**数据结构**

**单调队列：**

push:平均O(1),查询:O（1）

struct Node

{

int val,t; //val是值，t是加入的时间

Node(int val\_,int t\_):val(val\_),t(t\_){}

Node(){}

};

Node maxq[maxn];

int maxf,maxr;

void push(int n,int t)

{

if(maxq[maxf].t<t-k+1) //如果头超出范围，则pop

maxf++;

for(int i=maxr-1;i>=maxf;--i)//把队列中小于要push进去的值的点都pop掉

if(maxq[i].val<n)

maxr--;

else break;

maxq[maxr++]=Node(n,t);

}

**线段树：**

struct Node

{

int l,r; //表示的线段范围

int color; //存的值，这里为二进制表示的存在color数（状态压缩）

bool lazy; //lazy标记

};

Node tree[4\*maxn];

void buildtree(int l,int r,int n)

{

tree[n].r=r;

tree[n].l=l;

tree[n].color=1;

tree[n].lazy=0;

if(r==l)

{

return;

}

int m=(l+r)>>1;

buildtree(l,m,n<<1);

buildtree(m+1,r,n<<1|1);

}

void pushDown(int n)

{

if(tree[n].lazy)

{

tree[n<<1].color=tree[n<<1|1].color=tree[n].color;

tree[n].lazy=0;

tree[n<<1].lazy=tree[n<<1|1].lazy=1;

}

}

void pushUp(int n)

{

tree[n].color=tree[n<<1].color|tree[n<<1|1].color;

}

void change(int l,int r,int col,int n)

{

if(tree[n].l==l && tree[n].r==r)

{

tree[n].color=col;

if(l!=r)

tree[n].lazy=1;

return;

}

if(tree[n].color==col) return;

pushDown(n);

int m=(tree[n].l+tree[n].r)>>1;

if(r<=m) change(l,r,col,n<<1);

else if(l<=m && r>m)

{

change(l,m,col,n<<1);

change(m+1,r,col,n<<1|1);

}

else change(l,r,col,n<<1|1);

pushUp(n);

}

inline int toBin(const int n) //把颜色转换为二进制表示

{

return 1<<(n-1);

}

int count(int n) //计算一个color里有多少种颜色

{

int res=0;

while(n)

{

if(n&1) res++;

n>>=1;

}

return res;

}

**树状数组：**

int lowbit(int a)

{

return a&(-a);

}

int sum(int end)

{

int res=0;

for(;end>0;end-=lowbit(end))

res+=s[end];

return res;

}

void add(int n,int a)

{

for(;n<=N;n+=lowbit(n))

s[n]+=a;

}

**堆排序：**

void maxsort(float a[],int count);

void maxbuild(float a[],int count);

void heapify(float a[],int start,int count);

int main()

{

freopen("in.txt","r",stdin);

float a[100];

float \*p=a+1;

int count=0;

printf("please input the data,end up by 0\n");

scanf("%f",p);

while(\*p++)

{

scanf("%f",p);

count++;

}

maxbuild(a,count);

maxsort(a,count);

p=a+1;

for(;count>0;count--)

printf("%f\t",\*p++);

return 0;

}

void heapify(float a[],int start,int count)

{

int m=start;

int k,max;

float t;

while(1)

{

k=m\*2;

if(k>count)

break;

else if(k+1>count)

{

if(a[k]>a[m])

{

t=a[k];

a[k]=a[m];

a[m]=t;

}

break;

}

else

{

if(a[k]>a[k+1])

max=k;

else max=k+1;

if(a[max]>a[m])

{

t=a[max];

a[max]=a[m];

a[m]=t;

}

}

m=max;

}

}

void maxbuild(float a[],int count)

{

int i;

for(i=count/2;i>0;i--)

heapify(a,i,count);

}

void maxsort(float a[],int count)

{

float t;

while(count>1)

{

t=a[count];

a[count]=a[1];

a[1]=t;

heapify(a,1,--count);

}

}

**并查集：**

//两个基本功能：并/查~

int father[maxn];

int num[maxn];

int n,m;

void init()

{

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

num[i]=1;

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

father[i]=i;

}

int find(int a)

{

while(father[a]!=a)

{

a=father[a];

}

return a;

}

void Union(int a,int b)

{

a=find(a);

b=find(b);

if(a==b)

return;

father[a]=b;

num[b]+=num[a];

return;

}

**字符串Hash的一种：**

int APHash(const char \*str)

{

int hash = 0;

char ch;

for (long i = 0; ch = (int)\*str++; i++)

{

if ((i & 1) == 0)

{

hash ^= ((hash << 7) ^ ch ^ (hash >> 3));

}

else

{

hash ^= (~((hash << 11) ^ ch ^ (hash >> 5)));

}

}

return abs(hash)%pri;

}

RMQ:

实际就是一个dp

把查询区间按2的几次方分开查询

预处理O(nlgn)

查询O(1)

int num[maxn+10];

int dp[maxn+10][20];

int p2[20];

void pre()

{

p2[0]=1;

for(int i=1; i<20; i++)

p2[i]=p2[i-1]\*2;

}

int find2(int n)

{

return (int)(log(1.0\*n+1)/log(2.0));

}

void RMQ()

{

for(int i=1; i<=maxn; i++)

dp[i][0]=num[i];

for(int j=1; p2[j]<=maxn; j++)

for(int i=1; i<=maxn-p2[j-1]; i++)

dp[i][j]=max(dp[i][j-1],dp[i+p2[j-1]][j-1]);

}

int query(int a,int b)

{

int pp=find2(b-a);

return max(dp[a][pp],dp[b-p2[pp]+1][pp]);

}

求LCA:

(1)Tarjan(离线算法)

时间复杂度：O(n+Q),Q为查询数

使用并查集

int N,M;

bool vis[maxn];

int sum[maxn];

vector<int>son[maxn];

vector<int> que[maxn]; //每个点的查询

int fa[maxn];

void tarjan(int n)

{

for(size\_t i=0;i<son[n].size();i++)

{

tarjan(son[n][i]);

vis[son[n][i]]=1;

fa[son[n][i]]=n;

}

for(size\_t i=0;i<que[n].size();i++)

if(vis[que[n][i]])

sum[find(que[n][i])]++;

}

RMQ（在线算法）

Dfs一遍记录路径以及深度，并记录每点第一次出现的时间(first[i])。然后做RMQ(dpn表示最小值的下标)，查询a,b,就是查询first[a]和first[b]直接深度最小的点

int N,M;

vector<int>son[maxn];

int path[2\*maxn];

int deep[2\*maxn];

int dp[2\*maxn][20];

int dpn[2\*maxn][20];

int first[maxn];

int p;

void dfs(int n,int dep)

{

first[n]=p;

path[p]=n;

deep[p++]=dep;

for(size\_t i=0;i<son[n].size();i++)

{

dfs(son[n][i],dep+1);

path[p]=n;

deep[p++]=dep;

}

}

void RMQ()

{

for(int i=0;i<=p;i++)

{

dp[i][0]=deep[i];

dpn[i][0]=i;

}

for(int j=1;p2[j]<=p;j++)

for(int i=0;i<p-p2[j-1];i++)

if(dp[i][j-1]<dp[i+p2[j-1]][j-1])

{

dp[i][j]=dp[i][j-1];

dpn[i][j]=dpn[i][j-1];

}

else

{

dp[i][j]=dp[i+p2[j-1]][j-1];

dpn[i][j]=dpn[i+p2[j-1]][j-1];

}

}

int query(int a,int b)

{

int pp=find2(b-a);

if(dp[a][pp]<dp[b-p2[pp]+1][pp])

return path[dpn[a][pp]];

else return path[dpn[b-p2[pp]+1][pp]];

}