ACM ICPC Reference

University of Brasilia

June 23, 2018

Contents

.vimrc

```
set ai ts=4 sw=4 sta nu rnu sc stl+=%F
syntax on
```

.bashrc

```
alias cmp='g++ -Wall -Wshadow -Wconversion -fsanitize=address -std
```

Ad hoc

Small to Large

```cpp
void cnt_sz(int u, int p = -1){
    sz[u] = 1;

    for(int v : g[u]) if(v != p)
        cnt_sz(v, u), sz[u] += sz[v];
}


void add(int u, int p, int big = -1){
    // Update info about this vx in global answer

    for(int v : g[u]) if(v != p && v != big)
        add(v, u);
}


void dfs(int u, int p, int keep){

    int big = -1, mmx = -1;

    for(int v : g[u]) if(v != p && sz[v] > mmx)
        mmx = sz[v], big = v;

    for(int v : g[u]) if(v != p && v != big)
        dfs(v, u, 0);

    if(big != -1) dfs(big, u, 1);

    add(u, p, big);

    for(auto x : q[u]){
        // answer all queries for this vx
    }

    if(!keep){
        // Remove data from this subtree
    }
}
```

Convex Hull Trick

```cpp
const ll is_query = -(1LL<<62);
struct Line{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const{
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line>{ // will maintain upper hull for maximum
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
```

```cpp
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y-
    }
    void insert_line(ll m, ll b){
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*next(y);
        if(bad(y)) { erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

Estrutura de Dados

Merge Sort Tree

```cpp
struct MergeTree{
    int n;
    vector<vector<int>> st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){
            st[p].push_back(v[L]);
            return;
        }
        int mid = (L+R)/2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p].resize(R-L+1);
        merge(st[2*p].begin(), st[2*p].end(),
            st[2*p+1].begin(), st[2*p+1].end(),
            st[p].begin());
    }

    int query(int p, int L, int R, int i, int j, int x) const{
        if(L > j || R < i) return 0;
        if(L >= i && R <= j){
            int id = lower_bound(st[p].begin(), st[p].end(), x)
            return int(st[p].size()) - id;
        }
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j, x) +
            query(2*p+1, mid+1, R, i, j, x);
    }

public:
    MergeTree(int sz, const int v[]): n(sz), st(4*sz){
        build(1, 1, n, v);
    }

    //number of elements >= x on segment [i, j]
    int query(int i, int j, int x) const{
        if(i > j) swap(i, j);
        return query(1, 1, n, i, j, x);
    }
};
```

Paradigmas

FFT

```cpp
// typedef complex<double> base;
```

```
struct base{
    double r, i;
    base(double r = 0, double i = 0) : r(r), i(i) {}
    base operator*(const base &o){
        return base(r*o.r - i*o.i, r*o.i + o.r*i);
    }

    base& operator*=(const base &o){
        double newr = r*o.r - i*o.i, newi = r*o.i + o.r*i;
        r = newr, i = newi;
        return *this;
    }

    base& operator+=(const base &o){
        r += o.r, i += o.i;
        return *this;
    }

    base& operator/=(const double &o){
        r /= o, i /= o;
        return *this;
    }

    base& operator-=(const base &o){
        r -= o.r, i -= o.i;
        return *this;
    }

    base operator+(const base &o){
        return base(r + o.r, i + o.i);
    }

    base operator-(const base &o){
        return base(r - o.r, i - o.i);
    }
};

double PI = acos(-1);

void fft(vector<base> &a, bool inv){
    int n = (int)a.size();

    for(int i = 1, j = 0; i < n; i++){
        int bit = n >> 1;
        for(; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if(i < j)
            swap(a[i], a[j]);
    }

    for(int len = 2; len <= n; len <<= 1) {
        double ang = 2*PI/len * (inv ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += len){
            base w(1);
            for (int j = 0; j < len/2; j++){
                base u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv)
        for(int i = 0; i < n; i++)
            a[i] /= n;
```

```
}
void multiply(const vector<int> &a, const vector<int> &b, vect
    vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end())
    size_t n = 1;
    while(n < a.size()) n <<= 1;
    while(n < b.size()) n <<= 1;
    n <<= 1;
    fa.resize(n), fb.resize(n);

    fft(fa, false), fft (fb, false);
    for(size_t i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    res.resize (n);
    for(size_t i = 0; i < n; ++i)
        res[i] = int(fa[i].real() + 0.5);
}
```

NTT

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
    int n = (int) a.size();

    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
        for (; j>=bit; bit>>=1)
            j -= bit;
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }

    for (int len=2; len<=n; len<<=1) {
        int wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = int (wlen * 1ll * wlen % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j], v = int (a[i+j+len/2] * 1ll * w %
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                w = int (w * 1ll * wlen % mod);
            }
        }
    }
    if (invert) {
        int nrev = reverse (n, mod);
        for (int i=0; i<n; ++i)
            a[i] = int (a[i] * 1ll * nrev % mod);
    }
}
```

Matemática

Euclides Extendido

```
// a*x + b*y = gcd(a, b), <gcd, <x, y>>
typedef tuple<int, int, int> tiii;
tiii euclidesExt(int a, int b) {
    if(b == 0) return tiii(a, 1, 0);
    tiii ret = euclidesExt(b, a % b);
```

```
    int q, w, e;
    tie(q, w, e) = ret;

    get<1>(ret) = e;
    get<2>(ret) = w - e*(a / b);

    return ret;
}

// a*b = 1 (mod c) <-> a*b + c*k = 1
int invmult(int a, int b){
    return (get<1>(euclidesExt(a, b)) + b) % b;
}
```

Grafos

Dinic

```
const int N = 205;
const int E = 30006;
vector<int> g[N];

int ne;
struct Edge{
    int from, to;
    ll cap;
} edge[E];

int lvl[N], vis[N], pass, source, target, px[N];

ll run(int s, ll minE){
    if(s == target) return minE;

    ll ans = 0;

    for(; px[s] < g[s].size(); px[s]++){
        int e = g[s][ px[s] ];
        auto &v = edge[e], &rev = edge[e^1];
        if(lvl[v.to] != lvl[s]+1 || !v.cap) continue;
        ll tmp = run(v.to, min(minE, v.cap));
        v.cap -= tmp, rev.cap += tmp;
        ans += tmp, minE -= tmp;
        if(minE == 0) break;
    }
    return ans;
}

int bfs(){

    queue<int> q;
    q.push(source);
    lvl[source] = 1;
    vis[source] = ++pass;

    while(!q.empty()){
        int u = q.front(); q.pop();
        px[u] = 0;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0 || vis[v.to] == pass) continue;
            vis[v.to] = pass;
            lvl[v.to] = lvl[u]+1;
            q.push(v.to);
        }
    }
    return vis[target] == pass;
}
```

```
}

ll flow(){
    ll ans = 0;
    while(bfs()) ans += run(source, oo);
    return ans;
}

void addEdge(int u, int v, ll c){
    edge[ne] = {u, v, c};
    g[u].pb(ne++);
}
```

Min Cost Max Flow

```
const ll oo = 1e18;
const int N = 505;
const int E = 30006;

vector<int> g[N];

int ne;

struct Edge{
    int from, to;
    ll cap, cost;
} edge[E];

int lvl[N], vis[N], pass, source, target, p[N], px[N];

ll d[N];

ll back(int s, ll minE){
    if(s == source) return minE;

    int e = p[s];

    ll f = back(edge[e].from, min(minE, edge[e].cap));
    edge[e].cap -= f;
    edge[e^1].cap += f;
    return f;
}

int dijkstra(){
    forn(i, N) d[i] = oo;

    priority_queue<pair<ll, int> > q;

    d[source] = 0;

    q.emplace(0, source);

    while(!q.empty()){
        ll dis = -q.top().ff;
        int u = q.top().ss; q.pop();

        if(dis > d[u]) continue;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0) continue;
            if(d[u] + v.cost < d[v.to]){
                d[v.to] = d[u] + v.cost;
                p[v.to] = e;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
```

```cpp
    return d[target] != oo;
}

pair<ll, ll> mincost(){
    ll ans = 0, mf = 0;
    while(dijkstra()){
        ll f = back(target, oo);
        mf += f;
        ans += f * d[target];
    }
    return {mf, ans};
}

void addEdge(int u, int v, ll c, ll cost){
    edge[ne] = {u, v, c, cost};
    g[u].pb(ne++);
}
```

Junior e Falta de Ideias

```cpp
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
    subtree[v] = 1;
    for(pair<int,int> x : G[v])
        if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.ff,v);
    return subtree[v];
}

int centroid(int v, int p){
    for(pair<int,int> x : G[v]){
        if(x.ff == p || vis[x.ff]) continue;
        if(subtree[x.ff]*2 > treesize) return centroid(x.ff,v);
    }
    return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
    ans = min(ans, dist[k-d_atual] + custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            procurar_ans(x.ff,v,d_atual+1,custo+x.ss);
    }
}

void atualiza_distancia(int v, int p, int d_atual, ll custo){
    dist[d_atual] = min(dist[d_atual], custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            atualiza_distancia(x.ff,v,d_atual+1,custo+x.ss);
    }
}
```

```cpp
void decomp(int v, int p){
    treesize = dfs(v,v);
    // if(treesize < k) return;
    int cent = centroid(v,v);
    vis[cent] = 1;

    for(int i = 1; i <= treesize; i++)
        dist[i] = 1e18;

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff]){
            procurar_ans(x.ff,cent,1,x.ss);
            atualiza_distancia(x.ff,cent,1,x.ss);
        }
    }

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff])
            decomp(x.ff, cent);
    }
}

int main(){
    int n,i,a,b;

    scanf("%d%d", &n,&k);
    for(i = 2; i <= n; i++){
        scanf("%d%d", &a,&b);
        G[i].push_back(mp(a,b));
        G[a].push_back(mp(i,b));
    }
    ans = 1e18;
    decomp(1,-1);

    printf("%lld\n", ans == 1e18 ? -1 : ans);

    return 0;
}
```

Kosaraju

```cpp
vector<int> g[N], gt[N], S;

int vis[N], cor[N], tempo = 1;

void dfs(int u){
    vis[u] = 1;
    for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}
int e;
void dfst(int u){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v);
}

int main(){

    for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);

    e = 0;
    reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u])
        e++, dfst(u);

    return 0;
}
```

Tarjan

```
void dfs(int u, int p = -1){
    low[u] = num[u] = ++t;
    for(int v : g[u]){
        if(!num[v]){
            dfs(v, u);
            if(low[v] >= num[u]) u PONTO DE ARTICULACAO;
            if(low[v] > num[u]){
                ARESTA u->v PONTE;
                low[u] = min(low[u], low[v]);
            }
        }
        else if(v != p) low[u] = min(low[u], num[v]);
    }
}
```

Strings

Aho Corasick

```
void init_aho(){
    queue<int> q;

    q.push(0);

    while(!q.empty()){
        int t = q.front(); q.pop();

        for(int i = 0; i < 52; i++) if(trie[t][i]){
            int x = trie[t][i];
            Q.push(x);

            if(t){
                fn[x] = fn[t];

                while(fn[x] && trie[fn[x]][i] == 0) fn[x] = fn[fn[x]];
                if(trie[fn[x]][i]) fn[x] = trie[fn[x]][i];
            }
        }
    }
}
```

Suffix Array

```
char s[N];
int n, sa[N], tsa[N], lcp[N], r[N], nr[N], c[N];

void sort(int k, int mx){
    mx++;
    memset(c, 0, sizeof(int) * mx);
    for(int i = 0; i < n; i++) c[i + k < n ? r[i+k]+1 : 1]++;
    partial_sum(c, c+mx, c);
    int t;
    for(int i = 0; i < n; i++)
        t = sa[i]+k < n ? r[ sa[i]+k ] : 0,
        tsa[ c[t]++ ] = sa[i];
    memcpy(sa, tsa, sizeof(int) * n);
}

void build_sa(){

    for(int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];

    int t = 300, a, b;
    for(int sz = 1; sz < n; sz *= 2){
        sort(sz, t), sort(0, t);
        t = nr[ sa[0] ] = 0;
```

```
        for(int i = 1; i < n; i++){
            a = sa[i]+sz < n ? r[ sa[i]+sz ] : -1;
            b = sa[i-1]+sz < n ? r[ sa[i-1]+sz ] : -1;
            nr[ sa[i] ] = r[ sa[i] ] == r[ sa[i-1] ] && a == b ?
        }
        if(t == n-1) break;
        memcpy(r, nr, sizeof(int) * n);
    }
}

void build_lcp(){ // lcp[i] = lcp(s[:i], s[:i+1])
    int k = 0;
    for(int i = 0; i < n; i++) r[ sa[i] ] = i;

    for(int i = 0; i < n; i++){
        if(r[i] == n-1) k = 0;
        else{
            int j = sa[r[i]+1];
            while(i+k < n && j+k < n && s[i+k] == s[j+k]) k++;
        }
        lcp[r[i]] = k;
        if(k) k--;
    }
}
```

Z Algorithm

```
vector<int> z_algo(const string &s) {
    int n = s.size(), L = 0, R = 0;
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= R) z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ]) z[i]++;
        if(i+z[i]-1 > R) L = i, R = i + z[i] - 1;
    }
    return z;
}
```

Prefix function/KMP

```
vector<int> pf(const string &s){
    int n = s.size();
    vector<int> b(n+1);
    b[0] = -1;
    int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}

void kmp(const string &t, const string &p){
    vector<int> b = pf(p);
    int n = t.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
        j++;
        if(j == m){
            //patern found
            j = b[j];
        }
    }
}
```

Min rotation

```
int min_rotation(int *s, int N) {
  REP(i, N) s[N+i] = s[i];
```

```
  int a = 0;
  REP(b, N) REP(i, N) {
    if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1); break;}
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

All palindrome

```
void manacher(char *s, int N, int *rad) {
  static char t[2*MAX];
  int m = 2*N - 1;

  REP(i, m) t[i] = -1;
  REP(i, N) t[2*i] = s[i];

  int x = 0;
  FOR(i, 1, m) {
    int &r = rad[i] = 0;
    if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
    while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r+1]) ++r;
    if (i+r >= x+rad[x]) x = i;
  }

  REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
  REP(i, m) rad[i] /= 2;
}
```

Palindromic Tree

```
const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root with len 0
int suff; // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while(true){
        curlen = tree[cur].len;
        if (pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;

    if (tree[num].len == 1){
```

```
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true){
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if(pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[pos]){
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
    }

    return 0;
}
```

Geometria

Nearest Points

```
struct pt {
    int x, y, id;
};
inline bool cmp_x (const pt & a, const pt & b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

inline bool cmp_y (const pt & a, const pt & b) {
    return a.y < b.y;
}

pt a[MAXN];
double mindist;
int ansa, ansb;

inline void upd_ans (const pt & a, const pt & b) {
    double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.
    if (dist < mindist)
        mindist = dist, ansa = a.id, ansb = b.id;
}

void rec (int l, int r) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans (a[i], a[j]);
```

```
    sort (a+l, a+r+1, &cmp_y);                          int n;
    return;                                             vector < vector<double> > a (n, vector<double> (n));
}
                                                        double det = 1;
int m = (l + r) >> 1;                                   for (int i=0; i<n; ++i) {
int midx = a[m].x;                                          int k = i;
rec (l, m), rec (m+1, r);                                   for (int j=i+1; j<n; ++j)
static pt t[MAXN];                                              if (abs (a[j][i]) > abs (a[k][i]))
merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);                        k = j;
copy (t, t+r-l+1, a+l);                                     if (abs (a[k][i]) < EPS) {
                                                               det = 0;
int tsz = 0;                                                    break;
for (int i=l; i<=r; ++i)                                    }
    if (abs (a[i].x - midx) < mindist) {                    swap (a[i], a[k]);
        for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist;if (i!= k)
            upd_ans (a[i], t[j]);                               det = -det;
        t[tsz++] = a[i];                                    det *= a[i][i];
    }                                                      for (int j=i+1; j<n; ++j)
}                                                              a[i][j] /= a[i][i];
                                                           for (int j=0; j<n; ++j)
sort (a, a+n, &cmp_x);                                          if (j != i && abs (a[j][i]) > EPS)
mindist = 1E20;                                                     for (int k=i+1; k<n; ++k)
rec (0, n-1);                                                           a[j][k] -= a[i][k] * a[j][i];
                                                        }
```

Determinante

```
const double EPS = 1E-9;                                 cout << det;
```