ACM TEMPLATES

DaDa

dadamrxx@gmail.com
Beijing Institute of Technology

2017.4

Contests

Part I. Graph Theory

1. Graph

Description

有向带权图的邻接表存储，用静态链表实现。

若不需要权重 w，则将 w 的两处出现删去即可。

以邻接表存储图时，结点编号 1 到 n 或 0 到 n-1 均可。

Code

```
const int N = 1e3 + 10;
const int M = 1e3 + 10;

struct Edge
{
    int to, w, next;
} edge[M];
int adj[N], no;
int n, m;

void init()
{
    memset(adj, -1, sizeof(adj));
    no = 0;
}
void add(int u, int v, int w)
{
    edge[no].to = v;
    edge[no].w = w;
    edge[no].next = adj[u];
    adj[u] = no++;
}
```

2. Dijkstra

Description

堆优化的 Dijkstra 算法，时间复杂度 O(VlogV)。

输入：以邻接表存储的有向带权图 adj[]，起点 start。

输出：起点 start 到每个结点的最短距离 dis[]。

条件：可以有负边，但不允许有负圈。

Code

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int N = 1e3 + 10;
const int M = 1e3 + 10;

Graph...

int dis[N];
typedef pair<int, int> pii;
priority_queue<pii, vector<pii>, greater<pii> > pq;
void dijkstra(int start)
{
    memset(dis, 0x3f, sizeof(dis));
    dis[start] = 0;
    while (!pq.empty()) pq.pop();
    pq.push(pii(0, start));
    while (!pq.empty())
    {
        pii p = pq.top(); pq.pop();
        int u = p.second;
        if (dis[u] < p.first) continue;
        for (int i = adj[u]; i != -1; i = edge[i].next)
        {
            Edge &e = edge[i];
            int sum = dis[u] + e.w;
            if (sum < dis[e.to])
            {
                dis[e.to] = sum;
                pq.push(pii(dis[e.to], e.to));
            }
        }
    }
}

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    init();
    while (m--)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
    }
    dijkstra(1);
    for (int i = 1; i <= n; i++) printf("%d ", dis[i]);
    printf("%d\n", dis[n]);
    return 0;
}
```

Input

```
5 10
1 2 10
1 4 5
2 3 1
2 4 2
3 5 4
4 2 3
4 3 9
4 5 2
5 1 7
5 3 6
```

Output

```
0 8 9 5 7
```

3. SPFA

Description

SPFA 算法(Shortest Path Faster Algorithm)，时间复杂度 O(kE)。

输入：以邻接表存储的有向带权图 adj[]，起点 start。

输出：起点 start 到每个结点的最短距离 dis[]。

条件：允许有负边和负环，若存在负圈，函数返回 false。

图的定义和测试函数同 Dijkstra。

Code

```
int dis[N], cnt[N];
bool vis[N];
queue<int> q;
bool spfa(int start)
{
    memset(dis, 0x3f, sizeof(dis));
```

```
        dis[start] = 0;
        memset(vis, false, sizeof(vis));
        memset(cnt, 0, sizeof(cnt));
        while (!q.empty()) q.pop();
        q.push(start); vis[start] = true; cnt[start] = 1;
        while (!q.empty())
        {
            int u = q.front(); q.pop(); vis[u] = false;
            for (int i = adj[u]; i != -1; i = edge[i].next)
            {
                Edge &e = edge[i];
                int sum = dis[u] + e.w;
                if (sum < dis[e.to])
                {
                    dis[e.to] = sum;
                    if (!vis[e.to])
                    {
                       q.push(e.to);
                       vis[e.to] = true;
                       if (++cnt[u] > n) return false;
                    }
                }
            }
        }
        return true;
    }
```

4. Bellman-Ford

Description

Code

```
struct Edge
{
    int fr, to, w;
} e[M];
int n, m;

int dis[N];
void bellman(int start)
{
    memset(dis, 0x3f, sizeof(dis));
    dis[start] = 0;
    for (int i = 1; i < n; i++)
```

```
        for (int j = 1; j <= m; j++)
            dis[e[j].to] = min(dis[e[j].to], dis[e[j].fr] + e[j].w);
}
```

5. Floyd

Description

Code

```
void floyd()
{
    for (int i = 1; i <= n; i++) map[i][i] = 0;
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
}
```

6. Kruskal

Description

Kruscal 算法求最小生成树 MST，时间复杂度 O(ElogE)。

输入：以边集数组存储的带权无向图 e[]。

输出：最小生成树的边权和。

条件：连通图非连通图均可，若为非连通图，返回-1。

用并查集(Disjoint-set)判断结点间是否连通。

Code

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int N = 1e3 + 10;
const int M = 1e3 + 10;

struct Edge
{
    int u, v, w;
} e[M];
int n, m;
```

```
Disjoint-set...

bool cmp(Edge &e1, Edge &e2)
{
    return e1.w < e2.w;
}
int kruskal()
{
    sort(e + 1, e + m + 1, cmp);
    init();
    int ans = 0, cnt = 0;
    for (int i = 1; i <= m; i++)
        if (!same(e[i].u, e[i].v))
        {
            unite(e[i].u, e[i].v);
            ans += e[i].w;
            if (++cnt == n - 1) break;
        }
    if (cnt < n - 1) return -1;
    return ans;
}

int main()
{
    scanf("%d%d", &n, &m);
    init();
    for (int i = 1; i <= m; i++)
        scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].w);
    int ans = kruskal();
    printf("%d\n", ans);
}
```

Input

```
6 10
1 2 6
1 3 1
1 4 5
2 3 5
2 5 3
3 4 5
3 5 6
3 6 4
4 6 2
5 6 6
```

Output

```
15
```

7. Dinic

Description

Code

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int N = 1e3 + 10;
const int M = 1e3 + 10;

struct Edge
{
    int to, c, next;
} edge[2 * M];
int adj[N], no;

void init()
{
    memset(adj, -1, sizeof(adj));
    no = 0;
}

void add(int u, int v, int c)
{
    edge[no].to = v; edge[no].c = c;
    edge[no].next = adj[u];
    adj[u] = no++;
    edge[no].to = u; edge[no].c = 0;
    edge[no].next = adj[v];
    adj[v] = no++;
}

queue<int> q;
int level[N];
bool bfs(int s, int t)
{
    while (!q.empty()) q.pop();
    memset(level, -1, sizeof(level));
    level[s] = 0; q.push(s);
    while (!q.empty())
    {
        int u = q.front(); q.pop();
        for (int i = adj[u]; i != -1; i = edge[i].next)
        {
            Edge &e = edge[i];
            if (e.c && level[e.to] < 0)
            {
                level[e.to] = level[u] + 1;
                if (e.to == t) return true;
                q.push(e.to);
            }
        }
    }
    return false;
}

int cur[N];
int dfs(int u, int t, int flow)
{
    if (u == t) return flow;
    for (int &i = cur[u]; i != -1; i = edge[i].next)
    {
        Edge &e = edge[i];
        if (e.c && level[e.to] > level[u])
        {
            int f = dfs(e.to, t, min(flow, e.c));
            if (f)
            {
                e.c -= f;
                edge[i ^ 1].c += f;
                return f;
            }
        }
    }
    return 0;
}

int dinic(int s, int t)
{
    int flow = 0;
    while (bfs(s, t))
    {
        memcpy(cur, adj, sizeof(adj));
        int f;
        while (f = dfs(s, t, INF)) flow += f;
    }
    return flow;
}
```

8. MCMF

Description

Code

```
struct Node
{
    int u, v, flow, next, cost;
    Node(){};
    Node(int x, int y, int z, int w,int c) :
        u(x), v(y), next(z), flow(w),cost(c) {};
} p[M];
int adj[N], d[N], s, t, no, dis[N][N], vis[N], pre[N];

void add(int x, int y, int z,int c)
{
    p[no] = Node(x, y, adj[x], z, c);
    adj[x] = no++;
    p[no] = Node(y, x, adj[y], 0, -c);
    adj[y] = no++;
}

void init()
{
    memset(adj, -1, sizeof(adj));
    no = 0;
}

queue<int>q;
bool spfa()
{
    int i, x, y;
    while (!q.empty()) q.pop();
    memset(d, 0x3f, sizeof(d));
    memset(vis, false, sizeof(vis));
    memset(pre, -1, sizeof(pre));
```

```
        d[s] = 0; vis[s] = true; q.push(s);
        while(!q.empty())
        {
            x = q.front(); q.pop(); vis[x] = false;
            for(i = adj[x]; i != -1; i = p[i].next)
            {
                if(p[i].flow && d[y = p[i].v] > d[x] + p[i].cost)
                {
                    d[y] = d[x] + p[i].cost; pre[y] = i;
                    if(vis[y]) continue;
                    vis[y] = true;  q.push(y);
                }
            }
        }
        return d[t] != d[t + 1];
    }

    int mcmf()
    {
        int mincost = 0, maxflow = 0, minflow, i;
        while(spfa())
        {
            minflow =INF;
            for(i = pre[t]; i != -1; i = pre[p[i].u])
                minflow = min(minflow, p[i].flow);
            for(i = pre[t]; i != -1; i = pre[p[i].u])
            {
                p[i].flow -= minflow;
                p[i ^ 1].flow += minflow;
            }
            mincost += d[t] * minflow; maxflow += minflow;
        }
        return mincost;
    }
```

9. Hungary

Description

Code

```
    int left, right;
    bool g[N][N];
    int match[N];
    bool vis[N];
```

```
bool dfs(int u)
{
    for (int v = 1; v <= right; v++)
        if (g[u][v] && !vis[v])
        {
            vis[v] = true;
            if (match[v] < 0 || dfs(match[v]))
            {
                match[v] = u;
                return true;
            }
        }
    return false;
}
int hungary(int x, int y)
{
    left = x; right = y;
    int ans = 0;
    memset(match, -1, sizeof(match));
    for (int u = 1; u <= left; u++)
    {
        memset(vis, false, sizeof(vis));
        if (dfs(u)) ans++;
    }
    return ans;
}
```

10. Tarjan

Description

Code1 求连通分量。

Code2 求桥。

Code1

```
Graph...

int dfn[N], low[N], con[N];
int index, cnt;
stack<int> s;
bool instack[N];
void init_tarjan()
{
    memset(dfn, 0, sizeof(dfn));
    index = cnt = 0;
    while (!s.empty()) s.pop();
    memset(instack, false, sizeof(instack));
}
void tarjan(int u)
{
    dfn[u] = low[u] = ++index;
    s.push(u); instack[u] = true;
    for (int i = adj[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (instack[v])
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u])
    {
        cnt++;
        int v = -1;
        while (v != u)
        {
            v = s.top(); s.pop();
            instack[v] = false;
            con[v] = cnt;
        }
    }
}
```

Code2

```
Graph...

int dfn[N], low[N], con[N];
int index, cnt;
stack<int> s;
bool instack[N];
void init_tarjan()
{
    memset(dfn, 0, sizeof(dfn));
    index = cnt = 0;
    while (!s.empty()) s.pop();
```

```
        memset(instack, false, sizeof(instack));
    }
    void tarjan(int u, int from)
    {
        dfn[u] = low[u] = ++index;
        s.push(u); instack[u] = true;
        for (int i = adj[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if (!dfn[v])
            {
                tarjan(v, u);
                low[u] = min(low[u], low[v]);

                if (low[v] > dfn[u])
                {
                    edge[i].flag = true;
                    edge[i ^ 1].flag = true;
                    bridge++;
                }
            }
            else if (v != from)
            {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
```

Part II. Number Theory

1. GCD

Description

Code

```
    int gcd(int a, int b)
    {
        return b ? gcd(b, a % b) : a;
    }

    ll ext_gcd(ll a, ll b, ll &x, ll &y)
    {
        if (b == 0)
        {
            x = 1; y = 0;
            return a;
        }
        ll d = ext_gcd(b, a % b, y, x);
        y -= (a / b) * x;
        return d;
    }
```

2. Inverse

Description

Code

```
ll inverse(ll a, ll mod)
{
    ll x, y;
    ext_gcd(a, mod, x, y);
    x = (x % mod + mod) % mod;
    return x;
}
```

3. Fast Power

Description

Code

```
ll power(ll a, ll n, ll m)
{
    ll ans = 1;
    while (n)
    {
        if (n & 1) ans = ans * a % m;
        a = a * a % m;
        n >>= 1;
    }
    return ans;
}
```

4. Euler Function

Description

Code1：求欧拉函数值，时间复杂度 O(sqrt(n))。

Code2：线性筛欧拉函数，时间复杂度 O(n)。

Code1

```
int phi(int n)
{
    int ans = n;
    int s = sqrt(n);
    for (int i = 2; i <= s; i++)
        if (n % i == 0)
        {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
            s = sqrt(n);
        }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

Code2

```
int phi[N], prime[N];
void get_euler(int n)
{
    memset(phi, 0, sizeof(phi));
    phi[1] = 1;
    int res = 0;
    for (int i = 2; i <= n; i++)
    {
        if (!phi[i]) phi[i] = i - 1, prime[res++] = i;
        for (int j = 0; j < res && prime[j] * i <= n; j++)
        {
            if (i % prime[j]) phi[prime[j] * i] = phi[i] * (prime[j] - 1);
            else
            {
                phi[prime[j] * i] = phi[i] * prime[j];
                break;
            }
        }
    }
}
```

5. Prime

Description

Code1：判素数 O(sqrt(n))

Code2：素分解 O(sqrt(n))

Code3：素分解放到 map 中 O(sqrt(n))

Code3：线性筛素数 O(n)

Code5：阶乘素分解

Code1

```
bool prime(int n)
{
    if (n == 0 || n == 1) return false;
    int s = sqrt(n);
    for (int i = 2; i <= s; i++)
        if (n % i == 0) return false;
    return true;
}
```

Code2

```
int s = sqrt(n);
for (int i = 2; i <= s; i++)
    if (n % i == 0)
    {
        int r = 0;
        while (n % i == 0) n /= i, r++;
        printf("%d %d\n", i, r);
        s = sqrt(n);
    }
if (n > 1) printf("%d %d\n", n, 1);
```

Code3

```
map<int, int> factor;
map<int, int>::iterator iter;
void get_factor(int n)
{
    factor.clear();
    int s = sqrt(n);
    for (int i = 2; i <= s; i++)
        if (n % i == 0)
        {
            while (n % i == 0) n /= i, factor[i]++;
            s = sqrt(n);
        }
    if (n > 1) factor[n]++;
}
```

Code4

```
int mark[N], prime[N];
void get_prime(int n)
{
    memset(mark, 0, sizeof(mark));
    int res = 0;
    for (int i = 2; i <= n; i++)
    {
        if (!mark[i]) mark[i] = prime[res++] = i;
        for (int j = 0; j < res && prime[j] * i <= n; j++)
        {
            mark[i * prime[j]] = prime[j];
            if (i % prime[j] == 0) break;
        }
    }
}
```

Code5

```
map<int, int> factor;
map<int, int>::iterator iter;
void get_factor(int n)
{
    factor.clear();
    for (int i = 0; prime[i] <= n; i++)
        for (int j = prime[i]; j <= n; j *= prime[i])
            factor[prime[i]] += n / j;
}
```

6. Combination

Description

简单求组合数

Code

```
ll C(int n, int m)
{
    ll ans = 1;
    m = min(m, n - m);
    int k = n - m;
    for (int i = 1; i <= m; i++)
        ans = ans * (i + k) / i;
    return ans;
}
```

7. Game

Description

Code1：记忆话搜索求 sg 函数

Code2：打表求 sg 函数

Code1

```
int sg[N];
int dfs(int x)
{
    if (sg[x] != -1) return sg[x];
    bool vis[15];
    memset(vis, false, sizeof(vis));
    for (int i = 0; i <= k; i++)
    {
        if (i > x) break;
        vis[dfs(x - i)] = true;
    }
    int i = 0;
    while (vis[i]) i++;
    return sg[x] = i;
}
```

Code2

```
void get_sg()
{
    sg[0] = 0;
    for (int i = 1; i < H; i++)
    {
        memset(mex, 0, sizeof(mex));
        int j = 1;
        while (j <= k && i >= s[j])
        {
            mex[sg[i - s[j]]] = 1;
            j++;
        }
        j = 0;
        while (mex[j]) j++;
        sg[i] = j;
    }
}
```

Part III. Data Structure

1. Segment Tree

Description

Code

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cstring>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int N = 1e3 + 10;

struct Node
{
    int l, r;
    int sum;
    int lazy;
} tree[4 * N];
int fa[N], a[N];

inline int L(int i) { return i << 1; }
inline int R(int i) { return (i << 1) + 1; }
inline int P(int i) { return i >> 1; }

void build(int i, int left, int right)
{
    tree[i].l = left; tree[i].r = right;
    tree[i].lazy = 0;
    if (left == right)
    {
        tree[i].sum = a[left];
        fa[left] = i;
        return;
    }
    int mid = left + (right - left >> 1);
    build(L(i), left, mid);
    build(R(i), mid + 1, right);
    tree[i].sum = tree[L(i)].sum + tree[R(i)].sum;
}
```

```
void pushdown(int i)
{
    if (!tree[i].lazy) return;

    tree[L(i)].sum += (tree[L(i)].r - tree[L(i)].l + 1) * tree[i].lazy;
    tree[L(i)].lazy += tree[i].lazy;

    tree[R(i)].sum += (tree[R(i)].r - tree[R(i)].l + 1) * tree[i].lazy;
    tree[R(i)].lazy += tree[i].lazy;

    tree[i].lazy = 0;
}

void update(int i, int left, int right, int key)
{
    if (left <= tree[i].l && right >= tree[i].r)
    {
        tree[i].sum += (tree[i].r - tree[i].l + 1) * key;
        tree[i].lazy += key;
        return;
    }
    pushdown(i);
    int mid = tree[i].l + (tree[i].r - tree[i].l >> 1);
    if (left <= mid) update(L(i), left, right, key);
    if (right > mid) update(R(i), left, right, key);
    tree[i].sum = tree[L(i)].sum + tree[R(i)].sum;
}

void update(int i)
{
    if (i == 1) return;
    i = P(i);
    tree[i].sum = tree[L(i)].sum + tree[R(i)].sum;
    update(i);
}

int query(int i, int left, int right)
{
    if (left <= tree[i].l && right >= tree[i].r) return tree[i].sum;
    pushdown(i);
    int sum = 0;
    int mid = tree[i].l + (tree[i].r - tree[i].l >> 1);
    if (left <= mid) sum += query(L(i), left, right);
    if (right > mid) sum += query(R(i), left, right);
    return sum;
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", a + i);
    build(1, 1, n);
    int m;
    scanf("%d", &m);
    while (m--)
    {
        char s[3];
        scanf("%s", s);
        if (s[0] == 'u')
        {
            int l, r, a;
            scanf("%d%d%d", &l, &r, &a);
            update(1, l, r, a);
        }
        else if (s[0] == 'q')
        {
            int l, r;
            scanf("%d%d", &l, &r);
            int ans = query(1, l, r);
            printf("%d\n", ans);
        }
        else if (s[0] == 'i')
        {
            int id, key;
            scanf("%d%d", &id, &key);
            a[id] += key;
            tree[fa[id]].sum += key;
            update(fa[id]);
            printf("tree 3: %d\n", tree[3].sum);
        }
    }
    return 0;
}
```

2. ST

Description

Code

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int N = 1e3 + 10;

int a[N];
int dp[N][20], lg[N];
void ST(int n)
{
    lg[0] = -1;
    for (int i = 1; i <= n; i++)
    {
        lg[i] = ((i & (i - 1))) == 0 ? lg[i - 1] + 1 : lg[i - 1];
        dp[i][0] = a[i];
    }
    for (int j = 1; j <= lg[n]; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
int rmq(int left, int right)
{
    int k = lg[right - left + 1];
    return max(dp[left][k], dp[right - (1 << k) + 1][k]);
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", a + i);
    ST(n);
    int q;
    scanf("%d", &q);
    while (q--)
    {
        int a, b;
```

```
            scanf("%d%d", &a, &b);
            printf("%d ", rmq(a, b));
        }
        return 0;
    }
```

Input

```
    5
    2 3 0 1 4
    5
    1 5
    2 4
    1 3
    1 1
    4 4
```

Output

```
    4 3 3 2 1
```

3. Binary Indexed Tree

Description

Code

```
    int bit[N], n;
    inline int lowbit(int i) { return i & -i; }
    void init()
    {
        memset(bit, 0, sizeof(bit));
    }
    void add(int i, int key)
    {
        for (; i <= n; i += lowbit(i)) bit[i] += key;
    }
    int sum(int i)
    {
        int ans = 0;
        for (; i; i -= lowbit(i)) ans += bit[i];
        return ans;
    }
```

4. Disjoint-set

Description

Code

```
int fa[N], rank[N];
void init()
{
    memset(fa, -1, sizeof(fa));
    memset(rank, 0, sizeof(rank));
}
int find(int x)
{
    if (fa[x] == -1) return x;
    return fa[x] = find(fa[x]);
}
void unite(int x, int y)
{
    x = find(x); y = find(y);
    if (x == y) return;
    if (rank[x] < rank[y]) fa[x] = y;
    else
    {
        fa[y] = x;
        if (rank[x] == rank[y]) rank[x]++;
    }
}
bool same(int x, int y)
{
    return find(x) == find(y);
}
```

5. Heavy-Light Decomposition

Description

Code

```
struct Edge
{
    int to, next;
} edge[2 * N];
int head[N], no;
int siz[N], deep[N], fa[N], son[N];
int top[N], idx[N], pos[N], dfsOrder;
int n, val[N];
void init()
{
    memset(head, -1, sizeof(head));
    no = 0;
    dfsOrder = 0;
    deep[1] = fa[1] = siz[0] = 0;
}
void add(int u, int v)
{
    edge[no].to = v; edge[no].next = head[u]; head[u] = no++;
}
void dfs1(int x)
{
    siz[x] = 1;
    son[x] = 0;
    for (int i = head[x]; i != -1; i = edge[i].next)
    {
        int &u = edge[i].to;
        if (u != fa[x])
        {
            fa[u] = x;
            deep[u] = deep[x] + 1;
            dfs1(u);
            siz[x] += siz[u];
            if (siz[son[x]] < siz[u]) son[x] = u;
        }
    }
}
void dfs2(int u, int t)
{
    top[u] = t;
    idx[u] = ++dfsOrder;
    pos[dfsOrder] = u;
    if (son[u]) dfs2(son[u], t);
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int &v = edge[i].to;
        if (v != fa[u] && v != son[u]) dfs2(v, v);
    }
}


inline int L(int i) { return i << 1; }
inline int R(int i) { return (i << 1) + 1; }
```

```
struct Node
{
    int l, r, lazy, sum, max;
} tree[4 * N];
void pushup(int i)
{
    tree[i].max = max(tree[L(i)].max, tree[R(i)].max);
    tree[i].sum = tree[L(i)].sum + tree[R(i)].sum;
}
void build(int i, int left, int right)
{
    tree[i].l = left; tree[i].r = right;
    if (left == right)
    {
        tree[i].max = tree[i].sum = val[pos[left]];
        return;
    }
    int mid = left + (right - left >> 1);
    build(L(i), left, mid);
    build(R(i), mid + 1, right);
    pushup(i);
}

//单点更新
void update(int i, int x, int val)
{
    if (tree[i].l == x && tree[i].r == x)
    {
        tree[i].sum = val;
        return;
    }
    if (x <= tree[L(i)].r) update(L(i), x, val);
    else update(R(i), x, val);
    pushup(i);
}

//区间查询-求和
void pushdown(int i)
{
    if (!tree[i].lazy) return;
    tree[L(i)].sum += (tree[L(i)].r - tree[L(i)].l + 1) * tree[i].lazy;
    tree[L(i)].lazy += tree[i].lazy;
    tree[R(i)].sum += (tree[R(i)].r - tree[R(i)].l + 1) * tree[i].lazy;
    tree[R(i)].lazy += tree[i].lazy;
    tree[i].lazy = 0;
}
int query(int i, int left, int right)
{
    if (tree[i].l == left && tree[i].r == right) return tree[i].sum;
    pushdown(i);
    if (right <= tree[L(i)].r) return query(L(i), left, right);
    if (left >= tree[R(i)].l) return query(R(i), left, right);
    return query(L(i), left, tree[L(i)].r) + query(R(i), tree[R(i)].l, rig
ht);
}
int sum(int u, int v)
{
    int ans = 0;
    int topu = top[u], topv = top[v];
    while (topu != topv)
    {
        if (deep[topu] < deep[topv]) swap(topu, topv), swap(u, v);
        ans += query(1, idx[topu], idx[u]);
        u = fa[topu];
        topu = top[u];
    }
    if (deep[u] > deep[v]) swap(u, v);
    ans += query(1, idx[u], idx[v]);
    return ans;
}

//区间更新-增加
void pushdown(int i)
{
    if (!tree[i].lazy) return;
    tree[L(i)].sum += (tree[L(i)].r - tree[L(i)].l + 1) * tree[i].lazy;
    tree[L(i)].max += tree[i].lazy;
    tree[L(i)].lazy += tree[i].lazy;
    tree[R(i)].sum += (tree[R(i)].r - tree[R(i)].l + 1) * tree[i].lazy;
    tree[R(i)].max += tree[i].lazy;
    tree[R(i)].lazy += tree[i].lazy;
    tree[i].lazy = 0;
}
void update(int i, int left, int right, int key)
{
    if (left <= tree[i].l && right >= tree[i].r)
    {
        tree[i].sum += (tree[i].r - tree[i].l + 1) * key;
        tree[i].max += key;
```

```
            tree[i].lazy += key;
            return;
        }
        pushdown(i);
        if (left <= tree[L(i)].r) update(L(i), left, right, key);
        if (right >= tree[R(i)].l) update(R(i), left, right, key);
        pushup(i);
    }
    void change(int u, int v, int z)
    {
        int top1 = top[u], top2 = top[v];
        while (top1 != top2)
        {
            if (deep[top1] < deep[top2])
            {
                swap(top1, top2);
                swap(u, v);
            }
            update(1, id[top1], id[u], z);
            u = fa[top1];
            top1 = top[u];
        }
        if (deep[u] > deep[v]) swap(u, v);
        update(1, id[u], id[v], z);
    }

    //单点查询
    int query(int i, int x)
    {
        if (tree[i].l == x && tree[i].r == x) return tree[i].sum;
        pushdown(i);
        if (x <= tree[L(i)].r) return query(L(i), x);
        else return query(R(i), x);
    }

    //区间查询-找最大值
    void pushdown(int i)
    {
        if (!tree[i].lazy) return;
        tree[L(i)].max += tree[i].lazy;
        tree[L(i)].lazy += tree[i].lazy;
        tree[R(i)].max += tree[i].lazy;
        tree[R(i)].lazy += tree[i].lazy;
        tree[i].lazy = 0;
```

```
}
int query(int i, int left, int right)
{
    if (left == tree[i].l && right == tree[i].r) return tree[i].max;
    pushdown(i);
    if (right <= tree[L(i)].r) return query(L(i), left, right);
    if (left >= tree[R(i)].l) return query(R(i), left, right);
    return max(query(L(i), left, tree[L(i)]), query(R(i), tree[R(i)].l, ri
ght));
}
int qmax(int u, int v)
{
    int ans = 0;
    int topu = top[u], topv = top[v];
    while (topu != topv)
    {
        if (deep[topu] < deep[topv]) swap(topu, topv), swap(u, v);
        ans = max(ans, query(1, idx[topu], idx[u]);
        u = fa[topu];
        topu = top[u];
    }
    if (deep[u] > deep[v]) swap(u, v);
    ans = max(ans, query(1, idx[u], idx[v]));
    return ans;
}

//边权转点权
struct
{
    int u, v, w;
} e[N];

for (int i = 1; i < n; i++)
{
    if (deep[e[i].u] > deep[e[i].v]) swap(e[i].u, e[i].v);
    val[e[i].v] = e[i].w;
}

//区间查询-边权转点权后求和
void pushdown(int i)
{
    if (!tree[i].lazy) return;
    tree[L(i)].sum += (tree[L(i)].r - tree[L(i)].l + 1) * tree[i].lazy;
    tree[L(i)].lazy += tree[i].lazy;
    tree[R(i)].sum += (tree[R(i)].r - tree[R(i)].l + 1) * tree[i].lazy;
    tree[R(i)].lazy += tree[i].lazy;
    tree[i].lazy = 0;
}
int query(int i, int left, int right)
{
    if (tree[i].l == left && tree[i].r == right) return tree[i].sum;
    pushdown(i);
    if (right <= tree[L(i)].r) return query(L(i), left, right);
    if (left >= tree[R(i)].l) return query(R(i), left, right);
    return query(L(i), left, tree[L(i)].r) + query(R(i), tree[R(i)].l, rig
ht);
}
int sum(int u, int v)
{
    int ans = 0;
    int topu = top[u], topv = top[v];
    while (topu != topv)
    {
        if (deep[topu] < deep[topv]) swap(topu, topv), swap(u, v);
        ans += query(1, idx[topu], idx[u]);
        u = fa[topu];
        topu = top[u];
    }
    if (u == v) return ans;
    if (deep[u] > deep[v]) swap(u, v);
    ans += query(1, idx[son[u]], idx[v]);
    return ans;
}

//区间查询-边权转点权后找最大值
void pushdown(int i)
{
    if (!tree[i].lazy) return;
    tree[L(i)].max += tree[i].lazy;
    tree[L(i)].lazy += tree[i].lazy;
    tree[R(i)].max += tree[i].lazy;
    tree[R(i)].lazy += tree[i].lazy;
    tree[i].lazy = 0;
}
int query(int i, int left, int right)
{
    if (left == tree[i].l && right == tree[i].r) return tree[i].max;
    pushdown(i);
    if (right <= tree[L(i)].r) return query(L(i), left, right);
    if (left >= tree[R(i)].l) return query(R(i), left, right);
    return max(query(L(i), left, tree[L(i)]), query(R(i), tree[R(i)].l, ri
ght));
}
int qmax(int u, int v)
{
    int ans = 0;
    int topu = top[u], topv = top[v];
    while (topu != topv)
    {
        if (deep[topu] < deep[topv]) swap(topu, topv), swap(u, v);
        ans = max(ans, query(1, idx[topu], idx[u]);
        u = fa[topu];
        topu = top[u];
    }
    if (u == v) return ans;
    if (deep[u] > deep[v]) swap(u, v);
    ans = max(ans, query(1, idx[son[u]], idx[v]));
    return ans;
}

//LCA
int lca(int u, int v)
{
    int topu = top[u], topv = top[v];
    while (topu != topv)
    {
        if (deep[topu] < deep[topv]) swap(topu, topv), swap(u, v);
        u = fa[topu];
        topu = top[u];
    }
    return deep[u] < deep[v] ? u : v;
}
```

6. LCA

Description

Code

```
//RMQ O(nlogn)预处理 O(1)查询 ***
int dis[N], first[N];
int pos[2 * N], deep[2 * N], dfsOrder;
void dfs(int u, int father, int w)
{
    dfsOrder++;
    pos[dfsOrder] = u; first[u] = dfsOrder; deep[dfsOrder] = deep[father]
+ 1;
    dis[u] = dis[father] + w;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        Edge &e = edge[i];
        if (e.to != father)
        {
            dfs(e.to, u, e.w);
            dfsOrder++;
            pos[dfsOrder] = u;
            deep[dfsOrder] = deep[father] + 1;
        }
    }
}
int dp[2 * N][20], lg[2 * N];
void ST(int n)
{
    lg[0] = -1;
    for (int i = 1; i <= n; i++)
    {
        lg[i] = ((i & (i - 1))) == 0 ? lg[i - 1] + 1 : lg[i - 1];
        dp[i][0] = i;
    }
    for (int j = 1; j <= lg[n]; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
        {
            int x = dp[i][j - 1], y = dp[i + (1 << (j - 1))][j - 1];
            dp[i][j] = deep[x] < deep[y] ? x : y;
        }
}
int rmq(int left, int right)
{
    int k = lg[right - left + 1];
    int x = dp[left][k], y = dp[right - (1 << k) + 1][k];
    return deep[x] < deep[y] ? x : y;
}
int lca(int u, int v)
{
    int x = first[u], y = first[v];
    if (x > y) swap(x, y);
    return pos[rmq(x, y)];
}
```

```
//在线倍增法 O(nlogn)预处理 O(logn)查询
int p[N][20];
int deep[N], dis[N], fa[N];
void dfs(int u, int father, int w)
{
    fa[u] = father;
    deep[u] = deep[father] + 1;
    dis[u] = dis[father] + w;
    p[u][0] = father;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        Edge &e = edge[i];
        if (e.to != father) dfs(e.to, u, e.w);
    }
}
void init(int n)
{
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i <= n; i++)
            if (p[i][j - 1] != -1)
                p[i][j] = p[p[i][j - 1]][j - 1];
}
int lca(int u, int v)
{
    if (deep[u] < deep[v]) swap(u, v);
    int i = 0;
    while ((1 << i) <= deep[u]) i++;
    i--;
    for (int j = i; j >= 0; j--)
        if (deep[u] - (1 << j) >= deep[v])
            u = p[u][j];
    if (u == v) return u;
    for (int j = i; j >= 0; j--)
        if (p[u][j] != -1 && p[u][j] != p[v][j])
        {
            u = p[u][j];
            v = p[v][j];
        }
    return p[u][0];
}

//暴力
int deep[N], dis[N], fa[N];
void dfs(int u, int father, int w)
```

```
{
    fa[u] = father;
    deep[u] = deep[father] + 1;
    dis[u] = dis[father] + w;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        Edge &e = edge[i];
        if (e.to != father) dfs(e.to, u, e.w);
    }
}
int lca(int u, int v)
{
    if (deep[u] < deep[v]) swap(u, v);
    while (deep[u] > deep[v]) u = fa[u];
    while (u != v) u = fa[u], v = fa[v];
    return u;
}
```

Part IV. String

1. KMP

Description

Code

```
int next[N];
bool kmp(char text[], char pattern[])
{
    int lt = strlen(text);
    int lp = strlen(pattern);
    for (int i = 0, j = -1; i <= lp; i++, j++)
    {
        next[i] = j;
        while (~j && pattern[i] != pattern[j]) j = next[j];
    }
    int ans = 0;
    for (int i = 0, j = 0; i <= lt; i++, j++)
    {
        if (j == lp) return true;
        while (~j && text[i] != pattern[j]) j = next[j];
    }
    return false;
}
```

2. Manacher

Descripton

Code

```
const int N = 1e3 + 10;
char ma[N * 2];
int mp[N * 2];
void manacher(char s[])
{
    int len = strlen(len);
    int l = 0;
    ma[l++] = '$';
    ma[l++] = '#';
    for (int i = 0; i < len; i++)
    {
        ma[l++] = s[i];
        ma[l++] = '#';
    }
    ma[l] = 0;
    int mx = 0, id = 0;
    for (int i = 0; i < l; i++)
    {
        mp[i] = mx > i ? min(mp[2 * id - i], mx - i) : 1;
        while (ma[i + mp[i]] == ma[i - mp[i]])
            mp[i]++;
        if (i + mp[i] > mx)
        {
            mx = i + mp[i];
            id = i;
        }
    }
}
/*
* abaaba
* i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13
* ma[i]: $ # a # b # a # a $ b # a #
* mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1
*/
```

3. Suffix Array

Description

后缀数组（Suffix Array）

倍增算法 O(nlogn)

待排序数组长度为 n,放在 0~n-1 中，在最后面补一个 0

da(str,sa,rank,height,n,str 中最大值);

```
例如：n = 8;
num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };
       注意 num 最后一位为 0，其他大于 0
rank[]= { 4, 6, 8, 1, 2, 3, 5, 7, 0 };
       rank[0~n-1]为有效值，rank[n]必定为 0 无效值
sa[]  = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };
       sa[1~n]为有效值，sa[0]必定为 n 是无效值
height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };
       height[2~n]为有效值
```

Code

```
const int N = 2e4 + 10;
int t1[N], t2[N], c[N]; //求 SA 数组需要的中间变量，不需要赋值
//待排序的字符串放在 s 数组中，从 s[0]到 s[n-1],长度为 n,且最大值小于 m,
//除 s[n-1]外的所有 s[i]都大于 0，r[n-1]=0
//函数结束以后结果放在 sa 数组中
bool cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void da(int str[], int sa[], int rank[], int height[], int n, int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果 s 的最大值很大，可改为快速排序
    for (i = 0;i < m;i++)c[i] = 0;
    for (i = 0;i < n;i++)c[x[i] = str[i]]++;
    for (i = 1;i < m;i++)c[i] += c[i - 1];
    for (i = n - 1;i >= 0;i--)sa[--c[x[i]]] = i;
    for (j = 1;j <= n; j <<= 1)
    {
        p = 0;
        //直接利用 sa 数组排序第二关键字
        for (i = n - j; i < n; i++)y[p++] = i;
        //后面的 j 个数第二关键字为空的最小
        for (i = 0; i < n; i++)if (sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组y保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for (i = 0; i < m; i++)c[i] = 0;
        for (i = 0; i < n; i++)c[x[y[i]]]++;
        for (i = 1; i < m;i++)c[i] += c[i - 1];
        for (i = n - 1; i >= 0;i--)sa[--c[x[y[i]]]] = y[i];
        //根据sa和x数组计算新的x数组
        swap(x, y);
        p = 1; x[sa[0]] = 0;
        for (i = 1;i < n;i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
        if (p >= n)break;
        m = p; //下次基数排序的最大值
    }
    int k = 0;
    n--;
    for (i = 0;i <= n;i++)rank[sa[i]] = i;
    for (i = 0;i < n;i++)
    {
        if (k)k--;
        j = sa[rank[i] - 1];
        while (str[i + k] == str[j + k])k++;
        height[rank[i]] = k;
    }
}
int rank[N], height[N];
int RMQ[N];
int mm[N];
int best[20][N];
void initRMQ(int n)
{
    mm[0] = -1;
    for (int i = 1;i <= n;i++)
        mm[i] = ((i&(i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
    for (int i = 1;i <= n;i++)best[0][i] = i;
    for (int i = 1;i <= mm[n];i++)
        for (int j = 1;j + (1 << i) - 1 <= n;j++)
        {
            int a = best[i - 1][j];
            int b = best[i - 1][j + (1 << (i - 1))];
            if (RMQ[a]<RMQ[b])best[i][j] = a;
            else best[i][j] = b;
        }
```

```
}
int askRMQ(int a, int b)
{
    int t;
    t = mm[b - a + 1];
    b -= (1 << t) - 1;
    a = best[t][a];b = best[t][b];
    return RMQ[a]<RMQ[b] ? a : b;
}
int lcp(int a, int b)
{
    a = rank[a];b = rank[b];
    if (a>b)swap(a, b);
    return height[askRMQ(a + 1, b)];
}
char str[N];
int r[N];
int sa[N];
int main()
{
    while (scanf("%s", str) == 1)
    {
        int len = strlen(str);
        int n = 2 * len + 1;
        for (int i = 0;i < len;i++)r[i] = str[i];
        for (int i = 0;i < len;i++)r[len + 1 + i] = str[len - 1 - i];
        r[len] = 1;
        r[n] = 0;
        da(r, sa, rank, height, n, 128);
        for (int i = 1;i <= n;i++)RMQ[i] = height[i];
        initRMQ(n);
        int ans = 0, st;
        int tmp;
        for (int i = 0;i<len;i++)
        {
            tmp = lcp(i, n - i);//偶对称
            if (2 * tmp>ans)
            {
                ans = 2 * tmp;
                st = i - tmp;
            }
            tmp = lcp(i, n - i - 1);//奇数对称
            if (2 * tmp - 1>ans)
            {
                ans = 2 * tmp - 1;
                st = i - tmp + 1;
            }
        }
        str[st + ans] = 0;
        printf("%s\n", str + st);
    }
    return 0;
}
```

4. Aho-Corasick Algorithm

Description

Code

```
const int Z = 26;
int trie[M][Z], no;
int fail[M], ed[M];
void init()
{
    memset(trie[0], 0, sizeof(trie[0]));
    no = 1;
    memset(ed, 0, sizeof(ed));
}
void insert(char s[])
{
    int len = strlen(s);
    int p = 0;
    for (int i = 0; i < len; i++)
    {
        int c = s[i] - 'a';
        if (!trie[p][c])
        {
            memset(trie[no], 0, sizeof(trie[no]));
            trie[p][c] = no++;
        }
        p = trie[p][c];
    }
    ed[p]++;
}
queue<int> q;
void build()
{
    while (!q.empty()) q.pop();
```

```
        for (int i = 0; i < Z; i++)
            if (!trie[0][i]) trie[0][i] = 0;
            else
            {
                fail[trie[0][i]] = 0;
                q.push(trie[0][i]);
            }
        while (!q.empty())
        {
            int p = q.front(); q.pop();
            for (int i = 0; i < Z; i++)
                if (!trie[p][i]) trie[p][i] = trie[fail[p]][i];
                else
                {
                    fail[trie[p][i]] = trie[fail[p]][i];
                    q.push(trie[p][i]);
                }
        }
    }
    int query(char s[])
    {
        int ans = 0;
        int p = 0;
        int len = strlen(s);
        for (int i = 0; i < len; i++)
        {
            p = trie[p][s[i] - 'a'];
            int t = p;
            while (t)
            {
                ans += ed[t];
                ed[t] = 0;
                t = fail[t];
            }
        }
        return ans;
    }
```

Part V. Computational Geometry

1. Vector

Descripton

点，向量定义
点 - 点 = 向量
向量 + 向量 = 向量
数乘向量*
向量点乘 dot()
向量模 norm()
向量叉乘 cross()
三点叉乘 cross()
取符号 sign()
判两点相等==
两点距离 dist()

Code

```
typedef struct Point
{
    double x, y;
    Point() {}
    Point(double x, double y): x(x), y(y) {}
} Vector;

Vector operator-(Point A, Point B)
{
    return Vector(A.x - B.x, A.y - B.y);
}

Vector operator+(Vector A, Vector B)
{
    return Vector(A.x + B.x, A.y + B.y);
}

Vector operator*(double k, Vector A)
{
    return Vector(k * A.x, k * A.y);
}

double dot(Vector A, Vector B)
{
    return A.x * B.x + A.y * B.y;
}
double norm(Vector A)
{
    return sqrt(dot(A, A));
}

double cross(Vector A, Vector B)
{
    return A.x * B.y - B.x * A.y;
}

double cross(Point A ,Point B, Point C)
{
    return cross(B - A, C - A);
}

int sign(double x)
{
    if(abs(x) < eps) return 0;
    return x > 0 ? 1 : -1;
}
bool operator==(Point A, Point B)
{
    return sign(A.x - B.x) == 0 && sign(A.y - B.y) == 0;
}
```

2. Graham

Description

Code1：求凸包 Graham 算法，时间复杂度 O(nlogn)。

Code2：二分判断点在凸包内，时间复杂度 O(logn)。

Code1

```
int n, m;
Point p[N], c[N];
Point referPoint;
bool cmp1(Point p1, Point p2)
{
    if (sign(p1.y - p2.y) != 0) return p1.y < p2.y;
    return p1.x < p2.x;
}
bool cmp2(Point p1, Point p2)
{
    Vector A = p1 - referPoint, B = p2 - referPoint;
    int sgn = sign(cross(A, B));
    if (sgn != 0) return sgn > 0;
    return norm(A) < norm(B);
}
void graham()
{
    sort(p + 1, p + n + 1, cmp1);
    referPoint = p[1];
    sort(p + 2, p + n + 1, cmp2);
    m = 0;
    c[++m] = p[1];
    c[++m] = p[2];
    for (int i = 3; i <= n; i++)
    {
        while (cross(p[i] - c[m], c[m] - c[m - 1]) > 0)
            m--;
        c[++m] = p[i];
    }
}
```

Code2

```
bool in_convex(Point P)
{
    if (sign(cross(P, c[1], c[2])) > 0) return false;
    if (sign(cross(P, c[1], c[m])) < 0) return false;

    int low = 2, high = m;
    while (high - low > 1)
    {
        int mid = low + high >> 1;
        int sgn = sign(cross(P, c[1], c[mid]));
        if (sgn == 0)
            return sign(dist(P, c[1]) - dist(c[mid], c[1])) <= 0 ? true :
false;
        if (sgn < 0) low = mid;
        else high = mid;
    }

    return sign(cross(P, c[low], c[high])) <= 0 ? true : false;
}
```

Part VI. Others

1. Monotone Stack

Description

线性时间内得到数组 a[]中每个元素左边第一个比它小的元素。

Code

```
int n, a[N], l[N];
int st[N], t;
void monotone()
{
    t = 0;
    for (int i = 1; i <= n; i++)
    {
        while (t > 0 && a[st[t - 1]] >= a[i]) t--;
        l[i] = t == 0 ? 0 : st[t - 1];
    }
}
```

2. Two Pointers

Description

Code

```
int ans = INF;
int l = 1, r = 1, sum = 0;
while (true)
{
    while (r <= n && sum < k) sum += a[r++];
    if (sum < k) break;
    ans = min(ans, r - 1);
    sum -= a[l++];
}
```

3. Btoi

Description

二进制字符串转整形数据。

Code

```
int btoi(char s[])
```

```
{
    int len = strlen(s);
    int ans = 0;
    for (int i = 0; i < len; i++)
    {
        ans <<= 1;
        if (s[i] == '1') ans += 1;
    }
    return ans;
}
```