1.次短路径 堆优化 Dijkstra

#include<iostream>

#include<cstdio>

#include<cstring>

#include<queue>

using namespace std;

typedef long long  ll;

const ll INF=110000000000;

const int MAXN=200000+10;

struct Rec

{

    ll num,len;

    bool operator < (const Rec &a) const

    {

        return len>a.len;

    }

};

int u[MAXN\*2],v[MAXN\*2];

ll w[MAXN\*2];

ll dis[MAXN/2],secondis[MAXN/2];

ll first[MAXN/2],mynext[MAXN\*2];

int n,r;

void dijkstra()

{

    priority\_queue<Rec> que;

    for (int i=1;i<n;i++)

    {

        dis[i]=INF;

        secondis[i]=INF;

    }

    dis[0]=0;

    secondis[0]=INF;

    Rec temp;

    temp.len=0;temp.num=0;

    que.push(temp);

    while (!que.empty())

    {

        Rec head=que.top();

        que.pop();

        if (head.len>secondis[head.num])

            continue;

        ll k=first[head.num];

        while (k!=-1)

        {

            ll d=head.len+w[k];

            if (dis[v[k]]>d)

            {

                swap(dis[v[k]],d);

                temp.len=dis[v[k]];

                temp.num=v[k];

                que.push(temp);

            }

            if (dis[v[k]]<d && secondis[v[k]]>d)

            {

                secondis[v[k]]=d;

                temp.len=secondis[v[k]];temp.num=v[k];

                que.push(temp);

            }

            k=mynext[k];

        }

    }

}

int main()

{

    int t;

    cin>>t;

    while(t--)

    {

        scanf("%d%d",&n,&r);

        memset(first,-1,sizeof(first));

        for (int i=0;i<r;i++)

        {

            scanf("%d%d%lld",&u[i],&v[i],&w[i]);

            u[i]--;

            v[i]--;

            mynext[i]=first[u[i]];

            first[u[i]]=i;

            v[i+r]=u[i];

            u[i+r]=v[i];

            w[i+r]=w[i];

            mynext[i+r]=first[u[i+r]];

            first[u[i+r]]=i+r;

        }

        dijkstra();

        cout<<secondis[n-1]<<endl;

    }

    return 0;

}

2. 合数分解

/\*

 \* 合数的分解需要先进 素数的筛选

 \* factor[i][0]存放分解的素数

 \* factor[i][1]存放对应素数出现的次数

 \* fatCnt存放合数分解出的素数个数(相同的素数只算 次) \*/

const int MAXN = 10000;

int prime[MAXN + 1];

// 获取素数

void getPrime()

{

    memset(prime, 0, sizeof(prime));

    for (int i = 2; i <= MAXN; i++)

    {

        if (!prime[i]) {

            prime[++prime[0]] = i; }

        for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++) {

            prime[prime[j] \* i] = 1; if (i % prime[j] == 0)

            {

                break; }

        }

    }

    return ;

}

long long factor[100][2]; int fatCnt;

// 合数分解

int getFactors(long long x) {

    fatCnt = 0;

    long long tmp = x;

    for (int i = 1; prime[i] <= tmp / prime[i]; i++) {

        factor[fatCnt][1] = 0;

        if (tmp % prime[i] == 0) {

            factor[fatCnt][0] = prime[i];

            while (tmp % prime[i] == 0) {

                factor[fatCnt][1]++;

                tmp /= prime[i];

            }

            fatCnt++;

        }

    }

    if (tmp != 1) {

        factor[fatCnt][0] = tmp;

        factor[fatCnt++][1] = 1;

    }

    return fatCnt;

}

3.最大公约数、最小公倍数性质：

1.若a | m，b | m，则lcm(a，b) | m。

2.若d | a，d | b，则d | gcd(a，b)。

3.lcm(a，b) = a \* b / gcd(a，b)。

4.设m，a，b是正整数，则lcm(m\*a，m\*b) = m \* gcd(a，b)

5.若m是非零整数a1，a2，…，an的公倍数，则lcm(a1，a2，…，an) | m。

用素因子分解法求a和b的最大公约数和最小公倍数：

a = p1^r1 \* p2^r2 \* … \* pk^rk

b = p1^s1 \* p2\*s2 \* … \* pk^sk

gcd(a，b) = p1^min(r1,s1) \* p2^min(r2,s2) \* … \* pk^min(rk,sk)

lcm(a，b) = p1^max(r1,s1) \* p2^max(r2,s2) \* … \* pk^max(rk,sk)

最大公约数——欧几里得算法O(n)

int GCD(int a,int b)

{

    if(a < b)

        int temp = a, a = b, b = temp;

    if(b == 0) return a;

    return GCD(b,a%b);

}

Stein算法O( log(max(a,b)) )

int KGCD(int a,int b)

{

    if(a == 0) return b;

    if(b == 0) return a;

    if(~a & 1)

    {

        if(b & 1) return KGCD(a>>1, b);

        else return KGCD(a>>1, b>>1) << 1;

    }

    if(~b & 1) return KGCD(a, b>>1);

    if(a > b) return KGCD((a-b)>>1, b);

    return KGCD((b-a)>>1, a);

}

最小公倍数：

int LCM(int a,int b)

{

    return a/KGCD(a,b)\*b;

}

3.组合数打表

void Initial()

{

    int i, j;

    for (i = 0; i <= MAXN; ++i)

    {

        C[0][i] = 0;

        C[i][0] = 1;

    }

    for (i = 1; i <= MAXN; ++i)

    {

        for (j = 1; j <= MAXN; ++j)

            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;

    }

}

int Combination(int n, int m)

{

    if (n > m)

        return C[n][m];

    else

        return C[m][n];

}

直接算：

int com(int n, int r) {

    if (n - r > r) {

        r=n-r;

        // return C(n, r)

        // C(n,r)=C(n,n-r)

    }

    int i, j, s = 1;

    for (i = 0, j = 1; i < r; ++i) {

        s \*= (n - i);

        for (; j <= r && s % j == 0; ++j) {

            s /= j;

        }

    }

    return s;

}

4.集合分划

/\*

 \* n元集合分划为k类的 案数记为S(n, k),称为第 类Stirling数。

 \* 如{A,B,C}可以划分{{A}, {B}, {C}}, {{A, B}, {C}}, {{B, C}, {A}}, {{A, C}, {B}}, {{A, B, C}}。

 \* 即 个集合可以划分为 同集合(1...n个)的种类数

 \* CALL: compute(N); 每当输  个n,输出B[n]

 \*/

const int N = 2001;

int data[N][N], B[N];

void NGetM(int m, int n) // m 个数 n 个集合

{

    // data[i][j]: i个数分成j个集合 int min, i, j;

    data[0][0] = 1;

    for (i = 1; i <= m; i++) {

        data[i][0] = 0; }

    for (i = 0; i <= m; i++) {

        data[i][i + 1] = 0; }

    for (i = 1; i <= m; i++) {

        if (i < n) {

            min = i; }

        else {

            min = n; }

        for (j = 1; j <= min; j++) {

      data[i][j] = (j \* data[i - 1][j] + data[i - 1][j - 1]); }

    }

    return ; }

void compute(int m) {

    // b[i]: Bell数 NGetM(m, m);

    memset(B, 0, sizeof(B)); int i, j;

    for (i=1; i <= m; i++)

    {

        for (j = 0; j <= i; j++) {

            B[i] += data[i][j]; }

    }

    return ;

}

5. Sunday Algorithm

void SUNDAY(char \*text, char \*patt) {

    size\_t temp[256];

    size\_t \*shift = temp;

    size\_t i, patt\_size = strlen(patt), text\_size = strlen(text); cout << "size : " << patt\_size << endl;

    for(i = 0; i < 256; i++)

    {

        \*(shift+i) = patt\_size + 1; }

    for(i = 0; i < patt\_size; i++) {

        \*(shift + (unsigned char)(\*(patt+i))) = patt\_size-i;

    }

    size\_t limit = text\_size - patt\_size + 1;

    for(i = 0; i < limit; i += shift[text[i + patt\_size]])

        // shift['s']=6步,shitf['e']=5以此类推

    {

        if(text[i] == \*patt) {

            char \*match\_text = text + i + 1; size\_t match\_size = 1;

            do // 输出所有匹配的位置

            {

                if(match\_size == patt\_size) {

                    cout << "the NO. is " << i << endl; }

            }

            while((\*match\_text++) == patt[match\_size++]); }

    }

    cout << endl;

}

int main(void) {

    char text[100] = "substring searching algorithm search"; char patt[10] = "search";

    SUNDAY(text, patt);

    return 0;

}

6.多边形面积

/\*

 \* 要求按照逆时针 向输 多边形顶点 \* 可以是凸多边形或凹多边形

 \*/

double area\_of\_polygon(int vcount, double x[], double y[], Lpoint plg[]) {

    int i;

    double s;

    if (vcount < 3) {

        return 0; }

    s = plg[0].y \* (plg[vcount - 1].x - plg[1].x); for (i = 1; i < vcount; i++)

    {

        s += plg[i].y \* (plg[(i - 1)].x - plg[(i + 1) % vcount].x); }

    return s / 2;

}