Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos

1.1 Iterative-BS

```
int main()
{
    int l=1, r=N;
    int res=-1;

    while(l <= r)
    {
        int m = (l + r)/2;
        if(!ver(m))
        {
            l = m+1;
        }
        else
        {
            res = m;
            r = m-1;
        }
    }
    cout << res << endl;

    return 0;
}
```

2 Grafos

2.1 BFS

```
//BFS (Breadth First Search) O(V+A)

void BFS(int x)
{
    int atual, v, u;
    queue<int> fila;
    fila.push(x);

    componente[x] = valor;
    atual = 0;
    while(!fila.empty())
    {
        v = fila.front();
        fila.pop();

        for(int i = 0;i < (int)vizinhos[v].size();i++)
        {
            u = vizinhos[v][i];
            if(componente[u] == -1)
            {
                componente[u] = componente[v];
                fila.push(u);
            }
        }
    }
}
```

2.2 Find-bridges

```
#define vi vector<int>

vector< vector<int> > grafo;
vector<bool> visited;
vi t, low;
int timer=0;

void find_bridges(int v, int p=-1)
{
    visited[v] = true;
    t[v] = low[v] = timer++;
    for(int i=0;i<(int)grafo[v].size();i++)
    {
        int vert = grafo[v][i];
        if(vert == p)
            continue;
        if(visited[vert])
            low[v] = min(low[v], t[vert]);
        else
        {
            find_bridges(vert, v);
            low[v] = min(low[v], low[vert]);
            if(low[to] > t[v])
                IS_BRIDGE(v, vert);
        }
    }
}

int main()
{
    timer = 0;
    visited.assign(N+1, false);
    t.assign(N+1, 0);
    low.assign(N+1, 0);

    for(int i=0;i<N;i++)
        if(!visited[i])
            find_bridges(1);

    return 0;
}
```

2.3 Dijkstra

```
// Dijkstra - Shortest Path

#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pair<int,int> >
#define INF 0x3f3f3f3f

vector<vii> grafo;
vi distancia;
priority_queue< pii, vii, greater<pii> > fila;

void dijkstra(int k)
{
    int dist, vert, aux;
    distancia[k]=0;

    fila.push(mp(k, 0));

    while(!fila.empty())
    {
        aux=fila.top().f;
        fila.pop();

        for(int i=0; i<grafo[aux].size(); i++)
        {
            vert=grafo[aux][i].f;
            dist=grafo[aux][i].s;
            if(distancia[vert]>distancia[aux]+dist)
            {
                distancia[vert]=distancia[aux]+dist;
                fila.push(mp(vert, distancia[vert]));

            }
        }
    }
}

int main()
```

```
{
    dist.assign(N+1, INF);
    grafo.assign(N+1, vii());

    for(int i=0; i<M; i++)
    {
        cin >> a >> b >> p;
        grafo[a].pb(mp(b, p));
        grafo[b].pb(mp(a, p));
    }
}
```

2.4 Floyd-Warshall

```
// Floyd Warshall

int dist[MAX][MAX];

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

2.5 Kruskal

```
// Kruskal -  Minimum Spanning Tree

typedef struct
{
    int A, B;
    int dist;
} vertice;

vertice grafo[MAX];
int pai[MAX];

int find(int X) // Union-Find
{
    if(pai[X]==X)
        return X;
    else
        return pai[X]=find(pai[X]);
}

void join(int X, int Y)
{
    int paix = find(X);
    int paiy = find(Y);
    pai[paix]=paiy;
}

bool comp(vertice A, vertice B)
{
    return A.dist < B.dist;
}

void kruskal()
{
    for(int i=1;i<=N;i++)
        pai[i]=i;

    for(int i=1;i<=M;i++)
        cin >> grafo[i].A >> grafo[i].B >> grafo[i].dist;

    sort(grafo+1, grafo+M+1, comp);

    for(int i=1;i<M;i++)
    {
        if(find(grafo[i].A)!=find(grafo[i].B))
        {
            join(grafo[i].A, grafo[i].B);
            soma+=grafo[i].dist;
        }
    }

    cout << soma << endl;
}
```

2.6 DFS

```
//DFS (Depth First Search) O(V+A)

void DFS(int x)
{
    for(int i=0; i<(int)vizinhos[x].size(); i++)
    {
        int v = vizinhos[x][i];
        if(componente[v] == -1)
        {
            componente[v] = componente[x];
            DFS(v);
        }
    }
}
```

2.7 Represent

```
// Grafos

// List of edges

    vector< pair<int, int> > arestas;
    arestas.push_back(make_pair(1, 2));
    arestas.push_back(make_pair(1, 3));

// Adjacency Matrix

    int grafo[10][10];

    grafo[1][2] = grafo[2][1] = 1;
    grafo[1][3] = grafo[3][1] = 2;

// Adjacency List

    vector<int> vizinhos[10];

    vizinhos[1].push_back(2);
    vizinhos[1].push_back(2);
```

2.8 Prim

```
// Prim Algorithm
#define MAXN 10100
#define INFINITO 999999999

int n, m;
int distancia[MAXN];
int processado[MAXN];
vector<pii> vizinhos[MAXN];

int Prim()
{
    for(int i = 2;i <= n;i++) distancia[i] = INFINITO;
    distancia[1] = 0;

    priority_queue< pii, vector<pii>, greater<pii> > fila;
    fila.push( pii(distancia[1], 1) );

    while(1)
    {
        int davez = -1;

        while(!fila.empty())
        {
            int atual = fila.top().second;
            fila.pop();

            if(!processado[atual])
            {
                davez = atual;
                break;
            }
        }

        if(davez == -1)
            break;

        processado[davez] = true;

        for(int i = 0;i < (int)vizinhos[davez].size();i++)
        {

            int dist  = vizinhos[davez][i].first;
            int atual = vizinhos[davez][i].second;

            if( distancia[atual] > dist && !processado[atual])
            {
                distancia[atual] = dist;
                fila.push( pii(distancia[atual], atual) );
            }
        }
    }

    int custo_arvore = 0;
    for(int i = 1;i <= n;i++)
        custo_arvore += distancia[i];

    return custo_arvore;
}

int main(){

    cin >> n >> m;

    for(int i = 1;i <= m;i++){

        int x, y, tempo;
        cin >> x >> y >> tempo;

        vizinhos[x].pb( pii(tempo, y) );
        vizinhos[y].pb( pii(tempo, x) );
    }

    cout << Prim() << endl;

    return 0;
}
```

3 Geometria

3.1 Inter-Retas

```
// Intersection between lines

typedef struct
{
    int x, y;
} pnt;

bool collinear(pnt p, pnt q, pnt r)
{
    if(q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y<=max(p.y,r.y) && q.y>=min(p.y,r.y))
        return true;

    return false;
}

int orientation(pnt p, pnt q, pnt r)
{
    int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);

    if(val==0)
        return 0;
    else if(val>0)
        return 1;
    else
        return 2;
}

bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if(o1!=o2 and o3!=o4)
        return true;

    if(o1==0 && collinear(p1, p2, q1))
        return true;

    if(o2==0 && collinear(p1, q2, q1))
        return true;

    if(o3==0 && collinear(p2, p1, q2))
        return true;

    if(o4==0 && collinear(p2, q1, q2))
        return true;

    return false;

}
```

4 ED

4.1 Iterative-SegTree

```
// Segment Tree Iterativa - Range maximum query

#define N 100010

struct Segtree
{
    int t[2*N]={0};

    void build()
    {
        for(int i=N-1; i>0; i--)
            t[i]=max(t[i<<1], t[1<<1|1]);
    }

    int query(int l, int r)
    {
        int ans=0;
        for(i+=N, r+=N; l<r; l>>=1, r>>=1)
        {
            if(l&1)
                ans=max(ans, t[l++]);
            if(r&1)
                ans=max(ans, t[--r]);
        }

        return ans;
    }

    void update(int p, int value)
    {
        for(t[p+=n]=value; p>1; p>>=1)
            t[p>>1]= max(t[p], t[p^1]);
    }

};

int main()
{
    Segtree st;

    for(int i=0;i<n;i++)
    {
        cin >> aux;
        st.t[N+i]=aux; //Leaves are stored in continuous nodes with indices starting with N
    }

    st.build();
    x = st.query(inicio, fim);
    st.update(ind, value);

}
```

4.2 Recursive-SegTree

```
// Segment Tree Recursiva - Range maximum query

vector<int> val(MAX, 0);
vector<int> vet(N);

void monta(int i, int j, int no)
{
    if(i==j)
    {
        val[no]=vet[i];
        return;
    }

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    monta(i, meio, esq);
    monta(meio+1, j, dir);

    val[no]=max(val[esq], val[dir]);
}

void atualiza(int no, int i, int j, int pos, int novo_valor)
{
    if(i==j)
    {
        val[no]=novo_valor;
    }else
    {
        int esq = 2*no;
        int dir = 2*no+1;
        int meio = (i+j)/2;

        if(pos<=meio)
            atualiza(esq, i, meio, pos, novo_valor);
        else
            atualiza(dir, meio+1, j, pos, novo_valor);

        if(val[esq]>val[dir])
            val[no]=val[esq];
        else
            val[no]=val[dir];
    }
}

int consulta(int no, int i, int j, int A, int B)
{
    if(i>B || j<A)
        return -1;
    if(i>=A and j<=B)
        return val[no];

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    int resp_esq = consulta(esq, i, meio, A, B);
    int resp_dir = consulta(dir, meio+1, j, A, B);

    if(resp_dir==-1)
        return resp_esq;
    if(resp_esq==-1)
        return resp_dir;

    if(resp_esq>resp_dir)
        return resp_esq;
    else
        return resp_dir;
}

int main()
{
    monta(1, N, 1);
    atualiza(1, 1, N, pos, valor);
    x = consulta(1, 1, N, inicio, fim);

}
```

4.3 Delta-Encoding

```
// Delta encoding

for(int i=0;i<q;i++)
{
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;

for(int i=0;i<n;i++)
{
    atual += delta[i];
    v[i] += atual;
}
```

4.4 Seg-Tree-Farao

```
typedef struct
{
    pii prefix, sufix, total, maximo;
} no;

```

```
int noleft[MAX], noright[MAX]; //Guarda os valores dos nos para que nao sejam calculados novamente nas querys
int v[MAX];
no arvore[MAX];

pii somar(pii a, pii b) // une pairs
{
    return mp(a.f+b.f, a.s+b.s);
}

no une(no l, no r)
{
    if(l.total.s==0)
        return r;
    if(r.total.s==0)
        return l;

    no m;

    m.prefix = max(l.prefix, somar(l.total, r.prefix)); //prefixo
    m.sufix = max(r.sufix, somar(r.total, l.sufix)); //sufixo
    m.total = somar(l.total, r.total); //Soma de todos os elementos da subarvore
    m.maximo = max(max(l.maximo, r.maximo), somar(l.sufix, r.prefix)); //Resultado para cada subarvore

    return m;
}

no makenozero()
{
    no m;
    m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
    return m;
}

no makeno(int k)
{
    no m;
    m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
    return m;
}

void monta(int n)
{
    if(noleft[n]==noright[n])
    {
        arvore[n]=makeno(v[noleft[n]]);
        return;
    }

    int mid = (noleft[n]+noright[n])/2;
    noleft[2*n]=noleft[n]; noright[2*n]=mid;
    noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];

    monta(2*n);
    monta(2*n+1);

    arvore[n]=une(arvore[2*n], arvore[2*n+1]);
}

no busca(int n, int esq, int dir)
{
    if(noleft[n]>=esq and noright[n]<=dir)
        return arvore[n];
    if(noright[n]<esq or noleft[n]>dir)
        return makenozero();

    return une(busca(2*n, esq, dir),busca(2*n+1, esq, dir));
}

int main()
{
    int T, N, Q, A, B;
    no aux;

    scanf("%d", &T);

    while(T--)
    {
        scanf("%d", &N);
        for(int i=1;i<=N;i++)
            scanf("%d", &v[i]); //Elementos da arvore

        noleft[1]=1;noright[1]=N;
        monta(1);

        cin >> Q;
        while(Q--)
        {
            scanf("%d%d", &A, &B); //Intervalo da query
            aux = busca(1, A, B);
            printf("%d %d\n", aux.maximo.f, aux.maximo.s);
        }
    }


    return 0;
}
```

4.5 BIT-2D

```
// BIT 2D

int bit[MAX][MAX];

int sum(int x, int y)
{
    int resp=0;

    for(int i=x;i>0;i-=i&-i)
        for(int j=y;j>0;j-=j&-j)
            resp+=bit[i][j];

    return resp;
}

void update(int x, int y, int delta)
{
    for(int i=x;i<MAX;i+=i&-i)
        for(int j=y;j<MAX;j+=j&-j)
            bit[i][j]+=delta;
}

int query(int x1, y1, x2, y2)
{
    return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum(x1,y1);
}
```

4.6 BIT

```
// (BIT) Fenwick Tree

int bit[MAX];

int soma(int x)
{
    int resp=0;

    // for(int i=x;i>0;i-=i&-i)
    //      resp+=bit[i];

    while(x > 0)
    {
        resp += bit[x];
        x -= (x & -x);
    }

    return resp;
}

int query(int L, R)
{
    return soma(R)-soma(L);
}

void update(int x, int v)
{
    // for(;x<=n;x+=x&-x)
    //        bit[x] += v;

    while(x <= N)
    {
        bit[x] += v;
        x += (x & -x);
    }
}
```

4.7 Union-Find

```
// Union-Find Functions

int pai[MAX], peso[MAX];

int find(int aux)
{
    if(pai[aux]==aux)
        return aux;
    else
        return pai[aux]=find(pai[aux], pai);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);

    if(pesos[x]<pesos[y])
        pai[x] = y;
    else if(pesos[x]>pesos[y])
        pai[y] = x;
    else if(pesos[x]==pesos[y])
    {
        pai[x] = y;
        pesos[y]++;
    }
}

int main()
{
    for(int i=1;i<=N;i++)
        pai[i]=i;
}
```

5 Math

5.1 Linear-Diophantine-Equation

```
// Linear Diophantine Equation
int gcd(int a, int b, int &x, int &y)
{
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g)
{
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g)
        return false;

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

//  All solutions
//  x = x0 + k*b/g
//  y = y0 - k*a/g
```

5.2 Factorization-sqrt

```
// Factorization of a number in sqrt(n)

int main()
{
    ll N;
    vector<int> div;

    cin >> N;

    for(ll i=2;i*i<=N;i++)
    {
        if(N%i==0)
        {
            vet.pb(i);
            while(N%i==0)
                N/=i;
        }
    }
    if(N!=1)
        vet.pb(N);

    return 0;
}
```

5.3 Modular-Exponentiation

```
// Modular exponentiaion - (x^y)%mod in O(log y)
ll power(ll x, ll y, ll mod)
{
    ll res = 1;
    x%=mod;

    while(y)
    {
        if(y&1)
            res=(res*x)%mod;

        y=y>>1;
        x=(x*x)%mod;
    }
    return res;
}
```

5.4 Miller-Habin

```
#include <bits/stdc++.h>
#define mod 1000000007
#define Pi 3.14159265358979311599796346854
#define INF 0x3f3f3f3f
#define MAX 1000010
#define f first
#define s second
#define ll long long
#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pii >
#define sws ios_base::sync_with_stdio(false);cin.tie(NULL)
#define forn(i, n) for(int i=0; i<(int)(n); i++)
#define mdc(a, b) (__gcd((a), (b)))
#define mmc(a, b) (((a)/__gcd(a, b)) * b)
#define endl '\n'
#define teto(a, b) (a+b-1)/b

using namespace std;

ll llrand()
{
    ll tmp = rand();
    return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
{
    return (a + b)%c;
}

ll mul(ll a, ll b, ll c)
{
    ll ans = 0;
    while(b)
    {
        if(b & 1)
            ans = add(ans, a, c);
        a = add(a, a, c);
        b /= 2;
    }
    return ans;
}

ll rho(ll n)
{
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand()%n;
        c = llrand()%n;
        y = x, k = 4;
        do{
            if(++i == k)
            {
                y = x;
                k *= 2;
            }
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }
        while(d == 1);
    }
    while(d == n);

    return d;
}

ll fexp(ll a, ll b, ll c)
{
    ll ans = 1;
    while(b)
    {
        if(b & 1)
            ans = mul(ans, a, c);
        a = mul(a, a, c);
        b /= 2;
    }
    return ans;
}

bool rabin(ll n)
{
    if(n <= 1)
        return 1;
    if(n <= 3)
        return 1;

    ll s=0, d=n-1;
    while(d%2==0)
    {
        d/=2;
        s++;
    }

    for(int k = 0; k < 64*4; k++)
    {
        ll a = (llrand()%(n - 3)) + 2;
        ll x = fexp(a, d, n);
        if(x != 1 and x != n-1)
        {
            for(int r = 1; r < s; r++)
            {
                x = mul(x, x, n);
                if(x == 1)
                    return 0;
                if(x == n-1)
                    break;
            }
            if(x != n-1)
                return 0;
        }
    }

    return 1;
}


int main()
{
    //sws;
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);

    ll N, resp;
    vector<ll> div;

    cin >> N;
    resp = N;

    while(N>1 and !rabin(N))
    {
        ll d = rho(N);
        if(!rabin(d))
```

```
            continue;
        div.pb(d);
        while(N%d==0)
            N/=d;
    }
    if(N!=resp and N!=1)
        div.pb(N);


    if(div.empty())
        cout << resp << endl;
    else
    {
        for(int i=0;i<(int)div.size();i++)
            resp = __gcd(resp, div[i]);

        cout << resp << endl;
    }

    return 0;

}
```

5.5 Pollard-Rho

```
// Pollard Rho Algorithm

#include <bits/stdc++.h>
#define ll long long

using namespace std;

ll llrand()
{
    ll tmp = rand();
    return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
{
    return (a + b)%c;
}

ll mul(ll a, ll b, ll c)
{
    ll ans = 0;
    while(b)
    {
        if(b & 1)
            ans = add(ans, a, c);
        a = add(a, a, c);
        b /= 2;
    }
    return ans;
}

ll rho(ll n)
{
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand()%n;
        c = llrand()%n;
        y = x, k = 4;
        do{
            if(++i == k)
            {
                y = x;
                k *= 2;
            }
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }
        while(d == 1);
    }
    while(d == n);

    return d;
}

int main()
{
    srand(time(0));

    ll N;
    cin >> N;

    ll div = rho(N);
    cout << div << " " << N/div << endl;


    // Finding all divisors

    vector<ll> div;

    while(N>1 and !rabin(N))
    {
        ll d = rho(N);
        if(!rabin(d))
            continue;
        div.pb(d);
        while(N%d==0)
            N/=d;
    }
    if(N!=resp and N!=1)
        div.pb(N);

    return 0;

}
```

5.6 Verif-primo

```
// prime verification sqrt(N)

bool eh_primo(long long N)
{
    if(N==2)
        return true;
    else if(N==1 or N%2==0)
        return false;
    for(long long i=3;i*i<=N;i+=2)
        if(N%i==0)
            return false;
    return true;
}
```

5.7 Crivo

```
// Sieve of Eratosthenes

int N;
vector<bool> primos(100010, true);
cin >> N;

primos[0]=false;
primos[1]=false;

for(int i=2;i<=N;i++)
    if(primos[i])
        for(int j=i+i; j<=N; j+=i)
            primos[j]=false;
```

5.8 formulas

```
int sum_x2(ll N)
{
    return (2*N*N*N + 3*N*N + N)/6;
}
```

5.9 FFT-golfbot

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

using namespace std;

const int N = (1<<19);
const double two_pi = 4 * acos(0);

struct cpx
{
    cpx(){}
    cpx(double aa): a(aa){}
    cpx(double aa,double bb):a(aa),b(bb){}
    double a;
    double b;
    double modsq(void) const
    {
        return a*a+b*b;
    }
    cpx bar(void) const
    {
        return cpx(a,-b);
    }
};

cpx b[N+100];
cpx c[N+100];
cpx B[N+100];
cpx C[N+100];
int a[N+100];
int x[N+100];
double coss[N+100], sins[N+100];
int n,m,p;

cpx operator +(cpx a,cpx b)
{
    return cpx(a.a+b.a,a.b+b.b);
}

cpx operator *(cpx a,cpx b)
{
    return cpx(a.a*b.a-a.b*b.b,a.a*b.b+a.b*b.a);
}

cpx operator /(cpx a,cpx b)
{
    cpx r = a*b.bar();
    return cpx(r.a/b.modsq(),r.b/b.modsq());
}

cpx EXP(int i,int dir)
{
    return cpx(coss[i],sins[i]*dir);
}

void FFT(cpx *in,cpx *out,int step,int size,int dir)
{
    if(size<1) return;
    if(size==1)
    {
        out[0]=in[0];
        return;
    }
    FFT(in,out,step*2,size/2,dir);
    FFT(in+step,out+size/2,step*2,size/2,dir);
    for(int i=0;i<size/2;++i)
    {
        cpx even=out[i];
        cpx odd=out[i+size/2];
        out[i] = even+EXP(i*step,dir)*odd;
        out[i+size/2]=even+EXP((i+size/2)*step,dir)*odd;
    }
}

int main()
{
    for(int i=0;i<=N;++i)
    {
        coss[i]=cos(two_pi*i/N);
        sins[i]=sin(two_pi*i/N);
    }
    while(cin >> n) // Numero de tacadas possiveis
    {
        fill(x,x+N+100,0);
        fill(a,a+N+100,0);
        for(int i=0;i<n;++i)
        {
            cin >> p; // Distancia das tacadas
            x[p]=1;
        }
        for(int i=0;i<N+100;++i)
        {
            b[i]=cpx(x[i],0);
        }
        cin >> m; // Querys
        for(int i=0;i<m;++i)
        {
            cin >> a[i]; // Distancia da query
        }
        FFT(b,B,1,N,1);
        for(int i=0;i<N;++i)
            C[i]=B[i]*B[i];
        FFT(C,c,1,N,-1);
        for(int i=0;i<N;++i)
            c[i]=c[i]/N;
        int cnt=0;
        for(int i=0;i<m;++i)
            if(c[a[i]].a>0.5 || x[a[i]])
                cnt++;
        cout << cnt << endl;
    }
    return 0;
}
```

5.10 Modular-Factorial

```
// C++ program to comput n! % p using Wilson's Theorem
#include <bits/stdc++.h>
using namespace std;

int power(int x, unsigned int y, int p)
{
    int res = 1;
    x = x % p;

    while(y > 0)
    {
        if(y & 1)
            res = (res * x) % p;

        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}

int modInverse(int a, int p)
{
    return power(a, p-2, p);
}

int modFact(int n, int p)
{
    if (p <= n)
        return 0;

    int res = (p - 1);

    for(int i = n + 1; i < p; i++)
        res = (res * modInverse(i, p)) % p;
    return res;
}

int main()
{
    int n = 25, p = 29;
    cout << modFact(n, p);
    return 0;
}
```

5.11 Kamenetsky

```
// Number of digits in n! O(1)

#define Pi 3.14159265358979311599796346854
#define Eul 2.71828182845904509079559829842

long long findDigits(int n)
{
    double x;

    if (n < 0)
        return 0;
    if (n == 1)
        return 1;

    x = ((n * log10(n / euler) + log10(2 * Pi * n) /2.0));

    return floor(x) + 1;
}
```

6 Misc

6.1 Bitwise

```
// Bitwise

    unsigned char a = 5, b = 9; // a = (00000101), b = (00001001)

    AND -          a&b   // The result is 00000001 (1)
    OR -           a|b   // The result is 00001101 (13)
    XOR -          a^b   // The result is 00001100 (12)
    NOT -          ~a    // The result is 11111010 (250)
    Left shift -   b<<1  // The result is 00010010 (18)
    Right shift -  b>>1  // The result is 00000100 (4)

    // Exchange two int variables

        a^=b;
        b^=a;
        a^=b;

    // Even or Odd

        (x & 1)? printf("Odd"): printf("Even");

    // Turn on the j-th bit

        int S = 34; //(100010)
        int j = 3;

        S = S | (1<<j);

    // Turn off the j-th bit

        int S = 42; //(101010)
        int j = 1;

        S &= ~(1<<j)

        S == 40 //(101000)

    // Check the j-th element

        int S = 42; //(101010)
        int j = 3;

        T = S & (1<<j); // T = 0

    // Exchange o j-th element

        S ^= (1<<j)

    // Position of the first bit on

        T = (S & (-S))
        T -> 4 bit ligado //(1000)

    // Most significant digit of N


        double K = log10(N);
        K = K - floor(K);
        int X = pow(10, K);

    // Number of digits in N

        X =floor(log10(N)) + 1;

    // Power of two

        bool isPowerOfTwo(int x)
        {
            return x && (!(x&(x-1)));
        }
```

7 Strings

7.1 KMP

```
//KMP Algorithm

#include <bits/stdc++.h>

// Fills lps[] for given patttern pat[0..M-1]
void computeLPSArray(char* pat, int M, int* lps)
{
    // length of the previous longest prefix suffix
    int len = 0;

    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
```

```c
    int i = 1;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            // This is tricky. Consider the example.
            // AAACAAAA and i = 7. The idea is similar
            // to search step.
            if (len != 0) {
                len = lps[len - 1];

                // Also, note that we do not increment
                // i here
            }
            else // if (len == 0)
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}

// Prints occurrences of txt[] in pat[]
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // create lps[] that will hold the longest prefix suffix
    // values for pattern
    int lps[M];

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    int j = 0; // index for pat[]
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }

        if (j == M) {
            printf("Found pattern at index %d ", i - j);
            j = lps[j - 1];
        }

        // mismatch after j matches
        else if (i < N and pat[j] != txt[i]) {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}


// Driver program to test above function
int main()
{
    char txt[] = "ABABDABACDABABCABAB";
    char pat[] = "ABABCABAB";
    KMPSearch(pat, txt);
    return 0;
}
```