ICPC Templates For Africamonkey

Africamonkey

February 1, 2017

Contents

1 莫队算法

1.1 普通莫队

```
struct Q { int l, r, sqrtl, id; } q[N];
int n, m, v[N], ans[N], nowans;
bool cmp(const Q &a, const Q &b) {
    if (a.sqrtl != b.sqrtl) return a.sqrtl < b.sqrtl;
    return a.r < b.r;
}
void change(int x) { if (!v[x]) checkin(); else checkout(); }
int main() {
    ......
    for (int i=1;i<=m;i++) q[i].sqrtl = q[i].l / sqrt(n), q[i].id = i;
    sort(q+1, q+m+1, cmp);
    int L=1,R=0; nowans=0;
    memset(v, 0, sizeof(v));
    for (int i=1;i<=m;i++) {
        while (L<q[i].l) change(L++);
        while (L>q[i].l) change(--L);
        while (R<q[i].r) change(++R);
        while (R>q[i].r) change(R--);
        ans[q[i].id] = nowans;
    }
    ......
}
```

1.2 树上莫队

```
struct Query { int l, r, id, l_group; } query[N];
struct EDGE { int adj, next; } edge[N*2];
int n, m, top, gh[N], c[N], reorder[N], deep[N], father[N], size[N], son[N], Top[N];
void addedge(int x, int y) {
    edge[++top].adj = y;
    edge[top].next = gh[x];
    gh[x] = top;
}
void dfs(int x, int root=0) {
    reorder[x] = ++top; father[x] = root; deep[x] = deep[root] + 1;
    son[x] = 0; size[x] = 1; int dd = 0;
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].adj != root) {
            dfs(edge[p].adj, x);
            if (size[edge[p].adj] > dd) {
                son[x] = edge[p].adj;
                dd = size[edge[p].adj];
            }
            size[x] += size[edge[p].adj];
        }
}
void split(int x, int tp) {
    Top[x] = tp;
    if (son[x]) split(son[x], tp);
    for (int p=gh[x]; p; p=edge[p].next)
        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

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

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。

```
void find_next(char *pattern, int *next) {
    int n = strlen(pattern);
    for (int i=1;i<n;i++) {
        int j = i;
        while (j > 0) {
            j = next[j];
            if (pattern[j] == pattern[i]) {
                next[i+1] = j+1;
                break;
            }
        }
    }
}
int find_substring(char *pattern, char *text, int *next, int *ret) {
    find_next(pattern, next);
    int n = strlen(pattern);
    int m = strlen(text);
    int k = 0;
    for (int i=0,j=0;i<m;i++) {
        if (j<n && text[i]==pattern[j]) {
            j++;
        } else {
            while (j>0) {
                j = next[j];
                if (text[i] == pattern[j]) {
                    j++;
                    break;
                }
            }
        }
        if (j == n)
            ret[k++] = i-n+1;
    }
    return k;
```

```
}
```

2.3 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。

```
void get_next(char *a, int *next) {
    int i, j, k;
    int n = strlen(a);
    for (j = 0; j+1<n && a[j]==a[j+1];j++);
    next[1] = j;
    k = 1;
    for (i=2;i<n;i++) {
        int len = k+next[k], l = next[i-k];
        if (l < len-i) {
            next[i] = l;
        } else {
            for (j = max(0, len-i);i+j<n && a[j]==a[i+j];j++);
            next[i] = j;
            k = i;
        }
    }
}
void ExtendedKMP(char *a, char *b, int *next, int *ret) {
    get_next(a, next);
    int n = strlen(a), m = strlen(b);
    int i, j, k;
    for (j=0;j<n && j<m && a[j]==b[j];j++);
    ret[0] = j;
    k = 0;
    for (i=1;i<m;i++) {
        int len = k+ret[k], l = next[i-k];
        if (l < len-i) {
            ret[i] = l;
        } else {
            for (j = max(0, len-i);j<n && i+j<m && a[j]==b[i+j];j++);
            ret[i] = j;
            k = i;
        }
    }
}
```

2.4 Manacher

p[i] 表示以 i 为对称轴的最长回文串长度

```
char st[N*2], s[N];
int len, p[N*2];

while (scanf("%s", s) != EOF) {
    len = strlen(s);
    st[0] = '$', st[1] = '#';
    for (int i=1;i<=len;i++)
        st[i*2] = s[i-1], st[i*2+1] = '#';
    len = len * 2 + 2;
    int mx = 0, id = 0, ans = 0;
    for (int i=1;i<=len;i++) {
        p[i] = (mx > i) ? min(p[id*2-i]+1, mx-i) : 1;
        for (; st[i+p[i]] == st[i-p[i]]; ++p[i]) ;
        if (p[i]+i > mx) mx = p[i]+i, id = i;
        p[i] --;
        if (p[i] > ans) ans = p[i];
    }
    printf("%d\n", ans);
}
```

2.5 最小表示法

```
string smallestRepresation(string s) {
    int i, j, k, l;
    int n = s.length();
    s += s;
    for (i=0,j=1;j<n;) {
        for (k=0;k<n && s[i+k]==s[j+k];k++);
        if (k>=n) break;
        if (s[i+k]<s[j+k]) j+=k+1;
        else {
            l=i+k;
            i=j;
            j=max(l,j)+1;
        }
    }
    return s.substr(i, n);
}
```

2.6 AC自动机

```
struct Node {
    int next[**Size of Alphabet**];
    int terminal, fail;
} node[**Number of Nodes**];
int top;
void add(char *st) {
    int len = strlen(st), x = 1;
    for (int i=0;i<len;i++) {
        int ind = trans(st[i]);
        if (!node[x].next[ind])
            node[x].next[ind] = ++top;
        x = node[x].next[ind];
    }
    node[x].terminal = 1;
}
int q[**Number of Nodes**], head, tail;
void build() {
    head = 0, tail = 1; q[1] = 1;
    while (head != tail) {
        int x = q[++head];
        /*(when necessary) node[x].terminal |= node[node[x].fail].terminal; */
        for (int i=0;i<n;i++)
            if (node[x].next[i]) {
                if (x == 1) node[node[x].next[i]].fail = 1;
                else {
                    int y = node[x].fail;
                    while (y) {
                        if (node[y].next[i]) {
                            node[node[x].next[i]].fail = node[y].next[i];
                            break;
                        }
                        y = node[y].fail;
                    }
                    if (!node[node[x].next[i]].fail) node[node[x].next[i]].fail = 1;
                }
                q[++tail] = node[x].next[i];
            }
    }
}
```

2.7 后缀数组

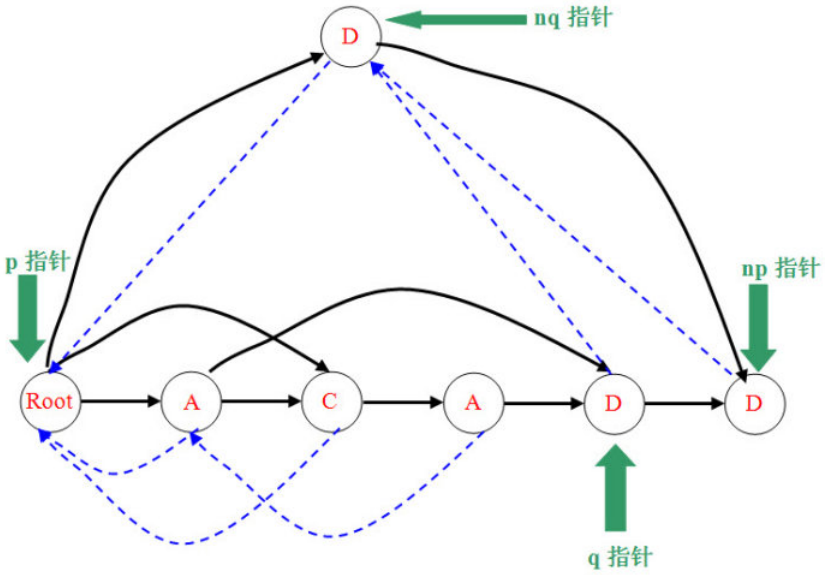
参数 m 表示字符集的大小，即 $0 \leq r_i < m$

```
#define rank rank2
int n, r[N], wa[N], wb[N], ws[N], sa[N], rank[N], height[N];
int cmp(int *r, int a, int b, int l, int n)
{
    if (r[a]==r[b])
    {
        if (a+l<n && b+l<n && r[a+l]==r[b+l])
            return 1;
    }
    return 0;
}
void suffix_array(int m)
{
    int i, j, p, *x=wa, *y=wb, *t;
    for (i=0;i<m;i++) ws[i]=0;
    for (i=0;i<n;i++) ws[x[i]=r[i]]++;
    for (i=1;i<m;i++) ws[i]+=ws[i-1];
    for (i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
    for (j=1,p=1;p<n;m=p,j<<=1)
    {
        for (p=0,i=n-j;i<n;i++) y[p++]=i;
        for (i=0;i<n;i++) if (sa[i]>=j) y[p++]=sa[i]-j;
        for (i=0;i<m;i++) ws[i]=0;
        for (i=0;i<n;i++) ws[x[y[i]]]++;
        for (i=1;i<m;i++) ws[i]+=ws[i-1];
        for (i=n-1;i>=0;i--) sa[--ws[x[y[i]]]]=y[i];
        for (t=x,x=y,y=t,x[sa[0]]=0,i=1,p=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j,n)?p-1:p++;
    }
    for (i=0;i<n;i++) rank[sa[i]]=i;
}
void calc_height()
{
    int j=0;
    for (int i=0;i<n;i++)
        if (rank[i])
        {
            while (r[i+j]==r[sa[rank[i]-1]+j]) j++;
            height[rank[i]]=j;
            if (j) j--;
        }
}
```

2.8 后缀自动机

下面的代码是求两个串的LCS（最长公共子串）。

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#define N 500001
using namespace std;
char st[N];
int pre[N<<1], son[26][N<<1], step[N<<1], last, total;
int apply(int x) { step[++total]=x; return total; }
void Extend(char x) {
    int p = last, np = apply(step[last]+1);
    for (; p && !son[x][p]; p=pre[p]) son[x][p] = np;
    if (!p) pre[np] = 1;
    else {
        int q = son[x][p];
        if (step[p]+1 == step[q]) pre[np] = q;
        else {
            int nq = apply(step[p]+1);
            for (int i=0;i<26;i++) son[i][nq] = son[i][q];
            pre[nq] = pre[q];
            pre[q] = pre[np] = nq;
            for (; p && son[x][p]==q; p=pre[p]) son[x][p] = nq;
        }
    }
    last = np;
}
void init() {
    last = total = 0;
    last = apply(0);
    scanf("%s",st);
    for (int i=0; st[i]; i++)
        Extend(st[i]-'a');
    scanf("%s",st);
}
int main() {
    init();
    int p = 1, now = 0, ans = 0;
    for (int i=0; st[i]; i++) {
        int index = st[i]-'a';
        for (; p && !son[index][p]; p = pre[p], now = step[p]) ;
        if (!p) p = 1;
        if (son[index][p]) {
            p = son[index][p];
            now++;
            if (now > ans) ans = now;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```



3 数据结构

3.1 ST表

```
int Log[N],f[17][N];
int ask(int x,int y){
    int k=log[y-x+1];
    return max(f[k][x],f[k][y-(1<<k)+1]);
}
int main(){
    for(i=2;i<=n;i++)Log[i]=Log[i>>1]+1;
    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)]);
}
```

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）

```
int queryl(int p, int left, int right, int y, int w) {
    if (right <= y) {
        if (! __condition__ ) return -1;
        else if (left == right) return left;
    }
    int mid = (left + right) / 2;
    if (y <= mid) return queryl(p<<1|0, left, mid, y, w);
    int ret = queryl(p<<1|1, mid+1, right, y, w);
    if (ret != -1) return ret;
    return queryl(p<<1|0, left, mid, y, w);
}
```

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）

```
int queryr(int p, int left, int right, int y, int w) {
    if (left >= x) {
        if (! __condition__ ) return -1;
        else if (left == right) return left;
    }
    int mid = (left + right) / 2;
    if (x > mid) return queryr(p<<1|1, mid+1, right, y, w);
    int ret = queryr(p<<1|0, left, mid, y, w);
    if (ret != -1) return ret;
    return queryr(p<<1|1, mid+1, right, y, w);
}
```

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

```
struct SPLAY {
    struct NODE {
        int w, min;
        int son[2], size, father, rev, lazy;
    } node[N];
    int top, rt;
    void pushdown(int x) {
        if (!x) return;
        if (node[x].rev) {
            node[node[x].son[0]].rev ^= 1;
            node[node[x].son[1]].rev ^= 1;
            swap(node[x].son[0], node[x].son[1]);
            node[x].rev = 0;
        }
        if (node[x].lazy) {
            node[node[x].son[0]].lazy += node[x].lazy;
            node[node[x].son[1]].lazy += node[x].lazy;
            node[x].w += node[x].lazy;
            node[x].min += node[x].lazy;
            node[x].lazy = 0;
        }
    }
    void pushup(int x) {
        if (!x) return;
        pushdown(node[x].son[0]);
        pushdown(node[x].son[1]);
        node[x].size = node[node[x].son[0]].size + node[node[x].son[1]].size + 1;
        node[x].min = node[x].w;
        if (node[x].son[0]) node[x].min = min(node[x].min, node[node[x].son[0]].min);
        if (node[x].son[1]) node[x].min = min(node[x].min, node[node[x].son[1]].min);
    }
    void sc(int x, int y, int w) {
        node[x].son[w] = y;
        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

```
#define mod 1000000007
struct Treap {
    struct Node {
        char key;
        bool reverse;
        int lc, rc, size;
    } node[N];
    int n, root, rd;
    int Rand() { rd = (rd * 20372052LL + 25022087LL) % mod; return rd; }
    void init() { n = root = 0; }
    inline int copy(int x) { node[++n] = node[x]; return n; }
    inline void pushdown(int x) {
        if (!node[x].reverse) return;
        if (node[x].lc) node[x].lc = copy(node[x].lc);
        if (node[x].rc) node[x].rc = copy(node[x].rc);
        swap(node[x].lc, node[x].rc);
        node[node[x].lc].reverse ^= 1;
        node[node[x].rc].reverse ^= 1;
        node[x].reverse = 0;
    }
    inline void pushup(int x) { node[x].size = node[node[x].lc].size + node[node[x].rc].size
         + 1; }
    int merge(int u, int v) {
        if (!u || !v) return u+v;
        pushdown(u); pushdown(v);
        int t = Rand() % (node[u].size + node[v].size), r;
        if (t < node[u].size) {
            r = copy(u);
            node[r].rc = merge(node[u].rc, v);
        } else {
            r = copy(v);
            node[r].lc = merge(u, node[v].lc);
        }
        pushup(r);
        return r;
    }
    int split(int u, int x, int y) {
        if (x > y) return 0;
        pushdown(u);
        if (x == 1 && y == node[u].size) return u;
        if (y <= node[node[u].lc].size) return split(node[u].lc, x, y);
        int t = node[node[u].lc].size + 1;
        if (x > t) return split(node[u].rc, x-t, y-t);
        int num = copy(u);
        node[num].lc = split(node[u].lc, x, t-1);
        node[num].rc = split(node[u].rc, 1, y-t);
        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, l, r), t3 = split(root, x+1, node[root].
                size);
60          root = merge(merge(t1, t2), t3);
61      }
62      void reverse(int x, int y) {
63          int t1 = split(root, 1, x-1), t2 = split(root, x, y), t3 = split(root, y+1, node[root].
                size);
64          node[t2].reverse ^= 1;
65          root = merge(merge(t1, t2), t3);
66      }
67      char query(int k) {
68          int x = root;
69          while (1) {
70              pushdown(x);
71              if (k <= node[node[x].lc].size) x = node[x].lc;
72              else
73              if (k == node[node[x].lc].size + 1) return node[x].key;
74              else
75              k -= node[node[x].lc].size + 1, x = node[x].rc;
76          }
77      }
78  } 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)

```
struct DynamicTREE{
    struct NODE{
        int father, son[2], top, size, reverse;
    } splay[N];
    void init(int i, int fat) {
        splay[i].father = splay[i].son[0] = splay[i].son[1] = 0;
        splay[i].top = fat; splay[i].size = 1; splay[i].reverse = 0;
    }
    void pushdown(int x) {
        if (!x) return;
        int s0 = splay[x].son[0], s1 = splay[x].son[1];
        if (splay[x].reverse) {
            splay[s0].reverse ^= 1;
            splay[s1].reverse ^= 1;
            swap(splay[x].son[0], splay[x].son[1]);
            splay[x].reverse = 0;
        }
        s0 = splay[x].son[0], s1 = splay[x].son[1];
        splay[s0].top = splay[s1].top = splay[x].top;
    }
    void pushup(int x) {
        if (!x) return;
        pushdown(splay[x].son[0]);
        pushdown(splay[x].son[1]);
        splay[x].size = splay[splay[x].son[0]].size + splay[splay[x].son[1]].size + 1;
    }
    void sc(int x, int y, int w, bool Auto=true) {
        splay[x].son[w] = y;
        splay[y].father = x;
        if (Auto) {
            pushup(y);
            pushup(x);
        }
    }
    int top, tush[N];
    void flowdown(int x) {
        for (top=1; x; top++, x = splay[x].father) tush[top] = x;
        for (; top; top--) pushdown(tush[top]);
    }
    void rotate(int x) {
        if (!x) return;
        int y = splay[x].father;
        int w = splay[y].son[1] == x;
        pushdown(y);
        pushdown(x);
        sc(splay[y].father, x, splay[splay[y].father].son[1]==y, false);
        sc(y, splay[x].son[w^1], w, false);
        sc(x, y, w^1, false);
        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 。

```
namespace UndirectedGraph{
    int n,m,i,x,y,d[N],g[N],v[M<<1],w[M<<1],vis[M<<1],nxt[M<<1],ed;
    int ans[M],cnt;
    void add(int x,int y,int z){
        d[x]++;
        v[++ed]=y;w[ed]=z;nxt[ed]=g[x];g[x]=ed;
    }
    void dfs(int x){
        for(int&i=g[x];i;){
            if(vis[i]){i=nxt[i];continue;}
            vis[i]=vis[i^1]=1;
            int j=w[i];
            dfs(v[i]);
            ans[++cnt]=j;
        }
    }
    void solve(){
        scanf("%d%d",&n,&m);
        for(i=ed=1;i<=m;i++)scanf("%d%d",&x,&y),add(x,y,i),add(y,x,-i);
        for(i=1;i<=n;i++)if(d[i]&1){puts("NO");return;}
        for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
        for(i=1;i<=n;i++)if(g[i]){puts("NO");return;}
        puts("YES");
        for(i=m;i;i--)printf("%d␣",ans[i]);
    }
}
namespace DirectedGraph{
    int n,m,i,x,y,d[N],g[N],v[M],vis[M],nxt[M],ed;
    int ans[M],cnt;
    void add(int x,int y){
        d[x]++;d[y]--;
        v[++ed]=y;nxt[ed]=g[x];g[x]=ed;
    }
    void dfs(int x){
        for(int&i=g[x];i;){
            if(vis[i]){i=nxt[i];continue;}
            vis[i]=1;
            int j=i;
            dfs(v[i]);
            ans[++cnt]=j;
        }
    }
```

```
    void solve(){
        scanf("%d%d",&n,&m);
        for(i=1;i<=m;i++)scanf("%d%d",&x,&y),add(x,y);
        for(i=1;i<=n;i++)if(d[i]){puts("NO");return;}
        for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
        for(i=1;i<=n;i++)if(g[i]){puts("NO");return;}
        puts("YES");
        for(i=m;i;i--)printf("%d ",ans[i]);
    }
}
```

5.2 最短路径

5.2.1 Dijkstra

```
#include <queue>
using namespace std;
struct EDGE { int adj, w, next; } edge[M*2];
struct dat { int id, dist; dat(int id=0, int dist=0) : id(id), dist(dist) {} };
struct cmp { bool operator () (const dat &a, const dat &b) { return a.dist > b.dist; } };
priority_queue < dat, vector<dat>, cmp > q;
int n, top, gh[N], v[N], dist[N];
void addedge(int x, int y, int w) {
    edge[++top].adj = y;
    edge[top].w = w;
    edge[top].next = gh[x];
    gh[x] = top;
}
int dijkstra(int s, int t) {
    memset(dist, 63, sizeof(dist));
    memset(v, 0, sizeof(v));
    dist[s] = 0;
    q.push(dat(s, 0));
    while (!q.empty()) {
        dat x = q.top(); q.pop();
        if (v[x.id]) continue; v[x.id] = 1;
        for (int p=gh[x.id]; p; p=edge[p].next) {
            if (x.dist + edge[p].w < dist[edge[p].adj]) {
                dist[edge[p].adj] = x.dist + edge[p].w;
                q.push(dat(edge[p].adj, dist[edge[p].adj]));
            }
        }
    }
    return dist[t];
}
```

5.2.2 SPFA

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

```
    gh[x] = top;
}
int spfa(int S, int T) {
    memset(v, 0, sizeof(v));
    memset(cnt, 0, sizeof(cnt));
    memset(dist, 63, sizeof(dist));
    head = 0, tail = 1;
    dist[S] = 0; q[1] = S;
    while (head != tail) {
        (head += 1) %= N;
        int x = q[head]; v[x] = 0;
        ++cnt[x]; if (cnt[x] > n) return -1;
        for (int p=gh[x]; p; p=edge[p].next)
            if (dist[x] + edge[p].w < dist[edge[p].adj]) {
                dist[edge[p].adj] = dist[x] + edge[p].w;
                if (!v[edge[p].adj]) {
                    v[edge[p].adj] = 1;
                    (tail += 1) %= N;
                    q[tail] = edge[p].adj;
                }
            }
    }
    return dist[T];
}
```

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

```
#include <queue>

#define N 200020
#define M 400020
#define LOGM 20
#define LL long long
#define inf (1LL<<61)

namespace pheap {
    struct Node {
        int next, son[2];
        LL val;
    } node[M*LOGM];
    int LOG[M];
    int root[M], size[M*LOGM], top;
    int add() {
        ++top; assert(top < M*LOGM);
        node[top].next = node[top].son[0] = node[top].son[1] = 0;
        node[top].val = inf;
        return top;
    }
    int copy(int x) { int t = add(); node[t] = node[x]; return t; }
    void init() {
```

```
        top = -1; add();
        for (int i=2;i<M;i++) LOG[i] = LOG[i>>1] + 1;
    }
    void upd(int x, int &next, LL &val) {
        if (val < node[x].val) {
            swap(val, node[x].val);
            swap(next, node[x].next);
        }
    }
    void insert(int x, int next, LL val) {
        int sz = size[root[x]] + 1;
        root[x] = copy(root[x]);
        size[root[x]] = sz; x = root[x];
        upd(x, next, val);
        for (int i=LOG[sz]-1;i>=0;i--) {
            int ind = (sz>>i)&1;
            node[x].son[ind] = copy(node[x].son[ind]);
            x = node[x].son[ind];
            upd(x, next, val);
        }
    }
};

namespace kthsp {
    using namespace pheap;
    struct EDGE {
        int adj, w, next;
    } edge[2][M];
    struct W {
        int x, y, w;
    } e[M];
    bool has_init = 0;
    int n, m, top[2], gh[2][N], v[N];
    LL dist[N];
    void init(int n1) {
        has_init = 1;
        n = n1; m = 0;
        memset(top, 0, sizeof(top));
        memset(gh, 0, sizeof(gh));
        for (int i=1;i<=n;i++) dist[i] = inf;
    }
    void addedge(int id, int x, int y, int w) {
        edge[id][++top[id]].adj = y;
        edge[id][top[id]].w = w;
        edge[id][top[id]].next = gh[id][x];
        gh[id][x] = top[id];
    }
    void add(int x, int y, int w) {
        assert(has_init);
        e[++m].x=x; e[m].y=y; e[m].w=w;
    }
    int q[N], best[N], bestw[N];
    int deg[N];
    void spfa(int S) {
        for (int i=1;i<=n;i++) deg[i] = 0;
        for (int i=1;i<=m;i++) deg[e[i].x] ++;
        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;
```

```
    }
};
```

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 $dfn[u] < low[v]$

```
struct EDGE { int adj, next; } edge[M];
int n, m, top, gh[N];
int dfn[N], low[N], cnt, ind, stop, instack[N], stack[N], belong[N];
void addedge(int x, int y) {
    edge[++top].adj = y;
    edge[top].next = gh[x];
    gh[x] = top;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++ind;
    instack[x] = 1; stack[++stop] = x;
    for (int p=gh[x]; p; p=edge[p].next)
        if (!dfn[edge[p].adj]) {
            tarjan(edge[p].adj);
            low[x] = min(low[x], low[edge[p].adj]);
        } else if (instack[edge[p].adj]) {
            low[x] = min(low[x], dfn[edge[p].adj]);
        }
    if (dfn[x] == low[x]) {
        ++cnt; int tmp=0;
        while (tmp!=x) {
            tmp = stack[stop--];
            belong[tmp] = cnt;
            instack[tmp] = 0;
        }
    }
}
```

5.5 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

```
namespace dominator {
    vector <int> g[N], rg[N], bucket[N], tree[N];
    int n, ind, idom[N], sdom[N], dfn[N], dsu[N], father[N], label[N], rev[N];
    void dfs(int x) {
```

```
        ++ind;
        dfn[x] = ind; rev[ind] = x;
        label[ind] = dsu[ind] = sdom[ind] = ind;
        for (auto p : g[x]) {
            if (!dfn[p]) dfs(p), father[dfn[p]] = dfn[x];
            rg[dfn[p]].push_back(dfn[x]);
        }
    }
    void init(int n1) {
        n = n1; ind = 0;
        for (int i = 1; i <= n; ++i) {
            g[i].clear();
            rg[i].clear();
            bucket[i].clear();
            tree[i].clear();
            dfn[i] = 0;
        }
    }
    void addedge(int u, int v) {
        g[u].push_back(v);
    }
    int find(int x, int step=0) {
        if (dsu[x] == x) return step ? -1 : x;
        int y = find(dsu[x], 1);
        if (y < 0) return x;
        if (sdom[label[dsu[x]]] < sdom[label[x]])
            label[x] = label[dsu[x]];
        dsu[x] = y;
        return step ? dsu[x] : label[x];
    }
    void work(int root) {
        dfs(root); n = ind;
        for (int i = n; i; --i) {
            for (auto p : rg[i])
                sdom[i] = min(sdom[i], sdom[find(p)]);
            if (i > 1) bucket[sdom[i]].push_back(i);
            for (auto p : bucket[i]) {
                int u = find(p);
                if (sdom[p] == sdom[u]) idom[p] = sdom[p];
                else idom[p] = u;
            }
            if (i > 1) dsu[i] = father[i];
        }
        for (int i = 2; i <= n; ++i) {
            if (idom[i] != sdom[i])
                idom[i] = idom[idom[i]];
            tree[rev[i]].push_back(rev[idom[i]]);
            tree[rev[idom[i]]].push_back(rev[i]);
        }
    }
};
```

5.6 网络流

5.6.1 最大流

```
struct EDGE { int adj, w, next; } edge[M];
int n, top, gh[N], nrl[N];
void addedge(int x, int y, int w) {
    edge[++top].adj = y;
    edge[top].w = w;
    edge[top].next = gh[x];
    gh[x] = top;
    edge[++top].adj = x;
    edge[top].w = 0;
    edge[top].next = gh[y];
    gh[y] = top;
}
int dist[N], q[N];
int bfs() {
    memset(dist, 0, sizeof(dist));
    q[1] = S; int head = 0, tail = 1; dist[S] = 1;
    while (head != tail) {
        int x = q[++head];
        for (int p=gh[x]; p; p=edge[p].next)
            if (edge[p].w && !dist[edge[p].adj]) {
                dist[edge[p].adj] = dist[x] + 1;
                q[++tail] = edge[p].adj;
            }
    }
    return dist[T];
}
int dinic(int x, int delta) {
    if (x==T) return delta;
    for (int& p=nrl[x]; p && delta; p=edge[p].next)
        if (edge[p].w && dist[x]+1 == dist[edge[p].adj]) {
            int dd = dinic(edge[p].adj, min(delta, edge[p].w));
            if (!dd) continue;
            edge[p].w -= dd;
            edge[p^1].w += dd;
            return dd;
        }
    return 0;
}
int main() {
    int ans = 0;
    while (bfs()) {
        memcpy(nrl, gh, sizeof(gh));
        int t; while (t = dinic(S, inf)) ans += t;
    }
}
```

5.6.2 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。

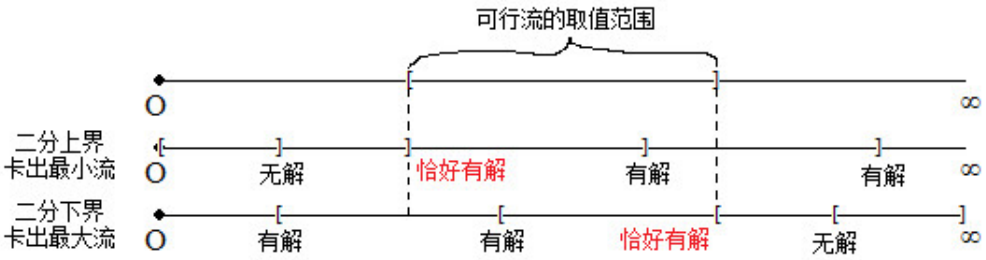
新建超级源汇 SS,TT ，对于 $in[i]>0$ 的点， SS 向 i 连容量为 $in[i]$ 的边。对于 $in[i]<0$ 的点， i 向 TT 连容量为 $-in[i]$ 的边。

求出以 SS,TT 为源汇的最大流，如果等于 $\Sigma in[i](in[i]>0)$ ，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。

5.6.3 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再作特殊处理。
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。



5.6.4 费用流

```
#define inf 0x3f3f3f3f
struct NetWorkFlow {
    struct EDGE {
        int adj, w, cost, next;
    } edge[M*2];
    int gh[N], q[N], dist[N], v[N], pre[N], prev[N], top;
    int S, T;
    void addedge(int x, int y, int w, int cost) {
        edge[++top].adj = y;
        edge[top].w = w;
        edge[top].cost = cost;
        edge[top].next = gh[x];
        gh[x] = top;
        edge[++top].adj = x;
        edge[top].w = 0;
        edge[top].cost = -cost;
        edge[top].next = gh[y];
        gh[y] = top;
    }
    void clear() {
        top = 1;
        memset(gh, 0, sizeof(gh));
    }
    int spfa() {
        memset(dist, 63, sizeof(dist));
        memset(v, 0, sizeof(v));
        int head = 0, tail = 1;
        q[1] = S; v[S] = 1; dist[S] = 0;
```

```
        while (head != tail) {
            (head += 1) %= N;
            int x = q[head];
            v[x] = 0;
            for (int p=gh[x]; p; p=edge[p].next)
                if (edge[p].w && dist[x] + edge[p].cost < dist[edge[p].adj]) {
                    dist[edge[p].adj] = dist[x] + edge[p].cost;
                    pre[edge[p].adj] = x;
                    prev[edge[p].adj] = p;
                    if (!v[edge[p].adj]) {
                        v[edge[p].adj] = 1;
                        (tail += 1) %= N;
                        q[tail] = edge[p].adj;
                    }
                }
        }
        return dist[T] != inf;
    }
    int work() {
        int ans = 0;
        while (spfa()) {
            int mx = inf;
            for (int x=T;x!=S;x=pre[x])
                mx = min(edge[prev[x]].w, mx);
            ans += dist[T] * mx;
            for (int x=T;x!=S;x=pre[x]) {
                edge[prev[x]].w -= mx;
                edge[prev[x]^1].w += mx;
            }
        }
        return ans;
    }
} nwf;
```

5.6.5 zkw费用流

```
#define inf 0x3f3f3f3f
struct NetWorkFlow {
    struct EDGE {
        int adj, w, cost, next;
    } edge[M*2];
    int gh[N], top;
    int S, T;
    void addedge(int x, int y, int w, int cost) {
        edge[++top].adj = y;
        edge[top].w = w;
        edge[top].cost = cost;
        edge[top].next = gh[x];
        gh[x] = top;
        edge[++top].adj = x;
        edge[top].w = 0;
        edge[top].cost = -cost;
        edge[top].next = gh[y];
        gh[y] = top;
    }
    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

```
int exgcd(int a,int b,int&x,int&y){
    if(!b)return x=1,y=0,a;
    int d=exgcd(b,a%b,x,y),t=x;
    return x=y,y=t-a/b*y,d;
}
void cal(ll a,ll b,ll n){//ax=b(mod n)
    ll x,y,d=exgcd(a,n,x,y);
    if(b%d)return;
    x=(x%n+n)%n;
    ans[cnt=1]=x*(b/d)%(n/d);
    for(ll i=1;i<d;i++)ans[++cnt]=(ans[1]+i*n/d)%n;
}
```

6.2 同余方程组

```
int n,flag,k,m,a,r,d,x,y;
int main(){
    scanf("%d",&n);
    flag=k=1,m=0;
    while(n--){
        scanf("%d%d",&a,&r);//ans%a=r
        if(flag){
            d=exgcd(k,a,x,y);
            if((r-m)%d){flag=0;continue;}
            x=(x*(r-m)/d+a/d)%(a/d),y=k/d*a,m=((x*k+m)%y)%y;
            if(m<0)m+=y;
            k=y;
        }
    }
    printf("%d",flag?m:-1);//若flag说明有解解为=1,,ki+m,为任意整数i
}
```

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) 点且不经过对角线 $x=y$ 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 Lucas定理

接口：
初始化： void lucas::init();
计算 $C(n,m)\%mod$ 的值： LL lucas::Lucas(LL n, LL m);

```cpp
#define mod 110119
#define LL long long
namespace lucas {
    LL fac[mod+1], facv[mod+1];
    LL power(LL base, LL times) {
        LL ans = 1;
        while (times) {
            if (times&1) (ans *= base) %= mod;
            (base *= base) %= mod;
            times >>= 1;
        }
        return ans;
    }
    void init() {
        fac[0] = 1; for (int i=1;i<mod;i++) fac[i] = (fac[i-1] * i) % mod;
        facv[mod-1] = power(fac[mod-1], mod-2);
    }
    LL C(unsigned LL n, unsigned LL m) {
        if (n < m) return 0;
        return (fac[n] * facv[m] % mod * facv[n-m] % mod) % mod;
    }
    LL Lucas(unsigned LL n, unsigned LL m)
    {
        if (m == 0) return 1;
        return (C(n%mod, m%mod) * Lucas(n/mod, m/mod)) %mod;
    }
};
```

6.5 高斯消元

6.5.1 行列式

```cpp
int ans = 1;
for (int i=0;i<n;i++) {
    for (int j=i;j<n;j++)
        if (g[j][i]) {
            for (int k=i;k<n;k++)
                swap(g[i][k], g[j][k]);
            if (j != i) ans *= -1;
            break;
        }
    if (g[i][i] == 0) {
        ans = 0;
        break;
    }
    for (int j=i+1;j<n;j++) {
        while (g[j][i]) {
            int t = g[i][i] / g[j][i];
            for (int k=i;k<n;k++)
                g[i][k] = (g[i][k] + mod - ((LL)t * g[j][k] % mod)) % mod;
            for (int k=i;k<n;k++)
                swap(g[i][k], g[j][k]);
            ans *= -1;
        }
    }
```

```
}
for (int i=0;i<n;i++)
    ans = ((LL)ans * g[i][i]) % mod;
ans = (ans % mod + mod) % mod;
printf("%d\n", ans);
```

6.5.2 Matrix-Tree定理

对于一张图，建立矩阵 C ， $C[i][i] = \mathrm{i}$ 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。

6.6 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 。

6.7 曼哈顿距离的变换

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

6.8 线性筛素数

```
mu[1]=phi[1]=1;top=0;
for (int i=2;i<N;i++) {
    if (!v[i]) prime[++top]=i, mu[i] = -1, phi[i] = i-1;
    for (int j=1;i*prime[j]<N && j<=top;j++) {
        v[i*prime[j]] = 1;
        if (i%prime[j]) {
            mu[i*prime[j]] = -mu[i];
            phi[i*prime[j]] = phi[i] * (prime[j]-1);
        } else {
            mu[i*prime[j]] = 0;
            phi[i*prime[j]] = phi[i] * prime[j];
            break;
        }
    }
}
```

6.9 FFT

```
typedef complex<double> comp;
namespace FFT {
    comp A[N], B[N],omega[N];
    void transform(comp *x, int len) {
        for (int i=1,j=len/2;i<len-1;i++) {
            if (i<j) swap(x[i], x[j]);
            int k = len/2;
            while (j>=k) {
                j-=k;
                k/=2;
            }
            if (j<k) j+=k;
```

```
        }
    }
    void fft(comp *x, int len, int reverse) {
        transform(x, len);
        for (int h=2;h<=len;h<<=1) {
            for (int i=0;i<h/2;i++) omega[i] = polar(1.0, 2*pi*reverse/h*i);
            for (int i=0;i<len;i+=h) {
                for (int j=i;j<i+h/2;j++) {
                    comp w = omega[j-i];
                    comp u = x[j];
                    comp v = (w * x[j+h/2]);
                    x[j] = u + v;
                    x[j+h/2] = u - v;
                }
            }
        }
        if (reverse == -1) {
            for (int i=0;i<len;i++)
                x[i] /= len;
        }
    }
    void work(int n, int *a, int *b) {
        int len = 1;
        while (len <= n*2) len *= 2;
        for (int i=0;i<len;i++) A[i] = B[i] = 0;
        for (int i=0;i<n;i++) A[i] = a[i], B[i] = b[i];
        fft(A, len, 1); fft(B, len, 1);
        for (int i=0;i<len;i++) A[i] = A[i] * B[i];
        fft(A, len, -1);
        for (int i=0;i<len;i++) {
            LL r = round(A[i].real());
            a[i] = r % mod;
        }
    }
};
```

6.10 NTT

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

```
#define mod 998244353
#define g 3
LL wi[N], wiv[N];
LL power(LL base, LL times) {
    LL ans = 1;
    while (times) {
        if (times&1) (ans *= base) %= mod;
        (base *= base) %= mod;
        times >>= 1;
    }
    return ans;
}
void transform(LL *x, int len) {
    for (int i=1,j=len/2;i<len-1;i++) {
        if (i<j) swap(x[i], x[j]);
        int k = len/2;
```

```
        while (j>=k) {
            j-=k;
            k/=2;
        }
        if (j<k) j+=k;
    }
}
void NTT(LL *x, int len, int reverse) {
    transform(x, len);
    for (int h=2;h<=len;h<<=1) {
        for (int i=0;i<len;i+=h) {
            LL w = 1, wn;
            if (reverse==1) wn = wi[h]; else wn = wiv[h];
            for (int j=i;j<i+h/2;j++) {
                LL u = x[j];
                LL v = (w * x[j+h/2]) % mod;
                x[j ] = (u + v) % mod;
                x[j+h/2] = (u - v + mod) % mod;
                (w *= wn) %= mod;
            }
        }
    }
    if (reverse == -1) {
        LL t = power(len, mod-2);
        for (int i=0;i<len;i++)
            (x[i] *= t) %= mod;
    }
}
LL A[N], B[N];
int main() {
    for (int i=1;i<N;i*=2) {
        wi[i] = power(g, (mod-1)/i);
        wiv[i] = power(wi[i], mod-2);
    }
    memset(A, 0, sizeof(A));
    memset(B, 0, sizeof(B));
    NTT(A, len, 1); NTT(B, len, 1);
    for (int i=0;i<len;i++) (A[i] *= B[i]) %= mod;
    NTT(A, len, -1);
}
```

6.11 组合数 lcm

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

6.12 区间 lcm 的维护

对于一个数，将其分解质因数，若有因子 p^k ，那么拆分出 k 个数 $p,p^2,...,p^k$ ，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

```
typedef complex<int> point;
#define X real()
#define Y imag()
int n;
long long cross(point a,point b) {
    return 1ll * a.X * b.Y - 1ll * a.Y * b.X;
}
bool cmp(point a,point b) {
    return make_pair(a.X, a.Y) < make_pair(b.X, b.Y);
}
int convexHull(point p[],int n,point ch[]) {
    sort(p, p + n, cmp);
    int m = 0;
    for(int i = 0; i < n; ++i) {
        while(m > 1 && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n - 2; i >= 0; --i) {
        while(m > k && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    return m;
}
```

8 黑科技和杂项

8.1 高精度计算

```
#include<algorithm>
using namespace std;
const int N_huge=850,base=100000000;
char s[N_huge*10];
struct huge{
    typedef long long value;
    value a[N_huge];int len;
    void clear(){len=1;a[len]=0;}
    huge(){clear();}
    huge(value x){*this=x;}
    huge operator =(huge b){
        len=b.len;for (int i=1;i<=len;++i)a[i]=b.a[i]; return *this;
    }
    huge operator +(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>len)tmp.a[i]+=b.a[i];
            else if (i>b.len)tmp.a[i]+=a[i];
            else {
```

```
                tmp.a[i]+=a[i]+b.a[i];
                if (tmp.a[i]>=base){
                    tmp.a[i]-=base;++tmp.a[i+1];
                }
            }
        }
        if (tmp.a[L+1])tmp.len=L+1;
            else tmp.len=L;
        return tmp;
    }
    huge operator -(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>b.len)b.a[i]=0;
            tmp.a[i]+=a[i]-b.a[i];
            if (tmp.a[i]<0){
                tmp.a[i]+=base;--tmp.a[i+1];
            }
        }
        while (L>1&&!tmp.a[L])--L;
        tmp.len=L;
        return tmp;
    }
    huge operator *(huge b){
        int L=len+b.len;huge tmp;
        for (int i=1;i<=L;++i)tmp.a[i]=0;
        for (int i=1;i<=len;++i)
            for (int j=1;j<=b.len;++j){
                tmp.a[i+j-1]+=a[i]*b.a[j];
                if (tmp.a[i+j-1]>=base){
                    tmp.a[i+j]+=tmp.a[i+j-1]/base;
                    tmp.a[i+j-1]%=base;
                }
            }
        tmp.len=len+b.len;
        while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
        return tmp;
    }
    pair<huge,huge> divide(huge a,huge b){
        int L=a.len;huge c,d;
        for (int i=L;i;--i){
        c.a[i]=0;d=d*base;d.a[1]=a.a[i];
            int l=0,r=base-1,mid;
            while (l<r){
                mid=(l+r+1)>>1;
                if (b*mid<=d)l=mid;
                    else r=mid-1;
            }
            c.a[i]=l;d-=b*l;
        }
        while (L>1&&!c.a[L])--L;c.len=L;
        return make_pair(c,d);
    }
    huge operator /(value x){
        value d=0;huge tmp;
        for (int i=len;i;--i){
```

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