Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Мегафакультет компьютерных технологий и управления

Кафедра информатики и прикладной математики

Алгоритмы и структуры данных

Лабораторная работа №2

«Нахождение кратчайших путей в графе»

Группа: P3218

Студент: Петкевич Константин

Преподаватель: Зинчик А. А.

2017г

Задание

Предлагается попарное сравнение алгоритмов нахождения кратчайших путей от вершины $s \in V$ до всех остальных вершин в графе G = (V, E), имеющем n вершин и m ребер.

А алгоритм Дейкстры, использующий метки. Сложность O(E + V*log(V))

Б алгоритм БеллманаФорда. Сложность O(V*E)

1) Написать программу, реализующую алгоритм А и алгоритм В

2) Написать программу, реализующую алгоритм А и алгоритм В, для проведения экспериментов, в которых можно выбирать:
   - число n вершин и число m ребер графа
   - натуральные числа q и r, являющиеся соответственно нижней и верхней границей для весов ребер графа.

3) Провести эксперименты на основе следующих данных, сформировать графики T (n) и T (n): a b
   * Количество вершин n=1, ..., 10^4+1, шаг=100. Колво рёбер m=n^2/10. Мин.вес=1, Макс.вес=10^6
   * Количество вершин n=1, ..., 10^4+1, шаг=100. Колво рёбер m=n^2. Мин.вес=1, Макс.вес=10^6

4) Сформулировать и обосновать вывод о том, каких случаях целесообразно использовать алгоритм А, а в каких алгоритм Б.

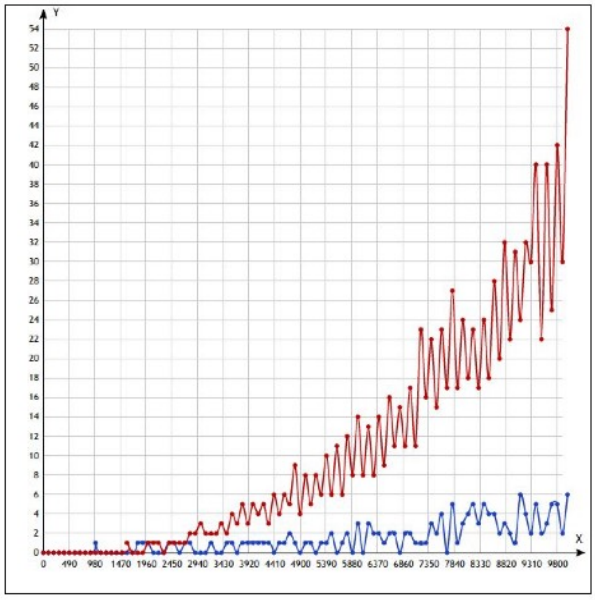
Листинг кода

```
// Lab 2
list<Vertex*> recoverShortestPath(vector<Vertex*> shortestPaths, Vertex* first, Vertex* last) {
   list<Vertex*> result;

   result.push_back(last);
   Vertex* i = last;
   do {
      i = shortestPaths[i->id];
      if (i == nullptr)
         return list<Vertex*>();

      result.push_back(i);
   } while (i != first);

   result.reverse();
   return result;
}
size_t getMinDistanceIndex(vector<int> &distances, vector<bool> &marked) {
   int min = INT_MAX;
   ssize_t minIndex = -1;

   for (int i = 0; i < distances.size(); i++) {
      if (!marked[i] && distances[i] <= min) {
         min = distances[i];
         minIndex = i;
      }
   }

   assert(minIndex >= 0);
   return (size_t)minIndex;
}
list<Vertex*> Dijkstra(Graph* graph, size_t startId, size_t destinationId) {
   vector<Vertex*> vertices = graph->getAllVertices();

   // Preparations
```

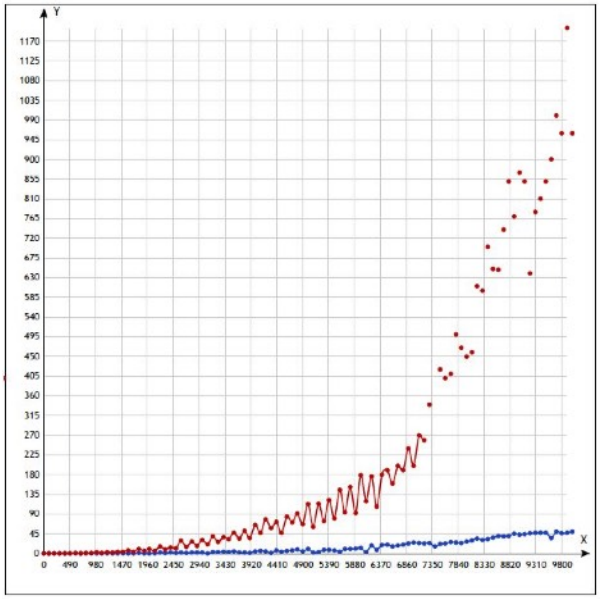
```
    vector<int> distances(graph->getVerticesAmount(), INT_MAX);
    vector<bool> marked(graph->getVerticesAmount(), false);
    vector<Vertex*> shortestPreviouses(graph->getVerticesAmount(), nullptr);
    distances[startId] = 0;

    // Find shortest path for all vertices
    for (size_t count = 0; count < vertices.size() - 1; count++) {
        int minIndex = getMinDistanceIndex(distances, marked);
        marked[minIndex] = true;

        if (minIndex == destinationId)
            break;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (list<Edge*>::iterator edgeIt = vertices[minIndex]->neighborhood.begin(); edgeIt != vertices[minIndex]-
>neighborhood.end(); edgeIt++) {
            if (!marked[(*edgeIt)->destination->id] && distances[minIndex] + (*edgeIt)->weight < distances[(*edgeIt)-
>destination->id]) {
                distances[(*edgeIt)->destination->id] = distances[minIndex] + (*edgeIt)->weight;
                shortestPreviouses[(*edgeIt)->destination->id] = (*edgeIt)->source;
            }
        }
    }

    return recoverShortestPath(shortestPreviouses, vertices[startId], vertices[destinationId]);
}
list<Vertex*> BellmanFord(Graph* graph, size_t startId, size_t destinationId) {
    vector<Vertex*> vertices = graph->getAllVertices();
    bool isDistancesUpdated = false;

    // Preparation
    vector<int> distances(graph->getVerticesAmount(), INT_MAX);
    vector<Vertex*> shortestPreviouses(graph->getVerticesAmount(), nullptr);
    distances[startId] = 0;

    for(size_t i = 0; i < vertices.size() - 1; i++) {
        for(vector<Vertex*>::iterator vertexIt = vertices.begin(); vertexIt != vertices.end(); vertexIt++) {
            if (distances[(*vertexIt)->id] == INT_MAX) { // int overflow may screw comparison up
                continue;
            } else {
                for (list<Edge *>::iterator neighborhoodIt = (*vertexIt)->neighborhood.begin(); neighborhoodIt != (*vertexIt)-
>neighborhood.end(); neighborhoodIt++) {
                    if (distances[(*neighborhoodIt)->destination->id] > distances[(*neighborhoodIt)->source->id] +
(*neighborhoodIt)->weight) {
                        isDistancesUpdated = true;
                        distances[(*neighborhoodIt)->destination->id] = distances[(*neighborhoodIt)->source->id] +
(*neighborhoodIt)->weight;
                        shortestPreviouses[(*neighborhoodIt)->destination->id] = (*neighborhoodIt)->source;
                    }
                }

            }
        }

        if (isDistancesUpdated == false)
            goto AllMinimumPathsHaveFoundLabel;
        else
            isDistancesUpdated = false;
    }

AllMinimumPathsHaveFoundLabel:
    return recoverShortestPath(shortestPreviouses, vertices[startId], vertices[destinationId]);
}
```

Эксперименты:

n = m^2/10



n = m^2



Вывод

Алгоритм Беллмана-Форда показывает себя медленней, чем алгоритм Дейкстры. Зато он более универсален, т.к. Может обрабатывать графы с отрицательными весами рёбер. Соответственно, при отрицательных рёбрах стоит его, во всех остальных случаях Дейкстра лучше.