

ICPC Templates For Africamonkey

Africamonkey

March 10, 2017

Contents

| | | |
|----------|-----------------------|-----------|
| 1 | 莫队算法 | 3 |
| 1.1 | 普通莫队 | 3 |
| 1.2 | 树上莫队 | 3 |
| 2 | 字符串 | 5 |
| 2.1 | 哈希 | 5 |
| 2.2 | KMP | 5 |
| 2.3 | 可动态修改的KMP | 6 |
| 2.4 | 扩展KMP | 6 |
| 2.5 | Manacher | 8 |
| 2.6 | 最小表示法 | 8 |
| 2.7 | AC自动机 | 9 |
| 2.8 | 后缀数组 | 10 |
| 2.9 | 后缀自动机 | 11 |
| 3 | 数据结构 | 12 |
| 3.1 | ST表 | 12 |
| 3.2 | 线段树小技巧 | 12 |
| 3.3 | Splay | 13 |
| 3.4 | 可持久化Treap | 16 |
| 4 | 树 | 18 |
| 4.1 | 动态树 | 18 |
| 5 | 图 | 20 |
| 5.1 | 欧拉回路 | 20 |
| 5.2 | 最短路径 | 21 |
| 5.2.1 | Dijkstra | 21 |
| 5.2.2 | SPFA | 21 |
| 5.3 | K 短路 | 22 |
| 5.4 | Tarjan | 25 |
| 5.5 | 2-SAT | 25 |
| 5.6 | 统治者树 (Dominator Tree) | 27 |
| 5.7 | 网络流 | 28 |

| | | |
|----------|-------------------------|-----------|
| 5.7.1 | 最大流 | 28 |
| 5.7.2 | 上下界有源汇网络流 | 29 |
| 5.7.3 | 上下界无源汇网络流 | 30 |
| 5.7.4 | 费用流 | 30 |
| 5.7.5 | zkw费用流 | 31 |
| 6 | 数学 | 33 |
| 6.1 | 扩展欧几里得解同余方程 | 33 |
| 6.2 | 同余方程组 | 33 |
| 6.3 | 卡特兰数 | 33 |
| 6.4 | 斯特林数 | 33 |
| 6.4.1 | 第一类斯特林数 | 33 |
| 6.4.2 | 第二类斯特林数 | 34 |
| 6.5 | 错排公式 | 34 |
| 6.6 | Lucas定理 | 34 |
| 6.7 | 高斯消元 | 34 |
| 6.7.1 | 行列式 | 34 |
| 6.7.2 | Matrix-Tree定理 | 35 |
| 6.8 | 调和级数 | 35 |
| 6.9 | 曼哈顿距离的变换 | 35 |
| 6.10 | 线性筛素数 | 35 |
| 6.11 | FFT | 36 |
| 6.12 | 求原根 | 37 |
| 6.13 | NTT | 38 |
| 6.14 | 组合数 lcm | 39 |
| 6.15 | 区间 lcm 的维护 | 39 |
| 7 | 几何 | 40 |
| 7.1 | 凸包 | 40 |
| 8 | 黑科技和杂项 | 40 |
| 8.1 | 高精度计算 | 40 |

1 莫队算法

1.1 普通莫队

```
1 struct Q { int l, r, sqrtl, id; } q[N];
2 int n, m, v[N], ans[N], nowans;
3 bool cmp(const Q &a, const Q &b) {
4     if (a.sqrtl != b.sqrtl) return a.sqrtl < b.sqrtl;
5     return a.r < b.r;
6 }
7 void change(int x) { if (!v[x]) checkin(); else checkout(); }
8 int main() {
9     .....
10    for (int i=1;i<=m;i++) q[i].sqrtl = q[i].l / sqrt(n), q[i].id = i;
11    sort(q+1, q+m+1, cmp);
12    int L=1,R=0; nowans=0;
13    memset(v, 0, sizeof(v));
14    for (int i=1;i<=m;i++) {
15        while (L<q[i].l) change(L++);
16        while (L>q[i].l) change(--L);
17        while (R<q[i].r) change(++R);
18        while (R>q[i].r) change(R--);
19        ans[q[i].id] = nowans;
20    }
21    .....
22 }
```

1.2 树上莫队

```
1 struct Query { int l, r, id, l_group; } query[N];
2 struct EDGE { int adj, next; } edge[N*2];
3 int n, m, top, gh[N], c[N], reorder[N], deep[N], father[N], size[N], son[N], Top[N];
4 void addedge(int x, int y) {
5     edge[++top].adj = y;
6     edge[top].next = gh[x];
7     gh[x] = top;
8 }
9 void dfs(int x, int root=0) {
10    reorder[x] = ++top; father[x] = root; deep[x] = deep[root] + 1;
11    son[x] = 0; size[x] = 1; int dd = 0;
12    for (int p=gh[x]; p; p=edge[p].next)
13        if (edge[p].adj != root) {
14            dfs(edge[p].adj, x);
15            if (size[edge[p].adj] > dd) {
16                son[x] = edge[p].adj;
17                dd = size[edge[p].adj];
18            }
19            size[x] += size[edge[p].adj];
20        }
21 }
22 void split(int x, int tp) {
23     Top[x] = tp;
24     if (son[x]) split(son[x], tp);
25     for (int p=gh[x]; p; p=edge[p].next)
26         if (edge[p].adj != father[x] && edge[p].adj != son[x])
```

```

27         split(edge[p].adj, edge[p].adj);
28     }
29     int lca(int x, int y) {
30         int tx = Top[x], ty = Top[y];
31         while (tx != ty) {
32             if (deep[tx] < deep[ty]) {
33                 swap(tx, ty);
34                 swap(x, y);
35             }
36             x = father[tx];
37             tx = Top[x];
38         }
39         if (deep[x] < deep[y]) swap(x, y);
40         return y;
41     }
42     bool cmp(const Query &a, const Query &b) {
43         if (a.l_group != b.l_group) return a.l_group < b.l_group;
44         return reorder[a.r] < reorder[b.r];
45     }
46     int v[N], ans[N];
47     void upd(int x) { if (!v[x]) checkin(); else checkout(); }
48     void go(int &u, int taru, int v) {
49         int lca0 = lca(u, taru);
50         int lca1 = lca(u, v);   upd(lca1);
51         int lca2 = lca(taru, v); upd(lca2);
52         for (int x=u; x!=lca0; x=father[x]) upd(x);
53         for (int x=taru; x!=lca0; x=father[x]) upd(x);
54         u = taru;
55     }
56     int main() {
57         memset(gh, 0, sizeof(gh));
58         scanf("%d%d", &n, &m); top = 0;
59         for (int i=1;i<n;i++) {
60             int x,y; scanf("%d%d", &x, &y);
61             addedge(x, y); addedge(y, x);
62         }
63         top = 0; dfs(1); split(1, 1);
64         for (int i=1;i<=m;i++) {
65             if (reorder[query[i].l] > reorder[query[i].r])
66                 swap(query[i].l, query[i].r);
67             query[i].id = i;
68             query[i].l_group = reorder[query[i].l] / sqrt(n);
69         }
70         sort(query+1, query+m+1, cmp);
71         int L=1,R=1; upd(1);
72         for (int i=1;i<=m;i++) {
73             go(L,query[i].l,R);
74             go(R,query[i].r,L);
75             ans[query[i].id] = answer();
76         }
77         .....
78     }

```

2 字符串

2.1 哈希

```
1 const int P=31,D=1000173169;
2 int n, pow[N], f[N]; char a[N];
3 int hash(int l, int r) { return (LL) (f[r]-(LL)f[l-1]*pow[r-l+1]%D+D)%D; }
4 int main() {
5     scanf("%d%s", &n, a+1);
6     pow[0] = 1;
7     for (int i=1;i<=n;i++) pow[i] = (LL)pow[i-1]*P%D;
8     for (int i=1;i<=n;i++) f[i] = (LL) ((LL)f[i-1]*P+a[i])%D;
9 }
```

2.2 KMP

接口: `int find_substring(char *pattern, char *text, int *next, int *ret);`

输入: 模式串, 匹配串

输出: 返回值表示模式串在匹配串中出现的次数

KMP的`next[i]`表示从0到i的字符串s, 前缀和后缀的最长重叠长度。

```
1 void find_next(char *pattern, int *next) {
2     int n = strlen(pattern);
3     for (int i=1;i<n;i++) {
4         int j = i;
5         while (j > 0) {
6             j = next[j];
7             if (pattern[j] == pattern[i]) {
8                 next[i+1] = j+1;
9                 break;
10            }
11        }
12    }
13 }
14 int find_substring(char *pattern, char *text, int *next, int *ret) {
15     find_next(pattern, next);
16     int n = strlen(pattern);
17     int m = strlen(text);
18     int k = 0;
19     for (int i=0,j=0;i<m;i++) {
20         if (j<n && text[i]==pattern[j]) {
21             j++;
22         } else {
23             while (j>0) {
24                 j = next[j];
25                 if (text[i] == pattern[j]) {
26                     j++;
27                     break;
28                 }
29             }
30         }
31         if (j == n)
32             ret[k++] = i-n+1;
33     }
34     return k;
35 }
```

35 }

2.3 可动态修改的KMP

支持：加入一个字符，删除一个字符。

时间复杂度： $O(n\alpha)$ ， α 为字符集大小。

代码中的字符为 '0' - '9'，可自行修改为 'a' - 'z'

```
1 char t[N];
2 int top, nxt[N], nxt_l[N][10];
3 inline void del_letter() { --top; }
4 inline void add_letter(char x) {
5     t[top++] = x;
6     int j = top-1;
7     memset(nxt_l[top], 0, sizeof(nxt_l[top]));
8     nxt[top] = nxt_l[top-1][x-'0'];
9     memcpy(nxt_l[top], nxt_l[nxt[top]], sizeof(nxt_l[nxt[top]]));
10    nxt_l[top][t[nxt[top]]-'0'] = nxt[top]+1;
11 }
```

2.4 扩展KMP

接口： void ExtendedKMP(char *a, char *b, int *next, int *ret);

输出：

next: a 关于自己每个后缀的最长公共前缀

ret: a 关于 b 的每个后缀的最长公共前缀

EXKMP的next[i]表示：从i到n-1的字符串st前缀和原串前缀的最长重叠长度。

```
1 void get_next(char *a, int *next) {
2     int i, j, k;
3     int n = strlen(a);
4     for (j = 0; j+1<n && a[j]==a[j+1];j++);
5     next[1] = j;
6     k = 1;
7     for (i=2;i<n;i++) {
8         int len = k+next[k], l = next[i-k];
9         if (l < len-i) {
10             next[i] = l;
11         } else {
12             for (j = max(0, len-i);i+j<n && a[j]==a[i+j];j++);
13             next[i] = j;
14             k = i;
15         }
16     }
17 }
18 void ExtendedKMP(char *a, char *b, int *next, int *ret) {
19     get_next(a, next);
20     int n = strlen(a), m = strlen(b);
21     int i, j, k;
22     for (j=0;j<n && j<m && a[j]==b[j];j++);
23     ret[0] = j;
24     k = 0;
```

```

25     for (i=1;i<m;i++) {
26         int len = k+ret[k], l = next[i-k];
27         if (l < len-i) {
28             ret[i] = l;
29         } else {
30             for (j = max(0, len-i); j<n && i+j<m && a[j]==b[i+j]; j++);
31             ret[i] = j;
32             k = i;
33         }
34     }
35 }

```

2.5 Manacher

$p[i]$ 表示以 i 为对称轴的最长回文串长度

```
1 char st[N*2], s[N];
2 int len, p[N*2];
3
4 while (scanf("%s", s) != EOF) {
5     len = strlen(s);
6     st[0] = '$', st[1] = '#';
7     for (int i=1;i<=len;i++)
8         st[i*2] = s[i-1], st[i*2+1] = '#';
9     len = len * 2 + 2;
10    int mx = 0, id = 0, ans = 0;
11    for (int i=1;i<=len;i++) {
12        p[i] = (mx > i) ? min(p[id*2-i]+1, mx-i) : 1;
13        for (; st[i+p[i]] == st[i-p[i]]; ++p[i]) ;
14        if (p[i]+i > mx) mx = p[i]+i, id = i;
15        p[i]--;
16        if (p[i] > ans) ans = p[i];
17    }
18    printf("%d\n", ans);
19 }
```

2.6 最小表示法

```
1 string smallestRepration(string s) {
2     int i, j, k, l;
3     int n = s.length();
4     s += s;
5     for (i=0,j=1;j<n;) {
6         for (k=0;k<n && s[i+k]==s[j+k];k++);
7         if (k>=n) break;
8         if (s[i+k]<s[j+k]) j+=k+1;
9         else {
10            l=i+k;
11            i=j;
12            j=max(l, j)+1;
13        }
14    }
15    return s.substr(i, n);
16 }
```


2.7 AC自动机

```
1 struct Node {
2     int next[**Size of Alphabet**];
3     int terminal, fail;
4 } node[**Number of Nodes**];
5 int top;
6 void add(char *st) {
7     int len = strlen(st), x = 1;
8     for (int i=0;i<len;i++) {
9         int ind = trans(st[i]);
10        if (!node[x].next[ind])
11            node[x].next[ind] = ++top;
12        x = node[x].next[ind];
13    }
14    node[x].terminal = 1;
15 }
16 int q[**Number of Nodes**], head, tail;
17 void build() {
18     head = 0, tail = 1; q[1] = 1;
19     while (head != tail) {
20         int x = q[++head];
21         /*(when necessary) node[x].terminal != node[node[x].fail].terminal; */
22         for (int i=0;i<n;i++)
23             if (node[x].next[i]) {
24                 if (x == 1) node[node[x].next[i]].fail = 1;
25                 else {
26                     int y = node[x].fail;
27                     while (y) {
28                         if (node[y].next[i]) {
29                             node[node[x].next[i]].fail = node[y].next[i];
30                             break;
31                         }
32                         y = node[y].fail;
33                     }
34                     if (!node[node[x].next[i]].fail) node[node[x].next[i]].fail = 1;
35                 }
36                 q[++tail] = node[x].next[i];
37             }
38     }
39 }
```

2.8 后缀数组

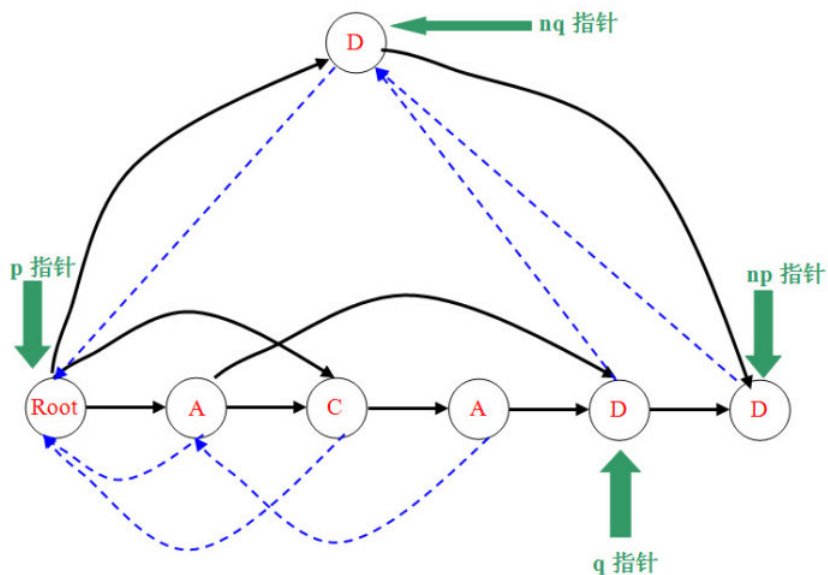
参数 m 表示字符集的大小, 即 $0 \leq r_i < m$

```
1  #define rank rank2
2  int n, r[N], wa[N], wb[N], ws[N], sa[N], rank[N], height[N];
3  int cmp(int *r, int a, int b, int l, int n)
4  {
5      if (r[a]==r[b])
6      {
7          if (a+l<n && b+l<n && r[a+l]==r[b+l])
8              return 1;
9      }
10     return 0;
11 }
12 void suffix_array(int m)
13 {
14     int i, j, p, *x=wa, *y=wb, *t;
15     for (i=0;i<m;i++) ws[i]=0;
16     for (i=0;i<n;i++) ws[x[i]=r[i]]++;
17     for (i=1;i<m;i++) ws[i]+=ws[i-1];
18     for (i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
19     for (j=1,p=1;p<n;m=p,j<=1)
20     {
21         for (p=0,i=n-j;i<n;i++) y[p++]=i;
22         for (i=0;i<n;i++) if (sa[i]>=j) y[p++]=sa[i]-j;
23         for (i=0;i<m;i++) ws[i]=0;
24         for (i=0;i<n;i++) ws[x[y[i]]]++;
25         for (i=1;i<m;i++) ws[i]+=ws[i-1];
26         for (i=n-1;i>=0;i--) sa[--ws[x[y[i]]]]=y[i];
27         for (t=x,x=y,y=t,x[sa[0]]=0,i=1,p=1;i<n;i++)
28             x[sa[i]]=cmp(y,sa[i-1],sa[i],j,n)?p-1:p++;
29     }
30     for (i=0;i<n;i++) rank[sa[i]]=i;
31 }
32 void calc_height()
33 {
34     int j=0;
35     for (int i=0;i<n;i++)
36         if (rank[i])
37         {
38             while (r[i+j]==r[sa[rank[i]-1]+j]) j++;
39             height[rank[i]]=j;
40             if (j) j--;
41         }
42 }
```

2.9 后缀自动机

下面的代码是求两个串的LCS（最长公共子串）。

```
1  #include <stdio>
2  #include <stdlib>
3  #include <string>
4  #define N 500001
5  using namespace std;
6  char st[N];
7  int pre[N<<1], son[26][N<<1], step[N<<1], last, total;
8  int apply(int x) { step[++total]=x; return total; }
9  void Extend(char x) {
10     int p = last, np = apply(step[last]+1);
11     for (; p && !son[x][p]; p=pre[p]) son[x][p] = np;
12     if (!p) pre[np] = 1;
13     else {
14         int q = son[x][p];
15         if (step[p]+1 == step[q]) pre[np] = q;
16         else {
17             int nq = apply(step[p]+1);
18             for (int i=0;i<26;i++) son[i][nq] = son[i][q];
19             pre[nq] = pre[q];
20             pre[q] = pre[np] = nq;
21             for (; p && son[x][p]==q; p=pre[p]) son[x][p] = nq;
22         }
23     }
24     last = np;
25 }
26 void init() {
27     last = total = 0;
28     last = apply(0);
29     scanf("%s",st);
30     for (int i=0; st[i]; i++)
31         Extend(st[i]-'a');
32     scanf("%s",st);
33 }
34 int main() {
35     init();
36     int p = 1, now = 0, ans = 0;
37     for (int i=0; st[i]; i++) {
38         int index = st[i]-'a';
39         for (; p && !son[index][p]; p = pre[p], now = step[p]) ;
40         if (!p) p = 1;
41         if (son[index][p]) {
42             p = son[index][p];
43             now++;
44             if (now > ans) ans = now;
45         }
46     }
47     printf("%d\n",ans);
48     return 0;
49 }
```



3 数据结构

3.1 ST表

```

1 int Log[N], f[17][N];
2 int ask(int x, int y) {
3     int k = log[y - x + 1];
4     return max(f[k][x], f[k][y - (1 << k) + 1]);
5 }
6 int main() {
7     for (i = 2; i <= n; i++) Log[i] = Log[i >> 1] + 1;
8     for (j = 1; j < K; j++) for (i = 1; i + (1 << j - 1) <= n; i++) f[j][i] = max(f[j - 1][i], f[j - 1][i + (1 << j - 1)]);
9 }

```

3.2 线段树小技巧

给定一个序列 a ，寻找一个最大的 i 使得 $i \leq y$ 且满足一些条件（如 $a[i] \geq w$ ，那么需要在线段树维护 a 的区间最大值）

```

1 int queryl(int p, int left, int right, int y, int w) {
2     if (right <= y) {
3         if (! __condition__ ) return -1;
4         else if (left == right) return left;
5     }
6     int mid = (left + right) / 2;
7     if (y <= mid) return queryl(p << 1 | 0, left, mid, y, w);
8     int ret = queryl(p << 1 | 1, mid + 1, right, y, w);
9     if (ret != -1) return ret;
10    return queryl(p << 1 | 0, left, mid, y, w);
11 }

```

给定一个序列 a ，寻找一个最小的 i 使得 $i \geq x$ 且满足一些条件（如 $a[i] \geq w$ ，那么需要在线段树维护 a 的区间最大值）

```

1 int queryr(int p, int left, int right, int y, int w) {
2     if (left >= x) {
3         if (! __condition__ ) return -1;
4         else if (left == right) return left;
5     }
6     int mid = (left + right) / 2;
7     if (x > mid) return queryr(p<<1|1, mid+1, right, y, w);
8     int ret = queryr(p<<1|0, left, mid, y, w);
9     if (ret != -1) return ret;
10    return queryr(p<<1|1, mid+1, right, y, w);
11 }

```

3.3 Splay

接口:

ADD $x\ y\ d$: 将 $[x, y]$ 的所有数加上 d

REVERSE $x\ y$: 将 $[x, y]$ 翻转

INSERT $x\ p$: 将 p 插入到第 x 个数的后面

DEL x : 将第 x 个数删除

```

1 struct SPLAY {
2     struct NODE {
3         int w, min;
4         int son[2], size, father, rev, lazy;
5     } node[N];
6     int top, rt;
7     void pushdown(int x) {
8         if (!x) return;
9         if (node[x].rev) {
10            node[node[x].son[0]].rev ^= 1;
11            node[node[x].son[1]].rev ^= 1;
12            swap(node[x].son[0], node[x].son[1]);
13            node[x].rev = 0;
14        }
15        if (node[x].lazy) {
16            node[node[x].son[0]].lazy += node[x].lazy;
17            node[node[x].son[1]].lazy += node[x].lazy;
18            node[x].w += node[x].lazy;
19            node[x].min += node[x].lazy;
20            node[x].lazy = 0;
21        }
22    }
23    void pushup(int x) {
24        if (!x) return;
25        pushdown(node[x].son[0]);
26        pushdown(node[x].son[1]);
27        node[x].size = node[node[x].son[0]].size + node[node[x].son[1]].size + 1;
28        node[x].min = node[x].w;
29        if (node[x].son[0]) node[x].min = min(node[x].min, node[node[x].son[0]].min);
30        if (node[x].son[1]) node[x].min = min(node[x].min, node[node[x].son[1]].min);
31    }
32    void sc(int x, int y, int w) {
33        node[x].son[w] = y;
34        node[y].father = x;

```

```

35     pushup(x);
36 }
37 void _ins(int w) {
38     top++;
39     node[top].w = node[top].min = w;
40     node[top].son[0] = node[top].son[1] = 0;
41     node[top].size = 1; node[top].father = 0; node[top].rev = 0;
42 }
43 void init() {
44     top = 0;
45     _ins(0); _ins(0); rt=1;
46     sc(1, 2, 1);
47 }
48 void rotate(int x) {
49     if (!x) return;
50     int y = node[x].father;
51     int w = node[y].son[1]==x;
52     sc(y, node[x].son[w^1], w);
53     sc(node[y].father, x, node[node[y].father].son[1]==y);
54     sc(x, y, w^1);
55 }
56 int q[N];
57 void flushdown(int x) {
58     int t=0; for (; x; x=node[x].father) q[++t]=x;
59     for (; t; t--) pushdown(q[t]);
60 }
61 void Splay(int x, int root=0) {
62     flushdown(x);
63     while (node[x].father != root) {
64         int y=node[x].father;
65         int w=node[y].son[1]==x;
66         if (node[y].father != root && node[node[y].father].son[w]==y) rotate(y);
67         rotate(x);
68     }
69 }
70 int find(int k) {
71     Splay(rt);
72     while (1) {
73         pushdown(rt);
74         if (node[node[rt].son[0]].size+1==k) {
75             Splay(rt);
76             return rt;
77         } else
78         if (node[node[rt].son[0]].size+1<k) {
79             k-=node[node[rt].son[0]].size+1;
80             rt=node[rt].son[1];
81         } else {
82             rt=node[rt].son[0];
83         }
84     }
85 }
86 int split(int x, int y) {
87     int fx = find(x);
88     int fy = find(y+2);
89     Splay(fx);
90     Splay(fy, fx);
91     return node[fy].son[0];

```

```

92     }
93     void add(int x, int y, int d) { //add d to each number in a[x]...a[y]
94         int t = split(x, y);
95         node[t].lazy += d;
96         Splay(t); rt=t;
97     }
98     void reverse(int x, int y) { // reverse the x-th to y-th elements
99         int t = split(x, y);
100        node[t].rev ^= 1;
101        Splay(t); rt=t;
102    }
103    void insert(int x, int p) { // insert p after the x-th element
104        int fx = find(x+1);
105        int fy = find(x+2);
106        Splay(fx);
107        Splay(fy, fx);
108        _ins(p);
109        sc(fy, top, 0);
110        Splay(top); rt=top;
111    }
112    void del(int x) { // delete the x-th element in Splay
113        int fx = find(x), fy = find(x+2);
114        Splay(fx); Splay(fy, fx);
115        node[fy].son[0] = 0;
116        Splay(fy); rt=fy;
117    }
118 } tree;

```

3.4 可持久化Treap

接口:

void insert(int x, char c); 在当前第 x 个字符后插入 c

void del(int x, int y); 删除第 x 个字符到第 y 个字符

void copy(int l, int r, int x); 复制第 l 个字符到第 r 个字符, 然后粘贴到第 x 个字符后

void reverse(int x, int y); 翻转第 x 个到第 y 个字符

char query(int k); 表示询问当前第 x 个字符是什么

```
1 #define mod 1000000007
2 struct Treap {
3     struct Node {
4         char key;
5         bool reverse;
6         int lc, rc, size;
7     } node[N];
8     int n, root, rd;
9     int Rand() { rd = (rd * 20372052LL + 25022087LL) % mod; return rd; }
10    void init() { n = root = 0; }
11    inline int copy(int x) { node[++n] = node[x]; return n; }
12    inline void pushdown(int x) {
13        if (!node[x].reverse) return;
14        if (node[x].lc) node[x].lc = copy(node[x].lc);
15        if (node[x].rc) node[x].rc = copy(node[x].rc);
16        swap(node[x].lc, node[x].rc);
17        node[node[x].lc].reverse ^= 1;
18        node[node[x].rc].reverse ^= 1;
19        node[x].reverse = 0;
20    }
21    inline void pushup(int x) { node[x].size = node[node[x].lc].size + node[node[x].rc].size
22        + 1; }
23    int merge(int u, int v) {
24        if (!u || !v) return u+v;
25        pushdown(u); pushdown(v);
26        int t = Rand() % (node[u].size + node[v].size), r;
27        if (t < node[u].size) {
28            r = copy(u);
29            node[r].rc = merge(node[u].rc, v);
30        } else {
31            r = copy(v);
32            node[r].lc = merge(u, node[v].lc);
33        }
34        pushup(r);
35        return r;
36    }
37    int split(int u, int x, int y) {
38        if (x > y) return 0;
39        pushdown(u);
40        if (x == 1 && y == node[u].size) return u;
41        if (y <= node[node[u].lc].size) return split(node[u].lc, x, y);
42        int t = node[node[u].lc].size + 1;
43        if (x > t) return split(node[u].rc, x-t, y-t);
44        int num = copy(u);
45        node[num].lc = split(node[u].lc, x, t-1);
46        node[num].rc = split(node[u].rc, 1, y-t);
47        pushup(num);
```



```

47     return num;
48 }
49 void insert(int x, char c) {
50     int t1 = split(root, 1, x), t2 = split(root, x+1, node[root].size);
51     node[++n].key = c; node[n].size = 1;
52     root = merge(merge(t1, n), t2);
53 }
54 void del(int x, int y) {
55     int t1 = split(root, 1, x-1), t2 = split(root, y+1, node[root].size);
56     root = merge(t1, t2);
57 }
58 void copy(int l, int r, int x) {
59     int t1 = split(root, 1, x), t2 = split(root, 1, r), t3 = split(root, x+1, node[root].
60         size);
61     root = merge(merge(t1, t2), t3);
62 }
63 void reverse(int x, int y) {
64     int t1 = split(root, 1, x-1), t2 = split(root, x, y), t3 = split(root, y+1, node[root].
65         size);
66     node[t2].reverse ^= 1;
67     root = merge(merge(t1, t2), t3);
68 }
69 char query(int k) {
70     int x = root;
71     while (1) {
72         pushdown(x);
73         if (k <= node[node[x].lc].size) x = node[x].lc;
74         else
75             if (k == node[node[x].lc].size + 1) return node[x].key;
76             else
77                 k -= node[node[x].lc].size + 1, x = node[x].rc;
78     }
79 }
80 } treap;

```

4 树

4.1 动态树

接口:

command(x, y) : 将 x 到 y 路径的 Splay Tree 分离出来。

linkcut(u1, v1, u2, v2) : 将树中原有的边 (u1, v1) 删除, 加入一条新边 (u2, v2)

```
1 struct DynamicTREE{
2     struct NODE{
3         int father, son[2], top, size, reverse;
4     } splay[N];
5     void init(int i, int fat) {
6         splay[i].father = splay[i].son[0] = splay[i].son[1] = 0;
7         splay[i].top = fat; splay[i].size = 1; splay[i].reverse = 0;
8     }
9     void pushdown(int x) {
10        if (!x) return;
11        int s0 = splay[x].son[0], s1 = splay[x].son[1];
12        if (splay[x].reverse) {
13            splay[s0].reverse ^= 1;
14            splay[s1].reverse ^= 1;
15            swap(splay[x].son[0], splay[x].son[1]);
16            splay[x].reverse = 0;
17        }
18        s0 = splay[x].son[0], s1 = splay[x].son[1];
19        splay[s0].top = splay[s1].top = splay[x].top;
20    }
21    void pushup(int x) {
22        if (!x) return;
23        pushdown(splay[x].son[0]);
24        pushdown(splay[x].son[1]);
25        splay[x].size = splay[splay[x].son[0]].size + splay[splay[x].son[1]].size + 1;
26    }
27    void sc(int x, int y, int w, bool Auto=true) {
28        splay[x].son[w] = y;
29        splay[y].father = x;
30        if (Auto) {
31            pushup(y);
32            pushup(x);
33        }
34    }
35    int top, tush[N];
36    void flowdown(int x) {
37        for (top=1; x; top++, x = splay[x].father) tush[top] = x;
38        for (; top; top--) pushdown(tush[top]);
39    }
40    void rotate(int x) {
41        if (!x) return;
42        int y = splay[x].father;
43        int w = splay[y].son[1] == x;
44        pushdown(y);
45        pushdown(x);
46        sc(splay[y].father, x, splay[splay[y].father].son[1]==y, false);
47        sc(y, splay[x].son[w^1], w, false);
48        sc(x, y, w^1, false);
49        pushup(y);
```

```

50     pushup(x);
51 }
52 void Splay(int x, int rt=0) {
53     if (!x) return;
54     flowdown(x);
55     while (splay[x].father != rt) {
56         int y = splay[x].father;
57         int w = splay[y].son[1]==x;
58         if (splay[y].father != rt && splay[splay[y].father].son[w] == y) rotate(y);
59         rotate(x);
60     }
61 }
62 void split(int x) {
63     int y = splay[x].son[1];
64     if (!y) return;
65     splay[y].father = 0;
66     splay[x].son[1] = 0;
67     splay[y].top = x;
68     pushup(x);
69 }
70 void access(int x) {
71     int y = 0;
72     while (x) {
73         Splay(x);
74         split(x);
75         sc(x, y, 1);
76         Splay(x);
77         y = x;
78         x = splay[x].top;
79     }
80 }
81 void changeroot(int x) {
82     access(x);
83     Splay(x);
84     splay[x].reverse = 1;
85     Splay(x);
86 }
87 void command(int x, int y, ...) {
88     LL ans = 0;
89     changeroot(x);
90     access(y);
91     Splay(x);
92     //then you can modify the Splay Tree
93 }
94 void linkcut(int u1, int v1, int u2, int v2) {
95     changeroot(u1);
96     access(v1);
97     Splay(u1); split(u1);
98     splay[v1].top = 0;
99     access(u2); changeroot(u2);
100    access(v2); changeroot(v2);
101    Splay(u2); Splay(v2);
102    splay[v2].top = u2;
103 }
104 } lct;

```

5 图

5.1 欧拉回路

欧拉回路:

无向图: 每个顶点的度数都是偶数, 则存在欧拉回路。

有向图: 每个顶点的入度 = 出度, 则存在欧拉回路。

欧拉路径:

无向图: 当且仅当该图所有顶点的度数为偶数, 或者除了两个度数为奇数外其余的全是偶数。

有向图: 当且仅当该图所有顶点出度 = 入度或者一个顶点出度 = 入度 + 1, 另一个顶点入度 = 出度 + 1, 其他顶点出度 = 入度。下面 $O(n+m)$ 求欧拉回路的代码中, n 为点数, m 为边数, 若有解则依次输出经过的边的编号, 若是无向图, 则正数表示 x 到 y , 负数表示 y 到 x 。

```
1 namespace UndirectedGraph{
2     int n,m,i,x,y,d[N],g[N],v[M<<1],w[M<<1],vis[M<<1],nxt[M<<1],ed;
3     int ans[M],cnt;
4     void add(int x,int y,int z){
5         d[x]++;
6         v[++ed]=y;w[ed]=z;nxt[ed]=g[x];g[x]=ed;
7     }
8     void dfs(int x){
9         for(int&i=g[x];i;){
10             if(vis[i]){i=nxt[i];continue;}
11             vis[i]=vis[i^1]=1;
12             int j=w[i];
13             dfs(v[i]);
14             ans[++cnt]=j;
15         }
16     }
17     void solve(){
18         scanf("%d%d",&n,&m);
19         for(i=ed=1;i<=m;i++) scanf("%d%d",&x,&y),add(x,y,i),add(y,x,-i);
20         for(i=1;i<=n;i++) if(d[i]&1){puts("NO");return;}
21         for(i=1;i<=n;i++) if(g[i]){dfs(i);break;}
22         for(i=1;i<=n;i++) if(g[i]){puts("NO");return;}
23         puts("YES");
24         for(i=m;i;i--)printf("%d_",ans[i]);
25     }
26 }
27 namespace DirectedGraph{
28     int n,m,i,x,y,d[N],g[N],v[M],vis[M],nxt[M],ed;
29     int ans[M],cnt;
30     void add(int x,int y){
31         d[x]++;d[y]--;
32         v[++ed]=y;nxt[ed]=g[x];g[x]=ed;
33     }
34     void dfs(int x){
35         for(int&i=g[x];i;){
36             if(vis[i]){i=nxt[i];continue;}
37             vis[i]=1;
38             int j=i;
39             dfs(v[i]);
40             ans[++cnt]=j;
41         }
42     }
```

```

43     void solve() {
44         scanf("%d%d", &n, &m);
45         for(i=1; i<=m; i++) scanf("%d%d", &x, &y), add(x, y);
46         for(i=1; i<=n; i++) if(d[i]) {puts("NO"); return;}
47         for(i=1; i<=n; i++) if(g[i]) {dfs(i); break;}
48         for(i=1; i<=n; i++) if(g[i]) {puts("NO"); return;}
49         puts("YES");
50         for(i=m; i; i--) printf("%d_", ans[i]);
51     }
52 }

```

5.2 最短路径

5.2.1 Dijkstra

```

1  #include <queue>
2  using namespace std;
3  struct EDGE { int adj, w, next; } edge[M*2];
4  struct dat { int id, dist; dat(int id=0, int dist=0) : id(id), dist(dist) {} };
5  struct cmp { bool operator () (const dat &a, const dat &b) { return a.dist > b.dist; } };
6  priority_queue < dat, vector<dat>, cmp > q;
7  int n, top, gh[N], v[N], dist[N];
8  void addedge(int x, int y, int w) {
9      edge[++top].adj = y;
10     edge[top].w = w;
11     edge[top].next = gh[x];
12     gh[x] = top;
13 }
14 int dijkstra(int s, int t) {
15     memset(dist, 63, sizeof(dist));
16     memset(v, 0, sizeof(v));
17     dist[s] = 0;
18     q.push(dat(s, 0));
19     while (!q.empty()) {
20         dat x = q.top(); q.pop();
21         if (v[x.id]) continue; v[x.id] = 1;
22         for (int p=gh[x.id]; p; p=edge[p].next) {
23             if (x.dist + edge[p].w < dist[edge[p].adj]) {
24                 dist[edge[p].adj] = x.dist + edge[p].w;
25                 q.push(dat(edge[p].adj, dist[edge[p].adj]));
26             }
27         }
28     }
29     return dist[t];
30 }

```

5.2.2 SPFA

```

1  struct EDGE { int adj, w, next; } edge[M*2];
2  int n, m, top, gh[N], v[N], cnt[N], q[N], dist[N], head, tail;
3  void addedge(int x, int y, int w) {
4      edge[++top].adj = y;
5      edge[top].w = w;
6      edge[top].next = gh[x];

```

```

7     gh[x] = top;
8 }
9 int spfa(int S, int T) {
10     memset(v, 0, sizeof(v));
11     memset(cnt, 0, sizeof(cnt));
12     memset(dist, 63, sizeof(dist));
13     head = 0, tail = 1;
14     dist[S] = 0; q[1] = S;
15     while (head != tail) {
16         (head += 1) %= N;
17         int x = q[head]; v[x] = 0;
18         ++cnt[x]; if (cnt[x] > n) return -1;
19         for (int p=gh[x]; p; p=edge[p].next)
20             if (dist[x] + edge[p].w < dist[edge[p].adj]) {
21                 dist[edge[p].adj] = dist[x] + edge[p].w;
22                 if (!v[edge[p].adj]) {
23                     v[edge[p].adj] = 1;
24                     (tail += 1) %= N;
25                     q[tail] = edge[p].adj;
26                 }
27             }
28     }
29     return dist[T];
30 }

```

5.3 K 短路

接口:

kthsp::init(n) : 初始化并设置节点个数为n

kthsp::add(x, y, w) : 添加一条x到y的有向边

kthsp::work(S, T, k) : 求S到T的第k短路

```

1 #include <queue>
2
3 #define N 200020
4 #define M 400020
5 #define LOGM 20
6 #define LL long long
7 #define inf (1LL<<61)
8
9 namespace pheap {
10     struct Node {
11         int next, son[2];
12         LL val;
13     } node[M*LOGM];
14     int LOG[M];
15     int root[M], size[M*LOGM], top;
16     int add() {
17         ++top; assert(top < M*LOGM);
18         node[top].next = node[top].son[0] = node[top].son[1] = 0;
19         node[top].val = inf;
20         return top;
21     }
22     int copy(int x) { int t = add(); node[t] = node[x]; return t; }
23     void init() {

```

```

24     top = -1; add();
25     for (int i=2;i<M;i++) LOG[i] = LOG[i>>1] + 1;
26 }
27 void upd(int x, int &next, LL &val) {
28     if (val < node[x].val) {
29         swap(val, node[x].val);
30         swap(next, node[x].next);
31     }
32 }
33 void insert(int x, int next, LL val) {
34     int sz = size[root[x]] + 1;
35     root[x] = copy(root[x]);
36     size[root[x]] = sz; x = root[x];
37     upd(x, next, val);
38     for (int i=LOG[sz]-1;i>=0;i--) {
39         int ind = (sz>>i)&1;
40         node[x].son[ind] = copy(node[x].son[ind]);
41         x = node[x].son[ind];
42         upd(x, next, val);
43     }
44 }
45 };
46
47 namespace kthsp {
48     using namespace pheap;
49     struct EDGE {
50         int adj, w, next;
51     } edge[2][M];
52     struct W {
53         int x, y, w;
54     } e[M];
55     bool has_init = 0;
56     int n, m, top[2], gh[2][N], v[N];
57     LL dist[N];
58     void init(int n1) {
59         has_init = 1;
60         n = n1; m = 0;
61         memset(top, 0, sizeof(top));
62         memset(gh, 0, sizeof(gh));
63         for (int i=1;i<=n;i++) dist[i] = inf;
64     }
65     void addedge(int id, int x, int y, int w) {
66         edge[id][++top[id]].adj = y;
67         edge[id][top[id]].w = w;
68         edge[id][top[id]].next = gh[id][x];
69         gh[id][x] = top[id];
70     }
71     void add(int x, int y, int w) {
72         assert(has_init);
73         e[++m].x=x; e[m].y=y; e[m].w=w;
74     }
75     int q[N], best[N], bestw[N];
76     int deg[N];
77     void spfa(int S) {
78         for (int i=1;i<=n;i++) deg[i] = 0;
79         for (int i=1;i<=m;i++) deg[e[i].x] ++;
80         int head = 0, tail = 1;

```

```

81     dist[S] = 0; q[1] = S;
82     while (head != tail) {
83         (head += 1) %= N;
84         int x = q[head];
85         for (int p=gh[1][x]; p; p=edge[1][p].next) {
86             if (dist[x] + edge[1][p].w < dist[edge[1][p].adj]) {
87                 dist[edge[1][p].adj] = dist[x] + edge[1][p].w;
88                 best[edge[1][p].adj] = x;
89                 bestw[edge[1][p].adj] = p;
90             }
91             if (!--deg[edge[1][p].adj]) {
92                 (tail += 1) %= N;
93                 q[tail] = edge[1][p].adj;
94             }
95         }
96     }
97 }
98 void dfs(int x) {
99     if (v[x]) return; v[x] = 1;
100    if (best[x]) root[x] = root[best[x]];
101    for (int p=gh[0][x]; p; p=edge[0][p].next)
102        if (dist[edge[0][p].adj] != inf && bestw[x] != p) {
103            insert(x, edge[0][p].adj, edge[0][p].w + dist[edge[0][p].adj] - dist[x]);
104        }
105    for (int p=gh[1][x]; p; p=edge[1][p].next)
106        if (best[edge[1][p].adj] == x)
107            dfs(edge[1][p].adj);
108 }
109 typedef pair<LL,int> pli;
110 priority_queue <pli, vector<pli>, greater<pli> > pq;
111 LL work(int S, int T, int k) {
112     assert(has_init);
113     n++; add(T, n, 0);
114     if (S == T) k ++;
115     T = n;
116     for (int i=1;i<=m;i++) {
117         addedge(0, e[i].x, e[i].y, e[i].w);
118         addedge(1, e[i].y, e[i].x, e[i].w);
119     }
120     spfa(T);
121     root[T] = 0; pheap::init();
122     memset(v, 0, sizeof(v));
123     dfs(T);
124     while (!pq.empty()) pq.pop();
125     if (k == 1) return dist[S];
126     if (root[S]) pq.push(make_pair(dist[S] + node[root[S]].val, root[S]));
127     while (k--) {
128         if (pq.empty()) return inf;
129         pli now = pq.top(); pq.pop();
130         if (k == 1) return now.first;
131         int x = node[now.second].next, u = node[now.second].son[0], v = node[now.second].
            son[1];
132         if (root[x]) pq.push(make_pair(now.first + node[root[x]].val, root[x]));
133         if (u) pq.push(make_pair(now.first - node[now.second].val + node[u].val, u));
134         if (v) pq.push(make_pair(now.first - node[now.second].val + node[v].val, v));
135     }
136     return 0;

```



```

137     }
138 };

```

5.4 Tarjan

割点的判断：一个顶点 u 是割点，当且仅当满足 (1) 或 (2)：

(1) u 为树根，且 u 有多于一个子树

(2) u 不为树根，且满足存在 (u, v) 为树枝边（ u 为 v 的父亲），使得 $dfn[u] \leq low[v]$

桥的判断：一条无向边 (u, v) 是桥，当且仅当 (u, v) 为树枝边，满足 $dfn[u] < low[v]$

```

1 struct EDGE { int adj, next; } edge[M];
2 int n, m, top, gh[N];
3 int dfn[N], low[N], cnt, ind, stop, instack[N], stack[N], belong[N];
4 void addedge(int x, int y) {
5     edge[++top].adj = y;
6     edge[top].next = gh[x];
7     gh[x] = top;
8 }
9 void tarjan(int x) {
10     dfn[x] = low[x] = ++ind;
11     instack[x] = 1; stack[++stop] = x;
12     for (int p=gh[x]; p; p=edge[p].next)
13         if (!dfn[edge[p].adj]) {
14             tarjan(edge[p].adj);
15             low[x] = min(low[x], low[edge[p].adj]);
16         } else if (instack[edge[p].adj]) {
17             low[x] = min(low[x], dfn[edge[p].adj]);
18         }
19     if (dfn[x] == low[x]) {
20         ++cnt; int tmp=0;
21         while (tmp!=x) {
22             tmp = stack[stop--];
23             belong[tmp] = cnt;
24             instack[tmp] = 0;
25         }
26     }
27 }

```

5.5 2-SAT

```

1 #define N number_of_vertex
2 #define M number_of_edges
3
4 struct MergePoint {
5     struct EDGE {
6         int adj, next;
7     } edge[M];
8     int ex[M], ey[M];
9     bool instack[N];
10    int gh[N], top, dfn[N], low[N], cnt, ind, stop, stack[N], belong[N];
11    void init() {
12        cnt = ind = stop = top = 0;
13        memset(dfn, 0, sizeof(dfn));
14        memset(instack, 0, sizeof(instack));

```

```

15     memset(gh, 0, sizeof(gh));
16 }
17 void addedge(int x, int y) { //reverse
18     std::swap(x, y);
19     edge[++top].adj = y;
20     edge[top].next = gh[x];
21     gh[x] = top;
22     ex[top] = x;
23     ey[top] = y;
24 }
25 void tarjan(int x) {
26     dfn[x] = low[x] = ++ind;
27     instack[x] = 1; stack[++stop] = x;
28     for (int p=gh[x]; p; p=edge[p].next)
29         if (!dfn[edge[p].adj]) {
30             tarjan(edge[p].adj);
31             low[x] = std::min(low[x], low[edge[p].adj]);
32         } else if (instack[edge[p].adj]) {
33             low[x] = std::min(low[x], dfn[edge[p].adj]);
34         }
35     if (dfn[x] == low[x]) {
36         ++cnt; int tmp = 0;
37         while (tmp!=x) {
38             tmp = stack[stop--];
39             belong[tmp] = cnt;
40             instack[tmp] = 0;
41         }
42     }
43 }
44 void work() {
45     for (int i = (__first__); i <= (__last__); ++i)
46         if (!dfn[i])
47             tarjan(i);
48 }
49 } merge;
50
51 struct Topsort {
52     struct EDGE {
53         int adj, next;
54     } edge[M];
55     int n, top, gh[N], ops[N], deg[N], ans[N];
56     std::queue<int> q;
57     void init() {
58         n = merge.cnt; top = 0;
59         memset(gh, 0, sizeof(gh));
60         memset(deg, 0, sizeof(deg));
61     }
62     void addedge(int x, int y) {
63         if (x == y) return;
64         edge[++top].adj = y;
65         edge[top].next = gh[x];
66         gh[x] = top;
67         ++deg[y];
68     }
69     void work() {
70         for (int i = 1; i <= n; ++i)
71             if (!deg[i])

```

```

72         q.push(i);
73     while (!q.empty()) {
74         int x = q.front();
75         q.pop();
76         for (int p = gh[x]; p; p = edge[p].next)
77             if (--deg[edge[p].adj])
78                 q.push(edge[p].adj);
79         if (ans[x]) continue;
80         ans[x] = -1; //not selected
81         ans[ops[x]] = 1; //selected
82     }
83 }
84 } ts;

```

调用示例:

```

1     merge.init();
2     merge.addedge();
3     merge.work();
4     for (int i = 1; i <= n; ++i) {
5         if (merge.belong[U(i, 0)] == merge.belong[U(i, 1)]) {
6             puts("NO");
7             return 0;
8         }
9         ts.ops[merge.belong[U(i, 0)]] = merge.belong[U(i, 1)];
10        ts.ops[merge.belong[U(i, 1)]] = merge.belong[U(i, 0)];
11    }
12    ts.init();
13    ts.work();
14    puts("YES");
15    for (int i = 1; i <= n; ++i) {
16        int x = U(i, 0), y = U(i, 1);
17        x = merge.belong[x], y = merge.belong[y];
18        x = ts.ans[x], y = ts.ans[y];
19        if (x == 1) puts("0_is_selected");
20        if (y == 1) puts("1_is_selected");
21    }

```

5.6 统治者树 (Dominator Tree)

Dominator Tree 可以解决判断一类有向图必经点的问题。

$idom[x]$ 表示离 x 最近的必经点 (重编号后)。将 $idom[x]$ 作为 x 的父亲, 构成一棵 Dominator Tree 接口:

`void dominator::init(int n);` 初始化, 有向图节点数为 n

`void dominator::addedge(int u, int v);` 添加一条有向边 (u, v)

`void dominator::work(int root);` 以 $root$ 为根, 建立一棵 Dominator Tree

结果的返回:

在执行 `dominator::work(int root);` 后, 树边保存在 `vector<int> tree[N]` 中

```

1 namespace dominator {
2     vector<int> g[N], rg[N], bucket[N], tree[N];
3     int n, ind, idom[N], sdom[N], dfn[N], dsu[N], father[N], label[N], rev[N];
4     void dfs(int x) {
5         ++ind;
6         dfn[x] = ind; rev[ind] = x;

```

```

7      label[ind] = dsu[ind] = sdom[ind] = ind;
8      for (auto p : g[x]) {
9          if (!dfn[p]) dfs(p), father[dfn[p]] = dfn[x];
10         rg[dfn[p]].push_back(dfn[x]);
11     }
12 }
13 void init(int n1) {
14     n = n1; ind = 0;
15     for (int i = 1; i <= n; ++i) {
16         g[i].clear();
17         rg[i].clear();
18         bucket[i].clear();
19         tree[i].clear();
20         dfn[i] = 0;
21     }
22 }
23 void addedge(int u, int v) {
24     g[u].push_back(v);
25 }
26 int find(int x, int step=0) {
27     if (dsu[x] == x) return step ? -1 : x;
28     int y = find(dsu[x], 1);
29     if (y < 0) return x;
30     if (sdom[label[dsu[x]]] < sdom[label[x]])
31         label[x] = label[dsu[x]];
32     dsu[x] = y;
33     return step ? dsu[x] : label[x];
34 }
35 void work(int root) {
36     dfs(root); n = ind;
37     for (int i = n; i; --i) {
38         for (auto p : rg[i])
39             sdom[i] = min(sdom[i], sdom[find(p)]);
40         if (i > 1) bucket[sdom[i]].push_back(i);
41         for (auto p : bucket[i]) {
42             int u = find(p);
43             if (sdom[p] == sdom[u]) idom[p] = sdom[p];
44             else idom[p] = u;
45         }
46         if (i > 1) dsu[i] = father[i];
47     }
48     for (int i = 2; i <= n; ++i) {
49         if (idom[i] != sdom[i])
50             idom[i] = idom[idom[i]];
51         tree[rev[i]].push_back(rev[idom[i]]);
52         tree[rev[idom[i]]].push_back(rev[i]);
53     }
54 }
55 };

```

5.7 网络流

5.7.1 最大流

```

1 struct EDGE { int adj, w, next; } edge[M];
2 int n, top, gh[N], nrl[N];

```

```

3 void addedge(int x, int y, int w) {
4     edge[++top].adj = y;
5     edge[top].w = w;
6     edge[top].next = gh[x];
7     gh[x] = top;
8     edge[++top].adj = x;
9     edge[top].w = 0;
10    edge[top].next = gh[y];
11    gh[y] = top;
12 }
13 int dist[N], q[N];
14 int bfs() {
15     memset(dist, 0, sizeof(dist));
16     q[1] = S; int head = 0, tail = 1; dist[S] = 1;
17     while (head != tail) {
18         int x = q[++head];
19         for (int p=gh[x]; p; p=edge[p].next)
20             if (edge[p].w && !dist[edge[p].adj]) {
21                 dist[edge[p].adj] = dist[x] + 1;
22                 q[++tail] = edge[p].adj;
23             }
24     }
25     return dist[T];
26 }
27 int dinic(int x, int delta) {
28     if (x==T) return delta;
29     for (int& p=nrl[x]; p && delta; p=edge[p].next)
30         if (edge[p].w && dist[x]+1 == dist[edge[p].adj]) {
31             int dd = dinic(edge[p].adj, min(delta, edge[p].w));
32             if (!dd) continue;
33             edge[p].w -= dd;
34             edge[p^1].w += dd;
35             return dd;
36         }
37     return 0;
38 }
39 int work() {
40     int ans = 0;
41     while (bfs()) {
42         memcpy(nrl, gh, sizeof(gh));
43         int t; while (t = dinic(S, inf)) ans += t;
44     }
45     return ans;
46 }

```

5.7.2 上下界有源汇网络流

T 向 S 连容量为正无穷的边，将有源汇转化为无源汇。

每条边容量减去下界，设 $in[i]$ 表示流入 i 的下界之和减去流出 i 的下界之和。

新建超级源汇 SS, TT ，对于 $in[i] > 0$ 的点， SS 向 i 连容量为 $in[i]$ 的边。对于 $in[i] < 0$ 的点， i 向 TT 连容量为 $-in[i]$ 的边。

求出以 SS, TT 为源汇的最大流，如果等于 $\sum in[i] (in[i] > 0)$ ，则存在可行流。再求出 S, T 为源汇的最大流即为最大流。

费用流：建完图后等价于求以 SS, TT 为源汇的费用流。

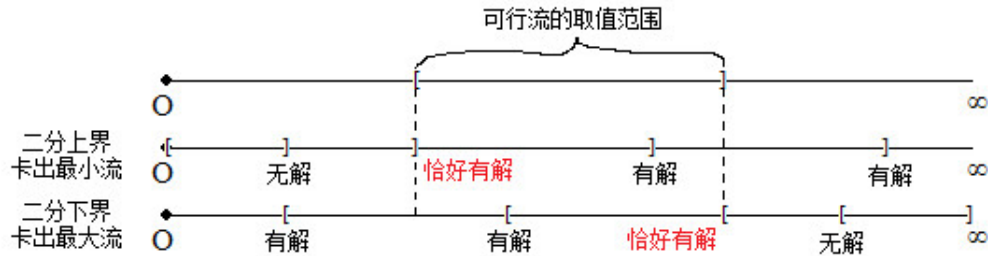
5.7.3 上下界无源汇网络流

1. 怎样求无源汇有上下界网络的可行流？

由于有源汇的网络我们先要转化成无源汇，所以本来就无源汇的网络不用作特殊处理。

2. 怎样求无源汇有上下界网络的最大流、最小流？

一种简易的方法是采用二分的思想，不断通过可行流的存在与否对 (t, s) 边的上下界 U, L 进行调整。求最大流时令 $U = \infty$ 并二分 L ；求最小流时令 $L = 0$ 并二分 U 。道理很简单，因为可行流的取值范围是一段连续的区间，我们只要通过二分找到有解和无解的分界线即可。



5.7.4 费用流

```
1 #define inf 0x3f3f3f3f
2 struct NetWorkFlow {
3     struct EDGE {
4         int adj, w, cost, next;
5     } edge[M*2];
6     int gh[N], q[N], dist[N], v[N], pre[N], prev[N], top;
7     int S, T;
8     void addedge(int x, int y, int w, int cost) {
9         edge[++top].adj = y;
10        edge[top].w = w;
11        edge[top].cost = cost;
12        edge[top].next = gh[x];
13        gh[x] = top;
14        edge[++top].adj = x;
15        edge[top].w = 0;
16        edge[top].cost = -cost;
17        edge[top].next = gh[y];
18        gh[y] = top;
19    }
20    void clear() {
21        top = 1;
22        memset(gh, 0, sizeof(gh));
23    }
24    int spfa() {
25        memset(dist, 63, sizeof(dist));
26        memset(v, 0, sizeof(v));
27        int head = 0, tail = 1;
28        q[1] = S; v[S] = 1; dist[S] = 0;
29        while (head != tail) {
30            (head += 1) %= N;
```

```

31     int x = q[head];
32     v[x] = 0;
33     for (int p=gh[x]; p; p=edge[p].next)
34         if (edge[p].w && dist[x] + edge[p].cost < dist[edge[p].adj]) {
35             dist[edge[p].adj] = dist[x] + edge[p].cost;
36             pre[edge[p].adj] = x;
37             prev[edge[p].adj] = p;
38             if (!v[edge[p].adj]) {
39                 v[edge[p].adj] = 1;
40                 (tail += 1) %= N;
41                 q[tail] = edge[p].adj;
42             }
43         }
44     }
45     return dist[T] != inf;
46 }
47 int work() {
48     int ans = 0;
49     while (spfa()) {
50         int mx = inf;
51         for (int x=T; x!=S; x=pre[x])
52             mx = min(edge[prev[x]].w, mx);
53         ans += dist[T] * mx;
54         for (int x=T; x!=S; x=pre[x]) {
55             edge[prev[x]].w -= mx;
56             edge[prev[x]^1].w += mx;
57         }
58     }
59     return ans;
60 }
61 } nwf;

```

5.7.5 zkw费用流

```

1  #define inf 0x3f3f3f3f
2  struct NetWorkFlow {
3      struct EDGE {
4          int adj, w, cost, next;
5      } edge[M*2];
6      int gh[N], top;
7      int S, T;
8      void addedge(int x, int y, int w, int cost) {
9          edge[++top].adj = y;
10         edge[top].w = w;
11         edge[top].cost = cost;
12         edge[top].next = gh[x];
13         gh[x] = top;
14         edge[++top].adj = x;
15         edge[top].w = 0;
16         edge[top].cost = -cost;
17         edge[top].next = gh[y];
18         gh[y] = top;
19     }
20     void clear() {
21         top = 1;
22         memset(gh, 0, sizeof(gh));

```

```

23     }
24     int cost, d[N], slk[N], v[N];
25     int aug(int x, int f) {
26         int left = f;
27         if (x == T) {
28             cost += f * d[S];
29             return f;
30         }
31         v[x] = true;
32         for (int p=gh[x]; p; p=edge[p].next)
33             if (edge[p].w && !v[edge[p].adj]) {
34                 int t = d[edge[p].adj] + edge[p].cost - d[x];
35                 if (t == 0) {
36                     int delt = aug(edge[p].adj, min(left, edge[p].w));
37                     if (delt > 0) {
38                         edge[p].w -= delt;
39                         edge[p^1].w += delt;
40                         left -= delt;
41                     }
42                     if (left == 0) return f;
43                 } else {
44                     if (t < slk[edge[p].adj])
45                         slk[edge[p].adj] = t;
46                 }
47             }
48         return f-left;
49     }
50     bool modlabel() {
51         int delt = inf;
52         for (int i=1; i<=T; i++)
53             if (!v[i]) {
54                 if (slk[i] < delt) delt = slk[i];
55                 slk[i] = inf;
56             }
57         if (delt == inf) return true;
58         for (int i=1; i<=T; i++)
59             if (v[i]) d[i] += delt;
60         return false;
61     }
62     int work() {
63         cost = 0;
64         memset(d, 0, sizeof(d));
65         memset(slk, 63, sizeof(slk));
66         do {
67             do {
68                 memset(v, 0, sizeof(v));
69             } while (aug(S, inf));
70         } while (!modlabel());
71         return cost;
72     }
73 } nwf;

```


6 数学

6.1 扩展欧几里得解同余方程

ans[] 保存的是循环节内所有的解

```
1 int exgcd(int a,int b,int&x,int&y){
2     if(!b) return x=1,y=0,a;
3     int d=exgcd(b,a%b,x,y),t=x;
4     return x=y,y=t-a/b*y,d;
5 }
6 void cal(ll a,ll b,ll n){//ax=b(mod n)
7     ll x,y,d=exgcd(a,n,x,y);
8     if(b%d) return;
9     x=(x%n+n)%n;
10    ans[cnt=1]=x*(b/d)%(n/d);
11    for(ll i=1;i<d;i++) ans[++cnt]=(ans[1]+i*n/d)%n;
12 }
```

6.2 同余方程组

```
1 int n,flag,k,m,a,r,d,x,y;
2 int main(){
3     scanf("%d",&n);
4     flag=k=1,m=0;
5     while(n--){
6         scanf("%d%d",&a,&r);//ans%a=r
7         if(flag){
8             d=exgcd(k,a,x,y);
9             if((r-m)%d){flag=0;continue;}
10            x=(x*(r-m)/d+a/d)%(a/d),y=k/d*a,m=(x*k+m)%y;
11            if(m<0)m+=y;
12            k=y;
13        }
14    }
15    printf("%d",flag?m:-1);//若flag说明有解解为=l,,ki+m,为任意整数i
16 }
```

6.3 卡特兰数

$$h_1 = 1, h_n = \frac{h_{n-1}(4n-2)}{n+1} = \frac{C(2n,n)}{n+1} = C(2n,n) - C(2n,n-1)$$

在一个格点阵列中, 从 $(0,0)$ 点走到 (n,m) 点且不过对角线 $x=y$ 的方案数 $(x>y)$:

$$C(n+m-1,m) - C(n+m-1,m-1)$$

在一个格点阵列中, 从 $(0,0)$ 点走到 (n,m) 点且不穿过对角线 $x=y$ 的方案数 $(x\geq y)$:

$$C(n+m,m) - C(n+m,m-1)$$

6.4 斯特林数

6.4.1 第一类斯特林数

第一类 Stirling 数 $S(p,k)$ 的一个组合学解释是: 将 p 个物体排成 k 个非空循环排列的方法数。

$S(p,k)$ 的递推公式: $S(p,k) = (p-1)S(p-1,k) + S(p-1,k-1), 1 \leq k \leq p-1$

边界条件: $S(p,0) = 0, p \geq 1$ $S(p,p) = 1, p \geq 0$

6.4.2 第二类斯特林数

第二类 Stirling 数 $S(p, k)$ 的一个组合学解释是：将 p 个物体划分成 k 个非空的不可辨别（可以理解为盒子没有编号）集合的方法数。

$S(p, k)$ 的递推公式： $S(p, k) = kS(p-1, k) + S(p-1, k-1), 1 \leq k \leq p-1$

边界条件： $S(p, 0) = 0, p \geq 1$ $S(p, p) = 1, p \geq 0$

也有卷积形式：

$$S(n, m) = \frac{1}{m!} \sum_{k=0}^m (-1)^k C(m, k) (m-k)^n = \sum_{k=0}^m \frac{(-1)^k (m-k)^n}{k! (m-k)!} = \sum_{k=0}^m \frac{(-1)^k}{k!} \times \frac{(m-k)^n}{(m-k)!}$$

6.5 错排公式

$$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-2} + D_{n-1})$$

6.6 Lucas定理

接口：

初始化： `void lucas::init();`

计算 $C(n, m) \% \text{mod}$ 的值： `LL lucas::Lucas(LL n, LL m);`

```
1 #define mod 110119
2 #define LL long long
3 namespace lucas {
4     LL fac[mod+1], facv[mod+1];
5     LL power(LL base, LL times) {
6         LL ans = 1;
7         while (times) {
8             if (times&1) (ans *= base) %= mod;
9             (base *= base) %= mod;
10            times >>= 1;
11        }
12        return ans;
13    }
14    void init() {
15        fac[0] = 1; for (int i=1; i<mod; i++) fac[i] = (fac[i-1] * i) % mod;
16        facv[mod-1] = power(fac[mod-1], mod-2);
17    }
18    LL C(unsigned LL n, unsigned LL m) {
19        if (n < m) return 0;
20        return (fac[n] * facv[m] % mod * facv[n-m] % mod) % mod;
21    }
22    LL Lucas(unsigned LL n, unsigned LL m)
23    {
24        if (m == 0) return 1;
25        return (C(n%mod, m%mod) * Lucas(n/mod, m/mod)) % mod;
26    }
27 };
```

6.7 高斯消元

6.7.1 行列式

```

1  int ans = 1;
2  for (int i=0;i<n;i++) {
3      for (int j=i;j<n;j++)
4          if (g[j][i]) {
5              for (int k=i;k<n;k++)
6                  swap(g[i][k], g[j][k]);
7              if (j != i) ans *= -1;
8              break;
9          }
10     if (g[i][i] == 0) {
11         ans = 0;
12         break;
13     }
14     for (int j=i+1;j<n;j++) {
15         while (g[j][i]) {
16             int t = g[i][i] / g[j][i];
17             for (int k=i;k<n;k++)
18                 g[i][k] = (g[i][k] + mod - ((LL)t * g[j][k] % mod)) % mod;
19             for (int k=i;k<n;k++)
20                 swap(g[i][k], g[j][k]);
21             ans *= -1;
22         }
23     }
24 }
25 for (int i=0;i<n;i++)
26     ans = ((LL)ans * g[i][i]) % mod;
27 ans = (ans % mod + mod) % mod;
28 printf("%d\n", ans);

```

6.7.2 Matrix-Tree定理

对于一张图，建立矩阵 C ， $C[i][i] = i$ 的度数，若 i, j 之间有边，那么 $C[i][j] = -1$ ，否则为 0。这张图的生成树个数等于矩阵 C 的 $n-1$ 阶行列式的值。

6.8 调和级数

$\sum_{i=1}^n \frac{1}{i}$ 在 n 较大时约等于 $\ln(n) + r$ ， r 为欧拉常数，约等于 0.5772156649015328。

6.9 曼哈顿距离的变换

$$|x_1 - x_2| + |y_1 - y_2| = \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|)$$

6.10 线性筛素数

```

1  mu[1]=phi[1]=1;top=0;
2  for (int i=2;i<N;i++) {
3      if (!v[i]) prime[++top]=i, mu[i] = -1, phi[i] = i-1;
4      for (int j=1;i*prime[j]<N && j<=top;j++) {
5          v[i*prime[j]] = 1;
6          if (i%prime[j]) {
7              mu[i*prime[j]] = -mu[i];
8              phi[i*prime[j]] = phi[i] * (prime[j]-1);

```

```

9         } else {
10             mu[i*prime[j]] = 0;
11             phi[i*prime[j]] = phi[i] * prime[j];
12             break;
13         }
14     }
15 }

```

6.11 FFT

```

1  typedef complex<double> comp;
2  namespace FFT {
3      comp A[N], B[N], omega[N];
4      void transform(comp *x, int len) {
5          for (int i=1, j=len/2; i<len-1; i++) {
6              if (i<j) swap(x[i], x[j]);
7              int k = len/2;
8              while (j>=k) {
9                  j-=k;
10                 k/=2;
11             }
12             if (j<k) j+=k;
13         }
14     }
15     void fft(comp *x, int len, int reverse) {
16         transform(x, len);
17         for (int h=2; h<=len; h<=1) {
18             for (int i=0; i<h/2; i++) omega[i] = polar(1.0, 2*pi*reverse/h*i);
19             for (int i=0; i<len; i+=h) {
20                 for (int j=i; j<i+h/2; j++) {
21                     comp w = omega[j-i];
22                     comp u = x[j];
23                     comp v = (w * x[j+h/2]);
24                     x[j] = u + v;
25                     x[j+h/2] = u - v;
26                 }
27             }
28         }
29         if (reverse == -1) {
30             for (int i=0; i<len; i++)
31                 x[i] /= len;
32         }
33     }
34     void work(int n, int *a, int *b) {
35         int len = 1;
36         while (len <= n*2) len *= 2;
37         for (int i=0; i<len; i++) A[i] = B[i] = 0;
38         for (int i=0; i<n; i++) A[i] = a[i], B[i] = b[i];
39         fft(A, len, 1); fft(B, len, 1);
40         for (int i=0; i<len; i++) A[i] = A[i] * B[i];
41         fft(A, len, -1);
42         for (int i=0; i<len; i++) {
43             LL r = round(A[i].real());
44             a[i] = r % mod;
45         }

```

```
46     }
47 };
```

6.12 求原根

接口: `LL p_root(LL p);`

输入: 一个素数 p

输出: p 的原根

```
1  #include <bits/stdc++.h>
2  #define LL long long
3
4  using namespace std;
5
6  vector <LL> a;
7
8  LL pow_mod(LL base, LL times, LL mod) {
9      LL ret = 1;
10     while (times) {
11         if (times&1) ret = ret * base % mod;
12         base = base * base % mod;
13         times>>=1;
14     }
15     return ret;
16 }
17
18 bool g_test(LL g, LL p) {
19     for (LL i = 0; i < a.size(); ++i)
20         if (pow_mod(g, (p-1)/a[i], p) == 1) return 0;
21     return 1;
22 }
23
24 LL p_root(LL p) {
25     LL tmp = p - 1;
26     for (LL i = 2; i <= tmp / i; ++i)
27         if (tmp % i == 0) {
28             a.push_back(i);
29             while (tmp % i == 0)
30                 tmp /= i;
31         }
32     if (tmp != 1) a.push_back(tmp);
33     LL g = 1;
34     while (1) {
35         if (g_test(g, p)) return g;
36         ++g;
37     }
38 }
39
40 int main() {
41     LL p;
42     cin >> p;
43     cout << p_root(p) << endl;
44 }
```

6.13 NTT

998244353 原根为 3 , 1004535809 原根为 3 , 786433 原根为 10 , 880803841 原根为 26 。

```
1 #define mod 998244353
2 #define g 3
3 LL wi[N], wiv[N];
4 LL power(LL base, LL times) {
5     LL ans = 1;
6     while (times) {
7         if (times&1) (ans *= base) %= mod;
8         (base *= base) %= mod;
9         times >>= 1;
10    }
11    return ans;
12 }
13 void transform(LL *x, int len) {
14     for (int i=1, j=len/2; i<len-1; i++) {
15         if (i<j) swap(x[i], x[j]);
16         int k = len/2;
17         while (j>=k) {
18             j-=k;
19             k/=2;
20         }
21         if (j<k) j+=k;
22     }
23 }
24 void NTT(LL *x, int len, int reverse) {
25     transform(x, len);
26     for (int h=2; h<=len; h<=1) {
27         for (int i=0; i<len; i+=h) {
28             LL w = 1, wn;
29             if (reverse==1) wn = wi[h]; else wn = wiv[h];
30             for (int j=i; j<i+h/2; j++) {
31                 LL u = x[j];
32                 LL v = (w * x[j+h/2]) % mod;
33                 x[j] = (u + v) % mod;
34                 x[j+h/2] = (u - v + mod) % mod;
35                 (w *= wn) %= mod;
36             }
37         }
38     }
39     if (reverse == -1) {
40         LL t = power(len, mod-2);
41         for (int i=0; i<len; i++)
42             (x[i] *= t) %= mod;
43     }
44 }
45 LL A[N], B[N];
46 int main() {
47     for (int i=1; i<N; i*=2) {
48         wi[i] = power(g, (mod-1)/i);
49         wiv[i] = power(wi[i], mod-2);
50     }
51     memset(A, 0, sizeof(A));
52     memset(B, 0, sizeof(B));
53     NTT(A, len, 1); NTT(B, len, 1);
54     for (int i=0; i<len; i++) (A[i] *= B[i]) %= mod;
```

```
55     NTT(A, len, -1);  
56 }
```

6.14 组合数 lcm

$$(n+1)lcm(C(n,0), C(n,1), \dots, C(n,k)) = lcm(n+1, n, n-1, \dots, n-k+1)$$

6.15 区间 lcm 的维护

对于一个数，将其分解质因数，若有因子 p^k ，那么拆分成 k 个数 p, p^2, \dots, p^k ，权值都为 p ，那么查询区间 $[l, r]$ 内所有数的 lcm 的答案 = 所有在该区间中出现过的数的权值之积，可持久化线段树维护即可。

7 几何

7.1 凸包

```
1 typedef complex<int> point;
2 #define X real()
3 #define Y imag()
4 int n;
5 long long cross(point a, point b) {
6     return 1ll * a.X * b.Y - 1ll * a.Y * b.X;
7 }
8 bool cmp(point a, point b) {
9     return make_pair(a.X, a.Y) < make_pair(b.X, b.Y);
10 }
11 int convexHull(point p[], int n, point ch[]) {
12     sort(p, p + n, cmp);
13     int m = 0;
14     for(int i = 0; i < n; ++i) {
15         while(m > 1 && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
16         ch[m++] = p[i];
17     }
18     int k = m;
19     for(int i = n - 2; i >= 0; --i) {
20         while(m > k && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
21         ch[m++] = p[i];
22     }
23     if(n > 1) m--;
24     return m;
25 }
```

8 黑科技和杂项

8.1 高精度计算

```
1 #include<algorithm>
2 using namespace std;
3 const int N_huge=850, base=1000000000;
4 char s[N_huge*10];
5 struct huge{
6     typedef long long value;
7     value a[N_huge]; int len;
8     void clear(){len=1; a[len]=0;}
9     huge(){clear();}
10    huge(value x){*this=x;}
11    huge operator =(huge b){
12        len=b.len; for (int i=1; i<=len; ++i) a[i]=b.a[i]; return *this;
13    }
14    huge operator =(value x){
15        len=0;
16        while (x) a[++len]=x%base, x/=base;
17        if (!len) a[++len]=0;
18        return *this;
19    }
20    huge operator +(huge b){
```



```

21     int L=len>b.len?len:b.len;huge tmp;
22     for (int i=1;i<=L+1;++i)tmp.a[i]=0;
23     for (int i=1;i<=L;++i){
24         if (i>len)tmp.a[i]+=b.a[i];
25         else if (i>b.len)tmp.a[i]+=a[i];
26         else {
27             tmp.a[i]+=a[i]+b.a[i];
28             if (tmp.a[i]>=base){
29                 tmp.a[i]-=base;++tmp.a[i+1];
30             }
31         }
32     }
33     if (tmp.a[L+1])tmp.len=L+1;
34     else tmp.len=L;
35     return tmp;
36 }
37 huge operator -(huge b){
38     int L=len>b.len?len:b.len;huge tmp;
39     for (int i=1;i<=L+1;++i)tmp.a[i]=0;
40     for (int i=1;i<=L;++i){
41         if (i>b.len)b.a[i]=0;
42         tmp.a[i]+=a[i]-b.a[i];
43         if (tmp.a[i]<0){
44             tmp.a[i]+=base;--tmp.a[i+1];
45         }
46     }
47     while (L>1&&!tmp.a[L])--L;
48     tmp.len=L;
49     return tmp;
50 }
51 huge operator *(huge b){
52     int L=len+b.len;huge tmp;
53     for (int i=1;i<=L;++i)tmp.a[i]=0;
54     for (int i=1;i<=len;++i)
55         for (int j=1;j<=b.len;++j){
56             tmp.a[i+j-1]+=a[i]*b.a[j];
57             if (tmp.a[i+j-1]>=base){
58                 tmp.a[i+j]+=tmp.a[i+j-1]/base;
59                 tmp.a[i+j-1]%=base;
60             }
61         }
62     tmp.len=len+b.len;
63     while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
64     return tmp;
65 }
66 pair<huge,huge> divide(huge a,huge b){
67     int L=a.len;huge c,d;
68     for (int i=L;i;--i){
69         c.a[i]=0;d=d*base;d.a[1]=a.a[i];
70         int l=0,r=base-1,mid;
71         while (l<r){
72             mid=(l+r+1)>>1;
73             if (b*mid<=d)l=mid;
74             else r=mid-1;
75         }
76         c.a[i]=l;d-=b*l;
77     }

```

```

78     while (L>1&&!c.a[L])--L;c.len=L;
79     return make_pair(c,d);
80 }
81 huge operator / (value x){
82     value d=0;huge tmp;
83     for (int i=len;i--i){
84         d=d*base+a[i];
85         tmp.a[i]=d/x;d%=x;
86     }
87     tmp.len=len;
88     while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
89     return tmp;
90 }
91 value operator % (value x){
92     value d=0;
93     for (int i=len;i--i)d=(d*base+a[i])%x;
94     return d;
95 }
96 huge operator / (huge b){return divide(*this,b).first;}
97 huge operator % (huge b){return divide(*this,b).second;}
98 huge &operator += (huge b){*this=*this+b;return *this;}
99 huge &operator -= (huge b){*this=*this-b;return *this;}
100 huge &operator *= (huge b){*this=*this*b;return *this;}
101 huge &operator ++() {huge T;T=1;*this=*this+T;return *this;}
102 huge &operator --() {huge T;T=1;*this=*this-T;return *this;}
103 huge operator ++(int) {huge T,tmp=*this;T=1;*this=*this+T;return tmp;}
104 huge operator --(int) {huge T,tmp=*this;T=1;*this=*this-T;return tmp;}
105 huge operator +(value x){huge T;T=x;return *this+T;}
106 huge operator -(value x){huge T;T=x;return *this-T;}
107 huge operator *(value x){huge T;T=x;return *this*T;}
108 huge operator *=(value x){*this=*this*x;return *this;}
109 huge operator +=(value x){*this=*this+x;return *this;}
110 huge operator -=(value x){*this=*this-x;return *this;}
111 huge operator /=(value x){*this=*this/x;return *this;}
112 huge operator %=(value x){*this=*this%x;return *this;}
113 bool operator ==(value x){huge T;T=x;return *this==T;}
114 bool operator !=(value x){huge T;T=x;return *this!=T;}
115 bool operator <=(value x){huge T;T=x;return *this<=T;}
116 bool operator >=(value x){huge T;T=x;return *this>=T;}
117 bool operator <(value x){huge T;T=x;return *this<T;}
118 bool operator >(value x){huge T;T=x;return *this>T;}
119 bool operator <(huge b){
120     if (len<b.len)return 1;
121     if (len>b.len)return 0;
122     for (int i=len;i--i){
123         if (a[i]<b.a[i])return 1;
124         if (a[i]>b.a[i])return 0;
125     }
126     return 0;
127 }
128 bool operator ==(huge b){
129     if (len!=b.len)return 0;
130     for (int i=len;i--i)
131         if (a[i]!=b.a[i])return 0;
132     return 1;
133 }
134 bool operator !=(huge b){return !(*this==b);}

```

```

135     bool operator > (huge b) {return !(*this<b || *this==b);}
136     bool operator <= (huge b) {return (*this<b) || (*this==b);}
137     bool operator >= (huge b) {return (*this>b) || (*this==b);}
138     void str(char s[]) {
139         int l=strlen(s);value x=0,y=1;len=0;
140         for (int i=l-1;i>=0;--i) {
141             x=x+(s[i' ']-0)*y;y*=10;
142             if (y==base)a[++len]=x,x=0,y=1;
143         }
144         if (!len||x)a[++len]=x;
145     }
146     void read() {
147         scanf("%s",s);this->str(s);
148     }
149     void print(){
150         printf("%d", (int)a[len]);
151         for (int i=len-1;i;--i){
152             for (int j=base/10;j>=10;j/=10){
153                 if (a[i]<j)printf("0");
154                 else break;
155             }
156             printf("%d", (int)a[i]);
157         }
158         printf("\n");
159     }
160 }f[1005];
161 int main() {
162     f[1]=f[2]=1;
163     for(int i=3;i<=1000;i++) f[i]=f[i-1]+f[i-2];
164 }

```