**Министерство науки и высшего образования РФ  
ФГБОУ ВО  
Московский авиационный институт**

**(национальный исследовательский университет)**

Институт №3

Системы управления, информатики и электроэнергетики

Кафедра 304

**Отчет по лабораторной работе №1**

**по учебной дисциплине**

**«Структуры и алгоритмы обработки данных»**

**на тему:**

***«Поиск кратчайших путей в графе.***

***Построение остовного дерева графа.»***

*Группа: М30-209Б-19*

*Вариант №4*

*Выполнил:*

*Кузнецов И.И.*

*Приняли:*

*Ким Р. В.*

*Ведьманов И. С.*

*Москва 2021*

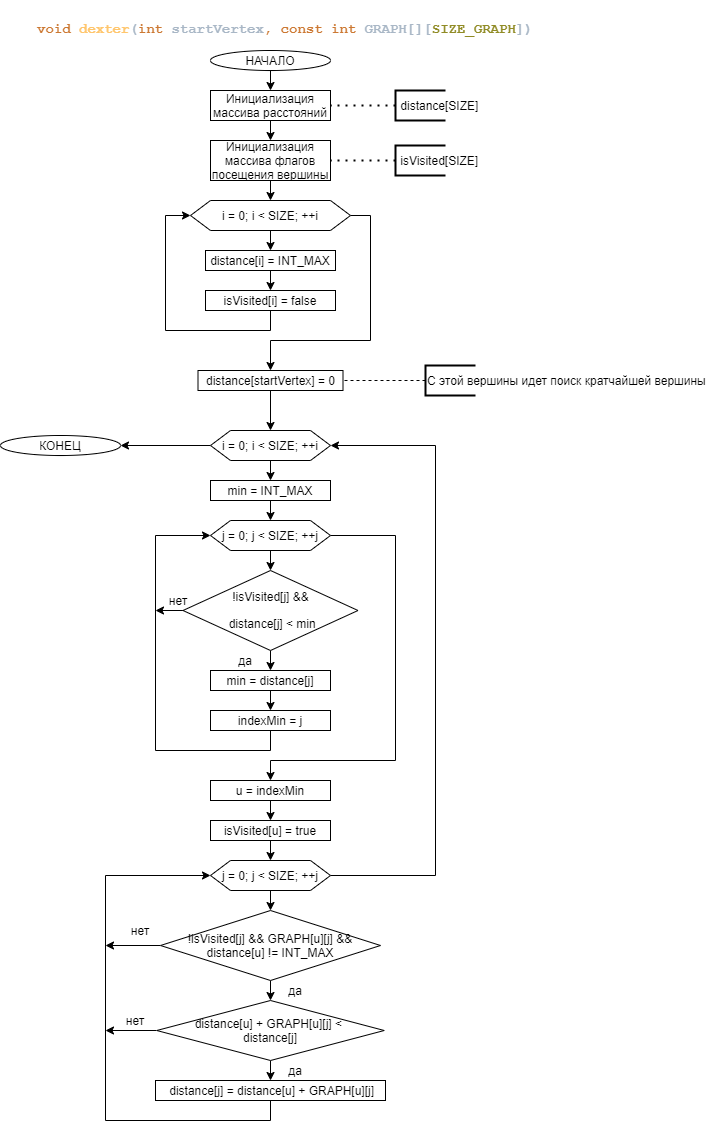
**Задание**  
Для созданного Вами взвешенного ориентированного графа, состоящего как минимум из 15 вершин, реализовать по вариантам:

1. Алгоритм поиска кратчайшего пути.
2. Сделав тот же самый граф неориентированным, построить его остовное дерево минимальной стоимости.

Вариант №4

|  |  |  |
| --- | --- | --- |
| № | Алгоритм, реализуемый в п.1 задания | Алгоритм, п.2 задания |
| 4 | Дейкстры | Крускала |

**Структурная схема**

****

**Текст программы**

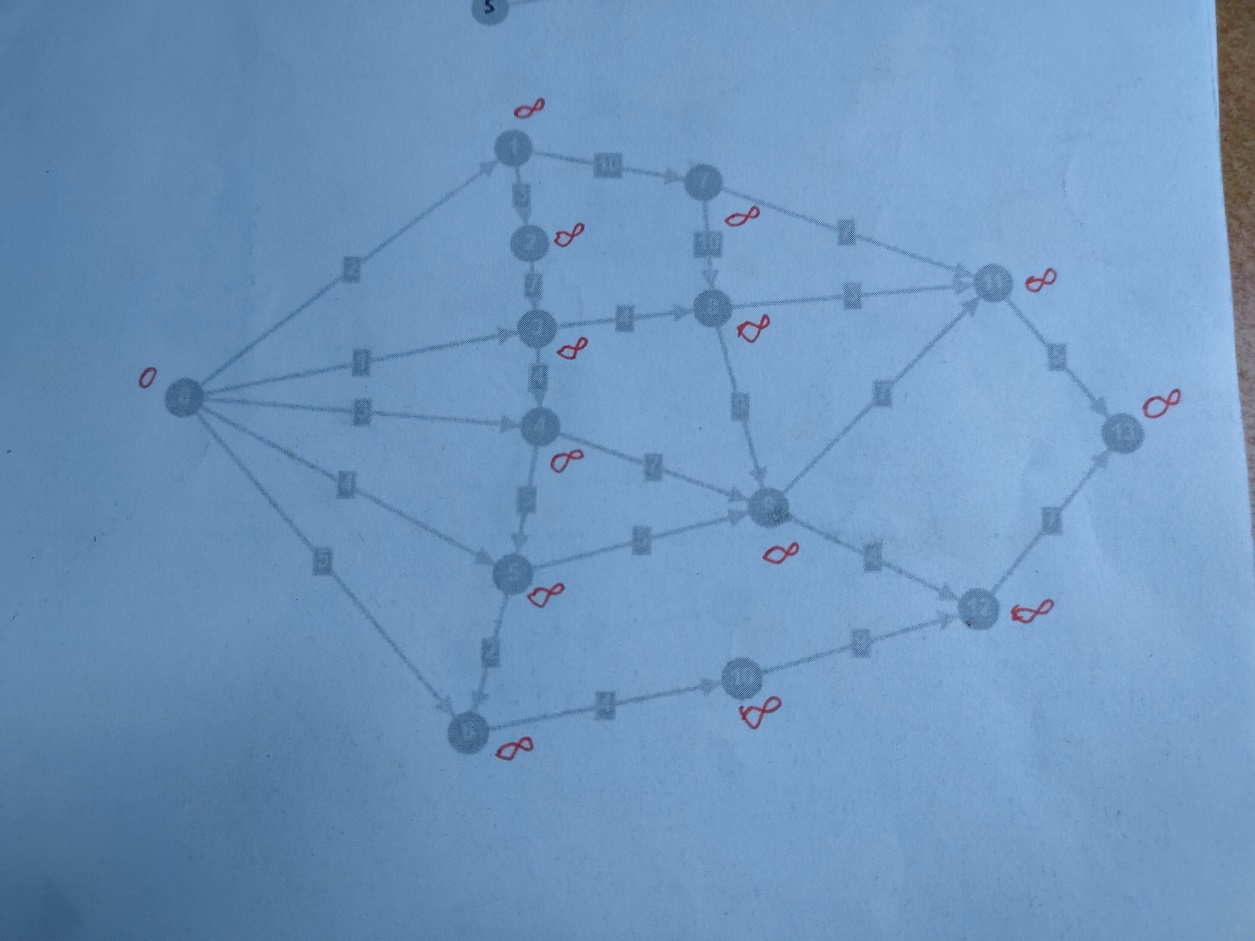
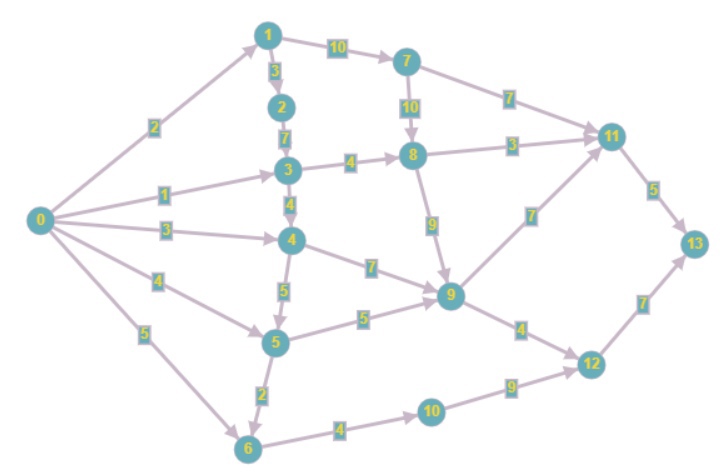
**void dexter(int startVertex, const int GRAPH[][SIZE\_GRAPH]) {  
 int \*distance = new int[SIZE\_GRAPH]; //Массив расстояний  
 bool \*isVisited = new bool[SIZE\_GRAPH]; //Массив флагов посещения вершины  
  
 for (int i = 0; i < SIZE\_GRAPH; ++i) {  
 distance[i] = INT\_MAX; //Начальное мин. расстояние до вершины  
 isVisited[i] = false; //Обнуляем флаги посещения вершины  
 }  
 distance[startVertex] = 0; //C этой вершины ищем кратчайшие пути  
  
 int indexMin;  
 int u;**

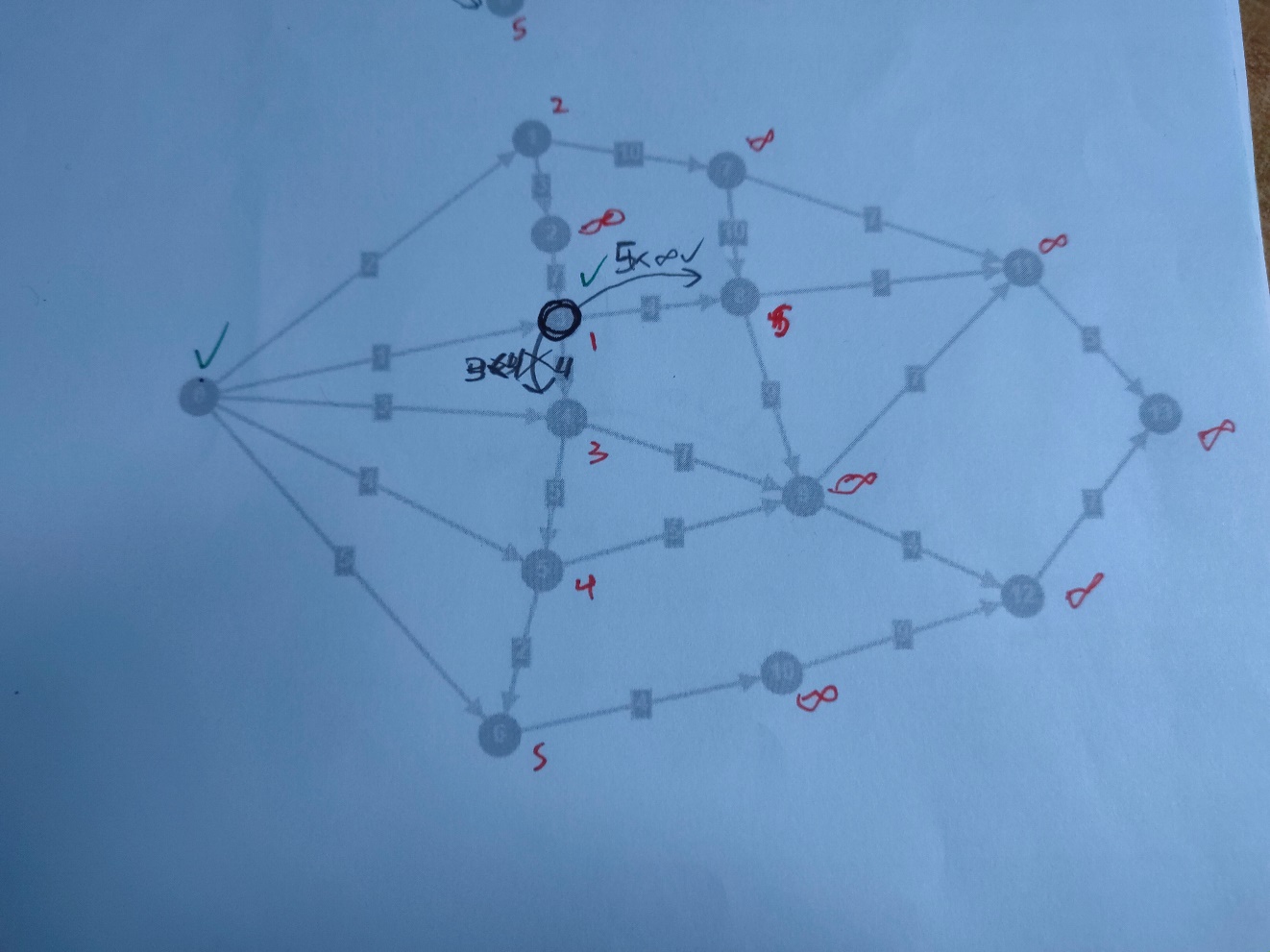
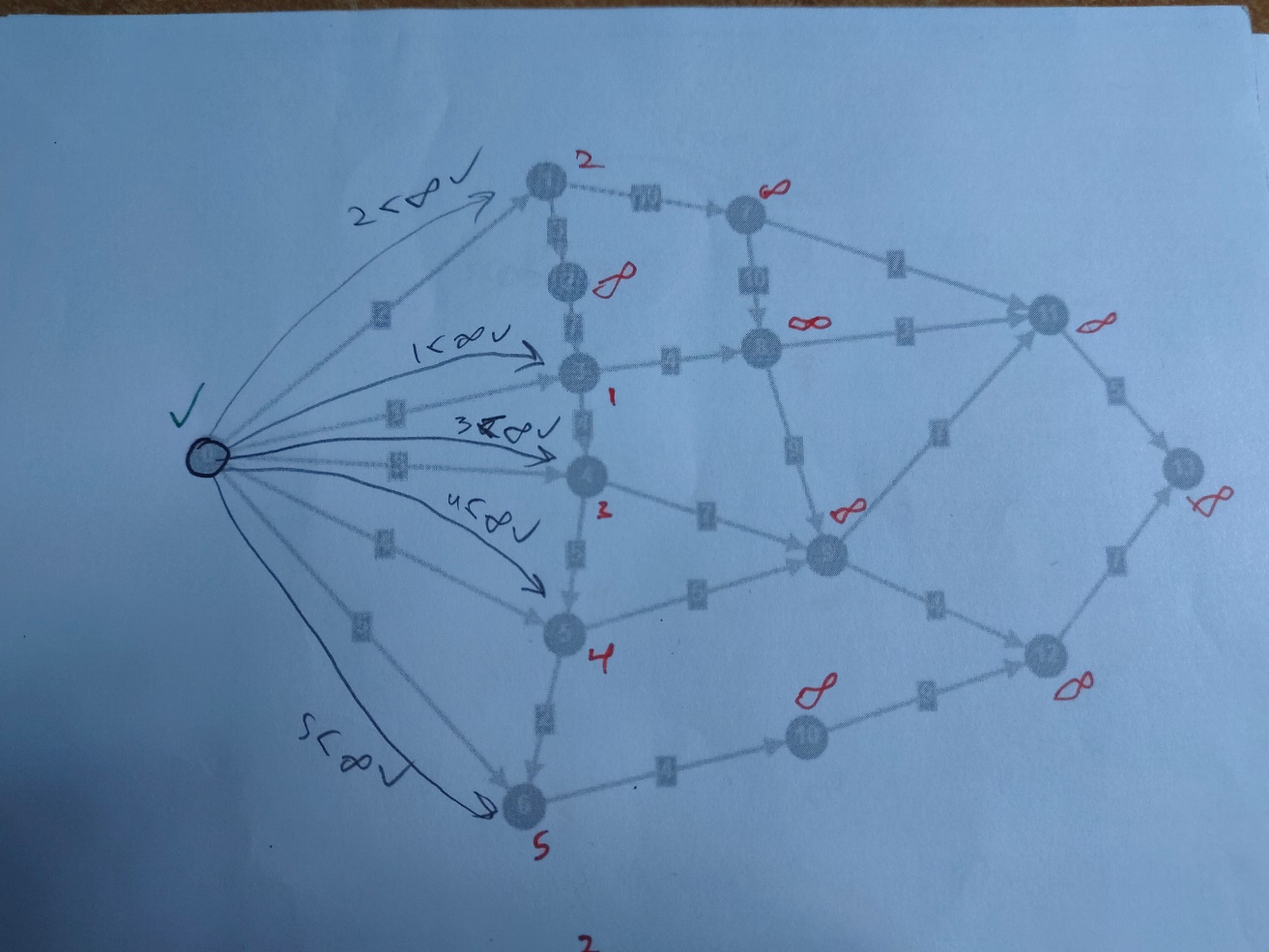
**for (int i = 0; i < SIZE\_GRAPH; ++i) {  
 int min = INT\_MAX;  
 for (int j = 0; j < SIZE\_GRAPH; ++j) { //ищем вершину с минимальным расстоянием  
 if (!isVisited[j] && distance[j] < min) {  
 min = distance[j];  
 indexMin = j;  
 }  
 }**

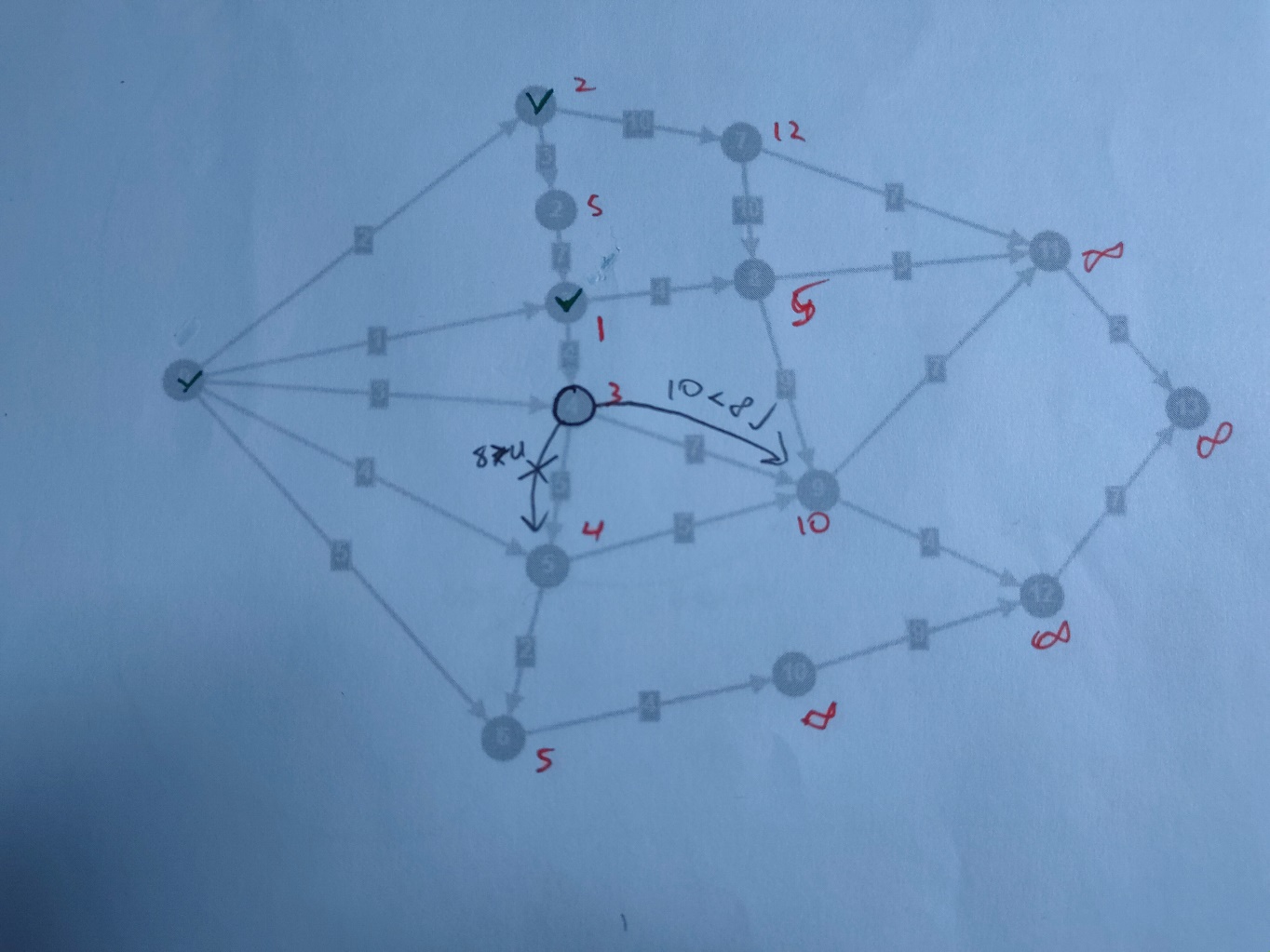
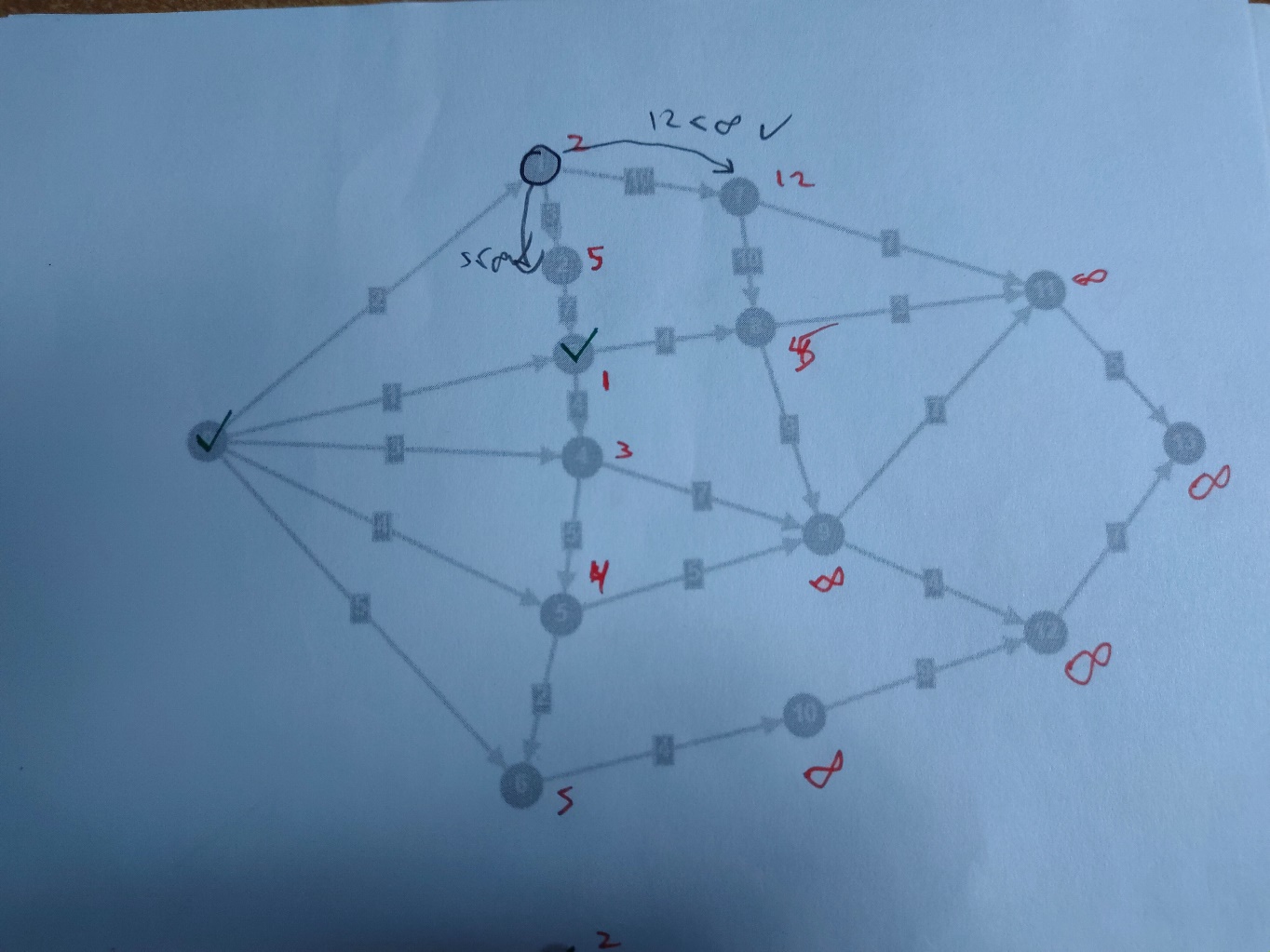
**u = indexMin;  
 isVisited[u] = true;**

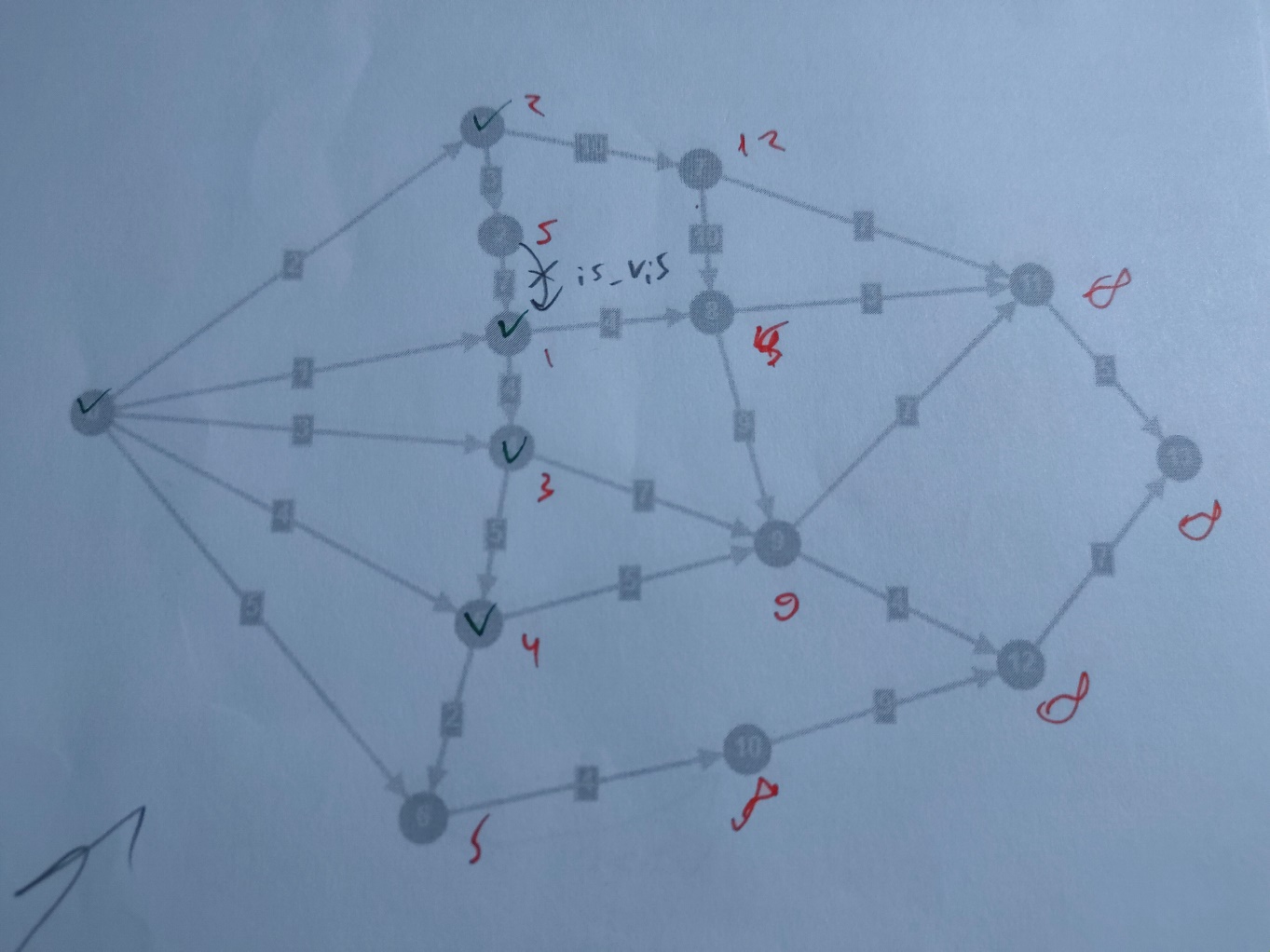
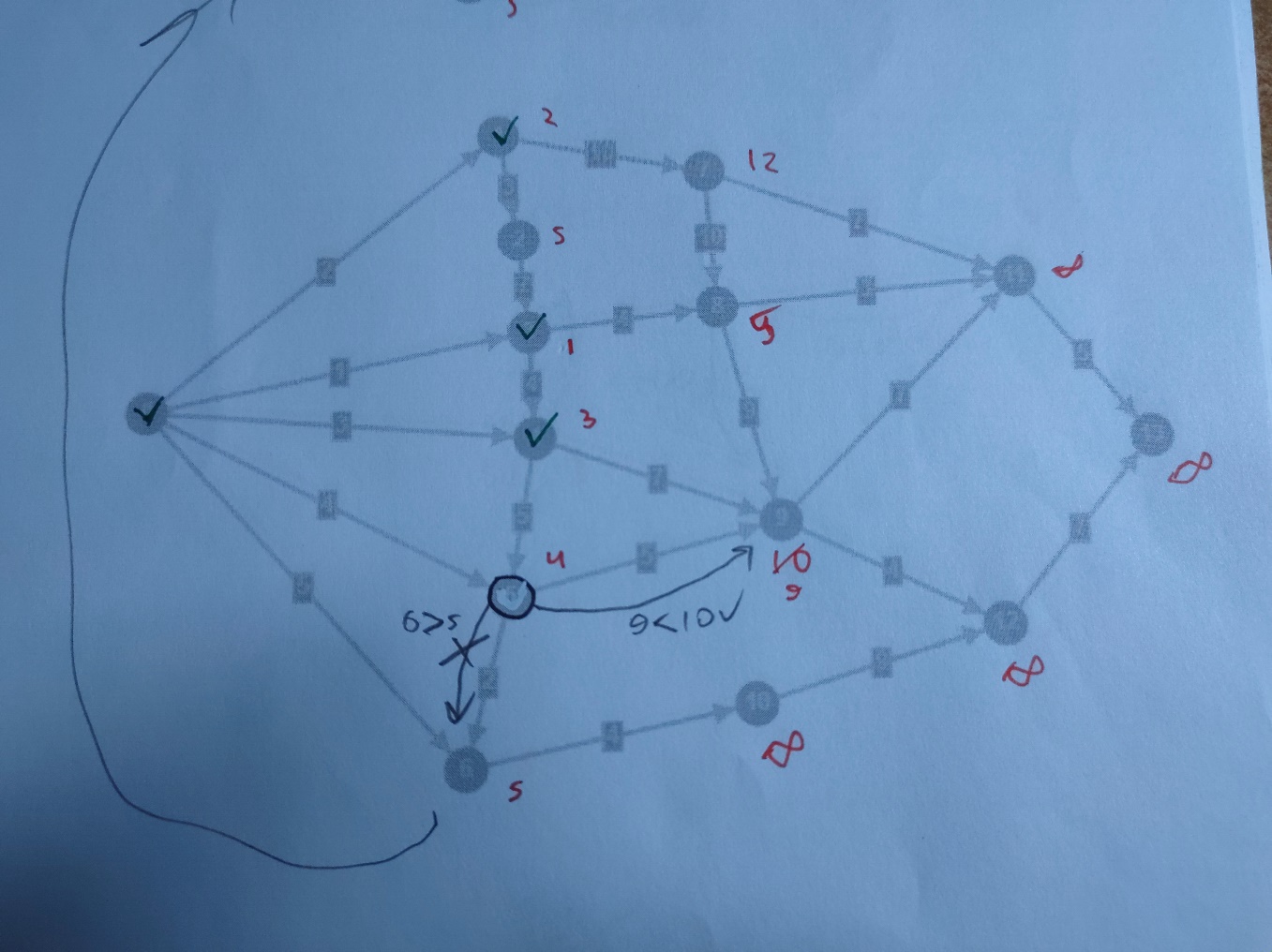
**for (int j = 0; j < SIZE\_GRAPH; ++j) {  
 if (!isVisited[j] && GRAPH[u][j] && distance[u] != INT\_MAX) {  
 if (distance[u] + GRAPH[u][j] < distance[j])  
 distance[j] = distance[u] + GRAPH[u][j];  
 }  
 }  
 }**

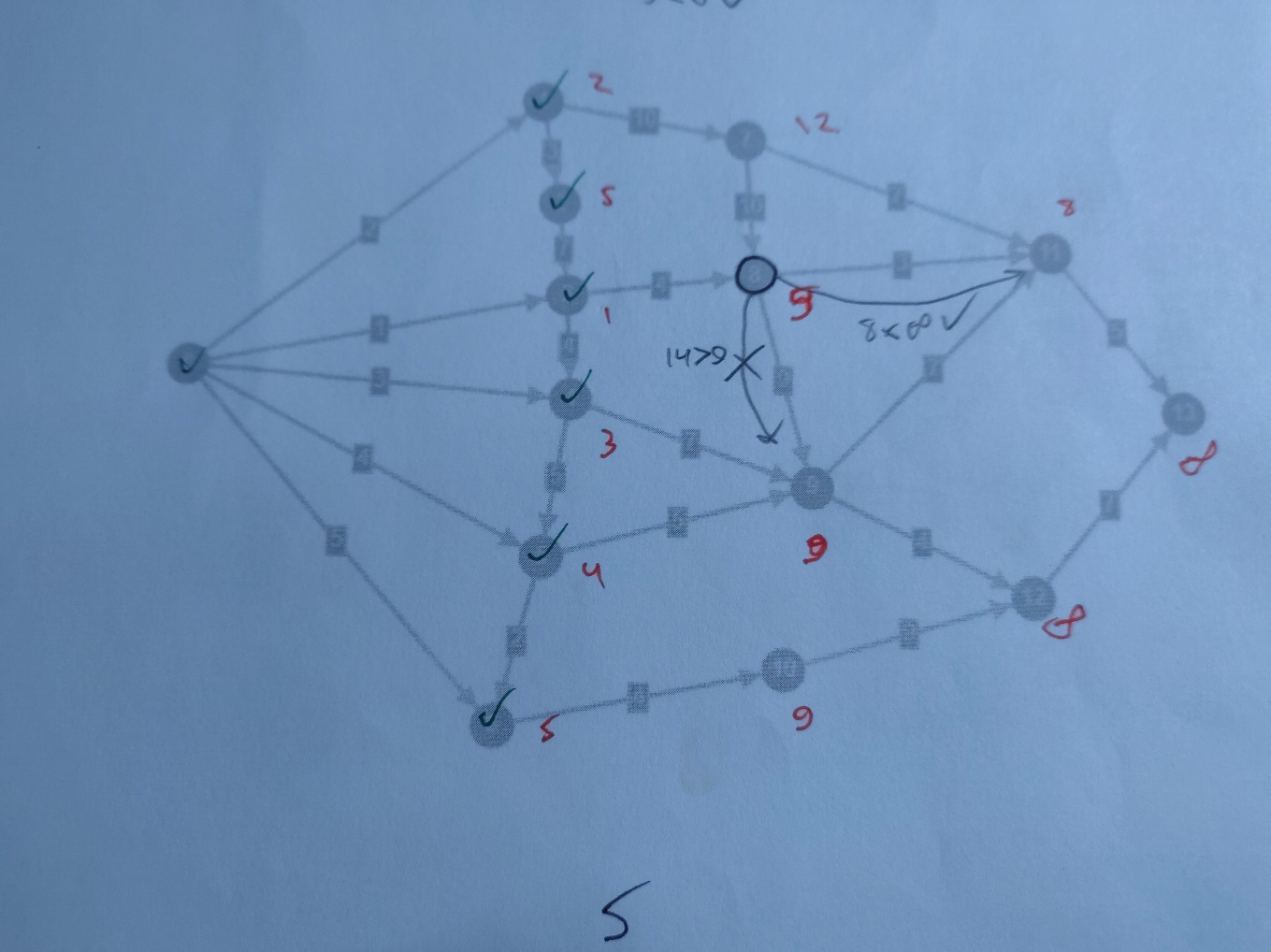
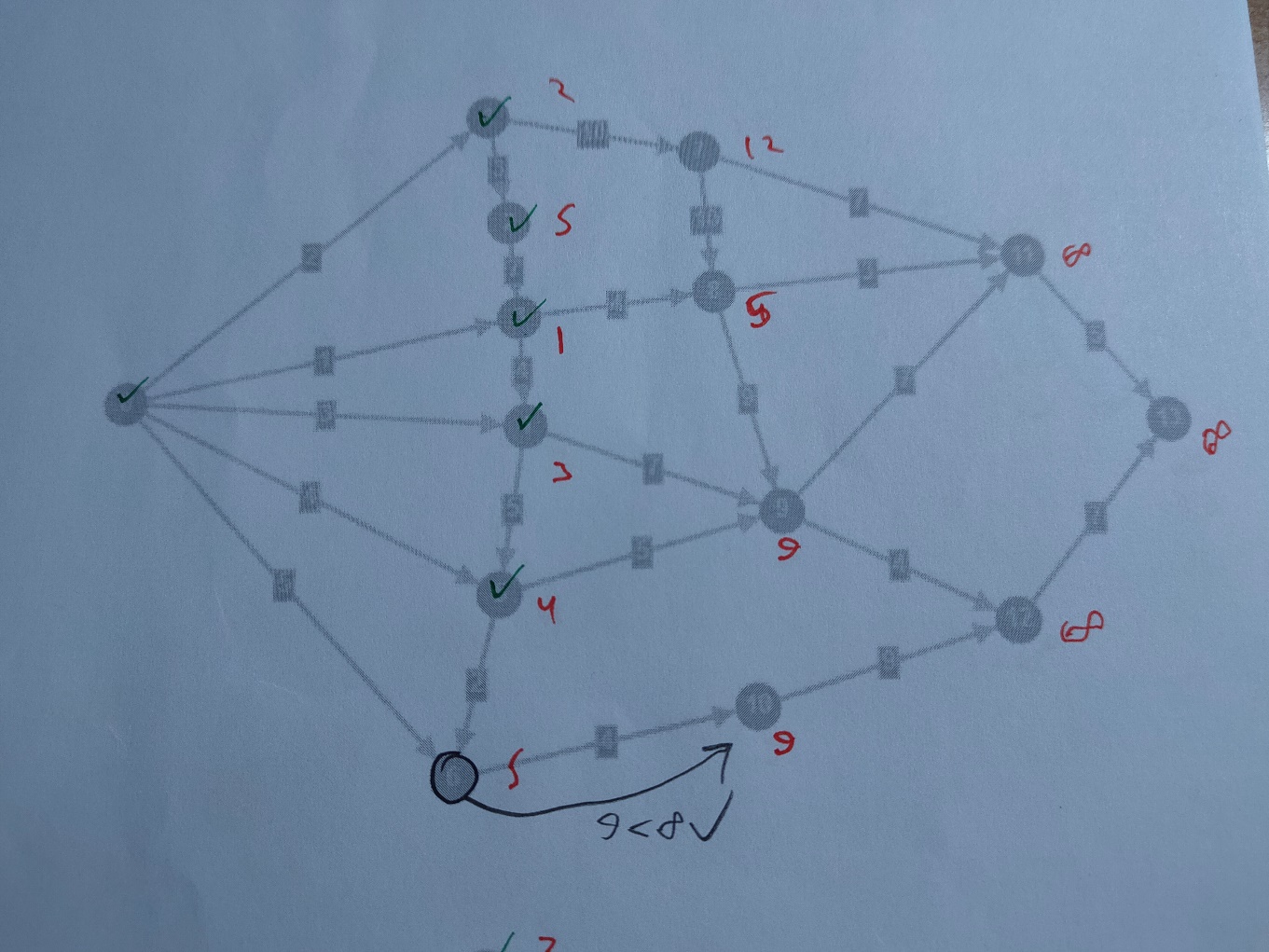
**cout << "Кратчайшие пути:\n";  
 for (int i = 0; i < SIZE\_GRAPH; i++) {  
 if (distance[i] != INT\_MAX) {  
 cout << startVertex << " -> " << i << " = " << distance[i] << endl;  
 } else {  
 cout << startVertex << " -> " << i << " = " << "error" << endl;  
 }  
 }  
}**

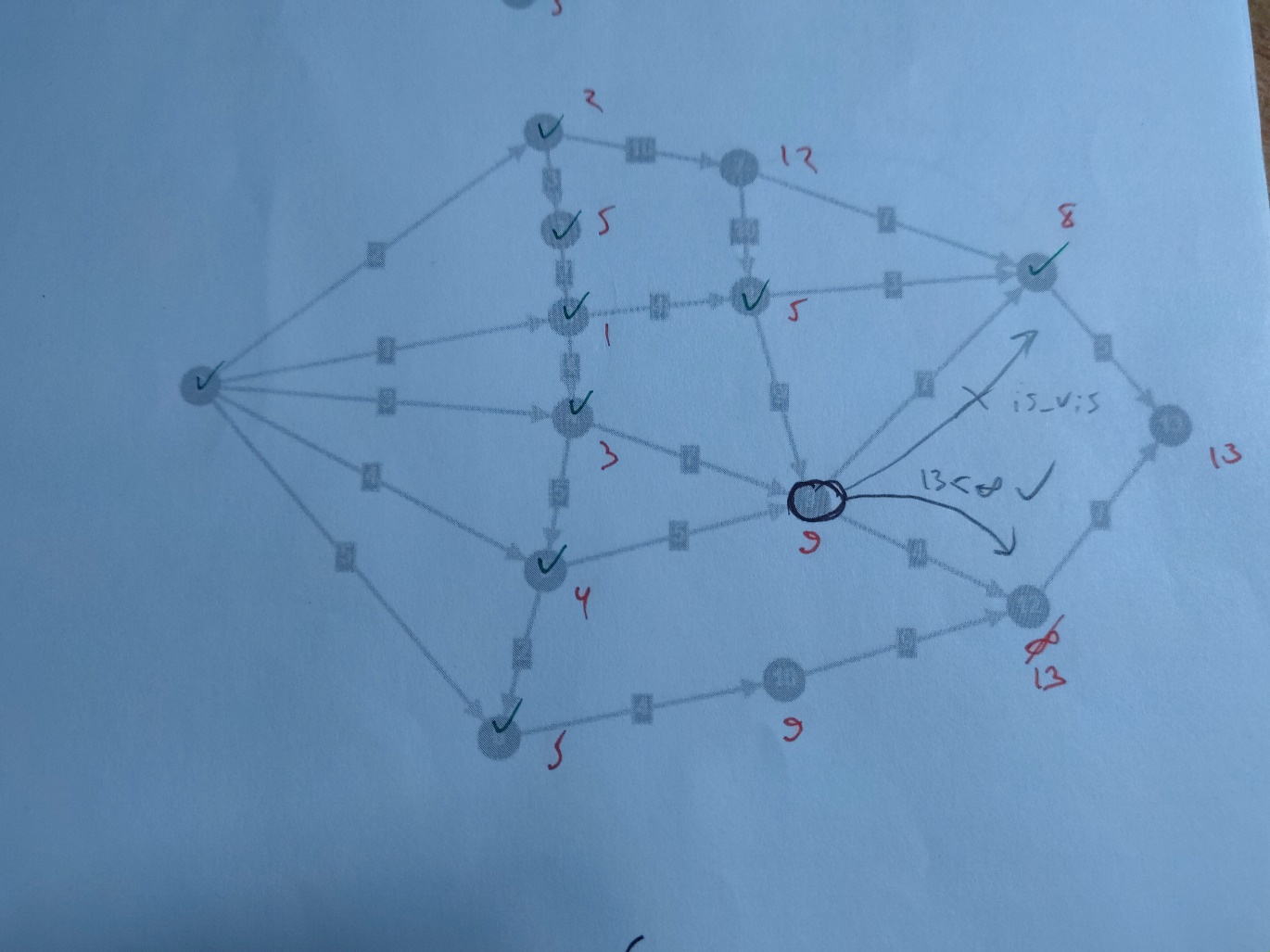
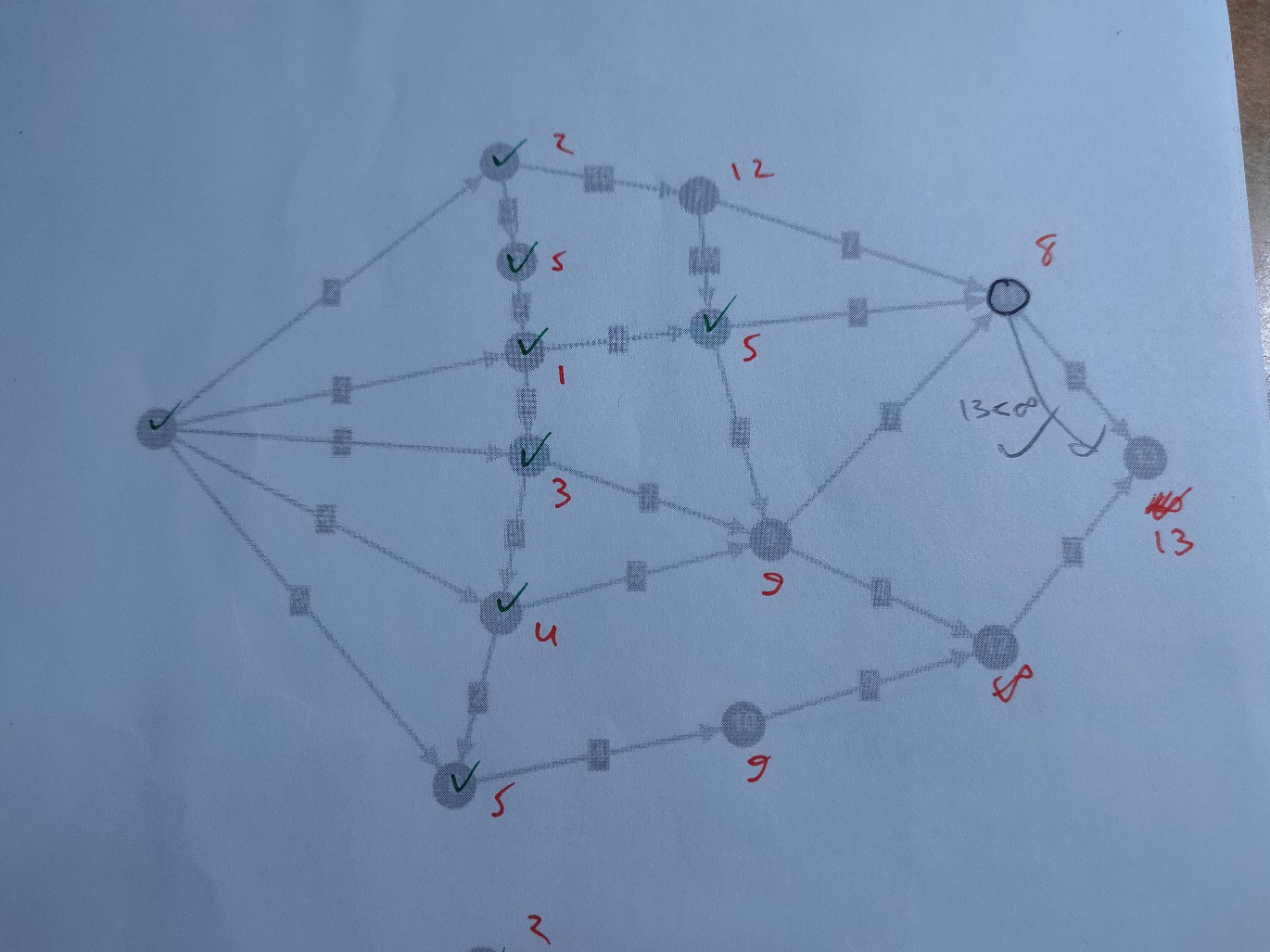
****

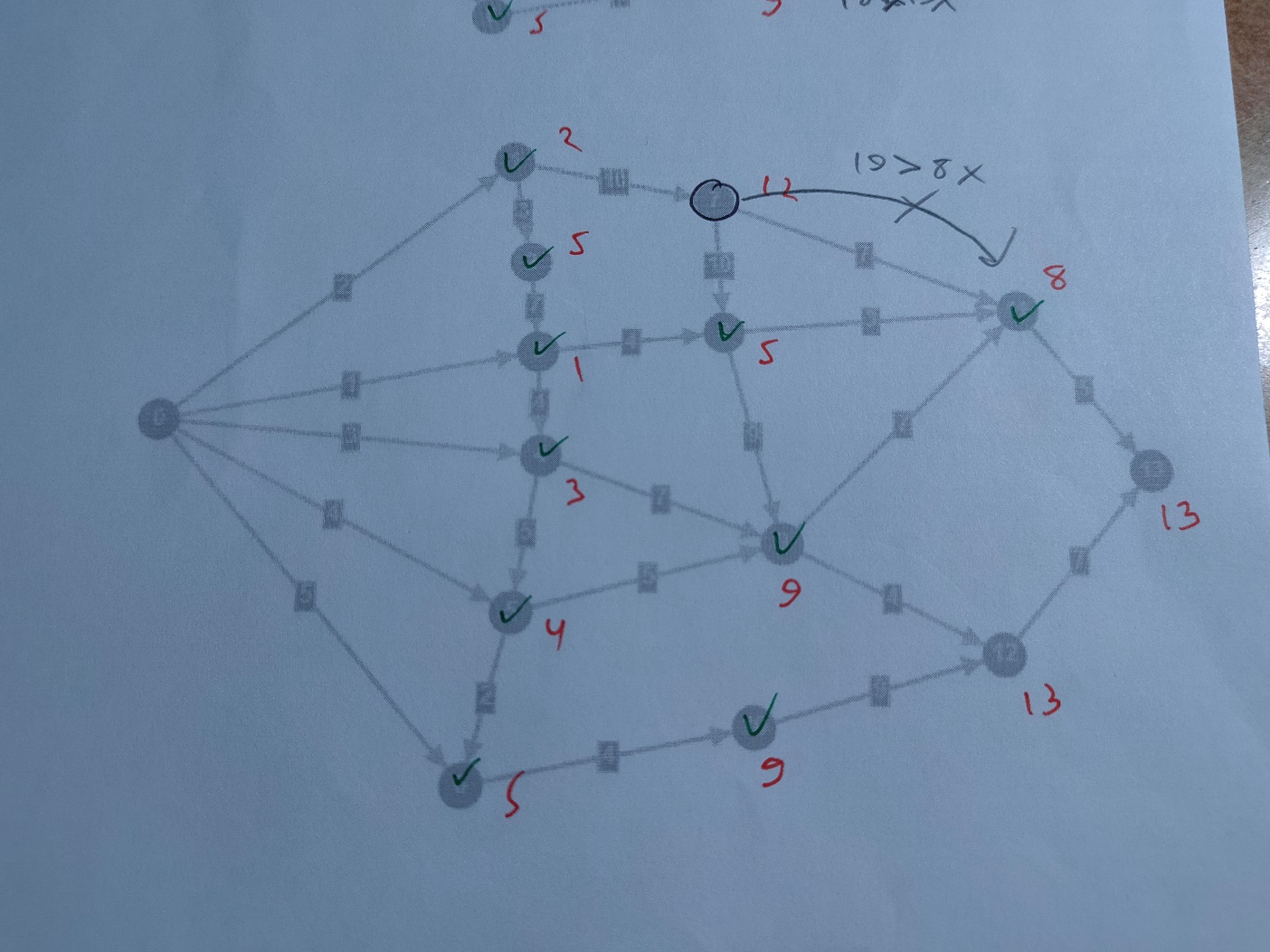
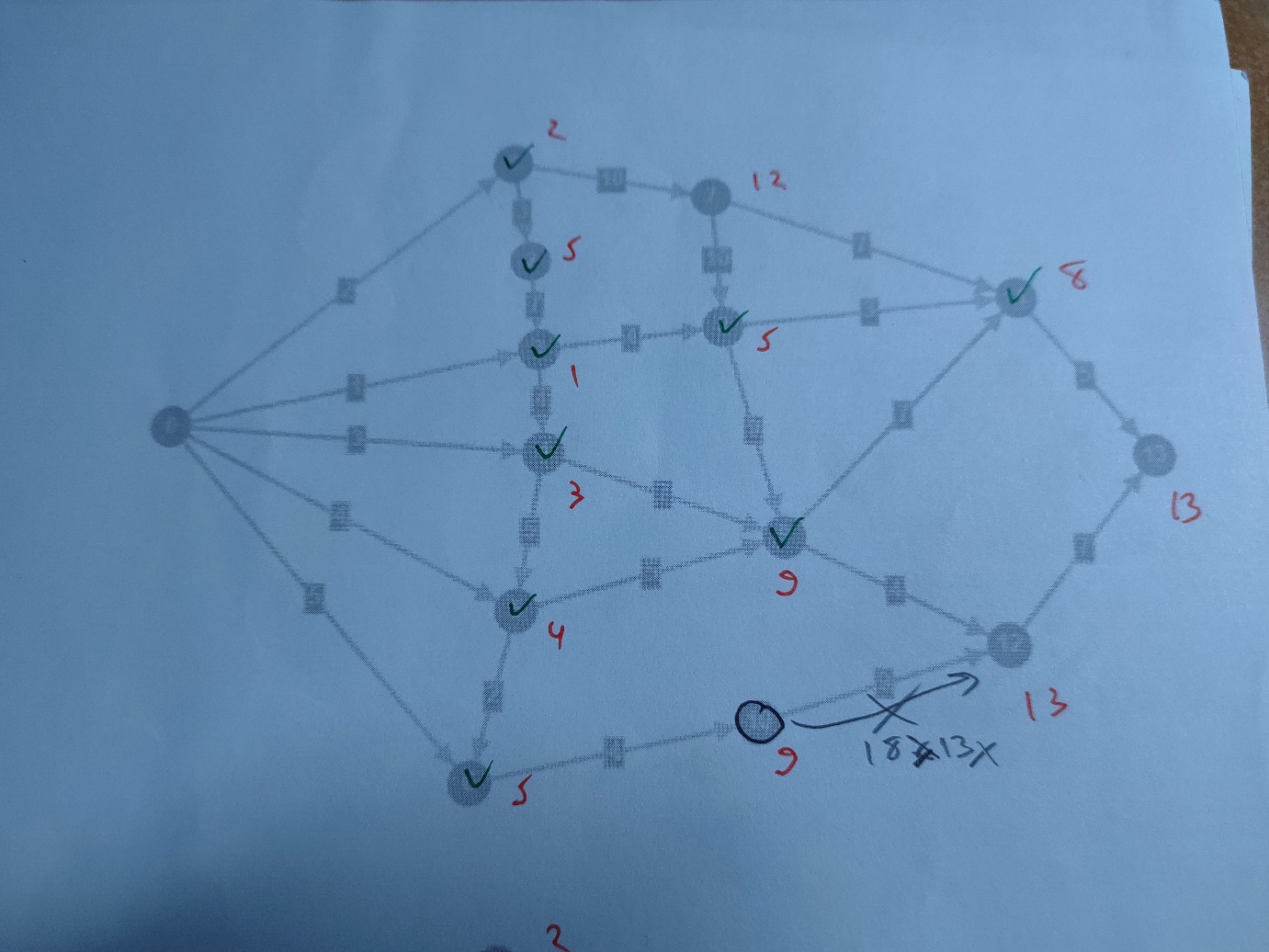


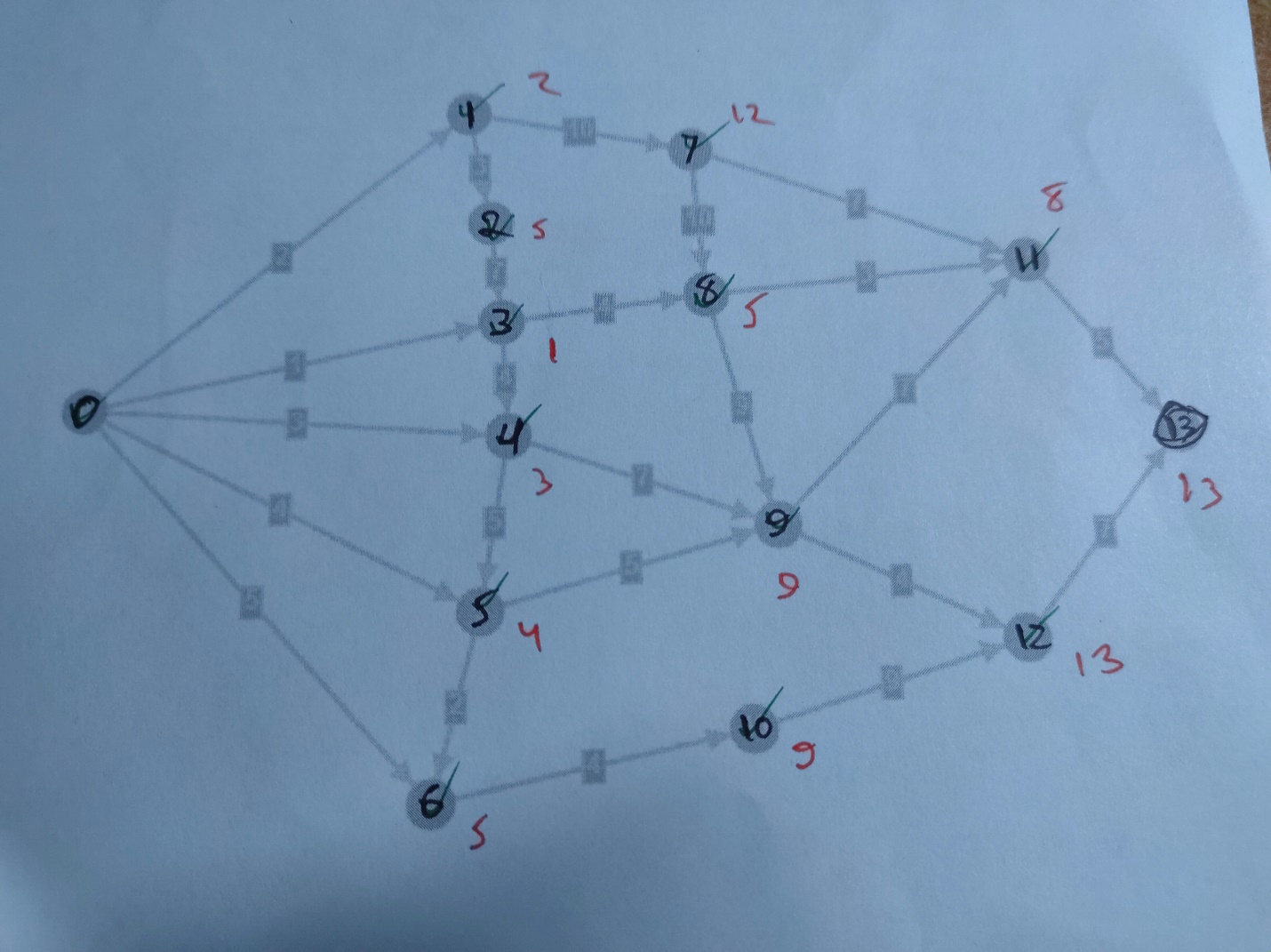
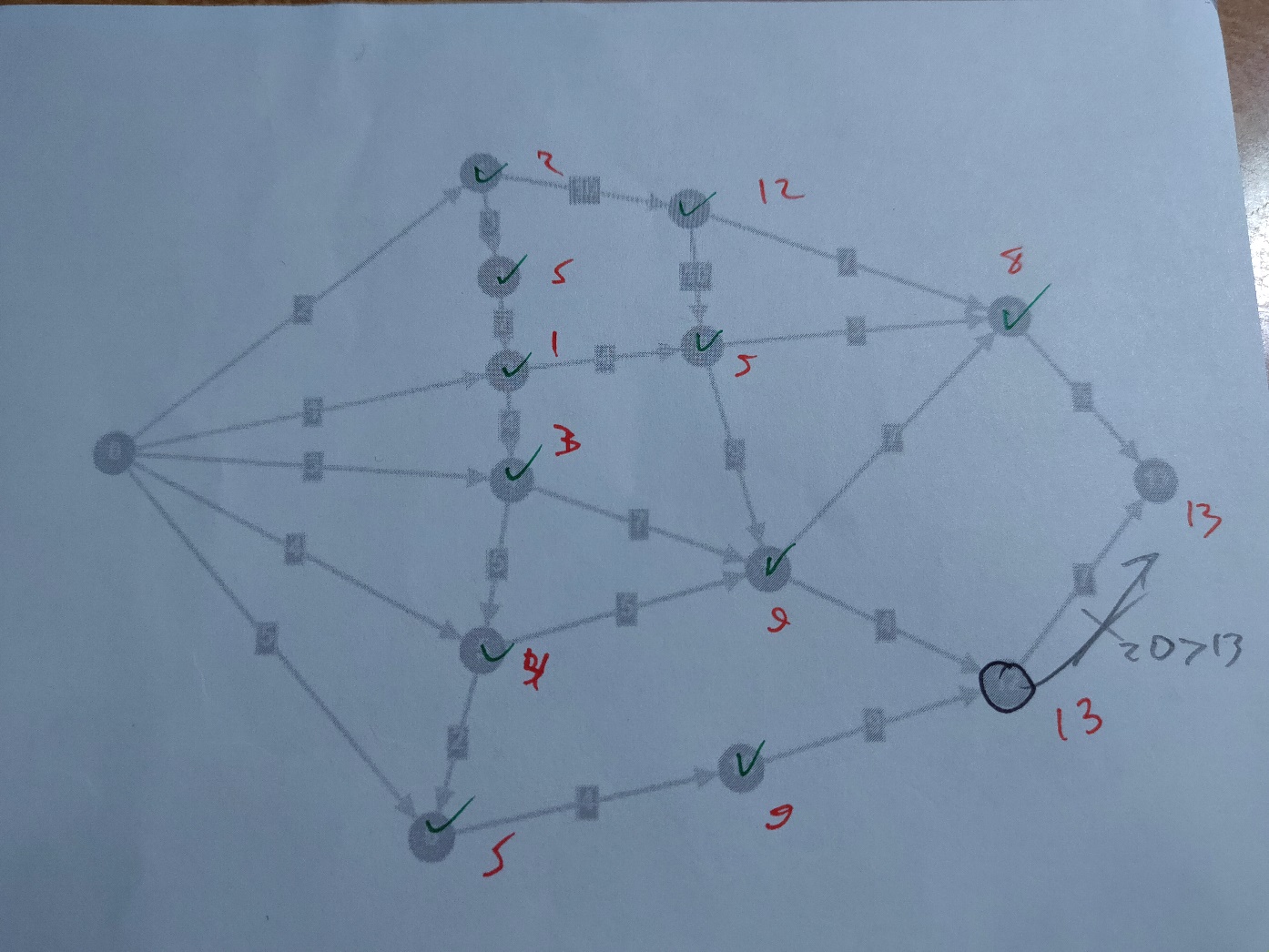




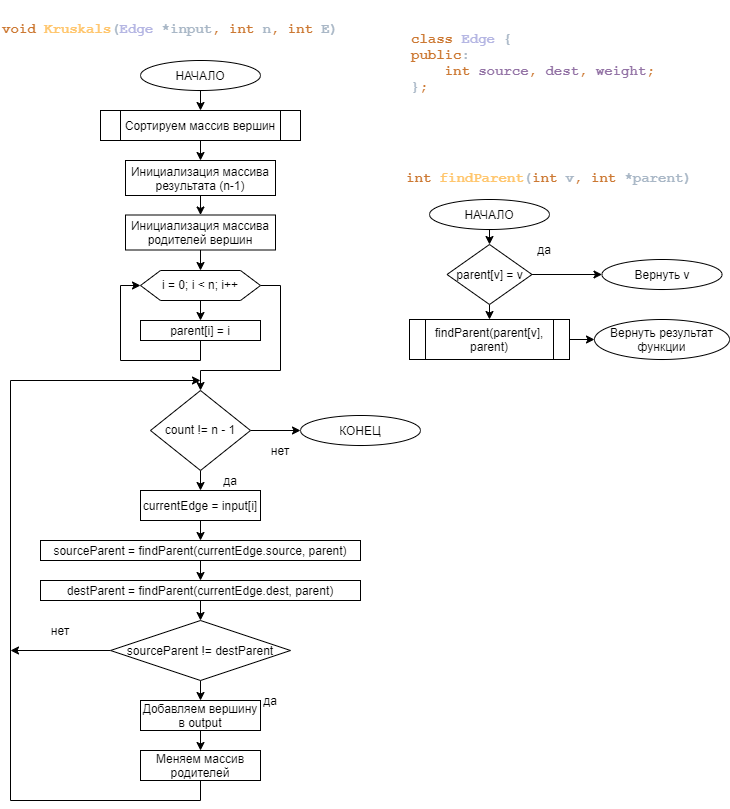


****

****

****

**Структурная схема**

****

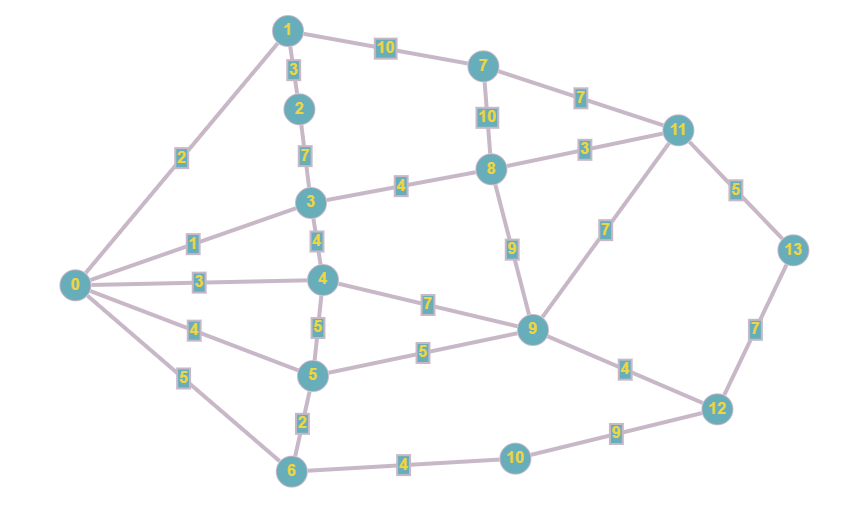
**Код программы**

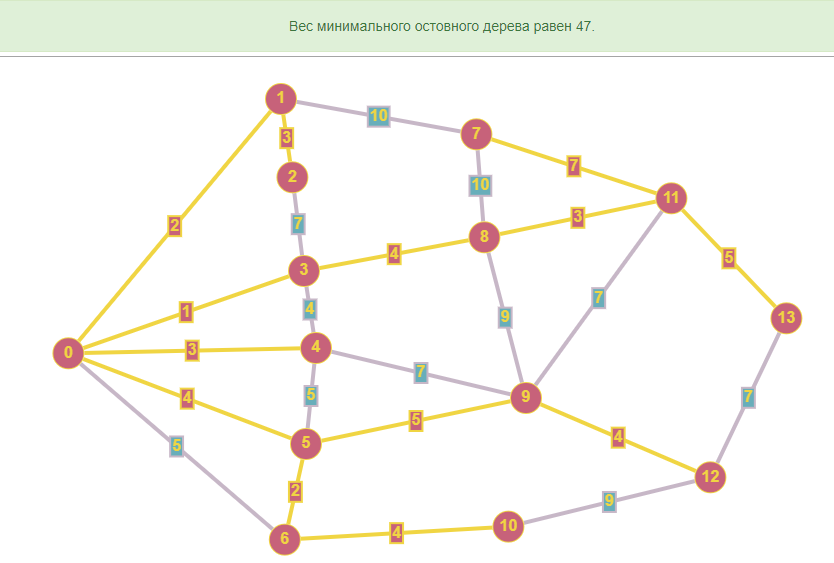
**class Edge {  
public:  
 int source, dest, weight;  
};**

**bool compare(Edge e1, Edge e2) {  
 return e1.weight < e2.weight;  
}**

**int findParent(int v, int \*parent) {  
 if (parent[v] == v)   
 return v;  
 return findParent(parent[v], parent);  
}**

**void Kruskals(Edge \*input, int n, int E) {  
 sort(input, input + E, compare);  
  
 Edge \*output = new Edge[n - 1];  
 int \*parent = new int[n];  
  
 for (int i = 0; i < n; i++)  
 parent[i] = i;  
  
 int i = 0, count = 0;  
  
 while (count != n - 1) {  
 Edge currentEdge = input[i];  
 int sourceParent = findParent(currentEdge.source, parent);  
 int destParent = findParent(currentEdge.dest, parent);  
 if (sourceParent != destParent) {  
 output[count] = currentEdge;  
 count++;  
 parent[sourceParent] = destParent;  
 }  
 i++;  
 }  
 int minWeight = 0;  
 for (i = 0; i < n - 1; i++) {  
 if (output[i].source < output[i].dest) {  
 cout << output[i].source << " -> " << output[i].dest << " : " << output[i].weight << endl;  
 } else {  
 cout << output[i].dest << " -> " << output[i].source << " : " << output[i].weight << endl;  
 }  
 minWeight += output[i].weight;  
 }  
 cout << "WEIGHT: " << minWeight << endl;  
}**

****

****

**Результаты работы программы**

Дейкстеры Крускула

