2017-8-11

FangLingfei

pick



XTUACM模板

目录

[字符串处理 2](#_Toc491685873)

[KMP 2](#_Toc491685874)

[扩展KMP 3](#_Toc491685875)

[字典树 3](#_Toc491685876)

[后缀数组-da 5](#_Toc491685877)

[数据结构 8](#_Toc491685878)

[一维树状数组 8](#_Toc491685879)

[二维树状数组 9](#_Toc491685880)

[二维线段树 10](#_Toc491685881)

[KD树 12](#_Toc491685882)

[RMQ 14](#_Toc491685883)

[树链剖分-点 15](#_Toc491685884)

[树链剖分-边 20](#_Toc491685885)

[图论 23](#_Toc491685886)

[最小生成树kruskal 23](#_Toc491685887)

[单源最短路Dijkstra 24](#_Toc491685888)

[LCA-tarjan-离线 26](#_Toc491685889)

[LCA-RMQ-在线 28](#_Toc491685890)

[动态规划 30](#_Toc491685891)

[01背包 30](#_Toc491685892)

[最长连续子序列的和 30](#_Toc491685893)

[其他 30](#_Toc491685894)

[矩阵快速幂 30](#_Toc491685895)

[大数（bigdecimal） 31](#_Toc491685896)

[C++STL 33](#_Toc491685897)

[其余比较方便的STL 36](#_Toc491685898)

# 字符串处理

## KMP

///kmp模板

///模式串在文本串中出现了多少次

const int maxn = 1005 ;

int f[maxn] ;

int ans =0 ;

void find(char \*T,char \*P){

int n = strlen(T);

int m = strlen(P) ;

int j = 0 ;

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

while(j&&P[j]!=T[i])j = f[j] ;

if(P[j]==T[i])j++ ;

if(j==m){

ans++ ;

j = 0 ;///匹配不重合，如果要重合的话j=f[j]

}

}

}

void getfail(char \*p){

int m = strlen(p) ;

f[0] = 0 ; f[1] = 0 ;

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

int j = f[i] ;

while(j&&p[i]!=p[j])j = f[j] ;

f[i+1] = (p[i]==p[j] ? j+1 : 0) ;

}

}

char str[maxn] ;

char p[maxn] ;

int main(){

while(~scanf("%s %s ",str,p)){

getfail(p);

find(str,p) ;

printf("%d\n",ans);

}

}

## 扩展KMP

///扩展kmp，求一个每个字母开始多少个位置和原串重合

///主要的就是Next数组

using namespace std;

const int N = 200005;

int Next[N];

void getNext(char \*T) {

int i,length = strlen(T);

Next[0] = length;

for(i = 0; i<length-1 && T[i]==T[i+1]; i++);

Next[1] = i;

int a = 1;

for(int k = 2; k < length; k++) {

int p = a+Next[a]-1, L = Next[k-a];

if((k-1)+L>=p) {

int j = (p-k+1)>0? (p-k+1) : 0;

while(k+j<length && T[k+j]==T[j]) j++;

Next[k] = j, a = k;

} else Next[k] = L;

}

}

abbabbabb

900600300

## 字典树

///给一堆单词，获得每一个单词的最佳缩写

///样例输入就是那些单词

///样例输出-最佳前缀

const int maxnode = 1000 \* 30;///构建树的所有字母

const int sigma\_size = 30;///出现的字符种类

char data[1005][30] ;

struct Trie {

int ch[maxnode][sigma\_size];

int val[maxnode];

int num[maxnode];

int sz;

void clear(){

sz = 1;

memset(ch[0], 0, sizeof(ch[0]));

memset(num, 0, sizeof(num));

}

int idx(char c) { return c - 'a'; }///字符到数字转换

void insert(const char \*s, int v) {

int u = 0, n = strlen(s);

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

int c = idx(s[i]);

if(!ch[u][c]) {///如果没有节点，那就扩展一个节点

memset(ch[sz], 0, sizeof(ch[sz]));

val[sz] = 0;

num[sz] = 0 ;

ch[u][c] = sz++;///扩展节点编号增加

}

u = ch[u][c];

num[u]++ ;

}

val[u] = v;

}

void find(char \*s,int x){///寻找字符串s

int u = 0;

int p = 0;

for(int i = 0; i < strlen(s); i++) {

if(s[i] == '\0'||num[u]==1)break;

int c = idx(s[i]);

u = ch[u][c];

p++ ;

}

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

printf("%c",s[i]);

}printf("\n");

}

};

char str[maxnode] ;

Trie trie;

int main() {

int c = 1 ;

trie.clear();

while(~scanf("%s",str)){

trie.insert(str, c);

strcpy(data[c++],str) ;

}

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

printf("%s ",data[i]);

trie.find(data[i],i);

}

return 0;

}

//carbohydrate carboh

//cart cart

//carburetor carbu

//caramel cara

//caribou cari

//carbonic carboni

//cartilage carti

//carbon carbon

//carriage carr

//carton carto

//car car

//carbonate carbona

## 后缀数组-da

题目大意：

* Given two strings *A*, *B* and one integer *K*, we define *S*, a set of triples (*i*, *j*, *k*):
* 表示从A的i和B的j开始长度为k的子串是相同的，k>=K
* 求有多少个这样的三元组
* 从const int 到main函数之间都是模板

解题思路：

* 两个串A串和B串拼在一起，用一个没有出现过的字符隔离开来
* 然后跑一边后缀数组求出height
* 然后通过题目给的k进行分组，题目就转化为，在一个分组内如何找出所有A和B的公共子串，并算出三元组的个数
* 使用单调栈，在一个分组内部，对于每一个B找前面有多少个A串，找到之后通过算出他们之间的最小值算出重复了多少个字符，但是这样做的复杂度是n^2的
* 通过使用单调栈，每遇到一个字符串，如果是A串我把它就放单调栈里并算出对 结果的贡献，如果是B串也放单调栈里，但是不算贡献，tot一直维护当前栈的总的值，s[maxn][1]表示单调栈中的一个元素覆盖了多少个。比如1，4，3，5由于3不能在单调栈中保存，我就要删了，但是实际在B去算的时候，他是存在的，那么就要在5这个元素记录，在我这我前面删除了s[5的pos][1]个元素，然后动态维护tot即可。

#include<bits/stdc++.h>

#define LL long long

using namespace std ;

const int maxn = 100005<<1 ;

int wa[maxn],wb[maxn],wv[maxn],wss[maxn];

int cmp(int \*r,int a,int b,int l) {

return r[a]==r[b]&&r[a+l]==r[b+l];

}

void da(int \*r,int \*sa,int n,int m) {

int i,j,p,\*x=wa,\*y=wb,\*t;

for(i=0; i<m; i++) wss[i]=0;

for(i=0; i<n; i++) wss[x[i]=r[i]]++;

for(i=1; i<m; i++) wss[i]+=wss[i-1];

for(i=n-1; i>=0; i--) sa[--wss[x[i]]]=i;

for(j=1,p=1; p<n; j\*=2,m=p) {

for(p=0,i=n-j; i<n; i++) y[p++]=i;

for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;

for(i=0; i<n; i++) wv[i]=x[y[i]];

for(i=0; i<m; i++) wss[i]=0;

for(i=0; i<n; i++) wss[wv[i]]++;

for(i=1; i<m; i++) wss[i]+=wss[i-1];

for(i=n-1; i>=0; i--) sa[--wss[wv[i]]]=y[i];

for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)

x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;

}

return;

}

int rrank[maxn],height[maxn];

void calheight(int \*r,int \*sa,int n) {

int i,j,k=0;

for(i=1; i<=n; i++) rrank[sa[i]]=i;

for(i=0; i<n; height[rrank[i++]]=k)

for(k?k--:0,j=sa[rrank[i]-1]; r[i+k]==r[j+k]; k++);

return;

}

int RMQ[maxn];

int mm[maxn];

int best[20][maxn];

void initRMQ(int n) {

int i,j,a,b;

for(mm[0]=-1,i=1; i<=n; i++)

mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

for(i=1; i<=n; i++) best[0][i]=i;

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

for(j=1; j<=n+1-(1<<i); j++) {

a=best[i-1][j];

b=best[i-1][j+(1<<(i-1))];

if(RMQ[a]<RMQ[b]) best[i][j]=a;

else best[i][j]=b;

}

return;

}

int askRMQ(int a,int b) {

int t;

t=mm[b-a+1];

b-=(1<<t)-1;

a=best[t][a];

b=best[t][b];

return RMQ[a]<RMQ[b]?a:b;

}

int lcp(int a,int b) {

int t;

a=rrank[a];

b=rrank[b];

if(a>b) {

t=a;

a=b;

b=t;

}

return(height[askRMQ(a+1,b)]);

}

char str1[maxn] ;

char str2[maxn] ;

int r[maxn] ;

int sa[maxn] ;

int k ;

int main(){

while(~scanf("%d",&k)){

if(k==0)break ;

scanf("%s%s",str1,str2) ;

int len1 = strlen(str1) ;

int len2 = strlen(str2) ;

str1[len1] = '$' ;

strcpy(&str1[len1+1],str2) ;

int len = strlen(str1) ;

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

r[i] = str1[i] ;

}

r[len] = 0 ;

da(r,sa,len+1,200) ;

calheight(r,sa,len) ;

int s[maxn][2];

LL tot,top;

tot=top=0;

LL sum=0;

///先对于每一个B找前面有多少个A

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

if(height[i]<k)top=tot=0;

else{

int cnt=0;

if(sa[i-1]<len1){

cnt++;

tot+=height[i]-k+1;

}

while(top>0&&height[i]<=s[top-1][0]){

top--;

tot-=s[top][1]\*(s[top][0]-height[i]);

cnt+=s[top][1];

}

s[top][0]=height[i];

s[top++][1]=cnt;

if(sa[i]>len1){

sum+=tot;

}

}

}

///先对于每一个A找前面有多少个B

tot=top=0;

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

if(height[i]<k) top=tot=0;///新区间更新

else{

int cnt=0;

if(sa[i-1]>len1){///如果属于B串

cnt++;

tot+=height[i]-k+1;

}

while(top>0&&height[i]<=s[top-1][0]){

top--;

tot-=s[top][1]\*(s[top][0]-height[i]);

cnt+=s[top][1];

}

s[top][0]=height[i];

s[top++][1]=cnt;

if(sa[i]<len1) sum+=tot;

}

}

printf("%I64d\n",sum);

}

}

# 数据结构

## 一维树状数组

///单点修改，区间求和

///区间修改，单点求值

///区间修改，区间求和

int A[maxn],C[maxn];

int lowbit(int x){

return x&(-x) ;

}

int add(int x,int v){

A[x]+=v ;

for(int i=x;i<=maxn;i+=lowbit(x)){

C[i]+=v ;

}

}

int sum(int x){

int ret = 0 ;

for(int i=x;i>0;i-=lowbit(x)){

ret+=C[i] ;

}

return ret ;

}

///区间修改，单点查询

构造A数组的一个差分数组B，对B同样使用树状数组的做法

所谓差分数组，就是从i=2开始，记录A[i]-A[i-1]的差值，(B[1]=A[1])

区间修改（L，R，V）就是B[L]+=v,B[R+1]+=v，

然后求B的1~i的前缀和的时候，就求出了A[i]

///区间修改，区间查询

构造差分数组B，注意到B的求1~6的前缀和，就求出了A[6]

B[1], B[2], B[3], B[4], B[5], B[6],

## 二维树状数组

//只做区间修改，单点查询

int lowbit(int x){return x&-x ;}

void add(int x,int y,int v){

for(int i=x;i<=n;i+=lowbit(i)){

for(int j=y;j<=n;j+=lowbit(j)){

C[i][j]+=v ;

}

}

}

int sum(int x,int y){

int ret = 0;

for(int i=x;i>0;i-=lowbit(i)){

for(int j=y;j>0;j-=lowbit(j)){

ret+=C[i][jj] ;

}

}

return ret;

}

///区间修改：

给（x,y）到（x1，y1）区间加V

add(x,y,V);

add(x,y1+1,-V);

add(x1+1,y,-V);

add(x1+1,y1+1,V);

//单点查询：sum(x,y)

## 二维线段树

输入一个n\*n的方阵q次操作

一种操作，输入x,y表示对于在x，y位置的数字，

在以他它为中心的l方阵里找一个最大值和最小值，然后输出（max+min）/2

然后用（max+min）/2替换这个数

树套树模板

#include<bits/stdc++.h>

#define lson l,m,rt<<1

#define rson m+1,r,rt<<1|1

using namespace std ;

const int maxn = 805 ;

const int INF = 0x3f3f3f3f;

int Max[maxn<<2][maxn<<2] ;

int Min[maxn<<2][maxn<<2] ;

int n,mat[maxn][maxn] ;

int minv,maxv ;

int xL,xR,yR,yL,val;

int pushup(int xrt,int rt){

Max[xrt][rt] = max(Max[xrt][rt<<1],Max[xrt][rt<<1|1]) ;

Min[xrt][rt] = min(Min[xrt][rt<<1],Min[xrt][rt<<1|1]) ;

}

void buildy(int xrt,int x,int l,int r,int rt){

if(l==r){

if(x!=-1)Max[xrt][rt] = Min[xrt][rt] = mat[x][l] ;

else{

Max[xrt][rt] = max(Max[xrt<<1][rt],Max[xrt<<1|1][rt]) ;

Min[xrt][rt] = min(Min[xrt<<1][rt],Min[xrt<<1|1][rt]) ;

}

return ;

}

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

buildy(xrt,x,lson);

buildy(xrt,x,rson);

pushup(xrt,rt) ;

}

void buildx(int l,int r,int rt){

if(l==r){

buildy(rt,l,1,n,1);

return ;

}

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

buildx(lson);

buildx(rson);

buildy(rt,-1,1,n,1) ;

}

void updatey(int xrt,int x,int l,int r,int rt){

if(l==r){

if(x!=-1)Max[xrt][rt] = Min[xrt][rt] = val ;

else{

Max[xrt][rt] = max(Max[xrt<<1][rt],Max[xrt<<1|1][rt]) ;

Min[xrt][rt] = min(Min[xrt<<1][rt],Min[xrt<<1|1][rt]) ;

}

return ;

}

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

if(yL<=m)updatey(xrt,x,lson) ;

else updatey(xrt,x,rson) ;

pushup(xrt,rt) ;

}

void updatex(int l,int r,int rt){

if(l==r){

updatey(rt,l,1,n,1) ;

return ;

}

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

if(xL<=m)updatex(lson) ;

else updatex(rson) ;

updatey(rt,-1,1,n,1) ;

}

void queryy(int xrt,int l,int r,int rt){

if(yL<=l&&r<=yR){

minv = min(minv,Min[xrt][rt]) ;

maxv = max(maxv,Max[xrt][rt]) ;

return ;

}

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

if(yL<=m)queryy(xrt,lson) ;

if(yR>m)queryy(xrt,rson) ;

}

void queryx(int l,int r,int rt){

if(xL<=l&&r<=xR){

queryy(rt,1,n,1) ;

return ;

}

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

if(xL<=m)queryx(lson) ;

if(xR>m)queryx(rson) ;

}

int main(){

int T,cas=1;

scanf("%d",&T) ;

while(T--){

scanf("%d",&n) ;

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

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

scanf("%d",&mat[i][j]) ;

}

}

buildx(1,n,1) ;

int q ;

scanf("%d",&q) ;

printf("Case #%d:\n",cas++) ;

while(q--){

int x,y,l ;

scanf("%d%d%d",&x,&y,&l);

l = (l+1)>>1;

xL = max(1,x-l+1) ;xR = min(n,x+l-1) ;

yL = max(1,y-l+1) ;yR = min(n,y+l-1) ;

minv = INF ;

maxv = -INF ;

queryx(1,n,1) ;

val = (maxv+minv)>>1 ;

xL = x ;yL = y ;

printf("%d\n",val) ;

updatex(1,n,1) ;

}

}

}

## KD树

//HDU 4347 输入n个点t次查询，然后是n个点，然后每次查询，输入m和一个点

//输出，最近的m个点

const int N=55555,K=5;

const int inf=0x3f3f3f3f;

#define sqr(x) (x)\*(x)

int k,n,idx; ///k为维数,n为点数

struct point {

int x[K];

bool operator < (const point &u) const {

return x[idx]<u.x[idx];

}

} po[N];

typedef pair<double,point> tp;///pair，排序先按照double从小到大，再按照point的排

priority\_queue<tp>nq;

struct kdTree {

point pt[N<<2];///树上的所有节点

int son[N<<2];///树上的所有节点、记录的是每个节点的极差

void build(int l,int r,int rt=1,int dep=0) {///默认参数

if(l>r) return;///边界

son[rt]=r-l;///son保存的是范围差，跨度-1

son[rt\*2]=son[rt\*2+1]=-1;///初始化左儿子、右儿子

idx=dep%k;///按照维度循环作为划分

int mid=(l+r)/2;///中间数的选取、选的是第mid大的数

///部分排序，第mid大的数在mid的位置，比它小的前面，比他大的在后面

///[first,last),nth\_element(array,array+m,array+n);

nth\_element(po+l,po+mid,po+r+1);

pt[rt]=po[mid];///把第mid大的点存在树中

build(l,mid-1,rt\*2,dep+1);///递归建立左儿子

build(mid+1,r,rt\*2+1,dep+1);///递归建立右儿子

}

void query(point p,int m,int rt=1,int dep=0) {///查询点p的最近m个相邻的点

if(son[rt]==-1) return;///边界

tp nd(0,pt[rt]);///pair的构造器

///计算分界的节点到p点的距离

for(int i=0;i<k;i++)nd.first+=sqr(nd.second.x[i]-p.x[i]);

int dim=dep%k;

int x=rt<<1,y=rt<<1|1,fg=0;

///如果p在dim维度上坐标大于等于父节点的值

if(p.x[dim]>=pt[rt].x[dim])swap(x,y);

///如果有儿子

if(son[x]!=-1) query(p,m,x,dep+1);

if(nq.size()<m) {///如果加入的点的数量小于m

nq.push(nd),fg=1;///放进去

} else {

if(nd.first<nq.top().first) {///如果节点小于里面最大的

nq.pop();///更新之

nq.push(nd);

}

///如果在dim维度上的差小于里面的，就认为另外一个相邻分支有可能有更近的点

if(sqr(p.x[dim]-pt[rt].x[dim])<nq.top().first) fg=1;

}

///如果有儿子，并且被认为是有更近的则查找之

if(son[y]!=-1&&fg)query(p,m,y,dep+1);

}

} kd;

void print(point &p) {

for(int j=0; j<k; j++) printf("%d%c",p.x[j],j==k-1?'\n':' ');

}

int main() {

while(scanf("%d%d",&n,&k)!=EOF) {

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

for(int j=0; j<k; j++)

scanf("%d",&po[i].x[j]);

kd.build(0,n-1); ///建立kd树

int t,m;

scanf("%d",&t) ;

while(t--) {

point ask;

for(int j=0; j<k; j++)scanf("%d",&ask.x[j]);

scanf("%d",&m);

kd.query(ask,m);

printf("the closest %d points are:\n", m);

point pt[20];

for(int j=0; !nq.empty(); j++) {

pt[j]=nq.top().second;

nq.pop();

}

for(int j=m-1; j>=0; j--)print(pt[j]);

}

}

}

## RMQ

//POJ 3368给出一个非降序排列的整数数组a1,a2,...an，

//你的任务是对于一系列询问(i, j)，回答ai,ai+1,...aj中出现最多次数的值所出现的次数？

const int maxn = 100005 ;

int value[maxn] ;

int num[maxn],left[maxn],right[maxn] ;

int d[maxn][50],d1[maxn][50],d2[maxn][50] ;

int n ;

void RMQ\_init(){

for(int i=0;i<n;i++)d[i][0] = num[i] ;

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

for(int i=0;i+(1<<j)-1<n;i++){

d[i][j] = max(d[i][j-1],d[i+(1<<(j-1))][j-1]) ;

}

}

}

int RMQ(int L,int R){

if(L>R)return 0;

int k=0;

while((1<<(k+1)) <= R-L+1) k++;

return max(d[L][k],d[R-(1<<k)+1][k]) ;

}

int main(){

int q;

while(~scanf("%d",&n)){

if(n==0)break ;

scanf("%d",&q) ;

for(int i=0;i<n;i++)scanf("%d",&value[i]) ;

left[0] = 0 ;

for(int i=1;i<n;i++)left[i]=(value[i]==value[i-1]?left[i-1]:i);

right[n-1]=n-1 ;

for(int i=n-2;i>=0;i--)right[i]=(value[i]==value[i+1]?right[i+1]:i) ;

for(int i=0;i<n;i++)num[i] = right[i]-left[i] + 1 ;

RMQ\_init() ;

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

int l,r ;

scanf("%d%d",&l,&r) ;

l--;r--;

if(left[l]==left[r]&&right[l]==right[r]){

printf("%d\n",r-l+1) ;

continue;

}

int ans = max(right[l]-l+1,r-left[r]+1);

ans = max(ans,RMQ(right[l]+1,left[r]-1));

printf("%d\n",ans) ;

}

}

return 0;

}

## 树链剖分-点

* 区间合并、区间查询、区间赋值。
* 区间合并，注意查询的时候区间两端的合并操作，在query和findnum中均有体现。

给定一棵有n个节点的无根树和m个操作，操作有2类：

1、将节点a到节点b路径上所有点都染成颜色c；

2、询问节点a到节点b路径上的颜色段数量,如“112221”由3段组成：“11”、“222”和“1”

n和m，分别表示节点数和操作数

n个正整数表示n个节点的初始颜色

两个整数x和y，表示*x*和*y*之间有一条无向边

“C a b c”表示这是一个染色操作，把节点a到节点b路径上所有点（包括a和b）都染成颜色c；

“Q a b”表示这是一个询问操作，询问节点a到节点b（包括a和b）路径上的颜色段数量。

#include<bits/stdc++.h>

#define lson l,m,rt<<1

#define rson m+1,r,rt<<1|1

using namespace std ;

const int maxn = 100005 ;

struct Edge{

int to,next ;

};

Edge edges[maxn<<1] ;

int head[maxn];

int fa[maxn],deep[maxn],son[maxn],top[maxn],num[maxn];

int p[maxn],fp[maxn] ;

int tot,pos ;

void init(){

tot = 0 ;

pos = 1 ;

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

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

}

void addedge(int u,int v){

edges[tot].to=v ;

edges[tot].next = head[u] ;

head[u] = tot++ ;

}

void dfs1(int u,int pre,int d){

deep[u] = d ;

fa[u] = pre ;

num[u] = 1;

for(int i=head[u];i!=-1;i=edges[i].next){

int v = edges[i].to ;

if(v!=pre){

dfs1(v,u,d+1) ;

num[u] += num[v] ;

if(son[u]==-1||num[v]>num[son[u]]){

son[u] = v ;

}

}

}

}

void getpos(int u,int sf){

top[u] = sf ;

p[u] = pos++ ;

fp[u] = u ;

if(son[u]==-1)return ;

getpos(son[u],sf) ;

for(int i=head[u];i!=-1;i=edges[i].next){

int v = edges[i].to ;

if(v!=son[u]&&v!=fa[u]){

getpos(v,v) ;

}

}

}

int lc[maxn<<2],rc[maxn<<2],setc[maxn<<2],numc[maxn<<2];

int n ;

void pushup(int rt){

lc[rt] = lc[rt<<1] ;

rc[rt] = rc[rt<<1|1] ;

numc[rt] =numc[rt<<1|1]+numc[rt<<1]+(rc[rt<<1]==lc[rt<<1|1]?-1:0) ;

}

void pushdown(int rt){

if(setc[rt]!=-1){

lc[rt<<1] = setc[rt] ;

rc[rt<<1] = setc[rt] ;

lc[rt<<1|1] = setc[rt] ;

rc[rt<<1|1] = setc[rt] ;

setc[rt<<1] = setc[rt] ;

setc[rt<<1|1] = setc[rt] ;

numc[rt<<1] = 1 ;

numc[rt<<1|1] = 1 ;

setc[rt] = -1 ;

}

}

void update(int L,int R,int val,int l,int r,int rt){

if(L<=l&&r<=R){

setc[rt] = val ;

numc[rt] = 1 ;

lc[rt] = val ;

rc[rt] = val ;

return ;

}

pushdown(rt) ;

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

if(L<=m)update(L,R,val,lson) ;

if(R>m)update(L,R,val,rson) ;

pushup(rt) ;

}

void change(int u,int v,int val){

int f1 = top[u] ;

int f2 = top[v] ;

while(f1!=f2){

if(deep[f1]<deep[f2]){

swap(f1,f2) ;

swap(u,v) ;

}

update(p[f1],p[u],val,1,n,1) ;

u = fa[f1] ;

f1 = top[u] ;

}

if(deep[u]>deep[v])swap(u,v) ;

update(p[u],p[v],val,1,n,1) ;

}

int querycolor(int q,int l,int r,int rt){

if(q==l&&q==r){

return lc[rt] ;

}

pushdown(rt) ;

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

int ret = -1 ;

if(q<=m)ret = querycolor(q,lson) ;

else ret = querycolor(q,rson) ;

pushup(rt) ;

return ret ;

}

int query(int L,int R,int l,int r,int rt){

if(L<=l&&r<=R){

return numc[rt];

}

pushdown(rt) ;

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

int ret = 0 ;

if(R<=m)ret+=query(L,R,lson) ;

else if(L>m)ret+=query(L,R,rson) ;

else{

ret+=query(L,R,lson) ;

ret+=query(L,R,rson) ;

if(rc[rt<<1]==lc[rt<<1|1])ret--;

}

pushup(rt) ;

return ret ;

}

int findnum(int u,int v){

int f1 = top[u] ;

int f2 = top[v] ;

int tmp = 0 ;

while(f1!=f2){

if(deep[f1]<deep[f2]){

swap(f1,f2) ;

swap(u,v) ;

}

tmp+=query(p[f1],p[u],1,n,1) ;

int x = querycolor(p[f1],1,n,1) ;

int y = querycolor(p[fa[f1]],1,n,1) ;

if(x==y){

tmp-- ;

}

u = fa[f1] ;

f1 = top[u] ;

}

if(deep[u]>deep[v])swap(u,v) ;

tmp+=query(p[u],p[v],1,n,1);

return tmp ;

}

int color[maxn] ;

int m;

int main(){

while(~scanf("%d%d",&n,&m)){

init() ;

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

scanf("%d",&color[i]) ;

}

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

int a,b ;

scanf("%d%d",&a,&b) ;

addedge(a,b) ;

addedge(b,a) ;

}

dfs1(1,0,0) ;

getpos(1,1) ;

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

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

change(i,i,color[i]) ;

}

while(m--){

char op[20] ;

scanf("%s",op) ;

if(op[0]=='C'){

int a,b,c;

scanf("%d%d%d",&a,&b,&c) ;

change(a,b,c) ;

}else{

int a,b ;

scanf("%d%d",&a,&b) ;

printf("%d\n",findnum(a,b));

}

}

}

return 0;

}

## 树链剖分-边

* 边操作最终会映射到点上，注意和点查询有一点不用，就是查询的时候sum(p[son[u]],p[v]) ;

有n座城市，由n-1条路相连通，使得任意两座城市之间可达。

每条路有过路费，要交过路费才能通过。每条路的过路费经常会更新，

现问你，当前情况下，从城市a到城市b最少要花多少过路费。

有多组样例，每组样例第一行输入两个正整数n,m(2 <= n<=50000，1<=m <= 50000),

n-1行，每行3个正整数a b c，(1 <= a,b <= n , a != b , 1 <= c <= 1000000000)

m行，表示m个操作，操作有两种：

0 a b，表示更新第a条路的过路费为b，1 <= a <= n-1 ；

1 a b，表示询问a到b最少要花多少过路费。

#include<iostream>

#include<cstring>

#include<cstdio>

#define LL long long

using namespace std ;

const int maxn = 50005 ;

struct Edge{

int to,next ;

};

Edge edges[maxn<<1] ;

int head[maxn] ;

int tot,pos;

int son[maxn] ;

int deep[maxn] ;

int num[maxn] ;

int top[maxn] ;

int fa[maxn] ;

int p[maxn],fp[maxn] ;

void init(){

tot = 0 ;

pos = 1 ;

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

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

}

void addedge(int u,int v){

edges[tot].to = v ;

edges[tot].next = head[u] ;

head[u] = tot++ ;

}

void dfs1(int u,int pre,int d){

deep[u] = d ;

fa[u] = pre ;

num[u] = 1 ;

for(int i=head[u];i!=-1;i=edges[i].next){

int v = edges[i].to ;

if(v==pre)continue ;

dfs1(v,u,d+1) ;

num[u]+=num[v] ;

if(son[u]==-1||num[v]>num[son[u]]){

son[u] = v ;

}

}

}

void getpos(int u,int sf){

top[u] = sf ;

p[u] = pos++ ;

fp[p[u]] = u ;

if(son[u]==-1)return ;

getpos(son[u],sf) ;

for(int i=head[u];i!=-1;i=edges[i].next){

int v = edges[i].to ;

if(v!=son[u]&&v!=fa[u]){

getpos(v,v) ;

}

}

}

int n ;

LL C[maxn<<2] ;

int lowbit(int x){return x&-x;}

LL sum(int a,int b){

LL suma = 0;

for(int i=a-1;i>0;i-=lowbit(i)){

suma+=C[i] ;

}

LL sumb = 0;

for(int i=b;i>0;i-=lowbit(i)){

sumb+=C[i] ;

}

return sumb-suma ;

}

void update(int x,LL v){

LL pre = sum(x,x) ;

for(int i=x;i<=n;i+=lowbit(i)){

C[i]+=(-pre);

}

for(int i=x;i<=n;i+=lowbit(i)){

C[i]+=v ;

}

}

LL querysum(int u,int v){

int f1 = top[u] ;

int f2 = top[v] ;

LL ret = 0 ;

while(f1!=f2){

if(deep[f1]<deep[f2]){

swap(f1,f2) ;

swap(u,v) ;

}

ret+=sum(p[f1],p[u]) ;

u = fa[f1] ;

f1 = top[u] ;

}

if(deep[u]>deep[v])swap(u,v) ;

ret+=sum(p[son[u]],p[v]) ;

return ret ;

}

int a[maxn][3] ;

int m;

int main(){

while(~scanf("%d%d",&n,&m)){

init() ;

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

scanf("%d%d%d",&a[i][0],&a[i][1],&a[i][2]) ;

addedge(a[i][0],a[i][1]) ;

addedge(a[i][1],a[i][0]) ;

}

dfs1(1,0,0) ;

getpos(1,1) ;

memset(C,0,sizeof(C)) ;

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

int x = (deep[a[i][0]]>deep[a[i][1]]?a[i][0]:a[i][1]) ;

update(p[x],a[i][2]) ;

}

while(m--){

int op,aa,b ;

scanf("%d%d%d",&op,&aa,&b) ;

if(op&1){

if(aa==b){

printf("0\n") ;

continue ;

}

LL ans = querysum(aa,b) ;

printf("%I64d\n",ans) ;

}else{

int x = (deep[a[aa][0]]>deep[a[aa][1]]?a[aa][0]:a[aa][1]) ;

update(p[x],(LL)b) ;

}

}

}

}

# 图论

## 最小生成树kruskal

///给你一个对称矩阵，表示i，j的距离，让你求最小生成树

#include<bits/stdc++.h>

using namespace std;

const int maxn = 100005 ;

const int INF = 0x3f3f3f3f ;

int pa[maxn] ;

int mapp[105][105] ;

vector<Edge>e;

int n,m;

int findset(int x){///并查集

return pa[x]!=x?pa[x]=findset(pa[x]):x ;

}

struct Edge{ ///存边

int u,v,d ;

Edge(int u,int v,int d):u(u),v(v),d(d){}

bool operator < (const Edge& p)const {///按照边从小到大排序

return d<p.d ;

}

};

int kruskal(){

sort(e.begin(),e.end()) ;

int ans = INF ;

int ret = 0 ;

int cnt = 0 ;

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

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

int u=findset(e[i].u) ;

int v=findset(e[i].v) ;

if(u!=v){

pa[v]=u ;

cnt++ ;

ret += e[i].d;

if(cnt==n-1)break ;

}

}

return ret ;

}

int main(){

while(~scanf("%d",&n)){

e.clear();///清空边集

m=n\*(n-1)/2 ;///矩阵不包括主对角线上，一共（n\*n-n）/2条边。

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

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

int a ;

scanf("%d",&a);

if(i>j)e.push\_back(Edge(i,j,a));

}

}

printf("%d\n",kruskal());

}

return 0;

}

## 单源最短路Dijkstra

///单源最短路，没有判断负环

///边权可以为负数,复杂度nlogn

#include<bits/stdc++.h>

using namespace std;

struct Edge{///边的起点，终点，和长度

int from,to,dist;

};

struct HeapNode{///优先队列的结点

int d,u ;///s到u当前最短的距离为d

bool operator < (const HeapNode& rhs) const{

return d>rhs.d ;

}

};

const int maxn = 200005 ;

const int INF = 0x3f3f3f3f ;

int n,m;///点数和边数

vector<Edge> edges ;///边列表

vector<int> G[maxn] ;///每个结点出发的边编号

bool done[maxn] ;///是否已永久标号

int d[maxn] ;///s到各个点的距离

int p[maxn] ;///最短路中的上一条边

void init(){

for(int i=0;i<n;i++)G[i].clear() ;///清空邻接表

edges.clear() ;///清空边表

}

void AddEdge(int from,int to,int dist){///如果是无向图，每条边需要调用两次AddEdge

edges.push\_back((Edge){from,to,dist}) ;

int mm = edges.size() ;

G[from].push\_back(mm-1) ;

}

void Dijkstra(int s){///求s到所有点的距离

priority\_queue<HeapNode> Q ;

for(int i=0;i<n;i++)d[i] = INF ;

d[s]=0;

memset(done,0,sizeof(done)) ;

Q.push((HeapNode){0,s}) ;

while(!Q.empty()){

HeapNode x = Q.top() ;

Q.pop() ;

int u = x.u ;

if(done[u])continue ;///如果判断负环的话，这里可以看u点出队列多少次

done[u] = true ;

for(int i=0;i<G[u].size();i++){///遍历该点相连的每条边

Edge& e = edges[G[u][i]] ;

if(d[e.to]>d[u]+e.dist){///如果更短则更新

d[e.to] = d[u] + e.dist ;

p[e.to] = G[u][i] ;

Q.push((HeapNode){d[e.to],e.to}) ;

}

}

}

}

int main(){

while(cin>>n>>m){

init() ;

for(int i=0;i<m;i++){ int a,b,c;cin>>a>>b>>c ;a--;b--;AddEdge(a,b,c) ;}

Dijkstra(0) ;///此算法点的编号从0开始

for(int i=1;i<n;i++)cout<<d[i]<<endl;

}}

// 最短路

//给定一个n个顶点，m条边的有向图（其中某些边权可能为负，但保证没有负环）。

//计算从1号点到其他点的最短路（顶点从1到n编号）。

//样例输入

//3 3 3个点，3条边

//1 2 -1

//2 3 -1

//3 1 2

//样例输出

//-1

//-2

## LCA-tarjan-离线

///输入T样例数目、输入一个树n个顶点，m个查询，输出两个点的最短距离

#include<bits/stdc++.h>

using namespace std;

const int maxn = 40005 ;

struct Edge {

int form,to,dist;

};

vector<Edge> edges ;

vector<int>G[maxn] ;

void add\_edge(int from,int to,int dist) {

edges.push\_back((Edge){from,to,dist}) ;

int mm = edges.size() ;

G[from].push\_back(mm-1) ;

}

vector<Edge> edges2 ;

vector<int>G2[maxn] ;

void add\_edge2(int from,int to,int dist=0) {

edges2.push\_back((Edge){from,to,dist}) ;

int mm = edges2.size() ;

G2[from].push\_back(mm-1) ;

}

int f[maxn] ;

bool vis[maxn] ;

int len[maxn] ;

int find(int x){

return x==f[x]?x:f[x]=find(f[x]);

}

void LCA(int u){

f[u] = u ;

vis[u] = true ;

for(int i=0;i<G[u].size();i++){

if(!vis[edges[G[u][i]].to]){

len[edges[G[u][i]].to] = len[u]+edges[G[u][i]].dist;

LCA(edges[G[u][i]].to) ;

f[edges[G[u][i]].to] = u ;

}

}

for(int i=0;i<G2[u].size();i++){

if(vis[edges2[G2[u][i]].to]){///这句话要记牢

edges2[G2[u][i]].dist = len[edges2[G2[u][i]].to]+len[u]-2\*len[find(edges2[G2[u][i]].to)];

edges2[G2[u][i]^1].dist = edges2[G2[u][i]].dist

}

}

}

int main() {

int t ;

scanf("%d",&t) ;

while(t--) {

memset(G,0,sizeof(G)) ;

memset(G2,0,sizeof(G2)) ;

memset(len,0,sizeof(len)) ;

memset(vis,false,sizeof(vis)) ;

edges.clear() ;

edges2.clear() ;

int n,m;

scanf("%d%d",&n,&m) ;

for(int i=0; i<n-1; i++) {

int u,v,w ;

scanf("%d%d%d",&u,&v,&w) ;

add\_edge(u,v,w) ;

add\_edge(v,u,w) ;

}

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

int u,v ;

scanf("%d%d",&u,&v) ;

add\_edge2(u,v) ;

add\_edge2(v,u) ;

}

LCA(1) ;

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

printf("%d\n",edges2[2\*i].dist) ;

}

}

return 0;

}

## LCA-RMQ-在线

///ZOJ 3195 LCA在线做法

///LCA转RMQ 应用深度搜索，把树变成序列，按照两个点第一次出现的位置，给出范围

///找范围最小深度的位置，从而找到最近公共祖先

#include<iostream>

using namespace std;

const int maxn = 100005 ;

struct Edge {

int form,to,dist;

};

vector<Edge> edges ;

vector<int>G[maxn] ;

bool vis[maxn] ;///标记数组

int len[maxn] ;///节点到根的长度

int ver[maxn<<1] ;///遍历的节点序列

int first[maxn] ;///第一次出现的位置

int R[maxn<<1] ;///遍历的深度

int d[maxn][50] ;///dp数组，里面放的是RMQ的位置，不是数值

int n,tot ;

void add\_edge(int from,int to,int dist) {

edges.push\_back((Edge) {from,to,dist}) ;

int mm = edges.size() ;

G[from].push\_back(mm-1) ;

}

void dfs(int u,int dept) {

vis[u] = true ;

ver[tot] = u ;

R[tot] = dept ;

first[u] = tot ;

tot++ ;

for(int i=0; i<G[u].size(); i++) {

if(!vis[edges[G[u][i]].to]) {

len[edges[G[u][i]].to] = len[u]+ edges[G[u][i]].dist ;

dfs(edges[G[u][i]].to,dept+1) ;

ver[tot] = u ;

R[tot] = dept ;

tot++ ;

}

}

}

void RMQ\_init(){

for(int i=0;i<tot;i++){///初始化，存的是位置

d[i][0] = i ;

}

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

for(int i=0;i+(1<<j)-1<tot;i++){///取最深度最小时的位置

int a = d[i][j-1] ;

int b = d[i+(1<<(j-1))][j-1] ;

if(R[a]<R[b])d[i][j] = d[i][j-1] ;

else d[i][j] = d[i+(1<<(j-1))][j-1] ;

}

}

}

int RMQ(int l,int r){///返回范围最小深度的位置

if(l>r)return 0;

int k=0;

while((1<<(k+1)) <= r-l+1) k++;

int a = d[l][k] ;

int b = d[r-(1<<k)+1][k] ;

if(R[a]<R[b])return a ;

else return b ;

}

int LCA(int u ,int v) ///返回点u和点v的LCA

{

int x = first[u] , y = first[v];

if(x > y) swap(x,y);

int res = RMQ(x,y);

return ver[res];

}

int main() {

int x=0;

while(~scanf("%d",&n)) {

if(x==1)printf("\n") ;

x=1;

memset(vis,false,sizeof(vis)) ;

memset(d,false,sizeof(d)) ;

memset(G,false,sizeof(G)) ;

edges.clear() ;

for(int i=0; i<n-1; i++) {

int u,v,w;

scanf("%d%d%d",&u,&v,&w) ;

add\_edge(u,v,w) ;

add\_edge(v,u,w) ;

}

tot = 0;

dfs(0,1) ;

RMQ\_init() ;

int q ;

scanf("%d",&q) ;

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

int a,b,c ;

scanf("%d%d%d",&a,&b,&c) ;

int ans = 0 ;

///和两个点的时候有点区别

ans+=len[a]+len[b]+len[c]-(len[LCA(a,b)]+len[LCA(a,c)]+len[LCA(b,c)]);

printf("%d\n",ans) ;

}

}

return 0;

}

# 动态规划

## 01背包

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

实现代码

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

for(int j=m;j>=0;j--){

if(j-w[i]>=0)dp[j] = max(dp[j],dp[j-w[i]]+v[i]) ;

}

}

观察DP数组，

对于每一行每次更新只与上一行有关

对于每一列每次更新只与前面的列有关

所以DP数组可以设定为一维的，代码如上所示

## 最长连续子序列的和

对于数组a[i]，X[i]表示包含a[i]的最大的和

X[i] = max(X[i]+a[i],a[i]) ;///在a[i]位置，只有两种选择，要么重新开始作为起点，要么就和前面连在一起

如果需要记录区间，则增加一个标记数组即可

# 其他

## 矩阵快速幂

struct matrix{

LL mat[maxn][maxn] ;

matrix operator \* (const matrix & b){ ///重载\*运算符，对于加减的运算符比较容易

matrix res ;

memset(res.mat,0,sizeof(res.mat)) ;

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

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

if(mat[i][j])

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

res.mat[i][k]+= mat[i][j]\*b.mat[j][k];

}

}

}

return res ;

}

};

///快速幂

matrix cal(matrix x,int k){

matrix res ;

memset(res.mat,0,sizeof(res.mat));

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

res.mat[i][i] = 1 ;

}

while(k){

if(k&1) res = res\*x ;

x = x\*x ;

k>>=1 ;

}

return res ;

}

## 大数（bigdecimal）

///平方根大搜索（大数和大浮点数）

/\*\*在二进制中，2的算术平方根，即sqrt(2)，

是一个无限小数1.0110101000001001111...

给定一个整数n和一个01串S，你的任务是在sqrt(n)的小数部分

（即小数点之后的部分）中找到S第一次出现的位置。

如果sqrt(n)是整数，小数部分看作是无限多个0组成的序列。

\*/

import java.math.BigDecimal;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int T = 0 ;

T = in.nextInt();

while(T-- >0){

int n = in.nextInt();

String str = in.next();

BigDecimal nn = new BigDecimal(n);

n = (int) Math.sqrt(n);

BigDecimal s = new BigDecimal(n);

BigDecimal esp = new BigDecimal("0.1");

int cnt=0;

while(cnt <50){

for(int i=1;i<10;i++){

s = s.add(esp);

if(s.multiply(s).compareTo(nn)>0){

s = s.subtract(esp);

break ;

}else if(s.multiply(s).compareTo(nn)==0){

break ;

}

}

esp = esp.multiply(new BigDecimal("0.1"));

cnt++ ;

}

String ans = "" ;

s=s.subtract(new BigDecimal(n));

for(int i=0;i<120;i++){

s=s.multiply(new BigDecimal("2"));

if(s.compareTo(new BigDecimal("1"))>=0){

ans+="1";

s = s.subtract(new BigDecimal("1"));

}else{

ans+="0";

}

}

System.out.println(ans.indexOf(str));

}

}

}

补充：

**public** **class** Main {

**public** **static** **void** main(String[] args) {

BigDecimal a = **new** BigDecimal(1.2);

BigDecimal b = **new** BigDecimal(9.5564231);

System.***out***.println(a);

System.***out***.println(b);

DecimalFormat df = **new** DecimalFormat("#.00");

// #.00 表示两位小数 #.0000四位小数 以此类推...

//df.format(b)使其满足预定的舍入方式，但是类型已经变了

System.***out***.println(df.format(b));

//注意如果再次使用需要new出来

BigDecimal temp = **new** BigDecimal(df.format(b));

System.***out***.println(temp);

///BigDecimal转BigInteger是舍掉小数部分，如果需要四舍五入，需要加0.5

BigInteger ab = temp.toBigInteger();

System.***out***.println(ab);

}

}

## C++STL

C++ Vectors

Vectors 包含着一系列连续存储的元素,其行为和数组类似。访问Vector中的任意元素或从末尾添加元素都可以在[常量级时间复杂度](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/complexity.html)内完成，而查找特定值的元素所处的位置或是在Vector中插入元素则是[线性时间复杂度](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/complexity.html)。

|  |  |
| --- | --- |
| [Constructors](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#Constructors) | 构造函数 |
| [Operators](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#Operators) | 对vector进行赋值或比较 |
| [assign()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#assign) | 对Vector中的元素赋值 |
| [at()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#at) | 返回指定位置的元素 |
| [back()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#back) | 返回最末一个元素 |
| [begin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#begin) | 返回第一个元素的迭代器 |
| [capacity()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#capacity) | 返回vector所能容纳的元素数量(在不重新分配内存的情况下） |
| [clear()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#clear) | 清空所有元素 |
| [empty()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#empty) | 判断Vector是否为空（返回true时为空） |
| [end()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#end) | 返回最末元素的迭代器(译注:实指向最末元素的下一个位置) |
| [erase()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#erase) | 删除指定元素 |
| [front()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#front) | 返回第一个元素 |
| [get\_allocator()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#get_allocator) | 返回vector的内存分配器 |
| [insert()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#insert) | 插入元素到Vector中 |
| [max\_size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#max_size) | 返回Vector所能容纳元素的最大数量（上限） |
| [pop\_back()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#pop_back) | 移除最后一个元素 |
| [push\_back()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#push_back) | 在Vector最后添加一个元素 |
| [rbegin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#rbegin) | 返回Vector尾部的逆迭代器 |
| [rend()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#rend) | 返回Vector起始的逆迭代器 |
| [reserve()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#reserve) | 设置Vector最小的元素容纳数量 |
| [resize()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#resize) | 改变Vector元素数量的大小 |
| [size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#size) | 返回Vector元素数量的大小 |
| [swap()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppvector_details.html#swap) | 交换两个Vector |

**C++ Stacks（堆栈）**

C++ Stack（堆栈） 是一个容器类的改编，为程序员提供了堆栈的全部功能，——也就是说实现了一个先进后出（FILO）的数据结构。

|  |  |
| --- | --- |
| [操作](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#Operators) | 比较和分配堆栈 |
| [empty()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#empty) | 堆栈为空则返回真 |
| [pop()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#pop) | 移除栈顶元素 |
| [push()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#push) | 在栈顶增加元素 |
| [size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#size) | 返回栈中元素数目 |
| [top()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppstack_details.html#top) | 返回栈顶元素 |

**C++ Sets**

集合(Set)是一种包含已排序对象的关联容器

|  |  |
| --- | --- |
| [begin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#begin) | 返回指向第一个元素的迭代器 |
| [clear()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#clear) | 清除所有元素 |
| [count()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#count) | 返回某个值元素的个数 |
| [empty()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#empty) | 如果集合为空，返回true |
| [end()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#end) | 返回指向最后一个元素的迭代器 |
| [equal\_range()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#equal_range) | 返回集合中与给定值相等的上下限的两个迭代器 |
| [erase()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#erase) | 删除集合中的元素 |
| [find()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#find) | 返回一个指向被查找到元素的迭代器 |
| [get\_allocator()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#get_allocator) | 返回集合的分配器 |
| [insert()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#insert) | 在集合中插入元素 |
| [lower\_bound()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#lower_bound) | 返回指向大于（或等于）某值的第一个元素的迭代器 |
| [key\_comp()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#key_comp) | 返回一个用于元素间值比较的函数 |
| [max\_size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#max_size) | 返回集合能容纳的元素的最大限值 |
| [rbegin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#rbegin) | 返回指向集合中最后一个元素的反向迭代器 |
| [rend()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#rend) | 返回指向集合中第一个元素的反向迭代器 |
| [size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#size) | 集合中元素的数目 |
| [swap()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#swap) | 交换两个集合变量 |
| [upper\_bound()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#upper_bound) | 返回大于某个值元素的迭代器 |
| [value\_comp()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppset_details.html#value_comp) | 返回一个用于比较元素间的值的函数 |

**C++ Maps**

C++ Maps是一种关联式容器，包含“关键字/值”对

|  |  |
| --- | --- |
| [begin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#begin) | 返回指向map头部的迭代器 |
| [clear()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#clear) | 删除所有元素 |
| [count()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#count) | 返回指定元素出现的次数 |
| [empty()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#empty) | 如果map为空则返回true |
| [end()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#end) | 返回指向map末尾的迭代器 |
| [equal\_range()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#equal_range) | 返回特殊条目的迭代器对 |
| [erase()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#erase) | 删除一个元素 |
| [find()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#find) | 查找一个元素 |
| [get\_allocator()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#get_allocator) | 返回map的配置器 |
| [insert()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#insert) | 插入元素 |
| [key\_comp()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#key_comp) | 返回比较元素key的函数 |
| [lower\_bound()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#lower_bound) | 返回键值>=给定元素的第一个位置 |
| [max\_size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#max_size) | 返回可以容纳的最大元素个数 |
| [rbegin()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#rbegin) | 返回一个指向map尾部的逆向迭代器 |
| [rend()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#rend) | 返回一个指向map头部的逆向迭代器 |
| [size()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#size) | 返回map中元素的个数 |
| [swap()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#swap) | 交换两个map |
| [upper\_bound()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#upper_bound) | 返回键值>给定元素的第一个位置 |
| [value\_comp()](mk:@MSITStore:F:\mycode\C++库函数参考.chm::/cppreference.com/cppmap_details.html#value_comp) | 返回比较元素value的函数 |

## 其余比较方便的STL

结构题重载运算符：

struct node{

int a,b ;

}

bool operator < (const node &p)const {

if(a==p.a)return b<p.b

return a<p.a

}

next\_permutation(array,array+n,cmp) ;

//使用前先sort(array,array+n,cmp)

do{

使用，但是不改变array

}while(next\_permutation(array,array+n,cmp);

binary\_search(array,array+n,k) ;

查找array中是否含有k，使用前也需要sort

fill(array,array+n,k)数组（1-n）全部填充为k

fill\_n(array,n,k)数组从开始位置填充n个k

nth\_element(array,array+m, array+n,cmp)

//部分排序函数，按照cmp比较规则，第m大的放在数组的m-1位置，前面都是比他小的，后面都是比他大的