/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **Warehouse**: Find nearest warehouse based on the position to {0, 0}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

class WareHouse {

public:

double distance;

pair<int, int> point;

WareHouse(int d, pair<int, int> p) : distance(d), point(p) {}

};

class MyComparator {

public:

int operator() (WareHouse\* &p1, WareHouse\* &p2) {

return p1->distance < p2->distance;

}

};

class Solution {

public:

vector<pair<int, int>> getNearest(vector<pair<int, int>> nodes, int k) {

vector<pair<int, int>> result;

priority\_queue<WareHouse\*, vector<WareHouse\*>, MyComparator> q;

for (auto node : nodes) {

double d = sqrt(pow(node.first, 2) + pow(node.second, 2));

if (q.size() >= k) {

if (!q.empty() && q.top()->distance > d) {

q.pop();

q.push(new WareHouse(d, node));

}

} else {

q.push(new WareHouse(d, node));

}

}

while (!q.empty()) {

auto t = q.top();

q.pop();

result.push\_back(t->point);

}

return result;

}

};

int main(void) {

Solution \*s = new Solution();

vector<pair<int, int>> nodes = {

{3, 2}, {2, 3}, {4, 5}, {5, 6}, {8, 9}, {1, 2}, {10, 11}

};

vector<pair<int, int>> result = s->getNearest(nodes, 3);

for (auto node : result) {

cout << node.first << ", " << node.second << endl;

}

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Check parenthesis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <string>

using namespace std;

class Solution {

public:

bool checkParenthesis(string parenthesis) {

stack<char> s;

for (char c : parenthesis) {

if (c == '(') {

s.push(c);

} else if (!s.empty()) {

s.pop();

} else {

return false;

}

}

return s.empty();

}

};

int main(void) {

Solution \*s = new Solution();

string s0 = "";

string s1 = "(((())))";

string s2 = "()(())()";

string s3 = "((())";

string s4 = ")(())";

string s5 = "(()))";

cout << s0 << ":" << s->checkParenthesis(s0) << endl;

cout << s1 << ":" << s->checkParenthesis(s1) << endl;

cout << s2 << ":" << s->checkParenthesis(s2) << endl;

cout << s3 << ":" << s->checkParenthesis(s3) << endl;

cout << s4 << ":" << s->checkParenthesis(s4) << endl;

cout << s5 << ":" << s->checkParenthesis(s5) << endl;

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Find top K ranking related movies except node itself

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <unordered\_map>

#include <queue>

using namespace std;

class Movie {

private:

int id;

float rate;

vector<Movie\*> similarMovies;

public:

int getID() {

return id;

}

float getRate() {

return rate;

}

vector<Movie\*> getSimilar() {

return similarMovies;

}

void setSimilar(vector<Movie\*> sms) {

similarMovies = sms;

}

Movie(int id, float rate) {

this->id = id;

this->rate = rate;

}

};

class MyComparator {

public:

int operator() (Movie\* &p1, Movie\* &p2) {

return p1->getRate() > p2->getRate();

}

};

class Solution {

public:

vector<Movie\*> find(Movie& movie, int N) {

priority\_queue<Movie\*, vector<Movie\*>, MyComparator> q;

unordered\_map<int, bool> visited;

vector<Movie\*> res;

visited[movie.getID()] = true;

for (auto nextMovie : movie.getSimilar()) {

if (visited[nextMovie->getID()]) {

continue;

}

visited[nextMovie->getID()] = true;

if (q.size() == N) {

Movie \*m = q.top();

if (m->getRate() < nextMovie->getRate()) {

q.pop();

q.push(nextMovie);

}

} else {

q.push(nextMovie);

}

}

while (!q.empty()) {

res.push\_back(q.top());

q.pop();

}

return res;

}

};

int main(void) {

Solution \*s = new Solution();

Movie\* m1 = new Movie(1, 1);

Movie\* m2 = new Movie(2, 2);

Movie\* m3 = new Movie(3, 3);

Movie\* m4 = new Movie(4, 4);

Movie\* m5 = new Movie(5, 5);

Movie\* m6 = new Movie(6, 6);

Movie\* m7 = new Movie(7, 7);

m1->setSimilar({m2, m3, m4, m6});

m2->setSimilar({m1, m3, m4, m6});

m3->setSimilar({m1, m2, m4, m6});

m4->setSimilar({m1, m2, m3, m6});

m6->setSimilar({m1, m2, m3, m4});

vector<Movie\*> res = s->find(\*m6, 2);

cout << "result: ";

for (auto m : res) {

cout << m->getID() << ",";

}

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Given two list<String, String>, one is the employee and favorate food type,

\* another is food type and food. Such as:

\* list1:

\* Zhang san, Chinese food

\* Li si, American food

\* Wang wu, Japanese food

\* Zhao liu, \*

\* list2:

\* Chinese food, Yu xiang rou si

\* Chinese food, Shui zhu yu

\* American food, Hamburg

\* Output:

\* Zhang san, Yu xiang rou si

\* Zhang san, Shui zhu yu

\* Li si, Hamburg

\* Zhao liu, Yu xiang rou si

\* Zhao liu, Shui zhu yu

\* Zhao liu, Hamburg

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

#include <vector>

#include <unordered\_set>

#include <unordered\_map>

using namespace std;

class Solution {

public:

vector<pair<string, string>> getMenu(vector<pair<string, string>> list1,

vector<pair<string, string>> list2) {

vector<pair<string, string>> res;

unordered\_map<string, unordered\_set<string>> m;

for (auto food : list2) {

m[food.first].insert(food.second);

}

for (auto cf : list1) {

if (cf.second == "\*") {

for (auto a : m) {

for (string f : a.second) {

res.push\_back({cf.first, f});

}

}

} else {

for (string f : m[cf.second]) {

res.push\_back({cf.first, f});

}

}

}

return res;

}

};

int main(void) {

Solution \*s = new Solution();

vector<pair<string, string>> list1 = {

{"Zhang san", "Chinese food"},

{"Li si", "American food"},

{"Wang wu", "Japanese food"},

{"Zhao liu", "\*"}

};

vector<pair<string, string>> list2 = {

{"Chinese food", "Yu xiang rou si"},

{"Chinese food", "Shui zhu yu"},

{"American food", "Hamburg"}

};

vector<pair<string, string>> res = s->getMenu(list1, list2);

for (auto r : res) {

cout << "{" << r.first << "," << r.second << "}, ";

}

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Check whether mouse can reach the cheese from point {0,0}

\* 0 is wall, 1 is road, 9 is cheese

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

class Solution {

public:

bool mazeCheck(vector<vector<int>> maze, pair<int, int> start) {

if (maze.size() == 0 || maze[0].size() == 0) {

return false;

}

if (maze[start.first][start.second] == 9) {

return true;

}

int m = maze.size();

int n = maze[0].size();

vector<pair<int, int>> dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

vector<vector<int>> visited(m, vector<int>(n, 0));

queue<pair<int, int>> curQueue;

curQueue.push(start);

visited[start.first][start.second] = 1;

while (!curQueue.empty()) {

queue<pair<int, int>> nextQueue;

while (!curQueue.empty()) {

pair<int, int> point = curQueue.front();

curQueue.pop();

for (auto dir : dirs) {

int x = dir.first + point.first;

int y = dir.second + point.second;

if (x >= 0 && x < m && y >= 0 && y < n && visited[x][y] == 0) {

if (maze[x][y] == 9) {

return true;

} else if (maze[x][y] == 1) {

nextQueue.push({x, y});

visited[x][y] = 1;

}

}

}

}

curQueue = nextQueue;

}

return false;

}

};

int main(void) {

Solution \*s = new Solution();

vector<vector<int>> maze1 = {

{1,0,0,0,0},

{1,1,1,1,1},

{1,0,0,0,0},

{0,0,9,0,0}

};

vector<vector<int>> maze2 = {

{1,0,0,0,0},

{1,1,1,1,1},

{1,0,0,0,1},

{0,0,9,1,1}

};

vector<vector<int>> maze3 = {

{1,1,1,1},

{1,0,0,0},

{1,9,0,0}

};

vector<vector<int>> maze4 = {

{9}

};

vector<vector<int>> maze5 = {

{1,1,1,1,1,1},

{1,1,1,1,0,0},

{0,0,1,0,0,0},

{1,1,1,1,1,1},

{1,0,0,0,1,0},

{1,1,1,0,9,0}

};

cout << "result: " << s->mazeCheck(maze5, {0, 0});

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Given an array of number, calculate the distance of BST of this array

\* Such as [5,6,3,1,2,4]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

using namespace std;

struct BstNode {

int val;

BstNode \*left;

BstNode \*right;

BstNode(int v): val(v), left(nullptr), right(nullptr) {}

};

class Solution {

private:

BstNode\* createBst(vector<int> values) {

BstNode \*root = nullptr;

if (values.size() > 0) {

root = new BstNode(values[0]);

}

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

addValue(root, values[i]);

}

return root;

}

void addValue(BstNode\* node, int value) {

if (node->val > value) {

if (node->left == nullptr) {

node->left = new BstNode(value);

return;

} else {

addValue(node->left, value);

}

} else {

if (node->right == nullptr) {

node->right = new BstNode(value);

return;

} else {

addValue(node->right, value);

}

}

}

BstNode\* findLCA(BstNode \*node, int node1, int node2) {

if (node->val >= node1 && node->val <= node2) {

return node;

} else if (node->val > node1 && node->val > node2) {

return findLCA(node->left, node1, node2);

} else {

return findLCA(node->right, node1, node2);

}

}

int calculateLen(BstNode \*node, int value) {

if (node == nullptr) {

return -1;

}

int res = 0;

if (node->val > value) {

res = calculateLen(node->left, value);

} else if(node->val < value) {

res = calculateLen(node->right, value);

} else {

res = 0;

}

return res == -1 ? res : res + 1;

}

public:

int bstDistance(vector<int> values, int node1, int node2) {

BstNode \*root = createBst(values);

if (node1 > node2) {

node1 += node2;

node2 = node1 - node2;

node1 = node1 - node2;

}

BstNode \*lca = findLCA(root, node1, node2);

int leftLen = calculateLen(lca, node1);

int rightLen = calculateLen(lca, node2);

if (leftLen == -1 || rightLen == -1) {

return -1;

} else {

return leftLen + rightLen - 2;

}

}

};

int main(void) {

Solution \*s = new Solution();

vector<int> values = {5,6,3,1,2,4};

cout << "result: " << s->bstDistance(values, 5, 5) << endl;

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Book association based on book pair <bookA, bookB>

\* Find bigest book collection

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <map>

#include <string>

using namespace std;

class Solution {

private:

void unionFind(map<string, int> &m, pair<string, string> as) {

int g1 = m[as.first];

int g2 = m[as.second];

for (auto a : m) {

if (a.second == g2) {

m[a.first] = g1;

}

}

}

public:

vector<string> findBigestBookCollection(vector<pair<string, string>> associations) {

vector<string> res;

map<string, int> m;

int n = 0;

for (auto as : associations) {

if (m.find(as.first) == m.end()) {

n++;

m[as.first] = n;

}

if (m.find(as.second) == m.end()) {

n++;

m[as.second] = n;

}

}

for (auto as : associations) {

unionFind(m, as);

}

int maxNum = 0;

for (auto a : m) {

if (a.second == -1) {

continue;

}

vector<string> temp;

for (auto aa : m) {

if (aa.second == a.second) {

temp.push\_back(aa.first);

aa.second = -1;

}

}

if (temp.size() > res.size()) {

res = temp;

}

}

return res;

}

};

int main(void) {

Solution\* s = new Solution();

vector<pair<string, string>> associations;

associations.push\_back({"D", "E"});

associations.push\_back({"E", "F"});

associations.push\_back({"A", "B"});

associations.push\_back({"A", "C"});

associations.push\_back({"G", "H"});

associations.push\_back({"G", "D"});

vector<string> res = s->findBigestBookCollection(associations);

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

cout << res[i] << ", ";

}

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Given a string array representing a throw ball blocks, each string is either a \* \* number, +, Z, X. Calculate total. If number, just add to total. If +, add last 2 \* scores to total.

\* If Z, remove last score from total. If X, double last score and add to toal.

\* Use 0 for any missing last score.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <vector>

#include <string>

using namespace std;

class Solution {

int getBaseballScore(vector<string> input) {

stack<int> s;

int res = 0;

for (string op : input) {

if (op[0] == 'z' && !s.empty()) {

s.pop();

} else if (op[0] == 'x' && !s.empty()) {

int a = s.top();

s.push(a \* 2);

} else if (op[0] == '+') {

int a = 0;

int b = 0;

int c = 0;

bool getB = false;

bool getA = false;

if (!s.empty()) {

b = s.top();

s.pop();

getB = true;

}

if (!s.empty()) {

a = s.top();

s.pop();

bool getA = true;

}

c = a + b;

if (getA) {

s.push(a);

}

if (getB) {

s.push(b);

}

s.push(c);

} else {

s.push(stoi(op));

}

}

while (!s.empty()) {

res += s.top();

s.pop();

}

return res;

}

};

int main(void) {

Solution \*s = new Solution();

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

using namespace std;

class Solution {

public:

bool isMatch(vector<vector<string>> codeList, vector<string> shoppingCart) {

int startIndex = 0;

for (auto codeLevel : codeList) {

bool isFind = false;

while (!isFind && startIndex + codeLevel.size() <= shoppingCart.size()) {

if (codeLevel[0] != shoppingCart[startIndex]) {

startIndex++;

continue;

}

int i = 0;

while (i < codeLevel.size() && codeLevel[i] == shoppingCart[startIndex+i]) {

i++;

}

if (i == codeLevel.size()) {

isFind = true;

startIndex = startIndex + codeLevel.size();

} else {

startIndex++;

}

}

if (!isFind) {

return false;

}

}

return true;

}

};

int main(void) {

Solution \*s = new Solution();

vector<vector<string>> codeList = {

{"apple", "apple"},

{"orange", "banana", "orange"}

};

vector<string> shoppingCart = {

"Orange", "apple", "apple", "lychee", "orange", "banana", "orange"

};

cout << "result: " << s->isMatch(codeList, shoppingCart);

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Check whether the binary of number is palindrome.

\* Such as 11011 (27) is palindrome, 11 (3) is palindrome

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

class Solution {

public:

bool isAnagram(int num) {

int len = 0;

int tmp = num;

while (tmp / 2 > 0) {

len++;

tmp /= 2;

}

tmp = num;

while (tmp > 0) {

if ((tmp >> len) != (tmp & 1)) {

return false;

}

tmp &= (1 << len) - 1;

tmp /= 2;

len -= 2;

}

return true;

}

};

int main() {

Solution\* s = new Solution();

int num = 5;

cout << "The number " << num << " is " << s->isAnagram(num) << endl;

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 1 is road, 0 is water, and num > 1 is tree.

\* Cut tree from low to high

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <queue>

#include <map>

using namespace std;

class Solution {

private:

vector<pair<int, int>> dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

public:

int flatFields(int numRows, int numColumns, vector<vector<int>> fields) {

vector<int> nums;

if (fields.size() == 0 || fields[0].size() == 0 || fields[0][0] == 0) {

return -1;

}

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

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

if (fields[i][j] > 1) {

nums.push\_back(fields[i][j]);

}

}

}

sort(nums.begin(), nums.end());

int result = 0;

int startPosX = numRows - 1;

int startPosY = numColumns - 1;

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

int steps = 0;

queue<pair<int, int>> curQueue;

vector<vector<bool>> visited(numRows, vector<bool>(numColumns, false));

curQueue.push({startPosX, startPosY});

bool isFind = false;

while (!curQueue.empty() && !isFind) {

queue<pair<int, int>> nextQueue;

while (!curQueue.empty()) {

auto t = curQueue.front();

curQueue.pop();

if (fields[t.first][t.second] == nums[i]) {

isFind =true;

result += steps;

startPosX = t.first;

startPosY = t.second;

fields[t.first][t.second] = 1;

break;

}

if (fields[t.first][t.second] > 1) {

continue;

}

for (auto dir : dirs) {

int x = t.first + dir.first;

int y = t.second + dir.second;

if (x < 0 || x >= numRows || y < 0 || y >= numColumns || visited[x][y]) {

continue;

}

visited[x][y] = true;

nextQueue.push({x, y});

}

}

steps++;

curQueue = nextQueue;

}

if (!isFind) {

return -1;

}

}

return result;

}

};

int main(void) {

Solution \*s = new Solution();

/\*vector<vector<int>> fields = {

{1, 3, 0, 2},

{1, 1, 3, 1}

};\*/

vector<vector<int>> fields = {

{1, 0},

{3, 2}

};

cout << s->flatFields(2, 4, fields) << endl;

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Given the association of the managers and employee,

\* find the nearest manager of them.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <string>

#include <queue>

#include <unordered\_map>

using namespace std;

struct TreeNode {

string val;

vector<TreeNode\*> child;

TreeNode(string v) : val(v), child({}) {}

};

class Solution {

private:

TreeNode\* buildTree(vector<pair<string, string>> associations) {

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

return nullptr;

}

unordered\_map<string, TreeNode\*> m1;

unordered\_map<string, int> m2;

for (auto as : associations) {

if (m1.find(as.first) == m1.end()) {

m1[as.first] = new TreeNode(as.first);

m2[as.first] = 0;

}

if (m1.find(as.second) == m1.end()) {

m1[as.second] = new TreeNode(as.second);

m2[as.second] = 1;

} else {

m2[as.second]++;

}

m1[as.first]->child.push\_back(m1[as.second]);

}

for (auto a : m2) {

if (a.second == 0) {

return m1[a.first];

}

}

}

bool findNodePath(TreeNode\* node, vector<TreeNode\*> &path, string value) {

if (node == nullptr) {

return false;

}

if (node->val != value) {

for (TreeNode\* nn : node->child) {

path.push\_back(node);

if (findNodePath(nn, path, value)) {

return true;

}

path.pop\_back();

}

return false;

} else {

return true;

}

}

public:

string getNearestManager(vector<pair<string, string>> associations, string node1, string node2) {

TreeNode \*root = buildTree(associations);

//outputTree(root);

vector<TreeNode\*> left, right;

left.push\_back(root);

right.push\_back(root);

bool vl = findNodePath(root, left, node1);

bool vr = findNodePath(root, right, node2);

if (!vl || !vr) {

return "";

}

int i = 0;

for (; i < left.size() && i < right.size(); i++) {

if (left[i]->val != right[i]->val) {

return left[i-1]->val;

}

}

return i == left.size() || i == right.size() ? left[i-1]->val : "";

}

};

int main(void) {

Solution \*s = new Solution();

vector<pair<string, string>> associations = {

{"A", "B"},

{"B", "C"},

{"C", "D"},

{"D", "E"},

{"D", "F"},

{"C", "G"},

{"C", "H"},

{"B", "I"},

{"I", "J"},

{"J", "K"},

{"A", "L"},

{"A", "M"},

{"M", "N"}

};

cout << "result for A, B: " << s->getNearestManager(associations, "A", "B") << endl;

cout << "result for N, K: " << s->getNearestManager(associations, "N", "K") << endl;

cout << "result for J, K: " << s->getNearestManager(associations, "J", "K") << endl;

cout << "result for E, K: " << s->getNearestManager(associations, "E", "K") << endl;

cout << "result for Z, K: " << s->getNearestManager(associations, "Z", "K") << endl;

cout << "result for A, Z: " << s->getNearestManager(associations, "A", "Z") << endl;

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Input [itemA, itemB] means item A has association with item B

\* Input [itemB, itemC] means item B has association with item C

\* given an array find big group.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <map>

#include <vector>

#include <string>

using namespace std;

class Solution {

public:

vector<string> findBigestGroup(vector<pair<string, string>> associations) {

vector<string> res;

map<string, int> m;

int index = 1;

for (auto ass : associations) {

if (m.find(ass.first) != m.end()) {

if (m.find(ass.second) != m.end()) {

int t1 = m[ass.first];

int t2 = m[ass.second];

for (auto as : m) {

if (as.second == t2) {

m[as.first] = t1;

}

}

} else {

m[ass.second] = m[ass.first];

}

} else if (m.find(ass.second) != m.end()) {

m[ass.first] = m[ass.second];

} else {

m[ass.first] = m[ass.second] = index;

index++;

}

}

for (auto as : m) {

if (as.second == 0) {

continue;

}

int val = as.second;

vector<string> tmp;

for (auto a : m) {

if (a.second == val) {

tmp.push\_back(a.first);

a.second = 0;

}

}

if (tmp.size() > res.size()){

res = tmp;

}

}

return res;

}

};

int main(void) {

Solution\* s = new Solution();

vector<pair<string, string>> associations;

associations.push\_back({"D", "E"});

associations.push\_back({"E", "F"});

associations.push\_back({"A", "B"});

associations.push\_back({"A", "C"});

associations.push\_back({"G", "H"});

//associations.push\_back({"G", "D"});

vector<string> res = s->findBigestGroup(associations);

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

cout << res[i] << ", ";

}

delete s;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Check whether the binary of number is anagram.

\* Such as 11011 (27) is anagram, 11 (3) is anagram

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

class Solution {

public:

bool isAnagram(int num) {

int len = 0;

int tmp = num;

while (tmp / 2 > 0) {

len++;

tmp /= 2;

}

tmp = num;

while (tmp > 0) {

if ((tmp >> len) != (tmp & 1)) {

return false;

}

tmp &= (1 << len) - 1;

tmp /= 2;

len -= 2;

}

return true;

}

};

int main() {

Solution\* s = new Solution();

int num = 5;

cout << "The number " << num << " is " << s->isAnagram(num) << endl;

delete s;

return 0;

}