头文件#include<algorithm>

Int num[maxn];

Sort(num,num+10) 前十个元素从小到大排列

逆序排列：

bool cmp(int a,int b)

{

    return a>b;

}

Sort(num,num+10,cmp) 前十个从大到小排序

头文件#include<cstring>

Memset(num,0,sizeof(num)); 常用于初始化0或-1.

大数

加法、乘法

#include <stdio.h>

#include <iostream>

#include <string.h>

using namespace std;

const int maxn = 10000 + 10;

//加法

string bigIntegerAdd(string s1,string s2){

int a[maxn],b[maxn];

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

int len1 = s1.size(),len2 = s2.size();

int maxL = max(len1,len2);

for(int i = 0; i < len1; i++)a[i] = s1[len1-1-i]-'0';

for(int i = 0; i < len2; i++)b[i] = s2[len2-1-i]-'0';

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

if(a[i]+b[i] >= 10){

int temp = a[i]+b[i];

a[i] = temp%10;

a[i+1] += (temp/10);

}

else a[i] += b[i];

}

string c = "";

if(a[maxL] != 0)

c += a[maxL] + '0';

for(int i = maxL-1; i >= 0; i--)c += a[i] + '0';

return c;

}

//乘法

string bigIntegerMul(string s1,string s2){

int a[maxn],b[maxn],c[maxn\*2 + 5];

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

memset(c,0,sizeof(c));

int len1 = s1.size(),len2 = s2.size();

for(int i = 0; i < len1; i++)a[i] = s1[len1-1-i]-'0'; //倒置

for(int i = 0; i < len2; i++)b[i] = s2[len2-1-i]-'0';

for(int i = 0; i < len1; i++){

for(int j = 0; j < len2; j++){

c[i+j] += a[i]\*b[j];

}

}

for(int i = 0; i < maxn\*2; i++){

if(c[i] >= 10){

c[i+1] += c[i]/10;

c[i] %= 10;

}

}

string ans = "";

int i;

for(i = maxn \* 2; i >= 0; i--)

if(c[i] != 0)

break;

for(;i >= 0; i--)ans += c[i] + '0';

return ans;

}

int main(){

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

int n;

string s[255];

s[0] = "1",s[1] = "1"; //注意0的时候是1

for(int i = 2;i <= 255; i++){

string temp = bigIntegerMul("2",s[i-2]);

s[i] = bigIntegerAdd(s[i-1],temp);

}

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

cout<<s[n]<<endl;

return 0;

}

减法

#include <bits/stdc++.h>

const int maxn = 200 + 10;

using namespace std;

typedef long long LL;

//具体实现

string subInfo(char \*s1,char \*s2){

int a[maxn],b[maxn];

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

int len1 = strlen(s1),len2 = strlen(s2);

int maxLen = max(len1,len2);

for(int i = 0; i < len1; i++) a[i] = s1[len1 - i - 1] - '0';

for(int i = 0; i < len2; i++) b[i] = s2[len2 - i - 1] - '0';

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

if(a[i]-b[i] < 0){

a[i] = a[i]+10-b[i];

a[i+1] -= 1;

}

else a[i] -= b[i];

}

string str = "";

int i;

for(i = maxLen-1; i >= 0; i--)if(a[i] != 0)break;

for(;i >= 0; i--)str += a[i]+'0';

return str;

}

//大数减法的模板

string bigIntegerSub(char \*s1,char \*s2){

if(s1 == s2)

return "0"; //相等

int len1 = strlen(s1),len2 = strlen(s2);

if(len1 > len2)

return subInfo(s1,s2);

else if(len1 < len2)

return "-" + subInfo(s2,s1); //负数

else { //长度相等时判断大小

for(int i = 0; i < len1; i++){

if(s1[i]-'0' > s2[i]-'0')

return subInfo(s1,s2);

else if(s1[i]-'0' < s2[i]-'0')

return "-" + subInfo(s2,s1);

}

}

}

int main(){

char s1[maxn],s2[maxn];

scanf("%s\n%s",s1,s2);

cout<<bigIntegerSub(s1,s2)<<endl;

return 0;

}

求阶乘以及位数

//大数阶乘的模板

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100000 + 10;

//大数计算阶乘位数

//lg(N!)=[lg(N\*(N-1)\*(N-2)\*......\*3\*2\*1)]+1 = [lgN+lg(N-1)+lg(N-2)+......+lg3+lg2+lg1]+1;

int factorialDigit(int n){

double sum = 0;

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

sum += log10(i);

}

return (int)sum+1;

}

//大数计算阶乘

string bigFactorial(int n){

int ans[maxn],digit = 1;

ans[0] = 1;

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

int num = 0;

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

int temp = ans[j]\*i + num;

ans[j] = temp%10;

num = temp/10;

}

while(num != 0){

ans[digit] = num%10;

num /= 10;

digit++;

}

}

string str = "";

for(int i = digit-1; i >= 0; i--)

str += ans[i] + '0';

return str;

}

int main(){

int n;

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

//cout<<factorialDigit(n)<<endl; //阶乘的位数

cout<<bigFactorial(n)<<endl; //求出阶乘

}

return 0;

}

二分

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

int cmp(const void \*a, const void \*b) {

return \*(int \*) a - \*(int \*) b;

}

//普通的二分查找

int bs(int \*arr,int L,int R,int target){

while( L <= R){

int mid = (L) + (R-L)/2;

if(arr[mid] == target)

return mid;

if(arr[mid] > target)

R = mid - 1;

else

L = mid + 1;

}

return -1; // not find

}

//求最小的i，使得a[i] = target，若不存在，则返回-1

//返回 如果有重复的 下界(比如1,2,2,2,3,4)查找2,返回1

int firstEqual(int arr[], int L, int R, int target) {

while (L < R) {

int mid = L + (R - L) / 2;

if (arr[mid] < target)

L = mid + 1;

else

R = mid;

}

if (arr[L] == target)

return L;

return -1;

}

//求最大的i的下一个元素的下标(c++中的upperbound函数)，使得a[i] = target，若不存在，则返回-1

int lastEqualNext(int arr[], int L, int R, int target) {

while (L < R) {

int m = L + (R - L) / 2;

if (arr[m] <= target)

L = m + 1;

else

R = m;

}

if (arr[L - 1] == target)

return L;

return -1;

}

//求最大的i，使得a[i] = target，若不存在，则返回-1

int lastEqual(int arr[], int L, int R, int target) {

while (L < R) {

int mid = L + ((R + 1 - L) >> 1);//向上取整

if (arr[mid] <= target)

L = mid;

else

R = mid - 1;

}

if (arr[L] == target)

return L;

return -1;

}

//求最小的i，使得a[i] > target，若不存在，则返回-1

int firstLarge(int arr[], int L, int R, int target) {

while (L < R) {

int m = L + ((R - L) >> 1);//向下取整

if (arr[m] <= target)

L = m + 1;

else

R = m;

}

if (arr[R] > target)

return R;

return -1;

}

//求最大的i，使得a[i] < target，若不存在，则返回-1

int lastSmall(int arr[], int L, int R, int target) {

while (L < R) {

int m = L + ((R + 1 - L) >> 1);//向上取整

if (arr[m] < target)

L = m;

else

R = m - 1;

}

if (arr[L] < target)

return L;

return -1;

}

int main() {

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

int n, a[maxn], v;

scanf("%d", &n);

for (int i = 0; i < n; i++)scanf("%d", &a[i]); //1 3 2 9 4 1 3 7 2 2

scanf("%d", &v); //input the number you need find

qsort(a, n, sizeof(a[0]), cmp); // 1 1 2 2 2 3 3 4 7 9

printf("after sorted : \n");

for (int i = 0; i < n; i++)printf("%d ", a[i]);

printf("\n-------------test----------------");

printf("\n%d\n", firstEqual(a, 0, n, v)); //output 2 第一个

printf("%d\n", lastEqualNext(a, 0, n, v)); //output 4 + 1,最后一个的下一个

printf("%d\n", lastEqual(a, 0, n, v)); //output 4 最后一个

printf("%d\n", firstLarge(a, 0, n, v)); //output 5(第一个3大于2)

printf("%d\n", lastSmall(a, 0, n, v)); //output 1(不是0)

return 0;

}

枚举排列

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

void permutation(int \*arr, int n, int cur){

if(cur == n){ // 边界

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

printf("%d ",arr[i]);

printf("\n");

}

else for(int i = 1; i <= n; i++){ //尝试在arr[cur]中填充各种整数

bool flag = true;

for(int j = 0; j < cur; j++)if(i == arr[j]){ // 如果i已经在arr[0]~arr[cur-1]中出现过，则不能选

flag = false;

break;

}

if(flag){

arr[cur] = i; //把i填充到当前位置

permutation(arr, n, cur+1);

}

}

}

// 求 1 ~ n 的全排列，arr数组作为中间打印数组

int main(int argc, char const \*\*argv)

{

int a[maxn], n;

scanf("%d", &n);

permutation(a, n, 0);

return 0;

}

可重集的全排列

#include <bits/stdc++.h>

const int maxn = 100 + 10;

void permutation(int \*arr,int \*p,int n,int cur){

if(cur == n){

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

printf("%d ",arr[i]);

printf("\n");

}else for(int i = 0; i < n; i++)if(!i || p[i] != p[i-1]){

int c1 = 0, c2 = 0;

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

if(p[j] == p[i]) // 重复元素的个数

c1++;

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

if(arr[j] == p[i]) // 前面已经排列的重复元素的个数

c2++;

if(c2 < c1){

arr[cur] = p[i];

permutation(arr, p, n, cur+1);

}

}

}

int main(){

int a[maxn], p[maxn] = {5, 6, 7, 5}; //可以有重复元素的全排列

std::sort(p, p+4);

permutation(a, p, 4, 0);

return 0;

}

子集生成

增量构造法

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

//打印0~n-1的所有子集

//按照递增顺序就行构造子集 防止子集的重复

void print\_subset(int \*arr, int n, int cur){

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

printf("%d ", arr[i]);

printf("\n");

int s = cur ? arr[cur-1] + 1 : 0; //确定当前元素的最小可能值

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

arr[cur] = i;

print\_subset(arr, n, cur+1);

}

}

int main(){

int n, arr[maxn];

scanf("%d", &n);

print\_subset(arr, n, 0);

return 0;

}

位向量法

// 1~n 的所有子集：位向量法

#include <bits/stdc++.h>

const int maxn = 100 + 10;

using namespace std;

int bits[maxn];//位向量bits[i] = 1,当且仅当i在子集 A 中

void print\_subset(int n,int cur){

if(cur == n){

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

if(bits[i])

printf("%d ",i);

printf("\n");

return;

}

bits[cur] = 1;

print\_subset(n,cur + 1);

bits[cur] = 0;

print\_subset(n,cur + 1);

}

int main() {

int n;

scanf("%d", &n);

print\_subset(n,0);

return 0;

}

二进制枚举

// 0 ~ n-1的所有子集：二进制法枚举0 ~ n-1的所有子集

#include <bits/stdc++.h>

const int maxn = 100 + 10;

using namespace std;

void print\_subset(int n,int cur){

//这一步其实就是判断 cur 的二进制的各个位上是不是1，如果是1,就输出对应的那个位置(位置从0开始)

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

if(1 & (cur >> i))

printf("%d ",i);

printf("\n");

}

int main(int argc, char const\*\* argv)

{

int n;

scanf("%d",&n);

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

print\_subset(n,i);//枚举各子集对应的编码 0,1,2...pow(2,n) - 1

return 0;

}

N皇后

#include <bits/stdc++.h>

const int maxn = 100 + 10;

using namespace std;

int sum,n,cnt; //解的个数，n皇后，递归次数

int C[maxn];

void Search(int cur){ //逐行放置皇后

cnt++;

if(cur == n)sum++;

else for(int i = 0; i < n; i++){ //尝试在各列放置皇后

bool flag = true;

C[cur] = i; //尝试把第cur行的皇后放在第i列//如果 等下不行的话 就下一个 i++

for(int j = 0; j < cur; j++){ //检查是否和已经放置的冲突

if(C[cur] == C[j] || C[cur] + cur == C[j] + j || cur - C[cur] == j - C[j]){//检查列,"副对角线","主对角线"

flag = false;break;

}

}

if(flag)Search(cur+1);

}

}

int main(){

scanf("%d",&n); //输入n皇后

sum = cnt = 0;//解的个数 和 递归的次数

Search(0);

printf("%d %d\n",sum,cnt);

return 0;

}

并查集

//题目大意 : 病毒传染，可以通过一些社团接触给出一些社团(0号人物是被感染的)问有多少人(0~n-1个人)被感染

#include <stdio.h>

const int maxn = 100000 + 10;

int parent[maxn], rank[maxn]; //parent[]保存祖先,rank记录每个'树的高度'

void init(){

for(int i = 0; i < maxn; i++)parent[i] = i; //注意这里

for(int i = 0; i < maxn; i++)rank[i] = 1;

}

//int findRoot(int v){

// return parent[v] == v ? v : parent[v] = findRoot(parent[v]);

//}

// 非递归

int findRoot(int v){

while(parent[v] != v){

parent[v] = parent[parent[v]]; // 路径压缩

v = parent[v];

}

return v;

}

void unions(int a, int b){

int aRoot = findRoot(a);

int bRoot = findRoot(b);

if (aRoot == bRoot)

return;

if (rank[aRoot] < rank[bRoot])

parent[aRoot] = bRoot;

else if(rank[aRoot] > rank[bRoot]){

parent[bRoot] = aRoot;

}else{

parent[aRoot] = bRoot;

rank[bRoot]++;

}

}

int is\_same(int x,int y){ //检查是不是在同一个集合中

return findRoot(x) == findRoot(y);

}

int main(){

int n,m,k,x,root;

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

init();

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

scanf("%d%d",&k,&root);

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

scanf("%d",&x);

unions(root,x);

}

}

int sum = 1;

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

if(findRoot(i) == findRoot(0))

sum++; //找和0是一个集合的

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

}

return 0;

}

KMP 查找模式串在文本串的位置

#include <stdio.h>

const int maxn = 1000000 + 10;

int n,m,a[maxn], b[10000 + 10],nexts[10000 + 10];

void getNext(int \*p,int next[]) { //优化后的求next数组的方法

int len = m;

next[0] = -1; //next 数组中的 最大长度值(前后缀的公共最大长度) 的第一个 赋值为 -1

int k = -1,j = 0;

while (j < len - 1) {

if (k == -1 || p[j] == p[k]) { //p[k]表示前缀 p[j] 表示后缀

k++; j++;

if(p[j] != p[k])next[j] = k;

else next[j] = next[k]; //因为不能出现p[j] = p[ next[j ]]，所以当出现时需要继续递归，k = next[k] = next[next[k]]

}

else k = next[k];

}

}

int KMPSerach(int \*s, int \*p) {

int sLen = n,pLen = m;

int i = 0, j = 0;

while (i < sLen && j < pLen) {

if (j == -1 || s[i] == p[j])i++, j++;

else j = nexts[j];

}

if (j == pLen)return i - j;

else return -1;

}

int main() {

int T;

scanf("%d", &T);

while (T--) {

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

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

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

getNext(b,nexts); //获取next数组

int ans = KMPSerach(a, b);

if (ans != -1)printf("%d\n", ans + 1);

else printf("-1\n");

}

return 0;

}

01背包问题

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn = 1000+5;

int w[maxn],v[maxn],dp[maxn][maxn],vis[maxn];

//打印解

void print(int n,int C){

for(int i = n; i > 1; i--){

if(dp[i][C] == dp[i-1][C-w[i]] + v[i]){

vis[i] = 1;

C -= w[i];

}

}

vis[1] = (dp[1][C] > 0) ? 1: 0;

}

int main(){

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

int n,C,T;

scanf("%d",&T);

while(T--){

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

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

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

memset(dp,0,sizeof(dp)); //dp[0][0~C]和dp[0~N][0]都为0，前者表示前0个物品无论装入多大的包中总价值都为0，后者表示体积为0的背包都装不进去。

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

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

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

dp[i][j] = dp[i-1][j]; //如果j不大于v[i]的话就dp[i][j] = dp[i-1][j];

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

}

}

printf("%d\n",dp[n][C]); //n个物品装入C容量的包能获得的最大价值

//print(n,C);

//for(int i = 1; i <= n; i++)if(vis[i])printf("%d ",i); //输出选中的物品

}

return 0;

}

完全背包

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn = 50000 + 10;

const int INF = -0X7ffff;

int w[maxn],v[maxn],dp[maxn];

int main(){

int T,n,C;

scanf("%d",&T);

while(T--){

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

for(int i = 1; i <= C; i++)dp[i] = INF;

dp[0] = 0;

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

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

for(int j = w[i]; j <= C; j++){ //从前往后递推，这样才能保证一种物品可以被使用多次

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

}

}

if(dp[C] > 0)printf("%d\n",dp[C]);

else printf("NO\n");

}

return 0;

}

最长上升子序列

O(n^2)复杂度

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn = 10000 + 10;

int dp[maxn]; /\* dp[i]记录到[0,i]数组的LIS \*/

int maxx;/\* LIS长度，初始化为1 \*/

int LIS(char \*arr, int n) {

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

dp[i] = 1;

for (int j = 0; j < i; j++) // 注意i只遍历比它小的元素

if (arr[j] < arr[i]) dp[i] = max(dp[i], dp[j] + 1); //改成arr[j] <= arr[i]就可以求非减的

maxx = max(maxx, dp[i]);

}

return maxx;

}

/\* 递归输出LIS，因为数组dp还充当了“标记”作用 \*/

void printLis(char \*arr, int index) {

bool isLIS = false;

if (index < 0 || maxx == 0)return;

if (dp[index] == maxx) {

--maxx;

isLIS = true;

}

printLis(arr, --index);

if (isLIS) printf("%c ", arr[index + 1]);

}

int main() {

char s[maxn];

int n;

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

while(n--){

maxx = 1;

scanf("%s",s);

printf("%d\n",LIS(s,strlen(s)));

//printLis(s,strlen(s)-1);printf("\n");

}

return 0;

}

O(nlgn)

#include <stdio.h>

#include <algorithm>

#include <vector>

#include <string.h>

using namespace std;

const int INF = 0x3f3f3f3f;

const int maxn = 100000 + 5;

int a[maxn], dp[maxn], pos[maxn], fa[maxn];

vector<int> ans;

//用于最长非递减子序列种的lower\_bound函数

int cmp(int a,int b){

return a <= b;

}

//最长上升子序列

//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

int LIS(int n){

memset(dp, 0x3f, sizeof(dp));

pos[0] = -1;

int i,lpos;

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

dp[lpos = (lower\_bound(dp, dp + n, a[i]) - dp)] = a[i];

pos[lpos] = i; // \*靠后打印

fa[i] = (lpos ? pos[lpos - 1] : -1);

}

n = lower\_bound(dp, dp + n, INF) - dp;

for (i = pos[n - 1]; ~fa[i]; i = fa[i]) ans.push\_back(a[i]);

ans.push\_back(a[i]);

return n;

}

//非递减的LIS

int LISS(int n){

memset(dp, 0x3f, sizeof(dp));

pos[0] = -1;

int i,lpos;

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

dp[lpos = (lower\_bound(dp, dp + n, a[i],cmp) - dp)] = a[i]; //注意这里cmp

pos[lpos] = i; // \*靠后打印

fa[i] = (lpos ? pos[lpos - 1] : -1);

}

n = lower\_bound(dp, dp + n, INF) - dp;

for (i = pos[n - 1]; ~fa[i]; i = fa[i]) ans.push\_back(a[i]);

ans.push\_back(a[i]);

return n;

}

int main(){

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

int n;

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

ans.clear();

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

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

// for(int i = ans.size()-1; i >= 0; i--) printf("%d ",ans[i]);printf("\n"); //路径打印

}

return 0;

}

最长公共子序列

#include <stdio.h>

#include <iostream>

#include <string.h>

using namespace std;

const int maxn = 1000 + 10;

int dp[maxn][maxn];

int path[maxn][maxn]; //记录路径

int LCS(char \*s1,char \*s2){

int len1 = strlen(s1)-1,len2 = strlen(s2)-1;//注意例如 abcfbc的strlen(s1)为7,所以是strlen(s1) - 1

for(int i = 0; i <= len1; i++) dp[i][0] = 0;

for(int i = 0; i <= len2; i++) dp[0][i] = 0;

for(int i = 1; i <= len1; i++){

for(int j = 1; j <= len2; j++)

if(s1[i] == s2[j]){

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

path[i][j] = 1;

}

else if(dp[i-1][j] >= dp[i][j-1]) {

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

path[i][j] = 2;

}

else {

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

path[i][j] = 3;

}

}

return dp[len1][len2];

}

//打印解

void print(int i,int j,char \*s){

if(i == 0 || j == 0)return ;

if(path[i][j] == 1){

print(i-1,j-1,s);

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

}else if(path[i][j] == 2){

print(i-1,j,s);

}else print(i,j-1,s);

}

int main(){

char s1[maxn],s2[maxn];

while(~scanf("%s%s",s1+1,s2+1)){ //注意s1[0]不取-注意例如 abcfbc的strlen(s1)为7

memset(path,0,sizeof(path));

printf("%d\n",LCS(s1,s2));

//print(strlen(s1)-1,strlen(s2)-1,s1);

}

return 0;

}

拓扑排序

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

int n, m;

int in[maxn];

vector<int>G[maxn];

void topo(){

queue<int>q;

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

if(in[i] == 0)

q.push(i);

bool flag = true;

while(!q.empty()){

int cur = q.front();

q.pop();

if(flag){

printf("%d",cur);

flag = false;

}

else

printf(" %d",cur);

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

if(--in[G[cur][i]] == 0)

q.push(G[cur][i]);

}

}

}

int main(int argc, char const \*\*argv)

{

int from, to;

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

memset(in, 0, sizeof(in));

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

G[i].clear();

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

scanf("%d%d", &from, &to);

in[to]++; //度

G[from].push\_back(to);

}

topo();

printf("\n");

}

return 0;

}

欧拉回路

#include <bits/stdc++.h>

const int maxn = 1000 + 5;

using namespace std;

int n, m, parent[maxn],ranks[maxn],in[maxn];

void init(){

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

parent[i] = i;

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

ranks[i] = 1;

}

//int findRoot(int v){

// return parent[v] == v ? v : parent[v] = findRoot(parent[v]);

//}

int findRoot(int v){

while(parent[v] != v){

parent[v] = parent[parent[v]];

v = parent[v];

}

return v;

}

void unions(int a, int b){

int aRoot = findRoot(a);

int bRoot = findRoot(b);

if (aRoot == bRoot)

return;

if (ranks[aRoot] < ranks[bRoot])

parent[aRoot] = bRoot;

else if(ranks[aRoot] > ranks[bRoot]){

parent[bRoot] = aRoot;

}else {

parent[aRoot] = bRoot;

ranks[bRoot]++;

}

}

int main(){

//freopen("in.tat","r",stdin);

int u, v;

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

init();

scanf("%d", &m);

memset(in,0,sizeof(in));

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

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

unions(u, v);

in[u]++;

in[v]++;

}

bool flag = true;

int temp = findRoot(1);

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

if(findRoot(i) != temp)

flag = false;

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

if(in[i] % 2 != 0)

flag = false; //判断度

cout << (flag ? 1 : 0) << endl;

}

return 0;

}

BFS

#include <stdio.h>

#include <iostream>

#include <stack>

#include <string.h>

#include <string>

#include <queue>

#pragma warning(disable : 4996)

using namespace std;

const int maxn = 100 + 10;

int n;

int map[maxn][maxn];

bool vis[maxn][maxn];

int dir[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};//对应上，右，下，左

struct Road {//路径记录

int x,y,d;//d表示方向

Road() {}//默认的构造函数

Road(int a,int b,int c):x(a),y(b),d(c) {}//带参数的构造函数

};

struct node {//节点类型

int x,y;

vector<Road> v;//记录路径的结构体

};

bool check(node u) {

if (!vis[u.x][u.y] && u.x >= 0 && u.x < n && u.y >= 0 && u.y < n && map[u.x][u.y] != 1)

return true;

else

return false;

}

void BFS(int x, int y) {

queue<node>q;

node now,next;

now.x = x;

now.y = y;

now.v.push\_back(Road(x,y,-1));

q.push(now);

while (!q.empty()) {

now = q.front();

q.pop();

if (now.x == n - 1 && now.y == n-1) {

for(int i = 0; i < now.v.size(); i++){

printf("(%d, %d)\n",now.v[i].x,now.v[i].y);

}

break;

}

for (int i = 0; i < 4; i++) {

next.x = now.x + dir[i][0];

next.y = now.y + dir[i][1];

if (check(next)) {

vis[next.x][next.y] = true;

next.v = now.v;

next.v[now.v.size()-1].d = i;

next.v.push\_back(Road(next.x,next.y,-1));

q.push(next);

}

}

}

}

int main() {

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

flag = false;

n = 5;

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

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

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

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

}

}

BFS(0, 0);

return 0;

}

DFS

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <string>

#include <vector>

#include <stack>

using namespace std;

const int maxn = 100+10;

const int INF = 0x3f3f3f3f;

int n,k,ans = INF;

int map[maxn][maxn];

bool vis[maxn][maxn];

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

struct node {

int x,y;

};

vector<node>dict;

stack<node>s;

void memcpy(stack<node>s){

dict.clear();

while(!s.empty()){

dict.push\_back(s.top());

s.pop();

}

}

bool Check(int x,int y){

if(!vis[x][y] && map[x][y] != 1 && x >= 0 && x < n && y >= 0 && y < n)return true;

else return false;

}

void DFS(int x,int y,int step){//深刻理解递归的含义

node now;

now.x = x;

now.y = y;

s.push(now);

if(now.x == n - 1 &&now.y == n - 1){

if(step < ans){//记录最短的路径

ans = step;

memcpy(s);

}

}

for(int i = 0; i < 4; i++){

int xx = now.x + dir[i][0];

int yy = now.y + dir[i][1];

if(Check(xx,yy)){

vis[xx][yy] = true;

DFS(xx,yy,step + 1);

vis[xx][yy] = false;//回溯

}

}

s.pop();//反弹的时候重新还原栈

}

int main(){

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

n = 5;

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

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

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

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

}

}

DFS(0,0,0);

for(int i = dict.size() - 1; i >= 0; i--){

printf("(%d, %d)\n",dict[i].x,dict[i].y);

}

return 0;

}

最小生成树

Prim

#include <stdio.h>

#include <string.h>

#include <vector>

#include <queue>

using namespace std;

const int maxn = 100 + 10;

const int INF = 1<<30;

struct Node{

int v,w;

Node (){}

Node (int v,int w):v(v),w(w){}

bool operator < (const Node&rhs ) const {

return rhs.w < w;

}

};

int n,d[maxn];

bool vis[maxn];

int Map[maxn][maxn];

void init(){

for(int i = 0; i < maxn; i++)vis[i] = false;

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

}

int Prim(int s){

priority\_queue<Node>q;

q.push(Node(s,0));

int ans = 0;

while(!q.empty()){

Node Now = q.top(); q.pop();

int v = Now.v;

if(vis[v]) continue;

ans += Now.w;

vis[v] = true;

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

int w2 = Map[v][i];

if(!vis[i] && d[i] > w2){

d[i] = w2;

q.push(Node(i,d[i]));

}

}

}

return ans;

}

int main(){

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

int m,a,b,c;

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

scanf("%d",&m);

init();

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

Map[i][i] = INF;

for(int j = 1; j < i; j++)Map[i][j] = Map[j][i] = INF;

}

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

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

if(c < Map[a][b])Map[a][b] = Map[b][a] = c; //注意重边，取小的

}

printf("%d\n",Prim(1));

}

return 0;

}

Kruskal

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <queue>

using namespace std;

const int maxn = 1e5 + 10;

int Fa[maxn],Rank[maxn];

void init(){

for(int i = 0; i <= maxn; i++)Fa[i] = i;

for(int i = 0; i <= maxn; i++)Rank[i] = 1;

}

int Find(int v){

return Fa[v] == v ? v : Fa[v] = Find(Fa[v]);

}

void Union(int x, int y){

int fx = Find(x);

int fy = Find(y);

if (fx == fy)return;

if (Rank[fx] < Rank[fy])

Fa[fx] = fy;

else{

Fa[fy] = fx;

if (Rank[fx] == Rank[fy])Rank[fy]++;

}

}

struct Edge{

int u,v,w;

Edge(){}

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

}edge[maxn\*2];

int cmp(const void \*a,const void \*b){

Edge \*aa = (Edge\*)a;

Edge \*bb = (Edge\*)b;

return aa->w > bb->w ? 1 : -1; //降序

}

int krusal(int n,int m){

qsort(edge,m,sizeof(edge[0]),cmp);

int ans = 0;

int cnt = 0;

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

int u = edge[i].u;

int v = edge[i].v;

if(Find(u) != Find(v)){

Union(u,v);

cnt++;

ans += edge[i].w;

}

if(cnt == n-1)break;

}

return ans;

}

int main(){

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

int n,m;

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

scanf("%d",&m);

init();

for(int i = 0; i < m; i++) scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w); //注意这题有重边但是没事

printf("%d\n",krusal(n,m));

}

return 0;

}

最短路

Dijkstra

//堆优化dijkstra

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e4 + 10;

const int INF = 1e9;

struct Node{

int v,w;

Node(int v,int w):v(v),w(w){}

bool operator < (const Node&rhs) const {

return rhs.w < w;

}

};

vector<Node>G[maxn];

bool vis[maxn];

int d[maxn];

int n,m;

void init(){

for(int i = 0; i < maxn; i++)G[i].clear();

for(int i = 0; i < maxn; i++)vis[i] = false;

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

}

int dijkstra(int s,int e){ //传入起点终点

priority\_queue<Node>q;

q.push(Node(s,0));

d[s] = 0;

while(!q.empty()){

Node now = q.top(); q.pop();

int v = now.v;

if(vis[v])continue;

vis[v] = true;

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

int v2 = G[v][i].v;

int w = G[v][i].w;

if(!vis[v2] && d[v2] > w+d[v]){

d[v2] = w+d[v];

q.push(Node(v2,d[v2]));

}

}

}

return d[e];

}

int main(){

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

int s,e;

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

init();

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

int x, y, z;

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

G[x].push\_back(Node(y,z));

G[y].push\_back(Node(x,z));

}

scanf("%d%d",&s,&e);

int ans = dijkstra(s,e);

if(INF != ans)printf("%d\n",ans);

else printf("-1\n");

}

return 0;

}

Floyd

#include <bits/stdc++.h>

#define maxn 1000

#define INF 0x1f1f1f1f

using namespace std;

int graph[maxn][maxn];

int low[maxn];

void floyd(int n) {

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

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

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

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

}

int main() {

int n, m, s, t;

while (cin >> n >> m) {

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

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

if (i == j)

graph[i][j] = 0;

else

graph[i][j] = INF;

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

int x, y, z;

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

if (graph[x][y] > z) graph[x][y] = graph[y][x] = z;

}

floyd(n);

scanf("%d%d", &s, &t);

if (graph[s][t] < INF) cout << graph[s][t] << endl;

else cout << "-1" << endl;

}

return 0;

}

最大公约数(GCD)和最小公倍数(LCM)

#include <bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

int gcd(int a,int b){

return b == 0?a:gcd(b,a%b);

}

int lcm(int a,int b){

return a/gcd(a,b)\*b;

}

int ngcd(int v[],int n){

if(n == 1) return v[0];

return gcd(v[n-1],ngcd(v,n-1));

}

int nlcm(int v[],int n){

if(n == 1) return v[0];

return lcm(v[n-1],nlcm(v,n-1));

}

int main(){

int n,m,a[maxn];

scanf("%d",&n);

while(n--){

scanf("%d",&m);

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

//printf("%d\n",ngcd(a,m));

printf("%d\n",nlcm(a,m));

}

return 0;

}

素数：埃式筛法

#include <stdio.h>

#include <string.h>

#include <queue>

using namespace std;

const int maxn = 10000 + 10;

int prime[maxn],s,e,vis[maxn];

bool is\_prime[maxn],flag;

//素数筛选的模板 : |埃式筛法|

int Sieve(int n) {

int k = 0;

for(int i = 0; i <= n; i++)is\_prime[i] = true;

is\_prime[0] = false,is\_prime[1] = false;

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

if(is\_prime[i]) {

prime[k++] = i;

for(int j = i\*i; j <= n; j += i)is\_prime[j] = false; // 轻剪枝，j必定是i的倍数

}

}

return k;

}

快速幂

#include <stdio.h>

//计算(a\*b)%c

long long mul(long long a,long long b,long long mod) {

long long res = 0;

while(b > 0){

if( (b&1) != 0) // 二进制最低位是1 --> 加上 a的 2^i 倍, 快速幂是乘上a的2^i ）

res = ( res + a) % mod;

a = (a << 1) % mod; // a = a \* 2 a随着b中二进制位数而扩大 每次 扩大两倍。

b >>= 1; // b -> b/2 右移 去掉最后一位 因为当前最后一位我们用完了，

}

return res;

}

//幂取模函数

long long pow1(long long a,long long n,long long mod){

long long res = 1;

while(n > 0){

if(n&1)

res = (res \* a)%mod;

a = (a \* a)%mod;

n >>= 1;

}

return res;

}

// 计算 ret = (a^n)%mod

long long pow2(long long a,long long n,long long mod) {

long long res = 1;

while(n > 0) {

if(n & 1)

res = mul(res,a,mod);

a = mul(a,a,mod);

n >>= 1;

}

return res;

}

//递归分治法求解

int pow3(int a,int n,int Mod){

if(n == 0)

return 1;

int halfRes = pow1(a,n/2,Mod);

long long res = (long long)halfRes \* halfRes % Mod;

if(n&1)

res = res \* a % Mod;

return (int)res;

}

int main(){

printf("%lld\n",mul(3,4,10));

printf("%lld\n",pow1(2,5,3));

printf("%lld\n",pow2(2,5,3));

printf("%d\n",pow3(2,5,3));

return 0;

}

字典树：用数字唯一标识字符串，字符串由字符集组成。

算法：

Insert(root , word)

P = root

For I = 1…word.len

If p.next[word[i]] == null 没有标识为word[i]的边需要新建节点

p.addChild(word[i], new Node())

p=p.next[word[i]]

p.markEndPoint() 标记为终节点

search(root, word)

p = root

for I = 1…word.len

if p.next[word[i]] == null

return false

p=p.next[word[i]]

return true

实现：

int insert(tree \*root, char \*word) {

tree \*p = root;

int i = 0;

while(word[i] != '\0') {

if (p->next[word[i]-'a'] == NULL) {

tree \*temp = new tree;

temp->isWord = false;

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

temp->next[j] = NULL;

}

temp->isWord = false;

temp->id = 0;

p->next[word[i]-'a'] = temp;

}

p = p->next[word[i]-'a'];

i++;

}

if (p->isWord) {

return p->id;

}

else {

p->isWord = true;

p->id = color++;

return p->id;

}

}