Алгоритмы и структуры данных-2

SET 6. Задача A2.

Весна 2024. Клычков М. Д.

Пункт 1. Покажем, что такой алгоритм вообще существует. Алгоритм находит кратчайшие пути (в плане описанной метрики) до каждой вершины, то есть, зафиксировав стартовую вершину s:

$$\forall v \in V : v \neq s \Rightarrow d(s, v) \to \min$$

Рассмотрим такое расстояние до вершины v_k:

$$d(s, v_k) = w(s, v_1) \cdot w(v_1, v_2) \cdot w(v_2, v_3) \cdot \ldots \cdot w(v_{k-1}, v_k)$$

Рассмотрим $\log d(s, v_k)$, разложим как логарифм произведения:

$$\log d(s, v_k) = \log w(s, v_1) + \log w(v_1, v_2) + \log w(v_2, v_3) + \ldots + \log w(v_{k-1}, v_k)$$

Получается, что можно заменить веса исходного графа на логарифмы этих же весов и решить задачу кратчайшего расстояния классическим Алгоритмом Дейкстры, очевидно что из полученного массива расстояний d' можно получить расстояния d. *Корректность доказана.*

Что касается ограничений (очевидно, что они есть, так как ими обладает классический Алгоритм Дейкстры), их можно вывести доказывая жадность алгоритма или, пользуясь найденным изоморфизмом между требуемым и классическим алгоритмами. Для классического $w'(v_i, v_j) \geq 0$, тогда для требуемого: $\log w'(v_i, v_j) \geq \log 0 \Leftrightarrow w(v_i, v_j) \geq 1$.

```cpp
using WeightAdj = std::pair<int, int>;
using AdjLists = std::vector<std::vector<WeightAdj>>;

std::vector<int> Dijkstra(const AdjLists& adj, int start) {
  std::priority_queue<WeightAdj, std::vector<WeightAdj>, std::greater<>> pq;
  std::vector<int> dist(adj.size(), INT_MAX);
  dist[start] = 1;
  pq.emplace(1, start);

  while (!pq.empty()) {
    auto [d, u] = pq.top();
    pq.pop();

    if (d > dist[u]) {
      continue;
    }

    for (const auto& [w, v] : adj[u]) {
      if (dist[v] > dist[u] * w) {
        dist[v] = dist[u] * w;
        pq.emplace(dist[v], v);
      }
    }
  }

  return dist;
}
```

Пункт 2. Алгоритм `RestoreGraph` будет восстанавливать граф по следующему правилу: будем для каждой пары вершин (v_i, v_j) таких, что $d(v_i, v_j) < \infty$ переберем все вершины $v_k \in V \setminus \{v_i, v_j\}$ и проверим $d(v_i, v_j) = d(v_i, v_k) + d(v_k, v_j)$, в зависимости от этого, поймем, есть ли ребро между двумя вершинами. Реализуем алгоритм на `C++`, будем считать, что матрица расстояний задана корректно и ответ существует (подробности далее):

```cpp
using DistMatrix = std::vector<std::vector<int>>;

AdjLists RestoreGraph(DistMatrix dist) {
  AdjLists adj(dist.size());
  int n = dist.size();

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (dist[i][j] == INT_MAX || i == j) {
        continue;
      }

      bool edge = true;
      for (int k = 0; k < n; ++k) {
        if (k == i || k == j) {
          continue;
        }

        if (dist[i][j] == dist[i][k] + dist[k][j]) {
          edge = false;
        }
      }

      if (edge) {
        adj[i].emplace_back(dist[i][j], j);
      }
    }
  }

  return adj;
}
```

Наложим некоторые ограничения на исходную матрицу. Во-первых, расстояния должны быть определены корректно, нет отрицательных циклов, в том числе должно выполняться неравенство $\forall v_k \in V : d(v_i, v_k) < \infty \wedge d(v_k, v_j) < \infty \Rightarrow d(v_i, v_j) \leq d(v_i, v_k) + d(v_k, v_j)$.

Интересное замечание: можно было бы провести ребра между всеми вершинами с весами равными $d(v_i, v_j)$, получился бы корректный граф...

Пункт 3. Рассмотрим алгоритм

```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Рис. 1: Код из условия

В алгоритме Флойда-Уоршелла определение расстояний, (*возможно*) включающих «промежуточную» вершину k должно начинаться только тогда, когда определены все расстояния (*возможно*) включающие вершину $k-1$, в этом и заключается ошибка в указанном алгоритме. Рассмотрим контрпример.

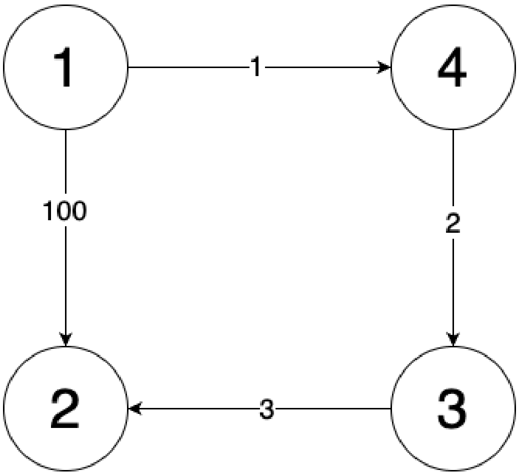

Рис. 2: Контрпример

Для начала выпишем ожидаемый ответ — матрица кратчайших расстояний между всеми парами вершин:

$$\begin{pmatrix} 0 & 6 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

Нулевым шагом (перед запускам тройного цикла) мы должны проинициализировать матрицу расстояний с помощью информации о весе ребер (буквально матрица смежности с 0 на диагонали):

$$\begin{pmatrix} 0 & 100 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

Теперь последовательно будем выписывать матрицы, которые будут получаться для после каждого окончания цикла по j, то есть для каждой уникальной пары (i, j).

$$(1,1): \begin{pmatrix} 0 & 100 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(1,2): \begin{pmatrix} 0 & 100 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(1,3): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(1,4): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(2,1): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(2,2): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(2,3): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(2,4): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(3,1): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(3,2): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(3,3): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(3,4): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(4,1): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(4,2): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(4,3): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

$$(4,4): \begin{pmatrix} 0 & 100 & 3 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

Можно заметить, что расстояние $d(1,2) = 100 \neq 6$.

Пункт 4. Достаточно легко придумать такой граф, в котором единственный путь $a \rightsquigarrow b$ и $b \rightsquigarrow a$, причем $a \rightsquigarrow v_i \to v_j \rightsquigarrow b$ и $b \rightsquigarrow v_i \to v_j \rightsquigarrow a$, ребра могут иметь различные веса (*про веса далее более подробно*).

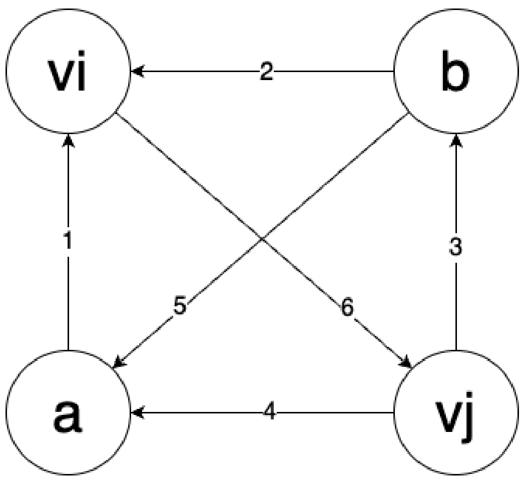

Рис. 3: Пример графа

Что касается структуры графа, то в нем всегда будет цикл, так как есть пути $a \rightsquigarrow b$ и $b \rightsquigarrow a$, то есть это уже не может быть дерево.

Также можно обсудить отрицательные циклы (*сумма весов рёбер которого отрицательна*): можно построить отрицательный цикл на четырех вершинах, при этом каждое ребро попадет в кратчайший путь между любыми двумя вершинами цикла. Однако, получается, что расстояние между любой парой вершин в этом случае $-\infty$. Корректность такого расстояния является *философским* вопросом (нужно формально вводить все определения и аксиомы для расстояний, чтобы понять это).

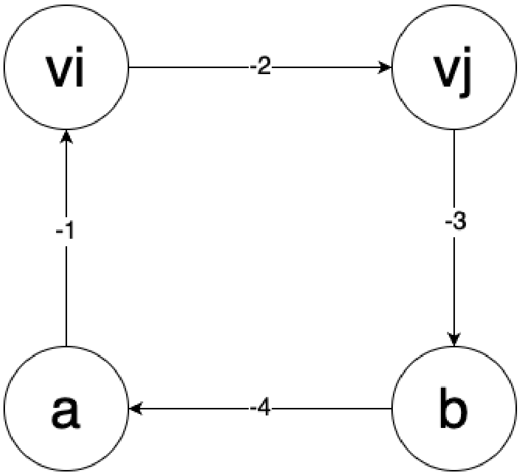

Рис. 4: Пример графа с отрицательным циклом

Никаких «аномалий» в графах, соответствующих условию, мы не нашли, поэтому и введенные на курсе алгоритмы для поиска кратчайших расстояний продолжают «работать». Так, алгоритм Дейкстры будет выдавать правильный результат только на графах с неотрицательными ребрами, а алгоритмы Беллмана-Форда и Флойда-Уоршелла на любых графах.