ICPC Template

VOLEKING

2017-05-17

Contents

1 Tips

1. pow 返回浮点数可能会导致误差，慎用！！！ 自己写。
2. 样例数用 long long 可能会 TLE
3. 多组数据记得初始化
4. 小心过程量 overflow
5. Type convertions
6. 在取余的情况下，要避免减法运算的结果出现负数 (...+ MOD) % MOD;
7. fill 注意初始范围为从 1 到 n 的情况
8. 提交换行
9. c++11: long long int abs (long long int n);
10. 输入读取判断其中一个为零
11. 搜索 flag 标记可能要加上方案数

12. 数据范围特别大时考虑不可能情况
13. SegmentTree 开大点以防 RE （1e5 -> 1 << 17)
14. 滚动数组未全部清零

---

2 Basic Algorithm

2.1 Sort

```
// Bubble Sort
for (int i = 0; i < N; i++)
    for (int j = 0; j < N - i - 1; j++)
        if (A[j] > A[j + 1]) swap(A[j], A[j + 1]);
```

```
// Insertion Sort
for (int i = 1; i < N; i++) {
    int tmp = A[i], j;
    for (j = i - 1; j >= 0 && A[j] > tmp; j--)
        A[j + 1] = A[j];
    A[++j] = tmp;
}
```

```
// Selection Sort
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (A[i] > A[j]) swap(A[i], A[j])
```

```
// Merge Sort
void merge_sort(int A[], int l, int r) {
    if (l >= r) return ;
    int mid = (l + r) / 2;
    merge_sort(A, l, mid);
    merge_sort(A, mid + 1, r);
    int i, j, k;
    i = l; j = mid + 1; k = l;
    while (i <= mid && j <= r) {
        if (A[i] <= A[j]) B[k] = A[i++];
        else B[k] = A[j++];
        k++;
```

```
    }
    while (i <= mid) {
        B[k] = A[i++];
        k++;
    }
    while (j <= r) {
        B[k] = A[j++];
        k++;
    }
    memcpy(A, B, r - l + 1);
    return ;
}
```

```
// Quick Sort
void quicksort(int A[], int l, int r) {
    int i = l, j = r, mid = A[(r - l) / 2 + l];
    while (i <= j) {
        while (A[i] < mid) i++;
        while (A[j] > mid) j--;
        if (i <= j) {
            swap(A[i], A[j]);
            ++i; --j;
        }
    }
    if (i < r) quicksort(A, i, r);
    if (l < j) quicksort(A, l, j);
    return ;
}
```

- Heap Sort（见堆的内容）

2.2 DP

2.2.1 LIS

```
int A[MAX_N];
long lis(int n) {
    int dp[MAX_N];
    fill(dp, dp + n, INF);
    for (int i = 0; i < n; ++i)
        *lower_bound(dp, dp + n, A[i]) = A[i];
```

```
        // lds: -A[i]; ln: upper_bound
    return lower_bound(dp, dp + n, INF) - dp;
}
```

2.2.2 Knapsack Problem

- 0/1 背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]]+v[i])$$

```
for (int i = 0; i < N; ++i)
    for (int j = W; j >= w[i]; --j)
        f[j] = max(f[j - w[i]] + c[i], f[j]);
```

- 完全背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]]+j[i])$$

```
for (int i = 0; i < N; ++i)
    for (int j = w[i]; j <= W; ++j)
        f[j] = max(f[j - w[i]] + c[i], f[v]);
```

注意循环顺序的不同背后思路。

- 一个简单的优化：若两件物品 i、j 满足 $w[i] \leq w[j]$ 且 $c[i] \geq c[j]$，则讲物品 j 去掉，不用考虑。
- 转化为 01 背包问题求解：
  - 第 i 种物品转化为 $\frac{V}{w[i]}$ 件费用于价值均不变的物品。
  - 第 i 种物品拆成费用为 $w[i] * 2^k$，价值为 $c[i] * 2^k$ 的若干件物品其中 k 满足 $w[i] * 2^k < V$
- 多重背包

$$f[i,j] = max(f[i-1,j-w[i]*k]+v[i]*k|0 <= k <= m[i])$$

- 优化：转化为 01 背包问题
  - 将第 i 件物品分成若干件物品，每件物品的系数分别为：$1, 2, 4, \dots, 2^{(k-1)}, n[i]-2^k$
  - ** 根据 w，v 范围改变 DP 对象，可以考虑针对不同价值计算最小的重量。（$f[i][j]$，其中 j 代表价值总和）**

```
for (int i = 0; i < N; ++i) {
    int num = m[i];
    for (int k = 1; num > 0; k <<= 1) {
        int mul = min(k, num);
        for (int j = W; j >= w[i] * mul; --j) {
            f[j] = max(f[j - w[i] * mul] + v[i] * mul, f[j]);
        }
        num -= mul;
    }
}
```

- 混合三种背包

  弄清楚上面三种背包后分情况就好

- 超大背包

  - $1 \leq n \leq 40, \ 1 \leq w_i, v_i \leq 10^{15}, 1 \leq W \leq 10^{15}$

```
int n;
ll w[MAX_N], v[MAX_N], W;
Pll ps[1 << (MAX_N / 2)]; // (w, v);
void solve() {
    int n2 = n / 2;
    for (int i = 0; i < 1 << n2; ++i) {
        ll sw = 0, sv = 0;
        for (int j = 0; j < n2; ++j)
            if (i >> j & 1) {
                sw += w[j];
                sv += v[j];
            }
        ps[i] = Pll(sw, sv);
    }
    sort(ps, ps + (1 << n2));
    int m = 1;
    for (int i = 1; i < 1 << n2; ++i)
        if (ps[m - 1].second < ps[i].second)
            ps[m++] = ps[i];
    ll res = 0;
    for (int i = 0; i < 1 << (n - n2); ++i) {
        ll sw = 0, sv = 0;
        for (int j = 0; j < n - n2; ++j)
            if (i >> j & 1) {
                sw += w[n2 + j];
```

```
                sv += v[n2 + j];
            }
        if (sw <= W) {
            ll tv = (lower_bound(ps, ps + m, \
                make_pair(W - sw, INF)) - 1)->second;
            res = max(res, sv + tv);
        }
    }
    printf("%lld\n", res);
}
```

- 二维费用用背包

$$f[i,j,k] = max(f[i-1,j,k], f[i-1,j-a[i],k-b[i]]+c[i])$$

二维费用可由最多取 m 件等方式隐蔽给出。

- 分组背包

$$f[k,j] = max(f[k-1,j], f[k-1,j-w[i]]+v[i]|i \in K)$$

```
for (int k = 0; k < K; ++k)
    for (j = W; j >= 0; --j)
        for (int i = 0; i <= m[k]; ++i)
            f[j] = max(f[j - w[i]] + v[i], f[j]);
```

显然可以对每组中物品应用完全背包中"一个简单有效的优化"

- 有依赖背包

由 NOIP2006 金明的预算方案引申，对每个附件先做一个 01 背包，再与组件得到一个 $V-w[i]+1$ 个物品组。更一般问题，依赖关系由「森林」形式给出，涉及到树形 DP 以及泛化物品，这里不表。

- 背包问题方案总数

$$f[i,j] = sum(f[i-1,j], f[i-1,j-w[i]]+v[i]), f[0,0]=0$$

更多内容详见「背包九讲」

2.2.3 Maximum Subarray Sum

```
int max_subarray_sum(int A[], int n) {
    int res, cur;
    if (!A || n <= 0) return 0;
    res = cur = a[0];
    for (int i = 0; i < n; ++i) {
        if (cur < 0) cur = a[i];
        else cur += a[i];
        res = max(cur, res);
    }
    return res;
}
```

2.3 Set

```
// 子集枚举
int sub = sup;
do {
    sub = (sub - 1) & sup;
} while (sub != sup); // -1 & sup = sup;
// 势为 k 的集合枚举
int comb = (1 << k) - 1;
while (comb < 1 << n) {
    int x = comb & -comb, y = comb + x;
    comb = ((comb & ~y) / x >> 1) | y;
}
// 排列组合
do {

} while (next_permutation(A, A + N)); // prev_permutation
```

2.4 Bit operation

```
int __builtin_ffs (unsigned int x)
//返回x的最后一位1的是从后向前第几位, 比如7368 (1110011001000) 返回4。
int __builtin_clz (unsigned int x)
// 返回前导的0的个数。
int __builtin_ctz (unsigned int x)
// 返回后面的0个个数, 和__builtiin_clz相对。
int __builtin_popcount (unsigned int x)
```

```
// 返回二进制表示中1的个数。
int __builtin_parity (unsigned int x)
// 返回x的奇偶校验位，也就是x的1的个数模2的结果。
```

3 Data Structure

```
// Heap
int heap[MAX_N], sz = 0;
void push(int x) {
    int i = sz++;

    while (i > 0) {
        int p = (i - 1) / 2;
        if (heap[p] <= x) break;
        heap[p] = heap[i];
        i = p;
    }
    heap[i] = x;
}
int pop() {
    int ret = heap[0];
    int x = heap[--sz];
    int i = 0;
    while (i * 2 + 1 < sz) {
        int a = i * 2 + 1, b = i * 2 + 2;
        if (b < sz && heap[b] < heap[a]) a = b;
        if (heap[a] >= x) break;
        heap[i] = heap[a];
        i = a;
    }
    heap[i] = x;
    return ret;
}
```

```
// Binary Search Tree
struct node {
    int val;
    node *lch, rch;
};
```

```
node *insert(node *p, int x) {
    if (p == NULL) {
        node *q = new node;
        q->val = x;
        q->lch = q->rch = NULL;
        return q;
    } else {
        if (x < p->val) p->lch = insert(p->lch, x);
        else p->rch = insert(p->rch, x);
        return p;
    }
}
bool find(node *p, int x) {
    if (p == NULL) return false;
    else if (x == p->val) return true;
    else if (x < p->val) return find(p->lch, x);
    else return find(p->rch, x);
}
node *remove(node *p, int x) {
    if (p == NULL) return NULL;
    else if (x < p->val) p->lch = remove(p->lch, x);
    else if (x > p->val) p->rch = remove(p->rch, x);
    else if (p->lch == NULL) {
        node *q = p->rch;
        delete p;
        return q;
    } else if (p->lch->rch == NULL) {
        node *q = p->lch;
        q->rch = p->rch;
        delete p;
        return q;
    }  else {
        // 把左儿子子孙中最大的节点提到需要删除的节点上
        node *q;
        for (q = p->lch; q->rch->rch != NULL; q = q->rch);
        node *r = q->rch;
        q->rch = r->lch;
        r->lch = p->lch;
        r->rch = p->rch;
        delete p;
        return r;
    }
    return p;
```

```
}
```

```
// Union-find Set
int par[MAX_N];
int rnk[MAX_N];
void init(int n) {
    for (int i = 0; i < n; ++i) {
        par[i] = i;
        rnk[i] = 0;
    }
}
int find(int x) {
    return par[x] == x? x : par[x] = find(par[x]);
}
bool same(int x, int y) {
    return find(x) == find(y);
}
void unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (rnk[x] < rnk[y]) {
        par[x] = y;
    } else {
        par[y] = x;
        if (rnk[x] == rnk[y]) rnk[x]++;
    }
}
```

当然，更快捷简单的做法，是使用 C++ 的 container。

```
// Segment Tree
const int MAX_N = 1 << 17;
int n, dat[2 * MAX_N - 1];
void init(int _n) {
    n = 1;
    while (n < _n) n <<= 1;
    for (int i = 0; i < 2 * n - 1; ++i)
        dat[i] = INF;
}
void update(int k, int a) {
    k += n - 1;
    dat[k] = a;
```

```
    while (k > 0) {
        k = (k - 1) / 2;
        dat[k] = min(dat[2 * k + 1], dat[2 * k + 2]);
    }
}
// query [a, b), index k in [l, r)
// query(a, b, 0, 0, n)
int query(int a, int b, int k, int l, int r) {
    if (r <= a || b <= l) return INF;
    if (a <= l && r <= b) return dat[k];
    else {
        int v1 = query(a, b, k * 2 + 1, l, (l + r) / 2);
        int v2 = query(a, b, k * 2 + 2, (l + r) / 2, r);
        return min(v1, v2);
    }
}
```

```
// RMQ
int n, dat[2 * MAX_N - 1];
void init(int _n) {
    n = 1;
    while (n < _n) n <<= 1;
    for (int i = 0; i < 2 * n - 1; ++i)
        dat[i] = INF;
}
void update(int k, int a) {
    k += n - 1;
    dat[k] = a;
    while (k > 0) {
        k = (k - 1) / 2;
        dat[k] = min(dat[2 * k + 1], dat[2 * k + 2]);
    }
}
// query [a, b), index k in [l, r)
// query(a, b, 0, 0, n)
int query(int a, int b, int k, int l, int r) {
    if (r <= a || b <= l) return INF;
    if (a <= l && r <= b) return dat[k];
    else {
        int v1 = query(a, b, k * 2 + 1, l, (l + r) / 2);
        int v2 = query(a, b, k * 2 + 2, (l + r) / 2, r);
        return min(v1, v2);
```

```
    }
}
```

```
//Sparse Table
const int MAX_N = 1e5 + 10;
const int MAX_K = 31 - __builtin_clz(MAX_N);
int n, ST[MAX_N][MAX_K + 1], A[MAX_N];
void build(int N) {
    for (int i = 0; i < N; ++i)
        ST[i][0] = A[i];
    int k = 31 - __builtin_clz(N);
    for (int j = 1; j <= k; ++j)
        for (int i = 0; i <= N - (1 << j); ++i)
            ST[i][j] = min(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
}
int query(int l, int r) {
    if (l >= r) return 0;
    int ans = INF, k = 31 - __builtin_clz(r - l);
    for (int j = k; j >= 0; --j)
        if (l + (1 << j) - 1 <= r) {
            ans = min(ans, ST[l][j]);
            l += 1 << j;
        }
    return ans;
}
int RMQ(int l, int r) {
    if (l >= r) return 0;
    int k = 31 - __builtin_clz(r - l);
    return min(ST[l][k], ST[r - (1 << k)][k]);
}
```

---

4 Graph

```
struct edge {
    int from;
    int to, dis;
};
vector<edge> G[MAX_V];
vector<edge> es;
```

```
bool vis[MAX_V];
int V, E, pre[MAX_V], dist[MAX_V];
// int cost[MAX_V][MAX_V];
```

```
// Shortest Way
void floyed() {
    for (int k = 1; k <= n; ++k)
        for (int i = 1; i <= n; ++i)
            for (int j = 1; i <= n; ++j)
                dis[i][j] = min(dis[i][k] + dis[k][j], dis[i][j]);
}
void dijkstra(int s) {
    // fisrt 是最短距离, second 是顶点编号
    priority_queue<Pii, vector<Pii>, greater<Pii> > que;
    fill(dist, dist + V, INF);
    dist[s] = 0; que.push(Pii(0, s));
    while (!que.empty()) {
        Pii p = que.top(); que.pop();
        int v = p.second;
        if (dist[v] < p.first) continue;
        for (int i = 0; i < G[v].size(); i++) {
            edge e = G[v][i];
            if (dist[e.to] > dist[v] + e.dis) {
                dist[e.to] = dist[v] + e.dis;
                que.push(Pii(dist[e.to], e.to));
            }
        }
    }
}
void bellman_ford(int s) {
    fill(dist, dist + V, INF);
    dist[s] = 0;
    while (true) {
        bool update = false;
        for (int i = 0; i < E; ++i) {
            edge e = es[i];
            if (dist[e.from] != INF && dist[e.from] + e.dis < dist[e.to])
                {
                update = true;
                dist[e.to] = dist[e.from] + e.dis;
            }
        }
```

```
        if (!update) break;
    }
}
bool find_negative_loop() {
    memset(dist, 0, sizeof dist);
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < E; ++j) {
            edge e = es[j];
            if (d[e.to] > d[e.from] + e.dis) {
                d[e.to] = d[e.from] + e.dis;
                if (i == V - 1) return true;
            }
        }
    return false;
}
void spfa(int s) {
    queue<int> que;
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[s] = 0; que.push(s); vis[s] = true;
    while (!que.empty()) {
        int v = que.front(); que.pop();
        vis[v] = false;
        for (int i = 0; i < G[v].size(); ++i) {
            int u = G[v][i].to;
            if (dist[u] > dist[v] + G[v][i].dis) {
                dist[u] = dist[v] + G[v][i].dis;
                if (!vis[u]) {
                    que.push(u);
                    vis[u] = true;
                }
            }
        }
    }
}
```

```
// Spanning Tree
int prime() {
    /*
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[0] = 0;
```

```
    int res = 0;
    while (true) {
        int v = -1;
        for (int u = 0; u < V; ++u) {
            if(!vis[u] && (v == -1 || dist[u] < dist[v])) v = u;
        }
        if (v == -1) break;
        vis[v] = true;
        res += dist[v];
        for (int u = 0; u < V; u++)
            dist[u] = min(dist[u], cost[v][u]);
    }
    //*/
    priority_queue<Pii, vector<Pii>, greater<Pii> > que;
    int res = 0;
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[0] = 0;
    que.push(Pii(0, 0));
    while (!que.empty()) {
        Pii p = que.top(); que.pop();
        int v = p.second;
        if (vis[v] || dist[v] < p.first) continue;
        res += dist[v]; vis[v] = true;
        for (int i = 0; i < G[v].size(); ++i) {
            edge e = G[v][i];
            if (dist[e.to] > e.dis) {
                dist[e.to] = e.dis;
                que.push(Pii(dist[e.to], e.to));
            }
        }
    }
    return res;
}
bool cmp(const edge e1, const edge e2) {
    return e1.dis < e2.dis;
}
int kruskal() {
    sort(es.begin(), es.end(), cmp);
    init(V);
    int res = 0;
    for (int i = 0; i < E; ++i) {
        edge e = es[i];
```

```
        if (!same(e.from, e.to)) {
            unite(e.from, e.to);
            res += e.dis;
        }
    }
    return res;
}
```

```
// SCC
int V, cmp[MAX_V];
vector<int> G[MAX_V], rG[MAX_V], vs;
bool used[MAX_V];

void add_edge(int from, int to) {
    G[from].push_back(to); rG[to].push_back(from);
}
void dfs(int v) {
    used[v] = true;
    for (int i = 0; i < G[v].size(); ++i)
        if (!used[G[v][i]]) dfs(G[v][i]);
    vs.push_back(v);
}
void rdfs(int v, int k) {
    used[v] = true;
    cmp[v] = k;
    for (int i = 0; i < rG[v].size(); ++i)
        if (!used[rG[v][i]]) rdfs(rG[v][i], k);
}
int scc() {
    memset(used, 0, sizeof used);
    vs.clear();
    for (int v = 0; v < V; ++v)
        if (!used[v]) dfs(v);
    memset(used, 0, sizeof used);
    int k = 0;
    for (int i = vs.size() - 1; i >= 0; --i)
        if (!used[vs[i]]) rdfs(vs[i], k++);
    return k;
}
```

```
// Bipartite Matching
void add_edge(int u, int v) {
```

```
    G[u].push_back(v); G[v].push_back(u);
}
bool dfs(int v) {
    used[v] = true;
    rep(i, 0, G[v].size()) {
        int u = G[v][i], w = match[u];
        if (w < 0 || (!used[w] && dfs(w))) {
            match[v] = u; match[u] = v;
            return true;
        }
    }
    return false;
}
int bipartite_matching() {
    int res = 0;
    memset(match, -1, sizeof match);
    rep (v, 0, V)
        if (match[v] < 0) {
            memset(used, false, sizeof used);
            if (dfs(v)) ++res;
        }
    return res;
}
```

```
// Network Flow
struct edge{
    int to, cap, rev;
};
vector<edge> G[MAX_V];
int level[MAX_V], iter[MAX_V];
void add_edge(int from, int to, int cap) {
    G[from].push_back((edge){to, cap, static_cast<int>(G[to].size())});
    G[to].push_back((edge){from, 0, static_cast<int>(G[from].size() - 1)})
        ;
}
// Ford-Fulkerson
int dfs(int v, int t, int f) {
    if (v == t) return f;
    flag[v] = true;
    rep(i, 0, G[v].size()) {
        edge &e = G[v][i];
        if (!flag[e.to] && e.cap > 0) {
```

```
                int d = dfs(e.to, t, min(f, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
    return 0;
}
int max_flow(int s, int t) {
    int flow = 0;
    for(;;) {
        memset(flag, false, sizeof flag);
        int f = dfs(s, t, INF);
        if (!f) return flow;
        flow += f;
    }
}
// Dinic
void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0; que.push(s);
    while (!que.empty()) {
        int v = que.front(); que.pop();
        for (int i = 0; i < G[v].size(); ++i) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}
int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int &i = iter[v]; i < G[v].size(); ++i) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
```

```
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}
int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof iter);
        int f;
        while ((f = dfs(s, t, INF)) > 0) {
            flow += f;
        }
    }
}
```

```
// min_cost_flow
void add_edge(int from, int to, int cap, int cost) {
    G[from].push_back((edge){to, cap, cost, (int)G[to].size()});
    G[to].push_back((edge){from, 0, -cost, (int)G[from].size() - 1});
}
int min_cost_flow(int s, int t, int f) {
    int res = 0;
    fill(h, h + V, 0);
    while (f > 0) {
        priority_queue<Pii, vector<Pii>, greater<Pii> > que;
        fill(dist, dist + V, INF);
        dist[s] = 0; que.push(Pii(0, s));
        while (!que.empty()) {
            Pii p = que.top(); que.pop();
            int v = p.second;
            if (dist[v] < p.first) continue;
            rep(i, 0, G[v].size()) {
                edge &e = G[v][i];
                if (e.cap > 0 \
                  && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
                    dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                    prevv[e.to] = v;
```

```
                        preve[e.to] = i;
                        que.push(Pii(dist[e.to], e.to));
                    }
                }
            }
            if (dist[t] == INF) return -1;
            rep(v, 0, V) h[v] += dist[v];
            int d = f;
            for (int v = t; v != s; v = prevv[v])
                d = min(d, G[prevv[v]][preve[v]].cap);
            f -= d;
            res += d * h[t];
            for (int v = t; v != s; v = prevv[v]) {
                edge &e = G[prevv[v]][preve[v]];
                e.cap -= d;
                G[v][e.rev].cap += d;
            }
        }
        return res;
}
```

```
// stoer_wagner 全局最小割
void search() {
    memset(vis, false, sizeof vis);
    memset(wet, 0, sizeof wet);
    S = T = -1;
    int imax, tmp;
    rep(i, 0, V) {
        imax = -INF;
        rep(j, 0, V)
            if (!cmb[j] && !vis[j] && wet[j] > imax) {
                imax = wet[j];
                tmp = j;
            }
        if (T == tmp) return;
        S = T; T = tmp;
        mc = imax;
        vis[tmp] = true;
        rep(j, 0, V)
            if (!cmb[j] && !vis[j])
                wet[j] += G[tmp][j];
    }
```

```
}
int stoer_wagner() {
    memset(cmb, false, sizeof cmb);
    int ans = INF;
    rep(i, 0, V - 1) {
        search();
        ans = min(ans, mc);
        if (ans == 0) return 0;
        cmb[T] = true;
        rep(j, 0, V)
            if (!cmb[j]) {
                G[S][j] += G[T][j];
                G[j][S] += G[j][T];
            }
    }
    return ans;
}
```

```
// LCA--Doubling
const int MAX_LOG_V = 32 - __builtin_clz(MAX_V);
vector<int> G[MAX_V];
int root, parent[MAX_LOG_V][MAX_V], depth[MAX_V];
void dfs(int v, int p, int d) {
    parent[0][v] = p;
    depth[v] = d;
    for (int i = 0; i < G[v].size(); i++)
        if (G[v][i] != p) dfs(G[v][i], v, d + 1);
}
void init(int V) {
    dfs(root, -1, 0);
    for (int k = 0; k + 1 < MAX_LOG_V; k++)
        for (int v = 0; v < V; v++)
            if (parent[k][v] < 0) parent[k + 1][v] = -1;
            else parent[k + 1][v] = parent[k][parent[k][v]];
}
int lca(int u, int v) {
    if (depth[u] > depth[v]) swap(u, v);
    for (int k = 0; k < MAX_LOG_V; k++)
        if ((depth[v] - depth[u]) >> k & 1)
            v = parent[k][v];
    if (u == v) return u;
    for (int k = MAX_LOG_V - 1; k >= 0; k--)
```

```
        if (parent[k][u] != parent[k][v])
            u = parent[k][u], v = parent[k][v];
    return parent[0][u];
}
// LCA--RMQ
vector<int> G[MAX_V];
int root, vs[MAX_V * 2 - 1], depth[MAX_V * 2 - 1], id[MAX_V];
int ST[2 * MAX_V][MAX_K];
void rmq_init(int* A, int N) {
    for (int i = 0; i < N; i++)
        ST[i][0] = i;
    int k = 31 - __builtin_clz(N);
    for (int j = 1; j <= k; j++)
        for (int i = 0; i <= N - (1 << j); ++i)
            if (A[ST[i][j - 1]] <= A[ST[i + (1 << (j - 1))][j - 1]])
                ST[i][j] = ST[i][j - 1];
            else ST[i][j] = ST[i + (1 << (j - 1))][j - 1];
}
int query(int l, int r) {
    if (l >= r) return -1;
    int k = 31 - __builtin_clz(r - l);
    if (depth[ST[l][k]] <= depth[ST[r - (1 << k)][k]])
        return ST[l][k];
    else return ST[r - (1 << k)][k];
}
void dfs(int v, int p, int d, int &k) {
    id[v] = k;
    vs[k] = v;
    depth[k++] = d;
    for (int i = 0; i < G[v].size(); i++) {
        if (G[v][i] != p) {
            dfs(G[v][i], v, d + 1, k);
            vs[k] = v;
            depth[k++] = d;
        }
    }
}
void init(int V) {
    int k = 0;
    dfs(root, -1, 0, k);
    rmq_init(depth, 2 * V - 1);
}
int lca(int u, int v) {
```

```
    return vs[query(min(id[u], id[v]), max(id[u], id[v]) + 1)];
}
```

5 Computational Geometry

```
const double EPS = 1e-8;
int sgn(double x) { return x < -EPS ? -1 : x > EPS ? 1 : 0;}
struct Point {
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}
    Point operator + (Point p) { return Point(x + p.x, y + p.y); }
    Point operator - (Point p) { return Point(x - p.x, y - p.y); }
    Point operator * (double d) { return Point(x * d, y * d); }
    bool operator < (Point p) {
        return x != p.x? x < p.x : y < p.y;
    }
    double dot(Point p) { return add(x * p.x, y * p.y); }// 内积
    double det(Point p) { return add(x * p.y, -y * p.x);}// 外积
};
bool on_seg(Point p1, Point p2, Point q) {
    return (p1 - q).det(p2 - q) == 0 && (p1 - q).dot(p2 - q) <= 0;
}
P intersection(Point p1, Point p2, Point q1, Point q2) {
    return p1 + (p2 - p1) * ((q2 - q1).det(q1 - p1) / (q2 - q1).det(p2 -
       p1));
}
// 凸包
int convex_hull(Point *ps, int n, Point *ch) {
    sort(ps, ps + n);
    int k = 0;
    for (int i = 0; i < n; ++i) {
        while (k > 1 && (ch[k - 1] - ch[k - 2].det(ps[i] - ch[k - 1])) <=
           0)
            k--;
        ch[k++] = ps[i];
    }
    for (int i = n - 2, t = k; i >= 0; --i) {
        while (k > t && (ch[k - 1] - ch[k - 2].det(ps[i] - ch[k - 1])) <=
           0)
```

```
            k--;
        ch[k++] = ps[i];
    }
    return k - 1;
}
```

Simpson 公式——二次函数近似原函数积分:

$$\int_a^b f(x)dx \approx \frac{b-a}{6} * \Big(f(a) + 4f(\frac{a+b}{2}) + f(b)\Big)$$

---

6 Math Problem

```
// returning count of nk in range [l, r], from Infinity
template<typename T> T mps(T l, T r, T k) {
    return ((r - (r % k + k) % k) - (l + (k - l % k) % k)) / k + 1;
}
template<typename T> T gcd(T a, T b) {
    //return (b)? gcd(b, a  % b) : a;
    while (b) { T t = a % b; a = b; b = t; } return a;
}
template<typename T> T lcm(T a, T b) {
    return a / gcd(a, b) * b;
}
// find (x, y) s.t. a x + b y = gcd(a, b) = d
template<typename T> T exgcd(T a, T b, T &x, T &y) {
    T d = a;
    if (b) {
        d = exgcd(b, a % b, y, x);
        y -= a / b * x;
    } else {
        x = 1; y = 0;
    }
    return d;
}
template<typename T> T modular_linear(T a, T b, T n) {
    T d, e, x, y;
    d = exgcd(a, n, x, y);
    if (b % d)
        return -1;
```

```
    e = x * (b / d) % n + n;
    return e % (n / d);
}
template<typename T> T mod_mult(T a, T b, T mod) {
  T res = 0;
  while (b) {
    if (b & 1) {
        res = (res + a) % mod;
        // res += a;
        // if (res >= mod) res -= mod;
    }
    a = (a + a) % mod;
    // a <<= 1;
    // if (a >= mod) a -= mod;
    b >>= 1;
  }
  return res;
}
template<typename T> T mod_pow(T x, T n, T mod) {
    T res = 1;
    while (n) {
        if (n & 1) res = mod_mult(res, x, mod);
        x = mod_mult(x, x, mod);
        n >>= 1;
    }
    return res;
    // return b ? \
        mod_pow(a * a % mod, b >> 1, mod) * (b & 1 ? a : 1) % mod : 1;
}
template<typename T> T mod_inverse(T a, T m) {
    T x, y;
    exgcd(a, m, x, y);
    return (m + x % m) % m;
}
// 线性求逆元 小心乘法溢出?
void init_inverse() {
    inv[1] = 1;
    for (int i = 2; i < MAX_N; i++)
        inv[i] = (MOD - (MOD / i) * inv[MOD % i] % MOD) % MOD;
}
//A[i] * x % M[i] = B[i];
std::pair<int, int> linear_congruence(const std::vector<int> &A, const std
    :: \
```

```
    vector<int> &B, const std::vector<int> &M) {
        // wa 了把中间量开大? * 溢出
        int x = 0, m = 1;
        for(int i = 0; i < A.size(); i++) {
                int a = A[i] * m, b = B[i] - A[i] * x, d = gcd(M[i], a);
                if(b % d != 0) return std::make_pair(0, -1); // no
                    solutioin
                int t = b / d * mod_inverse(a / d, M[i] / d) % (M[i] / d);
                x = x + m * t;
                m *= M[i] / d;
        }
        while (x < m) x += m;
        return std::make_pair(x % m, m);
}
ll CRT(vector<ll> &a, vector<ll> &m) {
    ll M = 1LL, res = 0;
    for (int i = 0; i < m.size(); ++i)
        M *= m[i];
    for (int i = 0; i < m.size(); ++i) {
        ll Mi, Ti;
        Mi = M / m[i]; Ti = mod_inverse(Mi, mi);
        res = (res + a[i] * (Mi * Ti % M) % M) % M;
    }
    return res;
}
// only for MOD < 1e6;
ll fact[MOD + 10];
void init() {
    fact[0] = 1;
    for (int i = 1; i <= MOD; ++i)
        fact[i] = fact[i - 1] * i % MOD;
}
int mod_fact(int n, int p, int &e) {
    e = 0;
    if (n == 0) return 1;
    int res = mod_fact(n / p, p, e);
    e += n / p;
    if (n / p % 2 != 0) return res * (p - fact[n % p]) % p;
    return res * fact[n % p] % p;
}
int mod_comb(int n, int k, int p) {
    if (n < 0 || k < 0 || n < k) return 0;
    if (n == 0) return 1;
```

```
    int e1, e2, e3;
    int a1 = mod_fact(n, p, e1);
    int a2 = mod_fact(k, p, e2);
    int a3 = mod_fact(n - k, p, e3);
    if (e1 > e2 + e3) return 0;
    return a1 * mod_inverse(a2 * a3 % p, p) % p;
}
ll lucas(ll n, ll k, const ll &p) {
    if (n < 0 || k < 0 || n < k) return 0;
    if (n == 0) return 1;
    return lucas(n / p, k / p, p) * mod_comb(n % p, k % p, p) % p;
}
```

```
// 矩阵快速幂
typedef vector<int> vec;
typedef vector<vec> mat;
mat G(MAX_N);
mat mat_mul(mat &A, mat &B) {
    mat C(A.size(), vec(B[0].size()));
    for (int i = 0; i < A.size(); ++i)
        for (int k = 0; k < B.size(); ++k)
            for (int j = 0; j < B[0].size(); ++j)
                C[i][j] = (C[i][j] + A[i][k] % MOD * B[k][j] % MOD \
                 + MOD) % MOD;
    return C;
}
mat mat_pow(mat A, ll n) {
    mat B(A.size(), vec(A.size()));
    for (int i = 0; i < A.size(); ++i)
        B[i][i] = 1;
    while (n > 0) {
        if (n & 1) B = mat_mul(B, A);
        A = mat_mul(A, A);
        n >>= 1;
    }
    return B;
}
```

```
// prime number
bool is_prime(int n) {
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
```

```
    return n != 1;
}
vector<int> divisor(int n) {
    vector<int> res;
    for (int i = 1; i * i <= n; ++i) {
        if (n % i == 0) {
            res.push_back(i);
            if (i != n / i) res.push_back(n / i);
        }
    }
    return res;
}
map<int, int> prime_factor(int n) {
    map<int, int> res;
    for (int i = 2; i * i <= n; ++i) {
        while (n % i == 0) {
            ++res[i];
            n /= i;
        }
    }
    if (n != 1) res[n] = 1;
    return res;
}
int prime[MAX_N];
bool isPrime[MAX_N + 1];
int seive(int n) {
    int p = 0;
    fill(isPrime, isPrime + n + 1, true);
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i <= n; ++i)
        if (isPrime[i]) {
            prime[p++] = i;
            for (int j = 2 * i; j <= n; j += i) isPrime[j] = false;
        }
    return p;
}
// the number of prime in [L, r)
// 对区间 [l, r) 内的整数执行筛法，prime[i - l] = true <=> i 是素数
bool segPrimeSmall[MAX_L];
bool segPrime[MAX_SQRT_R];
void segment_sieve(ll l, ll r) {
    for (int i = 0; (ll)i * i < r; ++i) segPrimeSmall[i] = true;
    for (int i = 0; i < r - l; ++i) segPrime[i] = true;
```

```
    for (int i = 2; (ll)i * i < r; ++i) {
        if (segPrimeSmall[i]) {
            for (int j = 2 * i; (ll)j * j <= r; j += i)
                segPrimeSmall[j] = false;
            for (ll j = max(2ll, (l + i - 1) / i) * i; j < r; j += i)
                segPrime[j - l] = false;
        }
    }
}
// Miller_Rabin
bool check(ll a, ll n, ll x, ll t) {
    ll res = mod_pow(a, x, n);
    ll last = res;
    for (int i = 1; i <= t; ++i) {
        res = mod_mult(res, res, n);
        if (res == 1 && last != 1 && last != n - 1) return true;
        last = res;
    }
    if (res != 1) return true;
    return false;
}
bool Miller_Rabin(ll n) {
    if (n < MAX_N) return isPrime[n]; // small number may get wrong answer
        ?!
    if (n < 2) return false;
    if (n == 2) return true;
    if ((n & 1) == 0) return false;
    ll x = n - 1, t = 0;
    while ((x & 1) == 0) {
        x >>= 1;
        ++t;
    }
    for (int i = 0; i < S; ++i) {
        ll a = rand() % (n - 1) + 1;
        if (check(a, n, x, t))
            return false;
    }
    return true;
}
// find factors
vector<ll> factor;
ll Pollard_rho(ll x, ll c) {
    ll i = 1, k = 2;
```

```
    ll x0 = rand() % x;
    ll y = x0;
    while (true) {
        ++i;
        x0 = (mod_mult(x0, x0, x) + c) % x;
        ll d;
        if (y == x0) d = 1;
        else
            if (y > x0)
                d = gcd(y - x0, x);
            else d = gcd(x0 - y, x);
        if (d != 1 && d != x) return d;
        if (y == x0) return x;
        if (i == k) {
            y = x0;
            k += k;
        }
    }
}
void find_factor(ll n) {
    if (n == 1) return ;
    if (Miller_Rabin(n)) {
        factor.push_back(n);
        return ;
    }
    ll p = n;
    while (p >= n) p = Pollard_rho(p, rand() % (n - 1) + 1);
    find_factor(p);
    find_factor(n / p);
}
```

```
#include<bits/stdc++>
//Meisell-Lehmer
const int MAX_N = 5e6 + 2;
bool np[MAX_N];
int prime[MAX_N], pi[MAX_N];
int getprime() {
    int cnt = 0;
    np[0] = np[1] = true;
    pi[0] = pi[1] = 0;
    for(int i = 2; i < MAX_N; ++i) {
        if(!np[i]) prime[++cnt] = i;
```

```
        pi[i] = cnt;
        for(int j = 1; j <= cnt && i * prime[j] < MAX_N; ++j) {
            np[i * prime[j]] = true;
            if(i % prime[j] == 0)   break;
        }
    }
    return cnt;
}
const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
int phi[PM + 1][M + 1], sz[M + 1];
void init() {
    getprime();
    sz[0] = 1;
    for(int i = 0; i <= PM; ++i)  phi[i][0] = i;
    for(int i = 1; i <= M; ++i) {
        sz[i] = prime[i] * sz[i - 1];
        for(int j = 1; j <= PM; ++j)
            phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];
    }
}
int sqrt2(ll x) {
    ll r = (ll)sqrt(x - 0.1);
    while(r * r <= x)   ++r;
    return int(r - 1);
}
int sqrt3(ll x) {
    ll r = (ll)cbrt(x - 0.1);
    while(r * r * r <= x)   ++r;
    return int(r - 1);
}
ll getphi(ll x, int s) {
    if(s == 0)  return x;
    if(s <= M)  return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    if(x <= prime[s]*prime[s])   return pi[x] - s + 1;
    if(x <= prime[s]*prime[s]*prime[s] && x < MAX_N) {
        int s2x = pi[sqrt2(x)];
        ll ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
        for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];
        return ans;
    }
    return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
}
```

```
ll getpi(ll x) {
    if(x < MAX_N)   return pi[x];
    ll ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
    for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i)
        ans -= getpi(x / prime[i]) - i + 1;
    return ans;
}
ll lehmer_pi(ll x) {
    if(x < MAX_N)   return pi[x];
    int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x));
    int c = (int)lehmer_pi(sqrt3(x));
    ll sum = getphi(x, a) +(ll)(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++) {
        ll w = x / prime[i];
        sum -= lehmer_pi(w);
        if (i > c) continue;
        ll lim = lehmer_pi(sqrt2(w));
        for (int j = i; j <= lim; j++)
            sum -= lehmer_pi(w / prime[j]) - (j - 1);
    }
    return sum;
}
int main() {
    init();
    ll n;
    while(~scanf("%lld",&n)) {
        printf("%lld\n",lehmer_pi(n));
    }
    return 0;
}
```

```
// 欧拉函数
int euler_phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            res = res / i * (i - 1);
            for (; n % i == 0; n /= i);
        }
    }
    if (n != 1) res = res / n * (n - 1);
```

```
    return res;
}
int euler[MAX_N];
void euler_phi_sieve() {
    for (int i = 0; i < MAX_N; ++i) euler[i] = i;
    for (int i = 2; i < MAX_N; ++i)
        if (euler[i] == i)
            for (int j = i; j < MAX_N; j += i)
                euler[j] = euler[j] / i * (i - 1);
}
```

- Moebius 如果

$$F(n) = \sum_{d|n} f(d)$$

，则

$$f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$$

对于 $\mu(d)$ 函数，有如下性质：

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{n = 1} \\ 0 & \text{n > 1} \end{cases}$$

$$\sum_{d|n} \frac{\mu(d)}{d} = \frac{\phi(n)}{n}$$

```
int mu[MAX_N];
void moebius() {
    int cnt = 0; mu[1] = 1;
    memset(vis, 0, sizeof vis);
    for (int i = 2; i < MAX_N; ++i) {
        if (!vis[i]) {
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < cnt && i * prime[j] < MAX_N; ++j) {
            vis[i * prime[j]] = true;
            if (i % prime[j])
                mu[i * prime[j]] = -mu[i];
            else
                mu[i * prime[j]] = 0, break;
        }
```

```
    }
}
map<int, int> moebius(int n) {
    map<int, int> res;
    vector<int> primes;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            primes.push_back(i);
            while (n % i == 0) n /= i;
        }
    }
    if (n != 1) primes.push_back(n);
    int m = primes.size();
    for (int i = 0; i < (1 << m); ++i) {
        int mu = 1, d = 1;
        for (int j = 0; j < m; ++j) {
            if (i >> j & 1) {
                mu *= -1;
                d *= primes[j];
            }
        }
        res[d] = mu;
    }
    return res;
}
```

```
// Guass_jordan
const double eps = 1e-8;
typedef vector<double> vec;
typedef vector<vec> mat;
vec gauss_joedan(const mat &A, const vec& b) {
    int n = A.size();
    mat B(n, vec(n + 1));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
    for (int i = 0; i < n; ++i) B[i][n] = b[i];
    for (int i = 0; i < n; ++i) {
        int pivot = i;
        for (int j = i; j < n; ++j)
            if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
        if (i != pivot) swap(B[i], B[pivot]);
        if (abs(B[i][i]) < eps) return vec();
```

```
        for (int j = i + 1; j <= n; ++j) B[i][j] /= B[i][i];
        for (int j = 0; j < n; ++j) if (i != j)
            for (int k = i + 1; k <= n; ++k) B[j][k] -= B[j][i] * B[i][k];
    }
    vec x(n);
    for (int i = 0; i < n; ++i) x[i] = B[i][n];
    return x;
}
vec gauss_joedan_xor(const mat& A, const vec& b) {
    int n = A.size();
    mat B(n, vec(n + 1));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
    for (int i = 0; i < n; ++i) B[i][n] = b[i];
    for (int i = 0; i < n; ++i) {
        int pivot = i;
        for (int j = i; j < n; ++j)
            if (B[j][i]) {
                pivot = j;
                break;
            }
        if (pivot != i) swap(B[i], B[pivot]);

        for (int j = 0; j < n; ++j) if (i != j && B[j][i])
                for (int k = i + 1; k <= n; ++k) B[j][k] ^= B[i][k];
            }
    }
    vec x(n);
    for (int i = 0; i < n; ++i) x[i] = B[i][n];
    return x;
}
```

---

7 String

1. Hash 2. KMP 3. Extend KMP 4. trie 树 poj2001 2503 3630 1056 hdu 1075 1251 1247 1298 1671 5. Manacher 算法 6. AC 自动机 7. 后缀数组 8. 后缀树 9. 后缀自动机 10. 回文自动机

```
// 最小最大表示法:
int getMinString(const string &s) {
```

```
    int len = (int)s.length();
    int i = 0, j = 1, k = 0;
    while(i < len && j < len && k < len) {
        int t = s[(i + k) % len] - s[(j + k) % len];
        if(t == 0) k++;
        else {
            if(t > 0) i += k + 1;//getMaxString: t < 0
            else j += k + 1;
            if(i == j) j++;
            k = 0;
        }
    }
    return min(i, j);
}
```

```
// KMP
int nxt[MAX_N];
void getNext(const string &str) {
    int len = str.length();
    int j = 0, k;
    k = nxt[0] = -1;
    while (j < len) {
        if (k == -1 || str[j] == str[k])
            nxt[++j] = ++k;
        else k = nxt[k];
    }
}
int kmp(const string &tar, const string &pat) {
    getNext(pat);
    int num, j, k;
    int lenT = tar.length(), lenP = pat.length();
    num = j = k = 0;
    while (j < lenT) {
        if(k == -1 || tar[j] == pat[k])
            j++, k++;
        else k = nxt[k];
        if(k == lenP) {
            // res = max(res, j - lenP);
            k = nxt[k];
            ++num;
        }
    }
```

```
    return num;//lenP - res - 1;
}
```

```
// Suffix Array & LCP Array
int n, k;
int lcp[MAX_N], sa[MAX_N];
int rnk[MAX_N], tmp[MAX_N];
bool compare_sa(int i, int j) {
    if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];
    else {
        int ri = i + k <= n? rnk[i + k] : -1;
        int rj = j + k <= n? rnk[j + k] : -1;
        return ri < rj;
    }
}
void construct_sa(string S, int *sa) {
    n = S.length();
    for (int i = 0; i <= n; i++) {
        sa[i] = i;
        rnk[i] = i < n? S[i] : -1;
    }
    for (k = 1; k <= n; k *= 2) {
        sort(sa, sa + n + 1, compare_sa);
        tmp[sa[0]] = 0;
        for (int i = 1; i <= n; i++)
            tmp[sa[i]] = tmp[sa[i - 1]] + \
             (compare_sa(sa[i - 1], sa[i]) ? 1 : 0);
        memcpy(rnk, tmp, sizeof(int) * (n + 1));
    }
}
void construct_lcp(string S, int *sa, int *lcp) {
    int n = S.length();
    for (int i = 0; i <= n; i++) rnk[sa[i]] = i;
    int h = 0;
    lcp[0] = 0;
    for (int i = 0; i < n; i++) {
        int j = sa[rnk[i] - 1];
        if (h > 0) h--;
        for (; j + h < n && i + h < n; h++)
            if (S[j + h] != S[i + h]) break;
        lcp[rnk[i] - 1] = h;
    }
```

```
}
```

```
// AC 自动机
int ans[MAX_N], d[MAX_N];
struct Trie {
    int nxt[MAX_N][26], fail[MAX_N], end[MAX_N];
    int root, L;
    int newnode() {
        for(int i = 0; i < 26; i++)
            nxt[L][i] = -1;
        end[L++] = 0;
        return L-1;
    }
    void init() {
        L = 0;
        root = newnode();
    }
    void insert(char buf[]) {
        int len = strlen(buf);
        int now = root;
        for(int i = 0; i < len; i++) {
            if(nxt[now][buf[i]-'a'] == -1)
                nxt[now][buf[i]-'a'] = newnode();
            now = nxt[now][buf[i]-'a'];
        }
        end[now] = 1;
        d[now] = len;
    }
    void build() {
        queue<int> Q;
        fail[root] = root;
        for(int i = 0; i < 26; i++)
            if(nxt[root][i] == -1)
                nxt[root][i] = root;
            else {
                fail[nxt[root][i]] = root;
                Q.push(nxt[root][i]);
            }
        while( !Q.empty() ) {
            int now = Q.front(); Q.pop();
            for(int i = 0; i < 26; i++)
                if(nxt[now][i] == -1)
```

```
                    nxt[now][i] = nxt[fail[now]][i];
                else {
                    fail[nxt[now][i]] = nxt[fail[now]][i];
                    Q.push(nxt[now][i]);
                }
        }
    }
    void solve(char buf[]) {
        int cur = root;
        int len = strlen(buf);
        int index;
        for(int i = 0; i < len; ++i) {
            if(buf[i] >= 'A' && buf[i] <= 'Z')
                index = buf[i] - 'A';
            else if(buf[i] >= 'a' && buf[i] <= 'z')
                index = buf[i] - 'a';
            else continue;
            cur = nxt[cur][index];
            int x = cur;
            while(x != root) {
                if(end[x]) {
                    ans[i + 1] -= 1;
                    ans[i - d[x] + 1] += 1;
                    break;
                }
                x = fail[x];
            }
        }
    }
};
Trie ac;
```

---

8 Others

8.1 Divide-and-Conquer Tree

```
//uva 12161
struct edge {
    int to, damage, length, next;
```

```
};
int G[MAX_N], En, N, M, T;
edge E[MAX_N * 2];
void add_edge(int from, int to, int damage, int length) {
    edge e = {to, damage, length, G[from]};
    E[En] = e;
    G[from] = En++;
}
int ans, subtree_size[MAX_N];
bool flag[MAX_N];
int s, t;
Pii ds[MAX_N];
int compute_subtree_size(int v, int p) {
    int c = 1;
    for (int j = G[v]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (w == p || flag[w]) continue;
        c += compute_subtree_size(w, v);
    }
    return subtree_size[v] = c;
}
Pii search_centroid(int v, int p, int t) {
    Pii res = Pii(INT_MAX, -1);
    int s = 1, m = 0;
    for (int j = G[v]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (w == p || flag[w]) continue;
        res = min(res, search_centroid(w, v, t));
        m = max(subtree_size[w], m);
        s += subtree_size[w];
    }
    m = max(m, t - s);
    res = min(res, Pii(m, v));
    return res;
}
void enumrate_path(int v, int p, int damage, int length) {
    ds[t++] = Pii(damage, length);
    for (int j = G[v]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (w == p || flag[w]) continue;
        if (damage + E[j].damage <= M) {
            enumrate_path(w, v, damage + E[j].damage, length + E[j].length
                );
```

```
        }
    }
}
void remove_useless(int s, int &t) {
    if (s == t) return;
    int tt;
    for (int i = tt = s + 1; i < t; i++) {
        if (ds[i].first == ds[tt - 1].first) continue;
        if (ds[i].second <= ds[tt - 1].second) continue;
        ds[tt++] = ds[i];
    }
    t = tt;
}
void solve_sub_problem(int v) {
    compute_subtree_size(v, -1);
    int c = search_centroid(v, -1, subtree_size[v]).second;
    flag[c] = true;
    for (int j = G[c]; ~j; j = E[j].next) {
        if (flag[E[j].to]) continue;
        solve_sub_problem(E[j].to);
    }
    s = t = 0;
    for (int j = G[c]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (flag[w]) continue;
        if (E[j].damage <= M)
            enumrate_path(w, v, E[j].damage, E[j].length);
        if (s > 0) {
            sort(ds + s, ds + t);
            remove_useless(s, t);
            for (int l = 0, r = t - 1; l < s && r >= s; l++) {
                while (r >= s && ds[l].first + ds[r].first > M) r--;
                if (r >= s)
                    ans = max(ans, ds[l].second + ds[r].second);
            }
        }
        sort(ds, ds + t);
        remove_useless(0, t);
        s = t;
    }
    flag[c] = false;
}
```