Dijkstra

> *Luogu P3371 //代码其实是由简化紫书上而来*

```
#include<cstdio>
#include<vector>
#include<cstring>

using namespace std;

struct Edge{
    int to,val;
};

vector<Edge> G[100010];
int vis[100010];
int dis[100010];

const int INF=0x3f3f3f3f;

void add_edge(int u,int v,int d)
{
    Edge t;
    t.to=v,t.val=d;
    G[u].push_back(t);
}

void dijkstra(int s,int n)
{
    memset(vis,0,sizeof(vis));
    for(int i=0;i<n;i++)
        dis[i] = (i == s ? 0 : INF);
    for(int i = 0;i<n;i++)
    {
        int x, minn = INF;
        for(int j = 0;j<n;j++){
            if(!vis[j] && dis[j]<=minn)
            {
                x=j;
                minn=dis[j];
            }
        }
        vis[x] = 1;
        for(unsigned int j = 0;j<G[x].size();j++)
        {
```

```
            int y = G[x][j].to;
            int d = G[x][j].val;
            dis[y] = min(dis[y],dis[x]+d);
        }
    }
}

int main()
{
    int n,m,s;
    scanf("%d%d%d",&n,&m,&s);
    for(int i=1;i<=m;i++)
    {
        int f,g,w;
        scanf("%d%d%d",&f,&g,&w);
        add_edge(f,g,w);
    }
    dijkstra(s,n+1);
    for(int i=1;i<=n;i++)
        printf("%d ",dis[i]==0x3f3f3f3f?2147483647:dis[i]);
    return 0;
}
```

求质数(邪门数学版本)

> *Luogu P3383*
> *定理:任何大于5的素数一定是与6相邻的数*

```
#include<cmath>
#include<cstdio>

using namespace std;

int n,m;

bool is_prime(int a)
{
    if(a==2||a==3)
        return true;
    if(a==1 || (a%6!=1 && a%6!=5))
        return false;
    int temp=sqrt(a);
    for(int i=5;i<=temp;i+=6)
    {
        if(a%i==0||a%(i+2)==0)
            return false;
    }
```

```
    return true;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int x;
        scanf("%d",&x);
        if(is_prime(x))
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

埃氏筛法

> *Luogu P3383*

```
#include<cstdio>
#include<cmath>

using namespace std;

int n,m,fw,tmp;
bool a[100000000];

int main()
{
    scanf("%d%d",&n,&m);
    fw=sqrt(n+0.5);
    a[1]=1;
    for(int i=2;i<=fw;i++)
    {
        if(a[i]==0)
        {
            for(int j=i*i;j<=n;j+=i)
                a[j]=1;
        }
    }
    for(int i=1;i<=m;i++)
    {
        scanf("%d",&tmp);
        if(a[tmp]==0)
```

```
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

快速幂

> *Luogu P1226*
> *输入b，p，k的值，求b^p mod k的值。其中b，p，k*k为长整型数。*

```
#include<iostream>

using namespace std;

long long q_pow(long long b,long long p,long long k)
{
    long long ans=1;
    while(p>0)
    {
        if(p&1)
            ans=ans*b%k;
        b=b*b%k;
        p>>=1;
    }
    return ans%k;
}

int main()
{
    long long b,p,k;
    cin>>b>>p>>k;
    cout<<b<<"^"<<p<<" mod "<<k<<"="<<q_pow(b,p,k)<<endl;
    return 0;
}
```

最小生成树

> *Luogu P3366*
> *第一行包含两个整数N、M，表示该图共有N个结点和M条无向边。*
> *接下来M行每行包含三个整数Xi、Yi、Zi，表示有一条长度为Zi的无向边连接结点Xi、Yi*
> *输出包含一个数，即最小生成树的各边的长度之和；如果该图不连通则输出orz*

```cpp
#include<cstdio>
#include<algorithm>

using namespace std;

struct grap
{
    int x,y,z;
}G[200003];

int n,m,flag,sum;
int r[5003];

int findx(int x)
{
    if(r[x]!=x)
        r[x]=findx(r[x]);
    return r[x];
}

bool chaxun(int x,int y)
{
    return findx(x)==findx(y);
}

void hebing(int x,int y)
{
    x=findx(x);
    y=findx(y);
    r[x]=y;
}

bool cmp(grap a,grap b)
{
    return a.z<b.z;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        r[i]=i;
    for(int i=1;i<=m;i++)
        scanf("%d%d%d",&G[i].x,&G[i].y,&G[i].z);
    sort(G+1,G+m+1,cmp);
    int i=1;
    while(i<=m && flag<=n-1)
    {
        if(!chaxun(G[i].x,G[i].y))
        {
```

```
            hebing(G[i].x,G[i].y);
            flag++;
            sum+=G[i].z;
        }
        i++;
    }
    printf("%d\n",sum);
    return 0;
}
```

并查集

> *Luogu P3367*
> *第一行包含两个整数N、M，表示共有N个元素和M个操作。*
> *接下来M行，每行包含三个整数Zi、Xi、Yi*
> *当Zi=1时，将Xi与Yi所在的集合合并*
> *当Zi=2时，输出Xi与Yi是否在同一集合内，是的话输出Y；否则话输出N*

```
#include<cstdio>

using namespace std;

int n,m;
int r[10003];

int findx(int x)
{
    if(r[x]!=x)
        r[x]=findx(r[x]);
    return r[x];
}

void hebing(int x,int y)
{
    x=findx(x);
    y=findx(y);
    r[x]=y;
}

bool chaxun(int x,int y)
{
    if(findx(x)==findx(y))
        return true;
    else
        return false;
}
```

```
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        r[i]=i;
    for(int i=1;i<=m;i++)
    {
        int z,x,y;
        scanf("%d%d%d",&z,&x,&y);
        if(z==1)
            hebing(x,y);
        else if(chaxun(x,y))
            printf("Y\n");
        else
            printf("N\n");
    }
    return 0;
}
```

树状数组

> *Luogu P3372*
> *第一行包含两个整数N、M，分别表示该数列数字的个数和操作的总个数。*
> *第二行包含N个用空格分隔的整数，其中第i个数字表示数列第i项的初始值。*
> *接下来M行每行包含3或4个整数，表示一个操作，具体如下：*
> *操作1： 格式：1 x y k 含义：将区间[x,y]内每个数加上k*
> *操作2： 格式：2 x y 含义：输出区间[x,y]内每个数的和*
> *输出包含若干行整数，即为所有操作2的结果。*

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

#define LL long long
#define MN 100500

using namespace std;

LL c1[MN],c2[MN];
LL n,m;
LL lb(LL x)
{
    return x&-x;
}

void addd(LL pos,LL data)
{
```

```
    for(LL i=pos;i<=n;i+=lb(i))
    {
        c1[i]+=data;
        c2[i]+=data*pos;
    }
}

LL gsum(LL pos)
{
    LL ans=0;
    for(LL i=pos;i;i-=lb(i))
        ans+=(pos+1)*c1[i]-c2[i];
    return ans;
}

int main()
{
    cin>>n>>m;
    for(LL i=1,_;i<=n;i++)
    {
        cin>>_;
        addd(i,_);
        addd(i+1,-_);
    }
    for(LL i=1,cz;i<=m;i++)
    {
        LL x,y,k;
        cin>>cz;
        if(cz==1)
        {
            cin>>x>>y>>k;
            addd(x,k);
            addd(y+1,-k);
        }
        if(cz==2)
        {
            cin>>x>>y;
            cout<<gsum(y)-gsum(x-1)<<endl;
        }
    }
}
```

线段树基本版

> *Luogu P3372*
> *同上*

```cpp
#include<iostream>
#include<cstdio>
#define MAXN 1000001
#define ll long long
using namespace std;
unsigned ll n,m,a[MAXN],ans[MAXN<<2],tag[MAXN<<2];
inline ll ls(ll x)
{
    return x<<1;
}
inline ll rs(ll x)
{
    return x<<1|1;
}
void scan()
{
    cin>>n>>m;
    for(ll i=1;i<=n;i++)
    scanf("%lld",&a[i]);
}
inline void push_up(ll p)
{
    ans[p]=ans[ls(p)]+ans[rs(p)];
}
void build(ll p,ll l,ll r)
{
    tag[p]=0;
    if(l==r){ans[p]=a[l];return ;}
    ll mid=(l+r)>>1;
    build(ls(p),l,mid);
    build(rs(p),mid+1,r);
    push_up(p);
}
inline void f(ll p,ll l,ll r,ll k)
{
    tag[p]=tag[p]+k;
    ans[p]=ans[p]+k*(r-l+1);
}
inline void push_down(ll p,ll l,ll r)
{
    ll mid=(l+r)>>1;
    f(ls(p),l,mid,tag[p]);
    f(rs(p),mid+1,r,tag[p]);
    tag[p]=0;
}
inline void update(ll nl,ll nr,ll l,ll r,ll p,ll k)
{
    if(nl<=l&&r<=nr)
    {
        ans[p]+=k*(r-l+1);
        tag[p]+=k;
```

```
        return ;
    }
    push_down(p,l,r);
    ll mid=(l+r)>>1;
    if(nl<=mid)update(nl,nr,l,mid,ls(p),k);
    if(nr>mid) update(nl,nr,mid+1,r,rs(p),k);
    push_up(p);
}
ll query(ll q_x,ll q_y,ll l,ll r,ll p)
{
    ll res=0;
    if(q_x<=l&&r<=q_y)return ans[p];
    ll mid=(l+r)>>1;
    push_down(p,l,r);
    if(q_x<=mid)res+=query(q_x,q_y,l,mid,ls(p));
    if(q_y>mid) res+=query(q_x,q_y,mid+1,r,rs(p));
    return res;
}
int main()
{
    ll a1,b,c,d,e,f;
    scan();
    build(1,1,n);
    while(m--)
    {
        scanf("%lld",&a1);
        switch(a1)
        {
            case 1:{
                scanf("%lld%lld%lld",&b,&c,&d);
                update(b,c,1,n,1,d);
                break;
            }
            case 2:{
                scanf("%lld%lld",&e,&f);
                printf("%lld\n",query(e,f,1,n,1));
                break;
            }
        }
    }
    return 0;
}
```

STL

```
#include<algorithm>
```

内置快排：`sort(起点指针,结束位置指针,排序函数);`

部分排序:nth_element(a+l,a+k,a+r);//使a这个数组中区间(l,r)内的第k大的元素处在第k个位置上(相对位置)(用途是拿来求中位数较多)
std::adjacent_find//在序列中查找第一对相邻且值相等的元素；
std::find//对一个输入序列，查找第一个等于给定值的元素；
std::find_end//查找有B定义的序列在A序列中最后一次出现的位置(B可能是A的子序列)；
std::find_first_of//查找A序列中第一个与序列B中任一元素值相等的元素位置；
std::find_if//在序列中返回满足谓词(给定的条件)的第一个元素；
std::find_if_not//在序列中返回不满足谓词的第一个元素；
std::all_of//如果序列中所有元素均满足给定的条件，则返回true；
std::any_of//如果序列中存在元素满足给定的条件，则返回true；
std::none_of//如果序列中所有元素均不满足给定的条件，则返回true；
std::binary_search//对一个升序序列做二分搜索，判断序列中是否有与给定值相等的元素；
std::search//在序列A中，搜索B首次出现的位置(B可能是A的子序列)；
std::search_n//在给定序列中，搜索给定值连续出现n次的位置；
std::copy//将一个序列中的元素拷贝到新的位置；
std::copy_backward//把一个序列复制到另一个序列，按照由尾到头顺序依次复制元素；
std::copy_if//将一个序列中满足给定条件的元素拷贝到新的位置；
std::copy_n//将一个序列中的前n个元素拷贝到新的位置；
std::count//返回序列中等于给定值元素的个数；
std::count_if//返回序列中满足给定条件的元素的个数；
std::equal//比较两个序列的对应元素是否相等；
std::equal_range//在已排序的序列中，查找元素等于给定值组成的子范围；
std::lower_bound//在升序的序列中，查找第一个不小于给定值的元素；
std::upper_bound//在已排序的序列中，查找第一个大于给定值的元素；
std::fill//用给定值填充序列中的每个元素；
std::fill_n//用给定值填充序列中的n个元素；
std::for_each//将指定函数应用于范围内的每一个元素；
std::generate//对序列中的每个元素，用依次调用函数gen的返回值赋值；
std::generate_n//对序列中的n个元素，用依次调用指定函数gen的返回值赋值；
std::includes//判断第二个已排序的序列是否全部都出现在第一个已排序的序列中；
std::inplace_merge//对两个升序的序列执行原地合并，合并后的序列仍保持升序；
std::merge//对两个升序的序列合并，结果序列保持升序；
std::is_heap//判断序列是否为二叉堆；
std::is_heap_until//查找第一个不是堆顺序的元素；
std::make_heap//对于一个序列，构造一个二叉堆；
std::pop_heap//堆的根节点被移除，堆的元素数目减1并保持堆性质；
std::push_heap//向堆中增加一个新元素，新元素最初保存在last-1位置；
std::sort_heap//对一个堆，执行原地堆排序，得到一个升序结果；
std::is_partitioned//判断序列是否按指定谓词划分过；
std::partition//对序列重排，使得满足谓词的元素位于最前；
std::partition_copy//输入序列中，满足谓词的元素复制到result_true，其它元素复制到result_false；
std::partition_point//输入序列已经是partition，折半查找到分界点；
std::stable_partiton//对序列重排，使得满足谓词的元素在前，不满足谓词的元素在后，且两组元素内部的相对顺序不变；
std::is_permutation//判断两个序列是否为同一元素的两个排列；
std::next_permutation//n个元素有n!中排列。这些排列中，规定升序序列为最小排列，降序序列为最大的排列，任意两个排列按照字典序分出大小。该函数返回当前序列作为一个排列按字典序的下一个排列；
std::prev_permutation//返回当前序列作为一个排列按字典序的上一个排列；
std::is_sorted//判断序列是否为升序；

std::is_sorted_until//查找序列中第一个未排序的元素；
std::nth_element//对序列重排，使得指定的位置出现的元素就是有序情况下应该在该位置出现的那个元素，且在指定位置之前的元素都小于指定位置元素，在指定位置之后的元素都大于指定位置元素；
std::partial_sort//对序列进行部分排序；
std::partial_sort_copy//拷贝部分排序的序列；
std::stable_sort//对序列进行稳定排序；
std::iter_swap//交换两个迭代器指向的元素；
std::swap//交换两个对象，优先使用移动语义；
std::swap_ranges//交换两个序列中对应元素；
std::lexicographical_compare//对两个序列做字典比较，如果第一个序列在字典序下小于第二个序列，则返回true；
std::min_element//返回序列中的最小值；
std::max_element//返回序列中的最大值；
std::minmax//返回由最小值与最大值构成的std::pair；
std::minmax_element//返回由序列中最小元素与最大元素构成的std::pair；
std::mismatch//比较两个序列的对应元素，返回用std::pair表示的第一处不匹配在两个序列的位置；
std::shuffle//使用均匀随机数生成器，随机打乱指定范围中的元素的位置；
std::random_shuffle//n个元素有!n个排列，该函数给出随机选择的一个排列；
std::remove//删除序列中等于给定值的所有元素；
std::remove_if//删除序列中满足给定谓词的元素；
std::remove_copy//把一个序列中不等于给定值的元素复制到另一个序列中；
std::remove_copy_if//把一个序列中不满足给定谓词的元素复制到另一个序列中；
std::replace//把序列中等于给定值的元素替换为新值；
std::replace_if//把序列中满足给定谓词的元素替换为新值；
std::replace_copy//拷贝序列，对于等于老值的元素复制时使用新值；
std::replace_copy_if//拷贝序列，对于满足给定谓词的元素复制时使用新值；
std::reverse//把序列中的元素逆序；
std::reverse_copy//拷贝序列的逆序到另一个序列中；
std::rotate//等效于循环左移序列，使得迭代器middle所指的元素成为首元素；
std::rotate_copy//等效于循环左移序列并拷贝到新的序列中，使得迭代器middle所指的元素成为首元素；
std::set_difference//两个升序序列之差；
std::set_intersection//两个升序序列的交；
std::set_symmetric_difference//两个升序序列的对称差；
std::set_union//两个升序序列的并；
std::transform//对序列中的每一个元素，执行一元操作，结果写入另一序列中；或对两个序列中对应的每一对元素，执行二元操作，结果写入另一序列中；
std::unique//对序列中一群连续的相等的元素，仅保留第一个元素；
std::unique_copy//把一个序列中的元素拷贝到另一个序列，对于一群连续的相等的元素，仅拷贝第一个元素。

```
#include<queue> //优先队列(堆)
```
大根堆定义：`priority_queue< int >pq`
小根堆定义：`priority_queue< int ,vector< int >,greater< int > >pq`
//（注意最后两个">"符号不要连在一起，否则会被很多（但不是所有）编译器误认为是'>>'运算符）

```
push() //元素入队
pop() //队首元素出队
top() //取队首元素
```

```
empty() //如果队列为空，则返回true（1），否则返回false（0）
size() //返回优先队列中拥有的元素个数
```

GCD && LCM

```
int gcd(int a,int b) {return b?gcd(b,a%b):a;}
int lcm(int a,int b) {return a*b/gcd(a,b);}
```

匈牙利算法

> *匹配方式:当妹子还没被抢走就直接上,妹子已经被其他男生抢走,赶走了再上。*
> *https://blog.csdn.net/dark_scope/article/details/8880547*
> *貌似以前看这个学的*

```
//匈牙利查找过程
bool find(int x){
    int i,j;
    for (j=1;j<=m;j++){     //扫描每个妹子
        if (line[x][j]==true && used[j]==false)
        //如果有暧昧并且还没有标记过（这里标记的意思是这次查找曾试图改变过该妹子的归属问题，但是没有成功，所以就不用瞎费工夫了）
        {
            used[j]=1;
            if (girl[j]==0 || find(girl[j])) {
                //名花无主或者能腾出个位置来，这里使用递归
                girl[j]=x;
                return true;
            }
        }
    }
    return false;
}
//在主程序我们这样做：
for (i=1;i<=n;i++)
{
    memset(used,0,sizeof(used));     //这个在每一步中清空
    if find(i) all+=1;
}
```