

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
дисциплины
«Искусственный интеллект в профессиональной сфере»

Выполнила:
Михеева Елена Александровна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
?

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

ТЕМА: ИССЛЕДОВАНИЕ ПОИСКА В ШИРИНУ

Цель: приобретение навыков по работе с поиском в ширину с помощью языка программирования Python версии 3.x

Порядок выполнения работы:

Репозиторий: https://github.com/helendddd/AI_2.git

Расширенный подсчет количества островов в бинарной матрице.

Дана бинарная матрица, где 0 представляет воду, а 1 представляет землю. Связанные единицы формируют остров. Необходимо подсчитать общее количество островов в данной матрице. Острова могут соединяться как по вертикали и горизонтали, так и по диагонали.

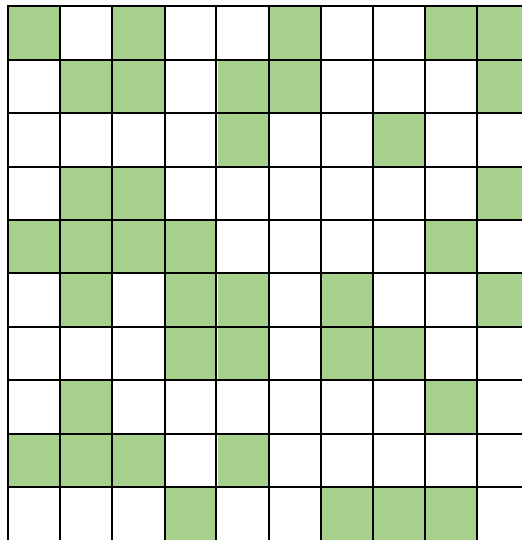


Рисунок 1. Наглядное представление островов в матрице

Или в бинарном представлении:

1	0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	0	0	0	1
0	0	0	0	1	0	0	1	0	0
0	1	1	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0	0	1
0	0	0	1	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	0
1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	1	1	0

```

4 from collections import deque
5
6
7 def count_islands_bfs(matrix):
8     """
9     Подсчет количества островов в бинарной матрице
10    с учетом диагональных соединений.
11    """
12    if not matrix or not matrix[0]:
13        return 0
14
15    rows, cols = len(matrix), len(matrix[0])
16    directions = [
17        (-1, 0), (1, 0), (0, -1), (0, 1), # Вверх, вниз, влево, вправо
18        (-1, -1), (-1, 1), (1, -1), (1, 1) # Диагонали
19    ]
20
21    def bfs(start_x, start_y):
22        # Создаем очередь и добавляем начальную точку
23        queue = deque([(start_x, start_y)])
24        matrix[start_x][start_y] = 0 # Помечаем как посещенную
25
26        while queue:
27            x, y = queue.popleft()
28            for dx, dy in directions:
29                nx, ny = x + dx, y + dy
30                # Если сосед является частью суши и не посещен
31                if 0 <= nx < rows and 0 <= ny < cols and matrix[nx][ny] == 1:
32                    matrix[nx][ny] = 0 # Помечаем как посещенную
33                    queue.append((nx, ny))
34
35    island_count = 0
36
37    for i in range(rows):
38        for j in range(cols):
39            if matrix[i][j] == 1: # Новый остров найден
40                island_count += 1
41                bfs(i, j)
42
43    return island_count
44
45
46 if __name__ == '__main__':
47     binary_matrix = [
48         [1, 0, 1, 0, 0, 1, 0, 0, 1, 1],
49         [0, 1, 1, 0, 1, 1, 0, 0, 0, 1],
50         [0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
51         [0, 1, 1, 0, 0, 0, 0, 0, 0, 1],
52         [1, 1, 1, 1, 0, 0, 0, 1, 0, 0],
53         [0, 1, 0, 1, 1, 0, 1, 0, 0, 1],
54         [0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
55         [0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
56         [1, 1, 1, 0, 1, 0, 0, 0, 0, 0],
57         [0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
58     ]
59     result = count_islands_bfs(binary_matrix)
60     print(f"Количество островов: {result}")
61

```

Рисунок 2. Код программы

```

(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/count_island.py
Количество островов: 9
(base) elenamiheeva@MacBook-Pro-Elena AI_2 %

```

Рисунок 3. Результат работы программы

Поиск кратчайшего пути в лабиринте.

Необходимо подготовить схему лабиринта, а также определить начальную и конечную позиции в лабиринте. Лабиринт представлен в виде бинарной матрицы, где 1 обозначает проход, а 0 — стену. На вход программа принимает лабиринт, координаты начальной и конечной точки. В результате работы программа выводит длину полученного пути. В качестве начальной позиции выбрана точка с координатами (1, 1), а для конечной — (7, 7).

1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1
1	0	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	0	0	1
1	0	1	0	1	0	0	0	0	1
1	0	1	1	1	0	1	1	1	1
1	0	0	0	1	0	0	1	0	1
1	1	1	1	1	1	1	1	0	1
1	1	1	0	0	0	0	0	0	1

```

1 from collections import deque
2
3
4 def is_valid(x, y, maze, visited):
5     """
6     Проверяет, является ли клетка допустимой для посещения.
7     """
8     return (
9         0 <= x < len(maze)
10        and 0 <= y < len(maze[0])
11        and maze[x][y] == 1
12        and not visited[x][y]
13    )
14
15
16 def bfs(maze, start, end):
17     """
18     Алгоритм поиска в ширину (BFS) для нахождения кратчайшего пути.
19     """
20     rows, cols = len(maze), len(maze[0])
21     visited = [[False for _ in range(cols)] for _ in range(rows)]
22     distance = [[-1 for _ in range(cols)] for _ in range(rows)]
23     queue = deque([start]) # Очередь для BFS
24     visited[start[0]][start[1]] = True
25     distance[start[0]][start[1]] = 1
26
27     while queue:
28         x, y = queue.popleft()
29
30         # Если достигли конечной клетки, возвращаем расстояние
31         if (x, y) == end:
32             return distance[x][y]
33
34         # Проверяем соседей
35         for dx, dy in DIRECTIONS:
36             nx, ny = x + dx, y + dy
37             if is_valid(nx, ny, maze, visited):
38                 visited[nx][ny] = True
39                 distance[nx][ny] = distance[x][y] + 1
40                 queue.append((nx, ny))
41
42     return None # Если путь не найден
43
44
45 if __name__ == '__main__':
46
47     # Лабиринт (где 0 - стена, 1 - свободный путь)
48     maze = [
49         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
50         [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
51         [1, 0, 1, 1, 1, 1, 1, 0, 0, 1],
52         [1, 0, 1, 0, 0, 0, 1, 1, 1, 1],
53         [1, 0, 1, 0, 1, 1, 1, 0, 0, 1],
54         [1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
55         [1, 0, 1, 1, 1, 0, 1, 1, 1, 1],
56         [1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
57         [1, 1, 1, 1, 1, 1, 1, 0, 1, 1],
58         [1, 1, 1, 0, 0, 0, 0, 0, 0, 1],
59     ]
60
61     # Направления движения: вверх, вниз, влево, вправо
62     DIRECTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]
63     # Начальная и конечная позиции
64     START = (0, 0)
65     END = (6, 6)
66
67     # Находим кратчайший путь
68     path_weight = bfs(maze, START, END)
69
70     # Вывод результата
71     if path_weight is not None:
72         print(f"Вес кратчайшего пути: {path_weight}")
73     else:
74         print("Путь не найден.")
75

```

Рисунок 4. Код программы

```

(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/labirinth.py
Вес кратчайшего пути: 19

```

Рисунок 5. Результат работы программы

Нахождение минимального расстояния между начальными и конечными пунктами с использованием алгоритма поиска в ширину.

Был построен граф из 20 населенных пунктов Италии. Узлы данного графа представляют населённые пункты, а рёбра — дороги, соединяющие их. Вес каждого ребра соответствует расстоянию между этими пунктами. В качестве начального населенного пункта выбран Кунео, а конечного – Верона.

Италия

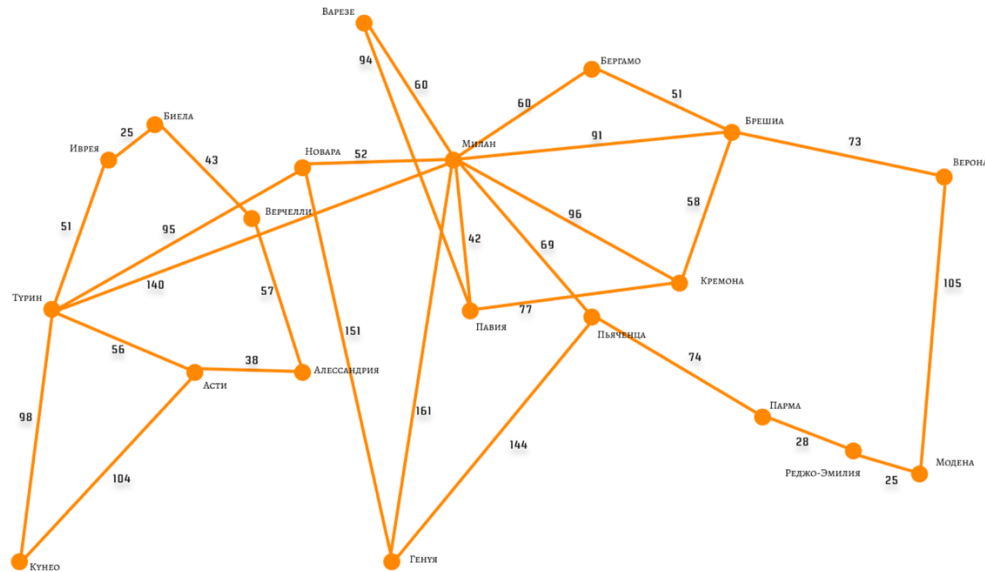


Рисунок 6. Граф из населенных пунктов

```
1 from collections import deque
2
3 class Node:
4     def __init__(self, state, parent=None, cost=0):
5         self.state = state # состояние узла
6         self.parent = parent # родитель узла
7         self.cost = cost # стоимость пути до узла
8
9 class Problem:
10     def __init__(self, initial, goal, graph):
11         self.initial = initial # начальное состояние
12         self.goal = goal # конечное состояние
13         self.graph = graph # граф с соседями для каждого состояния
14
15     def is_goal(self, state):
16         return state == self.goal
17
18     def expand(self, node):
19         """Возвращает список соседей для текущего узла."""
20         neighbors = []
21         for neighbor, cost in self.graph[node.state]:
22             neighbors.append(Node(neighbor, node, node.cost + cost))
23         return neighbors
24
25 def breadth_first_search(problem):
26     node = Node(problem.initial)
27
28     if problem.is_goal(problem.initial):
29         return node
30
31     frontier = deque([node]) # Очередь для хранения узлов
32     reached = {problem.initial} # Множество посещенных узлов
33
34     while frontier:
35         node = frontier.popleft() # Удаление узла из очереди
36
37         # Проходим по соседям узла
38         for child in problem.expand(node):
39             s = child.state
40
41             # Если цель найдена, возвращаем путь
42             if problem.is_goal(s):
43                 return child
44
45             if s not in reached:
46                 reached.add(s)
47                 frontier.append(child)
48
49     return None # Возвращаем None, если путь не найден
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
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Рисунок 7. Код программы для задания

```
(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/breadth-first_search.py
Минимальный путь: ['Кунео', 'Турин', 'Милан', 'Брешиа', 'Верона']
Минимальное расстояние: 402
(base) elenamiheeva@MacBook-Pro-Elena AI_2 %
```

Рисунок 8. Результат работы программы

Ответы на контрольные вопросы:

1. Какой тип очереди используется в стратегии поиска в ширину?

В стратегии поиска в ширину используется очередь (FIFO - First In, First Out).

2. Почему новые узлы в стратегии поиска в ширину добавляются в конец очереди?

Это делается для того, чтобы гарантировать, что узлы будут обрабатываться в порядке их появления, что обеспечивает уровень за уровнем (глубину) расширения узлов.

3. Что происходит с узлами, которые дольше всего находятся в очереди в стратегии поиска в ширину?

Узлы, которые находятся в очереди дольше, будут расширены ранее, чем более новые узлы, если они находятся на одном уровне (глубине) поиска.

4. Какой узел будет расширен следующим после корневого узла, если используются правила поиска в ширину?

Следующим будет узел, который был добавлен в очередь сразу после корневого узла, то есть первый дочерний узел корневого узла.

5. Почему важно расширять узлы с наименьшей глубиной в поиске в ширину?

Это позволяет находить кратчайшие пути к целевым узлам, так как узлы на меньшей глубине исследуются первыми, гарантируя, что цель будет достигнута с минимальным количеством шагов.

6. Как временная сложность алгоритма поиска в ширину зависит от коэффициента разветвления и глубины?

Временная сложность составляет $O(b^d)$, где b - коэффициент разветвления (среднее количество детей) и d - глубина решения. Это объясняется тем, что каждый уровень добавляет b новых узлов.

7. Каков основной фактор, определяющий пространственную сложность алгоритма поиска в ширину?

Основной фактор - это количество узлов, которые находятся на текущем уровне поиска, т.е. $O(b^d)$, особенно когда d является глубиной решения.

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если есть конечная глубина, на которой можно достигнуть целевое состояние, и если пространство состояний не бесконечно.

9. Объясните, почему поиск в ширину может быть неэффективен с точки зрения памяти.

Поиск в ширину хранит все узлы на текущем уровне, что может привести к большим требованиям к памяти, особенно в разветвленных графах с большой глубиной.

10. В чем заключается оптимальность поиска в ширину?

Поиск в ширину гарантирует нахождение кратчайшего пути к целевому узлу, если все переходы имеют одинаковую стоимость.

11. Какую задачу решает функция `breadthfirstsearch`?

Функция `breadth_first_search` решает задачу поиска целевого состояния в пространстве состояний, начиная с начального состояния.

12. Что представляет собой объект `problem`, который передается в функцию?

Объект `problem` представляет собой описание проблемы (задачи), включая начальное состояние, функцию перехода и целевое состояние.

13. Для чего используется узел `Node(problem.initial)` в начале функции?

Он инициализирует корневой узел поиска, который представляет начальное состояние задачи.

14. Что произойдет, если начальное состояние задачи уже является целевым?

Функция сразу вернет начальный узел как решение, так как нет необходимости в дальнейших поисках.

15. Какую структуру данных использует frontier и почему выбрана именно очередь FIFO?

frontier использует очередь FIFO, чтобы обеспечивать порядок обработки узлов по уровням глубины, что является ключевым в стратегии поиска в ширину.

16. Какую роль выполняет множество reached?

Множество reached отслеживает все достигнутые состояния, чтобы избежать повторного рассмотрения уже исследованных узлов.

17. Почему важно проверять, находится ли состояние в множестве reached?

Это важно для предотвращения цикла и избыточной работы, а также для уменьшения использования памяти.

18. Какую функцию выполняет цикл while frontier?

Цикл while frontier отвечает за повторное извлечение и обработку узлов из очереди до тех пор, пока в ней есть узлы.

19. Что происходит с узлом, который извлекается из очереди в строке `node = frontier.pop()`?

Извлеченный узел становится активным узлом для обработки, и с ним выполняются действия по его расширению.

20. Какова цель функции `expand(problem, node)` ?

Функция `expand` генерирует и возвращает всех дочерних узлов активного узла, основываясь на заданных правилах задачи.

21. Как определяется, что состояние узла является целевым? 22. Что происходит, если состояние узла не является целевым, но также не было ранее достигнуто?

Условие целевого состояния проверяется с использованием заранее определённой функции проверки, связанной с объектом проблемы. Если состояние узла не является целевым, то узел будет расширен, и его дочерние узлы будут добавлены в очередь для дальнейшей обработки.

22. Почему дочерний узел добавляется в начало очереди с помощью `appendleft(child)`?

Это может быть специфично для других стратегий поиска (например, для поиска в глубину), однако для поиска в ширину дочерние узлы добавляются в конец очереди.

23. Что возвращает функция `breadth_first_search`, если решение не найдено?

Если решение не найдено, функция обычно возвращает `None` или аналогичный объект, который указывает на отсутствие решения.

24. Каково значение узла `failure` и когда он возвращается?

Узел `failure` может использоваться для сигнализации о том, что алгоритм не нашел решение, и обычно возвращается в конце выполнения, если ни один узел не удовлетворяет условиям.

Вывод: были приобретены навыки по работе с поиском в ширину с помощью языка программирования Python версии 3.x