|  |
| --- |
| Университет итмо, кафедра вт |
| Лабораторная работа №4 по дисциплине «Алгоритмы и структуры данных» |
| «Алгоритмы на графах» |
| Группа Р3202 |
| **Выполнила: Орлова Кристина Александровна** |
| **Преподаватель: Косяков Михаил Сергеевич** |

|  |
| --- |
| *14.06.18* |

**№1080**

Решение данной задачи базируется на алгоритме dfs. При решении использовалась карта map<int, set<int>> countries, ключом в которой был номер страны, а значением был список соседей, которая фактически выступала за граф, а также карта map<int, int> country\_color, которая приводила соответствие цвета раскраски стране по ее номеру. Изначально карта country\_color инициализировалась -1, кроме первой страны, которой соответствовал цвет 0. Дальше происходила раскраска, то есть для каждой страны:  
1. Нужно было получить ее цвет.  
2. Раскрасить ее соседей в противоположный цвет в случае, если они еще не были раскрашены. Если они уже были раскрашены, но неправильно, то нужно установить флаг isBadColored. И в случае правильной раскраски, и в случае неправильной нужно было добавить соседа в trashList, чтобы затем убрать его из списка соседей этой страны (для того чтобы не возникало бесконечной рекурсии туда-обратно).  
3. Для каждого соседа сделать то же самое.  
  
Как только мы выйдем из этой рекурсии, нужно проверить флаг isBadColored. Если он установлен, выводим -1. Если он не установлен, то проверяем, что все страны раскрашены (то есть что до каждой страны можно было добраться). Если у какой-то из них стоит -1, выводим -1. Если все страны раскрашены, то выводим последовательно их цвета.

|  |
| --- |
| #include <iostream> |
|  | #include <map> |
|  | #include <set> |
|  | #include <vector> |
|  |  |
|  | using namespace std; |
|  |  |
|  | map<int, int> country\_color; |
|  | int n; |
|  | bool isBadColored = false; |
|  |  |
|  | void coloring(int currentCountryNumber, map<int, set<int>> &countries) { |
|  | auto it = countries.find(currentCountryNumber); |
|  | auto it\_neighbours = it->second.begin(); |
|  | vector<int> trashList; |
|  | int colorOfPaint; |
|  |  |
|  | if (country\_color.find(it->first)->second == 0) { |
|  | colorOfPaint = 1; |
|  | } else { |
|  | colorOfPaint = 0; |
|  | } |
|  |  |
|  | // раскраска |
|  | while (it\_neighbours != it->second.end()) { |
|  | int neighbourColor = country\_color.find(\*it\_neighbours)->second; |
|  |  |
|  | if (neighbourColor == -1) { |
|  | country\_color[\*it\_neighbours] = colorOfPaint; |
|  | } else { |
|  | if (neighbourColor != colorOfPaint) { |
|  | isBadColored = true; |
|  | } |
|  | trashList.push\_back(\*it\_neighbours); |
|  | } |
|  |  |
|  | it\_neighbours++; |
|  | } |
|  |  |
|  | // очистка списка соседей |
|  | for (int i : trashList) { |
|  | it->second.erase(i); |
|  | } |
|  |  |
|  | it\_neighbours = it->second.begin(); |
|  |  |
|  | // повторяем для соседей |
|  | while (it\_neighbours != it->second.end()) { |
|  | coloring(\*it\_neighbours, countries); |
|  | it\_neighbours++; |
|  | } |
|  | } |
|  |  |
|  | void addNeighbours(map<int, set<int>> &countries) { |
|  | auto it = countries.begin(); |
|  |  |
|  | while (it != countries.end()) { |
|  | int searchedNum = it->first; |
|  |  |
|  | if (searchedNum != 1) { |
|  | auto it\_deep = countries.begin(); |
|  |  |
|  | while (it\_deep != countries.end()) { |
|  | if (it\_deep != it) { |
|  | if (it\_deep->second.find(searchedNum) != it\_deep->second.end()) { |
|  | it->second.insert(it\_deep->first); |
|  | } |
|  | } |
|  |  |
|  | it\_deep++; |
|  | } |
|  | } |
|  | it++; |
|  | } |
|  | } |
|  |  |
|  | void checkColoring(string &sequence) { |
|  | auto it = country\_color.begin(); |
|  |  |
|  | while (it != country\_color.end()) { |
|  | if (it->second == -1) { |
|  | sequence = "-1"; |
|  | break; |
|  | } else { |
|  | sequence += to\_string(it->second); |
|  | } |
|  |  |
|  | it++; |
|  | } |
|  | } |
|  |  |
|  | void printAnswer(string &sequence) { |
|  | cout << sequence; |
|  | } |
|  |  |
|  | void getAnswer(map<int, set<int>> &countries) { |
|  | string sequence; |
|  | addNeighbours(countries); |
|  |  |
|  | country\_color[1] = 0; |
|  | coloring(1, countries); |
|  |  |
|  | if (isBadColored) { |
|  | sequence = "-1"; |
|  | } else { |
|  | checkColoring(sequence); |
|  | } |
|  |  |
|  | printAnswer(sequence); |
|  | } |
|  |  |
|  | int main() { |
|  | map<int, set<int>> countries; |
|  | cin >> n; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | set<int> neighbours; |
|  | int num; |
|  |  |
|  | cin >> num; |
|  |  |
|  | while (num != 0) { |
|  | neighbours.insert(num); |
|  |  |
|  | cin >> num; |
|  | } |
|  |  |
|  | countries[i + 1] = neighbours; |
|  | country\_color[i + 1] = -1; |
|  | } |
|  |  |
|  | getAnswer(countries); |
|  |  |
|  | return 0; |
|  | } |

**№1450**

Решение данной задачи базируется на алгоритме форда-беллмана. При решении данной задачи использовалась карта map<int, vector<apex>>, ключом в которой был номер вершины, а значением был список вершин, в которые от нее можно было дойти. Это карта фактически представляет собой граф. В структуре apex хранится номер вершины и прибыль, которую можно получить, дойдя от вершины-ключа до нее. Также использовалась карта apex\_gain, которая приводила соответствие суммарной прибыли, полученной на пути к этой вершине, номеру вершины. Изначально она инициализировалась -1, кроме стартовой страны, для которой прибыль 0.  
Производится n - 1 итераций, где n - число вершин. Из цикла можно выйти и раньше, если флаг isChanged установится в false. На каждой итерации для вершины:  
1. Считается ее прибыль по карте apex\_gain. Если она -1, то идем к другой вершине. Если нет, то для каждой вершины, к которой из нее можно дойти, смотрим прибыль от прохода, затем складываем ее с прибылью текущей вершины и получаем suggestedGain. Если она меньше прибыли, которая соответствует вершине по карте apex\_gain, то рассмотрим другую вершину-соседа. Если больше, то перезапишем на suggestedGain и поставим флаг isChanged.   
  
После всех итераций результатом будет значение из карты apex\_gain для вершины-финиша. Если оно -1, то выведем "No solution", иначе выведем это значение.

|  |
| --- |
| #include <iostream> |
|  | #include <vector> |
|  | #include <map> |
|  |  |
|  | using namespace std; |
|  |  |
|  | typedef struct { |
|  | int num; |
|  | long long gain; |
|  | } apex; |
|  |  |
|  | map<int, vector<apex>> apexes; |
|  | vector<long long> apex\_gain; |
|  | string answer = "No solution"; |
|  |  |
|  | void printAnswer() { |
|  | printf("%s", answer.c\_str()); |
|  | } |
|  |  |
|  | void getAnswer(int finish) { |
|  | long long result; |
|  | int n = apex\_gain.size(); |
|  | bool isChanged = true; |
|  | int iteration = 1; |
|  |  |
|  | while (iteration < n && isChanged) { |
|  | auto it = apexes.begin(); |
|  | isChanged = false; |
|  |  |
|  | while (it != apexes.end()) { |
|  | long long currentGain = apex\_gain[it->first - 1]; |
|  |  |
|  | if (currentGain != -1) { |
|  | for (apex nextApex: it->second) { |
|  | long long suggestedGain = currentGain + nextApex.gain; |
|  |  |
|  | if (apex\_gain[nextApex.num - 1] < suggestedGain) { |
|  | apex\_gain[nextApex.num - 1] = suggestedGain; |
|  | isChanged = true; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | it++; |
|  | } |
|  |  |
|  | iteration++; |
|  | } |
|  |  |
|  | result = apex\_gain[finish - 1]; |
|  |  |
|  | if (result != -1) { |
|  | answer = to\_string(result); |
|  | } |
|  |  |
|  | printAnswer(); |
|  | } |
|  |  |
|  | int main() { |
|  | int n, m; |
|  | int start, finish; |
|  |  |
|  | scanf("%d %d", &n, &m); |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | apex\_gain.push\_back(-1); |
|  | } |
|  |  |
|  | for (int i = 0; i < m; i++) { |
|  | int mainApex; |
|  | apex nextApex; |
|  |  |
|  | scanf("%d %d %lld", &mainApex, &nextApex.num, &nextApex.gain); |
|  |  |
|  | auto it = apexes.find(mainApex); |
|  |  |
|  | if (it != apexes.end()) { |
|  | it->second.push\_back(nextApex); |
|  | } else { |
|  | vector<apex> paths; |
|  | paths.push\_back(nextApex); |
|  | apexes[mainApex] = paths; |
|  | } |
|  | } |
|  |  |
|  | scanf("%d %d", &start, &finish); |
|  |  |
|  | apex\_gain[start - 1] = 0; |
|  | getAnswer(finish); |
|  |  |
|  | return 0; |
|  | } |

**№1160**

Решение данной задачи базируется на алгоритме Краскала. При решении данной задачи использовался массив ребер vector<connection> connections. Структура connection содержит номер первой вершины, второй вершины и вес ребра (длину кабеля), их соединяющего.   
Массив ребер сортируется по весам в порядке возрастания. Далее пригодится vector<int> apexes - массив графов, в которых лежат вершины. В начальный момент времени каждая вершина лежит в своем графе. До тех пор, пока у нас не останется всего 1 граф (мы это узнаем, так как в массиве графов для каждой вершины будет одинаковое значение), пройдемся по отсортированному массиву ребер: если первая вершина ребра лежит в одном графе, а вторая в другом, то положим вторую вершину в граф, где лежит первая, а также все остальные вершины, лежащие вместе со второй в одном графе. Как только получим 1 целый граф, выпишем все использованные ребра и вес последнего использованного.

|  |
| --- |
| #include <iostream> |
|  | #include <vector> |
|  | #include <algorithm> |
|  |  |
|  | using namespace std; |
|  |  |
|  | typedef struct { |
|  | int first; |
|  | int second; |
|  | int length; |
|  | } connection; |
|  |  |
|  | bool operator<(const connection &lhs, const connection &rhs) { |
|  | return lhs.length < rhs.length; |
|  | } |
|  |  |
|  | int n; |
|  | int m; |
|  | vector<connection> connections; |
|  |  |
|  | void printAnswer(int maxLength, int count, string &answer) { |
|  | cout << maxLength << endl << count << endl << answer; |
|  | } |
|  |  |
|  | void getAnswer() { |
|  | vector<int> apexes; |
|  | int i = 0; |
|  | int countOfTheSame = 1; |
|  | int maxLength = 0; |
|  | string answer; |
|  |  |
|  | for (int j = 0; j < n; j++) { |
|  | apexes.push\_back(j); |
|  | } |
|  |  |
|  | while (countOfTheSame != n) { |
|  | connection connection = connections[i]; |
|  | maxLength = connection.length; |
|  |  |
|  | if (apexes[connection.first - 1] != apexes[connection.second - 1]) { |
|  | int newSet = apexes[connection.first - 1]; |
|  | int oldSet = apexes[connection.second - 1]; |
|  |  |
|  | for (int j = 0; j < n; j++) { |
|  | if (apexes[j] == oldSet) { |
|  | apexes[j] = newSet; |
|  | } |
|  | } |
|  |  |
|  | countOfTheSame++; |
|  | } |
|  |  |
|  | answer += to\_string(connection.first) + " " + to\_string(connection.second) + "\n"; |
|  |  |
|  | i++; |
|  | } |
|  |  |
|  | printAnswer(maxLength, i, answer); |
|  | } |
|  |  |
|  | int main() { |
|  | cin >> n >> m; |
|  |  |
|  | for (int i = 0; i < m; i++) { |
|  | connection connection{}; |
|  | cin >> connection.first >> connection.second >> connection.length; |
|  |  |
|  | connections.push\_back(connection); |
|  | } |
|  |  |
|  | sort(connections.begin(), connections.end()); |
|  | getAnswer(); |
|  |  |
|  | return 0; |
|  | } |