VNU HCM - University of Science

HCMUS - Lattis

November 30, 2016

Contents

1 BST variants and Search methods

1.1 Treap

```
struct item {
    int key, prior;
    item * l, * r;
    item() { }
    item(int key, int prior) : key(key), prior(prior), l(NULL), r(NULL
        ) {}
};

typedef item* pitem;

void split(pitem t, int key, pitem & l, pitem & r) {
    if (!t)
        l = r = NULL;
    else if (key < t->key)
        split(t->l, key, l, t->l), r = t;
    else
        split(t->r, key, t->r, r), l = t;
}

void insert(pitem & t, pitem it) {
    if (!t) t = it;
    else if (it->prior > t->prior)
        split(t, it->key, it->l, it->r), t = it;
    else
        insert(it->key < t->key ? t->l : t->r, it);
}

void merge(pitem & t, pitem l, pitem r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge(l->r, l->r, r), t = l;
    else
        merge(r->l, l, r->l), t = r;
}

void erase(pitem & t, int key) {
    if (t->key == key)
        merge(t, t->l, t->r);
    else
        erase(key<t->key ? t->l : t->r,key);
}

pitem union(pitem l, pitem r) {
    if (!l || !r) return l ? l : r;
    if (l->prior < r->prior) swap (l, r);
    pitem lt, rt;
    split(r, l->key, lt, rt);
    l->l = union (l->l, lt);
    l->r = union (l->r, rt);
    return l;
}
```

1.2 Splay tree

```
#include <stdlib.h>
#include <assert.h>

struct node {
   node *ll, *rr, *pp;
   int size, id;
   int value, max_value, lazy_add;
   node(int _id, int x, node* nil){
      ll=rr=pp=nil;
      id=_id; size=1;
      value=x; max_value=x; lazy_add=0;
   }
};

bool maximize(int &a, int b){ if (a<b) a=b; else return false; return
    true; }

node *nil, *root;

int llindex(node* a){
   if (a==nil) return 0;
   else return a->id-a->ll->size;
}

int rrindex(node* a){
   if (a==nil) return 0;
   else return a->id+a->rr->size;
}


// Assertion

int a[12309];

node* left_most(node* a){
   if (a->ll==nil) return a;
   else return left_most(a->ll);
}

node* right_most(node* a){
   if (a->rr==nil) return a;
   else return right_most(a->rr);
}

void hard_add(int ll, int rr, int value){
   return ;
   for (int i=ll; i<=rr; i++) a[i]+=value;
}

ostream& operator << (ostream& cout, node* x){
   if (x==nil) return cout;
   return cout << "(" << x->ll << " " << x->value << " " << x->rr << "
       )";
}

void ensure(node* x){
   return ;
   if (x==nil) return;
// cout << x << endl;
   assert(x->value==a[x->id]);
   assert(x->pp==nil || x==x->pp->ll || x==x->pp->rr);
   assert(x->max_value == *max_element(a+llindex(x), a+rrindex(x)+1));
   assert(x->size == right_most(x)->id - left_most(x)->id + 1);
   if (x->ll->lazy_add==0) ensure(x->ll);
   if (x->rr->lazy_add==0) ensure(x->rr);
}

//

node* update_lazy(node* a){
   if (a==nil) return nil;
   if (a->lazy_add != 0){
      a->value += a->lazy_add;
      a->max_value += a->lazy_add;
      if (a->ll!=nil) a->ll->lazy_add += a->lazy_add;
      if (a->rr!=nil) a->rr->lazy_add += a->lazy_add;
      a->lazy_add=0;
   }
   return a;
}

node* update_size(node* a){
   if (a==nil) return nil;
   a->size = 1 + a->ll->size + a->rr->size;
}

node* update_data(node* a){
   if (a==nil) return nil;
   a->max_value = a->value;
   if (a->ll!=nil) { update_lazy(a->ll); maximize(a->max_value, a->ll
       ->max_value); }
   if (a->rr!=nil) { update_lazy(a->rr); maximize(a->max_value, a->rr
       ->max_value); }
// cout << a << endl;
// assert(a->max_value == *max_element(::a+llindex(a), ::a+rrindex(a)
    +1));
}

node* lljoin(node* a, node* b){
   if (b==nil) { a->pp=nil; root=a; return root; }
   else { a->pp=b; b->ll=a; return b; }
}
node* rrjoin(node* a, node* b){
   if (b==nil) { a->pp=nil; root=a; return root; }
   else { a->pp=b; b->rr=a; return b; }
}
```

```
node* create(int ll, int rr){
   if (ll>rr) return nil;
   int mm=(ll+rr)/2;
   node* u=new node(mm, 0, nil);
   lljoin(create(ll, mm-1), u);
   rrjoin(create(mm+1, rr), u);
   update_size(u); update_data(u);
   return u;
}


node* zig(node* a){
   if (a==nil || a==root) return a;
// cout << "zig " << a << endl;
   update_lazy(a);
   node* b=a->pp;
   node* c=b->pp;
   if (a==b->ll) { lljoin(a->rr, b); rrjoin(b, a); }
   else { rrjoin(a->ll, b); lljoin(b, a); }
   if (b==c->ll) lljoin(a, c); else rrjoin(a, c);
   update_size(b); update_data(b);
   update_size(a); update_data(a);
// cout << "zig last " << a << endl;
   ensure(a);
   return a;
}

node* splay(node* a){
   if (a==nil) return nil;
   while (a->pp!=nil){
      node* b=a->pp;
      node* c=b->pp;
      if (c!=nil){
         if (a==b->ll xor b==c->ll) zig(a);
         else zig(b);
      }
      zig(a);
   }
   update_lazy(a);
   ensure(a);
   return a;
}

node* at(int pos, node* root){
   node* a=root;
   while (a->id!=pos){
      if (a==nil) return nil;
      update_lazy(a);
      if (pos<a->id) a=a->ll;
      else a=a->rr;
   }
   update_lazy(a);
   ensure(a);
   return splay(a);
}

node *part1, *part2, *part3;

node* split(node* &a, int pos){
   node* b=at(pos, a);
   if (b==nil) return nil;
   a=b->ll;
   update_lazy(b); update_lazy(a);
   b->ll=nil;
   a->pp=nil;
   update_size(b); update_data(b);
   ensure(b);
   return b;
}

node* at(int ll, int rr){
   part3 = split(root, rr+1);
   part2 = split(root, ll);
   part1 = root;
   update_lazy(part2);
// cout << part1 << " | " << part2 << " | " << part3 << endl;
   ensure(part2);
   return part2;
}

node* reconnect(){
   update_lazy(part2);
// cout << part1 << " | " << part2 << " | " << part3 << endl;
   part1 = rrjoin(part2, at(rrindex(part1), part1));
   part3 = lljoin(part1, at(llindex(part3), part3));
   update_size(part1); update_data(part1);
   update_size(part3); update_data(part3);
   root=part3;
   ensure(root);
   return root;
}


int n, m;

main(){
   int i, p, q, w, option;
   nil=new node(0, 0, nil); nil->size=0;
   scanf("%d%d", &n, &m);
   srand(n+1000+m);
   root=create(1, n);
// cout << root << endl;
   ensure(root);

   for (i=1; i<=m; i++){
      //option = rand() % 2;
      //p=rand()%n+1; q=rand()%n+1; w = rand()%100;
      //if (p>q) swap(p,q);
      //printf(" %d %d %d %d\n", option, p, q, w);
      scanf("%d", &option);
      if (option==0) scanf("%d%d%d", &p, &q, &w);
      else scanf("%d%d", &p, &q);

      if (option==0) { at(p,q)->lazy_add+=w; hard_add(p,q,w); }
      else printf("%d\n", at(p,q)->max_value);
      reconnect();
//    cout << root << endl;
   }
}
```

1.3 Ternary search

```
LL=minX, RR=maxX;
ML=(LL+LL+RR)/3, MR=(LL+RR+RR)/3;

while (LL!=ML) and (ML!=MR) and (MR!=RR) do
    if f(ML)>f(MR) then LL=ML;
    else RR=MR:
    ML=(LL+LL+RR)/3, MR=(LL+RR+RR)/3;

print (LL+RR)/2;
```

2 Strings

2.1 Suffix Array

```
char str1[N]; //input
int rank1[N], pos1[N]; //outputint cnt1[N], next1[N]; //internal
bool bh1[N], b2h1[N];
bool smaller_first_char(int a, int b){
    return str1[a] < str1[b];
}
void suffixSort(int n){
    for (int i=0; i<n; ++i)
        pos1[i] = i;
    sort(pos1, pos1 + n, smaller_first_char);
    for (int i=0; i<n; ++i){
        bh1[i] = i == 0 || str1[pos1[i]] != str1[pos1[i-1]];
        b2h1[i] = false;
    }
    for (int h = 1; h < n; h <<= 1){
        int buckets = 0;
        for (int i=0, j; i < n; i = j){
            j = i + 1;
            while (j < n && !bh1[j]) j++;
            next1[i] = j;buckets++;
        }
        if (buckets == n) break;
        for (int i = 0; i < n; i = next1[i]){
            cnt1[i] = 0;
            for (int j = i; j < next1[i]; ++j)
```

```
                rank1[pos1[j]] = i;
        }
        cnt1[rank1[n - h]]++; b2h1[rank1[n - h]] = true;
        for (int i = 0; i < n; i = next1[i]){
            for (int j = i; j < next1[i]; ++j){
                int s = pos1[j] - h;
                if (s >= 0){
                    int head = rank1[s];
                    rank1[s] = head + cnt1[head]++;
                    b2h1[rank1[s]] = true;
                }
            }
            for (int j = i; j < next1[i]; ++j){
            int s = pos1[j] - h;
            if (s >= 0 && b2h1[rank1[s]]){
                for (int k = rank1[s]+1; !bh1[k] && b2h1[k];
                k++) b2h1[k] = false;
                }
            }
        }
        for (int i=0; i<n; ++i){
                pos1[rank1[i]] = i;bh1[i] |= b2h1[i];
        }
    }
    for (int i=0; i<n; ++i){rank1[pos1[i]] = i;}}

int height[N];

void getHeight(int n){
    for (int i=0; i<n; ++i) rank1[pos1[i]] = i;
    height[0] = 0;
    for (int i=0, h=0; i<n; ++i){
        if (rank1[i] > 0){
            int j = pos1[rank1[i]-1];
            while (i + h < n && j + h < n && str1[i+h] == str1[j+h]) h
                ++;
            height[rank1[i]] = h;
            if (h > 0) h--;
        }
    }
}
```

2.2 KMP

```
int main(){
    scanf("%s%s", &s, &x);
    int m = strlen(s), n = strlen(x);
    memset(ne, -1, sizeof(ne));
    int jj = -1, j = 0; ne[0] = -1;
    while(j < n){
        while(jj >= 0 && x[jj] != x[j])jj = ne[jj];
        j++; jj++;
        if(j >= n || x[j] != x[jj])ne[j] = jj;
        else ne[j] = ne[jj];
    }
    int i = 0; j = 0;
    do{
        while(j >= 0 && x[j] != s[i]) j = ne[j];
        j++; i++;
        if(j >= n){
            printf("%d ", i - j + 1);
            j = ne[j];
        }
    }while(i < m);
}
```

2.3 Manacher

```
void manacher(const string &s){
    int len = s.length();
    int c = 0, r = 0, m = 0, n = 0;
    p[0] = 0;
    for(int i = 1; i < len; i++){
        if(i > r){
            p[i] = 0;
            m = i - 1;
            n = i + 1;
        } else {
            int i2 = (c << 1) - i;
            if(p[i2] < r - i){
                p[i] = p[i2];
                m = -1;
            } else{
                p[i] = r - i;
                m = i - p[i] - 1;
                n = r + 1;
            }
        }
        while(m >= 0 && n < len && s[m] == s[n]){
            p[i]++; m--; n++;
        }
        if(i + p[i] > r){
            c = i;r = i + p[i];
        }
    }
}
```

2.4 Z Function

```
char s[N];
char st[N];
int Z[N];
//Memset Z by 0 before using ZFunction method
void ZFunction(char * s, const int &n, int* Z){
    Z[0]=n;
    int l=1; int r=1;
    for (int i=2; i<=n; i++){
        if (i<=r)
            Z[i - 1]=min(Z[i-1],r-i+1);
        while ((i+Z[i - 1]<=n) && (s[i+Z[i - 1]-1]==s[1+Z[i - 1]-1])){
            Z[i - 1]++;
        }
        if (i+Z[i - 1]-1>r){
            l=i; r=i+Z[i - 1]-1;
        }
    }
}
```

2.5 Ahococrasick

```
const int MAXS = 500;
// Maximum number of characters in input alphabet
const int MAXC = 26;
// OUTPUT FUNCTION IS IMPLEMENTED USING out[]
// Bit i in this mask is one if the word with index i
// appears when the machine enters this state.
int out[MAXS];
// FAILURE FUNCTION IS IMPLEMENTED USING f[]
int f[MAXS];
// GOTO FUNCTION (OR TRIE) IS IMPLEMENTED USING g[][]
int g[MAXS][MAXC];
// Builds the string matching machine.
// arr -   array of words. The index of each keyword is important:
//         ”out[state] & (1 << i)” is > 0 if we just found word[i]
//         in the text.
// Returns the number of states that the built machine has.
// States are numbered 0 up to the return value - 1, inclusive.
int buildMatchingMachine(string arr[], int k){
    memset(out, 0, sizeof out);
    memset(g, -1, sizeof g);
    int states = 1;
    for (int i = 0; i < k; ++i){
        const string &word = arr[i];
        int currentState = 0;
        for (int j = 0; j < word.size(); ++j){
            int ch = word[j] - 'a';
            if (g[currentState][ch] == -1)
                g[currentState][ch] = states++;

            currentState = g[currentState][ch];
        }
        out[currentState] |= (1 << i);
    }
    for (int ch = 0; ch < MAXC; ++ch)
        if (g[0][ch] == -1)
            g[0][ch] = 0;
    memset(f, -1, sizeof f);
    queue<int> q;
    for (int ch = 0; ch < MAXC; ++ch){
        if (g[0][ch] != 0){
            f[g[0][ch]] = 0;
            q.push(g[0][ch]);
        }
    }
    while (q.size()){
        int state = q.front();
        q.pop();
        for (int ch = 0; ch <= MAXC; ++ch){
            if (g[state][ch] != -1){
                int failure = f[state];
                while (g[failure][ch] == -1)
                      failure = f[failure];
                failure = g[failure][ch];
                f[g[state][ch]] = failure;
                out[g[state][ch]] |= out[failure];
                q.push(g[state][ch]);
            }
        }
    }

    return states;
}
int findNextState(int currentState, char nextInput)
{
    int answer = currentState;
    int ch = nextInput - 'a';
    while (g[answer][ch] == -1)
        answer = f[answer];

    return g[answer][ch];
}
void searchWords(string arr[], int k, string text){
    buildMatchingMachine(arr, k);
    int currentState = 0;
    for (int i = 0; i < text.size(); ++i){
        currentState = findNextState(currentState, text[i]);
        if (out[currentState] == 0)
             continue;
        for (int j = 0; j < k; ++j){
            if (out[currentState] & (1 << j)){
                cout << ”Word ” << arr[j] << ” appears from ”
                     << i - arr[j].size() + 1 << ” to ” << i << endl;
            }
        }
    }
}
int main(){
    string arr[] = {”he”, ”she”, ”hers”, ”his”};
    string text = ”ahishers”;
    int k = sizeof(arr)/sizeof(arr[0]);
    searchWords(arr, k, text);
    return 0;
}
```

3 LCA

3.1 Using dynamic tree

```
int main() {
    d[1] = 0;
    maxd = 0;
    dfs(1);
    logg = int(log(maxd) / log(2)) + 1;
    for(int j = 1; j <= logg; j++)
        for(int i = 2; i <= n; i++){
            B[i][j] = B[B[i][j - 1]][j - 1];
            Bmax[i][j] = max(Bmax[i][j - 1], Bmax[B[i][j - 1]][j - 1])
                ;
            Bmin[i][j] = min(Bmin[i][j - 1], Bmin[B[i][j - 1]][j - 1])
                ;
        }
}
//after init, for each query, invoke lca(x, y)
void dfs(int x){
    v[x] = true;
    maxd = max(maxd, d[x]);
    for(int i = star[x]; i < star[x + 1]; i++)
        if(v[ke[i]] == false){
            B[ke[i]][0] = x;
            Bmax[ke[i]][0] = gt[i];
            Bmin[ke[i]][0] = gt[i];
            d[ke[i]] = d[x] + 1;
            dfs(ke[i]);
        }
}

const int getbit(const int &a, const int &b){
    return (a >> b) & 1;
}

int lca(int x, int y){
    int r;
    if(d[x] < d[y]) swap(x, y);
    if(d[x] > d[y]){
        int t = d[x] - d[y];
        int i = 0;
        while(t > 0){
            if(t & 1 == 1){
                x = B[x][i];
                if(d[x] == d[y])
                break;
            }
            i++;  t >>= 1;
        }
    }
    if(x == y)
        return x;
    if(x != y){
        int logm = int(log(d[x]) / log(2)) + 1;
        for(int i = logm; i >= 0; i--)
            if(B[x][i] != B[y][i]){
                x = B[x][i];
                y = B[y][i];
            }
            else  r = B[x][i];
    }
    return r;
}
```

3.2 Using RMQ

```
void dfs(int x){
    bd[x] = sld;b[sld++] = x;v[x] = true;
    for(int i = star[x]; i < star[x + 1]; i++)
        if(v[ke[i]] == false){
            d[ke[i]] = d[x] + 1; dfs(ke[i]); b[sld++] = x;
        }
}
int getRMQ(int i, int j){
    i = bd[i]; j = bd[j];
    if (i > j) swap(i, j);
    int k = int(log(j - i + 1) * 1.0/log(2));
    if(d[rmq[i][k]] < d[rmq[j - (1 << k) + 1][k]])
        return rmq[i][k];
    return rmq[j - (1 << k) + 1][k];}
    //init code,
    for(int i = 0; i < sld; i++)
        rmq[i][0] = b[i];
    for(int j = 1; 1 << j <= sld; j++)
        for(int i = 0; i + (1 << j) - 1 < sld; i++){
            rmq[i][j] = rmq[i][j - 1];
            if(d[rmq[i][j]] > d[rmq[i + (1 << (j - 1))][j - 1]])
                rmq[i][j] = rmq[i + (1 << (j - 1))][j - 1];
        }
//dfs and getRMQ(x, y) for each query(x, y)
```

4 Flows and matchings

4.1 Dinic - Max Flow

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

using namespace std;

const int maxN = 5003;
const int maxM = 30004;

typedef long long LL;

struct TAdjNode {
 int v, c, f, link;
 TAdjNode() {}
 TAdjNode(int _v, int _c, int _link): v(_v), c(_c), f(0), link(_link)
     {}
};

int head[maxN], lev[maxN], marked[maxN], trace[maxN], ptr[maxN];
TAdjNode Adj[2*maxM];
int n, m, s, t;
int Q[maxN], top, bot;
LL flowVal;

bool buildLevelGraph() {
 bot = top = 0;
 memset(lev, 255, sizeof(lev));
 lev[Q[top++] = s] = 1;
 while (top > bot) {
  int u = Q[bot++];
  for (int i = head[u]; i >= 0; i = Adj[i].link) {
   int v = Adj[i].v;
   if (lev[v] == -1 && Adj[i].c > Adj[i].f) {
    lev[v] = lev[u]+1;
    if (v == t) return true;
    Q[top++] = v;
   }
  }
 }
 return false;
}

int constructBlockingFlow(int u, int flow) {
 if (!flow) return 0;
 if (u == t) return flow;
 for (int i = ptr[u]; ptr[u] >= 0; i = ptr[u] = Adj[ptr[u]].link) {
  int v = Adj[i].v;
  if (lev[v] != lev[u]+1) continue;
  int incF = constructBlockingFlow(v, min(flow, Adj[i].c-Adj[i].f));
  if (incF) {
   Adj[i].f += incF;
   Adj[i^1].f -= incF;
   return incF;
  }
 }
 return 0;
}

int main() {
 scanf(”%d %d”, &n, &m);
 memset(head, 255, sizeof(head));
 for (int i = 0; i < m; ++i) {
  int u, v, c;
  scanf(”%d %d %d”, &u, &v, &c);
  Adj[2*i] = TAdjNode(v, c, head[u]);
  head[u] = 2*i;
  Adj[2*i+1] = TAdjNode(u, c, head[v]);
  head[v] = 2*i+1;
 }
 s = 1, t = n;
 flowVal = 0;
 while (buildLevelGraph()) {
  for (int u = 1; u <= n; ++u) ptr[u] = head[u];
  while (int incF = constructBlockingFlow(s, (int)1E9+1))
   flowVal += incF;
 }
 printf(”%lld\n”, flowVal);
 return 0;
}
```

4.2 Fast Matching - Unweighted

```
const int maxNM = (int)5E4+4;
const int oo = (int)1E9+1;

int n, m, p;
vector<int> Adj[maxNM];
int pairU[maxNM], pairV[maxNM], dist[maxNM];

bool BFS() {
    queue<int> Q;
    for (int u = 1; u <= n; ++u)
        if (pairU[u] == 0)
            dist[u] = 0, Q.push(u);
        else dist[u] = oo;
    dist[0] = oo;

    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();

        if (dist[u] < dist[0])
            for (int i = 0, sz = Adj[u].size(); i < sz; ++i) {
                int v = Adj[u][i];
                if (dist[pairV[v]] == oo)
                    dist[pairV[v]] = dist[u]+1, Q.push(pairV[v]);
            }
    }
    return dist[0] != oo;
}

bool DFS(int u) {
    if (u != 0) {
        for (int i = 0, sz = Adj[u].size(); i < sz; ++i) {
            int v = Adj[u][i];
            if (dist[pairV[v]] == dist[u]+1)
                if (DFS(pairV[v])) {
                    pairV[v] = u, pairU[u] = v;
                    return true;
                }
        }
        dist[u] = oo;
        return false;
    }
    return true;
}


int main() {
    scanf("%d %d %d", &n, &m, &p);
    for (int i = 0; i < p; ++i) {
        int u, v;
        scanf("%d %d", &u, &v);
        Adj[u].push_back(v);
    }
    memset(pairU, 0, sizeof(pairU));
    memset(pairV, 0, sizeof(pairV));

    int result = 0;
    while (BFS()) for (int u = 1; u <= n; ++u)
        if (pairU[u] == 0 && DFS(u))
            ++result;
    printf("%d\n", result);
    return 0;
}
```

4.3 Minimum Cut

```
// Minimum cut between every pair of vertices (Stoer Wagner)
pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();
    VI used(N), cut, best_cut;
    int best_weight = -1;
    for (int phase = N-1; phase >= 0; phase--) {
        VI w = weights[0];
        VI added = used;
        int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last;
            last = -1;
            for (int j = 1; j < N; j++)
                if (!added[j] && (last == -1 || w[j] > w[last]))
                    last = j;
            if (i == phase-1) {
                for (int j = 0; j < N; j++) weights[prev][j] +=
                    weights[last][j];
                for (int j = 0; j < N; j++) weights[j][prev] = weights
                    [prev][j];
                used[last] = true;
                cut.push_back(last);
                if (best_weight == -1 || w[last] < best_weight) {
                    best_cut = cut;
                    best_weight = w[last];
                }
            }
            else {
                for (int j = 0; j < N; j++) w[j] += weights[last][j];
                added[last] = true;
            }
        }
    }
    return make_pair(best_weight, best_cut);
}
```

4.4 Maximum Flow with Minimum Cost

```
/*
* mincostflow implementation. Vertex indices from 0
* Usage:
* for i = 1..N: int v = mcf.addV();
* for i = 1..E: MinCostFlow<int,int>::Edge* e = mcf.addEdge(u, v, flow
    , cost);
* mcf.minCostFlow --> return pair<flow, cost>
* DANGEROUS!!!!!!! If need to find flow through each edge, remember
    that there can
* be flow through both (u, v) and (v, u)
*/
// Slow version (Ford Bellman)
// Can work with negative edges. (not loop!)
// Must be careful when used with double.
#define _MAX_COST INT_MAX
#define _MAX_FLOW INT_MAX
template<class Flow = int, class Cost = int>
struct MinCostFlow {
struct Edge {
    int t;
    Flow f;
    Cost c;
    Edge*next, *rev;
    Edge(int _t, Flow _f, Cost _c, Edge*_next) :
    t(_t), f(_f), c(_c), next(_next) {}
};

vector<Edge*> E;
int addV() {
    E.push_back((Edge*) 0);
    return E.size() - 1;
}

Edge* makeEdge(int s, int t, Flow f, Cost c) {
    return E[s] = new Edge(t, f, c, E[s]);
}

Edge* addEdge(int s, int t, Flow f, Cost c) {
    Edge*e1 = makeEdge(s, t, f, c), *e2 = makeEdge(t, s, 0, -c);
    e1->rev = e2, e2->rev = e1;
    return e1;
}
pair<Flow, Cost> minCostFlow(int vs, int vt) {
    int n = E.size();
    Flow flow = 0;
    Cost cost = 0;
    const Cost MAX_COST = _MAX_COST;
    const Flow MAX_FLOW = _MAX_FLOW;
    for (;;) {
        vector<Cost> dist(n, MAX_COST);
        vector<Flow> am(n, 0);
        vector<Edge*> prev(n);
        vector<bool> inQ(n, false);
        queue<int> que;
        dist[vs] = 0;
        am[vs] = MAX_FLOW;
        que.push(vs);
        inQ[vs] = true;
        while (!que.empty()) {
            int u = que.front();
            Cost c = dist[u];
            que.pop();
            inQ[u] = false;
            for (Edge*e = E[u]; e; e = e->next)
                if (e->f > 0) {
                    Cost nc = c + e->c;
                    if (nc < dist[e->t]) {
                        dist[e->t] = nc;
                        prev[e->t] = e;
                        am[e->t] = min(am[u], e->f);
                        if (!inQ[e->t]) {
                            que.push(e->t);
                            inQ[e->t] = true;
                        }
                    }
                }
        }
        if (dist[vt] == MAX_COST) // careful with double
            break;
        Flow by = am[vt];
        int u = vt;
        flow += by;
        cost += by * dist[vt];
        while (u != vs) {
            Edge*e = prev[u];
            e->f -= by;
            e->rev->f += by;
            u = e->rev->t;
        }
    }
    return make_pair(flow, cost);
}
};
```

5 Data structures

5.1 Heap

```
void upd(int v){
    int child = vtri[v];
    if(child == 0) child = ++nheap;
    int parent = child >> 1;
    while(parent > 0 && d[heap[parent]] > d[v]){
        heap[child] = heap[parent];
        vtri[heap[child]] = child;
        child = parent;
        parent = child >> 1;
    }
    heap[child] = v;
    vtri[v] = child;
}
int pop(){
    int r, c, v;
    int res = heap[1];
    r = 1; v = heap[nheap];
    nheap--;
    while(r << 1 <= nheap){
        c = r << 1;
        if(c < nheap && d[heap[c + 1]] < d[heap[c]])c++;
        if(d[v] <= d[heap[c]]) break;
        heap[r] = heap[c];
        vtri[heap[r]] = r;
        r = c;
    }
    heap[r] = v;
    vtri[v] = r;
    return res;
}
```

5.2 Persistent Segment Tree - COT

```
const int maxn = 200100;

int key;
int a[maxn], bb[maxn];
int bd[maxn];
int sld;
int b[maxn];
bool v[maxn];
int cha[maxn];
int star[maxn], a1[maxn], a2[maxn], ke[2 * maxn], d[maxn];
int c[maxn];
int rmq[maxn][19];
int n, q;
int cs[maxn];

bool cmp(int x, int y){
    return a[x] < a[y];
}

long long int read_int(){
 char r;
 bool start=false,neg=false;
 long long int ret=0;
 while(true){
  r=getchar();
  if((r-'0'<0 || r-'0'>9) && r!='-' && !start){
   continue;
  }
  if((r-'0'<0 || r-'0'>9) && r!='-' && start){
   break;
  }
  if(start)ret*=10;
  start=true;
  if(r=='-')neg=true;
  else ret+=r-'0';
 }
 if(!neg)
  return ret;
 else
  return -ret;
}

struct ITNode{
    int cc;
    ITNode * left;
    ITNode * right;
    ITNode(){}
    ITNode(int x, ITNode * l, ITNode * r){
        cc = x;
        left = l;
        right = r;
    }
    ITNode *insertNode(int l, int r, int w);
} ;

ITNode *root[maxn];

ITNode *null = new ITNode(0, NULL, NULL);

ITNode *ITNode::insertNode(int l, int r, int w){
    if(l <= w && w <= r){
        if(l == r)
            return new ITNode(cc + 1, null, null);
        int mid = (l + r) >> 1;
        return new ITNode(cc + 1, left->insertNode(l, mid, w), right->
            insertNode(mid + 1, r, w));
    }
    return this;
}


void dfs(int x, int pre){
    bd[x] = sld;
    b[sld++] = x;
    v[x] = true;
    cha[x] = pre;
    if(pre == -1)
        root[x] = null->insertNode(1, key, bb[x]);
    else
        root[x] = root[pre]->insertNode(1, key, bb[x]);
    for(int i = star[x + 1] - 1; i >= star[x]; i--)
        if(v[ke[i]] == false){
            d[ke[i]] = d[x] + 1;
            dfs(ke[i], x);
            b[sld++] = x;
        }
}

int getLCA(int i, int j){
    i = bd[i]; j = bd[j];
    if(i > j){
        int temp = i;
        i = j;
        j = temp;
    }
    int k = int(log(j - i + 1) * 1.0/log(2));
    if(d[rmq[i][k]] < d[rmq[j - (1 << k) + 1][k]])
        return rmq[i][k];
    return rmq[j - (1 << k) + 1][k];
}

void initLCA(){
    dfs(0, -1);
    for(int i = 0; i < sld; i++)
        rmq[i][0] = b[i];
    for(int j = 1; 1 << j <= sld; j++)
        for(int i = 0; i + (1 << j) - 1 < sld; i++){
            rmq[i][j] = rmq[i][j - 1];
            if(d[rmq[i][j]] > d[rmq[i + (1 << (j - 1))][j - 1]])
                rmq[i][j] = rmq[i + (1 << (j - 1))][j - 1];
        }
}

int getQuery(ITNode *a, ITNode *b, ITNode *c, ITNode *d, int l, int r,
     int k){
    if(l == r)
        return l;
    int cc = a->left->cc + b->left->cc - c->left->cc - d->left->cc;
    int mid = (l + r) >> 1;
    if(cc >= k)
        return getQuery(a->left, b->left, c->left, d->left, l, mid, k)
            ;
    return getQuery(a->right, b->right, c->right, d->right, mid + 1, r
        , k - cc);
}
```

```
int main(){
    n = read_int();
    q = read_int();
    for(int i = 0; i < n; i++){
        scanf("%d", &a[i]);
        cs[i] = i;
    }
    null->left = null->right = null;
    sort(cs, cs + n, cmp);
    key = 0;
    for(int i = 0; i < n; i++)
        if(i == 0 || a[cs[i]] != a[cs[i - 1]]){
            key++;
            c[key] = a[cs[i]];
            bb[cs[i]] = key;
        }
        else bb[cs[i]] = key;
    for(int i = 0; i < n- 1; i++){
        a1[i] = read_int();
        a2[i] = read_int();
        a1[i]--;
        a2[i]--;
        star[a1[i]]++;
        star[a2[i]]++;
    }
    for(int i = 1; i <= n; i++) star[i] += star[i - 1];
    for(int i = 0; i < n - 1; i++){
        ke[--star[a1[i]]] = a2[i];
        ke[--star[a2[i]]] = a1[i];
    }
    initLCA();
    for(int i = 0; i < q; i++){
        int u, v, k;
        u = read_int();
        v = read_int();
        k = read_int();
        u--;
        v--;
        int lca = getLCA(u, v);
        printf("%d\n", c[getQuery(root[u], root[v], root[lca], (lca ==
             0? null : root[cha[lca]]), 1, key, k)]);
    }
}
```

5.3 Heavy light decomposition - QTREE

```
#include <cstdio>
#include <vector>
using namespace std;

#define root 0
#define N 10100
#define LN 14

vector <int> adj[N], costs[N], indexx[N];
int baseArray[N], ptr;
int chainNo, chainInd[N], chainHead[N], posInBase[N];
int depth[N], pa[LN][N], otherEnd[N], subsize[N];
int st[N*6], qt[N*6];

/*
 * make_tree:
 * Used to construct the segment tree. It uses the baseArray for
    construction
 */
void make_tree(int cur, int s, int e) {
 if(s == e-1) {
  st[cur] = baseArray[s];
  return;
 }
 int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
 make_tree(c1, s, m);
 make_tree(c2, m, e);
 st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}

/*
 * update_tree:
 * Point update. Update a single element of the segment tree.
 */
void update_tree(int cur, int s, int e, int x, int val) {
 if(s > x || e <= x) return;
 if(s == x && s == e-1) {
  st[cur] = val;
  return;
 }
 int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
 update_tree(c1, s, m, x, val);
 update_tree(c2, m, e, x, val);
 st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}

/*
 * query_tree:
 * Given S and E, it will return the maximum value in the range [S,E)
 */
void query_tree(int cur, int s, int e, int S, int E) {
 if(s >= E || e <= S) {
  qt[cur] = -1;
  return;
 }
 if(s >= S && e <= E) {
  qt[cur] = st[cur];
  return;
 }
 int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
 query_tree(c1, s, m, S, E);
 query_tree(c2, m, e, S, E);
 qt[cur] = qt[c1] > qt[c2] ? qt[c1] : qt[c2];
}

/*
 * query_up:
 * It takes two nodes u and v, condition is that v is an ancestor of u
 * We query the chain in which u is present till chain head, then move
     to next chain up
 * We do that way till u and v are in the same chain, we query for
    that part of chain and break
 */

int query_up(int u, int v) {
 if(u == v) return 0; // Trivial
 int uchain, vchain = chainInd[v], ans = -1;
 // uchain and vchain are chain numbers of u and v
 while(1) {
  uchain = chainInd[u];
  if(uchain == vchain) {
   // Both u and v are in the same chain, so we need to query from u
       to v, update answer and break.
   // We break because we came from u up till v, we are done
   if(u==v) break;
   query_tree(1, 0, ptr, posInBase[v]+1, posInBase[u]+1);
   // Above is call to segment tree query function
   if(qt[1] > ans) ans = qt[1]; // Update answer
   break;
  }
  query_tree(1, 0, ptr, posInBase[chainHead[uchain]], posInBase[u]+1);
  // Above is call to segment tree query function. We do from
      chainHead of u till u. That is the whole chain from
  // start till head. We then update the answer
  if(qt[1] > ans) ans = qt[1];
  u = chainHead[uchain]; // move u to u's chainHead
  u = pa[0][u]; //Then move to its parent, that means we changed
      chains
 }
 return ans;
}

/*
 * LCA:
 * Takes two nodes u, v and returns Lowest Common Ancestor of u, v
 */
int LCA(int u, int v) {
 if(depth[u] < depth[v]) swap(u,v);
 int diff = depth[u] - depth[v];
 for(int i=0; i<LN; i++) if( (diff>>i)&1 ) u = pa[i][u];
 if(u == v) return u;
 for(int i=LN-1; i>=0; i--) if(pa[i][u] != pa[i][v]) {
  u = pa[i][u];
  v = pa[i][v];
 }
 return pa[0][u];
}
```

```
void query(int u, int v) {
 /*
  * We have a query from u to v, we break it into two queries, u to
      LCA(u,v) and LCA(u,v) to v
  */
 int lca = LCA(u, v);
 int ans = query_up(u, lca); // One part of path
 int temp = query_up(v, lca); // another part of path
 if(temp > ans) ans = temp; // take the maximum of both paths
 printf("%d\n", ans);
}

/*
 * change:
 * We just need to find its position in segment tree and update it
 */
void change(int i, int val) {
 int u = otherEnd[i];
 update_tree(1, 0, ptr, posInBase[u], val);
}

/*
 * Actual HL-Decomposition part
 * Initially all entries of chainHead[] are set to -1.
 * So when ever a new chain is started, chain head is correctly
    assigned.
 * As we add a new node to chain, we will note its position in the
    baseArray.
 * In the first for loop we find the child node which has maximum sub-
    tree size.
 * The following if condition is failed for leaf nodes.
 * When the if condition passes, we expand the chain to special child.
 * In the second for loop we recursively call the function on all
    normal nodes.
 * chainNo++ ensures that we are creating a new chain for each normal
    child.
 */
void HLD(int curNode, int cost, int prev) {
 if(chainHead[chainNo] == -1) {
  chainHead[chainNo] = curNode; // Assign chain head
 }
 chainInd[curNode] = chainNo;
 posInBase[curNode] = ptr; // Position of this node in baseArray which
     we will use in Segtree
 baseArray[ptr++] = cost;

 int sc = -1, ncost;
 // Loop to find special child
 for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] != prev)
    {
  if(sc == -1 || subsize[sc] < subsize[adj[curNode][i]]) {
   sc = adj[curNode][i];
   ncost = costs[curNode][i];
  }
 }

 if(sc != -1) {
  // Expand the chain
  HLD(sc, ncost, curNode);
 }

 for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] != prev)
     {
  if(sc != adj[curNode][i]) {
   // New chains at each normal node
   chainNo++;
   HLD(adj[curNode][i], costs[curNode][i], curNode);
  }
 }
}

/*
 * dfs used to set parent of a node, depth of a node, subtree size of
     a node
 */
void dfs(int cur, int prev, int _depth=0) {
 pa[0][cur] = prev;
 depth[cur] = _depth;
 subsize[cur] = 1;
 for(int i=0; i<adj[cur].size(); i++)
  if(adj[cur][i] != prev) {
   otherEnd[indexx[cur][i]] = adj[cur][i];
   dfs(adj[cur][i], cur, _depth+1);
   subsize[cur] += subsize[adj[cur][i]];
  }
}

int main() {
 int t;
 scanf("%d ", &t);
 while(t--) {
  ptr = 0;
  int n;
  scanf("%d", &n);
  // Cleaning step, new test case
  for(int i=0; i<n; i++) {
   adj[i].clear();
   costs[i].clear();
   indexx[i].clear();
   chainHead[i] = -1;
   for(int j=0; j<LN; j++) pa[j][i] = -1;
  }
  for(int i=1; i<n; i++) {
   int u, v, c;
   scanf("%d %d %d", &u, &v, &c);
   u--; v--;
   adj[u].push_back(v);
   costs[u].push_back(c);
   indexx[u].push_back(i-1);
   adj[v].push_back(u);
   costs[v].push_back(c);
   indexx[v].push_back(i-1);
  }

  chainNo = 0;
  dfs(root, -1); // We set up subsize, depth and parent for each node
  HLD(root, -1, -1); // We decomposed the tree and created baseArray
  make_tree(1, 0, ptr); // We use baseArray and construct the needed
      segment tree

  // Below Dynamic programming code is for LCA.
  for(int i=1; i<LN; i++)
   for(int j=0; j<n; j++)
    if(pa[i-1][j] != -1)
     pa[i][j] = pa[i-1][pa[i-1][j]];

  while(1) {
   char s[100];
   scanf("%s", s);
   if(s[0]=='D') {
    break;
   }
   int a, b;
   scanf("%d %d", &a, &b);
   if(s[0]=='Q') {
    query(a-1, b-1);
   } else {
    change(a-1, b);
   }
  }
 }
}
```

6 Graph theory

6.1 Bridge and Cut

```
int numBridge, numVertexCut;
int cc, n, m;
int num[10100], low[10100];
int star[10100];
bool isUsed[100100];
int dhvan[100100], ke[100100];
int nC[10100];
bool mark[10100];
int a1[50100], a2[50100];

void VisitBridge(int u){
    cc++;
    num[u] = cc;
    low[u] = n + 1;
    for(int i = star[u]; i < star[u + 1]; i++)
```

```
        if(isUsed[i]){
            isUsed[dhvan[i]] = false;
            if(num[ke[i]] == 0){
                VisitBridge(ke[i]);
                if(low[ke[i]] > num[u])
                    numBridge++;
                if(low[u] > low[ke[i]])
                    low[u] = low[ke[i]];
            }
            else if(low[u] > num[ke[i]]) low[u] = num[ke[i]];
        }
}

void VisitCut(int u){
    cc++;
    num[u] = cc;
    low[u] = n + 1;
    nC[u] = 0;
    mark[u] = false;
    for(int i = star[u]; i < star[u + 1]; i++)
        if(num[ke[i]] == 0){
            nC[u]++;
            VisitCut(ke[i]);
            mark[u] = mark[u] || (low[ke[i]] >= num[u]);
            if(low[u] > low[ke[i]])
                low[u] = low[ke[i]];
        }
    else if(low[u] > num[ke[i]])
    low[u] = num[ke[i]];
}

int main(){
    memset(isUsed, false, sizeof(isUsed));
    scanf("%d%d", &n, &m);
    for(int i = 0; i < m; i++){
        scanf("%d%d", &a1[i], &a2[i]);
        a1[i]--;
        a2[i]--;
        star[a1[i]]++;
        star[a2[i]]++;
    }
    for(int i = 1; i <= n; i++)
        star[i] += star[i - 1];
    for(int i = 0; i < m; i++){
        ke[--star[a1[i]]] = a2[i];
        ke[--star[a2[i]]] = a1[i];
        dhvan[star[a1[i]]] = star[a2[i]];
        dhvan[star[a2[i]]] = star[a1[i]];
        isUsed[star[a1[i]]] = true;
        isUsed[star[a2[i]]] = true;
    }
    cc = 0;
    for(int i = 0; i < n; i++)
    if(num[i] == 0)
    VisitBridge(i);
    memset(num, 0, sizeof(num));
    memset(low, 0, sizeof(low));
    cc = 0;
    memset(mark, false, sizeof(mark));
    for(int i = 0; i < n; i++){
        if(num[i] == 0){
            VisitCut(i);
            if(nC[i] < 2)
                mark[i] = false;
        }
        if(mark[i]) numVertexCut++;
    }
    cout << numVertexCut << " " <<
    numBridge;
}
```

6.2 Tarjan

```
int top, star[MAXN], stac[MAXN], res,
num[MAXN], low[MAXN], ccount, n, m,
ke[MAXM], a1[MAXM], a2[MAXM];
bool fre[MAXN];

void dfs(int x){
    ccount++;
    num[x] = ccount;
    low[x] = ccount;
    top++;
    stac[top] = x;
    for(int i = star[x]; i < star[x + 1]; i++){
        int h = ke[i];
        if(fre[h]){
            if(num[h] == 0){
                dfs(h);
                low[x] = min(low[x], low[h]);
            } else
                low[x] = min(low[x], num[h]);
        }
    }
    if(num[x] == low[x]){
        res++;
        int h;
        repeat{
            h = stac[top];
            top--;
            fre[h] = false;
        }until(h == x);
    }
}
int main(){
scanf("%d%d", &n, &m);
memset(star, 0, sizeof(star));
for(int i = 0; i < m; i++){
```

```
scanf(”%d%d”, &a1[i], &a2[i]);
star[a1[i]]++;}
for(int i = 2; i <= n + 1; i++)star[i] += star[i - 1];
for(int i = 0; i < m; i++)
ke[--star[a1[i]]] = a2[i];
memset(num, 0, sizeof(num));
memset(low, 0, sizeof(low));
memset(fre, true, sizeof(fre));
ccount = 0;
top = 0;
res = 0;
for(int i = 1; i <= n; i++)
if(num[i] == 0)
dfs(i);
cout << res;}
```

6.3 Biconnected Component

```
// http://vn.spoj.com/problems/SAFENET2/
// Problem: Output the maximum size of a biconnected component
#include <bits/stdc++.h>

using namespace std;
int n;
vector <int> a[30000];

struct BiconnectedComponent {
    vector<int> low, num, s;
    vector< vector<int> > components;
    int counter;
    BiconnectedComponent() : num(n, -1), low(n, -1),
    counter(0) {
        for (int i = 0; i < n; i++)
            if (num[i] < 0)
                dfs(i, 1);
    }
    void dfs(int x, int isRoot) {
        low[x] = num[x] = ++counter;
        if (a[x].empty()) {
            components.push_back(vector<int>(1, x));
            return;
        }
        s.push_back(x);
        for (int i = 0; i < a[x].size(); i++) {
            int y = a[x][i];
            if (num[y] > -1) low[x] = min(low[x], num[y]);
            else {
                dfs(y, 0);
                low[x] = min(low[x], low[y]);
                if (isRoot || low[y] >= num[x]) {
                    components.push_back(vector<int>(1, x));
                    while (1) {
                        int u = s.back();
                        s.pop_back();
                        components.back().push_back(u);
                        if (u == y) break;
                    }
                }
            }
        }
    }
};

int main() {
    int m, x, y;
    scanf(”%d%d”, &n, &m);
    while (m--) {
        scanf(”%d%d”, &x, &y);
        a[--x].push_back(--y);
        a[y].push_back(x);
    }
    BiconnectedComponent bc;
    int ans = 0;
    for (int i = 0; i < bc.components.size(); i++)
        ans = max(ans, int(bc.components[i].size()));
    printf(”%d\n”, ans);
}
```

6.4 Centroid Decomposition

```
int find(int u) {
 int t=1;q[0]=u;f[u]=-1;
 rep(i,0,t) {
  u=q[i];
  rep(j,0,e[u].size()) {
   int v=e[u][j].fi;
   if (!vis[v]&&v!=f[u]) f[q[t++]=v]=u;
  }
  ms[q[i]]=0;
  sz[q[i]]=1;
 }
 for (int i=t-1;i>=0;i--) {
  ms[q[i]]=max(ms[q[i]],t-sz[q[i]]);
  if (ms[q[i]]*2<=t) return q[i];
  sz[f[q[i]]]+=sz[q[i]];
  ms[f[q[i]]]=max(ms[f[q[i]]],sz[q[i]]);
 }
 return 0;
}
```

Another code

```
/* The truth is everyone is going to hurt you. You just got to find
    the ones worth suffering for... */

#include <bits/stdc++.h>

using namespace std;
```

```
const int maxN = (int)1E5+5;
const int maxK = 18;

int n;
vector<int> Adj[maxN], curTree[maxN], nextLev[maxN], centroidTree[maxN
    ];
int deg[maxN], szSub[maxN], visit_T[maxN];
int cur_T;

void DFS(int u, int start_T) {
 visit_T[u] = ++cur_T;
 szSub[u] = 1;
 curTree[u].clear();
 for (int i = 0, sz = Adj[u].size(); i < sz; ++i)
  if (deg[Adj[u][i]] == 0 && visit_T[Adj[u][i]] <= start_T) {
   DFS(Adj[u][i], start_T);
   szSub[u] += szSub[Adj[u][i]];
   curTree[u].push_back(Adj[u][i]);
  }
}

int getCentroid(int u, int curN) {
 for (int i = 0, sz = curTree[u].size(); i < sz; ++i)
  if (szSub[curTree[u][i]] > curN/2)
   return getCentroid(curTree[u][i], curN);
 for (int i = 0, sz = Adj[u].size(); i < sz; ++i)
  if (deg[Adj[u][i]] == 0)
   nextLev[u].push_back(Adj[u][i]);
 return u;
}

int decomposeTree(int u, int curDeg) {
 DFS(u, cur_T);
 int r = getCentroid(u, szSub[u]);
 deg[r] = curDeg;
 for (int i = 0, sz = nextLev[r].size(); i < sz; ++i)
  centroidTree[r].push_back(decomposeTree(nextLev[r][i], curDeg+1));
 return r;
}

int main() {
 scanf("%d", &n);
 for (int u = 1; u <= n; ++u)
  Adj[u].clear();
 for (int i = 0; i < n-1; ++i) {
  int u, v;
  scanf("%d %d", &u, &v);
  Adj[u].push_back(v);
  Adj[v].push_back(u);
 }

 memset(deg, 0, sizeof(deg));
 memset(visit_T, 255, sizeof(visit_T));
 for (int u = 1; u <= n; ++u) {
  nextLev[u].clear();
  centroidTree[u].clear();
 }
 cur_T = 0;
 decomposeTree(1, 1);
 for (int u = 1; u <= n; ++u)
  for (int i = 0, sz = centroidTree[u].size(); i < sz; ++i)
   printf("%d -> %d\n", u, centroidTree[u][i]);
 return 0;
}
/*


*/
```

6.5 A small theorem

Given a non-negative integers array d, n elements are arranged in a non-decreasing order. We have a theorem:

$$\begin{cases}\sum_{u=1}^{n} d_u \text{ is even}\\ \sum_{u=1}^{k} d_u \leq k(k-1)+\sum_{v=k+1}^{n} min(d_v,k), \forall k=1,2,...,2\end{cases}$$

if and only if exist a graph $G=(V,E)$ satisfies $|V|=n, |E|=\frac{1}{2}\sum_{u=1}^{n} d_u$ and d_u is the degree of the vertex u.

6.6 PTREE

```
int res[202][202],c[202],star[202],ke[404],k,ress,n,i,a1[202],a2[202],
    j,cha[202];
int trace[202][202];
bool v[202];

void dfs(int n1){
    int i1,j1,k1;
    res[n1][1] = c[n1];
    v[n1] = true;
    for (i1 = star[n1]+1; i1 <= star[n1+1];i1++)
        if (v[ke[i1]] == false){
            cha[ke[i1]] = n1;
            dfs(ke[i1]);
            for (j1 = k; j1 >= 2; j1--)
                for (k1 = 1; k1 < j1; k1++)
                    if (res[n1][j1] < res[n1][k1] + res[ke[i1]][j1-k1
                        ]){
                        res[n1][j1] = res[n1][k1] + res[ke[i1]][j1-k1
                            ];
                        trace[ke[i1]][j1] = j1-k1;}}}

void findpath(int n1, int k1){
    int i1;
    for (i1 = star[n1+1]; i1 >= star[n1]+1; i1--)
```

```
        if (cha[ke[i1]] == n1 && trace[ke[i1]][k1] > 0){
            findpath(ke[i1],trace[ke[i1]][k1]);
            k1-=trace[ke[i1]][k1];}
    cout << n1 << " ";}

int main(){
    cin >> n >> k;
    for (i = 1; i <= n; i++) cin >> c[i];
    for (i = 1; i < n; i++){
        cin >> a1[i] >> a2[i];
        star[a1[i]]++;
        star[a2[i]]++;}
    for (i = 2; i <= n; i++)
        star[i]+=star[i-1];
    star[n+1]=star[n];
    for (i = 1; i < n; i++){
        ke[star[a1[i]]] = a2[i];
        star[a1[i]]--;
        ke[star[a2[i]]] = a1[i];
        star[a2[i]]--;}
    for (i = 1; i <= n; i++) for (j = 1; j <= k; j++) res[i][j] =
        -1000000000;
    memset(v,false,sizeof(v));
    memset(trace,0,sizeof(trace));
    memset(cha,0,sizeof(cha));
    ress = 1;
    dfs(1);
    for (i = 2; i <= n; i++) if (res[ress][k] < res[i][k]) ress = i;
    findpath(ress,k);}
```

7 Geometry

7.1 Rotation matrix

$$\begin{bmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{bmatrix}$$

7.2 Convex hull

```
typedef long long LL;

struct TPoint {
 int x, y;
 TPoint(int _x = 0, int _y = 0): x(_x), y(_y) {}
 const bool operator < (const TPoint &Other) const {
  return x < Other.x || (x == Other.x && y < Other.y);
 }
 const TPoint operator - (const TPoint &Other) const {
  return TPoint(x-Other.x, y-Other.y);
 }
};

LL Cross(TPoint O, TPoint A, TPoint B) {
 TPoint OA = A-O, OB = B-O;
 return (LL)OA.x*OB.y-(LL)OB.x*OA.y;
}

vector<TPoint> MonotoneChain(vector<TPoint> P) {
 sort(P.begin(), P.end());

 int n = P.size(), m = 0;
 vector<TPoint> CH(2*n);
 for (int i = 0; i < n; ++i) {
  while (m >= 2 && Cross(CH[m-2], CH[m-1], P[i]) <= 0) --m;
  CH[m++] = P[i];
 }
 for (int i = n-2, t = m+1; i >= 0; --i) {
  while (m >= t && Cross(CH[m-2], CH[m-1], P[i]) <= 0) --m;
  CH[m++] = P[i];
 }
 CH.resize(m-1);
 return CH;
}
```

7.3 Other methods

```
struct Line{
    double a, b, c;
};

const int compare(const double &x, const double &y){
    if(fabs(x - y) < EPS) return 0;
    return (x > y) - (x < y);
}

void pointToLine(const Point &p1, const Point &p2, Line &l){
    if(fabs(p1.x - p2.x) < EPS){
        l.a = 1.0; l.b = 0.0; l.c = -p1.x;
    }else{
        l.a = -(double)(p1.y - p2.y) / (p1.x -
        p2.x);
        l.b = 1.0;
        l.c = -(double)(l.a *p1.x) - p1.y;
    }
}

bool areIntersect(const Line &l1, const Line &l2, Point &p){
    if(areParallel(l1, l2))
        return false;
    p.x = (l2.b * l1.c - l1.b *l2.c) / (l2.a * l1.b - l1.a * l2.b);
    if(fabs(l1.b) > EPS) p.y = -(l1.a *p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c);
    return true;
}

const double distanceToLine(const Point &p, const Point &a, const
    Point &b, Point &c){
```

```
    Vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    c = translate(a, scale(ab, u));
    return p.Distance(c);
}

const double distToLineSegment(const Point &p, const Point &a, const
    Point &b, Point &c){
    Vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    if(u < 0.0){
        c = a;
        return p.Distance(a);
    }
    if(u > 1.0){
        c = b;
        return p.Distance(b);
    }
    return distanceToLine(p, a, b, c);
}

const double angle(const Point &a, const Point &o, const Point &b){
    Vec oa = toVec(o, a), ob = toVec(o, b);
    return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}

const int insideCircle(const Point &p, const Point &c, const double &r
    ){
    double dx = p.x - c.x, dy = p.y - c.y;
    double Euc = dx * dx + dy * dy, rSq = r * r;
    return compare(Euc, rSq) < 0 ? 0 : compare(Euc, rSq) == 0? 1 : 2;
}

const bool circle2PtsRad(const Point &p1, const Point &p2, const
    double &r, Point &c){
    double d2 = (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y
        - p2.y);
    double det = r * r / d2 - 0.25;
    if(det < 0.0) return false;
    double h = sqrt(det);
    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true;
}

bool inPolygon(const Point &pt, const vector<Point> &P){
    if(P.size() == 0)
        return false;
    double sum = 0;
    for(int i = 0; i < P.size() - 1; i++)
        if(ccw(pt, P[i], P[i + 1]))
            sum += angle(P[i], pt, P[i + 1]);
        else
            sum -= angle(P[i], pt, P[i + 1]);
    return fabs(fabs(sum) - 2*M_PI) < EPS;
}

double area(const vector<Point> &p){
    double res = 0.0, x1, y1, x2, y2;
    for(int i = 0; i < p.size() - 1; i++){
        x1 = p[i].x; x2 = p[i + 1].x;
        y1 = p[i].y; y2 = p[i + 1].y;
        res += (x1 * y2 - x2 * y1);
    }
    return fabs(res) / 2.0;
}
```

8 Number theory

8.1 Extended Euclidean

```
long long xgcd(long long b, long long n)
{
    long long sn = n;
    long long x0, x1, y0, y1;
    x0 = 1, x1 = 0, y0 = 0, y1 = 1;
    while (n != 0)
    {
        long long a1 = b / n, a2 = n, a3 = b % n;
        long long q = a1;
        b = a2;
        n = a3;
        a1 = x1, a2 = x0 - q * x1;
        x0 = a1, x1 = a2;
        a1 = y1, a2 = y0 - q * y1;
        y0 = a1, y1 = a2;
    }
    //result is b, x0, y0
    return ((x0 % sn) + sn) %sn;
}
```

8.2 Chinese Remainder Theorem

Equations. $x = a_i(\bmod m_i)$ and $1 \leq i \leq r$
Result. $x = \sum a_i * b_i * \frac{M}{m_i}(\bmod M)$
Where $M = m_1 * m_2 * ... * m_r$ and $b_i * \frac{M}{m_i} = 1(\bmod m_i)$.

8.3 Sieve

```
//O(N) prime sieve,
bool isprime[n];
int lp[n+1];

void sieve(int n, vi &pr) {
    pr.clear();
    fori(i,2,n) {
        if (lp[i]==0){
            pr.pb(lp[i]=i);
            isprime[i]=true;
        }
    for (int j=0;j<(int)sz(pr) && pr[j]<=lp[i] && i*pr[j]<=n;j++)
        lp[i*pr[j]]=pr[j];
    }
}
//factorization using sieve,
int lp[n+1];
void sievefac(int n, vpii &a) {
    a.clear();
    while (n>1) {
        a.pb(mp(lp[n], 1));
        if (sz(a)>1 && a[(int)sz(a)-1].first==a[(int)sz(a)-2].first) {
            a.pop_back();
            a.back().second++;
        }
        n/=lp[n];
    }
}
```

8.4 Rabin Miller

```
/* This function calculates (ab)%c */
int modulo(int a,int b,int c){
    long long x=1,y=a; // long long is taken to avoid overflow of
        intermediate results
    while(b > 0){
        if(b%2 == 1){
            x=(x*y)%c;
        }
        y = (y*y)%c; // squaring the base
        b /= 2;
    }
    return x%c;
}

/* this function calculates (a*b)%c taking into account that a*b might
     overflow */
long long mulmod(long long a,long long b,long long c){
    long long x = 0,y=a%c;
    while(b > 0){
        if(b%2 == 1){
            x = (x+y)%c;
        }
        y = (y*2)%c;
        b /= 2;
    }
    return x%c;
}

/* Miller-Rabin primality test, iteration signifies the accuracy of
    the test */
bool Miller(long long p,int iteration){
    if(p<2){
        return false;
    }
    if(p!=2 && p%2==0){
        return false;
    }
    long long s=p-1;
    while(s%2==0){
        s/=2;
    }
    for(int i=0;i<iteration;i++){
        long long a=rand()%(p-1)+1,temp=s;
        long long mod=modulo(a,temp,p);
        while(temp!=p-1 && mod!=1 && mod!=p-1){
            mod=mulmod(mod,mod,p);
            temp *= 2;
        }
        if(mod!=p-1 && temp%2==0){
            return false;
        }
    }
    return true;
}
```

8.5 Catalan numbers

Formula. $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n} \frac{n+k}{k}$ for $n \geq 0$.
Alternative. $C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}$
Or. $C_0 = 1$ and $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$
Or. $C_n = \frac{1}{n+1} \sum_{i=0}^{n} \binom{n}{i}^2$
Or. $C_0 = 1$ and $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$
Or. $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$
For instance.1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, ...

8.6 Fibonacci sequence

Initialize. $F_1 = F_2 = 1$
if n is odd. $F_n = F_{(n-1)/2}^2 + F_{(n+1)/2}^2$
if n is even. $F_n = (2F_{n/2-1} + F_{n/2})F_{n/2}$

8.7 Other methods

```
ll _sieve_size;
```

```
bitset<10000010> bs;
vi primes;

void sieve(ll upperbound) {
    _sieve_size = upperbound + 1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
        for (ll j = i * i; j <= _sieve_size; j += i)
            bs[j] = 0;
        primes.push_back((int)i);
    }
}
bool isPrime(ll N) {
    if (N <= _sieve_size) return bs[N];
    for (int i = 0; i < (int)primes.size(); i++)
        if (N % primes[i] == 0) return false;
    return true;
}
// second part
vi primeFactors(ll N) {
    vi factors;
    ll PF_idx = 0, PF = primes[PF_idx];
    while (N != 1 && (PF * PF <= N)) {
        while (N % PF == 0) {
            N /= PF;
            factors.push_back(PF);
        }
        PF = primes[++PF_idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}
// third part
ll numPF(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N)) {
        while (N % PF == 0) { N /= PF; ans++; }
        PF = primes[++PF_idx];
    }
    if (N != 1) ans++;
    return ans;
}

ll numDiffPF(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N)) {
        if (N % PF == 0) ans++;
        while (N % PF == 0) N /= PF;
        PF = primes[++PF_idx];
    }
    if (N != 1) ans++;
    return ans;
}
```

```
ll sumPF(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N)) {
        while (N % PF == 0) {
            N /= PF; ans += PF;
        }
        PF = primes[++PF_idx];
    }
    if (N != 1) ans += N;
        return ans;
}

ll numDiv(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = 1;
    while (N != 1 && (PF * PF <= N)) {
        ll power = 0;
        while (N % PF == 0) { N /= PF; power++; }
        ans *= (power + 1);
        PF = primes[++PF_idx];
    }
    if (N != 1) ans *= 2;
    return ans;
}

ll sumDiv(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = 1;
    while (N != 1 && (PF * PF <= N)) {
        ll power = 0;
        while (N % PF == 0) { N /= PF; power++; }
        ans *= ((ll)pow((double)PF, power + 1.0) - 1) / (PF - 1);
        PF = primes[++PF_idx];
    }
    if (N != 1) ans *= ((ll)pow((double)N, 2.0) - 1) / (N -
    1);
    return ans;
}
ll EulerPhi(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = N;
    while (N != 1 && (PF * PF <= N)) {
        if (N % PF == 0) ans -= ans / PF;
        while (N % PF == 0) N /= PF;
        PF = primes[++PF_idx];
    }
    if (N != 1) ans -= ans / N;
    return ans;
}
```

9 Algebra

9.1 Gaussian method for solving linear system

```
const int MAXN = 100;
struct AMatrix{
    double mat[MAXN][MAXN + 1];
};

struct CVector{
    double vec[MAXN];
};

CVector Gauss(const int &N, AMatrix Aug){
    int i, j, k, l; double t; CVector X;
    for(j = 0; j < N - 1; j++){
        l = j;
        for(i = j + 1; i < N; i++)
            if(fabs(Aug.mat[i][j]) > fabs(Aug.mat[l][j])) l = i;
        for(k = j; k <= N; k++){
            t = Aug.mat[j][k];
            Aug.mat[j][k] = Aug.mat[l][k];
            Aug.mat[l][k] = t;
        }
        for(i = j + 1;i < N; i++)
            for(k = N; k >= j; k--)
                Aug.mat[i][k] -= Aug.mat[j][k] * Aug.mat[i][j] / Aug.
                    mat[j][j];
    }
    for(j = N - 1; j >=0; j--){
        for(t = 0.0, k = j + 1; k < N; k++)
            t += Aug.mat[j][k] * X.vec[k];
        X.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j];
    }
    return X;
}
```

10 Other Libraries

10.1 EXT ROPE

```
#include <iostream>
#include <cstdio>
#include <ext/rope> //header with rope
using namespace std;
using namespace __gnu_cxx; //namespace with rope and some additional
    stuff
int main() {
    ios_base::sync_with_stdio(false);
    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    int l, r;
    for(int i = 0; i < m; ++i){
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope <int>::iterator it = v.mutable_begin(); it != v.
        mutable_end(); ++it)
        cout << *it << " ";
    return 0;
}
```

10.2 Order Statistics

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree< int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
int main() {
    ordered_set X;
    for(int i = 1; i <= 16; i *= 2)
        X.insert(i);
    cout << *X.find_by_order(1) << endl; // 2
    cout << *X.find_by_order(2) << endl; // 4
    cout << *X.find_by_order(4) << endl; // 16
    cout << (X.end()==X.find_by_order(6)) << endl; // true
    cout<<X.order_of_key(-5)<<endl; // 0
    cout<<X.order_of_key(1)<<endl; // 0
    cout<<X.order_of_key(3)<<endl; // 2
    cout<<X.order_of_key(4)<<endl; // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

10.3 LYNDON

```
// Decompose s = w1w2..wk : k max and w1 >= w2
>= ...
#include <string>
#include <cstdio>
#include <iostream>
using namespace std;
void lyndon(string s) {
    int n = (int) s.length();
    int i = 0;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else ++k;
            ++j;
        }
        while (i <= k) {
            cout << s.substr(i, j - k) << ' ';
            i += j - k;
        }
    }
    cout << endl;
}
```

10.4 MINMOVE

```
// Tinh vi tri cua xau xoay vong co thu tu tu dien nho nhat cua xau s
    []
int minmove(string s) {
    int n = s.length();
    int x, y, i, j, u, v; // x is the smallest string before string y
    for (x = 0, y = 1; y < n; ++ y) {
        i = u = x;
        j = v = y;
        while (s[i] == s[j]) {
            ++ u; ++ v;
            if (++ i == n) i = 0;
            if (++ j == n) j = 0;
            if (i == x) break; // All strings are equal
        }
        if (s[i] <= s[j]) y = v;
        else {
            x = y;
            if (u > y) y = u;
        }
    }
    return x;
}
```

10.5 $\sum_{k=1}^{n} k^2$ with $gcd(k,n) = 1$.

```
/*Example for the sequence:*/
0, 1, 5, 10, 30, 26, 91, 84, 159, 140, 385, 196, 650, 406, 620, 680,
    1496, 654, 2109, 1080, 1806, 1650, 3795, 1544, 4150, 2756, 4365,
    3164, 7714, 2360, 9455, 5456, 7370, 6256, 9940, 5196, 16206, 8778,
     12324, 8560, 22140, 6972, 25585, ...

/*Formula:*/
if (n = p_1^e_1 * ... *p_r^e_r) => a(n) = n^2*phi(n)/3 + (-1)^r*p_1
    *..._p_r*phi(n)/6.
```

10.6 FFT

```
typedef complex<double> comp;
const double pi = acos(-1.0);
int n, x, sa[100010], sb[100010], neo=0;
vector<int> a, b, ans;
int res = 0LL;

int get_inv(int n, int &log_n){
    int ans=0;
    FOR(i, 0, log_n-1)
    if ( n&(1<<i) ) ans += (1<<(log_n-i-1));
    return ans;
}

void fft(vector<comp> &a,int n_lg, bool inv){
    int n = sz(a);
    if(n<=1)return;
    FOR(i,0,n-1)
        if (i < get_inv(i,n_lg))
            swap(a[i], a[get_inv(i,n_lg)]);
    for(int len=2;len<=n;len<<=1){
        for(int i=0;i<n;i+=len) {
            double ang = 2*pi/len * (inv?-1:1);
            comp w(1), ws(cos(ang),sin(ang));
            for(int j=0; j<len/2; ++j){
                comp w1=a[i+j], w2=a[i+j+len/2]*w;
                a[i+j] = w1+w2;
                a[i+j+len/2]=w1-w2;
                w* = ws;
            }
        }
    }
    if (inv)
        FOR(i,0,n-1)
            a[i]/=double(n);
}
void multiply(const vector<int> &a,const vector<int> &b, vector<int> &
    ans){
    vector<comp> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = max(sz(fa), sz(fb));
    n<<=1;
    int lg = 0;
    while ((1<<lg) < n) lg++;
    n=(1<<lg);
    fa.resize(n), fb.resize(n);
    fft(fa,lg,false);
    fft(fb,lg,false);
    FOR(i,0,n-1) fa[i]*=fb[i];
    fft(fa,lg,true);
    ans.resize(n);
    FOR(i,0,n-1) ans[i] = int(fa[i].real()+0.5);
}
```

11 Sequences

11.1 Ordered Bell Numbers

1, 1, 3, 13, 75, 541, 4683, 47293, 545835, 7087261, 102247563, ...
$a(n) = \sum_{i=1}^{n} \binom{n}{i} a(n-i)$

11.2 Bell numbers

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322

General. $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$

Modulo p.

$B_{p+n} \equiv B_n + B_{n+1} \pmod{p}$

$B_{p^m+n} \equiv mB_n + B_{n+1} \pmod{p}$.