МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Построение и анализ алгоритмов»

Тема: «Коммивояжер (TSP)»

Студент группы 1304 _____ Завражин Д.Г.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы

Изучить и на практике освоить подходы к нахождению точного решения задачи коммивояжёра.

Задание

Дана карта городов в виде ассиметричного, неполного графа $G = (V, E)$, где V ($|V| = nV$) – это вершины графа, соответствующие городам; E ($|E| = m$) – это ребра между вершинами графа, соответствующие путям сообщения между этими городами. Каждому ребру m_{ij} (переезд из города i в город j) можно сопоставить критерий выгодности маршрута (вес ребра) равный w_i (натуральное число $[1, 1000]$), $m_{ij} = \infty$, если $i = j$. Если маршрут включает в себя ребро m_{ij}, то $x_{ij} = 1$, иначе $x_{ij} = 0$. Требуется найти минимальный маршрут (минимальный гамильтонов цикл).

Входные параметры: Матрица графа.

Выходные параметры: Кратчайший путь, вес кратчайшего пути, скорость решения задачи: [1, 2, 3, 4, 1], 4, 0mc Задача должна решаться на размере матрицы 20x20 не дольше 3 минут в среднем.

1 Подход к решению задачи

В качестве подкода к решению данной задачи вместо применения перебора с возвратом (факториальная временная сложность) был реализован алгоритм *Беллмана-Хелда-Карпа*, находящий решение задачи коммивояжёра за *экспоненциальное* время – $\Theta\left(n^2 2^n\right)$, но также имеющий и экспоненциальную сложность по памяти.

Так как по условию допустимые веса принадлежат отрезку $[1, 1000]$, то созданная программа интерпретирует все остальные числовые значение как указание на отсутствие соответствующей ему дуги, в том числе и нулевой вес. Эта особенность применяется при генерации матриц в двух составленных скриптах – в них отсутствие дуги обозначается как 0.

В генерируемых матрицах строка соответствует исходному пункту, а столбец – пункту назначения. Измеренное время приводится в *наносекундах*.

2 Использованные функции и структуры данных

Программа написана на парадигме объектно-ориентированного программирования. Основная логика заключена в классе *TripPlanner*, конструктор кото-

рого инициализирует все необходимые для поиска решения переменные, а сам поиск начинается посредством вызова метода *plan()*.

В классе *TripPlanner* имеется вспомогательный вложенный класс *Map*, реализующий интерфейс работы с графом с взвешенными дугами. У класса *Map* имеются следующие методы:

- *Weight get_weight(Location x, Location y) const;* – принимает задающие дугу узлы и возвращает соответствующий ей вес;
- *Weight &get_weight(Location x, Location y;* – принимает задающие дугу узлы и возвращает указатель на соответствующий ей вес.

Как видно, все они являются вспомогательными.

В свою очередь, класс *TripPlanner* содержит следующие методы:

- *Map::Cardinality arity(Permutation permutation);* – Принимает кодирующее своими битами подмножество узлов графа беззнаковое целое число и возвращает количество содержащихся в кодируюмом подмножестве узлов;
- *void use_bellman_held_karp_algorithm()* – Не принимает параметров и не возвращает значения, реализует построение минимального гамильтонова цикла по алгоритму Беллмана-Хелда-Карпа;
- *TripPlanner &plan()* – Не принимает параметров, делегирует построение пути методу *use_bellman_held_karp_algorithm()* с измерением затрачиваемого на построение решения времени, затем возвращает ссылку на хранящий построенное решение объект класса *TripPlanner*.

Полный исходный код программы представлен в Листинге 3 в Приложении А.

3 Проверка на корректность

Для проверки работоспособности основной программы были также составлены два направленные на генерацию матриц скрипта на языке программирования *Python*. Первый из них, осуществляющий генерацию матрицы заданного вида, приведён в Листинге 1. Он генерирует матрицу на основе переданных ему аргументов командной строки с выводом результата в *stdout*.

Листинг 1 — Содержащийся в файле *full_graph.py* исходный код

```
#!/usr/bin/python3
from sys import argv
from random import randint

# The script has four parameters:
# (1) The size of a matrix to be generated;
# (2) What to generate:
```

```
#  .  A hollow matrix
#  M  A regular matrix
#  U  An upper triangular matrix
#  L  A lower triangular matrix
# (3) The weight for non-diagonal elements. It can be:
#  -- A number - all weights are the same
#  -- A '\*" - all weights are random, some vertices being disconnected
#  -- A '." - all weights are random, no vertices being disconnected
# (4) If applicable, max possible weight

cardinality = int(argv[1])
mode = argv[2]
weight = argv[3]

condition = lambda i, j: i != j
if mode == 'M':
    condition = lambda i, j: True
elif mode == 'U':
    condition = lambda i, j: i <= j
elif mode == 'L':
    condition = lambda i, j: i >= j

generate_weight = lambda: weight
if weight == '.':
    generate_weight = lambda: randint(1, int(argv[4]))
elif weight == '*':
    generate_weight = lambda: randint(0, int(argv[4]))

print(cardinality)
for i in range(cardinality):
    for j in range(cardinality):
        print(generate_weight() if condition(i, j) else 0, end=" ")
    print()
```

Рассмотрим применение реализованного алгоритма к ряду случайно сгенерированных квадратных матриц размером 20×20:

```
> clang++ main.cpp && ./full_graph.py 20 . 9 | tee e.dat | ./a.out; cat e.dat
[1, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 180, 4103906067ns
20
0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0
```

```
> clang++ main.cpp && ./full_graph.py 20 . . 9 | tee e.dat | ./a.out; cat e.dat
[1, 17, 5, 4, 18, 15, 3, 12, 13, 2, 16, 8, 20, 6, 7, 19, 9, 10, 14, 11, 1], 23, 6958291035ns
20
0 7 8 9 5 3 5 5 7 5 4 5 7 5 2 7 1 1 9 7
4 0 5 2 8 8 9 4 8 4 8 3 3 9 3 1 4 6 6 7
```

```
1 1 0 8 5 2 5 9 4 5 9 1 4 3 4 9 6 4 8 5
8 2 3 0 2 1 9 9 5 4 9 7 6 9 2 2 4 1 6 6
4 6 7 1 0 3 5 2 8 4 4 2 3 3 4 5 8 4 3 3
1 1 6 3 1 0 1 8 3 3 6 5 7 6 3 1 5 8 6 1
4 8 1 9 9 5 0 4 2 8 9 4 3 2 9 4 5 9 1 4
3 1 1 3 2 2 1 0 6 8 4 8 5 6 4 6 2 4 3 1
4 6 3 9 2 2 2 5 0 1 3 7 7 9 1 4 5 8 8 6
2 1 1 4 3 2 4 1 9 0 4 1 6 1 9 6 6 6 1 4
1 1 1 6 4 7 1 4 4 7 0 3 9 4 4 6 3 5 8 5
5 7 3 7 9 1 2 3 7 2 9 0 1 4 1 3 7 3 7 9
2 1 6 3 8 6 6 9 3 5 8 1 0 6 4 1 8 6 9 9
6 8 6 8 5 2 6 1 7 1 3 4 7 0 5 2 3 6 3 6
4 6 2 6 7 3 4 3 7 6 6 6 9 9 0 3 5 8 4 2
6 6 8 1 3 6 2 1 8 4 4 6 5 9 3 0 9 3 7 4
1 7 8 2 1 7 5 2 1 7 3 1 2 4 7 7 0 1 4 7
5 2 4 9 7 6 6 8 5 3 5 3 9 3 1 5 7 0 5 9
9 4 1 8 9 4 5 9 1 6 8 4 8 9 8 5 1 3 0 6
5 7 3 6 4 1 3 7 7 9 4 8 7 4 6 5 9 2 4 0
```

```
> clang++ main.cpp && ./full_graph.py 20 . \* 1000 | tee e.dat | ./a.out; cat e.dat
[1, 9, 3, 2, 20, 10, 5, 17, 19, 14, 16, 11, 8, 7, 4, 18, 6, 15, 13, 12, 1], 1560, 7582868427ns
20
0 394 850 825 623 539 685 680 109 126 621 660 349 997 906 42 129 423 845 480
779 0 934 931 188 277 332 889 476 491 453 42 561 952 463 512 493 464 711 81
9 42 0 892 361 128 808 606 701 74 367 794 164 294 448 506 663 486 644 81
968 904 897 0 448 659 679 561 890 556 609 518 733 900 742 886 479 236 308 645
143 853 594 679 0 629 489 740 325 504 252 938 200 562 819 48 62 965 946 966
73 333 666 53 715 0 565 541 387 460 310 446 638 792 25 162 486 373 556 604
369 557 752 119 885 941 0 944 892 107 89 483 455 408 517 570 775 862 718 521
822 527 90 496 764 358 97 0 691 701 317 451 524 113 652 544 812 211 934 84
562 726 2 530 562 372 355 926 0 195 479 66 550 612 800 630 657 546 345 345
354 314 329 891 84 828 147 818 738 0 940 380 679 37 221 115 804 444 90 402
217 489 90 15 962 420 224 42 224 368 0 586 113 446 111 608 357 195 812 6
39 874 214 758 596 674 341 220 428 22 78 0 945 630 604 778 887 330 919 741
411 892 491 101 591 402 911 303 609 80 406 49 0 795 3 581 431 906 798 677
305 543 990 604 953 362 926 244 493 214 189 294 466 0 738 13 811 432 383 378
354 907 771 538 347 968 896 429 542 809 124 654 183 185 0 841 248 670 767 397
962 837 769 816 197 822 827 505 692 26 55 100 805 4 70 0 17 587 160 472
98 627 87 252 223 416 604 478 167 691 892 722 800 377 846 995 0 540 101 990
745 694 282 896 257 45 645 44 446 88 555 770 46 905 49 448 889 0 268 89
586 690 933 671 456 320 352 706 873 354 311 230 474 135 321 44 544 237 0 302
121 675 187 299 302 134 386 877 780 41 413 276 408 455 44 361 610 341 557 0
```

```
> lang++ main.cpp && ./full_graph.py 20 M \* 1200 |  tee e.dat | ./a.out; cat e.dat
[1, 14, 11, 13, 6, 17, 3, 9, 18, 4, 16, 7, 20, 5, 8, 10, 19, 12, 15, 2, 1], 2285, 7253774945ns
20
24 100 242 913 91 189 629 849 902 443 914 717 590 139 1146 926 371 128 979 86
152 1018 1140 493 481 938 452 576 599 159 1030 103 247 385 1175 1015 1170 594 33 723
825 454 197 856 444 505 559 307 208 876 623 477 129 250 1181 1159 157 225 487 938
961 668 235 877 892 509 849 383 576 316 192 465 1002 223 202 52 1110 857 359 727
333 1121 1 1078 515 567 892 384 1070 1187 250 431 1154 1133 719 1135 1150 269 542 337
800 746 683 1140 367 725 842 810 1086 833 1051 295 163 1194 194 858 9 633 93 677
465 919 297 462 338 574 778 860 712 1170 913 731 286 1107 747 137 787 376 929 43
628 148 660 1015 703 575 961 1036 559 69 58 1129 869 874 867 219 415 261 416 944
1134 274 726 783 471 499 439 346 145 1139 504 1126 648 436 643 571 1025 137 1026 363
335 1057 755 830 714 536 1182 1193 249 1110 1123 440 785 1185 542 872 1182 950 137 228
1190 1006 936 170 785 321 1165 524 766 443 626 304 100 442 351 459 430 100 772 455
1183 933 790 10 691 70 1015 252 644 27 976 669 176 578 33 76 934 733 725 144
570 304 740 1063 644 29 467 1061 432 876 992 332 549 898 395 131 84 535 546 695
244 516 172 633 228 124 572 493 1016 630 82 154 819 689 564 1136 487 607 1080 672
712 78 666 324 701 81 828 182 464 967 45 104 373 346 1042 791 384 1082 962 647
356 946 822 873 274 512 395 785 161 386 1107 303 192 674 219 446 1085 926 3 514
389 961 43 1093 560 740 688 534 454 370 872 891 1111 461 1043 207 977 320 894 78
471 1096 848 78 174 1104 839 500 143 906 897 859 664 647 1031 426 532 242 663 611
375 335 1018 121 617 462 406 521 1018 645 1106 101 881 543 144 456 211 900 1051 32
1020 126 1193 582 16 618 871 222 283 815 314 338 418 1176 150 521 341 5 324 1056
```

```
> clang++ main.cpp && ./full_graph.py 20 M \* 1 | tee e.dat | ./a.out; cat e.dat
[1, 19, 18, 17, 12, 20, 16, 15, 13, 11, 10, 14, 6, 9, 7, 8, 5, 3, 4, 2, 1], 20, 3702962214ns
```

```
20
0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 0
1 1 1 1 0 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1
0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1
1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1
1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0
1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1
1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1
1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 0 1 1 0 0
1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 0 0 1
0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0
0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0
1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1
1 1 0 0 1 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0
1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0
1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 0 1 1
1 0 0 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 1
1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 0
1 1 0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 0
0 1 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 1 1 1
1 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0
```

Как можно заметить, среднее время решения задачи для таких матриц не превышает *восьми* секунд. Рассмотрим ряд случаев, когда решения не существует:

```
> clang++ main.cpp && ./full_graph.py 20 . 0 | tee e.dat | ./a.out; cat e.dat
[], No path exists, 937135723ns
20
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
> clang++ main.cpp && ./full_graph.py 20 U \* 1000 |  tee e.dat | ./a.out; cat e.dat
[], No path exists, 1031818190ns
20
271 588 658 847 966 858 243 896 426 17 980 526 698 42 261 760 97 939 134 166
0 89 913 159 837 457 391 591 382 964 162 273 693 869 160 289 105 650 433 714
0 0 362 586 699 267 931 860 878 696 342 805 692 671 450 337 271 484 582 540
0 0 0 269 340 715 876 382 45 790 938 412 955 605 965 29 255 239 297 730
0 0 0 0 466 282 297 96 883 330 557 958 266 394 452 495 521 982 361 879
0 0 0 0 0 781 500 281 399 300 156 713 341 578 823 688 736 563 45 282
0 0 0 0 0 0 981 150 261 958 206 859 993 779 474 730 787 162 411 100
0 0 0 0 0 0 0 831 345 501 264 595 657 194 692 222 475 119 612 695
0 0 0 0 0 0 0 0 525 236 925 558 517 312 404 578 760 474 984 542
0 0 0 0 0 0 0 0 0 282 211 934 990 871 39 354 651 362 622 27
0 0 0 0 0 0 0 0 0 0 279 912 425 821 811 740 890 45 351 486
0 0 0 0 0 0 0 0 0 0 0 646 9 646 935 374 726 570 36 954
```

```
0 0 0 0 0 0 0 0 0 0 0 0 349 330 456 809 335 204 284 211
0 0 0 0 0 0 0 0 0 0 0 0 0 0 547 105 645 798 341 114 582
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 23 81 512 789 477 317
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 575 816 441 76 774
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 880 4 145 266
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 508 431 375
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 54 710
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 712
```

```
> clang++ main.cpp && ./full_graph.py 20 L \* 9 | tee e.dat | ./a.out; cat e.dat
[], No path exists, 950250904ns
20
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 5 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9 7 5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
7 1 7 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 4 8 7 7 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8 4 6 2 5 9 1 0 0 0 0 0 0 0 0 0 0 0 0 0
7 8 2 7 4 9 0 6 0 0 0 0 0 0 0 0 0 0 0 0
8 0 5 2 0 3 7 1 4 0 0 0 0 0 0 0 0 0 0 0
8 2 8 0 0 4 5 1 6 3 0 0 0 0 0 0 0 0 0 0
0 7 7 9 7 3 2 2 2 1 2 0 0 0 0 0 0 0 0 0
2 0 7 4 9 9 2 6 7 9 0 2 0 0 0 0 0 0 0 0
4 0 5 2 4 9 2 5 3 5 0 9 1 0 0 0 0 0 0 0
3 9 1 7 0 1 6 2 9 2 2 9 7 5 0 0 0 0 0 0
3 5 6 3 1 9 5 7 4 4 1 3 4 3 9 0 0 0 0 0
9 3 3 3 9 6 4 3 3 0 1 7 2 4 8 5 0 0 0 0
4 0 0 9 4 9 2 3 3 9 9 0 5 2 9 8 9 0 0 0
2 2 8 0 3 4 0 9 5 9 6 6 2 7 9 5 7 2 0 0
9 7 9 1 7 9 5 2 7 7 0 8 1 6 8 4 3 5 3 0
6 7 5 2 1 4 0 7 1 7 0 4 2 8 8 1 9 8 7 1
```

Можно отметить, что из полученных результатов можно сделать вывод, что на таких графах без гамильтонова цикла программа отрабатывает где-то за *одну* секунду.

Для генерации содержащих только несколько путей матриц был составлен представленный на Листинге 2 скрипт. Он заполняет пустую матрицу заданным количеством возможно пересекающихся путей, а затем выводит результат в *stdout*.

Листинг 2 — Содержащийся в файле *single_path.py* исходный код

```python
#!/usr/bin/python3
from sys import argv, stderr
from random import randint, sample

# The script has four parameters:
# (1) The size of a matrix to be generated;
# (2) The number of path to generate;
# (3) The weight for non-diagonal elements. It can be:
#  -- A number - all weights are the same
#  -- A '.” - all weights are random, no vertices being disconnected
# (4) If applicable, max possible weight

cardinality = int(argv[1])
n_paths = int(argv[2])
weight = argv[3]

generate_weight = lambda: int(weight)
if weight == '.':
```

```
    generate_weight = lambda: randint(1, int(argv[4]))

matrix = [0] * (cardinality * cardinality)

for i in range(n_paths):

    path = [0] + sample(list(range(1, cardinality)), cardinality - 1) + [0]
    for i in range(cardinality):
        if matrix[path[i] * cardinality + path[i + 1]] == 0:
            matrix[path[i] * cardinality + path[i + 1]] = generate_weight()
    weight = sum([matrix[path[i] * cardinality + path[i + 1]] for i in range(cardinality)])
    print(str([p + 1 for p in path]) + ',', weight, file=stderr)

print(cardinality)
for i in range(cardinality):
    for j in range(cardinality):
        print(matrix[i * cardinality + j], end=" ")
    print()
```

При помощи данного скрипта были проведены в том числе и следующие тесты:

```
> clang++ main.cpp && ./single_path.py 20 1 . 9 | tee e.dat | ./a.out; cat e.dat
[1, 11, 18, 10, 13, 6, 9, 3, 15, 16, 12, 8, 14, 5, 19, 20, 4, 17, 2, 7, 1], 108
[1, 11, 18, 10, 13, 6, 9, 3, 15, 16, 12, 8, 14, 5, 19, 20, 4, 17, 2, 7, 1], 108, 960069469ns
20
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0
0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0
0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
> clang++ main.cpp && ./single_path.py 20 2 . 9 | tee e.dat | ./a.out; cat e.dat
[1, 3, 19, 12, 11, 14, 8, 2, 16, 9, 13, 18, 17, 10, 20, 4, 6, 5, 7, 15, 1], 100
[1, 19, 6, 7, 14, 18, 16, 12, 5, 3, 20, 13, 17, 4, 2, 11, 15, 9, 8, 10, 1], 89
[1, 19, 6, 7, 14, 18, 16, 12, 5, 3, 20, 13, 17, 4, 2, 11, 15, 9, 8, 10, 1], 89, 949046900ns
20
0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 6 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 6
0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 6 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 9 1 0 0 0 0 0
0 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 6 0 0 0 0 3 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8
0 0 0 0 0 0 0 0 0 0 0 0 0 1 5 0 0 0 0 0
0 0 0 0 4 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 5 0 0
0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 3 0 0
```

```
4 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 0 0 8 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 1 0 0 0
0 0 0 0 0 5 0 0 0 0 0 2 0 0 0 0 0 0 0 0
0 0 0 9 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0
```

```
clang++ main.cpp && ./single_path.py 20 3 . 9 | tee e.dat | ./a.out; cat e.dat
[1, 10, 8, 13, 7, 4, 14, 17, 19, 20, 11, 9, 12, 15, 6, 18, 3, 2, 16, 5, 1], 103
[1, 13, 2, 12, 7, 15, 18, 4, 20, 17, 9, 6, 19, 8, 5, 10, 16, 3, 11, 14, 1], 72
[1, 16, 19, 15, 3, 2, 14, 10, 17, 5, 18, 4, 8, 20, 9, 12, 7, 6, 13, 11, 1], 111
[1, 13, 2, 12, 7, 15, 18, 4, 20, 17, 9, 6, 19, 8, 5, 10, 16, 3, 11, 14, 1], 72, 994191665ns
20
0 0 0 0 0 0 0 0 0 5 0 0 2 0 0 9 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 0 4 0 7 0 0 0 0
0 5 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 7 0 0 0 0 0 1
7 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 7 2 0
0 0 0 6 0 5 0 0 0 0 0 0 0 0 2 0 0 0 0 0
0 0 0 0 2 0 0 0 0 0 0 0 4 0 0 0 0 0 0 7
0 0 0 0 0 2 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 2 6 0 0 0
6 0 0 0 0 0 0 0 7 0 0 0 0 2 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 4 0 0 0 0 0
0 6 0 0 0 0 4 0 0 0 8 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 1 0 0 0
0 0 2 0 0 5 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 6 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 9 0 0 0 4 0 0 0 0 0 0 0 0 1 0
0 0 3 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0 0 8 0 0 0 0 4
0 0 0 0 0 0 0 0 6 0 9 0 0 0 0 0 8 0 0 0
```

Как видно из приведённого вывода, реализованная программа корректно находит минимальный из нескольких непересекающихся путей. Впрочем, так как скрипт не гарантирует того, что будут заданы именно *непересекающиеся* пути, то его применение к большему количеству задаваемых путей нецелесообразно – в графе с подавляющей вероятностью в связи с пересечением путей найдётся и меньший всех сгенерированных путь. Например:

```
> clang++ main.cpp && ./single_path.py 20 4 . 9 | tee e.dat | ./a.out; cat e.dat
[1, 20, 8, 4, 6, 17, 3, 19, 11, 15, 18, 10, 7, 5, 2, 13, 9, 14, 12, 16, 1], 98
[1, 18, 6, 5, 20, 8, 2, 15, 7, 13, 4, 14, 16, 10, 12, 3, 9, 17, 11, 19, 1], 85
[1, 19, 16, 12, 3, 7, 9, 10, 4, 8, 13, 20, 15, 6, 17, 2, 18, 14, 11, 5, 1], 107
[1, 11, 2, 13, 10, 17, 15, 16, 8, 3, 20, 7, 14, 19, 6, 4, 18, 5, 12, 9, 1], 79
[1, 11, 15, 18, 14, 12, 16, 8, 2, 13, 4, 6, 5, 20, 7, 9, 10, 17, 3, 19, 1], 60, 1354388179ns
20
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 8 4 9
0 0 0 0 0 0 0 0 0 0 0 0 2 0 5 0 0 9 0 0
0 0 0 0 0 0 6 0 6 0 0 0 0 0 0 0 0 0 1 2
0 0 0 0 0 8 0 6 0 0 0 0 0 7 0 0 0 3 0 0
9 9 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 3
0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0
0 0 0 0 4 0 0 0 2 0 0 0 1 7 0 0 0 0 0 0
0 2 7 3 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 6 0 0 0 6 0 0 4 0 0 0
0 0 0 1 0 0 5 0 0 0 0 2 0 0 0 0 2 0 0 0
0 8 0 0 8 0 0 0 0 0 0 0 0 0 6 0 0 0 4 0
0 0 7 0 0 0 0 0 9 0 0 0 0 0 0 8 0 0 0 0
0 0 0 1 0 0 0 0 0 1 6 0 0 0 0 0 0 0 0 7
0 0 0 0 0 0 0 0 0 0 3 6 0 0 0 9 0 0 2 0
```

```
0 0 0 0 0 8 1 0 0 0 0 0 0 0 0 1 0 3 0 0
6 0 0 0 0 0 0 2 0 6 0 6 0 0 0 0 0 0 0 0
0 7 1 0 0 0 0 0 0 0 5 0 0 0 3 0 0 0 0 0
0 0 0 0 4 7 0 0 0 3 0 0 0 2 0 0 0 0 0 0
1 0 0 0 0 7 0 0 0 0 9 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 4 0 0 0 0 0 0 7 0 0 0 0 0
```

```
clang++ main.cpp && ./single_path.py 20 3 . 500 |  tee e.dat | ./a.out; cat e.dat
[1, 14, 4, 7, 17, 6, 12, 5, 11, 13, 10, 18, 8, 16, 9, 15, 19, 2, 20, 3, 1], 5345
[1, 19, 2, 5, 4, 6, 7, 11, 8, 18, 3, 15, 16, 14, 20, 12, 17, 9, 13, 10, 1], 4659
[1, 9, 11, 18, 2, 20, 8, 17, 5, 3, 13, 16, 19, 6, 4, 15, 12, 7, 10, 14, 1], 5072
[1, 19, 2, 20, 8, 16, 14, 4, 6, 7, 17, 9, 15, 12, 5, 11, 18, 3, 13, 10, 1], 4016, 1015018713ns
20
0 0 0 0 0 0 0 0 478 0 0 0 0 263 0 0 0 0 113 0
0 0 0 0 253 0 0 0 0 0 0 0 0 0 0 0 0 0 0 165
248 0 0 0 0 0 0 0 0 0 0 0 149 0 131 0 0 0 0 0
0 0 0 0 0 97 448 0 0 0 0 0 0 0 498 0 0 0 0 0
0 0 299 144 0 0 0 0 0 0 356 0 0 0 0 0 0 0 0 0
0 0 0 139 0 0 56 0 0 0 0 206 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 368 406 0 0 0 0 0 173 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 37 496 155 0 0
0 0 0 0 0 0 0 0 0 0 186 0 450 0 415 0 0 0 0 0
273 0 0 0 0 0 0 0 0 0 0 0 0 51 0 0 0 46 0 0
0 0 0 0 0 0 0 280 0 0 0 0 421 0 0 0 0 64 0 0
0 0 0 0 210 0 360 0 0 0 0 0 0 0 0 0 0 407 0 0 0
0 0 0 0 0 0 0 0 0 174 0 0 0 0 0 75 0 0 0 0
395 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 136
0 0 0 0 0 0 0 0 0 0 0 325 0 0 0 390 0 0 404 0
0 0 0 0 0 0 0 0 466 0 0 0 0 399 0 0 0 0 130 0
0 0 0 0 140 378 0 0 468 0 0 0 0 0 0 0 0 0 0 0
0 237 74 0 0 0 0 258 0 0 0 0 0 0 0 0 0 0 0 0
0 180 0 0 0 232 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 494 0 0 0 0 285 0 0 0 73 0 0 0 0 0 0 0 0
```

Как видно из полученных результатов, такие разреженные матрицы также обрабатываются порядка *одной* секунды.

Выводы

Были изучены и на практике освоены подходы к нахождению точного решения задачи коммивояжёра. Был реализован алгоритм Беллмана-Хелда-Карпа на языке программирования *C++* при использовании парадигмы объектно-ориентированного программирования.

Было многократно замерено затрачиваемое на нахождение решения время. По результатам измерений можно сделать вывод, что разработанная программа вписывается в поставленные в условии трёхминутные рамки, а затрачиваемое в среднем на решение задачи время увеличивается с ростом количества дуг в рассматриваемом графе.

При тестировании был рассмотрен ряд крайних случаев для задающих оринтированный граф матриц, в том числе задающую полный и пустой граф, треугольные и разреженные матрицы. Согласно условию были рассмотрены графы с весами от 1 до 1000 включительно.

Также были отточены навыки составления простых скриптов на языке программирования *Python*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Листинг 3 — Содержащийся в файле *main.cpp* исходный код

```
/** @file */

#include <algorithm>
#include <chrono>
#include <iostream>
#include <map>
#include <vector>

/** 'DISCONNECTED' corresponds to a weight of infinity. */
constexpr unsigned DISCONNECTED = 1 << 18;

/**
 * 'TripPlanner' encapsulates the logic of finding a circuit to the travelling
 * salesman problem (TSP) using an approach based on the Bellman-Held-Karp
 * algorithm. It reads the required data from stdin.
 */
class TripPlanner
{
private:
   /**
    * 'Map' is an auxiliary class used to represent a bipartite graph with
    * weighed edged.
    */
   class Map
   {
   public:
       /** A type to denote a number ov verrtices. */
      using Cardinality = unsigned;
      /** A type to denote a vertex as an index. */
      using Vertex = unsigned;
      /** A type to represent a weight of a given edge. */
      using Weight = unsigned;

      /** The number of vertices in the first part. */
      const Cardinality cardinality;
      /** The number of vertices in the first part. */
      const Cardinality second_cardinality;
   private:
      /** The vector of weights associated with vertices. */
      std::vector<Weight> weights;

   public:
      /**
       * This constructor constructs an empty object.
       */
      Map() : Map(0) { }

      /**
       * The constructor with one parameter constructs a graph with n x n
       * potential edges.
       *
       * \param cardinality The number of nodes in the graph.
       */
      Map(Cardinality cardinality)
      : cardinality{cardinality}
      , second_cardinality{cardinality}
      , weights(cardinality * cardinality, DISCONNECTED) { }

      /**
       * The constructor with both parameter constructs a bipartite graph
       * with n x m potential edges.
       *
```

```
63          * @param cardinality The number of nodes in the graph.
64          * @param second_cardinality The number of nodes in the graph.
65          */
66         Map(Cardinality cardinality, Cardinality second_cardinality)
67         : cardinality{cardinality}
68         , second_cardinality{second_cardinality}
69         , weights(cardinality * second_cardinality, DISCONNECTED) { }
70
71         /**
72          * The copy constructor copies the graph.
73          *
74          * @param map The graph to be copied.
75          */
76         Map(const Map &map)
77         : cardinality{map.cardinality}
78         , second_cardinality{map.second_cardinality}
79         , weights{map.weights} { }
80
81         /**
82          * The assignment operator copies the graph.
83          *
84          * @param map The graph to be copied.
85          * @returns Pointer to the created copy.
86          */
87         Map &operator=(const Map &map)
88         {  if(this == &map)
89            {  return *this;  }
90            this->~Map();
91            return *new(this) Map(map);
92         }
93
94         /**
95          * 'get_weight' is a getter for a weight associated with a given edge.
96          *
97          * @param x The first vertex of the edge.
98          * @param y The second vertex of the edge.
99          * @returns A weight associated with the specified vertex.
100         */
101        Weight get_weight(Vertex i, Vertex j) const
102        {  return this->weights[i * this->cardinality + j];  }
103
104        /**
105         * 'get_weight' is a getter/setter for a weight associated with a given
106         * edge.
107         *
108         * @param x The first vertex of the edge.
109         * @param y The second vertex of the edge.
110         * @returns A reight associated with the specified vertex.
111         */
112        Weight &get_weight(Vertex i, Vertex j)
113        {  return this->weights[i * this->cardinality + j];  }
114    };
115
116    /** A variable to hold a given graph. */
117    Map map;
118    /** A variable to hopefully hold a correct circuit. */
119    std::vector<Map::Vertex> circuit;
120    /** A variable to hopefully hold the weight of a correct circuit. */
121    Map::Weight circuit_weight = DISCONNECTED;
122    /** A variable for code benchmarking. */
123    std::chrono::nanoseconds benchmark;
124
125    /** A type to represent a permutation of vertices. */
126    // It would be nice to have a class instead, but all the hassle with pointers
127    // significantly decreases performance (about 1.5 seconds for n = 20,
128    // rendom weights).
129    using Subset = unsigned long long;
```

```
130
131     /**
132      * 'arity' determines the number of elements in a given permutation.
133      *
134      * @param permutation A permutation to find the arity of.
135      * @returns Arity of the given permutation.
136      */
137     Map::Cardinality arity(Subset subset) const
138     {  Map::Cardinality arity = 0;
139        for (; subset; arity++)
140        {  subset &= subset - 1;  }
141        return arity;
142     }
143
144     #define SUBSET
145
146     /**
147      * 'use_bellman_held_karp_algorithm' uses the Bellman-Held-Karp algorithm
148      * to solve the travelling salesman problem in exponential time, but with
149      * higher memory requirements.
150      */
151     void use_bellman_held_karp_algorithm()
152     {
153        const Map::Cardinality variable_vertex_count = this->map.cardinality - 1;
154        Subset set = (1 << variable_vertex_count) - 1;
155        Map paths(variable_vertex_count, set + 1);
156        std::map<std::pair<Subset, Map::Vertex>, Map::Vertex> tags;
157        constexpr Map::Vertex FIXED_VERTEX = 0, SECOND_VERTEX = 0;
158
159        for(Map::Vertex i = SECOND_VERTEX; i < variable_vertex_count; ++i)
160        {  paths.get_weight(1 << i, i) = this->map.get_weight(FIXED_VERTEX, i + 1);  }
161        for(Map::Cardinality arity = 2; arity <= variable_vertex_count; arity++)
162        {  for (Subset subset = 1; subset <= set; ++subset)
163           {  if(this->arity(subset) != arity)
164              {  continue;  }
165              for (Map::Vertex i = SECOND_VERTEX; i < variable_vertex_count; ++i)
166              {  if(not (subset & 1 << i))
167                 {  continue;  }
168                 Subset previous = subset ^ 1 << i;
169                 for (Map::Vertex j = SECOND_VERTEX; j < variable_vertex_count; ++j)
170                 {  if(not (subset & 1 << j))
171                    {  continue;  }
172                    Map::Weight weight = paths.get_weight(previous, j);
173                    weight += this->map.get_weight(j + 1, i + 1);
174                    if(weight < paths.get_weight(subset, i))
175                    {  paths.get_weight(subset, i) = weight;
176                       tags[{subset, i}] = j + 1;
177        }  }  }  }  }
178
179        this->circuit_weight = DISCONNECTED;
180        Map::Vertex vertex = 0;
181        for(int i = FIXED_VERTEX; i < variable_vertex_count; ++i)
182        {  Map::Weight weight = paths.get_weight(set, i);
183           weight += this->map.get_weight(i + 1, FIXED_VERTEX);
184           if(this->circuit_weight > weight)
185           {  this->circuit_weight = weight;
186              vertex = i;
187        }  }
188        if(this->circuit_weight == DISCONNECTED)
189        {  return;  }
190
191        this->circuit.clear();
192        this->circuit.insert(circuit.end(), {FIXED_VERTEX, vertex + 1});
193        while(this->arity(set) != 0)
194        {  this->circuit.push_back(tags[{set, vertex}]);
195           Map::Vertex next_vertex = vertex;
196           vertex = tags[{set, vertex}] - 1;
```

```
197            set ^= 1 << next_vertex;
198         }
199         std::reverse(this->circuit.begin(), this->circuit.end());
200      }
201
202   public:
203      /**
204       * The constructor reads the required data from stdin and initialises the
205       * graph accordingly.
206       */
207      TripPlanner()
208      {  unsigned cardinality;
209         std::cin >> cardinality;
210         this->map = Map(cardinality);
211
212         constexpr unsigned MIN_WEIGHT = 1;
213         constexpr unsigned MAX_WEIGHT = 1000;
214         for(unsigned i = 0; i < cardinality; ++i)
215         {  for(unsigned j = 0; j < cardinality; ++j)
216            {  std::cin >> this->map.get_weight(i, j);
217               if(this->map.get_weight(i, j) < MIN_WEIGHT or
218                  MAX_WEIGHT < this->map.get_weight(i, j))
219               {  this->map.get_weight(i, j) = DISCONNECTED;  }
220      }  }  }
221
222      /**
223       * 'plan' calls the implemented algorithm and benchmarks it.
224       */
225      TripPlanner &plan()
226      {  if(this->map.cardinality == 0)
227         {  return *this;  }
228         auto begin = std::chrono::high_resolution_clock::now();
229         this->use_bellman_held_karp_algorithm();
230         auto time = std::chrono::high_resolution_clock::now() - begin;
231         this->benchmark = std::chrono::duration_cast<std::chrono::nanoseconds>(time);
232         return *this;
233      }
234
235      friend std::ostream &operator<<(std::ostream &out, const TripPlanner &planner);
236   };
237
238   /**
239    * The overloaded operator is used to output a circuit.
240    */
241   std::ostream &operator<<(std::ostream &out, const TripPlanner &planner)
242   {  if(planner.circuit.size() == 0)
243      {  out << "[], ";  }
244      else
245      {  out << '[' << planner.circuit[0] + 1;
246         for(int i = 1; i < planner.circuit.size(); i++)
247         {  out << ", " << planner.circuit[i] + 1;  }
248         out << "], ";
249      }
250      if(planner.circuit_weight == DISCONNECTED)
251      {  out << "No path exists";  }
252      else
253      {  out << planner.circuit_weight;  }
254      return out << ", " << planner.benchmark.count() << "ns";
255   }
256
257   int main()
258   {  return std::cout << TripPlanner().plan() << std::endl, 0;  }
259
```