Contents

1 Basic

1.1 .vimrc

```
colo torte
syn on
se cin ai ar sm nu ru is
se mouse=a bs=2 ww+=<,>,[,] so=6 ts=4 sw=4 ttm=100
se makeprg=g++\ -Wall\ -Wshadow\ -O2\ -o\ %<\ %
au BufNewFile *.cpp 0r ~/default.cpp

map <F7> <ESC>:wa<CR>:make!<CR>
imap <F7> <ESC>:wa<CR>:make!<CR>
map <C-F7> <ESC>:tabe %<.in<CR>
map <F8> :cope <CR>
map <S-F8> :ccl <CR>
map <F9> :!./%< <CR>
map <C-F9> :!./%< < %<.in <CR>
```

1.2 IncreaseStackSize

```
//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

1.3 Default Code

```
#include<bits/stdc++.h>
#include<cmath>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
#define _SZ(n) memset((n),0,sizeof(n))
#define _SMO(n) memset((n),-1,sizeof(n))
#define _MC(n,m) memcpy((n),(m),sizeof(n))
#define _F first
#define _S second
#define _MP make_pair
#define _PB push_back
#define FOR(x,y) for(__typeof(y.begin())x=y.begin();x
    !=y.end();x++)
#define IOS ios_base::sync_with_stdio(0)
// Let's Fight!

int main()
{
  return 0;
}
```

2 Data Structure

2.1 Bigint

```
const int bL = 1000;
const int bM = 10000;

struct Bigint{
    int v[bL],l;
    Bigint(){ memset(v, 0, sizeof(v));l=0; }

    void n(){
        for(;l;l--) if(v[l-1]) return;
    }

    Bigint(long long a){
        for(l=0;a;v[l++]=a%bM,a/=bM);
    }
    Bigint(char *a){
        l=0;
        int t=0,i=strlen(a),q=1;
        while(i){
            t+=(a[--i]-'0')*q;
            if((q*=10)>=bM) {
                v[l++]=t; t=0; q=1;
            }
        }
        if(t) v[l++]=t;
    }

    void prt() {
        if(l==0){ putchar('0');return; }
        printf("%d",v[l-1]);
        for(int i=l-2;i>=0;i--) printf("%.4d",v[i]);
    }

    int cp3(const Bigint &b)const {
        if(l!=b.l) return l>b.l?1:-1;
        for(int i=l-1;i>=0;i--)
            if(v[i]!=b.v[i])
                return v[i]>b.v[i]?1:-1;
        return 0;
    }

    bool operator < (const Bigint &b)const{ return
        cp3(b)==-1; }
    bool operator == (const Bigint &b)const{ return
        cp3(b)==0; }
    bool operator > (const Bigint &b)const{ return
        cp3(b)==1; }

    Bigint operator + (const Bigint &b) {
        Bigint r;
        r.l=max(l,b.l);
        for(int i=0;i<r.l;i++) {
            r.v[i]+=v[i]+b.v[i];
            if(r.v[i]>=bM) {
                r.v[i+1]+=r.v[i]/bM;
                r.v[i]%=bM;
            }
        }
        if(r.v[r.l]) r.l++;
        return r;
    }

    Bigint operator - (const Bigint &b) {
        Bigint r;
        r.l=l;
        for(int i=0;i<l;i++) {
            r.v[i]+=v[i];
            if(i<b.l) r.v[i]-=b.v[i];
            if(r.v[i]<0) {
                r.v[i]+=bM;
                r.v[i+1]--;
            }
        }
        r.n();
        return r;
    }

    Bigint operator * (const Bigint &b) {
        Bigint r;
        r.l=l+b.l;
        for(int i=0;i<l;i++) {
            for(int j=0;j<b.l;j++) {
                r.v[i+j]+=v[i]*b.v[j];
                if(r.v[i+j]>=bM) {
                    r.v[i+j+1]+=r.v[i+j]/bM;
                    r.v[i+j]%=bM;
                }
            }
        }
        r.n();
        return r;
    }

    Bigint operator / (const Bigint &b) {
        Bigint r;
        r.l=max(1,l-b.l+1);
        for(int i=r.l-1;i>=0;i--) {
            int d=0,u=bM-1,m;
            while(d<u) {
                m=(d+u+1)>>1;
                r.v[i]=m;
                if((r*b)>(*this)) u=m-1;
                else d=m;
            }
            r.v[i]=d;
        }
        r.n();
        return r;
    }

    Bigint operator % (const Bigint &b) {
        return (*this)-(*this)/b*b;
    }
};
```

2.2 Leftist Heap

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

using namespace std;

const int MAXSIZE = 10000;

class Node{
public:
  int num,lc,rc;
  Node () : num(0), lc(-1), rc(-1) {}
  Node (int _v) : num(_v), lc(-1), rc(-1) {}
}tree[MAXSIZE];

int merge(int x, int y){
    if (x == -1) return y;
    if (y == -1) return x;
    if (tree[x].num < tree[y].num)
        swap(x, y);
    tree[x].rc = merge(tree[x].rc, y);
    swap(tree[x].lc, tree[x].rc);
    return x;
}

/* Usage
merge: root = merge(x, y)
delmin: root = merge(root.lc, root.rc)
*/
```

2.3 $\texttt{extc}_{b}alance_{t}ree$

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> set_t;

int main()
{
  // Insert some entries into s.
  set_t s;
  s.insert(12);
  s.insert(505);

  // The order of the keys should be: 12, 505.
  assert(*s.find_by_order(0) == 12);
  assert(*s.find_by_order(3) == 505);

  // The order of the keys should be: 12, 505.
  assert(s.order_of_key(12) == 0);
  assert(s.order_of_key(505) == 1);

  // Erase an entry.
  s.erase(12);

  // The order of the keys should be: 505.
  assert(*s.find_by_order(0) == 505);

  // The order of the keys should be: 505.
  assert(s.order_of_key(505) == 0);

}
```

2.4 Treap

```
class Node{
public:
  int pri,num,cnt,lc,rc;
  Node () : pri(-1), num(0), cnt(0), lc(0), rc(0) {}
  Node (int _num){
    pri = (rand()<<15) + rand();
    num = _num;
    cnt = 1;
    lc = rc = 0;
  }
}tree[MX];
int nMem;

int get_rand(){
  return (rand()<<15) + rand();
}
int get_node(){
  tree[nMem] = Node();
  if (nMem >= MX) while(1);
  return nMem++;
}
void upd_node(int rt){
  if (!rt) return ;
  int lc=tree[rt].lc;
  int rc=tree[rt].rc;
  tree[rt].cnt = tree[lc].cnt + tree[rc].cnt + 1;
}
int merge(int a, int b){
  if (!a) return b;
  if (!b) return a;
  int res=0;
  if (tree[a].pri > tree[b].pri){
    res = a; //get_node();
    tree[res] = tree[a];
    tree[res].rc = merge(tree[res].rc,b);
  } else {
    res = b; //get_node();
    tree[res] = tree[b];
    tree[res].lc = merge(a,tree[res].lc);
  }
  upd_node(res);
  return res;
}
pair<int,int> split(int a, int k){
  if (k == 0) return MP(0,a);
  if (k == tree[a].cnt) return MP(a,0);
  int lc=tree[a].lc, rc=tree[a].rc;
  pair<int,int> res;
  int np=a; //get_node();
  //tree[np] = tree[a];
  if (tree[lc].cnt >= k){
    res = split(lc,k);
    tree[np].lc = res._S;
    res._S = np;
  } else {
    res = split(rc,k-tree[lc].cnt-1);
    tree[np].rc = res._F;
    res._F = np;
  }
  upd_node(res._F);
  upd_node(res._S);
  return res;
}
```

3 Graph

3.1 Tarjan

```
const int MAXV = 101000;

int V, E;
vector<int> el[MAXV];
int dfn[MAXV], low[MAXV], did;
bool ins[MAXV];
stack<int> st;
int scc[MAXV], scn;

void tarjan(int u){
  cout << u << endl;
  dfn[u] = low[u] = ++did;
  st.push(u); ins[u] = true;

  for(int i=0; i<(int)el[u].size(); i++){
    int v = el[u][i];
    if(!dfn[v]){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if(ins[v]){
      low[u] = min(low[u], dfn[v]);
```

```
    }
  }

  if(dfn[u] == low[u]){
    int v;
    do{
      v = st.top();
      st.pop();
      scc[v] = scn;
      ins[v] = false;
    }while(v != u);
    scn ++;
  }
}

void calcscc(){
  did = scn = 0;
  for(int i=0; i<V; i++){
    if(!dfn[i]) tarjan(i);
  }
}
```

3.2 Strongly Connected Components:Kosaraju's Algorithm

```
class Scc{
public:
  int n,vst[MAXN];
  int nScc,bln[MAXN];
  vector<int> E[MAXN], rE[MAXN], vc;
  void init(int _n){
    n = _n;
    for (int i=0; i<MAXN; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u]._PB(v);
    rE[v]._PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    FOR(it,E[u]){
      if (!vst[*it])
        DFS(*it);
    }
    vc._PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    FOR(it,rE[u]){
      if (!vst[*it])
        rDFS(*it);
    }
  }
  void solve(){
    nScc=0;
    vc.clear();
    _SZ(vst);
    for (int i=0; i<n; i++){
      if (!vst[i])
        DFS(i);
    }
    reverse(vc.begin(),vc.end());
    _SZ(vst);
    FOR(it,vc){
      if (!vst[*it]){
        rDFS(*it);
        nScc++;
      }
    }
  }
};
```

3.3 DMST

```
struct Edge{
  int u,v,c;
}eg[N*N];
int n, m, nSpe, nE, bln[N], vst[N], dis[N], ismrg[N],
    prev[N];

int dmst(int root){
  _SZ(ismrg);
  int curW, ww;
  curW=ww=0;
  while (1){
    _SMO(prev);
    for (int i=0; i<nSpe; i++)
      dis[i] = INF;
    for (int i=0; i<nE; i++){
      if (eg[i].v!=eg[i].u && eg[i].v!=root && dis[eg
          [i].v] > eg[i].c){
        dis[eg[i].v] = eg[i].c;
        prev[eg[i].v] = eg[i].u;
      }
    }

    // find cycle
    int sign=1;
    curW=0;
    _SMO(bln);
    _SMO(vst);
    for (int i=0; i<nSpe; i++){
      if (ismrg[i]) continue;
      if (prev[i]==-1 && i!=root) return INF;
      if (i!=root) curW += dis[i];
      int s;
      for (s=i; s!=-1 && vst[s]==-1; s=prev[s])
        vst[s]=i;

      if (s!=-1 && vst[s]==i){
        sign=0;
        int j=s;
        while (1){
          ismrg[j]=1;
          bln[j]=s;
          ww += dis[j];
          j=prev[j];
          if (j==s) break;
        }
        ismrg[s]=0;
      }
    }

    if (sign) break;

    // merge
    for (int i=0; i<nE; i++){
      if (bln[eg[i].v]!=-1) eg[i].c -= dis[eg[i].v];
      if (bln[eg[i].u]!=-1) eg[i].u = bln[eg[i].u];
      if (bln[eg[i].v]!=-1) eg[i].v = bln[eg[i].v];
      if (eg[i].u==eg[i].v) eg[i--] = eg[--nE];
    }
    //  system("pause");
  }
  return curW+ww;
}
```

3.4 DMST$_{w}ith_{s}ol$

```
const int INF = 1029384756;

struct edge_t{
    int u,v,w;
    set< pair<int,int> > add,sub;
    edge_t(){
        u = -1;
        v = -1;
        w = 0;
    }
    edge_t(int _u, int _v, int _w){
        u = _u;
        v = _v;
        w = _w;
        add.insert(_MP(_u,_v));
```

```
    }
    edge_t& operator += (const edge_t& obj) {
        w += obj.w;
        FOR (it, obj.add) {
            if (!sub.count(*it)) add.insert(*it);
            else sub.erase(*it);
        }
        FOR (it, obj.sub) {
            if (!add.count(*it)) sub.insert(*it);
            else add.erase(*it);
        }
        return *this;
    }

    edge_t& operator -= (const edge_t& obj) {
        w -= obj.w;
        FOR (it, obj.sub) {
            if (!sub.count(*it)) add.insert(*it);
            else sub.erase(*it);
        }
        FOR (it, obj.add) {
            if (!add.count(*it)) sub.insert(*it);
            else add.erase(*it);
        }
        return *this;
    }
}eg[MXN*MXN],prv[MXN],EDGE_INF(-1,-1,INF);
int N,M;
int cycid,incycle[MXN],contracted[MXN];
vector<int> E[MXN];

edge_t dmst(int rt){
    edge_t cost;
    for (int i=0; i<N; i++){
        contracted[i] = 0;
        incycle[i] = 0;
        prv[i] = EDGE_INF;
    }
    cycid = 0;
    int u,v;
    while (true){
        for (v=0; v<N; v++){
            if (v != rt && !contracted[v] && prv[v].w
                 == INF)
                break;
        }
        if (v >= N) break; // end
        for (int i=0; i<M; i++){
            if (eg[i].v == v && eg[i].w < prv[v].w){
                prv[v] = eg[i];
            }
        }
        if (prv[v].w == INF){ // not connected
            return EDGE_INF;
        }
        cost += prv[v];
        for (u=prv[v].u; u!=v && u!=-1; u=prv[u].u);
        if (u == -1) continue;
        incycle[v] = ++cycid;
        for (u=prv[v].u; u!=v; u=prv[u].u){
            contracted[u] = 1;
            incycle[u] = cycid;
        }
        for (int i=0; i<M; i++){
            if (incycle[eg[i].u] != cycid && incycle[
                eg[i].v] == cycid){
                eg[i] -= prv[eg[i].v];
            }
        }
        for (int i=0; i<M; i++){
            if (incycle[eg[i].u] == cycid) eg[i].u =
                v;
            if (incycle[eg[i].v] == cycid) eg[i].v =
                v;
            if (eg[i].u == eg[i].v) eg[i--] = eg[--M
                ];
        }
        for (int i=0; i<N; i++){
            if (contracted[i]) continue;
            if (prv[i].u>=0 && incycle[prv[i].u] ==
                cycid)
                prv[i].u = v;
        }
        prv[v] = EDGE_INF;
    }
    return cost;
}

void solve(){
    edge_t cost = dmst(0);
    FOR(it,cost.add){ // find a solution
        E[it->_F]._PB(it->_S);
        prv[it->_S] = edge_t(it->_F,it->_S,0);
    }
}
```

4 Flow

4.1 ISAP

```
class Isap{
public:
  class Edge{
  public:
    int v,f,re;
    Edge (){ v=f=re=-1; }
    Edge (int _v, int _f, int _r){
      v = _v;
      f = _f;
      re = _r;
    }
  };
  int n,s,t,h[N],gap[N];
  vector<Edge> E[N];
  void init(int _n, int _s, int _t){
    n = _n;
    s = _s;
    t = _t;
    for (int i=0; i<N; i++)
      E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u]._PB(Edge(v,f,E[v].size()));
    E[v]._PB(Edge(u,f,E[u].size()-1));
  }
  int DFS(int u, int nf, int res=0){
    if (u == t) return nf;
    FOR(it,E[u]){
      if (h[u]==h[it->v]+1 && it->f>0){
        int tf = DFS(it->v,min(nf,it->f));
        res += tf;
        nf -= tf;
        it->f -= tf;
        E[it->v][it->re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (nf){
      if (--gap[h[u]] == 0) h[s]=n;
      gap[++h[u]]++;
    }
    return res;
  }
  int flow(int res=0){
    _SZ(h);
    _SZ(gap);
    gap[0] = n;
    while (h[s] < n)
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

4.2 Bipartite Matching

```
bool DFS(int u){
  FOR(it,E[u]){
    if (!vst[*it]){
```

```
      vst[*it]=1;
      if (match[*it] == -1 || DFS(match[*it])){
        match[*it] = u;
        match[u] = *it;
        return true;
      }
    }
  }
  return false;
}
int DoMatch(int res=0){
  MSET(match,-1);
  for (int i=1; i<=m; i++){
    if (match[i] == -1){
      memset(vst,0,sizeof(vst));
      DFS(i);
    }
  }
  for (int i=1; i<=m; i++)
    if (match[i] != -1) res++;
  return res;
}
```

4.3 SW-Mincut

```
// ——— hanhanW v1.1 ———
#include <cmath>
#include <ctime>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#define MSET(x, y) memset(x, y, sizeof(x))
#define REP(x,y,z) for(int x=y; x<=z; x++)
#define FORD(x,y,z) for(int x=y; x>=z; x--)
#define PB push_back
#define SZ size()
#define MP make_pair
#define F first
#define S second

typedef long long LL;
typedef long double LD;
typedef std::pair<int,int> PII;

const int N=514;
const int INF=2147483647>>1;

int n, m, del[N], vst[N], wei[N], rd[N][N];

PII sw(){
    MSET(vst,0);
    MSET(wei,0);
    int p1=-1,p2=-1,mx,cur=0;
    while(1){
        mx=-1;
        REP(i,1,n){
            if (!del[i] && !vst[i] && mx<wei[i]){
                cur=i;
                mx=wei[i];
            }
        }
        if (mx==-1) break;
        vst[cur]=1;
        p1=p2;
        p2=cur;
        REP(i,1,n)
            if (!vst[i] && !del[i])
                wei[i]+=rd[cur][i];
    }
    return std::MP(p1,p2);
}
void input(){
    REP(i,1,n){
        del[i]=0;
        REP(j,1,n)
            rd[i][j] = 0;
    }
    REP(i,1,m){
        int u,v,c;
        scanf("%d%d%d",&u,&v,&c);
        ++u; ++v;
        rd[u][v]+=c;
        rd[v][u]+=c;
    }
}
void solve(){
    int ans=INF;
    PII tmp;
    REP(i,1,n-1){
        tmp=sw();
        int x=tmp.F;
        int y=tmp.S;
        if (wei[y] < ans) ans=wei[y];
        del[y]=1;
        REP(j,1,n){
            rd[j][x]+=rd[j][y];
            rd[x][j]+=rd[y][j];
        }
    }
    printf("%d\n", ans);
}

int main(){
    while (~scanf("%d%d", &n, &m)){
        input();
        solve();
    }
    return 0;
}
```

4.4 Maximum Simple Graph Matching

```
const int MAX = 300;

int V, E;
int el[MAX][MAX];
int mtp[MAX];
int djs[MAX];
int bk[MAX], pr[MAX], vt[MAX];
queue<int> qu;

int ffa(int a){
  return (djs[a] == -1) ? a : djs[a] = ffa(djs[a]);
}

void djo(int a, int b){
  int fa = ffa(a), fb = ffa(b);
  if (fa != fb) djs[fb] = fa;
}

int lca(int u, int v){
  static int ts = 0;
  ts ++;
  while(1){
    if( u != -1 ){
      u = ffa(u);
      if(vt[u] == ts) return u;
      vt[u] = ts;
      if(pr[u] != -1) u = bk[pr[u]];
      else u = -1;
    }
    swap(u, v);
  }
  return u;
}

void flower(int u, int w){
  while(u != w){
    int v1 = pr[u], v2 = bk[v1];
    if(ffa(v2) != w) bk[v2] = v1;
    if(mtp[v1] == 1){
      qu.push(v1);
      mtp[v1] = 0;
    }
    if(mtp[v2] == 1){
      qu.push(v2);
```

```
      mtp[v2] = 0;
    }
    djo(v1, w);
    djo(v2, w);
    djo(u, w);
    u = v2;
  }
}
bool flow(int s){
  memset(mtp, -1, sizeof(mtp));
  while(qu.size()) qu.pop();
  qu.push(s);
  mtp[s] = 0; bk[s] = pr[s] = -1;

  while(qu.size() && pr[s] == -1){
    int u = qu.front(); qu.pop();
    for(int v=0; v<V; v++){

      if (el[u][v] == 0) continue;
      if (ffa(v) == ffa(u)) continue;

      if(pr[v] == -1){
        do{
          int t = pr[u];
          pr[v] = u; pr[u] = v;
          v = t; u = t==-1?-1:bk[t];
        }while( v != -1 );
        break;
      }else if(mtp[v] == 0){
        int w = lca(u, v);
        if(ffa(w) != ffa(u)) bk[u] = v;
        if(ffa(w) != ffa(v)) bk[v] = u;
        flower(u, w);
        flower(v, w);
      }else if(mtp[v] != 1){
        bk[v] = u;
        mtp[v] = 1;
        mtp[pr[v]] = 0;
        qu.push(pr[v]);
      }
    }
  }
  return pr[s] != -1;
}


int match(){

  memset(pr, -1, sizeof(pr));
  int a = 0;
  for (int i=0; i<V; i++){
    if (pr[i] == -1){
      if(flow(i)) a++;
      else mtp[i] = i;
    }
  }
  return a;
}
```

5 Math

5.1 ax+by=gcd

```
typedef pair<int, int> pii;

pii gcd(int a, int b){
  if(b == 0) return make_pair(1, 0);
  else{
    int p = a / b;
    pii q = gcd(b, a % b);
    return make_pair(q.second, q.first - q.second * p
        );
  }
}
```

5.2 Chinese Remainder

```
int pfn; // number of distinct prime factors
int pf[MAXNUM]; // prime factor powers
int rem[MAXNUM]; // corresponding remainder
int pm[MAXNUM];
inline void generate_primes() {
  int i,j;
  pnum=1;
  prime[0]=2;
  for(i=3;i<MAXVAL;i+=2) {
    if(nprime[i]) continue;
    prime[pnum++]=i;
    for(j=i*i;j<MAXVAL;j+=i) nprime[j]=1;
  }
}
inline int inverse(int x,int p) {
  int q,tmp,a=x,b=p;
  int a0=1,a1=0,b0=0,b1=1;
  while(b) {
    q=a/b; tmp=b; b=a-b*q; a=tmp;
    tmp=b0; b0=a0-b0*q; a0=tmp;
    tmp=b1; b1=a1-b1*q; a1=tmp;
  }
  return a0;
}
inline void decompose_mod() {
  int i,p,t=mod;
  pfn=0;
  for(i=0;i<pnum&&prime[i]<=t;i++) {
    p=prime[i];
    if(t%p==0) {
      pf[pfn]=1;
      while(t%p==0) {
        t/=p;
        pf[pfn]*=p;
      }
      pfn++;
    }
  }
  if(t>1) pf[pfn++]=t;
}
inline int chinese_remainder() {
  int i,m,s=0;
  for(i=0;i<pfn;i++) {
    m=mod/pf[i];
    pm[i]=(long long)m*inverse(m,pf[i])%mod;
    s=(s+(long long)pm[i]*rem[i])%mod;
  }
  return s;
}
```

5.3 Miller Rabin

```
long long power(long long x,long long p,long long mod
    ){
  long long s=1,m=x;
  while(p) {
    if(p&1) s=mult(s,m,mod);
    p>>=1;
    m=mult(m,m,mod);
  }
  return s;
}
bool witness(long long a,long long n,long long u,int
    t){
  long long x=power(a,u,n);
  for(int i=0;i<t;i++) {
    long long nx=mult(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(long long n,int s=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n==2;
  long long u=n-1;
  int t=0;
  // n-1 = u*2^t
```