目录

[AC自动机 2](#_Toc514452338)

[二分匹配 4](#_Toc514452339)

[BM算法 4](#_Toc514452340)

[卡特兰大数 5](#_Toc514452341)

[迪杰斯特拉 6](#_Toc514452342)

[欧拉函数 7](#_Toc514452343)

[分解质因数 7](#_Toc514452344)

[得到随机数 8](#_Toc514452345)

[图模板类 8](#_Toc514452346)

[Kmp算法 9](#_Toc514452347)

[LCA算法 9](#_Toc514452348)

[LIS算法 10](#_Toc514452349)

[LUCAS求选择数 11](#_Toc514452350)

[最大流 12](#_Toc514452351)

[MCMF 13](#_Toc514452352)

[RMQ算法 15](#_Toc514452353)

[SPFA 15](#_Toc514452354)

[后缀数组 16](#_Toc514452355)

[Tarjan算法 18](#_Toc514452356)

# AC自动机

struct node{

node \*\_next[MAXC],\*\_fail;

ll flag,vis;

void init(){

memset(\_next,0,sizeof \_next);

flag = 0; \_fail = 0; vis = -1;

}

node(){init();}

void clear(){

for(ll i=0;i<MAXC;i++)

if(\_next[i]){

\_next[i]->clear();

delete \_next[i];

}

init();

}

node \*&next(char c){

return \_next[c-CHAR0];

}

void add\_word(char \*word,ll id){

if(\*word==0)flag = id;

else{

if(!next(\*word))next(\*word) = new node;

next(\*word)->add\_word(word+1,id);

}

}

void AC\_init(){

queue<node\*> que;

\_fail = this;

for(ll i=0;i<MAXC;i++){

if(\_next[i]){

\_next[i]->\_fail = this;

que.push(\_next[i]);

}

}

node \*f,\*tp;

while(!que.empty()){

f = que.front(); que.pop();

for(ll i=0;i<MAXC;i++){

if(f->\_next[i]){

tp = f->\_fail;

while(tp){

if(tp->\_next[i]){

f->\_next[i]->\_fail = tp->\_next[i];

break;

}

if(tp == this){

tp = 0;

break;

}

tp = tp->\_fail;

}

if(!tp)f->\_next[i]->\_fail = this;

que.push(f->\_next[i]);

}

}

}

}

void match(char \*s,ll len){

node \*p,\*tp = this,\*np;

for(ll i=0;i<=len;i++){

np = 0;

p = tp;

while(p != this){

if(p->flag)times[p->flag] ++;

p = p->\_fail;

}

while(tp != this){

if(i!=len && !np && tp->next(s[i])){

np = tp->next(s[i]);

break;

}

tp = tp->\_fail;

}

if(i!=len && !np && tp->next(s[i]))

np = tp->next(s[i]);

if(np)tp = np;

}

}

}\_root;

# 二分匹配

bool binMatch(ll x,ll \*os,bool \*used,const graph &G){

ll \*head = G.head; edge \*E = G.E;

for(ll i=head[x];i!=-1;i=E[i].next){

ll s = E[i].to;

if(!used[s]){

used[s]=1;

if(os[s]==-1||binMatch(os[s],os,used,G)){

os[s]=x;

return 1;

}

}

}

return 0;

}

ll mainBinMatch(ll nodecon,ll \*other\_side,const graph &G){

ll res = 0;

static bool \*used = new bool[G.\_maxn];

memset(other\_side,-1,sizeof(ll)\*G.\_maxn);

for(ll i=1;i<=nodecon;i++){

memset(used,0,sizeof(bool)\*G.\_maxn);

if(binMatch(i,other\_side,used,G))res ++;

}

return res;

}

# BM算法

#define BM\_TYPE char

struct BM{

const static int MAXC = 26,CHAR0 = 'a';

ll \*bc = 0,\*ss = 0,\*gs = 0,sizeP = 0;

BM\_TYPE \*P = 0;

void buildBC(){

bc = new ll[MAXC];

for(ll j=0;j<MAXC;j++)bc[j] = -1;

for(ll j=0;j<sizeP;j++)bc[P[j]-CHAR0] = j;

}

void buildSS(){

ss = new ll[sizeP];

ss[sizeP-1] = sizeP;

for(ll lo=sizeP-1,hi=sizeP-1,j=lo-1;j>=0;j--){

if((lo<j) && (ss[sizeP-hi+j-1]<=j-lo))

ss[j] = ss[sizeP-hi+j-1];

else{

hi = j; lo = min(lo,hi);

while((0<=lo) && (P[lo]==P[sizeP-hi+lo-1]))

lo--;

ss[j] = hi-lo;

}

}

}

void buildGS(){

gs = new ll[sizeP];

for(ll j=0;j<sizeP;j++)gs[j] = sizeP;

for(ll i=0,j=sizeP-1;j>=0;j--)

if(j+1 == ss[j])

while(i < sizeP-j-1)

gs[i++] = sizeP-j-1;

for(ll j=0;j<sizeP-1;j++)

gs[sizeP-ss[j]-1] = sizeP-j-1;

}

void setP(BM\_TYPE \*\_P,ll \_sizeP){

if(bc)delete []bc;

if(gs)delete []gs;

if(ss)delete []ss;

P = \_P; sizeP = \_sizeP;

buildBC(); buildSS(); buildGS();

}

ll match(const BM\_TYPE \*T,ll sizeT){

ll i = 0;

while(sizeT >= i+sizeP){

ll j = sizeP-1;

while(P[j] == T[i+j])

if(0 > --j)break;

if(0 > j)break;

else i += max(gs[j],j-bc[T[i+j]-CHAR0]);

}

return i;

}

};

# 卡特兰大数

ll a[105][105],b[105];

void catalan(ll N){

ll len,carry,temp;

a[1][0] = b[1] = 1;

len = 1;

for(ll i=2;i<=N;i++){

for(ll j=0;j<len;j++)

a[i][j] = a[i-1][j]\*(4\*(i-1)+2);

carry = 0;

for(ll j=0;j<len;j++){

temp = a[i][j] + carry;

a[i][j] = temp % 10;

carry = temp / 10;

}

while(carry){

a[i][len++] = carry % 10;

carry /= 10;

}

carry = 0;

for(ll j=len-1;j>=0;j--){

temp = carry\*10 + a[i][j];

a[i][j] = temp/(i+1);

carry = temp%(i+1);

}

while(!a[i][len-1])len --;

b[i] = len;

}

}

# 迪杰斯特拉

#define INF 0x3f3f3f3f

void dij(ll sta,ll \*d,graph G){

ll \*head = G.head; edge \*E = G.E;

memset(d,INF,sizeof(ll)\*G.\_maxn);

d[sta] = 0;

priority\_queue<P2,vector<P2>,greater<P2> >que;

que.push(P2(0,sta));

ll t,s,p,l;

while(!que.empty()){

t = que.top().second; p = que.top().first;

que.pop();

if(p > d[t])continue;

for(ll i=head[t];i!=-1;i=E[i].next){

s = E[i].to; l = E[i].len;

if(p+l < d[s]){

d[s] = p+l;

que.push(P2(d[s],s));

}

}

}

}

# 欧拉函数

void euler1(ll \*euler,ll \_n){

memset(euler,0,sizeof(ll)\*\_n);

for(ll i=1;i<=\_n;i++)euler[i] = i;

for(ll i=2;i<=\_n;i++)

if(euler[i] == i)

for(ll j=i;j<=\_n;j+=i)

euler[j] = euler[j]/i\*(i-1);

}

ll euler2(ll n){

ll res=n,a=n;

for(ll i=2;i\*i<=a;i++){

if(a%i==0){

res=res/i\*(i-1);

while(a%i==0) a/=i;

}

}

if(a>1) res=res/a\*(a-1);

return res;

}

# 分解质因数

ll get\_prime\_factors(ll \_n,ll \*res){//res中会被存放上质因数

ll n2 = \_n,rid = 0;

if(\_n<2)return 0;

for(ll i=2;i\*i<=n2;i++)if(n2%i == 0){

while(n2%i==0)n2=n2/i;

res[rid++] = i;

}

if(n2!=1)res[rid++] = n2;

return rid; //质因数数目

}

# 得到随机数

ll get\_rand(ll lo,ll hi){

static bool has\_init = 0;

if(!has\_init){

srand(time(0));

has\_init = 1;

}

return rand()%(hi-lo+1)+lo;

}

# 图模板类

struct edge{

ll to,next,len;

edge(ll a=0,ll b=-1,ll c=0){

to = a; next = b; len = c;

}

};

struct graph{

ll \*head,eid,\_maxn;

edge \*E;

graph(ll maxn,ll maxe){

head = new ll[maxn+1];

E = new edge[maxe+1];

\_maxn = maxn;

init();

}

~graph(){

delete[] head;

delete[] E;

}

void init(){

memset(head,-1,sizeof(ll)\*\_maxn);

eid = 0;

}

void add(ll h,ll t,ll len=0){

E[eid] = edge(t,head[h],len);

head[h] = eid ++;

}

};

# Kmp算法

template<typename \_type>struct KMP{

\_type \*P; ll lenP,\*next = 0;

void build\_next(){

ll j = 0;

next = new ll[lenP+1];

ll t = next[0] = -1;

while(j < lenP){

if(0>t || P[j]==P[t]){

j++; t++;

next[j] = (P[j]!=P[t]?t:next[t]);

//next[j] = t;

}else t = next[t];

}

}

void setP(\_type \*\_P,ll \_lenP){

if(next)delete[]next;

P = \_P; lenP = \_lenP;

build\_next();

}

ll match(\_type \*T,ll lenT,ll r = 0){

build\_next();

ll i=0,j=0;

\_type \*TT = T+r; lenT -= r;

while(j<lenP && i<lenT){

if(0>j || TT[i]==P[j]){

i++; j++;

}else j = next[j];

}

lenT += r;

return i-j+r;

}

};

# LCA算法

struct LCA\_rmq{

RMQ \*r;

ll \*n\_ft,\*t\_n,\*n\_p,\*n\_r,\*r\_n,dfst,nid,maxn;

LCA\_rmq(ll Maxn,ll maxl){

maxn = Maxn,r = new RMQ(maxl);

n\_ft = new ll[maxn],n\_p = new ll[maxn];

n\_r = new ll[maxn],r\_n = new ll[maxn];

t\_n = new ll[maxl];

}

~LCA\_rmq(){

delete[] n\_ft;

delete[] n\_p;

delete[] n\_r;

delete[] r\_n;

delete[] t\_n;

delete r;

}

void init(ll nodecon,const graph &T){

dfst = nid = 0;

memset(n\_r,-1,sizeof(ll)\*maxn);

for(ll i=1;i<=nodecon;i++)if(n\_r[i] == -1){

dfs(i,-1,0,T);

t\_n[dfst ++] = -1;

}

r->init(t\_n,dfst);

}

void dfs(ll r,ll f,ll p,const graph &T){

ll \*head = T.head; edge \*E = T.E;

n\_r[r] = nid,r\_n[nid++] = r;

n\_p[r] = p,n\_ft[r] = dfst,t\_n[dfst ++] = n\_r[r];

for(ll i=head[r];i!=-1;i=E[i].next){

if(E[i].to == f)continue;

dfs(E[i].to,r,p+E[i].len,T);

t\_n[dfst ++] = n\_r[r];

}

}

ll find\_path(ll a,ll b){

if(n\_ft[a] > n\_ft[b])swap(a,b);

ll pa = n\_p[a],pb = n\_p[b];

ll lca = r->find(n\_ft[a],n\_ft[b]+1);

if(lca == -1)return -1;

ll pc = n\_p[r\_n[lca]];

return pa+pb-2\*pc;

}

};

# LIS算法

template<class \_type>struct LIS{

static bool comp(const \_type &a,const \_type &b){

return a <= b;

}

ll getLisCon(\_type \*arr,ll len){

\_type l[len],\*\_t,\*\_end = l;

for(ll i=0;i<len;i++){

\_t = lower\_bound(l,\_end,arr[i],LIS::comp);

if(\_t == \_end)\_end ++;

\*\_t = arr[i];

}

return \_end - l;

}

};

# LUCAS求选择数

ll quick\_mod(ll a, ll b){

ll ans = 1;

a %= MOD;

while(b){

if(b & 1){

ans = ans \* a % MOD;

b--;

}

b >>= 1;

a = a\*a%MOD;

}

return ans;

}

ll C(ll n, ll m){

if(m > n)return 0;

ll ans = 1;

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

ll a = (n+i-m)%MOD;

ll b = i%MOD;

ans = ans\*(a\*quick\_mod(b,MOD-2)%MOD)%MOD;

}

return ans;

}

ll Lucas(ll n, ll m){

if(m == 0) return 1;

return C(n%MOD,m%MOD)\*Lucas(n/MOD,m/MOD)%MOD;

}

ll fact[100005];

void init(){

fact[0]=1;

for(ll i=1;i<=MOD;i++)

fact[i]=fact[i-1]\*i%MOD;

}

ll pow\_m(ll a,ll k,ll p){

ll ans=1;

ll tmp=a%p;

while(k){

if(k&1)ans=ans\*tmp%p;

tmp=tmp\*tmp%p;

k>>=1;

}

return ans;

}

ll C2(ll n,ll m,ll p){

if(m>n)return 0;

return fact[n]\*pow\_m(fact[m]\*fact[n-m]%p,p-2,p);

}

ll Lucas(ll n,ll m,ll p){

ll ans=1;

while(n&&m){

ans=ans\*C2(n%p,m%p,p)%p;

n/=p;

m/=p;

}

return ans;

}

# 最大流

ll n,flow[MAXN][MAXN],pre[MAXN],use[MAXN];

bool bfs(){

queue<ll> que; que.push(1);

ll tn;

memset(pre,0,sizeof pre);

memset(use,0,sizeof use);

while(!que.empty()){

tn = que.front(); que.pop();

if(tn == n)return 1;

for(ll i=1;i<=n;i++){

if(!use[i] && flow[tn][i]){

use[i] = 1;

pre[i] = tn;

que.push(i);

}

}

}

return 0;

}

ll max\_flow(){

ll minf,res = 0;

while(bfs()){

minf = 1e9;

for(ll i=n;i!=1;i=pre[i])

minf = min(minf,flow[pre[i]][i]);

for(ll i=n;i!=1;i=pre[i]){

flow[pre[i]][i] -= minf;

flow[i][pre[i]] += minf;

}

res += minf;

}

return res;

}

# MCMF

struct MCMF{

ll n,m;

struct Edge{

ll from, to, cap, flow, cost;

Edge(ll a,ll b,ll c,ll d,ll e){

from = a;to = b;cap = c;flow = d;cost = e;

}

};

vector<Edge> edge;

vector<ll> g[MAXN];

bool inq[MAXN];

ll d[MAXN],p[MAXN],a[MAXN];

void init(ll n){

this->n = n;

for(ll i=1;i<=n;i++)g[i].clear();

edge.clear();

}

void add\_edge(ll from,ll to,ll cap,ll cost){

edge.push\_back(Edge(from,to,cap,0,cost));

edge.push\_back(Edge(to,from,0,0,-cost));

m = edge.size();

g[from].push\_back(m-2);

g[to].push\_back(m-1);

}

bool spfa(ll s,ll t,ll& flow,ll& cost){

memset(d,0x3f,sizeof d);

memset(inq,0,sizeof inq);

d[s] = 0, inq[s] = true, p[s] = 0, a[s] = INF;

queue<ll> q;

q.push(s);

while(!q.empty()){

ll u = q.front();q.pop();

inq[u] = false;

ll sz = g[u].size();

for(ll i=0;i<sz;i++){

Edge& e = edge[g[u][i]];

if(e.cap > e.flow && d[e.to] > d[u]+e.cost){

d[e.to] = d[u]+e.cost;

p[e.to] = g[u][i];

a[e.to] = min(a[u], e.cap-e.flow);

if(!inq[e.to]) q.push(e.to), inq[e.to] = true;

}

}

}

if(INF == d[t]) return false;

flow += a[t];

cost += (ll)d[t]\*(ll)a[t];

for(ll u = t; u != s; u = edge[p[u]].from){

edge[p[u]].flow += a[t];

edge[p[u]^1].flow -= a[t];

}

return true;

}

ll mincostmaxflow(ll s, ll t, ll & cost){

ll flow = 0;

cost = 0;

while(spfa(s,t,flow,cost));

return flow;

}

}mcmf;

# RMQ算法

struct RMQ{

ll \*\*st,maxl,maxh; double log2;

RMQ(const ll &Maxl){

log2 = log(2);

maxl = Maxl;

maxh = log(maxl)/log2;

st = new ll\*[maxh+1];

for(ll i=0;i<=maxh;i++)

st[i] = new ll[maxl+1-(1<<i)];

}

~RMQ(){

for(ll i=0;i<=maxh;i++)delete[] st[i];

delete[] st;

}

void init(ll \*arr,const ll &len){

ll col = log(len)/log2;

for(ll i=0;i<len;i++)st[0][i] = arr[i];

for(ll i=1;i<=col;i++)for(ll j=0;j<=len-(1<<i);j++)

st[i][j] = comp(st[i-1][j],st[i-1][j+(1<<(i-1))]);

}

ll find(ll lo,ll hi){ // zuo bi you kai

ll col = log(hi-lo)/log2;

return comp(st[col][lo],st[col][hi-(1<<col)]);

}

ll comp(ll a,ll b){

return a<b?a:b;

}

};

# SPFA

bool spfa(ll sta,ll \*d,ll nodecon,graph G){

memset(d,INF,sizeof(ll)\*G.\_maxn);

ll \*head = G.head; edge \*E = G.E;

queue<ll> que; que.push(sta);

static bool \*in = new bool[G.\_maxn];

memset(in,0,sizeof(bool)\*G.\_maxn);

static ll \*times = new ll[G.\_maxn];

memset(times,0,sizeof(ll)\*G.\_maxn);

d[sta] = 0; in[sta] = 1; times[sta] = 1;

while(!que.empty()){

ll f = que.front();que.pop();

in[f] = 0;

for(ll i=head[f];i!=-1;i=E[i].next){

ll s = E[i].to;

if(d[f]+E[i].len < d[s]){

d[s] = d[f]+E[i].len;

if(!in[s]){

in[s] = 1;

times[s] ++;

if(times[s] > nodecon)return 0;

que.push(s);

}

}

}

}

return 1;

}

# 后缀数组

struct suffix\_array{

ll \*Rank,\*suffArr,\*height,\*bCon,\*tArr1,\*tArr2,\*tArr3;

suffix\_array(ll maxl){

Rank = new ll[maxl+7];

suffArr = new ll[maxl+7];

height = new ll[maxl+7];

bCon = new ll[maxl+7];

tArr1 = new ll[maxl+7];

tArr2 = new ll[maxl+7];

tArr3 = new ll[maxl+7];

}

~suffix\_array(){

delete[] Rank;

delete[] bCon;

delete[] tArr1;

delete[] tArr2;

delete[] tArr3;

}

ll updata(ll base,ll len){

if(base > len)return -1;

memset(bCon,0,sizeof(ll)\*(len+2));

for(ll i=0;i<len;i++)

bCon[(i+base<len?Rank[i+base]:0)+1]++;

//for(ll i=0;i<=len+1;i++)cout<<bCon[i]<<(i==len+1?"\n":" ");

for(ll i=1;i<=len+1;i++)

bCon[i] += bCon[i-1];

for(ll i=0;i<len;i++){

ll nv = (i+base<len?Rank[i+base]:0);

tArr1[bCon[nv]++] = i;

}

//for(ll i=0;i<len;i++)cout<<tArr1[i]<<(i==len-1?"\n":" ");

memset(bCon,0,sizeof(ll)\*(len+2));

for(ll i=0;i<len;i++)bCon[Rank[i]+1]++;

for(ll i=1;i<=len+1;i++)bCon[i] += bCon[i-1];

for(ll i=0;i<len;i++)

tArr2[bCon[Rank[tArr1[i]]]++] = tArr1[i];

//for(ll i=0;i<len;i++)cout<<tArr2[i]<<(i==len-1?"\n":" ");

ll con = 0,r1,r2,v11,v12,v21,v22;

for(ll i=0;i<len;i++){

if(i == 0){

tArr3[tArr2[i]] = ++con;

continue;

}

r1 = tArr2[i],r2 = tArr2[i-1];

v11 = Rank[r1],v12 = Rank[r2];

v21 = r1+base<len?Rank[r1+base]:0;

v22 = r2+base<len?Rank[r2+base]:0;

if(v11==v12 && v21==v22)

tArr3[tArr2[i]] = tArr3[tArr2[i-1]];

else tArr3[tArr2[i]] = ++con;

}

//for(ll i=0;i<len;i++)cout<<tArr3[i]<<(i==len-1?"\n\n":" ");

memcpy(Rank,tArr3,sizeof(ll)\*len);

return base<<1;

}

void setArr(const char \*arr,ll len,bool getHeight = 0){

ll has[256],con = 0,base = 1,k = 0,j;

memset(has,0,sizeof has);

for(ll i=0;i<len;i++)has[(ll)arr[i]] = 1;

for(ll i=0;i<256;i++)if(has[i])has[i] = ++con;

for(ll i=0;i<len;i++)Rank[i] = has[(ll)arr[i]];

while((base=updata(base,len)) != -1);

for(ll i=0;i<len;i++)suffArr[Rank[i]] = i;

if(!getHeight)return;

for(ll i=0; i<len; i++){

if(k)k--;

j=suffArr[Rank[i]-1];

while(i+k<len&&j+k<len&&arr[i+k]==arr[j+k])k++;

height[Rank[i]-1]=k;

}

}

};

# Tarjan算法

ll vid,\*dfn,\*low,\*insta,\*block;

stack<ll> sta;

void tarjan(ll r,ll f,vector<vector<ll>> &res,const graph &G,bool mode){

ll \*head = G.head,s,t;edge \*E = G.E;

bool haskip = 0;

sta.push(r);

insta[r] = 1,dfn[r] = low[r] = ++vid;

for(ll i=head[r];i!=-1;i=E[i].next){

s = E[i].to;

if(s == f && mode && !haskip){

haskip = 1; continue;

}

if(!dfn[s]){

tarjan(s,r,res,G,mode);

if(low[s] < low[r])low[r] = low[s];

}else if(dfn[s] < low[r]&&insta[s])

low[r] = dfn[s];

}

if(dfn[r] == low[r]){

res.push\_back(\*(new vector<ll>()));

do{

t = sta.top(); sta.pop();

insta[t] = 0;

res.back().push\_back(t);

block[t] = res.size();

}while(t != r);

}

}

void mainTarjan(ll nodecon,vector<vector<ll>> &res,const graph &G,bool mode){

// mode == 1 shuang lian tong mode == 0 qiang lian tong

static bool first\_time = 1;

if(first\_time){

first\_time = 0;

dfn = new ll[G.\_maxn];

low = new ll[G.\_maxn];

insta = new ll[G.\_maxn];

block = new ll[G.\_maxn];

}

ll arrsize = sizeof(ll)\*(nodecon+1);

memset(low,0,arrsize);

memset(dfn,0,arrsize);

memset(insta,0,arrsize);

vid = 0;

for(ll i=1;i<=nodecon;i++){

if(!dfn[i]){

tarjan(i,-1,res,G,mode);

}

}

}