

Лабораторная работа №13_17

По теме 6.1-6.7: «Алгоритмы, внешняя память, геометрия, нечёткие множества, теория игр, машинное обучение и нейронные сети.»

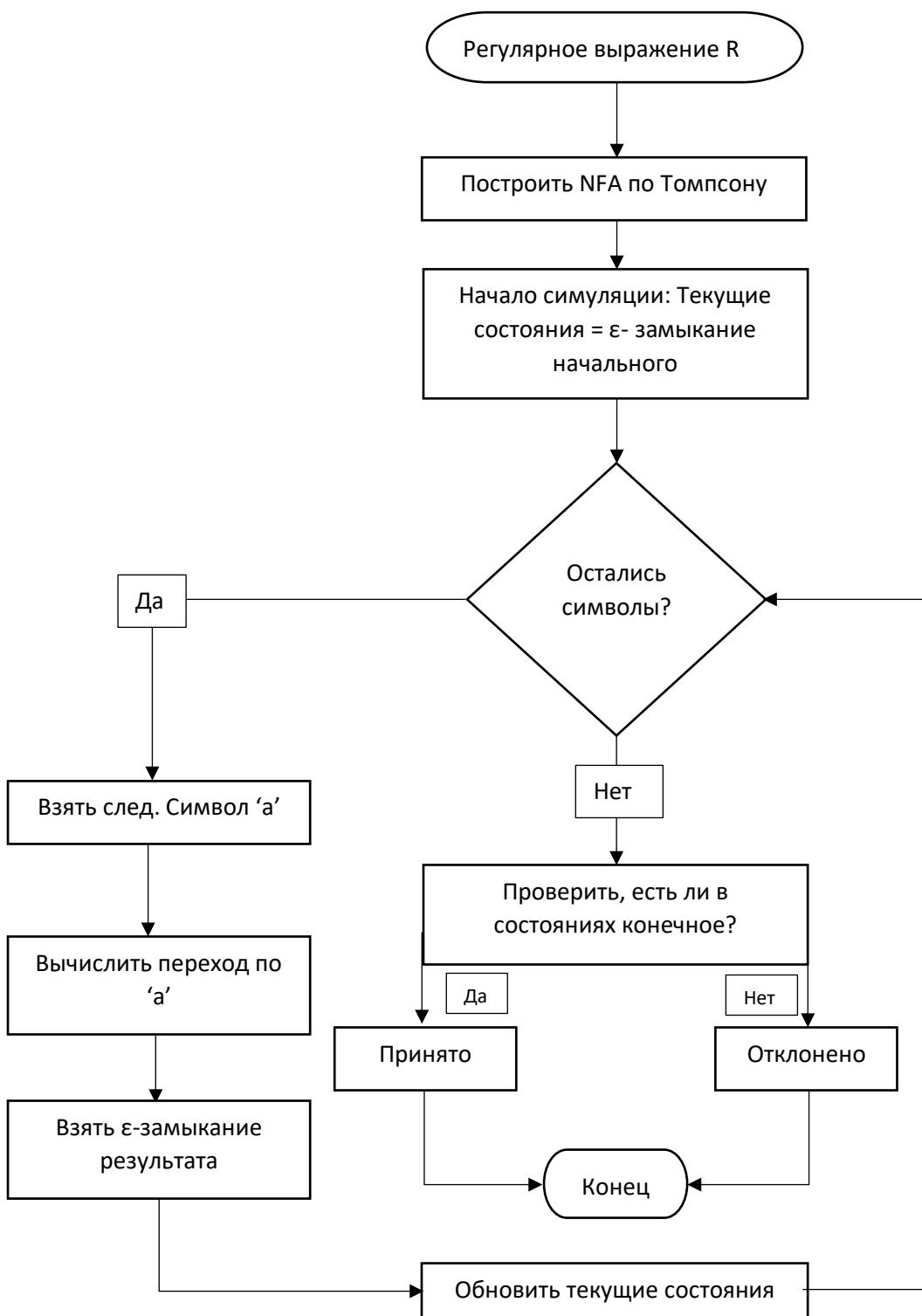
Выполнил студент Гогадзе Арсений Давидович,
1 курса ВШКМиС
Группы ПИ04

Москва, 2025

Предметная область №1: «Автоматы и регулярные выражения — NFA»

Краткое описание области: Конечные автоматы используются для распознавания шаблонов в строках; существуют DFA (детерминированные) и NFA (недетерминированные). Thompson's construction даёт способ превратить регулярное выражение в NFA.

Блок-схема:



Пошаговый анализ:

1. Инициализировать множество текущих состояний стартовым состоянием (и его ϵ -замыканием).
2. Для каждого входного символа: для каждого текущего состояния взять переходы по этому символу; собрать новые состояния; расширить их ϵ -переходами (ϵ -closure).
3. После обработки всех символов проверить, встретилось ли какое-то принимающее состояние. Если да — строка принимается.

Результат:

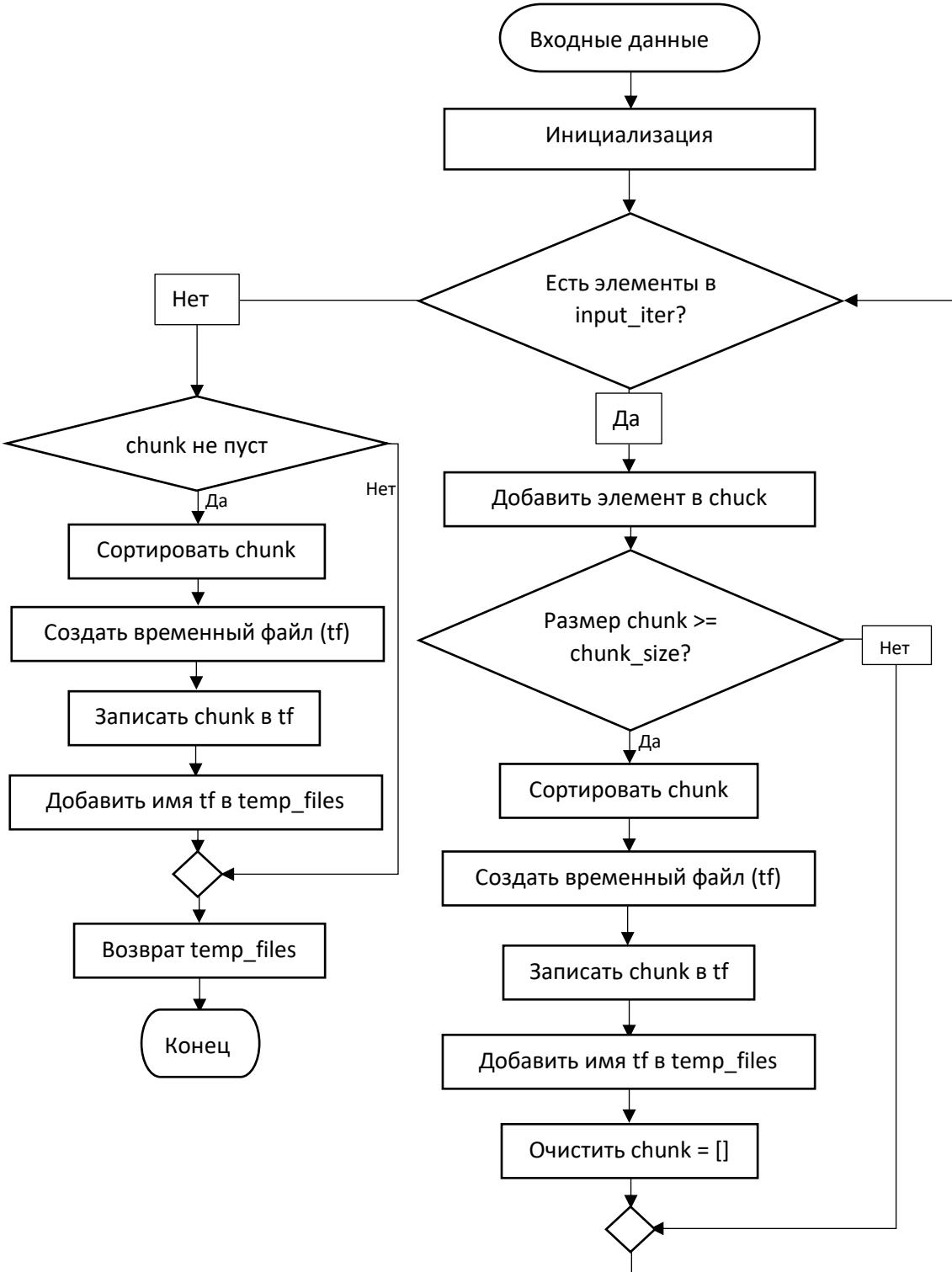
The screenshot shows a code editor with two panes. The left pane contains the Python code for the `nfa_simulator.py` script. The right pane shows the output of the script's execution, which includes the code and its results.

```
1 # nfa_simulator.py
2 from collections import defaultdict, deque
3
4 class SimpleNFA:
5     def __init__(self):
6         self.start = None
7         self.accepts = set()
8         # transitions: state -> symbol -> set(states); epsilon transitions use symbol None
9         self.trans = defaultdict(lambda: defaultdict(set))
10
11     def add_transition(self, frm, sym, to):
12         self.trans[frm][sym].add(to)
13
14     def epsilon_closure(self, states):
15         stack = list(states)
16         closure = set(states)
17         while stack:
18             s = stack.pop()
19             for t in self.trans[s].get(None, []): # None = ε
20                 if t not in closure:
21                     closure.add(t)
22                     stack.append(t)
23         return closure
24
25     def match(self, s):
26         if self.start is None:
27             return False
28         cur = self.epsilon_closure({self.start})
29         for ch in s:
30             nxt = set()
31             for state in cur:
32                 for to in self.trans[state].get(ch, []):
33                     nxt.add(to)
34             cur = self.epsilon_closure(nxt)
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
249
249
250
251
252
253
253
254
255
255
256
257
257
258
259
259
260
261
261
262
263
263
264
265
265
266
267
267
268
269
269
270
271
271
272
273
273
274
275
275
276
277
277
278
279
279
280
281
281
282
283
283
284
285
285
286
287
287
288
289
289
290
291
291
292
293
293
294
295
295
296
297
297
298
299
299
300
301
301
302
303
303
304
305
305
306
307
307
308
309
309
310
311
311
312
313
313
314
315
315
316
317
317
318
319
319
320
321
321
322
323
323
324
325
325
326
327
327
328
329
329
330
331
331
332
333
333
334
335
335
336
337
337
338
339
339
340
341
341
342
343
343
344
345
345
346
347
347
348
349
349
350
351
351
352
353
353
354
355
355
356
357
357
358
359
359
360
361
361
362
363
363
364
365
365
366
367
367
368
369
369
370
371
371
372
373
373
374
375
375
376
377
377
378
379
379
380
381
381
382
383
383
384
385
385
386
387
387
388
389
389
390
391
391
392
393
393
394
395
395
396
397
397
398
399
399
400
401
401
402
403
403
404
405
405
406
407
407
408
409
409
410
411
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
471
472
473
473
474
475
475
476
477
477
478
479
479
480
481
481
482
483
483
484
485
485
486
487
487
488
489
489
490
491
491
492
493
493
494
495
495
496
497
497
498
499
499
500
501
501
502
503
503
504
505
505
506
507
507
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518

```

Краткое описание области: Когда данные не помещаются в RAM, используются внешние алгоритмы: делим на блоки (чанки), сортируем каждый чанк во внутренней памяти, затем делаем k-агу (многостороннее) слияние по файлам. Сложность выражают через число блоков и операций I/O.

Блок-схема:

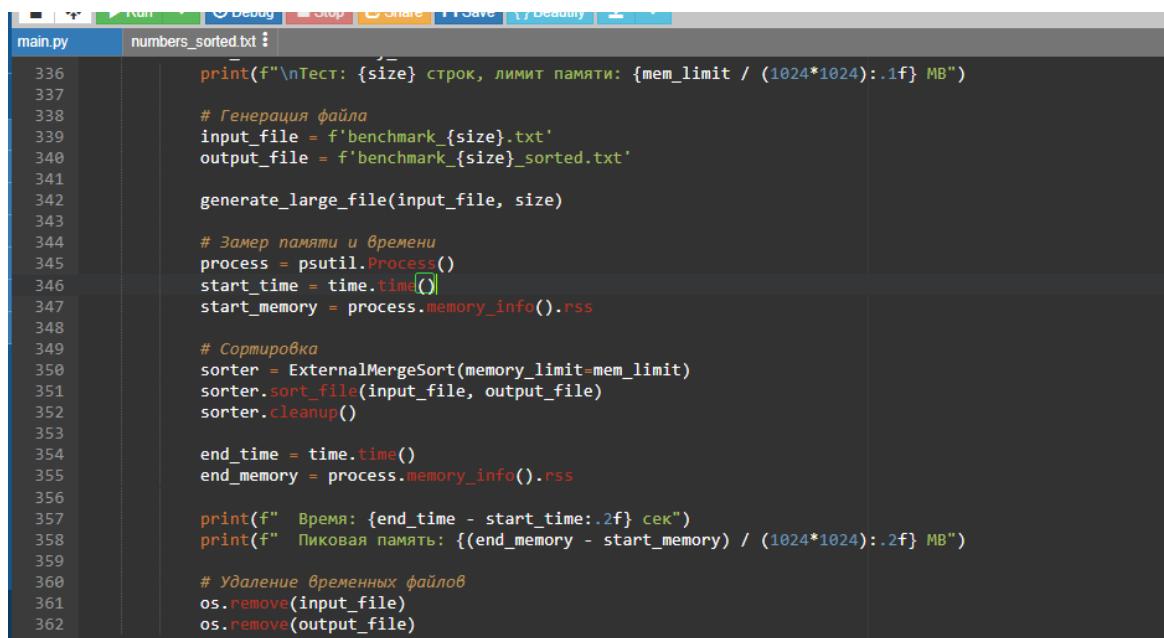


Пошаговый анализ:

1. Разделение входного файла на $\text{ceil}(N/M)$ чанков, каждый размером $\leq M$.

2. В памяти сортировка каждого чанка (любым подходящим алгоритмом).
3. Запись отсортированных чанков во временные файлы.
4. Открытие всех временных файлов и одновременное k-way слияние: поддерживать min-кучу (heap) первой строки из каждого файла; извлекать минимальный элемент, записывать в выход, читать следующий элемент из того же файла и помещать в кучу; повторять до опустошения.
5. Удаление временных файлов.

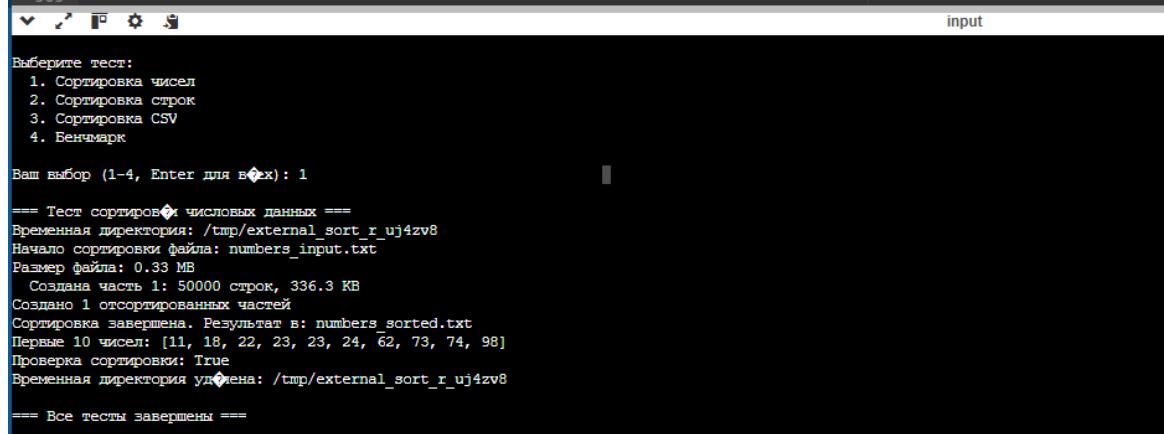
Результат:



```

main.py numbers_sorted.txt
336     print(f"\nТест: {size} строк, лимит памяти: {mem_limit / (1024*1024):.1f} MB")
337
338     # Генерация файла
339     input_file = f'benchmark_{size}.txt'
340     output_file = f'benchmark_{size}_sorted.txt'
341
342     generate_large_file(input_file, size)
343
344     # Замер памяти и времени
345     process = psutil.Process()
346     start_time = time.time()
347     start_memory = process.memory_info().rss
348
349     # Сортировка
350     sorter = ExternalMergeSort(memory_limit=mem_limit)
351     sorter.sort_file(input_file, output_file)
352     sorter.cleanup()
353
354     end_time = time.time()
355     end_memory = process.memory_info().rss
356
357     print(f" Время: {end_time - start_time:.2f} сек")
358     print(f" Пиковая память: {(end_memory - start_memory) / (1024*1024):.2f} MB")
359
360     # Удаление временных файлов
361     os.remove(input_file)
362     os.remove(output_file)
363

```



```

Выберите тест:
1. Сортировка чисел
2. Сортировка строк
3. Сортировка CSV
4. Бенчмарк

Ваш выбор (1-4, Enter для выхода): 1

== Тест сортировки числовых данных ==
Временная директория: /tmp/external_sort_r_uj4zv8
Начало сортировки файла: numbers_input.txt
Размер файла: 0.33 MB
Создана часть 1: 50000 строк, 336.3 KB
Создано 1 отсортированных частей
Сортировка завершена. Результат в: numbers_sorted.txt
Первые 10 чисел: [11, 18, 22, 23, 23, 24, 62, 73, 74, 98]
Проверка сортировки: True
Временная директория удалена: /tmp/external_sort_r_uj4zv8

== Все тесты завершены ==

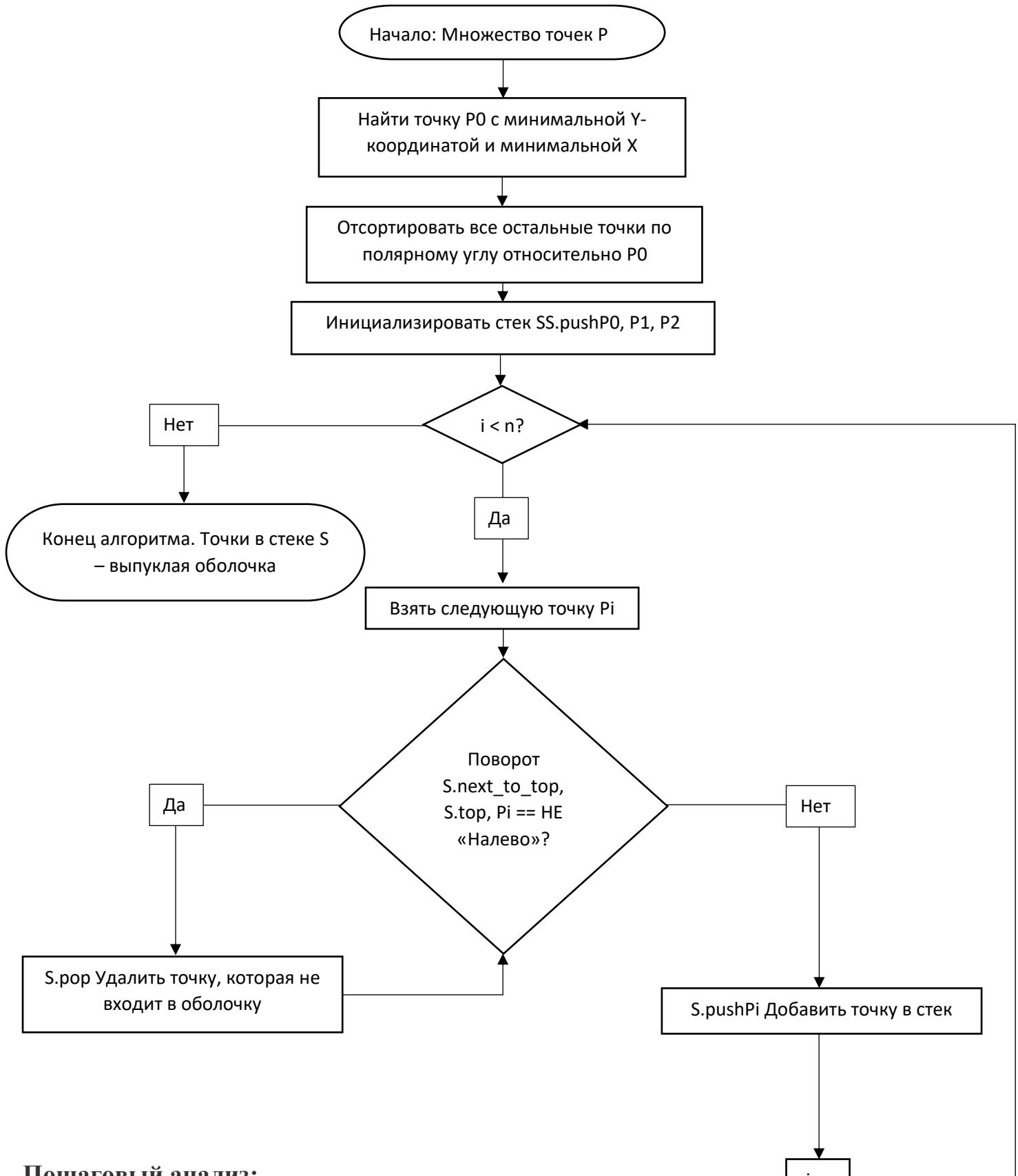
```

Оценка сложности:

- I/O-число операций: примерно $O((N/B)*\log(M/B)(N/B))$ в модели внешней памяти (M — внутренняя память, B — размер блока). Для практической реализации время доминирует за счёт чтения/записи блоков.

Краткое описание области: Вычислительная геометрия решает задачи о точках, многоугольниках, диаграммах Вороного и т.д. Graham Scan строит выпуклую оболочку множества точек за $O^*(n * \log n)$: сортировка по углу + проход стека.

Блок-схема:



Пошаговый анализ:

1. Найти точку p_0 с минимальной y (если tie — минимальная x).

2. Отсортировать остальные точки по полярному углу относительно $p_0p_1p_0$.
3. Инициализировать стек первыми 2–3 точками.
4. Перебирать остальные точки: пока последние два в стеке и новая точка делают неповорот влево (не CCW), убирать вершину стека; затем пушить новую точку.
5. В стеке останутся вершины выпуклой оболочки в порядке обхода.

Результат:

```

main.py
484 if __name__ == "__main__":
485     print("== Алгоритм Грэхема (Graham Scan) для построения выпуклой оболочки ==")
486
487     # Демонстрационный пример
488     print("\n--- Демонстрационный пример ---")
489
490     # Генерируем случайные точки
491     points = generate_random_points(50, seed=42)
492
493     # Строим выпуклую оболочку
494     hull = GrahamScan.convex_hull(points)
495
496     print(f"Всего точек: {len(points)}")
497     print(f"Точек в выпуклой оболочке: {len(hull)}")
498     print(f"Выпуклая оболочка: {hull}")
499
500     # Визуализация
501     plot_convex_hull(points, hull, "Graham Scan - Демонстрация")
502
503     # Бенчмарк
504     benchmark_graham_scan()
505
506     # Тестирование граничных случаев
507     test_edge_cases()
508
509     # Интерактивная демонстрация
510     # interactive_demo()

== Алгоритм Грэхема (Graham Scan) для построения выпуклой оболочки ==
--- Демонстрационный пример ---
Всего точек: 50
Точек в выпуклой оболочке: 12
Выпуклая оболочка: [(80.94, 0.65), (89.22, 8.69), (95.72, 33.66), (99.75, 50.95), (98.95, 64.00), (97.11, 86.08), (53.62, 97.31), (21.10, 94.29), (10.96, 62.74), (2.65, 1.15)]
/home/main.py:305: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
  plt.show()
== Бенчмарк алгоритма Грэхема ==
Точек: 100 -> Выпуклая оболочка: 12 точек Время: 0.25 мс
Точек: 500 -> Выпуклая оболочка: 19 точек Время: 2.20 мс
Точек: 1000 -> Выпуклая оболочка: 19 точек Время: 4.08 мс
Точек: 5000 -> Выпуклая оболочка: 27 точек Время: 11.29 мс
Точек: 10000 -> Выпуклая оболочка: 28 точек Время: 21.39 мс

== Тестирование граничных случаев ==
1 точка: [(10.00, 20.00)] -> hull: [(10.00, 20.00)]
2 точки: [(10.00, 20.00), (30.00, 40.00)] -> hull: [(10.00, 20.00), (30.00, 40.00)]
3 точки (треугольник): hull: [(0.00, 0.00), (10.00, 0.00), (5.00, 10.00)]
Коллинеарные точки: hull: [(0.00, 0.00), (15.00, 15.00)]
Прямоугольник + внутренние точки: hull имеет 4 вершины

...Program finished with exit code 0
Press ENTER to exit console.

```

Оценка сложности:

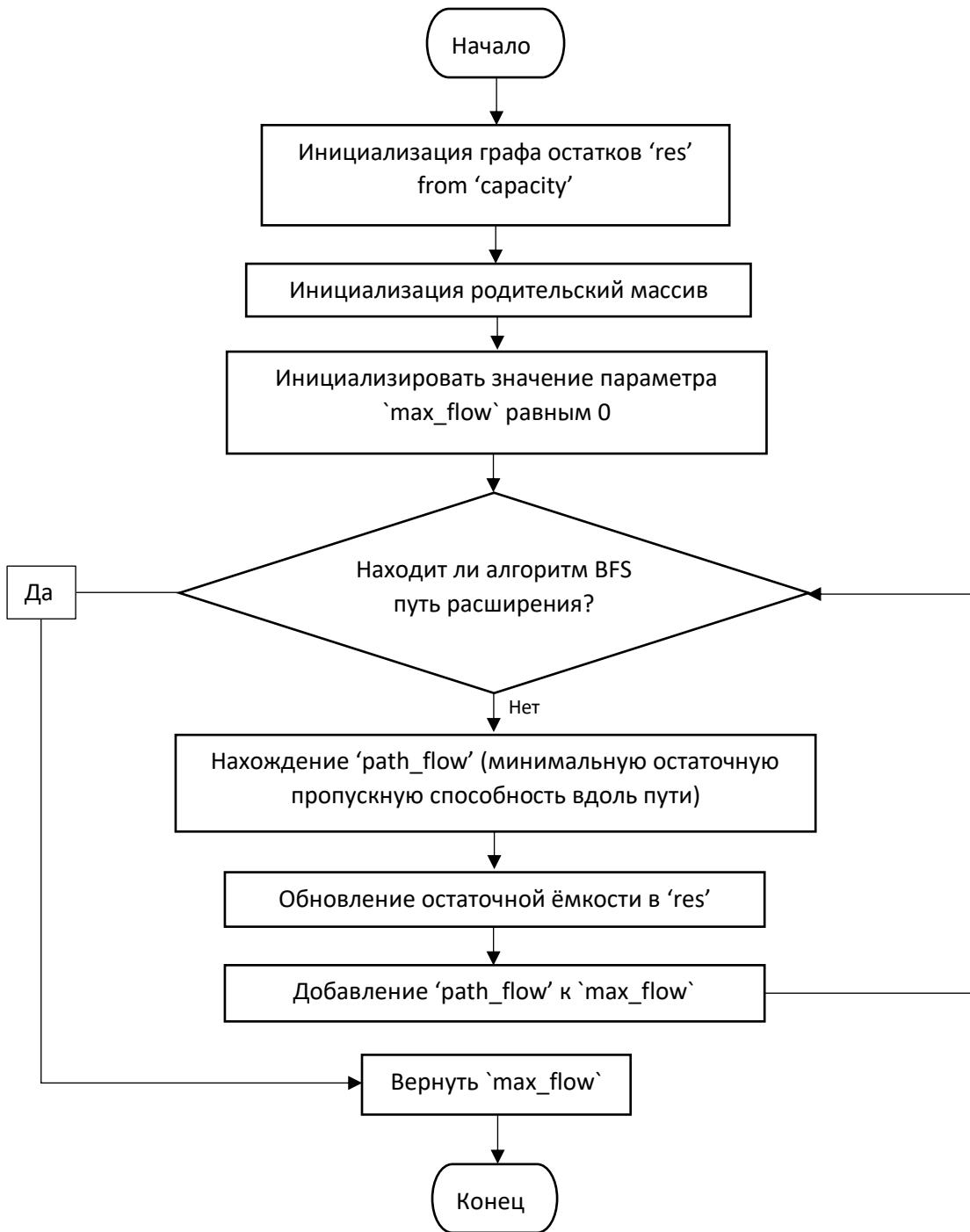
- Сортировка точек по углу: $O(n \log n)$.
 - Проход стека: $O(n)$.
- Итого $O(n \log n)$.

Предметная область №4: «Теория расписаний / потоки — Ford–Fulkerson / Edmonds–Karp»

Краткое описание области: Алгоритмы потоков ищут максимальный поток из источника в сток. Ford–Fulkerson ищет увеличивающие пути (любым методом),

Edmonds–Karp — частный случай с BFS (короткие по ребрам пути), что даёт полиномиальную оценку.

Блок-схема:



Пошаговый анализ:

- Построить остаточную сеть равную исходной (capabilities).
- Выполнять BFS, чтобы найти путь из s в t в остаточной сети (ребро с положительной пропускной способностью).

3. Если путь найден — вычислить минимальную пропускную способность на пути (`path_flow`), обновить остаточные ёмкости по пути (убавить в прямом направлении, прибавить в обратном), увеличить `max_flow` на `path_flow`.
4. Повторять, пока не останется увеличивающих путей.

Результат:

```

main.py
701     print("Неверный выбор, использую BFS")
702     max_flow = network.max_flow_bfs(source, sink)
703     algo_name = "BFS"
704
705     print(f"\n{algo_name}: максимальный поток = {max_flow}")
706
707     # Показываем потоки
708     print("\nПотоки на ребрах:")
709     for edge in network.edges:
710         if edge.flow > 0 or edge.capacity > 0:
711             print(f" {edge.u} → {edge.v}: {edge.flow:.1f}/{edge.capacity:.1f}")
712
713     # Визуализация
714     visualize = input("\nВизуализировать сеть? (y/n): ").strip().lower()
715     if visualize == 'y':
716         network.visualize(f"Максимальный поток = {max_flow}")
717
718
719 if __name__ == "__main__":
720     print("== Алгоритм Форда-Фалкерсона для поиска максимального потока ==")
721
722     # Демонстрация на классическом примере
723     solve_max_flow_problem()
724
725     # Демонстрация приложений
726     solve_bipartite_matching()
727     solve_transport_problem()
728
    == Алгоритм Форда-Фалкерсона для поиска максимального потока ==
    == Пример 1: Классическая сеть ==
Исходная сеть:
0 → 1: capacity = 16
0 → 2: capacity = 13
1 → 2: capacity = 10
1 → 3: capacity = 12
2 → 1: capacity = 4
2 → 4: capacity = 14
3 → 2: capacity = 9
3 → 5: capacity = 20
4 → 3: capacity = 7
4 → 5: capacity = 4
/home/main.py:307: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()

Максимальный поток из 0 в 5: 23.0

Потоки на ребрах:
0 → 1: 12.0/16.0
0 → 2: 11.0/13.0
1 → 2: 0.0/10.0
1 → 3: 12.0/12.0
2 → 1: 0.0/4.0
2 → 4: 11.0/14.0

```

Оценка сложности:

- Edmonds–Karp гарантирует $O(VE^2)$ временной сложности (BFS даёт кратчайшие пути, число увеличений потоков $\leq O(VE)$). Ford–Fulkerson без ограничения метода поиска может иметь экспоненциальный худший случай.