**2015年天勤考研机试模拟赛解题报告**

A: 判断三角形

这题可以用map建立单词与数字之间的映射，也可以用if...else来写。唯一可能成为坑的是前几个单词可能是zero。

代码：http://paste.ubuntu.com/10616118/

#include <cstdio>

#include <cstring>

int strToNum(char str[]) {

if(strcmp(str, "zero") == 0) return 0;

if(strcmp(str, "one") == 0) return 1;

if(strcmp(str, "two") == 0) return 2;

if(strcmp(str, "three") == 0) return 3;

if(strcmp(str, "four") == 0) return 4;

if(strcmp(str, "five") == 0) return 5;

if(strcmp(str, "six") == 0) return 6;

if(strcmp(str, "seven") == 0) return 7;

if(strcmp(str, "eight") == 0) return 8;

if(strcmp(str, "nine") == 0) return 9;

}

int main() {

int n, len[3] = {0};

char c, str[10];

for(int i = 0; i < 3; i++) {

scanf("%d", &n);

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

scanf("%s", str);

len[i] = len[i] \* 10 + strToNum(str);

}

}

if(len[0] + len[1] > len[2] &&

len[0] + len[2] > len[1] &&

len[1] + len[2] > len[0]) {

printf("YES\n");

} else printf("NO\n");

return 0;

}

B: 删除结点

此题为15年408数据结构大题第一题（算法设计题），仿照PAT出题风格而出，所有的坑都是仿照PAT的题目而设计。坑点有：

1. 虽然结点的Address地址不会是-1，但是第一个结点的地址可能是-1，由此可能导致部分代码输出超限或者运行超时或者运行错误。（这是PAT中题目的坑点之一）
2. 可能有不在单链表上的结点，也就是无效结点。这可能导致答案错误。（这也是PAT中题目的坑点之一）

代码：http://paste.ubuntu.com/10616121/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 100005;

const int TABLE = 1000010;

struct Node { //定义静态链表（第一步）

int address, data, next;

int order; //结点在链表上的序号，无效结点记为maxn

}node[maxn];

bool isExist[TABLE] = {false};

bool cmp(Node a, Node b) {

return a.order < b.order;

}

int main() {

memset(isExist, false, sizeof(isExist));

for(int i = 0; i < maxn; i++) { //初始化（第二步）

node[i].order = maxn;

}

int n, begin, address;

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

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

scanf("%d", &address);

scanf("%d%d", &node[address].data, &node[address].next);

node[address].address = address;

}

int count = 0, p = begin;

while(p != -1) { //枚举链表，计数有效结点个数（第三步）

if(!isExist[abs(node[p].data)]){

isExist[abs(node[p].data)] = true;

node[p].order = count++;

}

p = node[p].next;

}

if(count == 0) { //特判，新链表中没有结点时输出0 -1

printf("0 -1\n");

} else {

sort(node, node + maxn, cmp); //筛选有效结点，并按结点顺序排序（第四步）

//输出结果（第五步）

printf("%d %05d\n", count, node[0].address); //防止-1被%05d化，因此提前判断

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

if (i != count - 1) {

printf("%05d %d %05d\n", node[i].address, node[i].data, node[i+1].address);

} else {

printf("%05d %d -1\n", node[i].address, node[i].data);

}

}

}

return 0;

}

C: 收集数码晶体

此题为15年408数据结构大题第二题。本题其实是求这样一个问题：求从起点S到终点T的长度为L的路径条数，而这恰好就是邻接矩阵的L次幂。注意在求幂时也要时刻模上1000000007，否则可能爆long long使结果变成负数。

本题用DFS/BFS会超时，复杂度爆棚了。

代码：http://paste.ubuntu.com/10616125/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 60;

const long long MOD = 1000000007;

long long G[maxn][maxn], ans[maxn][maxn], temp[maxn][maxn];

void MatrixPow(int n, int L) {

for(int times = 0; times < L; times++) {

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

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

temp[i][j] = 0;

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

temp[i][j] = (temp[i][j] + (ans[i][k] \* G[k][j] % MOD + MOD)) % MOD;

}

}

}

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

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

ans[i][j] = temp[i][j];

}

}

}

}

void init() {

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

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

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

ans[i][i] = 1;

}

}

int main() {

init();

int n, m, L, u, v;

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

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

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

G[u][v] = 1;

}

MatrixPow(n, L);

int k;

scanf("%d", &k);

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

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

printf("%lld\n", (ans[u][v] % MOD + MOD) % MOD);

}

return 0;

}

D: 还原二叉树

做多了先序+中序、后序+中序，来个层序+中序就不会了吗= =

在先序+中序中，先序的第一个结点是根结点，然后用中序区分出左右子树。到这里为止，层序+中序也是一样的，层序的第一个结点是根结点，然后用中序区分出左右子树。

不同在于后面的步骤。

由于先序中是按照“根结点-左子树所有结点-右子树所有结点”的顺序的，因此可以直接对左子树跟右子树进行递归，但是层序不行，层序的左右子树结点在层序序列中可能是分散的。因此必须想办法把左子树所有结点跟右子树所有结点都找出来。

于是可以开两个vector，一个存左子树所有结点的层序，一个存右子树所有结点的层序。然后遍历当前层序序列的所有结点，根据中序序列判断其属于左子树还是右子树，如果是左子树就push\_back到左子树的vector，否则就push\_back到右子树的vector。

这样就可以继续递归了，递归参数中用vector表示层序，而中序可以仍然用数组下标。

代码：http://paste.ubuntu.com/10616131/

#include <cstdio>

#include <cstring>

#include <vector>

using namespace std;

const int maxn = 50;

struct node {

int data;

node\* lchild;

node\* rchild;

};

int in[maxn]; //中序

//当前二叉树的层序序列为layer，中序序列区间为[inL, inR]

//create函数返回构建出的二叉树的根结点地址

node\* create(vector<int> layer, int inL, int inR) {

if(layer.size() == 0) {

return NULL;

}

node\* root = new node; //新建一个新的结点，用来存放当前二叉树的根结点

root->data = layer[0]; //新结点的数据域为根结点的值

int k;

for(k = inL; k <= inR; k++) {

if(in[k] == layer[0]) {

break;

}

}

vector<int> layerLeft;

vector<int> layerRight;

for(int i = 1; i < layer.size(); i++) {

bool isLeft = false;

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

if(layer[i] == in[j]) {

isLeft = true;

break;

}

}

if(isLeft) {

layerLeft.push\_back(layer[i]);

} else {

layerRight.push\_back(layer[i]);

}

}

root->lchild = create(layerLeft, inL, k - 1);

root->rchild = create(layerRight, k + 1, inR);

return root; //返回根结点地址

}

int num;

void preOrder(node\* root,int n) {

if(root == NULL) {

return;

}

printf("%d", root->data);

num++;

if(num < n) printf(" ");

else printf("\n");

preOrder(root->lchild, n);

preOrder(root->rchild, n);

}

void postOrder(node\* root,int n) {

if(root == NULL) {

return;

}

postOrder(root->lchild, n);

postOrder(root->rchild, n);

printf("%d", root->data);

num++;

if(num < n) printf(" ");

else printf("\n");

}

int main() {

vector<int> layer;

int n, temp;

scanf("%d", &n);

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

scanf("%d", &temp);

layer.push\_back(temp);

}

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

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

}

node\* root = create(layer, 0, n - 1); //建树

num = 0;

preOrder(root, n);

num = 0;

postOrder(root, n);

return 0;

}

E: 调查黑暗气息

本题的代码量相对大一些，但是思路还是很简单的。先用BFS计算各个顶点需要增加的辐能，然后用Dijkstra求出所有最短路径，接着用DFS枚举所有最短路径，求出各数据最优值。

容易坑到的地方：

1. 增加的辐能是要向上取整的。
2. 图不连通没关系，但是起点到终点一定要连通。

代码：http://paste.ubuntu.com/10616137/

#include <cstdio>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

const int MAXV = 510; //最大顶点数

const int INF = 1000000000; //无穷大

//n为顶点数，m为边数，G为邻接矩阵，weight为点权

//d[]记录最短距离，numPath记录最短路径条数，maxModW记录最大模，minHalfW记录最小后半段点权之和

int n, m, L, K, S, T, G[MAXV][MAXV], weight[MAXV];

int d[MAXV], numPath = 0, maxModW = -1, minHalfW = INF;

bool vis[MAXV] = {false}; //vis[i]==true表示顶点i已访问，初始均为false

bool inq[MAXV] = {false};

vector<int> pre[MAXV]; //前驱

vector<int> tempPath, path; //临时路径、最优路径

void init() {

fill(G[0], G[0] + MAXV \* MAXV, INF); //初始化图G

numPath = 0;

maxModW = -1;

minHalfW = INF;

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

memset(inq, false, sizeof(inq));

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

pre[i].clear();

}

tempPath.clear();

path.clear();

}

struct Node {

int id; //结点编号

int layer; //结点层号

};

void BFS() {

queue<Node> q;

Node start;

start.id = S;

start.layer = 0;

q.push(start);

inq[start.id] = true;

while(!q.empty()) {

Node topNode = q.front();

q.pop();

int u = topNode.id;

if(topNode.layer < L) {

weight[u] += (int)(ceil(weight[u] \* 1.0 \* (L - topNode.layer) / L) + 0.5);

}

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

if(!inq[v] && G[u][v] != INF) {

Node next;

next.id = v;

next.layer = topNode.layer + 1;

q.push(next);

inq[next.id] = true;

}

}

}

}

void Dijkstra() { //s为起点

fill(d, d + MAXV, INF);

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

pre[i].push\_back(i); //初始化pre数组

}

d[S] = 0;

for(int i = 0; i < n; i++) { //循环n次

int u = -1, MIN = INF; //u使d[u]最小，MIN存放该最小的d[u]

for(int j = 0; j < n; j++) { //找到未访问的顶点中d[]最小的

if(vis[j] == false && d[j] < MIN) {

u = j;

MIN = d[j];

}

}

vis[u] = true; //标记u为已访问

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

//如果v未访问 && u能到达v

if(vis[v] == false && G[u][v] != INF) {

if(d[u] + G[u][v] < d[v]) { //以u为中介点使d[v]更小

d[v] = d[u] + G[u][v]; //优化d[v]

pre[v].clear(); //清空pre[v]

pre[v].push\_back(u); //u为v的前驱

} else if(d[u] + G[u][v] == d[v]) { //找到相同长度的路径

pre[v].push\_back(u); //u为v的前驱之一

}

}

}

}

}

void DFS(int v) { //v为当前结点

if(v == S) { //递归边界，到达叶子结点（路径起点）

tempPath.push\_back(v);

numPath++;

int tempW = 0, tempHalfW = 0; //临时路径tempPath的点权之和

for(int i = 0; i < tempPath.size() / 2; i++) { //tempPath中结点为倒序

int id = tempPath[i];

tempW += weight[id];

tempHalfW += weight[id];

}

for(int i = tempPath.size() / 2; i < tempPath.size(); i++) {

int id = tempPath[i];

tempW += weight[id];

}

if(tempW % K > maxModW) {

maxModW = tempW % K;

minHalfW = tempHalfW;

path = tempPath;

} else if(tempW % K == maxModW && tempHalfW < minHalfW) {

minHalfW = tempHalfW;

path = tempPath;

}

tempPath.pop\_back();

return;

}

tempPath.push\_back(v);

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

DFS(pre[v][i]);

}

tempPath.pop\_back();

}

int main() {

init();

scanf("%d%d%d%d%d%d", &n, &m, &L, &K, &S, &T);

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

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

}

int u, v, w;

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

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

G[u][v] = G[v][u] = w;

}

BFS();

if(!inq[T]) {

printf("-1\n");

return 0;

}

Dijkstra(); //Dijkstra算法入口

DFS(T); //获取最优路径

printf("%d %d %d %d\n", numPath, d[T], maxModW, minHalfW);

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

printf("%d", path[i]); //倒着输出路径上的结点

if(i > 0) printf("->");

else printf("\n");

}

return 0;

}