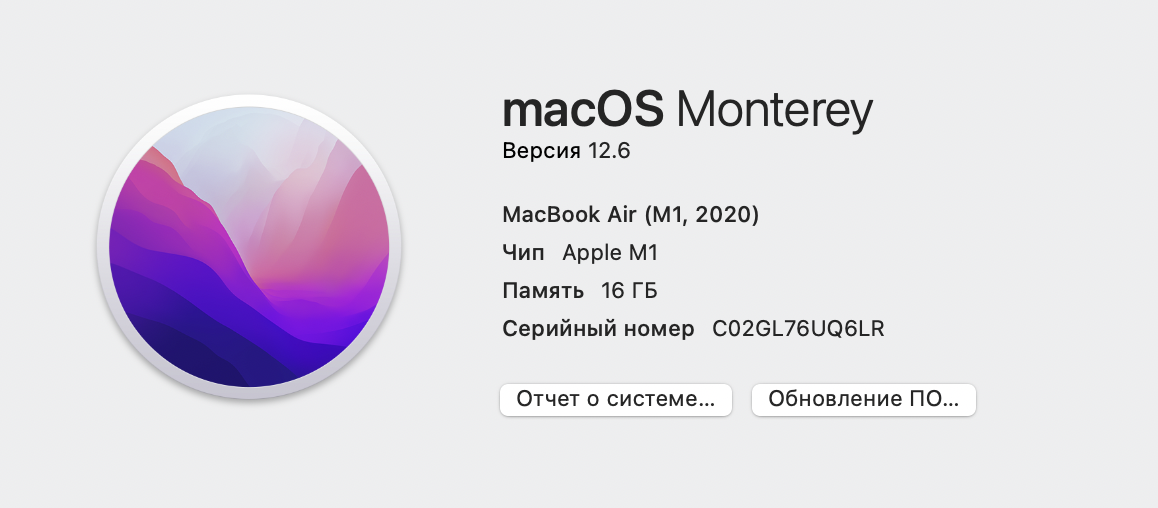
Отчет

Постановка задачи

Реализовать библиотеку алгоритмов поиска пути на графе (минимум 4; с поддержкой орграфов и весов) и протестировать их по производительности, использованию памяти. Предоставить отчёт.

Параметры вычислительного узла, на котором выполнялось тестирование



Краткое описание тестируемых алгоритмов

1. **Breadth First Search (BFS)** — алгоритм обхода графа в ширину. Граф — это структура из «вершин» и «ребер», соединяющих между собой вершины. По ребрам можно передвигаться от одной вершине к другой, и BFS делает это поуровнево: сначала проходит по всем ближайшим от начальной точки вершинам, потом спускается глубже.   
   Выглядит это так: алгоритм начинается в заранее выбранной вершине и сначала «посещает» и отмечает всех соседей этой вершины. Потом он переходит к соседям посещенных вершин, затем — дальше по тому же принципу.   
     
   Основные шаги алгоритма BFS:

* отметить начальную вершину как посещенную и поместить ее в очередь
* извлечь вершину из очереди
* поместить все ее смежные вершины, которые еще не были посещены, в очередь и отметить их как посещенные
* повторять пункты 2, 3, пока очередная вершина не окажется искомой или очередь не опустеет

BFS имеет сложность О (V + E), где V – количество вершин, E – количество рёбер.

1. **Depth First Search (DFS)** — алгоритм обхода графа в глубину. Граф — это структура из «вершин» и «ребер», соединяющих между собой вершины. По ребрам можно передвигаться от одной вершине к другой. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину.

Основные шаги алгоритма DFS:

* кладём в стек начальную вершину
* извлекаем вершину из стека
* если вершина не посещена – отмечаем ее как посещенную и кладем в стек смежные с ней вершины, которые еще не были посещены
* повторять пункты 2, 3, пока очередная вершина не окажется искомой или стек не опустеет

DFS имеет сложность О (V + E), где V – количество вершин, E – количество рёбер.

1. **Алгоритм Дейкстры** — это алгоритм нахождения кратчайшего пути между двумя вершинами во взвешенном графе с дугами неотрицательного веса. Он начинает с одной из начальных вершин и постепенно находит кратчайшие пути до всех остальных вершин графа.

Основные шаги алгоритма Дейкстры:

* устанавливаем начальной вершине, из которой ищем кратчайший путь, вес равным 0, а всем остальным вершинам — бесконечность
* все вершины графа помечаются как не посещенные
* из множества не посещенных вершин выбираем вершину с наименьшим известным весом и отмечаем ее как посещенную
* для всех смежных с выбранной вершиной вершин, кроме посещенных, рассмотрим новую длину пути, равную сумме значений текущего веса выбранной вершины и длины ребра, соединяющего ее со смежной ей вершиной
* если полученное значение длины меньше значения веса соседа, заменяем вес полученным значением длины
* рассмотрев всех соседей, помечаем выбранную вершину как посещённую
* повторяем пункты 2–6, пока все вершины не будут посещены

Алгоритм Дейкстры имеет сложность О(V^2)

1. **Алгоритм Беллмана —** **Форда** — алгоритм поиска кратчайшего [пути](https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D1%82%D1%8C_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) во [взвешенном графе](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B2%D0%B5%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84). Аалгоритм находит кратчайшие пути от одной [вершины](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D1%88%D0%B8%D0%BD%D0%B0_(%D0%B3%D1%80%D0%B0%D1%84)) графа до всех остальных. В отличие от [алгоритма Дейкстры](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B), алгоритм Беллмана — Форда допускает [рёбра](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B1%D1%80%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) с отрицательным [весом](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%81_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)).   
   Как и в других задачах динамического программирования, алгоритм вычисляет кратчайшие пути снизу вверх. Сначала он вычисляет самые короткие расстояния, то есть пути длиной не более, чем в одно ребро. Затем он вычисляет кратчайшие пути длиной не более двух ребер и так далее. После i-й итерации внешнего цикла вычисляются кратчайшие пути длиной не более i ребер. В любом простом пути может быть максимум |V|-1 ребер, поэтому внешний цикл выполняется именно |V|-1 раз. Идея заключается в том, что если мы вычислили кратчайший путь с не более чем i ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем i + 1 ребрами.

Основные шаги алгоритма Беллмана-Форда:

* Инициализируем расстояния от исходной вершины до всех остальных вершин, как бесконечные, а расстояние до самой исходной вершины примем равным 0. Создаем массив dist[] размера |V| со всеми значениями равными бесконечности, за исключением элемента dist[src], где src — исходная вершина.
* Вторым шагом вычисляем самые короткие расстояния. Следующие шаги нужно выполнять |V|-1 раз, где |V| — число вершин в данном графе.
  + Для каждого ребра u-v:  
    Если dist[v] > dist[u] + вес ребра uv, то обновляем dist[v]  
    dist [v] = dist [u] + вес ребра uv
* На этом шаге сообщается, присутствует ли в графе цикл отрицательного веса. Для каждого ребра u-v необходимо выполнить следующее:
  + Если dist[v] > dist[u] + вес ребра uv, то в графе присутствует цикл отрицательного веса.

Алгоритм Беллмана-Форда имеет сложность O(|V|\*|E|).

Результаты

Алгоритмы были протестированы на разных типах (лучшие, худшие, случайные) и объемах (100, 500, 1000 вершин) данных. После многократных тестовых запусков было выявлено, что лучшими входными данными для этих алгоритмов являются графы, не содержащие ребер, а худшими – полные графы.

**Затраты времени на лучших данных (в секундах)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.00010246 | 8.525e-05 | 8.4834e-05 |
| **DFS** | 0.00010425 | 7.7333e-05 | 7.7333e-05 |
| **Dijkstra** | 0.0018208 | 0.0084315 | 0.01715 |
| **Bellman-Ford** | 0.0019737 | 0.0095449 | 0.019321 |

**Затраты времени на худших данных (в секундах)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.091328 | 2.4476 | 9.9151 |
| **DFS** | 0.030436 | 0.78755 | 3.2755 |
| **Dijkstra** | 0.058923 | 1.5586 | 6.724 |
| **Bellman-Ford** | 0.19173 | 4.9776 | 20.553 |

**Затраты времени на случайных данных (в секундах)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.013104 | 0.34538 | 1.4395 |
| **DFS** | 0.010251 | 0.17885 | 0.71113 |
| **Dijkstra** | 0.019131 | 0.34997 | 1.1617 |
| **Bellman-Ford** | 0.044638 | 1.0571 | 4.2256 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.002104 | 0.00064 | 0.00064 |
| **DFS** | 0.00092 | 0.000312 | 0.000312 |
| **Dijkstra** | 0.010248 | 0.036084 | 0.081464 |
| **Bellman-Ford** | 0.007968 | 0.036084 | 0.081464 |

**Затраты памяти на лучших данных (в Кб)**

**Затраты памяти на худших данных (в Кб)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.049624 | 1.0548 | 4.1467 |
| **DFS** | 0.022816 | 0.08256 | 0.13059 |
| **Dijkstra** | 0.016688 | 0.0915 | 0.19244 |
| **Bellman-Ford** | 0.007032 | 0.03654 | 0.10908 |

**Затраты памяти на случайных данных (в Кб)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм** | 100 вершин | 500 вершин | 1000 вершин |
| **BFS** | 0.01144 | 0.13144 | 0.47217 |
| **DFS** | 0.011648 | 0.047032 | 0.05532 |
| **Dijkstra** | 0.015376 | 0.075284 | 0.11707 |
| **Bellman-Ford** | 0.007032 | 0.036084 | 0.081464 |

Обоснование результатов

Основываясь на результатах, мы можем видеть, что алгоритмы BFS и DFS имеют схожее время выполнения, при этом DFS немного быстрее. Алгоритм Дейкстры, как правило, быстрее алгоритма Беллмана-Форда, но разница во времени выполнения более заметна для больших графов.

Для графов наилучшего случая (графов без ребер) алгоритмы BFS и DFS будут принимать O(1), поскольку у начальной вершины нет соседей, и весь путь будет состоять из одной (начальной) вершины. Алгоритмы Дейкстры и Беллмана-Форда будут принимать O(V), так как они посетят только начальную вершину, но им необходимо инициализировать расстояния до всех вершин равными бесконечности, а затем установить расстояние до начальной вершины равным нулю.

Для графов наихудшего случая (где присутствуют все возможные ребра) алгоритм Беллмана-Форда занимает намного больше времени, чем алгоритм Дейкстры, поскольку алгоритм Дейкстры не работает с ребрами с отрицательным весом, в то время как алгоритм Беллмана-Форда работает. Это означает, что Дейкстра может предполагать, что кратчайший путь к вершине не содержит ребер с отрицательным весом, что позволяет ему оптимизировать свой поиск. Напротив, алгоритм Беллмана-Форда должен учитывать возможность ребер с отрицательным весом и ему приходится выполнять больше итераций, чтобы сходиться к правильным расстояниям. Касательно памяти алгоритм Дейкстры требует память для хранения информации о посещенных вершинах, приоритетной очереди для выбора вершины с наименьшим весом и структуры данных для хранения кратчайших путей. Беллман-Форд требует памяти для хранения приоритетной очереди для выбора вершины с наименьшим весом и структуры данных для хранения кратчайших путей. Поэтому получилось, что в конкретной реализации алгоритм Беллмана-Форда использует меньше памяти, чем алгоритм Дейкстры, однако сильная разница заметна только в худших сценариях.

На производительность может повлиять выбор структуры данных, используемой в DFS и BFS. DFS использует стек для отслеживания посещаемых узлов, в то время как BFS использует очередь (реализованную как deque) для посещения узлов в порядке возрастания ширины. В худшем случае, когда граф представляет собой полный граф со всеми возможными ребрами, BFS должна исследовать все узлы в каждом слое, прежде чем переходить к следующему слою. Это может привести к тому, что deque станет очень большим и потребует много памяти для хранения всех узлов. В отличие от этого, DFS потребуется только сохранить путь от начального узла к текущему узлу в верхней части стека. В результате в этом сценарии DFS может быть более эффективной с точки зрения использования памяти и более быстрой, чем BFS.

Для случайно сгенерированных графиков время выполнения варьируется в зависимости от вероятности существования ребер.

Анализ результатов

Причина, по которой BFS и DFS могут по-разному работать с одними и теми же данными, заключается в том, что эти функции используют разные стратегии для обхода графика.  
В зависимости от структуры графика один алгоритм может быть быстрее или эффективнее другого. Например, если график глубокий, но не очень широкий, DFS может быть быстрее, потому что он может глубоко исследовать его перед возвратом. Напротив, если график неглубокий, но широкий, BFS может быть быстрее, потому что он может быстро охватить все узлы на каждой глубине, прежде чем переходить к следующей глубине.

Алгоритм Дейкстры прекращает исследование вершины, как только она находит кратчайший путь к этой вершине, в то время как алгоритм Беллмана-Форда продолжает исследовать вершины до тех пор, пока не найдет кратчайший путь ко всем вершинам. Таким образом, алгоритм Дейкстры может быть быстрее, если есть много вершин, до которых можно добраться из начальной вершины короткими путями, в то время как алгоритм Беллмана-Форда может быть быстрее, если есть много вершин, до которых можно добраться из начальной вершины длинными путями.

С точки зрения стоимости памяти, как BFS, так и DFS имеют наихудшую пространственную сложность O(V), где V - количество вершин в графе. Это связано с тем, что они оба используют структуры данных для отслеживания посещенных вершин и сохранения вершин в очереди или стеке.

Для алгоритма Дейкстры пространственная сложность в наихудшем случае также равна O(V), поскольку ему необходимо хранить информацию о посещенных вершинах, приоритетной очереди для выбора вершины с наименьшим весом и структуры данных для хранения кратчайших путей.

Алгоритм Беллмана-Форда имеет пространственную сложность O (V), поскольку ему необходимо хранить приоритетную очередь для выбора вершины с наименьшим весом и структуры данных для хранения кратчайших путей.

В целом, стоимость памяти для каждой функции зависит от размера графика и конкретной реализации. Однако все они имеют пространственную сложность не более O (V), поэтому их можно применять к графам разумного размера без нехватки памяти.