<数据结构>

闭散列法整数 hash

```
#define max 4000037
int hash[max],c[max];
//hash 存关键字，c 存该位置关键字出现的次数
bool use[max];
int n,m,ans;
int k[6],p[6];

int locate(int k)
//hash 函数
{
	int tmp;
	tmp = k;
	while(tmp<0)tmp+=max;
	while(tmp>=max)tmp-=max;
	while(use[tmp]&&hash[tmp]!=k)
	{
		tmp++;
		if(tmp>=max)tmp-=max;
	}
	return tmp;
}

void insert(int k)
{
	int pos = locate(k);
	hash[pos] = k;
	use[pos] = 1;
	c[pos]++;
}

int onlyfind(int k)
{
	int pos = locate(k);
	if(hash[pos]==k)
		return c[pos];
	else return -1;
}
```

整数 hash,开散列

```
#include <stdio.h>
#include <memory.h>
#define MAX 3400000
#define M 28999999

int head[M],next[MAX];
int lkey[MAX],up,ans;

int n,m;
int k[6],p[6];

int hash(int key)
{
	int h;
	//h = ((key>>16)&&key)^(key>>8);
	h = ((~key)>>8)&&key ||( (key<<16) );
	h%=M;
	return h;
}

int find(int key)
{
	int h,p;
	h = hash(key);
	for(p=head[h];p>=0;p=next[p])
		if(lkey[p]==key)return 1;
	lkey[up] = key;
	next[up] = head[h];
	head[h] = up++;
	return -1;
}

int onlyfind(int key)
{
	int h,p;
	h = hash(key);
	for(p=head[h];p>=0;p=next[p])
		if(lkey[p]==key)return 1;
	return -1;
}
```

字符串 hash

```
int ELFhash(char *key) {
	unsigned h=0;
	while(*key) {
		h=(h<<4) + *key++;
		unsigned g=h & 0Xf0000000L;
		if(g) h^=g>>24;
		h &= ~g;
	}
	return h% M;
}
```

堆

```
int heap[MAX+1],heapsize;

void push(int key)
{
	int pos;
	pos = ++heapsize;
	while(pos>0&&heap[pos>>1]>key)
	{
		heap[pos] = heap[pos>>1];
		pos>>=1;
	}
	heap[pos] = key;
}

void pop()
{
	int key = heap[heapsize--];
	int pos,npos;
	pos = 1;npos = pos<<1;
	while(npos<=heapsize)
	{
		if(npos+1<=heapsize&&heap[npos+1]<heap[npos])npos+=1;
		if(heap[npos]>=key)break;
		heap[pos] = heap[npos];
		pos = npos;
		npos = pos<<1;
	}
	heap[pos] = key;
}
```

二维树状数组

```
int lowbit(int x)
{
	return x&(-x);
}
void updata(int x,int y,int w)
{
	int i,j;
	for(i=x;i<=n;i+=lowbit(i)){
		for(j=y;j<=n;j+=lowbit(j)){
			c[i][j]+=w;
		}
	}
}
int sum(int x,int y)
{
	int t=0,i,j;
	for(i=x;i>0;i-=lowbit(i)){
		for(j=y;j>0;j-=lowbit(j)){
			t+=c[i][j];
		}
	}
	return t;
}
```

Trie 树

```
#include<iostream>
#define keyNum 26
#define MaxN 50
struct trie...{
	struct trieNode...{//trie 结点的结构
	trieNode *link[keyNum];//下标为  'A' , 'B' , 'C' ,    , 'Z'  的指针数组
	int num[keyNum];//插入 key 的次数
	trieNode()...{
		memset(num,0,sizeof(num));
		memset(link,NULL,sizeof(link));
		}
	void init()...{
		memset(link,NULL,sizeof(link));
		memset(num,0,sizeof(num));
	}
	};
	trieNode* root;
```

```
    trie()
    ...{
        root=(trieNode*)malloc(sizeof(trieNode));//初始化时为 root 申请了空间
        root->init();
    }
    bool Search(char *);
    void Insert(char []);
    void Delete(trieNode*);
};
bool trie::Search(char * x)
...{
    if(*x==0)return false;
    trieNode* current=root;
    x++;
    while(*x)...{
        if(current->link[*(x-1)-'a'])
            current=current->link[*(x-1)-'a'];
        else break;
        x++;
    }
    if(*x==0&&current->num[*(x-1)-'a'])
        return true;
    else return false;
}
void trie::Delete(trieNode* t)
...{
    int i;
    for(i=0;i<keyNum;i++)
        if(t->link[i])Delete(t->link[i]);
    memset(t->num,0,sizeof(t->num));
    delete(t);
}
void trie::Insert(char x[])
...{
    trieNode *current=root;
    int i=1;
    while(x[i])...{
        if(current->link[x[i-1]-'a']==NULL)...{
            current->link[x[i-1]-'a']=(trieNode*)malloc(sizeof(trieNode));
            (current->link[x[i-1]-'a'])->init();
        }
        current=current->link[x[i-1]-'a'];
        i++;
    }
```

```
        (current->num[x[i-1]-'a'])++;
}
char c[ 50000 ][MaxN],tmp;
int main()
...{
        trie a;
        int i=0,j,num;
        while(scanf("%s",c[i])!=EOF)
                a.Insert(c[i++]);
        num=i;
        for(i=0;i<num;i++)
                for(j=1;c[i][j];j++)...{
                        tmp=c[i][j];
                        c[i][j]=0;
                        if(a.Search(c[i]))...{
                                c[i][j]=tmp;
                                if(a.Search(&c[i][j]))...{
                                        printf("%s ",c[i]);
                                        break;}
                        }
                        else c[i][j]=tmp;
                }
        a.Delete(a.root);
        return 0;
}
```

二叉查找树

```
//key 是字符串
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct BiTNode
{
        char data[31];
        struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;

BiTree p=NULL;

void InorderBST(BiTree T)
{
        if(T->lchild) InorderBST(T->lchild);
```

```
    printf("%s\n",T->data);
    if(T->rchild) InorderBST(T->rchild);
}

int SearchBST(BiTree T,char key[],BiTree f)
{
int tmp1,tmp2;
tmp1=tmp2=0;
if(!T) {p=f;return 0;}\\找不到返回路径上最后一点
else if(!strcmp(key,T->data)) {p=T;return 1;}
else if(strcmp(key,T->data)<0) tmp1=SearchBST(T->lchild,key,T);
else tmp2=SearchBST(T->rchild,key,T);
if(tmp1||tmp2) return 1;
else return 0;
}

BiTree InsertBST(BiTree T,char e[])
{
    BiTree s;
    if(!SearchBST(T,e,NULL))//查找不到，（若已经存在则放弃）
    {
        s=(BiTree)malloc(sizeof(BiTNode));
        strcpy(s->data,e);
        s->lchild=s->rchild=NULL;
        if(!p) return s;
        else
        {
            if(strcmp(e,p->data)<0) p->lchild=s;
            else p->rchild=s;
        }
    }
    //else      //若出现过做相应处理
}

/*int main()
{
    BiTree T=NULL;
    char key[31];
    while(gets(key)!=NULL)
    {
        if(T==NULL)
            T=InsertBST(T,key);//空树将返回指向根节点的指针
        else InsertBST(T,key);
    }
```

```
    InorderBST(T);
    return 0;
}*/
```

线段树

```
#include<stdio.h>
#include<memory.h>
#define MAX 100001
int up;
int count[31];
struct NODE
{
    int l,r,flag;
    NODE *lcd,*rcd;
    void build(int i,int j);
    void insert(int i,int j,int k);
    void get(int i,int j);
}node[MAX*4],*root=&node[0];
void NODE::build(int i,int j)
{
    l=i;
    r=j;
    flag=1;
    if(l==r){
        lcd=rcd=NULL;
    }
    else{
        lcd=&node[up++];
        rcd=&node[up++];
        lcd->build(l,(l+r)/2);
        rcd->build((l+r)/2+1,r);
    }
}
void NODE::insert(int i,int j,int k)
{
    if(i>r||j<l)return;
    if(i<=l&&j>=r){
        flag=k;
        return;
    }
    else{
        if(flag!=-1){
            lcd->insert(l,r,flag);
```

```
                rcd->insert(l,r,flag);
            }
            lcd->insert(i,j,k);
            rcd->insert(i,j,k);
            flag=-1;
        }
}
void NODE::get(int i,int j)
{
        if(i>r||j<l)return;
        if(flag!=-1){
            count[flag]=1;
        }
        else{
            lcd->get(i,j);
            rcd->get(i,j);
        }
}
int main()
{
        int l,t,o,i,j,k,ans;
        char s[5];
//      freopen("in.txt","r",stdin);
        scanf("%d%d%d",&l,&t,&o);
        up=1;
        root->build(1,l);
        while(o--){
            scanf("%s",s);
            if(s[0]=='C'){
                scanf("%d%d%d",&i,&j,&k);
                root->insert(i,j,k);
            }
            else{
                scanf("%d%d",&i,&j);
                memset(count,0,sizeof(count));
                root->get(i,j);
                ans=0;
                for(i=1;i<=t;i++){
                    if(count[i]){
                        ans++;
                    }
                }
                printf("%d\n",ans);
            }
```

```
    }
    return 0;
}
```

线段树 动态运用

```
#include<stdio.h>
#include<memory.h>
#include<algorithm>
using namespace std;
#define MAXN 1000001
#define MAXM 50001
int value[MAXN],dog[MAXN],b[MAXM],e[MAXM],rank[MAXM],ans[MAXM],ind[MAXM];
int up=1;
struct NODE
{
    int now,count;
    NODE *rcd,*lcd;
    void build(int i,int j);
    void insert(int key);
    void dele(int key);
    int get(int k);
}node[MAXN*4],*root=&node[0];
bool cmp(int p,int q)
{
    return b[p]<b[q]||b[p]==b[q]&&e[p]<e[q];
}
void NODE::build(int i,int j)
{
    now=(i+j)/2;
    if(i==j){
        lcd=rcd=NULL;
    }
    else{
        lcd=&node[up++];
        rcd=&node[up++];
        lcd->build(i,(i+j)/2);
        rcd->build((i+j)/2+1,j);
    }
}
void NODE::insert(int key)
{
    count++;
    if(lcd==NULL)return;
```

```
        if(key<=value[now]){
                lcd->insert(key);
        }
        else{
                rcd->insert(key);
        }
}
void NODE::dele(int key)
{
        count--;
        if(lcd==NULL)return;
        if(key<=value[now]){
                lcd->dele(key);
        }
        else{
                rcd->dele(key);
        }
}
int NODE::get(int k)
{
        if(lcd==NULL)return value[now];
        if(lcd->count>=k){
                return lcd->get(k);
        }
        else{
                return rcd->get(k-lcd->count);
        }
}
int main()
{
        int n,m,i,j;
//      freopen("d:/in.txt","r",stdin);
        scanf("%d%d",&n,&m);
        for(i=0;i<n;i++){
                scanf("%d",&dog[i]);
                value[i]=dog[i];
        }
        sort(value,value+n);
        n=unique(value,value+n)-value;
        root->build(0,n-1);
        for(i=0;i<m;i++){
                scanf("%d%d%d",&b[i],&e[i],&rank[i]);
                b[i]--;
                e[i]--;
```

```
            ind[i]=i;
        }
        sort(ind,ind+m,cmp);
        for(i=b[ind[0]];i<=e[ind[0]];i++){
            root->insert(dog[i]);
        }
        ans[ind[0]]=root->get(rank[ind[0]]);
        for(i=1;i<m;i++){
            for(j=b[ind[i-1]];j<b[ind[i]];j++){
                root->dele(dog[j]);
            }
            for(j=e[ind[i-1]]+1;j<=e[ind[i]];j++){
                root->insert(dog[j]);
            }
            ans[ind[i]]=root->get(rank[ind[i]]);
        }
        for(i=0;i<m;i++){
            printf("%d\n",ans[i]);
        }
        return 0;
}
```

RMQ

```
void RMQ(int M[MAXN][LOGMAXN], int A[MAXN], int N)
   {
        int i, j;

   //initialize M for the intervals with length 1
        for (i = 0; i < N; i++)
         M[i][0] = i;
   //compute values from smaller to bigger intervals
       for (j = 1; 1 << j <= N; j++)
          for (i = 0; i + (1 << j) - 1 < N; i++)
              if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])
                   M[i][j] = M[i][j - 1];
             else
                 M[i][j] = M[i + (1 << (j - 1))][j - 1];
   }
//产生的 M[i][j]为[ i , j ]中最小的数
//查询：[ l , r ]
//k:=ln((r-l+1)/ln(2));      ans:=min(M[l，k],M[r-2^k+1,k]);
```

LCA+RMQ

```
#include <stdio.h>
#include <math.h>
#include <memory.h>
#define MAX 1000

int tree[MAX][MAX],first[MAX];
int len[MAX],count[MAX];
int node[2*MAX],deep[2*MAX],top;
int M[MAX][32],in[MAX];
int N;

void dfs(int u,int pre,int d)
{
	int i,t;
	node[top] = u;
	deep[top] = d;
	if(first[u]==-1)first[u] = top;
	top++;
	for(i=0;i<len[u];i++)
	{
		t = tree[u][i];
		if(pre==t)continue;
		dfs(t,u,d+1);
		node[top] = u;
		deep[top] = d;
		top++;
	}
}

void RMQ_init()
{
	int i,j;
	for(i=0;i<top;i++)
		M[i][0] = i;
	for(j=1;(1<<j)<=top;j++)
	for(i=0;i+(1<<j)-1<top;i++)
		if(deep[M[i][j-1]]<deep[M[i+(1<<(j-1))][j-1]])
			M[i][j] = M[i][j-1];
		else
			M[i][j] = M[i+(1<<(j-1))][j-1];
}

int RMQ_query(int u,int v)
```

```
{
	int l,r,t;
	l = first[u];
	r = first[v];
	if(l>r){
		t = l;
		l = r;
		r = t;
	}
	t = log((double)(r-l+1))/log(2.0);
	if(deep[M[l][t]]<=deep[M[r-(1<<t)+1][t]])
		return node[M[l][t]];
	else
		return node[M[r-(1<<t)+1][t]];
}

int main()
{
	//freopen("in.txt","r",stdin);

	int i,u,v,k,j,T,s;
	char ch;
	while(scanf("%d",&N)!=EOF)
	{
		memset(len,0,sizeof(len));
		memset(in,0,sizeof(in));
		for(i=1;i<=N;i++)
		{
			scanf("%d",&u);
			while((ch=getchar())!=':');
			while((ch=getchar())!='(');
			scanf("%d",&k);
			while((ch=getchar())!=')');
			for(j=1;j<=k;j++)
			{
				scanf("%d",&v);
				in[v]++;
				tree[u][len[u]++] = v;
				//tree[v][len[v]++] = u;
			}
		}
		top = 0;
		memset(first,-1,sizeof(first));
		for(i=1;i<=N;i++)if(in[i]==0){
```

```
                s = i;
                break;
            }
            dfs(s,-1,0);
            RMQ_init();
            scanf("%d",&T);
            memset(count,0,sizeof(count));
            for(i=1;i<=T;i++)
            {
                while((ch=getchar())!='(');
                scanf("%d%d",&u,&v);
                while((ch=getchar())!=')');
                s = RMQ_query(u,v);
                count[s]++;
            }
            for(i=1;i<=N;i++)
                if(count[i]>0)printf("%d:%d\n",i,count[i]);
        }
        return 0;
}
```

SB-Tree

```
#include <stdio.h>
int n,f;

void solve(int a,int b,int c,int d)
{ //0,1,1,1
    if (b+d>n) return;
    solve(a,b,a+c,b+d);
    if(f)
    {
        printf("%d/%d",a+c,b+d);
        f=0;
    }
    else
        printf(",%d/%d",a+c,b+d);
    solve(a+c,b+d,c,d);
};
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
```

```
    {
        scanf("%d",&n);
        f=1;
        solve(0,1,1,1);
        printf("\n");
    }
    return 0;
}
```