A

C

M

算法模板

Code by yeates

目录

[1 数据结构 3](#_Toc483334756)

[1.1 二分 3](#_Toc483334757)

[1.2 线段树 4](#_Toc483334758)

[1.2.1 基础线段树(单点更新) 4](#_Toc483334759)

[1.2.2 优化线段树(成段更新) 5](#_Toc483334760)

[1.2.3 线段数区间合并 6](#_Toc483334761)

[1.3 并查集 9](#_Toc483334762)

[1.4 带权并查集 9](#_Toc483334763)

[2 图论 10](#_Toc483334764)

[2.1 最小生成树 10](#_Toc483334765)

[2.1.1 普利姆算法 10](#_Toc483334766)

[2.1.2 克鲁斯卡尔算法 10](#_Toc483334767)

[2.2 最短路 11](#_Toc483334768)

[2.2.1 Dijkstra 11](#_Toc483334769)

[2.2.2 Floyd: 11](#_Toc483334770)

[2.2.3 SPFA 11](#_Toc483334771)

[2.2.4 优先队列对dijkstra算法的优化 12](#_Toc483334772)

[3 组合游戏 14](#_Toc483334773)

[3.1 Nim博弈 14](#_Toc483334774)

[4 动态规划 14](#_Toc483334775)

[4.1 记忆化搜索 14](#_Toc483334776)

[5 数论 15](#_Toc483334777)

[5.1 素数筛法 15](#_Toc483334778)

[5.2 求质因子 15](#_Toc483334779)

[6 字符串 16](#_Toc483334780)

[6.1 KMP 16](#_Toc483334781)

[6.2 AC自动机 18](#_Toc483334782)

[7 容器 22](#_Toc483334783)

[7.1 map： 22](#_Toc483334784)

[7.2 vector： 23](#_Toc483334785)

[7.3 queue(先进先出)： 23](#_Toc483334786)

[7.4 stack(先进后出) 23](#_Toc483334787)

[7.5 输入输出流重定向 23](#_Toc483334788)

[8 数学公式 23](#_Toc483334789)

[8.1 求三角形面积 23](#_Toc483334790)

[8.2 点到直线的距离 24](#_Toc483334791)

[8.3 卡特兰数 24](#_Toc483334792)

[8.4 判断一个点是否在已知三点坐标的三角形内 24](#_Toc483334793)

[9 杂项 25](#_Toc483334794)

[9.1 大整数取模 25](#_Toc483334795)

[9.2 约瑟夫环 25](#_Toc483334796)

[9.3 求字典序值 25](#_Toc483334797)

# 数据结构

## 二分

最简单的二分查找

while (low <= high)

{

int mid = (low + high) / 2;

if (dp[mid] >= tmp) high = mid - 1;

else low = mid + 1;

}

dp[low] = tmp;

//几种二分查找(O(logn))，区别在于是否数组中含有多个相同的查找值

//lower\_bound返回等于item的第一个位置

template<typename T>

int mine\_lower\_bound(T \*a, T item, int n) {

int l = 0, r = n - 1;

while (l < r) {

int m = (r - l) / 2 + l;

if (a[m] >= item) r = m;

else l = m + 1;

}

if (a[l] == item) return l;

else return -1;

}

//upper\_bound返回等于item的最后一个位置的下一个位置。。

template<typename T>

int mine\_upper\_bound(T \*a, T item, int n) {

int l = 0, r = n - 1;

while (l < r) {

int m = (r - l) / 2 + l;

if (a[m] > item) r = m;

else l = m + 1;

}

if (a[l - 1] == item) return l - 1;

else return -1;

}

template<typename T>

int mine\_binary\_search(T \*a, T item, int n) {

int l = 0, r = n - 1;

while (l < r) {

int m = (r - l) / 2 + l;

if (a[m] == item) return m;

else if (a[m] > item) r = m - 1;

else l = m + 1;

}

return -1;

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(0);

int n;

while (cin >> n) {

int \_array[MAX];

REP(i, n) cin >> \_array[i];

sort(\_array, \_array + n);

REP(i, n) fcout(\_array[i], i, n);

int item;

cin >> item;

cout << "upper\_bound:" << endl;

cout << mine\_upper\_bound(\_array, item, n) << endl << endl;

cout << "lower\_bound:" << endl;

cout << mine\_lower\_bound(\_array, item, n) << endl << endl;

cout << "binary\_search:" << endl;

cout << mine\_binary\_search(\_array, item, n) << endl << endl;

}

return 0;

}

## 线段树

### 基础线段树(单点更新)

a[]是线段树；

build()是初始化线段树：在叶子结点输入值；

update()是从叶子结点开始改变线段树上的值，变化值即为data；query()的代码类似于build()，查询[l, r]段的值。

struct node {

int l, r;

int sum;

}a[4 \* 50005];

void build(int i, int l, int r) {

a[i].l = l;

a[i].r = r;

int mid = (l + r) / 2;

if (l == r) {

scanf("%d", &a[i].sum);

return;

}

build(i \* 2, l, mid);

build(i \* 2 + 1, mid + 1, r);

a[i].sum = a[i \* 2].sum + a[i \* 2 + 1].sum;

}

void updata(int i, int index, int data) {

if (a[i].l == a[i].r) {

a[i].sum += data;

return;

}

int mid = (a[i].l + a[i].r) / 2;

if (index <= mid) updata(i \* 2, index, data);

else updata(i \* 2 + 1, index, data);

a[i].sum = a[i \* 2].sum + a[i \* 2 + 1].sum;

}

int query(int i, int l, int r) {

int mid = (a[i].l + a[i].r) / 2;

int sum = 0;

if (a[i].l == l&&a[i].r == r) return a[i].sum;

if (r <= mid) return query(i \* 2, l, r);

else if (l > mid) return query(i \* 2 + 1, l, r);

else {

sum += query(i \* 2, l, mid);

sum += query(i \* 2 + 1, mid + 1, r);

return sum;

}

}

### 优化线段树(成段更新)

**暂时记忆以减少复杂度，可实现成段更新**

typedef long long LL;

struct node {

int l, r, m;

LL sum, mark;

}T[maxn << 2];

int a[maxn];

void build(int id, int l, int r) {

T[id].l = l; T[id].r = r; T[id].m = (l + r) >> 1; T[id].mark = 0;

if (l == r) { T[id].sum = a[l]; return; }

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

build(id << 1, l, m); build((id << 1) + 1, m + 1, r);

T[id].sum = (T[id << 1].sum + T[(id << 1) + 1].sum);

}

void update(int id, int l, int r, int val) {

if (T[id].l == l&&T[id].r == r) {

T[id].mark += val; return;

}

T[id].sum += (LL)val\*(r - l + 1);

if (T[id].m >= r)

update(id << 1, l, r, val);

else if (T[id].m<l)

update((id << 1) + 1, l, r, val);

else {

update(id << 1, l, T[id].m, val);

update((id << 1) + 1, T[id].m + 1, r, val);

}

}

LL query(int id, int l, int r) {

if (T[id].l == l&&T[id].r == r) return T[id].sum + T[id].mark\*(LL)(r - l + 1);

if (T[id].mark != 0) {

T[id << 1].mark += T[id].mark;

T[(id << 1) + 1].mark += T[id].mark;

T[id].sum += (LL)(T[id].r - T[id].l + 1)\*T[id].mark; T[id].mark = 0;

}

if (T[id].m >= r) {

return query(id << 1, l, r);

}

else if (T[id].m<l) {

return query((id << 1) + 1, l, r);

}

else {

return query(id << 1, l, T[id].m) + query((id << 1) + 1, T[id].m + 1, r);

}

}

### 线段数区间合并

例子：给一个由0,1组成的序列，有两种操作，一种是翻转给定区间的数(0->1,1->0)，另一种是查询给定区间内由1组成的子串的最大长度。重点在**区间合并**和**延迟标记**。

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

#define lson l,m,rt<<1

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

const int maxn = 111111;

int sum[maxn << 2];

int lb[maxn << 2], rb[maxn << 2];

int col[maxn << 2];

int wm[maxn << 2];

int wl[maxn << 2];

int wr[maxn << 2];

int max(int a, int b) {

return a>b ? a : b;

}

int Min(int a, int b) {

return a<b ? a : b;

}

void pushup(int m, int rt) {

wl[rt] = wl[rt << 1];

lb[rt] = lb[rt << 1];

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

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

if (wl[rt] == m - (m >> 1)) wl[rt] += wl[rt << 1 | 1];

if (lb[rt] == m - (m >> 1)) lb[rt] += lb[rt << 1 | 1];

if (wr[rt] == (m >> 1)) wr[rt] += wr[rt << 1];

if (rb[rt] == (m >> 1)) rb[rt] += rb[rt << 1];

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

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

wm[rt] = max(wm[rt], wr[rt << 1] + wl[rt << 1 | 1]);

sum[rt] = max(sum[rt], rb[rt << 1] + lb[rt << 1 | 1]);

}

void make(int rt) {

swap(sum[rt], wm[rt]);

swap(wl[rt], lb[rt]);

swap(wr[rt], rb[rt]);

}

void pushdown(int rt) {

if (col[rt]) {

col[rt << 1] ^= 1;

col[rt << 1 | 1] ^= 1;

col[rt] = 0;

make(rt << 1);

make(rt << 1 | 1);

}

}

int num[100001];

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

col[rt] = lb[rt] = rb[rt] = wl[rt] = wr[rt] = sum[rt] = wm[rt] = 0;

if (l == r) {

if (num[l] == 1) sum[rt] = lb[rt] = rb[rt] = 1;

else {

wm[rt] = wl[rt] = wr[rt] = 1;

}return;

}

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

build(lson);

build(rson);

pushup(r - l + 1, rt);

}

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

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

col[rt] ^= 1;

make(rt);

return;

}

pushdown(rt);

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

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

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

pushup(r - l + 1, rt);

}

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

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

return sum[rt];

}

pushdown(rt);

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

if (R <= m)

return query(L, R, lson);

if (L > m)

return query(L, R, rson);

int t1 = query(L, R, lson);

int t2 = query(L, R, rson);

int a = Min(m - L + 1, rb[rt << 1]);

int b = Min(R - m, lb[rt << 1 | 1]);

return max(max(t1, t2), a + b);

}

int main() {

int n, m, i, j, k, a, b, op;

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

for (i = 1; i <= n; i++) scanf("%d", &num[i]);

build(1, n, 1);

scanf("%d", &m);

while (m--) {

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

if (op == 1)

update(a, b, 1, n, 1);

else {

int ans = query(a, b, 1, n, 1);

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

}

}

}

return 0;

}

## 并查集

**基础算法**

void init**(){**

**for(**int i**=**1**;**i**<**MAX**;**i**++){**

bin**[**i**]=**i**;**

crank**[**i**]=**1**;**

**}**

**}**

int findr**(**int x**){**

**if(**x**!=**bin**[**x**]){**

bin**[**x**]=**findr**(**bin**[**x**]);** //路径压缩

**}**

**return** bin**[**x**];**

**}**

void mergebin**(**int x**,**int y**){**

int rx**,**ry**;**

rx**=**findr**(**x**);**

ry**=**findr**(**y**);**

**if(**rx**!=**ry**){**

bin**[**rx**]=**ry**;**

crank**[**ry**]+=**crank**[**rx**];**

ans**=**max**(**ans**,**crank**[**ry**]);**

**}**

**}**

## 带权并查集

# 图论

## 最小生成树

### 普利姆算法

void prim(int v, int n) {

int i, j, k, tmin;

int ans = 0;

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

dis[i] = mpt[v][i];

vis[i] = true;

}

vis[v] = false;

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

tmin = INF;

k = 0;

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

if (vis[i] && dis[i] < tmin) {

tmin = dis[i];

k = i;

}

}

if (tmin == INF)

break;

vis[k] = false;

ans += tmin;

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

if (vis[i] && dis[i] > mpt[k][i]) {

dis[i] = mpt[k][i];

}

}

}

cout << ans << endl;

}

### 克鲁斯卡尔算法

## 最短路

要注意

1. 输入坑点，比如双向路径、相同路多次输入等。这种坑有的是读题不仔细，有的是题上既没有说而且不能用常识判断。所以要注意输入控制。
2. 初始化

### Dijkstra

int mpt[MAX][MAX];

int vis[MAX], dis[MAX];

int n;

int dijkstra(int s, int d) {

init(); //初始化mpt二维数组、vis数组和dis数组

dis[s] = 0;

dis[0] = 0; //这些初值要设置好

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

int t = 0;

for (int i = 1; i <= n; i++) { //从1开始计数的dis数组

if (vis[i]) continue;

t = dis[t] > dis[i] ? i : t;

}

vis[t] = 1;

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

if (vis[i] || mpt[t][i] == INF) continue;

dis[i] = Min(dis[i], dis[t] + mpt[t][i]);

}

}

return dis[d];

}

### Floyd:

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

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

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

dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);

### SPFA

int mpt[MAX][MAX];

int vis[MAX], dis[MAX];

int n;

int spfa(int s, int d) {

init();

queue<int>q;

q.push(s);

vis[s] = 1; // 这里的vis数组是用来判断当前下标是否在队列中。

dis[s] = 0;

while (!q.empty()) {

int e = q.front();

q.pop();

vis[e] = 0;

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

if (mpt[e][i] + dis[e] >= dis[i]) continue;

dis[i] = mpt[e][i] + dis[e];

if (vis[i]) continue;

q.push(i);

vis[i] = 1;

}

}

return dis[d];

}

### 优先队列对dijkstra算法的优化

/\*

priority\_queue实现的dijkstra最短路算法，时间复杂度O(mlogn)

相比原始dijkstra的O(n^2)复杂度，该算法不仅在稀疏图的情况下大大提高效率，且在稠密图下也常常不会慢于原始算法。

\*/

#include <iostream>

#include <queue>

#include <string.h>

using namespace std;

#define Max a > b ? a: b

#define Min a < b ? a: b

#define MAX 1005

#define INF 0x3f3f3f3f

struct Edge {

int from, to, dist; //起点，终点，边权

Edge(int f, int t, int d) :from(f), to(t), dist(d) {};

};

int n, m; //n是节点数，m是边数

vector<Edge>edges; //边数组

vector<int>G[MAX]; //邻接图，存储边的序号，不过注意第二维下标不再表示边终点!

int dis[MAX]; //图中每个点距离单源起点s的最短距离

//添加边到edges中，并且用G保存边的编号

void AddEdge(int s, int d, int w) {

edges.push\_back(Edge(s, d, w));

int no = edges.size() - 1;

G[s].push\_back(no);

}

void init() {

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

dis[i] = INF;

G[i].clear();

}

edges.clear();

}

struct Node {

int d, u;

bool operator < (const Node& tp) const { //自定义优先级策略，在priority\_queue会使用

return d > tp.d; //d即dis[u]

}

};

//core code

int dijkstra(int s, int d) {

int vis[MAX];

priority\_queue<Node>Q;

//预处理部分

memset(vis, 0, sizeof(vis));

dis[s] = 0;

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

while (!Q.empty()) {

Node tp = Q.top();

Q.pop();

int u = tp.u; //简化

if (vis[u]) continue;

vis[u] = 1;

//松弛操作,和原版不一样了

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

Edge& e = edges[G[u][i]]; //简化

if (vis[e.to] == 0 && dis[e.to] > dis[u] + e.dist) {

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

Q.push((Node) { dis[e.to], e.to });

}

}

}

return dis[d];

}

int main() {

ios::sync\_with\_stdio(0);

while (cin >> m >> n) {

init();

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

int s, d, w;

cin >> s >> d >> w;

AddEdge(s, d, w);

AddEdge(d, s, w);

}

cout << dijkstra(1, n) << endl;

}

return 0;

}

# 组合游戏

## Nim博弈

**取^（异或），结果为0则先手输。**

上次的代码：

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

**{**

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

key **=** key **^** a**[**i**];**

**}**

**if(**key **==** 0**)**

**{**

printf**(**"No\n"**);**

**continue;**

**}**

**else**

**{**

printf**(**"Yes\n"**);**

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

**{**

int temp **=** key**^**a**[**i**];**

**if(**temp**<**a**[**i**])** printf**(**"%d %d\n"**,**a**[**i**],**temp**);**

**}**

**}**

# 动态规划

## 记忆化搜索

/\*

题目：POJ - 1088 滑雪

\*/

int a[105][105], dp[105][105], n, m;

int ca[4][2] = { 1,0,0,-1,-1,0,0,1 };

int DFS(int x, int y) {

if (dp[x][y] != 0) return dp[x][y];

int i, j, k, tmp, ans = 1;

for (k = 0; k<4; k++) {

i = x + ca[k][0];

j = y + ca[k][1];

if (i >= n || j >= m || i<0 || j<0) continue;

if (a[i][j] >= a[x][y]) continue;

tmp = DFS(i, j) + 1;

ans = max(tmp, ans);

}

dp[x][y] = ans;

return ans;

}

int main() {

while (cin >> n >> m) {

int i, j, ans = 0;

memset(dp, 0, sizeof(dp));

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

for (j = 0; j<m; j++)

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

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

for (j = 0; j<m; j++) {

ans = max(ans, DFS(i, j));

}

cout << ans << endl;

}

return 0;

}

# 数论

## 素数筛法

//素数筛法优化版本

memset(prime, 0, sizeof(prime));

prime[0] = prime[1] = 1;

for (int i = 2; i < MAX; i++) {

if (prime[i] == 1) continue;

if (i > MAX / i) continue; //点睛之笔

for (int j = i \* i; j < MAX; j += i)

prime[j] = 1;

}

## 求质因子

int main**(){**

LL num**;**

memset**(**prime**,** 0**,** **sizeof(**prime**));**

prime**[**0**]** **=** prime**[**1**]** **=** 1**;**

**//筛法**

**for(**LL i **=** 2**;** i **<** MAX**;** i**++){**

**if(**prime**[**i**]** **==** 1**)** **continue;**

**if(**i **\*** i **>=** MAX**)** **continue;**

**for(**LL j **=** i **\*** i**;** j **<** MAX**;** j **+=** i**){**

prime**[**j**]** **=** 1**;**

**}**

**}**

**//输入一个数num，求其质因子(no是输出格式开关变量)**

**while(~**scanf**(**"%lld"**,** **&**num**)){**

int no **=** 0**;**

**for(**LL j **=** 2**;** j **<=** num**;** j**++){**

**if(**prime**[**j**]** **==** 1**)** **continue;**

**if(**num **%** j **==** 0**){**

**if(**no **==** 0**){**

printf**(**"%lld"**,** j**);**

no **=** 1**;**

**}**

**else** printf**(**"\*%lld"**,** j**);**

num **/=** j**;**

j **=** 1**;**

**}**

**}**

cout**<<**endl**;**

**}**

**return** 0**;**

**}**

# 字符串

## KMP

s是正文串，t是模式串

void makeNext(int m) {

int i = 0, j = -1;

Next[0] = -1;

while (i < m) {

if (j == -1 || t[i] == t[j]) {

i++, j++;

Next[i] = j;

}

else j = Next[j];

}

}

int kmp(int n, int m) {

int i = 0, j = -1;

makeNext(m);

while (i < n && j < m) {

if (j == -1 || t[j] == s[i]) {

i++, j++;

}

else j = Next[j];

}

if (i >= n && j < m) return -1;

else return i - m;

}

**next数组的使用(字符串str，当有n个相同子串x整合到一起满足为str字符串时，求x串的长度)：**

void makeNext(int n) {

int i = 0, j = -1;

Next[0] = -1;

while (i < n) {

if (j == -1 || s[i] == s[j]) {

Next[++i] = ++j;

if (j == 1) {

ans = i - 1;

}

}

else j = Next[j];

}

}

int main() {

while (cin >> s && s[0] != '.') {

int n = strlen(s);

ans = 1;

makeNext(n);

if (Next[n] >= ans && Next[n] % ans == 0) cout << n / ans << endl;

else cout << "1" << endl;

}

return 0;

}

## AC自动机

判断多个字符串在一段长文本中的存在情况。

举例：给出n个字符串，在这n个中有多少个存在于str[]中？

struct node {

int cnt;

struct node \*next[CAP];

struct node \*fail;

void init() {

cnt = 0;

for (int i = 0; i < CAP; i++) next[i] = NULL;

fail = NULL;

}

};

void build(char str[MAX], struct node \*&root) {

struct node \*tmp = root;

int j = 0;

int m = strlen(str);

while (j < m) {

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

if (tmp->next[i] == NULL) {

tmp->next[i] = new struct node;

tmp->next[i]->init(); //important

}

tmp = tmp->next[i];

j++;

}

tmp->cnt++;

}

void setFail(struct node \*&root) {

root->fail = NULL;

queue<struct node \*>que;

struct node \*son, \*p, \*tmp;

que.push(root);

p = root;

while (!que.empty()) {

tmp = que.front();

que.pop();

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

if (tmp->next[i] == NULL) continue;

son = tmp->next[i];

if (tmp == root) son->fail = root;

else {

p = tmp->fail;

while (p) {

if (p->next[i]) {

son->fail = p->next[i];

break;

}

else p = p->fail;

}

if (p == NULL) son->fail = root;

}

que.push(son);

}

}

}

void query(char str[MAX], struct node \*root) {

struct node \*p, \*tmp;

int ans = 0;

p = root;

int len = strlen(str);

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

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

while (p->next[pos] == NULL && p != root) p = p->fail;

p = p->next[pos];

if (p == NULL) p = root;

tmp = p;

while (tmp != NULL)

{

if (tmp->cnt >= 0) {

ans += tmp->cnt;

tmp->cnt = -1;

}

else break;

tmp = tmp->fail;

}

}

cout << ans << endl;

}

int main() {

int T;

cin >> T;

while (T--) {

int n;

struct node \*root = new struct node;

root->init();

scanf("%d", &n);

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

char str[MAX];

scanf("%s", str);

build(str, root);

}

setFail(root);

char str[MAX];

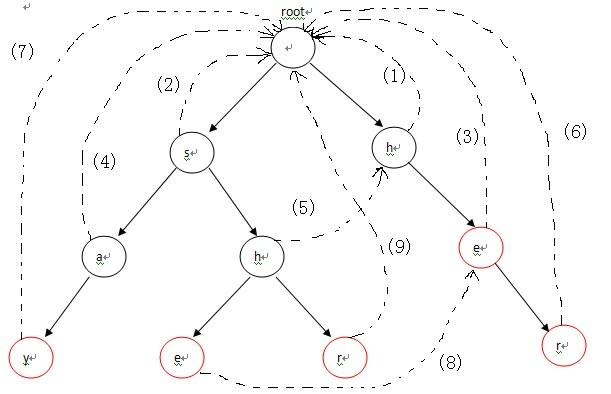
scanf("%s", str);

query(str, root);

}

return 0;

}



（失败指针指法如图，理解:当模式匹配过程中**失配时**，指针则指向**失败指针**指向的内存）

AC自动机变形：（不仅要知道哪些字符串存在，还要分别其求出在长文本中的个数）

struct node {

int no;

int cnt;

struct node \*next[CAP];

struct node \*fail;

void init() {

for (int i = 0; i < CAP; i++) next[i] = NULL;

fail = NULL;

no = -1;

cnt = 0;

}

};

char s[1000][55];

void build(char str[MAX], struct node \*&root, int no) {

struct node \*tmp = root;

int j = 0;

int m = strlen(str);

while (j < m) {

int i = str[j] - 'A';

if (tmp->next[i] == NULL) {

tmp->next[i] = new struct node;

tmp->next[i]->init(); //important

}

tmp = tmp->next[i];

j++;

}

tmp->no = no;

tmp->cnt++;

}

void setFail(struct node \*&root) {

root->fail = NULL;

queue<struct node \*>que;

struct node \*son, \*p, \*tmp;

que.push(root);

p = root;

while (!que.empty()) {

tmp = que.front();

que.pop();

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

if (tmp->next[i] == NULL) continue;

son = tmp->next[i];

if (tmp == root) son->fail = root;

else {

p = tmp->fail;

while (p) {

if (p->next[i]) {

son->fail = p->next[i];

break;

}

else p = p->fail;

}

if (p == NULL) son->fail = root;

}

que.push(son);

}

}

}

void query(char str[MAX], struct node \*root) {

struct node \*p, \*tmp;

int vis[1001];

int ans[1001];

memset(vis, 0, sizeof(vis));

memset(ans, 0, sizeof(ans));

p = root;

int len = strlen(str);

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

int pos = str[i] - 'A';

if (pos < 0 || pos > 25) { p = root; continue; }

while (p->next[pos] == NULL && p != root) p = p->fail;

p = p->next[pos];

if (p == NULL) p = root;

tmp = p;

while (tmp != NULL)

{

vis[tmp->no] = 1;

ans[tmp->no] += tmp->cnt;

tmp = tmp->fail;

}

}

for (int i = 0; i < 1000; i++) {

if (vis[i]) {

printf("%s: %d\n", s[i], ans[i]);

}

}

}

int main() {

int n;

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

struct node \*root = new struct node;

root->init();

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

scanf("%s", s[i]);

build(s[i], root, i);

}

setFail(root);

char str[MAX];

scanf("%s", str);

query(str, root);

}

return 0;

}

# C++ STL

## map：

声明语法（例子），map<string, int>mpt;

然后就可以把mpt当成一个一维数组来使用，只不过数组的下标变成了string类型的值。

## vector：

添加，push\_back()

删除vec的第i+1个元素：vec.erase(vec.begin()+i);

删除vec的[i,j-1]区间的所有元素：vec.erase(vec.begin()+i,vec.begin()+j);

其余和数组差不多

## queue(先进先出)：

添加，push()

删除，pop()

## stack(先进后出)

添加，push()

删除，pop()

## 重载运算符

/\*重载运算符让自定义变量使用sort函数\*/

template <typename T>

struct NODE {

T x, y;

NODE(T a = 0, T b = 0) : x(a), y(b) {}; //构造函数

};

template<typename T>

//运算符<的返回值是布尔变量

bool operator < (const NODE<T>& A, const NODE<T>& B) {

if (A.x > B.x) return false;

else return true;

}

template<typename T>

ostream& operator << (ostream& out, const NODE<T>& A) {

out << "x=" << A.x << endl;

out << "y=" << A.y << endl;

return out;

}

int main() {

int n;

while (cin >> n) {

NODE<double> tp[MAX];

REP(i, n) cin >> tp[i].x >> tp[i].y;

sort(tp, tp + n); //若是自定义变量需要重载<运算符才能使用sort函数！

REP(i, n) cout << tp[i];

}

return 0;

}

## 模版函数

//模版函数的使用

template <typename T>

struct NODE {

T x, y;

NODE(T a, T b) : x(a), y(b) {}; //构造函数

};

/\*

重载+运算符

注意每个使用到模版函数的都要修改保留字格式:后面加上<typename>

\*/

template <typename T> //每段使用模版函数的函数前都应该加上模版函数的声明

NODE<T> operator + (const NODE<T>& A, const NODE<T>& B) {

return NODE<T>(A.x + B.x, A.y + B.y);

}

//重载<<运算符,ostream是<<运算符的数据类型

template <typename T>

ostream& operator << (ostream& out, const NODE<T>& A) {

out << "x=" << A.x << endl;

out << "y=" << A.y << endl;

return out;

}

int main() {

double \_1, \_2, \_3, \_4; //下划线\_不能作为变量的后缀

while (cin >> \_1 >> \_2 >> \_3 >> \_4) {

NODE<double> m(\_1, \_2), n(\_3, \_4);

cout << m << endl;

cout << n << endl;

cout << m + n << endl;

}

return 0;

}

# 数学公式

## 求三角形面积

http://img.blog.csdn.net/20160713164451694?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

* 1. 海伦公式描述：公式中a，b，c分别为三角形三边长，p为半周长，S为三角形的面积。
  2. 行列式求三角形面积：

S=（1/2)\*(下面行列式）= (1/2)\*(x1y2+x2y3+x3y1-x1y3-x2y1-x3y2)

|x1 y1 1|

|x2 y2 1|

|x3 y3 1|

## 点到直线的距离

方法一：距离公式直接求

http://img.blog.csdn.net/20160713164514773?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

公式描述：公式中的直线方程为Ax+By+C=0，点P的坐标为(x0,y0)。但是直线方程不是够直接。推荐使用方法二。

方法二：先用海伦公式求面积然后求三角形高

## 卡特兰数

令h(1)＝1,h(0)=1，catalan 数满足递归式：  h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)h(0) (其中n>=2)

另类递归式：

该递推关系的解为：h(n)=C(2n,n)/(n+1) (n=1,2,3,...)

## 判断一个点是否在已知三点坐标的三角形内

//判断一个点是否在已知三点坐标的三角形内（向量法）

struct node {

double x, y;

};

double dot(node a, node b) {

return a.x \* b.x + a.y \* b.y;

}

bool JudgePointInTriangle(node v0, node v1, node v2) {

double fenzi = dot(v0, v0) \* dot(v1, v1) - dot(v0, v1) \* dot(v1, v0);

double u = dot(v1, v1) \* dot(v2, v0) - dot(v1, v0) \* dot(v2, v1);

u /= fenzi;

if (u < 0 || u > 1) return false;

double v = dot(v0, v0) \* dot(v2, v1) - dot(v0, v1) \* dot(v2, v0);

v /= fenzi;

if (v < 0 || v > 1) return false;

return u + v <= 1;

}

# 杂项

## 大整数取模

//求a^n的前位数

LL POW(LL a, LL n, LL MOD) {

LL ans = 1, tmp = a;

while (n != 0) {

if (n % 2 == 1) {

ans = (ans \* tmp) % MOD;

}

tmp = (tmp \* tmp) % MOD;

n /= 2;

}

return ans;

}

## 约瑟夫环

int jsf(int n) {

if (n == 3) return 3;

if (n == 2) return 1;

if (n == 1) return 1;

bool isOdd;

if (n % 2 == 1) isOdd = true;

else isOdd = false;

int ans;

if (isOdd) {

ans = jsf((n - 1) / 2);

return ans \* 2 + 1;

}

else {

ans = jsf(n / 2);

return ans \* 2 - 1;

}

}

## 求字典序值

// 求字典序值ans

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

**{**

ans **+=** beishu**(**a**,** i**,** n**)** **\*** jiecheng**(**n**-** i **-** 1**);**

**}**

int jiecheng**(**int x**)**

**{**

int ans **=** 1**;**

**for(**int i **=** 2**;** i **<=** x**;** i**++)**

**{**

ans **\*=** i**;**

**}**

**return** ans**;**

**}**

int beishu**(**int a**[],** int x**,** int n**)**

**{**

int ans **=** 0**;**

**for(**int i **=** x **+** 1**;** i **<** n**;** i**++)**

**if(**a**[**i**]** **<** a**[**x**])**

ans**++;**

**return** ans**;**

**}**

## 输入输出流重定向

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

freopen("output.txt", "w", stdout);