**快排：**

const int N=1000010;

int n;

int q[N]；

void quick\_sort(int q[],int l,int r)

{

if (l>=r) return;

int x=q[(l+r)/2],i=l-1,j=r+1;

while(i<j)

{

do i++;while(q[i]<x);

do j--;while(q[j]>x);

if(i<j) swap(q[i],q[j]);

}

quick\_sort(q,l,j); //这里要用j，用i会报错

quick\_sort(q,j+1,r);

}

**归并：**

const int N=1000010;

int n;

int q[N],tmp[N];

void merge\_sort(int q[],int l,int r)

{

if(l>=r) return;

int mid=l+r>>1;

merge\_sort(q,l,mid),merge\_sort(q,mid+1,r);

int k=0,i=l,j=mid+1;

while(i<=mid&&j<=r)

{

if(q[i]<=q[j]) tmp[k++]=q[i++];

else tmp[k++]=q[j++];

}

while(i<=mid) tmp[k++]=q[i++];

while(j<=r) tmp[k++]=q[j++];

for(i=l,j=0;i<=r;i++,j++) q[i]=tmp[j];

}

**整数二分：**

版本1

当我们将区间[l, r]划分成[l, mid]和[mid + 1, r]时，其更新操作是r = mid或者l = mid + 1;，计算mid时不需要加1。

**找最小的不可以的**

C++ 代码模板：

int bsearch\_1(int l, int r)

{

while (l < r)

{

int mid = l + r >> 1;

if (check(mid)) r = mid;

else l = mid + 1;

}

return l;

}

版本2

当我们将区间[l, r]划分成[l, mid - 1]和[mid, r]时，其更新操作是r = mid - 1或者l = mid;，此时为了防止死循环，计算mid时需要加1。

**找最大的可以的**

C++ 代码模板：

int bsearch\_2(int l, int r)

{

while (l < r)

{

int mid = l + r + 1 >> 1;

if (check(mid)) l = mid;

else r = mid - 1;

}

return l;

}

**KMP:**

#include<iostream>

using namespace std;

const int M=1000010,N=1000010;

int n,m;

char p[N],s[M];

int ne[N];

int main()

{

cin>>n>>p+1>>m>>s+1;

// 求next

for(int i=2,j=0;i<=n;i++){

while(j && p[i]!=p[j+1]) j=ne[j];

if(p[i]==p[j+1]) j++;

ne[i]=j;

}

// kmp

for(int i=1,j=0;i<=m;i++){

while(j && s[i]!=p[j+1]) j=ne[j];

if(s[i]==p[j+1]) j++;

if(j==n){

// 匹配成功

printf("%d ",i-n);

j=ne[j];

}

}

return 0;

}

**Trie树：**

char str[N];

int son[N][26]; // 26为每个节点的最大可能边数，可灵活修改

int cnt[N]; // 当前节点的孩子有多少个

int idx; //当前用到的下标,根节点是0且空

void insert(char str[])

{

int p=0;

for(int i=0;str[i];i++){

int u=str[i]-'a'; // 映射到0-25

if(!son[p][u]) son[p][u]=++idx;

p=son[p][u];

}

cnt[p]++;

}

int query(char str[])

{

int p=0;

for(int i=0;str[i];i++){

int u=str[i]-'a';

if(!son[p][u]) return 0;

p=son[p][u];

}

return cnt[p];

}

# 并查集

#include<iostream>

using namespace std;

const int N=100010;

int p[N];// 父节点

int n,m;

int find(int x) // 返回x所在集合编号

{

if(p[x]!=x) p[x]=find(p[x]);

return p[x];

}

int main()

{

cin>>n>>m;

for(int i=1;i<=n;i++) p[i]=i;

while(m--){

char op;

int a,b;

cin>>op>>a>>b;

if(op=='M') p[find(a)]=find(b);

if(op=='Q'){

if(find(a)==find(b)) cout<<"Yes"<<endl;

else cout<<"No"<<endl;

}

}

return 0;

}

# 堆排序

#include<iostream>

#include<algorithm>

using namespace std;

const int N=100010;

int n,m;

int h[N],siz;

void down(int u)

{

int t=u;

if(u\*2<=siz && h[u\*2]<h[t]) t=u\*2;

if(u\*2+1<=siz && h[1+u\*2]<h[t]) t=u\*2+1;

if(u!=t){

swap(h[u],h[t]);

down(t);

}

}

void up(int u)

{

while(u/2>0 && h[u/2]>h[u]){

swap(h[u/2],h[u]);

u/=2;

}

}

int main()

{

cin>>n>>m;

for(int i=1;i<=n;i++) cin>>h[i];

siz=n;

for (int i = n/2; i ; i--) // 从倒数第二层开始down

{

down(i);

}

while (m--)

{

cout<<h[1]<<" ";

h[1]=h[siz];

siz--;

down(1);

}

return 0;

}

**哈希表：**

**开放寻址法：**

const int N = 200003, null = 0x3f3f3f3f;

int h[N];

int find(int x)

{

int t = (x % N + N) % N;

while (h[t] != null && h[t] != x)

{

t ++ ;

if (t == N) t = 0;

}

return t;

}

**拉链法：**

int h[N], e[N], ne[N], idx;

void insert(int x)

{

int k = (x % N + N) % N;

e[idx] = x;

ne[idx] = h[k];

h[k] = idx ++ ;

}

bool find(int x)

{

int k = (x % N + N) % N;

for (int i = h[k]; i != -1; i = ne[i])

if (e[i] == x)

return true;

return false;

}

**字符串哈希：**

#include<iostream>

using namespace std;

const int N=100010,P=131;

typedef unsigned long long ULL;

int n,m;

char str[N];

ULL h[N],p[N];

ULL get(int l,int r)

{

return h[r]-h[l-1]\*p[r-l+1];

}

int main()

{

scanf("%d%d%s",&n,&m,str+1);

p[0]=1;

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

p[i]=p[i-1]\*P;

h[i]=h[i-1]\*P+str[i];

}

while(m--){

int l1,r1,l2,r2;

cin>>l1>>r1>>l2>>r2;

if(get(l1,r1)==get(l2,r2)) puts("Yes");

else puts("No");

}

return 0;

}

**Dfs：**

int path[N];

int n;

bool st[N];

void dfs(int u){

if(u==n){

for(int i=0;i<n;i++) cout<<path[i]<<" ";

puts("");

return;

}

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

if(!st[i]){

path[u]=i;

st[i]=true;

dfs(u+1);

st[i]=false;

}

}

}

**Dijkstra:**

int dijkstra()

{

memset(dst,0x3f,sizeof dst);

dst[1]=0;

for(int i=0;i<n;i++){

int t=-1;

for(int j=1;j<=n;j++){

if(!st[j] && (t==-1 || dst[t]>dst[j])){

t=j;

}

}

st[t]=true;

for(int j=1;j<=n;j++){

dst[j]=min(dst[j],dst[t]+g[t][j]);

}

}

if(dst[n]==0x3f3f3f3f) return -1;

else return dst[n];

}