Dijkstra求最短路径：从未选过点中找最短的一条路，然后更新其他点到起点的距离。例如选点k，则dj[k]表示起点到k的距离，更新其他点到起点的距离,dj[j]=min(dj[j],dj[k]+Map[k][j]);

模板：

void Dijkstra()

{

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

for (int i = 0;i <= n;i++) dj[i] = Map[0][i];

vis[0] = 1;

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

int mindj = INF;

int pos;

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

if (dj[j] < mindj && !vis[j]){

mindj = dj[j];

pos = j;

}

}

vis[pos] = 1;

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

if (!vis[j] && dj[j] > dj[pos] + Map[pos][j]) {

dj[j] = dj[pos] + Map[pos][j];

}

}

}

}

并查集：合并不在同一个集合中的两个元素。

//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;

}

Prim算法：每次选择到当前距离生成树最小的点，加入到生成树中。

int Prim() {

int i, j, k, tmp, ans;

int mindis = 0;

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

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

vis[1] = 1;

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

tmp = INF;

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

if (!vis[j] && tmp>dis[j]) {

tmp = dis[j];

k = j;

}//找出最小距离的节点

}

vis[k] = 1;//标记已访问节点

mindis += tmp;

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

if (!vis[j] && dis[j]>Map[k][j]) {

dis[j] = Map[k][j];

}//更新加入最近点后对未加入点的距离

}

}

return mindis;//返回MST的权值

}

Kruskal：选择当前未加入生成树的最小边，若没有形成环则加入生成树。

题目：省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。经过调查评估，得到的统计表中列出了有可能建设公路的若干条道路的成本。现请你编写程序，计算出全省畅通需要的最低成本。

int parent[MAXN], rank[MAXN];

struct record{

int beg;

int end;

int value;

}s[MAXN];

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 (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 a, int b) {

return findRoot(a) == findRoot(b);

}

bool cmp(const record a, const record b) {

return a.value < b.value;

}

void init() {

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

parent[i] = i;

}

}

int Kruskal() {

init();

sort(s, s+road, cmp);

int sum = 0;

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

if (findRoot(s[i].beg) != findRoot(s[i].end)){

uninos(s[i].beg, s[i].end);

sum += s[i].value;

}

}

return sum;

}

二分查找：

普通二分查找：

int b\_search(int l, int r, int target) {

while (l <= r) {

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

if (arr[mid] == target) return mid;

else if (arr[mid] > target) r = mid - 1;

else l = mid + 1;

}

return -1;

}

求最小的i（第一个），使得a[i]==target，如不存在，返回-1.

int firstEqual(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 l, int r, int target) {

while (l < r) {

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

if (arr[m] <= target) l = mid + 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 l, int r, int target) {

while (l < r) {

int mid = l + (r-l+1) / 2;

if (arr[m] <= target) l = mid + 1;

else r = m - 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 l, int r, int target) {

while (l < r) {

int mid = l + (r-l+1) / 2;

if (arr[m] <= target) l = mid + 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 firstLarge(int l, int r, int target) {

while (l < r) {

int mid = l + (r-l+1) / 2;

if (arr[m] < target) l = mid;

else r = m - 1;

}

if (arr[l] < target) return l;

return -1;

}

结构体按照字符串排序：

struct word{

char eng[11];

char forei[11];

}words[MAXN];

int cmp(struct word a, struct word b) {

return strcmp(a.forei, b.forei) < 0;

}

字符串操作：strcmp(a,b) a<b:-1 a==b:0 a>b:1

读操作:gets(s)读入字符串到s，读到换行为止，cin和scanf一样。

sscanf(s, "%s%s", words[k].eng, words[k].forei); 以字符串s为标准输入。

字符串哈希：ELFHash方法

#include <set>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <cstdio>

#include <cmath>

#define INF 2e8

#define MAXN 100010

using namespace std;

struct Hash{

char eng[11];

char forei[11];

bool used;

Hash\* next;

Hash() {

used=false;

next=NULL;

}

Hash(char \*f, char \*e) {

strcpy(forei, f);

strcpy(eng, e);

used = false;

next = NULL;

}

}words[MAXN];

int ELFHash(char \*key) {

unsigned long h = 0, x = 0;

while(\*key) {

h = (h<<4)+(\*key++);

if ((x = h&0xf0000000L) != 0) {

h ^= (x>>24);

h &= ~x;

}

}

return h%MAXN;

}

int main()

{

char en[11], fn[11];

char s[25];

struct Hash\* p;

while (gets(s)) {

if (s[0] == '\0') break;

sscanf(s, "%s %s", en, fn);

int h = ELFHash(fn);

if (!words[h].used) {

words[h].used = true;

strcpy(words[h].eng, en);

strcpy(words[h].forei, fn);

}

else{

p = &words[h];

while(p->next != NULL) p = p->next;

p->next = new Hash(fn, en);

}

}

while (gets(s)) {

int h = ELFHash(s);

if (!words[h].used) printf("eh\n");

else{

p = &words[h];

while (p != NULL) {

if (!strcmp(s, p->forei)){

printf("%s\n", p->eng);

break;

}

else {

p = p->next;

}

}

if(p==NULL) printf("eh\n");

}

}

return 0;

}