(int b = n + 1; b \le n_x; ++b)
164
                    if(st[b] == b \&\& S[b] == 1)d = min(d, lab[b]/2);
                for(int x = 1; x \le n_x; ++x)
166
                    if(st[x] == x && slack[x]){
167
                         if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                         else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
169
170
                for(int u = 1; u \le n; ++u){
                    if(S[st[u]] == 0){
                         if(lab[u] <= d)return 0;</pre>
173
                         lab[u] -= d;
174
                    }else if(S[st[u]] == 1)lab[u] += d;
175
176
                for(int b = n+1; b <= n_x; ++b)
177
```

3.5. 无向图最小割 21

```
if(st[b] == b){
178
                         if(S[st[b]] == 0) lab[b] += d * 2;
179
                         else if(S[st[b]] == 1) lab[b] -= d * 2;
180
                     }
181
                q=queue<int>();
182
                for(int x = 1; x \le n x; ++x)
183
                     if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
184
     → 0)
                         if(on_found_edge(g[slack[x]][x]))return true;
185
                for(int b = n + 1; b \le n_x; ++b)
186
                     if(st[b] == b \&\& S[b] == 1 \&\& lab[b] == 0)expand_blossom(b);
187
            }
188
            return false;
189
190
       inline pair<long long, int> solve(){
191
            memset(match + 1, 0, sizeof(int) * n);
193
            n_x = n;
            int n_matches = 0;
194
            long long tot_weight = 0;
195
            for(int u = 0; u \le n; ++u) st[u] = u, flower[u].clear();
196
            int w_max = 0;
197
            for(int u = 1; u \le n; ++u)
198
                for(int v = 1; v \le n; ++v){
199
                     flower_from[u][v] = (u == v ? u : 0);
200
                     w_{max} = max(w_{max}, g[u][v].w);
201
            for(int u = 1; u <= n; ++u) lab[u] = w_max;</pre>
203
            while(matching()) ++n_matches;
204
            for(int u = 1; u \le n; ++u)
205
                if(match[u] && match[u] < u)</pre>
206
                     tot_weight += g[u][match[u]].w;
207
            return make_pair(tot_weight, n_matches);
209
       inline void init(){
            for(int u = 1; u \le n; ++u)
211
                for(int v = 1; v \le n; ++v)
212
                     g[u][v]=edge(u, v, 0);
213
       }
214
215 };
```

无向图最小割

```
/*

* Stoer Wagner 全局最小割 O(V ^ 3)

* 1base, 点数 n, 邻接矩阵 edge[MAXN] [MAXN]
```

```
* 返回值为全局最小割
5
6
7
  int StoerWagner() {
      static int v[MAXN], wage[MAXN];
8
      static bool vis[MAXN];
9
      for (int i = 1; i \le n; ++i) v[i] = i;
11
12
      int res = INF;
13
14
15
       for (int nn = n; nn > 1; --nn) {
           memset(vis, 0, sizeof(bool) * (nn + 1));
16
           memset(wage, 0, sizeof(int) * (nn + 1));
17
18
           int pre, last = 1; // vis[1] = 1;
19
20
           for (int i = 1; i < nn; ++i) {
21
               pre = last; last = 0;
22
               for (int j = 2; j \le nn; ++j) if (!vis[j]) {
23
                   wage[j] += edge[v[pre]][v[j]];
24
                   if (!last || wage[j] > wage[last]) last = j;
25
26
               vis[last] = 1;
27
           }
28
29
          res = std::min(res, wage[last]);
30
31
           for (int i = 1; i <= nn; ++i) {
32
               edge[v[i]][v[pre]] += edge[v[last]][v[i]];
33
               edge[v[pre]][v[i]] += edge[v[last]][v[i]];
34
35
           v[last] = v[nn];
36
37
      return res;
38
  }
39
```

最大带权带花树

```
//maximum weight blossom, change g[u][v].w to INF - g[u][v].w when minimum weight blossom

→ is needed

//type of ans is long long

//replace all int to long long if weight of edge is long long

struct WeightGraph {
```

3.6. 最大带权带花树 23

```
static const int INF = INT_MAX;
6
       static const int MAXN = 400;
7
       struct edge{
8
9
           int u, v, w;
           edge() {}
10
           edge(int u, int v, int w): u(u), v(v), w(w) {}
11
      };
12
       int n, n_x;
13
       edge g[MAXN * 2 + 1][MAXN * 2 + 1];
14
       int lab [MAXN * 2 + 1];
15
       int match [MAXN * 2 + 1], slack [MAXN * 2 + 1], st [MAXN * 2 + 1], pa [MAXN * 2 + 1];
16
       int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
17
       vector<int> flower[MAXN * 2 + 1];
18
       queue<int> q;
19
       inline int e_delta(const edge &e){ // does not work inside blossoms
20
           return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
21
22
      inline void update_slack(int u, int x){
23
           if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))</pre>
24
25
               slack[x] = u;
26
       inline void set_slack(int x){
27
           slack[x] = 0;
28
           for(int u = 1; u \le n; ++u)
29
               if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
30
                    update_slack(u, x);
31
      }
32
       void q_push(int x){
33
           if(x \le n)q.push(x);
34
           else for(size_t i = 0;i < flower[x].size(); i++)</pre>
35
               q_push(flower[x][i]);
36
37
       inline void set_st(int x, int b){
38
           st[x]=b;
39
           if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)</pre>
40
                        set_st(flower[x][i], b);
41
42
       inline int get_pr(int b, int xr){
43
           int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
44
           if(pr \% 2 == 1){
               reverse(flower[b].begin() + 1, flower[b].end());
46
               return (int)flower[b].size() - pr;
47
           } else return pr;
48
      }
49
       inline void set_match(int u, int v){
50
           match[u]=g[u][v].v;
51
```

```
if(u > n){
52
               edge e=g[u][v];
53
               int xr = flower_from[u][e.u], pr=get_pr(u, xr);
54
               for(int i = 0; i < pr; ++i)
55
                    set_match(flower[u][i], flower[u][i ^ 1]);
56
               set match(xr, v);
57
               rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
58
           }
59
      }
60
       inline void augment(int u, int v){
61
           for(; ; ){
62
               int xnv=st[match[u]];
63
               set_match(u, v);
64
               if(!xnv)return;
65
               set_match(xnv, st[pa[xnv]]);
66
               u=st[pa[xnv]], v=xnv;
67
           }
68
      }
69
      inline int get_lca(int u, int v){
70
71
           static int t=0;
           for(++t; u || v; swap(u, v)){
72
               if(u == 0)continue;
73
               if(vis[u] == t)return u;
74
               vis[u] = t;
75
               u = st[match[u]];
76
               if(u) u = st[pa[u]];
77
           }
78
79
           return 0;
80
       inline void add_blossom(int u, int lca, int v){
81
           int b = n + 1;
82
           while(b \leq n_x && st[b]) ++b;
83
           if(b > n_x) ++n_x;
84
85
           lab[b] = 0, S[b] = 0;
           match[b] = match[lca];
86
           flower[b].clear();
87
           flower[b].push_back(lca);
88
           for(int x = u, y; x != lca; x = st[pa[y]]) {
89
               flower[b].push_back(x),
90
               flower[b].push_back(y = st[match[x]]),
91
               q_push(y);
92
           }
93
           reverse(flower[b].begin() + 1, flower[b].end());
94
           for(int x = v, y; x != lca; x = st[pa[y]]) {
95
               flower[b].push_back(x),
96
               flower[b].push_back(y = st[match[x]]),
97
```

3.6. 最大带权带花树 25

```
q_push(y);
98
            }
99
            set_st(b, b);
100
            for(int x = 1; x \le n_x; ++x) g[b][x].w = g[x][b].w = 0;
101
            for(int x = 1; x \le n; ++x) flower_from[b][x] = 0;
102
            for(size_t i = 0 ; i < flower[b].size(); ++i){</pre>
103
                int xs = flower[b][i];
104
                for(int x = 1; x \le n_x; ++x)
105
                     if(g[b][x].w == 0 \mid \mid e_delta(g[xs][x]) < e_delta(g[b][x]))
106
                         g[b][x] = g[xs][x], g[x][b] = g[x][xs];
107
                for(int x = 1; x \le n; ++x)
108
                     if(flower_from[xs][x]) flower_from[b][x] = xs;
109
            }
            set_slack(b);
111
       inline void expand_blossom(int b){ // S[b] == 1
113
            for(size_t i = 0; i < flower[b].size(); ++i)</pre>
114
                set_st(flower[b][i], flower[b][i]);
115
            int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
116
            for(int i = 0; i < pr; i += 2){
117
                int xs = flower[b][i], xns = flower[b][i + 1];
118
                pa[xs] = g[xns][xs].u;
119
                S[xs] = 1, S[xns] = 0;
120
                slack[xs] = 0, set_slack(xns);
121
                q_push(xns);
122
            }
123
            S[xr] = 1, pa[xr] = pa[b];
124
            for(size_t i = pr + 1;i < flower[b].size(); ++i){</pre>
                int xs = flower[b][i];
126
                S[xs] = -1, set_slack(xs);
127
            }
128
            st[b] = 0;
130
       inline bool on_found_edge(const edge &e){
            int u = st[e.u], v = st[e.v];
            if(S[v] == -1){
133
                pa[v] = e.u, S[v] = 1;
134
                int nu = st[match[v]];
                slack[v] = slack[nu] = 0;
136
                S[nu] = 0, q_push(nu);
            else if(S[v] == 0){
138
                int lca = get_lca(u, v);
139
                if(!lca) return augment(u, v), augment(v, u), true;
140
                else add_blossom(u, lca, v);
141
142
            return false;
143
```

```
144
       inline bool matching(){
145
           memset(S + 1, -1, sizeof(int) * n_x);
146
            memset(slack + 1, 0, sizeof(int) * n_x);
147
            q = queue<int>();
148
            for(int x = 1; x \le n x; ++x)
149
                if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
150
            if(q.empty())return false;
151
            for(;;){
                while(q.size()){
153
                    int u = q.front();q.pop();
154
                    if(S[st[u]] == 1)continue;
155
                    for(int v = 1; v \le n; ++v)
156
                         if(g[u][v].w > 0 \&\& st[u] != st[v]){
157
                             if(e_delta(g[u][v]) == 0){
158
                                  if(on_found_edge(g[u][v]))return true;
159
                             }else update_slack(u, st[v]);
160
                         }
161
                }
162
163
                int d = INF;
                for(int b = n + 1; b \le n_x; ++b)
164
                    if(st[b] == b \&\& S[b] == 1)d = min(d, lab[b]/2);
165
                for(int x = 1; x \le n_x; ++x)
166
                    if(st[x] == x && slack[x]){
167
                         if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
168
                         else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
170
                for(int u = 1; u \le n; ++u){
171
                    if(S[st[u]] == 0){
                         if(lab[u] <= d)return 0;</pre>
173
                         lab[u] -= d;
174
                    }else if(S[st[u]] == 1)lab[u] += d;
176
                for(int b = n+1; b \le n_x; ++b)
                    if(st[b] == b){
                         if(S[st[b]] == 0) lab[b] += d * 2;
                         else if(S[st[b]] == 1) lab[b] -= d * 2;
180
                    }
181
                q=queue<int>();
182
                for(int x = 1; x \le n_x; ++x)
183
                    if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
184
     if(on_found_edge(g[slack[x]][x]))return true;
185
                for(int b = n + 1; b \le n_x; ++b)
186
                    if(st[b] == b \&\& S[b] == 1 \&\& lab[b] == 0)expand_blossom(b);
187
            }
188
```

```
return false;
189
       }
190
        inline pair<long long, int> solve(){
191
            memset(match + 1, 0, sizeof(int) * n);
192
            n_x = n;
193
            int n matches = 0;
194
            long long tot_weight = 0;
195
            for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();</pre>
196
            int w_max = 0;
197
            for(int u = 1; u \le n; ++u)
198
                 for(int v = 1; v \le n; ++v){
199
                     flower_from[u][v] = (u == v ? u : 0);
200
                     w_{max} = max(w_{max}, g[u][v].w);
201
                 }
202
            for(int u = 1; u <= n; ++u) lab[u] = w_max;</pre>
203
            while(matching()) ++n_matches;
204
            for(int u = 1; u \le n; ++u)
205
                 if(match[u] && match[u] < u)</pre>
206
                     tot_weight += g[u][match[u]].w;
207
            return make_pair(tot_weight, n_matches);
208
       }
209
        inline void init(){
210
            for(int u = 1; u \le n; ++u)
211
                 for(int v = 1; v \le n; ++v)
212
                     g[u][v]=edge(u, v, 0);
213
       }
214
215 };
```

必经点 dominator tree

```
//solve(s, n, raw_g): s is the root and base accords to base of raw_g
  //idom[x] will be x if x does not have a dominator, and will be -1 if x is not reachable from
    ⇒S.
3
4
  struct dominator_tree {
      int base, dfn[N], sdom[N], idom[N], id[N], f[N], fa[N], smin[N], stamp;
5
      Graph *g;
6
      void predfs(int u) {
7
           id[dfn[u] = stamp++] = u;
8
           for (int i = g -> adj[u]; ~i; i = g -> nxt[i]) {
9
               int v = g \rightarrow v[i];
10
               if (dfn[v] < 0) {</pre>
11
                   f[v] = u;
12
                   predfs(v);
13
               }
14
```

```
}
15
16
       int getfa(int u) {
17
           if (fa[u] == u) return u;
18
           int ret = getfa(fa[u]);
19
           if (dfn[sdom[smin[fa[u]]]] < dfn[sdom[smin[u]]])</pre>
20
               smin[u] = smin[fa[u]];
21
           return fa[u] = ret;
22
       }
23
       void solve (int s, int n, Graph *raw_graph) {
24
           g = raw_graph;
25
26
           base = g \rightarrow base;
           memset(dfn + base, -1, sizeof(*dfn) * n);
27
           memset(idom + base, -1, sizeof(*idom) * n);
28
           static Graph pred, tmp;
29
           pred.init(base, n);
30
           for (int i = 0; i < n; ++i) {
31
               for (int p = g -> adj[i + base]; ~p; p = g -> nxt[p])
32
                    pred.ins(g -> v[p], i + base);
33
           }
34
           stamp = 0; tmp.init(base, n); predfs(s);
35
           for (int i = 0; i < stamp; ++i) {</pre>
36
               fa[id[i]] = smin[id[i]] = id[i];
37
           }
38
           for (int o = stamp - 1; o >= 0; --o) {
39
               int x = id[o];
40
               if (o) {
41
                    sdom[x] = f[x];
42
                    for (int i = pred.adj[x]; ~i; i = pred.nxt[i]) {
43
                        int p = pred.v[i];
                        if (dfn[p] < 0) continue;</pre>
45
                        if (dfn[p] > dfn[x]) {
                             getfa(p);
47
48
                             p = sdom[smin[p]];
49
                        if (dfn[sdom[x]] > dfn[p]) sdom[x] = p;
50
51
                    tmp.ins(sdom[x], x);
52
53
               while (~tmp.adj[x]) {
                    int y = tmp.v[tmp.adj[x]];
55
                    tmp.adj[x] = tmp.nxt[tmp.adj[x]];
56
57
                    getfa(y);
                    if (x != sdom[smin[y]]) idom[y] = smin[y];
58
                    else idom[y] = x;
59
               }
60
```

```
for (int i = g \rightarrow adj[x]; ~i; i = g \rightarrow nxt[i])
61
                       if (f[g \rightarrow v[i]] == x) fa[g \rightarrow v[i]] = x;
62
            }
63
64
            idom[s] = s;
            for (int i = 1; i < stamp; ++i) {</pre>
65
                 int x = id[i];
66
                 if (idom[x] != sdom[x]) idom[x] = idom[idom[x]];
67
            }
68
        }
69
70 };
```

Chapter 4

字符串

Manacher

```
//prime is the origin string(0-base)
  //-10,-1,-20 are added to s
3 //length of s is exactly 2 * 1 + 3
4 inline void manacher(char prime[]) {
      int 1 = strlen(prime), n = 0;
      s[n++] = -10;
6
      s[n++] = -1;
7
      for (int i = 0; i < 1; ++i) {
8
          s[n++] = prime[i];
9
          s[n++] = -1;
10
11
      s[n++] = -20; f[0] = 1;
12
      int mx = 0, id = 0;
13
      for (int i = 1; i + 1 < n; ++i) {
14
          f[i] = i > mx ? 1 : min(f[id * 2 - i], mx - i + 1);
15
          while (s[i + f[i]] == s[i - f[i]]) ++f[i];
16
          if (i + f[i] - 1 > mx) {
17
               mx = i + f[i] - 1;
18
               id = i;
19
          }
      }
21
  }
```

数组版后缀自动机

```
/*
* Suffix Automaton - array version

* SAMSAMSAM? SAMSAMSAM!

*/
5
```

32 CHAPTER 4. 字符串

```
6 namespace SAM {
       int to[100005 << 1][26], parent[100005 << 1], step[100005 << 1], tot;
7
       int root, np;
8
9
       int sam_len;
10
       int newnode(int STEP = 0) {
11
           ++tot:
12
           memset(to[tot], 0, sizeof to[tot]);
13
           parent[tot] = 0;
14
           step[tot] = STEP;
15
           return tot;
16
      }
17
18
      void init() {
19
           tot = 0;
20
           root = np = newnode(sam_len = 0);
21
22
23
      void extend(char ch) {
24
           int x = ch - 'a';
25
           int last = np; np = newnode(++sam_len);
26
           for (; last && !to[last][x]; last = parent[last])
27
               to[last][x] = np;
28
           if (!last) parent[np] = root;
29
           else {
30
31
               int q = to[last][x];
               if (step[q] == step[last] + 1) parent[np] = q;
32
               else {
33
                    nq = newnode(step[last] + 1);
34
                    memcpy(to[nq], to[q], sizeof to[q]);
35
                    parent[nq] = parent[q];
36
                   parent[q] = parent[np] = nq;
                    for (; last && to[last][x] == q; last = parent[last])
38
                        to[last][x] = nq;
39
               }
40
           }
41
42
      }
43 }
```

指针版后缀自动机

```
/*

* Suffix Automaton - pointer version

* SAMSAMSAM? SAMSAMSAM!

*/
```

4.3. 指针版后缀自动机 33

```
5
  namespace SAM {
6
       struct Node *pool_pointer;
7
8
       struct Node {
           Node *to[26], *parent;
9
           int step;
10
11
           Node(int STEP = 0): step(STEP) {
12
               memset(to, 0, sizeof to);
13
               parent = 0;
14
               step = 0;
15
           }
16
17
           void *operator new (size_t) {
18
               return pool_pointer++;
19
           }
20
       } pool[100005 << 1], *root, *np;</pre>
21
       int sam_len;
22
23
       void init() {
24
           pool_pointer = pool;
25
           root = np = new Node(sam_len = 0);
26
       }
27
28
       void extend(char ch) {
29
30
           static Node *last, *q, *nq;
31
           int x = ch - 'a';
32
           last = np; np = new Node(++sam_len);
33
           for (; last && !last->to[x]; last = last->parent)
               last->to[x] = np;
35
           if (!last) np->parent = root;
           else {
37
38
               q = last->to[x];
               if (q->step == last->step + 1) np->parent = q;
39
40
               else {
                    nq = new Node(*q);
41
                    nq->step = last->step + 1;
42
                    q->parent = np->parent = nq;
43
                    for (; last && last->to[x] == q; last = last->parent)
                        last->to[x] = nq;
45
               }
46
           }
47
       }
48
49 }
```

34 CHAPTER 4. 字符串

广义后缀自动机

```
1 /*
  * EX Suffix Automaton - pointer version
   * SAMSAMSAM? SAMSAMSAM!
   */
5
  namespace SAM {
6
       struct Node *pool_pointer;
7
      struct Node {
8
           Node *parent, *to[26];
9
           int step;
10
11
           Node(int step = 0): step(step) {
12
               memset(to, 0, sizeof to);
13
               parent = 0;
14
           }
15
16
           void *operator new (size_t) {
17
               return pool_pointer++;
18
           }
19
       } pool[100005 * 10 << 1], *root, *np;</pre>
20
       int sam_len, now_len;
21
22
       void init() {
23
           sam_len = now_len = 0;
24
           pool_pointer = pool;
25
           root = new Node();
26
       }
27
28
       void new_str() { // a new string start
29
           now_len = 0;
30
31
           np = root;
       }
32
33
       void extend(char ch) {
34
           static Node *last, *q, *nq;
35
36
           int x = ch - 'a';
37
           if (np->to[x]) {
38
               np = np->to[x];
39
               ++now_len;
40
           }
41
           else {
42
               last = np; np = new Node(++now_len);
43
               for (; last && !last->to[x]; last = last->parent)
44
```

4.4. 广义后缀自动机 35

```
last->to[x] = np;
45
               if (!last) np->parent = root;
46
               else {
47
                   q = last->to[x];
48
                   if (q->step == last->step + 1) np->parent = q;
49
50
                        nq = new Node(*q);
51
                        nq->step = last->step + 1;
52
                        q->parent = np->parent = nq;
53
                        for (; last && last->to[x] == q; last = last->parent)
54
                            last->to[x] = nq;
55
                   }
56
               }
57
           }
58
59
           sam_len = std::max(sam_len, now_len);
60
      }
61
62 }
```

Chapter 5

技巧

无敌的读入优化

```
1 // getchar() 读入优化 << 关同步 cin << 此优化
  |// 用 isdigit() 会小幅变慢
₃|// 返回 false 表示读到文件尾
  namespace Reader {
      const int L = (1 << 15) + 5;
5
      char buffer[L], *S, *T;
6
      __inline bool getchar(char &ch) {
7
          if (S == T) {
8
              T = (S = buffer) + fread(buffer, 1, L, stdin);
9
              if (S == T) {
10
                  ch = EOF;
11
                  return false;
12
              }
13
          }
          ch = *S++;
15
          return true;
16
17
      __inline bool getint(int &x) {
18
          char ch; bool neg = 0;
19
          for (; getchar(ch) && (ch < '0' || ch > '9'); ) neg ^= ch == '-';
20
          if (ch == EOF) return false;
21
          x = ch - '0';
22
          for (; getchar(ch), ch >= '0' && ch <= '9'; )
23
              x = x * 10 + ch - '0';
24
          if (neg) x = -x;
25
          return true;
26
      }
27
28 }
```

38 CHAPTER 5. 技巧

真正释放 STL 内存

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

梅森旋转算法

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

32-bit/64-bit 随机素数

32-bit	64-bit
73550053	1249292846855685773
148898719	1701750434419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3