ACM算法模板・一些常用的算法模板-模板合集（打比赛专用）

0.头文件

```
#define _CRT_SBCURE_NO_DEPRECATE
#include <set>
#include <cmath>
#include <queue>
#include <stack>
#include <vector>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <functional>
using namespace std;

const int maxn = 110;
const int INF = 0x3f3f3f3f;
```

经典

1.埃拉托斯特尼筛法

```
/*
    |埃式筛法|
    |快速筛选素数|
    |16/11/05ztx|
*/
int prime[maxn];
bool is_prime[maxn];

int sieve(int n){
    int p = 0;
    for(int i = 0; i <= n; ++i)
        is_prime[i] = true;
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i <= n; ++i){   //  注意数组大小是n
        if(is_prime[i]){
            prime[p++] = i;
            for(int j = i + i; j <= n; j += i)  //  轻剪枝，j必定是i的倍数
                is_prime[j] = false;
        }
    }
    return p;   //  返回素数个数
```

1

2.快速幂

```
/*
    |快速幂|
    |16/11/05ztx|
*/
typedef long long LL;   //  视数据大小的情况而定

LL powerMod(LL x, LL n, LL m)
{
    LL res = 1;
    while (n > 0){
        if  (n & 1) //  判断是否为奇数，若是则true
            res = (res * x) % m;
        x = (x * x) % m;
        n >>= 1;    //  相当于n /= 2;
    }
    return res;
}
```

-

3.大数模拟

大数加法

```
/*
    |大数模拟加法|
    |用string模拟|
    |16/11/05ztx, thanks to caojiji|
*/
string add1(string s1, string s2)
{
    if (s1 == "" && s2 == "")   return "0";
    if (s1 == "")   return s2;
    if (s2 == "")   return s1;
    string maxx = s1, minn = s2;
    if (s1.length() < s2.length()){
        maxx = s2;
        minn = s1;
    }
    int a = maxx.length() - 1, b = minn.length() - 1;
    for (int i = b; i >= 0; --i){
        maxx[a--] += minn[i] - '0'; //  a一直在减 ， 额外还要减个'0'
    }
    for (int i = maxx.length()-1; i > 0;--i){
        if (maxx[i] > '9'){
            maxx[i] -= 10;//注意这个是减10
            maxx[i - 1]++;
        }
    }
    if (maxx[0] > '9'){
        maxx[0] -= 10;
        maxx = '1' + maxx;
    }
    return maxx;
}
```

大数阶乘

```
/*
    |大数模拟阶乘|
    |用数组模拟|
    |16/12/02ztx|
*/
#include <iostream>
#include <cstdio>

using namespace std;

typedef long long LL;

const int maxn = 100010;

int num[maxn], len;

/*
    在mult函数中，形参部分：len每次调用函数都会发生改变，n表示每次要乘以的数，最终返回的是结果的长度
    tip: 阶乘都是先求之前的(n-1)!来求n!
    初始化Init函数很重要，不要落下
*/

void Init() {
    len = 1;
    num[0] = 1;
}

int mult(int num[], int len, int n) {
    LL tmp = 0;
    for(LL i = 0; i < len; ++i) {
         tmp = tmp + num[i] * n;    //从最低位开始，等号左边的tmp表示当前位，右边的tmp表示进位（之前进的位）
```

```
        num[i] = tmp % 10; //  保存在对应的数组位置，即去掉进位后的一位数
        tmp = tmp / 10;    //  取整用于再次循环,与n和下一个位置的乘积相加
    }
    while(tmp) {    //  之后的进位处理
        num[len++] = tmp % 10;
        tmp = tmp / 10;
    }
    return len;
}

int main() {
    Init();
    int n;
    n = 1977; // 求的阶乘数
    for(int i = 2; i <= n; ++i) {
        len = mult(num, len, i);
    }
    for(int i = len - 1; i >= 0; --i)
        printf("%d",num[i]);    //  从最高位依次输出,数据比较多采用printf输
出
    printf("\n");
    return 0;
}
```

4.GCD

```
/*
    |辗转相除法|
    |欧几里得算法|
    |求最大公约数|
    |16/11/05ztx|
*/
int gcd(int big, int small)
{
    if (small > big) swap(big, small);
    int temp;
    while (small != 0){ //  辗转相除法
        if (small > big) swap(big, small);
        temp = big % small;
        big = small;
        small = temp;
    }
    return(big);
}
```

5.LCM

```
/*
    |辗转相除法|
    |欧几里得算法|
    |求最小公倍数|
    |16/11/05ztx|
*/
int gcd(int big, int small)
{
    if (small > big) swap(big, small);
    int temp;
    while (small != 0){ //  辗转相除法
        if (small > big) swap(big, small);
        temp = big % small;
        big = small;
        small = temp;
    }
    return(big);
}
```

6.全排列

```
/*
    |求1到n的全排列, 有条件|
    |16/11/05ztx, thanks to wangqiqi|
*/
void Pern(int list[], int k, int n) {   //  k表示前k个数不动仅移动后面n-k位
数
    if (k == n - 1) {
        for (int i = 0; i < n; i++) {
            printf("%d", list[i]);
        }
        printf("\n");
    }else {
        for (int i = k; i < n; i++) {   //  输出的是满足移动条件所有全排列
            swap(list[k], list[i]);
            Pern(list, k + 1, n);
            swap(list[k], list[i]);
        }
    }
}
```

7.二分搜索

```
/*
    |二分搜索|
    |要求：先排序|
    |16/11/05ztx, thanks to wangxiaocai|
*/
```

```
//  left为最开始元素, right是末尾元素的下一个数, x是要找的数
int bsearch(int *A， int left， int right， int x){
    int m;
    while (left < right){
        m = left + (right - left) / 2;
        if (A[m] >= x)  right = m;   else left = m + 1;
        // 如果要替换为 upper_bound， 改为:if (A[m] <= v) x = m+1; else y = m;
    }
    return left;
}

/*
    最后left == right

如果没有找到135577找6, 返回7

如果找有多少的x, 可以用lower_bound查找一遍, upper_bound查找一遍, 下标相减

C++自带的lower_bound(a,a+n,x)返回数组中最后一个x的下一个数的地址

upper_bound(a,a+n,x)返回数组中第一个x的地址

如果a+n内没有找到x或x的下一个地址, 返回a+n的地址

lower_bound(a,a+n,x） -upper_bound(a,a+n,x)返回数组中x的个数
*/
```

数据结构

并查集

8.并查集

```
/*
    |合并节点操作|
    |16/11/05ztx, thanks to chaixiaojun|
*/
int father[maxn];   //  储存i的father父节点

void makeSet() {
    for (int i = 0; i < maxn; i++)
        father[i] = i;
}

int findRoot(int x) {   //  迭代找根节点
```

```
    int root = x; // 根节点
    while (root != father[root]) { // 寻找根节点
        root = father[root];
    }
    while (x != root) {
        int tmp = father[x];
        father[x] = root; // 根节点赋值
        x = tmp;
    }
    return root;
}

void Union(int x, int y) {  //  将x所在的集合和y所在的集合整合起来形成一个集合。
    int a, b;
    a = findRoot(x);
    b = findRoot(y);
    father[a] = b;  // y连在x的根节点上   或father[b] = a为x连在y的根节点上；
}

/*
    在findRoot(x)中：
    路径压缩 迭代 最优版
    关键在于在路径上的每个节点都可以直接连接到根上
*/
```

图论

MST

最小生成树

Kruskal

9.克鲁斯卡尔算法

```
/*
    |Kruskal算法|
    |适用于 稀疏图 求最小生成树|
    |16/11/05ztx thanks to wangqiqi|
*/
/*
    第一步：点、边、加入vector，把所有边按从小到大排序
    第二步：并查集部分 + 下面的code
*/
```

```
void Kruskal() {
    ans = 0;
    for (int i = 0; i<len; i++) {
        if (Find(edge[i].a) != Find(edge[i].b)) {
            Union(edge[i].a, edge[i].b);
            ans += edge[i].len;
        }
    }
}
```

Prim

10.普里姆算法

```
/*
    |Prim算法|
    |适用于 稠密图 求最小生成树|
    |堆优化版，时间复杂度：O(elgn)|
    |16/11/05ztx, thanks to chaixiaojun|
*/
struct node {
    int v, len;
    node(int v = 0, int len = 0) :v(v), len(len) {}
    bool operator < (const node &a)const {  // 加入队列的元素自动按距离从小
到大排序
        return len> a.len;
    }
};

vector<node> G[maxn];
int vis[maxn];
int dis[maxn];

void init() {
    for (int i = 0; i<maxn; i++) {
        G[i].clear();
        dis[i] = INF;
        vis[i] = false;
    }
}
int Prim(int s) {
    priority_queue<node>Q; // 定义优先队列
    int ans = 0;
    Q.push(node(s,0));  // 起点加入队列
    while (!Q.empty()) {
        node now = Q.top(); Q.pop();  // 取出距离最小的点
        int v = now.v;
        if (vis[v]) continue;  // 同一个节点，可能会推入2次或2次以上队列，这样第一个被标记后，剩下的需要直接跳过。
        vis[v] = true;  // 标记一下
        ans += now.len;
        for (int i = 0; i<G[v].size(); i++) {  // 开始更新
            int v2 = G[v][i].v;
            int len = G[v][i].len;
            if (!vis[v2] && dis[v2] > len) {
                dis[v2] = len;
                Q.push(node(v2, dis[v2]));  // 更新的点加入队列并排序
            }
        }
    }
    return ans;
}
```

Bellman-Ford

单源最短路

Dijkstra

11.迪杰斯特拉算法

```
/*
    |Dijkstra算法|
    |适用于边权为正的有向图或者无向图|
    |求从单个源点出发，到所有节点的最短路|
    |优化版：时间复杂度 O(elbn)|
    |16/11/05ztx, thanks to chaixiaojun|
*/
struct node {
    int v, len;
    node(int v = 0, int len = 0) :v(v), len(len) {}
    bool operator < (const node &a)const {  //  距离从小到大排序
        return len > a.len;
    }
};

vector<node>G[maxn];
bool vis[maxn];
int dis[maxn];

void init() {
    for (int i = 0; i<maxn; i++) {
        G[i].clear();
        vis[i] = false;
        dis[i] = INF;
    }
}
int dijkstra(int s, int e) {
    priority_queue<node>Q;
    Q.push(node(s, 0)); //  加入队列并排序
    dis[s] = 0;
    while (!Q.empty()) {
        node now = Q.top();      //  取出当前最小的
        Q.pop();
        int v = now.v;
        if (vis[v]) continue;   //  如果标记过了，直接continue
        vis[v] = true;
        for (int i = 0; i<G[v].size(); i++) {   //  更新
            int v2 = G[v][i].v;
            int len = G[v][i].len;
            if (!vis[v2] && dis[v2] > dis[v] + len) {
                dis[v2] = dis[v] + len;
                Q.push(node(v2, dis[v2]));
            }
        }
    }
    return dis[e];
}
```

SPFA

12.最短路径快速算法（Shortest Path Faster Algorithm）

```
/*
    |SPFA算法|
    |队列优化|
    |可处理负环|
*/
vector<node> G[maxn];
bool inqueue[maxn];
int dist[maxn];

void Init()
{
    for(int i = 0 ; i < maxn ; ++i){
        G[i].clear();
        dist[i] = INF;
    }
}
int SPFA(int s,int e)
{
    int v1,v2,weight;
    queue<int> Q;
    memset(inqueue,false,sizeof(inqueue)); // 标记是否在队列中
    memset(cnt,0,sizeof(cnt)); // 加入队列的次数
    dist[s] = 0;
    Q.push(s); // 起点加入队列
    inqueue[s] = true; // 标记
    while(!Q.empty()){
        v1 = Q.front();
        Q.pop();
        inqueue[v1] = false; // 取消标记
        for(int i = 0 ; i < G[v1].size() ; ++i){ // 搜索v1的链表
            v2 = G[v1][i].vex;
            weight = G[v1][i].weight;
            if(dist[v2] > dist[v1] + weight){ // 松弛操作
                dist[v2] = dist[v1] + weight;
                if(inqueue[v2] == false){  // 再次加入队列
                    inqueue[v2] = true;
                    //cnt[v2]++;  // 判负环
                    //if(cnt[v2] > n) return -1;
                    Q.push(v2);
                } } }
    }
    return dist[e];
}

/*
    不断的将s的邻接点加入队列，取出不断的进行松弛操作，直到队列为空

如果一个结点被加入队列超过n-1次，那么显然图中有负环
*/
```

Floyd-Warshall

13.弗洛伊德算法

```
/*
    |Floyd算法|
    |任意点对最短路算法|
    |求图中任意两点的最短距离的算法|
*/
for (int i = 0; i < n; i++) {   //  初始化为0
    for (int j = 0; j < n; j++)
        scanf("%lf", &dis[i][j]);
}
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
        }
    }
}
```

二分图

14.染色法

```
/*
    |交叉染色法判断二分图|
    |16/11/05ztx|
*/
int bipartite(int s) {
    int u, v;
    queue<int>Q;
    color[s] = 1;
    Q.push(s);
    while (!Q.empty()) {
        u = Q.front();
        Q.pop();
        for (int i = 0; i < G[u].size(); i++) {
            v = G[u][i];
            if (color[v] == 0) {
                color[v] = -color[u];
                Q.push(v);
            }
            else if (color[v] == color[u])
                return 0;
        }
    }
    return 1;
}
```

15..匈牙利算法

```
/*
    |求解最大匹配问题|
    |递归实现|
    |16/11/05ztx|
*/
vector<int>G[maxn];
bool inpath[maxn];  //  标记
int match[maxn];    //  记录匹配对象
void init()
{
    memset(match, -1, sizeof(match));
    for (int i = 0; i < maxn; ++i) {
        G[i].clear();
    }
}
bool findpath(int k) {
    for (int i = 0; i < G[k].size(); ++i) {
        int v = G[k][i];
        if (!inpath[v]) {
            inpath[v] = true;
            if (match[v] == -1 || findpath(match[v])) { // 递归
                match[v] = k; // 即匹配对象是“k妹子”的
                return true;
            }
        }
    }
    return false;
}

void hungary() {
    int cnt = 0;
    for (int i = 1; i <= m; i++) {  // m为需要匹配的“妹子”数
        memset(inpath, false, sizeof(inpath)); // 每次都要初始化
        if (findpath(i)) cnt++;
    }
    cout << cnt << endl;
}

/*
    /求解最大匹配问题/
    /dfs实现/
    /16/11/05ztx/
*/
int v1, v2;
bool Map[501][501];
bool visit[501];
int link[501];
int result;

bool dfs(int x)  {
    for (int y = 1; y <= v2; ++y)  {
        if (Map[x][y] && !visit[y])  {
            visit[y] = true;
            if (link[y] == 0 || dfs(link[y]))  {
                link[y] = x;
                return true;
            } } }
    return false;
}

void Search()  {
    for (int x = 1; x <= v1; x++)  {
        memset(visit,false,sizeof(visit));
        if (dfs(x))
            result++;
    }
}
```

动态规划

背包

16.17.18背包问题

```
/*
    |01背包|
    |完全背包|
    |多重背包|
    |16/11/05ztx|
*/
//  01背包:

void bag01(int cost,int weight)  {
    for(i = v; i >= cost; --i)
    dp[i] = max(dp[i], dp[i-cost]+weight);
}

//  完全背包:

void complete(int cost, int weight)  {
    for(i = cost ; i <= v; ++i)
    dp[i] = max(dp[i], dp[i - cost] + weight);
}

//  多重背包:

void multiply(int cost, int weight, int amount)  {
    if(cost * amount >= v)
        complete(cost, weight);
    else{
        k = 1;
        while (k < amount){
            bag01(k * cost, k * weight);
            amount -= k;
            k += k;
        }
        bag01(cost * amount, weight * amount);
    }
}

// other

int dp[1000000];
int c[55], m[110];
int sum;

void CompletePack(int c) {
    for (int v = c; v <= sum / 2; ++v){
        dp[v] = max(dp[v], dp[v - c] + c);
    }
}

void ZeroOnePack(int c) {
    for (int v = sum / 2; v >= c; --v) {
        dp[v] = max(dp[v], dp[v - c] + c);
    }
}

void multiplePack(int c, int m)  {
    if (m * c > sum / 2)
        CompletePack(c);
    else{
        int k = 1;
        while (k < m){
            ZeroOnePack(k * c);
            m -= k;
            k <<= 1;
        }
        if (m != 0){
            ZeroOnePack(m * c);
        }
    }
}
```

LIS

19.最长上升子序列

```
/*
    |最长上升子序列|
    |状态转移|
    |16/11/05ztx|
*/
/*
    状态转移dp[i] = max{ 1.dp[j] + 1 };  j<i; a[j]<a[i];
    d[i]是以i结尾的最长上升子序列
    与i之前的 每个a[j]<a[i]的 j的位置的最长上升子序列+1后的值比较
*/

void solve(){    // 参考挑战程序设计入门经典;
    for(int i = 0; i < n; ++i){
        dp[i] = 1;
        for(int j = 0; j < i; ++j){
            if(a[j] < a[i]){
                dp[i] = max(dp[i], dp[j] + 1);
            } } }
}


/*
    优化方法:
    dp[i]表示长度为i+1的上升子序列的最末尾元素
    找到第一个比dp末尾大的来代替
*/

void solve() {
        for (int i = 0; i < n; ++i){
            dp[i] = INF;
        }
        for (int i = 0; i < n; ++i) {
            *lower_bound(dp, dp + n, a[i]) = a[i];  //  返回一个指针
        }
        printf("%d\n", *lower_bound(dp, dp + n, INF) - dp;
    }


/*
    函数lower_bound()返回一个 iterator 它指向在[first,last)标记的有序序列中
可以插入value，而不会破坏容器顺序的第一个位置，而这个位置标记了一个不小于
value的值。
*/
```

LCS

20.最长公共子序列

```
/*
    |求最长公共子序列|
    /递推形式/
    /16/11/05ztx/
*/
void solve() {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (s1[i] == s2[j]) {
                dp[i + 1][j + 1] = dp[i][j] + 1;
            }else {
                dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j]);
            } } }
}
```

计算几何

21.向量基本用法

```
/*
    /16/11/06ztx/
*/
struct node {
    double x; // 横坐标
    double y; // 纵坐标
};

typedef node Vector;

Vector operator + (Vector A, Vector B) { return Vector(A.x + B.x, A.y +
B.y); }
Vector operator - (Point A, Point B) { return Vector(A.x - B.y, A.y -
B.y); }
Vector operator * (Vector A, double p) { return Vector(A.x*p, A.y*p); }
Vector operator / (Vector A, double p) { return Vector(A.x / p, A.y*p); }

double Dot(Vector A, Vector B) { return A.x*B.x + A.y*B.y; } // 向量点乘
double Length(Vector A) { return sqrt(Dot(A, A)); }  // 向量模长
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) /
Length(B)); }  // 向量之间夹角

double Cross(Vector A, Vector B) { // 叉积计算 公式
    return A.x*B.y - A.y*B.x;
}

Vector Rotate(Vector A, double rad) // 向量旋转 公式  {
    return Vector(A.x*cos(rad) - A.y*sin(rad), A.x*sin(rad) +
A.y*cos(rad));
}
```

```
Point getLineIntersection(Point P, Vector v, Point Q, Vector w) { // 两直线交点t1 t2计算公式
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);  // 求得是横坐标
    return P + v*t;  // 返回一个点
}
```

22.求多边形面积

```
/*
    /16/11/06ztx/
*/
node G[maxn];
int n;

double Cross(node a, node b) { // 叉积计算
    return a.x*b.y - a.y*b.x;
}

int main()
{
    while (scanf("%d", &n) != EOF && n) {
        for (int i = 0; i < n; i++)
            scanf("%lf %lf", &G[i].x, &G[i].y);
        double sum = 0;
        G[n].x = G[0].x;
        G[n].y = G[0].y;
        for (int i = 0; i < n; i++) {
                sum += Cross(G[i], G[i + 1]);
        }
        // 或者
            //for (int i = 0; i < n; i++) {
                //sum += fun(G[i], G[（i + 1）% n]);
            //}
        sum = sum / 2.0;
        printf("%.1f\n", sum);
    }
    system("pause");
    return 0;
}
```

23..判断线段相交

```
/*
    /16/11/06ztx/
*/
```

```
node P[35][105];

double Cross_Prouct(node A,node B,node C) {      //  计算BA叉乘CA
    return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
}
bool Intersect(node A,node B,node C,node D)  {  //  通过叉乘判断线段是否相
交；
    if(min(A.x,B.x)<=max(C.x,D.x)&&         //  快速排斥实验；
       min(C.x,D.x)<=max(A.x,B.x)&&
       min(A.y,B.y)<=max(C.y,D.y)&&
       min(C.y,D.y)<=max(A.y,B.y)&&
       Cross_Prouct(A,B,C)*Cross_Prouct(A,B,D)<0&&       //  跨立实验；
       Cross_Prouct(C,D,A)*Cross_Prouct(C,D,B)<0)        //  叉乘异号表示在
两侧；
       return true;
    else return false;
}
```

24.求三角形外心

```
/*
    /16/11/06ztx/
*/
Point circumcenter(const Point &a, const Point &b, const Point &c) { //返回
三角形的外心
    Point ret;
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1 + b1*b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2 + b2*b2) / 2;
    double d = a1*b2 - a2*b1;
    ret.x = a.x + (c1*b2 - c2*b1) / d;
    ret.y = a.y + (a1*c2 - a2*c1) / d;
    return ret;
}
```

24.极角排序

```
/*
    /16/11/06ztx/
*/
double cross(point p1, point p2, point q1, point q2) {  // 叉积计算
    return (q2.y - q1.y)*(p2.x - p1.x) - (q2.x - q1.x)*(p2.y - p1.y);
}
bool cmp(point a, point b)  {
    point o;
    o.x = o.y = 0;
    return cross(o, b, o, a) < 0; // 叉积判断
}
```

```
sort(convex + 1, convex + cnt, cmp); // 按角排序，从小到大
```

字符串

kmp

25.克努特-莫里斯-普拉特操作

```
/*
    |kmp算法|
    |字符串匹配|
    |17/1/21ztx|
*/
void getnext(char str[maxn], int nextt[maxn]) {
    int j = 0, k = -1;
    nextt[0] = -1;
    while (j < m) {
        if (k == -1 || str[j] == str[k]) {
            j++;
            k++;
            nextt[j] = k;
        }
        else
            k = nextt[k];
    }
}

void kmp(int a[maxn], int b[maxn]) {
    int nextt[maxm];
    int i = 0, j = 0;
    getnext(b, nextt);
    while (i < n) {
        if (j == -1 || a[i] == b[j]) { // 母串不动，子串移动
            j++;
            i++;
        }
        else {
            // i不需要回溯了
            // i = i - j + 1;
            j = nextt[j];
        }
        if (j == m) {
            printf("%d\n", i - m + 1); // 母串的位置减去子串的长度+1
            return;
        }
    }
    printf("-1\n");
}
```

26.kmp扩展

```
/*
    /16/11/06ztx/
*/
#include<iostream>
#include<cstring>

using namespace std;

const int MM=100005;

int next[MM],extand[MM];
char S[MM],T[MM];

void GetNext(const char *T) {
    int len = strlen(T),a = 0;
    next[0] = len;
    while(a < len - 1 && T[a] == T[a + 1]) a++;
    next[1] = a;
    a = 1;
    for(int k = 2; k < len; 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 < len && T[k + j] == T[j]) j++;
            next[k] = j;
            a = k;
        }else next[k] = L;
    }
}
void GetExtand(const char *S,const char *T) {
    GetNext(T);
    int slen = strlen(S),tlen = strlen(T),a = 0;
    int MinLen = slen < tlen ? slen : tlen;
    while(a < MinLen && S[a] == T[a]) a++;
    extand[0] = a;
    a = 0;
    for(int k = 1; k < slen; k ++) {
        int p = a + extand[a] - 1, L = next[k - a];
        if( (k - 1) + L >= p) {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while(k + j < slen && j < tlen && S[k + j] == T[j]) j ++;
            extand[k] = j;
            a = k;
        } else
            extand[k] = L;
    }
}
void show(const int *s,int len){
    for(int i = 0; i < len; i ++)
            cout << s[i] << ’ ’;
    cout << endl;
}

int main() {
    while(cin >> S >> T) {
        GetExtand(S,T);
        show(next,strlen(T));
        show(extand,strlen(S));
    }
    return 0;
}
```

字典树

27.字典树

```
/*
    |16/11/06ztx|
*/
struct Trie{
    int cnt;
    Trie *next[maxn];
    Trie(){
        cnt = 0;
        memset(next,0,sizeof(next));
    }
};

Trie *root;

void Insert(char *word)  {
    Trie *tem = root;
    while(*word != ’\0’)  {
        int x = *word - ’a’;
        if(tem->next[x] == NULL)
            tem->next[x] = new Trie;
        tem = tem->next[x];
        tem->cnt++;
        word++;
    }
}

int Search(char *word)  {
    Trie *tem = root;
    for(int i=0;word[i]!=’\0’;i++)  {
        int x = word[i]-’a’;
        if(tem->next[x] == NULL)
            return 0;
        tem = tem->next[x];
    }
    return tem->cnt;
}

void Delete(char *word,int t) {
    Trie *tem = root;
    for(int i=0;word[i]!=’\0’;i++)  {
        int x = word[i]-’a’;
        tem = tem->next[x];
        (tem->cnt)-=t;
    }
    for(int i=0;i<maxn;i++)
        tem->next[i] = NULL;
}

int main() {
    int n;
    char str1[50];
    char str2[50];
    while(scanf(″%d″,&n)!=EOF)  {
        root = new Trie;
        while(n--)  {
            scanf(″%s %s″,str1,str2);
            if(str1[0]==’i’) {
                Insert(str2);
            }else if(str1[0] == ’s’)  {
                if(Search(str2))
                    printf(″Yes\n″);
                else
                    printf(″No\n″);
            }else  {
                int t = Search(str2);
                if(t)
                    Delete(str2,t);
            } } }
    return 0;
}
```

28.AC自动机

```
/*
    |16/11/06ztx|
*/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>

using namespace std;

#define N 1000010

char str[N], keyword[N];
int head, tail;

struct node {
    node *fail;
    node *next[26];
    int count;
    node() { //init
        fail = NULL;// 默认为空
        count = 0;
        for(int i = 0; i < 26; ++i)
            next[i] = NULL;
    }
}*q[N];

node *root;

void insert(char *str)  { // 建立Trie
    int temp, len;
    node *p = root;
    len = strlen(str);
    for(int i = 0; i < len; ++i)  {
        temp = str[i] - 'a';
        if(p->next[temp] == NULL)
            p->next[temp] = new node();
        p = p->next[temp];
    }
    p->count++;
}

void build_ac() { // 初始化fail指针，BFS 数组模拟队列：
    q[tail++] = root;
    while(head != tail)  {
        node *p = q[head++]; // 弹出队头
        node *temp = NULL;
        for(int i = 0; i < 26; ++i)  {
            if(p->next[i] != NULL)  {
                if(p == root) { // 第一个元素fail必指向根
                    p->next[i]->fail = root;
                }else {
                    temp = p->fail; // 失败指针
                    while(temp != NULL) { // 2种情况结束：匹配为空or找到匹
配
                        if(temp->next[i] != NULL) { // 找到匹配
                            p->next[i]->fail = temp->next[i];
                            break;
                        }
                        temp = temp->fail;
                    }
                    if(temp == NULL) // 为空则从头匹配
                        p->next[i]->fail = root;
                }
                q[tail++] = p->next[i]; // 入队
            } } }
}

int query() // 扫描
{
    int index, len, result;
    node *p = root; // Tire入口
    result = 0;
    len = strlen(str);
    for(int i = 0; i < len; ++i)
    {
        index = str[i] - ’a’;
        while(p->next[index] == NULL && p != root) // 跳转失败指针
            p = p->fail;
        p = p->next[index];
        if(p == NULL)
            p = root;
        node *temp = p; // p不动，temp计算后缀串
        while(temp != root && temp->count != -1)   {
            result += temp->count;
            temp->count = -1;
            temp = temp->fail;
        }
    }
    return result;
}
```

```
int main() {
    int num;
    head= tail = 0;
    root = new node();
    scanf("%d", &num);
    getchar();
    for(int i = 0; i < num; ++i) {
       scanf("%s",keyword);
        insert(keyword);
    }
    build_ac();
    scanf("%s", str);
    if(query())
        printf("YES\n");
    else
        printf("NO\n");
    return 0;
}
```

/*   假设有N个模式串，平均长度为L；文章长度为M。 建立Trie树：O(N*L) 建立fail指针：O(N*L) 模式匹配：O(M*L) 所以，总时间复杂度为:O((N+M)*L)。*/

线段树

29.线段树

1）点更新

```
/*
    /16/12/07ztx/
*/
struct node
{
    int left, right;
    int max, sum;
};

node tree[maxn << 2];
int a[maxn];
int n;
int k = 1;
int p, q;
string str;

void build(int m, int l, int r)//m 是 树的标号
{
    tree[m].left = l;
    tree[m].right = r;
    if (l == r){
        tree[m].max = a[l];
        tree[m].sum = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(m << 1, l, mid);
    build(m << 1 | 1, mid + 1, r);
    tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);
    tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;
}

void update(int m, int a, int val)//a 是 节点位置， val 是 更新的值（加减的值）
{
    if (tree[m].left == a && tree[m].right == a){
        tree[m].max += val;
        tree[m].sum += val;
        return;
    }
    int mid = (tree[m].left + tree[m].right) >> 1;
    if (a <= mid){
        update(m << 1, a, val);
    }
    else{
        update(m << 1 | 1, a, val);
    }
    tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);
    tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;
}

int querySum(int m, int l, int r)
{
    if (l == tree[m].left && r == tree[m].right){
        return tree[m].sum;
    }
    int mid = (tree[m].left + tree[m].right) >> 1;
    if (r <= mid){
        return querySum(m << 1, l, r);
    }
    else if (l > mid){
        return querySum(m << 1 | 1, l, r);
    }
    return querySum(m << 1, l, mid) + querySum(m << 1 | 1, mid + 1, r);
}
```

```
int queryMax(int m, int l, int r)
{
    if (l == tree[m].left && r == tree[m].right){
        return tree[m].max;
    }
    int mid = (tree[m].left + tree[m].right) >> 1;
    if (r <= mid){
        return queryMax(m << 1, l, r);
    }
    else if (l > mid){
        return queryMax(m << 1 | 1, l, r);
    }
    return max(queryMax(m << 1, l, mid), queryMax(m << 1 | 1, mid + 1, r));
}

build(1,1,n);
update(1,a,b);
query(1,a,b);
```

2)区间更新

```
/*
    /16/11/06ztx/
*/
typedef long long ll;
const int maxn = 100010;

int t,n,q;
ll anssum;

struct node{
    ll l,r;
    ll addv,sum;
}tree[maxn<<2];

void maintain(int id) {
    if(tree[id].l >= tree[id].r)
        return ;
    tree[id].sum = tree[id<<1].sum + tree[id<<1|1].sum;
}

void pushdown(int id) {
    if(tree[id].l >= tree[id].r)
        return ;
    if(tree[id].addv){
        int tmp = tree[id].addv;
        tree[id<<1].addv += tmp;
        tree[id<<1|1].addv += tmp;
        tree[id<<1].sum += (tree[id<<1].r - tree[id<<1].l + 1)*tmp;
        tree[id<<1|1].sum += (tree[id<<1|1].r - tree[id<<1|1].l + 1)*tmp;
        tree[id].addv = 0;
    }
}

void build(int id,ll l,ll r) {
    tree[id].l = l;
    tree[id].r = r;
    tree[id].addv = 0;
    tree[id].sum = 0;
    if(l==r)  {
        tree[id].sum = 0;
        return ;
    }
    ll mid = (l+r)>>1;
    build(id<<1,l,mid);
    build(id<<1|1,mid+1,r);
    maintain(id);
}

void updateAdd(int id,ll l,ll r,ll val) {
    if(tree[id].l >= l && tree[id].r <= r)
    {
        tree[id].addv += val;
        tree[id].sum += (tree[id].r - tree[id].l+1)*val;
        return ;
    }
    pushdown(id);
    ll mid = (tree[id].l+tree[id].r)>>1;
    if(l <= mid)
        updateAdd(id<<1,l,r,val);
    if(mid < r)
        updateAdd(id<<1|1,l,r,val);
    maintain(id);
}

void query(int id,ll l,ll r) {
    if(tree[id].l >= l && tree[id].r <= r){
        anssum += tree[id].sum;
        return ;
    }
    pushdown(id);
    ll mid = (tree[id].l + tree[id].r)>>1;
    if(l <= mid)
        query(id<<1,l,r);
    if(mid < r)
        query(id<<1|1,l,r);
    maintain(id);
}

int main() {
    scanf("%d",&t);
    int kase = 0 ;
    while(t--){
        scanf("%d %d",&n,&q);
        build(1,1,n);
        int id;
        ll x,y;
        ll val;
        printf("Case %d:\n",++kase);
        while(q--){
            scanf("%d",&id);
            if(id==0){
                scanf("%lld %lld %lld",&x,&y,&val);
                updateAdd(1,x+1,y+1,val);
            }
            else{
                scanf("%lld %lld",&x,&y);
                anssum = 0;
                query(1,x+1,y+1);
                printf("%lld\n",anssum);
            } } }
    return 0;
}
```

30.树状数组

```
/*
    /16/11/06ztx/
*/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
#include<cmath>

using namespace std;

typedef long long ll;

const int maxn = 50005;
```

```
int a[maxn];
int n;

int lowbit(const int t) {
    return t & (-t);
}

void insert(int t, int d) {
    while (t <= n){
        a[t] += d;
        t = t + lowbit(t);
    }
}

ll getSum(int t) {
    ll sum = 0;
    while (t > 0){
        sum += a[t];
        t = t - lowbit(t);
    }
    return sum;
}

int main() {
    int t, k, d;
    scanf("%d", &t);
    k= 1;
    while (t--){
        memset(a, 0, sizeof(a));
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &d);
            insert(i, d);
        }
        string str;
        printf("Case %d:\n", k++);
        while (cin >> str) {
            if (str == "End")   break;
            int x, y;
            scanf("%d %d", &x, &y);
            if (str == "Query")
                printf("%lld\n", getSum(y) - getSum(x - 1));
            else if (str == "Add")
                insert(x, y);
            else if (str == "Sub")
                insert(x, -y);
        }
    }
    return 0;
}
```

其他

31.中国剩余定理（孙子定理）

```
/*
    /16/11/06ztx/
*/
int CRT(int a[],int m[],int n)  {
    int M = 1;
    int ans = 0;
    for(int i=1; i<=n; i++)
        M *= m[i];
    for(int i=1; i<=n; i++)  {
        int x, y;
        int Mi = M / m[i];
        extend_Euclid(Mi, m[i], x, y);
        ans = (ans + Mi * x * a[i]) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}

void extend_Euclid(int a, int b, int &x, int &y)  {
    if(b == 0) {
        x = 1;
        y = 0;
        return;
    }
    extend_Euclid(b, a % b, x, y);
    int tmp = x;
    x = y;
    y = tmp - (a / b) * y;
}
```