Лабораторная работа #3

Кратчайшие пути от вершины 1 до всех остальных в связном графе и граф-решётке сохранены в файле: result.txt. А измеренное время выводится при работе программы:

```
Связный граф из 20 вершин:
Время выполнения: 0.000008 секунд

Граф-решетка размерностью 100x100 вершин:
Время выполнения: 0.245680 секунд
```

Контрольные вопросы

1. Основные способы представления графов в памяти:

➔ Матрица смежности (adjacency matrix) — эффективна для насыщенных графов

➔ Список смежности (adjacency list) — эффективен для разреженных графов

Матрица A смежности (adjacency matrix) — это матрица из $n \times n$ элементов, в которой

a_{ij}={1 , если $(i, j) \in E$, 0, иначе.

Объём требуемой памяти — $O(|V|^2)$

Быстрое определение наличия ребра (i, j) в графе — за время O(1) получаем доступ к элементу a_{ij} матрицы.

Эффективна для насыщенных графов ($|E| \approx |V|^2$)

int a[n][n];

sizeof(int) = 4 байта

8 Гб = $8 \cdot 2^{30}$ байт

$8 \cdot 2^{30} / 4 = 2 \cdot 2^{30}$ — можно разместить 231 = 2 147 483 648 элементов типа int

$n = \lfloor\sqrt{2^{31}}\rfloor = 46\,340$ — количество строк и столбцов

int a[46340][46340];

Надо учесть, что часть памяти занята ОС и другими программами (предположим, что доступно 90% памяти, ~7 Гб, тогда n = 43 347)

Список смежности (adjacency list) — это массив A[n],

каждый элемент A[i] которого содержит список узлов, смежных с вершиной i

Эффективен для разреженных графов ($|E| \approx |V|$)

Реализация списка смежных вершин на основе массивов A[n + 1] и L[2m]

Список смежных вершин узла i: L[A[i]], L[A[i] + 1], ..., L[A[i + 1] – 1]

2. Алгоритм Дейкстры (Dijkstra’s algorithm, 1959) — алгоритм поиска кратчайшего пути в графе из заданной вершины во все остальные (single-source shortest path problem)

Пример: найти кратчайший путь из вершины 1 в вершину 5

Введём обозначения:

➔ H — множество посещённых вершин

➔ D[i] — текущее известное расстояние от вершины s до вершины i

➔ prev[i] — номер вершины, предшествующей i в кратчайшем пути

1. Устанавливаем расстояние D[i] от начальной вершины s до всех остальных в ∞

2. Полагаем D[s] = 0

3. Помещаем все вершины в очередь с приоритетом Q (min-heap): приоритет вершины i — это значение D[i]

4. Запускаем цикл из n итераций (по числу вершин):

а. Извлекаем из очереди Q вершину v с минимальным приоритетом — ближайшую к s вершину

б. Отмечаем вершину v как посещенную (помещаем во множество H)

в. Возможно, пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u, смежной с v и не включённой в H, проверяем и корректируем расстояние D[u]

В массиве D[1:n] содержатся длины кратчайших путей из начальной вершины s = 1

➔ D[1] — длина пути из 1 в 1

➔ D[2] — длина пути из 1 в 2

➔ D[3] — длина пути из 1 в 3

➔ D[4] — длина пути из 1 в 4

➔ D[5] — длина пути из 1 в 5

Восстановление кратчайшего пути:

➔ Массив prev[i] содержит номер вершины, предшествующей i в пути

➔ Восстанавливаем путь с конца:

➢ Вершина 5

➢ Вершина prev[5] = 3

➢ Вершина prev[3] = 4

➢ Вершина prev[4] = 1

3. Вычислительная сложность алгоритма Дейкстры определяется следующими факторами:

1. Выбор структуры данных для хранения графа (матрица смежности, список смежности)

2. Способ поиска вершины с минимальным расстоянием D[i]:

➔ Очередь с приоритетом: бинарная куча — O(logn), фибоначчиева куча, ...

➔ Сбалансированное дерево поиска: красно-чёрное дерево — O(logn), АВЛ-дерево, ...

➔ Линейный поиск — O(n)

В худшем случае функция PriorityQueueExtractMin() вызывается n раз, суммарная сложность — O(nlogn)

В худшем случае функция PriorityQueueDecreaseKey() вызывается для каждого из m рёбер графа,

суммарная сложность — O(mlogn)

Вариант 1. D — это массив или список: поиск за время O(n)

$$T_{Dijkstra} = O(n^2 + m) = O(|V|^2 + |E|)$$

Вариант 2. D — это бинарная куча

$$T_{Dijkstra} = O(n\log n + m\log n) = O(m\log n)$$

Вариант 3. D — это фибоначчиева куча (Fibonacci heap)

$$T_{Dijkstra} = O(m + n\log n)$$

4. Бинарная куча (пирамида, сортирующее дерево, binary heap) — это двоичное дерево,

удовлетворяющее следующим условиям:

➔ Приоритет любой вершины не меньше (≥) приоритета её потомков

➔ Дерево является полным бинарным деревом (complete binary tree) — все уровни заполнены слева направо (возможно, за исключением последнего)

Реализация бинарной кучи на основе массива

Создание пустой кучи

Удаление кучи. Обмен узлов кучи

Поиск максимального элемента

Вставка элемента в бинарную кучу

Удаление максимального элемента

Восстановление свойств кучи

Увеличение приоритета элемента

Построение бинарной кучи

На основе бинарной кучи можно реализовать алгоритм сортировки с вычислительной

сложностью O(nlogn) в худшем случае