ИКН НИТУ МИСИС

Комбинаторика и теория графов

Алгоритм Беллмана-Форда построения кратчайших путей

Исполнитель:

Лосев В.Л. БИВТ-23-18

(https://github.com/SomethingWF/Bellman-Ford_Alrotithm)

Москва 2024 год

Формальная постановка задачи

Имеется направленный граф G = (V, E), каждому ребру которого (i, j) ∈ E присвоен вес c_{ij}, который представляет собой стоимость прямого перехода от узла i к узлу j в графе.

Для заданного графа G решить, существует ли в G отрицательный цикл, то есть направленный цикл C, для которого сумма весов ребер c_{ij} (ij ∈ C), входящих в данный цикл, меньше нуля. Если граф не содержит отрицательных циклов, найти путь P от начального узла s к конечному узлу t с минимальной общей стоимостью. То есть сумма весов ребер c_{ij} (ij ∈ P), входящих в путь P, должна быть минимально возможной для любого пути s-t.

Именно эта задача и известна как «Задача нахождения кратчайшего пути», на решение которой и рассчитан алгоритм Беллмана-Форда.

Теоретическое описание алгоритма

На вход алгоритма подаются граф и начальная вершина source.

Результатом, возвращаемым алгоритмом является массив, содержащий длины кратчайших путей от sourse до всех вершин.

1. Инициализируются расстояния от исходной вершины до всех остальных, как бесконечные, кроме расстояния до исходной вершины, равного нулю. Реализуется в виде массива distance, индексы которого ставятся в соответствие порядковым номерам вершин.
2. В цикле v – 1 раз (где v – количество вершин графа), вычисляются самые короткие расстояния. А именно, на каждой итерации цикла для каждого ребра ij производится действие: если distance[j] > distance[i] + c_{ij}, то distance[j] = distance[i] + c_{ij}.
3. Производится проверка наличия в графе отрицательного цикла. Для каждого ребра *u*-*v* необходимо выполнить следующее: если distance[j] > distance[i] + c_{ij}, то в графе присутствует цикл отрицательного веса (из соображений того, что пункт 2 гарантирует подсчет кратчайшего расстояния до каждой из вершин).

Пример работы алгоритма

Возьмем за начальную вершину 0. Принимаем расстояния до всех вершин, кроме вершины 0, за бесконечные. Общее число вершин графа равно 5, значит все ребра нужно будет пройти 4 раза.

Ребра обрабатываются в порядке, указанном на рисунке 1.

Первая же итерация гарантирует, что кратчайшие пути будут иметь стоимость не более 7. Оставшиеся три итерации не изменят состояния массива distance.

Номера вершин	0	1	2	3	4	5	Пройденные ребра
Итерация 0	0	∞	∞	∞	∞	∞	
Итерация 1	0	5	7	9	11	∞	01-14
	0	5	7	9	8	11	34-35
	0	5	7	9	8	5	45

Порядок обхода ребер графа
01
02
12
13
14
34
35
45

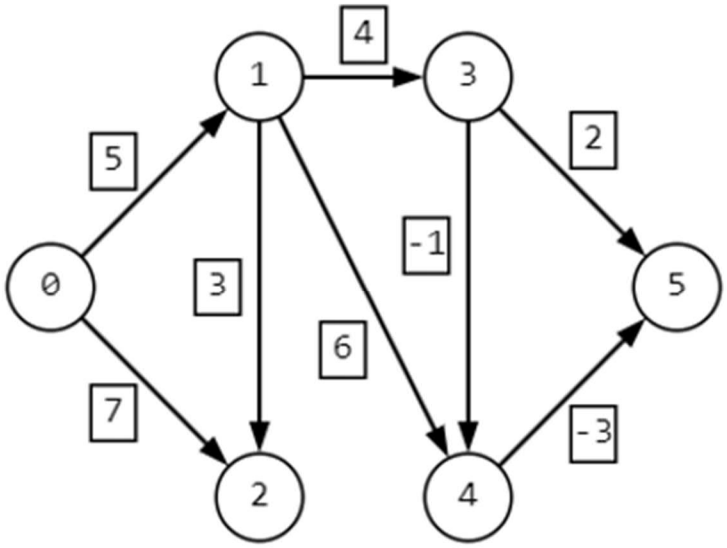

Рис 1 – Демонстрация работы алгоритма

Реализация алгоритма

В данной реализации граф представлен списком ребер. Сам граф передается по ссылке, для избежания лишнего копирования данных, отдельно передается количество вершин графа и исходная вершина.

Также данная реализация использует вектора. Именно он и создается для сохранения кратчайших расстояний от исходной вершины до остальных по вышеописанным правилам.

```
std::vector<int> bellmanFord(std::vector<Edge>& graph, int vertices, int source) {
  std::vector<int> distance(vertices, int_max);
  distance[source] = 0;

  for (int i = 0; i < vertices - 1; ++i) {
    for (const Edge& e : graph) {
      if (distance[e.source_] != int_max && distance[e.source_] + e.weight_ < distance[e.destination_]) {
        distance[e.destination_] = distance[e.source_] + e.weight_;
      }
    }
  }

  for (const Edge& e : graph) {
    if (distance[e.source_] != int_max && distance[e.source_] + e.weight_ < distance[e.destination_]) {
      throw std::runtime_error("Graph contins negative weight cycle");
    }
  }

  return distance;
}
```

Рис 2 – Реализация алгоритма Беллмана-Форда

Анализ временной и пространственной сложности

Пространственная сложность алгоритма составляет O(n).

Хранение массива кратчайших расстояний от исходной вершины до остальных составляет O(n), так как количество его ячеек составляет количество вершин графа, на котором применяется алгоритм.

Временная сложность алгоритма составляет O(v*e), где v – количество вершин, t – количество ребер.

Обход e ребер производится v - 1 раз. Так как константы при подсчете асимптотической сложности не учитываются, а обращение к элементам массива исполняется за константное время, то сложность данного цикла составляет O(v*e).

Следом производится еще один цикл – разовый проход по ребрам графа, для выявления отрицательных циклов.

При сложении асимптотических сложностей разного порядка учитывается лишь сложность более высокого порядка, итого O(v*e).

Анализ аналогов

Алгоритм Дейкстры – это алгоритм для нахождения кратчайших путей от одной начальной вершины до всех остальных вершин в графе с неотрицательными весами рёбер.

Пространственная сложность алгоритма Дейкстры составляет O(v), так как необходимо хранить расстояния до всех вершин и, возможно, предшественников для восстановления пути.

При использовании простого массива для хранения расстояний временная сложность составляет $O(v^2)$, где v — количество вершин. При использовании структуры данных "куча" (например, двоичной кучи) временная сложность снижается до O((v+r)logv), где e — количество рёбер.

Алгоритм Беллмана-Форда имеет более высокую временную сложность, что делает его менее эффективным для разреженных графов по сравнению с Дейкстрой, особенно при использовании кучи. Зато он имеет возможность работать с отрицательными весами ребер, что и определяет ключевую разницу в области применения данных алгоритмов.

Описание алгоритма Дейкстры

1. Инициализация. Устанавливается расстояние до начальной вершины, равное 0, а до всех остальных – бесконечность. Создается множество непосещённых вершин.
2. Выбор вершины. Выбирается непосещённая вершина с наименьшим расстоянием (начиная с начальной вершины).
3. Обновление расстояний. Для каждой соседней вершины текущей вершины обновляется расстояние, если найденный путь через текущую вершину короче, чем ранее известное расстояние.
4. Пометка вершины как посещённой. После обработки всех соседей текущая вершина помечается как посещённую.
5. Повторение. Шаги 2-4 повторяются, пока не будут посещены все вершины или пока не будет достигнута целевая вершина.

Примеры реального использования

Алгоритм Беллмана-Форда используется в различных приложениях, особенно в тех случаях, когда графы могут содержать рёбра с отрицательными весами. Вот несколько примеров реального использования этого алгоритма:

1. Финансовые приложения: В финансовых системах, таких как системы управления активами или кредитные рейтинги, могут возникать ситуации, когда некоторые транзакции имеют отрицательные веса (например, скидки или возвраты). Алгоритм Беллмана-Форда может использоваться для анализа таких графов и нахождения оптимальных путей или минимальных затрат.
2. Сетевые протоколы: В сетевых протоколах, таких как RIP (Routing Information Protocol), алгоритм Беллмана-Форда используется для вычисления кратчайших путей в сетях. Он позволяет учитывать изменения в топологии сети и обновлять

маршруты, даже если некоторые из них могут иметь отрицательные веса (например, при учете штрафов за перегрузку).

3. Оптимизация маршрутов: В задачах оптимизации маршрутов, где необходимо учитывать различные факторы, такие как время в пути, стоимость проезда и другие параметры, алгоритм Беллмана-Форда может помочь найти оптимальные маршруты, даже если некоторые из этих параметров могут быть отрицательными (например, временные скидки).

Ссылка на реализацию

https://github.com/SomethingWF/Bellman-Ford_Alrotithm

Список источников

1. https://www.geeksforgeeks.org/bellman-ford-algorithm-in-cpp/
2. https://en.wikipedia.org/wiki/Bellman–Ford_algorithm
3. https://habr.com/ru/companies/otus/articles/484382/
4. https://habr.com/ru/companies/otus/articles/748470/
5. Дж. Клейнберг, Е. Тардос, Алгоритмы: разработка и применение. Классика Computers Science.