Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

Анализ сложности алгоритма Беллмана-Форда

Работу выполнил Панюшин Даниил Васильевич группа 19.Б12-пу

- *Описание алгоритма*

Алгоритм Беллмана-Форда ищет длины кратчайших путей от исходной вершины до всех остальных вершин в взвешенном (ориентированном или нет) графе. Алгоритм применим также и к графам, содержащим рёбра отрицательного веса. При наличии отрицательного цикла, кратчайшего пути от некоторых вершин может не существовать (так как вес кратчайшего пути должен быть равен минус бесконечности).

Алгоритм носит имя двух американских учёных: Ричарда Беллмана и Лестера Форда. Форд фактически изобрёл этот алгоритм в 1956 г. при изучении другой математической задачи, подзадача которой свелась к поиску кратчайшего пути в графе, и Форд дал набросок решающего эту задачу алгоритма. В 1958 г. Беллман опубликовал статью, посвящённую конкретно задаче нахождения кратчайшего пути, в которой он чётко сформулировал данный алгоритм в современном виде.

Алгоритм вычисляет кратчайшие пути снизу-вверх. Сначала он вычисляет самые короткие расстояния (длиной в одно ребро), а затем кратчайшие пути длиной не более двух ребер и так далее. В простом пути может быть максимум $|V|-1$ ребер, поэтому внешний цикл выполняется именно $|V|-1$ раз, где $|V|$- мощность множества вершин графа. Если вычислить кратчайший путь с не более чем i ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем $i + 1$ ребрами.

- Математический анализ алгоритма

Алгоритм выполняет $|V|-1$ итерацию, на каждой из которых происходит релаксация рёбер $|E|$ (релаксацией ребра (u, v) называется уменьшение значения d[v] до d[u] + w, если второе значение меньше первого, где d[x] –расстояние от вершины-источника до ребра x, а w – расстояние (вес) ребра (u, v)). В итоге получаем $O(|V||E|)$ операций в худшем случае ($|E|$ - мощность множества ребер графа).

- Входные и выходные данные

Объём входных данных: $O(|V|+|E|)$

В данной работе используется количество вершин (V), равное от 10 до 500 (включительно) с шагом 10.

Затраты памяти: $O(|V|)$

Выходные данные (могут быть нескольких видов):

  - для каждой вершины v исходного графа – последнее ребро, лежащее на кратчайшем пути от вершины u к v, или соответствующая вершина w;
  - для каждой вершины v исходного графа – суммарный вес кратчайшего пути от от вершины u к v.

Таким образом объём выходных данных: $O(|V|)$.

- Единицы измерения трудоёмкости

В рамках данной работы для измерения трудоемкости используется как время работы алгоритма, так и количество операций, которое он выполняет.

- Способ генерации входных данных

Используются полные графы со случайными весами на рёбрах (диапазон значений весов возможно изменить – по умолчанию от 1 до 100). Величины весов не влияют на сложность алгоритма.

Так как при анализе используются полные графы (в которых $|V|-1$ ребро), то сложность алгоритма $O(|V|^2)$. Сложность не зависит от начальной точки, так как граф полный.

Генерация полного графа происходит в конструкторе класса, представляющего граф.

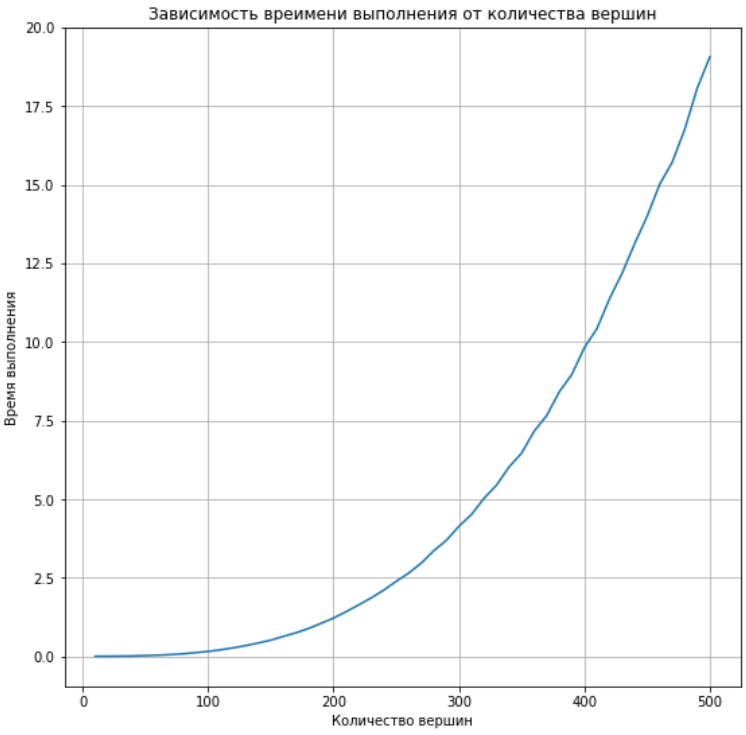
```
# Class to represent a graph
class Graph:

    def __init__(self, vertices, generate_complete_graph=False, min_weight=1, max_weight=100):
        self.V = vertices # No. of vertices
        self.graph = [] # default dictionary to store graph
        if generate_complete_graph:
            combinations = itertools.combinations(range(vertices), 2)
            for edge in combinations:
                self.addEdge(edge[0], edge[1], randint(min_weight, max_weight))
```

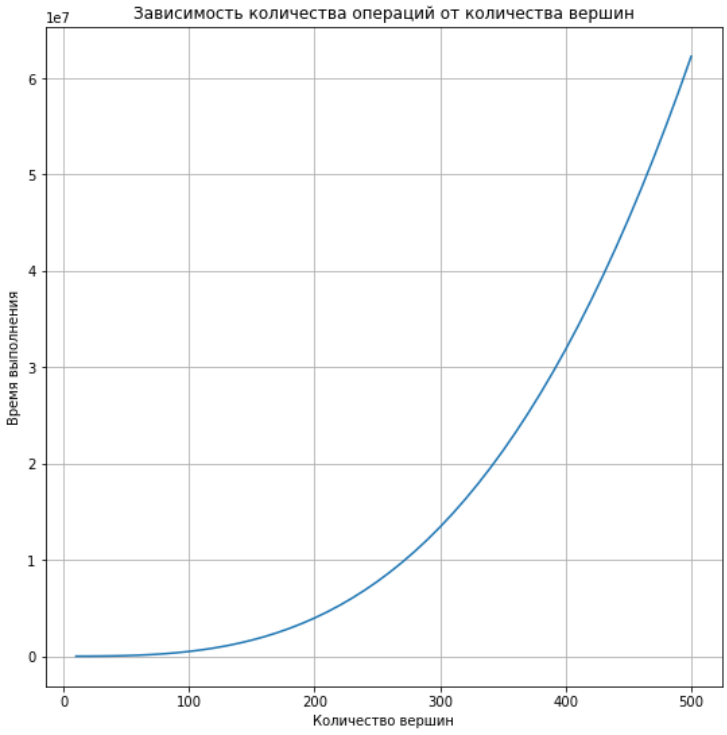
В каждом эксперименте создаётся новый граф с заданным количеством вершин.

- Код программы

Находится в репозитории [GitHub]

- Эксперименты





Отношение времени выполнения при 2n и n вершинах

Количество вершин, n	Время выполнения, сек	Отношение 2n к n
20	0:00:00.002000	5.0
30	0:00:00.003999	8.752188047011753
40	0:00:00.010000	7.8
50	0:00:00.020000	7.7
60	0:00:00.035000	7.657142857142857
70	0:00:00.053000	7.943415094339623
80	0:00:00.078000	8.08974358974359
90	0:00:00.115001	7.669472439370092
100	0:00:00.154000	7.883123376623377
110	0:00:00.205000	7.951214634146342
120	0:00:00.268000	7.843287313432835
130	0:00:00.337999	7.831390033698325
140	0:00:00.421001	7.973857544281367
150	0:00:00.511000	8.08806457925636
160	0:00:00.631000	7.958797147385104
170	0:00:00.745000	8.081880536912752
180	0:00:00.881997	8.11794031045457
190	0:00:01.043998	8.055571945540125
200	0:00:01.214001	8.085658084301413
210	0:00:01.416000	8.028954802259888
220	0:00:01.629999	8.04724604125524
230	0:00:01.854000	8.094929881337647
240	0:00:02.102001	7.972403914175112
250	0:00:02.381002	8.004613183861249

Отношение количества операций при 2n и n вершинах

Количество вершин, n	Количество операций	Отношение 2n к n
20	3657	8.347552638774951
30	12684	8.24787133396405
40	30527	8.186785468601565
50	60141	8.154353934919605
60	104616	8.12611837577426
70	166852	8.1093064512226237
80	249918	8.095351275218272
90	356755	8.085366708245154
100	490411	8.07693139020128
110	653851	8.070211714901408
120	850122	8.064339000755187
130	1082169	8.059580342811152
140	1353054	8.05517148613433
150	1665700	8.051632947109324
160	2023174	8.048438740315959
170	2428379	8.045793099017905
180	2884495	8.043195082674783
190	3394315	8.041007979518696
200	3961016	8.038951622513013
210	4587410	8.037232556061046
220	5276716	8.035482864721164
230	6031809	8.033929787896135
240	6855672	8.032551440617345
250	7751246	8.031383212453843

Из данных, предоставленных на таблице видно, что при увеличении количества вершин вдвое, время выполнения и число операций увеличиваются в восемь раз.

Результаты:

Судя по полученным графикам и отношению результатов эксперимента при 2n и n вершинах можно сделать вывод, что время выполнения и количество операций алгоритма зависит от количества вершин графа нелинейно т.е. сложность алгоритма равна *O(n^3)*. Она отличается от теоретической сложности так как для экспериментов используются полные графы, а значит количество рёбер каждого графа равно n (n-1) / 2 (n – количество вершин). Таким образом сложность алгоритма равна *O(n^2 (n - 1) / 2) = O(n^3)*.

- Характеристики использованной среды и оборудования.
  - Вычисления произведены в среде Jupyther Notebook.
  - Версия Python: 3.7.11

    - Процессор: AMD Ryzen 7 3700X
- Список литературы
    - R. Bellman: On a Routing Problem // Quarterly of Applied Mathematics. 1958. Vol 16, No. 1. C. 87-90, 1958.
    - L. R. Ford, Jr., D. R. Fulkerson. Flows in Networks, Princeton University Press, 1962.
    - https://e-maxx.ru/algo/ford_bellman
    - https://algowiki-project.org/ru/Алгоритм_Беллмана-Форда
    - https://habr.com/ru/company/otus/blog/484382/