

I designed my algorithm to be extraordinary modular therefore reducing the amount of work needed in order to code each algorithm. This is because by having different data structures that choose which nodes should be analyzed allows for each of these algorithms to turn into one another. Since stack will choose the most recent in it will simulate DFS where it will shoot down paths as it keeps choosing the first node it sees. Queues work like BFS where it continuously keeps checking the first things it saw before going to next one. While priority queues can operate like UFS if you set it to cost as priority since the lowest cost will always be next analyzed. I found it also necessary how each node got found in order to be able to back track the path as required by the assignment. This algorithm likely has room for improvement here as it is $O(n)$ in order to keep get back the path and I imagine could've been done alongside the search algorithm but IDK how. The algorithm seems to have some struggle with direct assignment over taking from input for whatever reason when I took input from user even if it were identical to the string if it were hard coded the algorithms would break. I couldn't find the underlying cause of this problem and managed to get around by having a map that assigns a hardcoded value corresponding to the users input.

Code Screenshots:

```

1 import time
2 def trackBack(parents, startCity, targetCity):
3     currentCity = targetCity
4     path = []
5     while currentCity is not startCity:
6         path.insert(0, currentCity)
7         currentCity = parents[currentCity]
8     path.insert(0, currentCity)
9     return path
10
11 def BFS(graph, startCity, targetCity):
12     cityQueue = []
13     cityQueue.append(startCity)
14     visitedSet = set()
15     visitedSet.add(startCity)
16     parents = {}
17     nodesChecked = 0
18     while cityQueue:
19         currentCity = cityQueue.pop(0)
20         for city in list(graph[currentCity].keys()):
21             nodesChecked += 1
22             if city not in visitedSet:
23                 cityQueue.append(city)
24                 visitedSet.add(city)
25                 parents[city] = currentCity
26         if city is targetCity:
27             print("BFS Number of Nodes Checked: ", nodesChecked)
28             return trackBack(parents, startCity, targetCity)
29
30 def DFS(graph, startCity, targetCity):
31     cityStack = []
32     cityStack.append(startCity)
33     visitedSet = set()
34     visitedSet.add(startCity)
35     parents = {}
36     nodesChecked = 0
37
38 def DFS(graph, startCity, targetCity):
39     nodesChecked = 0
40     while cityStack:
41         currentCity = cityStack.pop()
42         for city in list(graph[currentCity].keys()):
43             nodesChecked += 1
44             if city not in visitedSet:
45                 cityStack.append(city)
46                 visitedSet.add(city)
47                 parents[city] = currentCity
48         if city is targetCity:
49             print("DFS Number of Nodes Checked: ", nodesChecked)
50             return trackBack(parents, startCity, targetCity)
51
52 from queue import PriorityQueue
53 def UFS(graph, startCity, targetCity):
54     cityPQ = PriorityQueue()
55     cityPQ.put((0, startCity))
56     visitedSet = set()
57     visitedSet.add(startCity)
58     parents = {}
59     nodesChecked = 0
60     while cityPQ:
61         cost, currentCity = cityPQ.get()
62         for city in list(graph[currentCity].keys()):
63             nodesChecked += 1
64             if city not in visitedSet:
65                 cityPQ.put((graph[currentCity][city], city))
66                 visitedSet.add(city)
67                 parents[city] = currentCity
68         if city is targetCity:
69             print("UFS Number of Nodes Checked: ", nodesChecked)
70             return trackBack(parents, startCity, targetCity)
71
72 graph = {
73     'Buffalo': {'Buffalo': 450, 'Pittsburgh': 219,
74     'Boston': {'Buffalo': 450, 'New York': 210,
75     'Pittsburgh': {'New York': 370, 'Buffalo': 219, 'Philadelphia': 304, 'Baltimore': 248,
76     'New York': {'Boston': 210, 'Pittsburgh': 370, 'Philadelphia': 94,
77     'Philadelphia': {'New York': 90, 'Pittsburgh': 304, 'Baltimore': 181, 'Salisbury': 130,
78     'Baltimore': {'Philadelphia': 181, 'Pittsburgh': 248, 'Washington DC': 45, 'Salisbury': 117,
79     'Salisbury': {'Philadelphia': 130, 'Baltimore': 117, 'Washington DC': 116, 'Norfolk': 132,
80     'Washington DC': {'Baltimore': 45, 'Salisbury': 116, 'Richmond': 118,
81     'Richmond': {'Washington DC': 110, 'Norfolk': 93,
82     'Norfolk': {'Richmond': 93, 'Salisbury': 132}
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     }
1000    }

```

Output:

```

onnlladder-:/Codes_for_School/COSC411/Project1$ python graph.py
Please Input Starting City Name:
New York
Please Input Target City Name:
Richmond

BFS Number of Nodes Checked: 26
Shortest Path Distance : 773
Shortest Path: ['New York', 'Pittsburgh', 'Baltimore', 'Washington DC', 'Richmond']
BFS Search Time is 0.0001080399840464815

DFS Number of Nodes Checked: 12
Depth First Search Distance : 457
DFS Path: ['New York', 'Philadelphia', 'Salisbury', 'Norfolk', 'Richmond']
DFS Search Time is 4.063700180267915e-05

UFS Number of Nodes Checked: 14
Uniform Cost Search Distance : 350
UFS Path: ['New York', 'Philadelphia', 'Baltimore', 'Washington DC', 'Richmond']
UFS Search Time is 0.00013104900426696986

```