

Experiment Title- 3.1

STUDENT NAME :- YASH KUMAR

UID :- 20BCS9256

SECTION :- 616 'B'

SEMESTER :- 5TH

SUBJECT:- DESIGN OF ANALYSIS AND ALGORITHM

AIM :- Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as (i) to find the topological sort of a directed acyclic graph, OR (ii) to find a path from source to goal in a maze.

Program Code :-

a) Code and analyze to do a depth-first search (DFS) on an undirected graph

```
#include <bits/stdc++.h>

using namespace std;

class Graph {
public:
    map<int, bool> visited;
    map<int, list<int> > adj;
    void addEdge(int v, int w);
    void DFS(int v);
};

void Graph::addEdge(int v, int w)
```

```
{  
  
    adj[v].push_back(w);  
  
}  
  
void Graph::DFS(int v)  
{  
  
    visited[v] = true;  
  
    cout << v << " ";  
  
  
  
    list<int>::iterator i;  
    for (i = adj[v].begin(); i != adj[v].end(); ++i)  
        if (!visited[*i])  
            DFS(*i);  
}  
  
int main()  
{  
  
  
  
    Graph g;  
  
    g.addEdge(0, 1);  
    g.addEdge(0, 2);  
    g.addEdge(1, 2);  
    g.addEdge(2, 0);  
    g.addEdge(2, 3);  
    g.addEdge(3, 3);  
  
  
  
    cout << "Following is Depth First Traversal"  
  
    " (starting from vertex 2) \n";
```

g.DFS(2);

return 0;

}

Output :-

The screenshot shows a web browser window with the OnlineGDB online C++ compiler. The code in the editor is as follows:

```

26 visited[v] = true;
27 cout << v << " ";
28
29
30 list<int>::iterator i;
31 for (i = adj[v].begin(); i != adj[v].end(); ++i)
32     if (!visited[*i])
33         DFS(*i);
34 }
35
36 int main()
37 {
38     Graph g;
39     g.addEdge(0, 1);
40     g.addEdge(0, 2);
41     g.addEdge(1, 3);

```

The output window shows the following text:

```

Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

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

```

b) to find the topological sort of a directed acyclic graph

Program Code :- #include <bits/stdc++.h>

using namespace std;

class Graph {

int V;

list<int>* adj;

void topologicalSortUtil(int v, bool visited[],
stack<int>& Stack);

public:

Graph(int V);

void addEdge(int v, int w);

void topologicalSort();

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push_back(w);

}

void Graph::topologicalSortUtil(int v, bool visited[],

stack<int>& Stack)

```
{  
    visited[v] = true;  
  
    list<int>::iterator i;  
    for (i = adj[v].begin(); i != adj[v].end(); ++i)  
        if (!visited[*i])  
            topologicalSortUtil(*i, visited, Stack);  
  
    Stack.push(v);  
}  
  
void Graph::topologicalSort()  
{  
    stack<int> Stack;  
  
    bool* visited = new bool[V];  
    for (int i = 0; i < V; i++)  
        visited[i] = false;  
  
    for (int i = 0; i < V; i++)  
        if (visited[i] == false)  
            topologicalSortUtil(i, visited, Stack);  
  
    while (Stack.empty() == false) {  
        cout << Stack.top() << " ";  
        Stack.pop();  
    }  
}  
  
int main()  
{
```

Graph g(6);

```
g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

g.addEdge(3, 1);


cout << "Following is a Topological Sort of the given "

    "graph \n";

g.topologicalSort();


return 0;

}
```

Output :-

The screenshot shows the OnlineGDB web interface. The browser tabs include '(45) Aashiqui-2', 'Chandigarh Univ', 'Welcome to C++', 'Worksheet 3.1', 'Online C++ Compiler', 'Topological Sort', and 'Find the shortest'. The address bar shows 'onlinegdb.com/online_c++_compiler'. The left sidebar contains links for 'OnlineGDB beta', 'code.compile.run.debug.share.', 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Sign Up', 'Login', and social media links. The main editor shows a C++ program for topological sort. The console output displays the topological sort result: 'Following is a Topological Sort of the given graph' followed by the sequence '5 4 2 3 1 0'. The program finished with exit code 0.

```

main.cpp
33 topologicalSortUtil(int u, vector<int> &visited, Stack);
34
35 while (Stack.empty() == false) {
36     cout << Stack.top() << " ";
37     Stack.pop();
38 }
39 }
40
41 int main()
42 {
43
44     Graph g(6);
45     g.addEdge(5, 2);
46     g.addEdge(5, 0);
47     g.addEdge(4, 0);
48     g.addEdge(4, 1);
49     g.addEdge(2, 3);
50
51     topologicalSortUtil(5, visited, Stack);
52
53     while (Stack.empty() == false) {
54         cout << Stack.top() << " ";
55         Stack.pop();
56     }
57 }
58
59 int main()
60 {
61
62     Graph g(6);
63     g.addEdge(5, 2);
64     g.addEdge(5, 0);
65     g.addEdge(4, 0);
66     g.addEdge(4, 1);
67     g.addEdge(2, 3);
68
69     topologicalSortUtil(5, visited, Stack);
70
71     while (Stack.empty() == false) {
72         cout << Stack.top() << " ";
73         Stack.pop();
74     }
75 }
76
77 }
78
79 int main()
80 {
81
82     Graph g(6);
83     g.addEdge(5, 2);
84     g.addEdge(5, 0);
85     g.addEdge(4, 0);
86     g.addEdge(4, 1);
87     g.addEdge(2, 3);
88
89     topologicalSortUtil(5, visited, Stack);
90
91     while (Stack.empty() == false) {
92         cout << Stack.top() << " ";
93         Stack.pop();
94     }
95 }
96
97 }
98
99 int main()
100 {
101
102     Graph g(6);
103     g.addEdge(5, 2);
104     g.addEdge(5, 0);
105     g.addEdge(4, 0);
106     g.addEdge(4, 1);
107     g.addEdge(2, 3);
108
109     topologicalSortUtil(5, visited, Stack);
110
111     while (Stack.empty() == false) {
112         cout << Stack.top() << " ";
113         Stack.pop();
114     }
115 }
116
117 }
118
119 int main()
120 {
121
122     Graph g(6);
123     g.addEdge(5, 2);
124     g.addEdge(5, 0);
125     g.addEdge(4, 0);
126     g.addEdge(4, 1);
127     g.addEdge(2, 3);
128
129     topologicalSortUtil(5, visited, Stack);
130
131     while (Stack.empty() == false) {
132         cout << Stack.top() << " ";
133         Stack.pop();
134     }
135 }
136
137 }
138
139 int main()
140 {
141
142     Graph g(6);
143     g.addEdge(5, 2);
144     g.addEdge(5, 0);
145     g.addEdge(4, 0);
146     g.addEdge(4, 1);
147     g.addEdge(2, 3);
148
149     topologicalSortUtil(5, visited, Stack);
150
151     while (Stack.empty() == false) {
152         cout << Stack.top() << " ";
153         Stack.pop();
154     }
155 }
156
157 }
158
159 int main()
160 {
161
162     Graph g(6);
163     g.addEdge(5, 2);
164     g.addEdge(5, 0);
165     g.addEdge(4, 0);
166     g.addEdge(4, 1);
167     g.addEdge(2, 3);
168
169     topologicalSortUtil(5, visited, Stack);
170
171     while (Stack.empty() == false) {
172         cout << Stack.top() << " ";
173         Stack.pop();
174     }
175 }
176
177 }
178
179 int main()
180 {
181
182     Graph g(6);
183     g.addEdge(5, 2);
184     g.addEdge(5, 0);
185     g.addEdge(4, 0);
186     g.addEdge(4, 1);
187     g.addEdge(2, 3);
188
189     topologicalSortUtil(5, visited, Stack);
190
191     while (Stack.empty() == false) {
192         cout << Stack.top() << " ";
193         Stack.pop();
194     }
195 }
196
197 }
198
199 int main()
200 {
201
202     Graph g(6);
203     g.addEdge(5, 2);
204     g.addEdge(5, 0);
205     g.addEdge(4, 0);
206     g.addEdge(4, 1);
207     g.addEdge(2, 3);
208
209     topologicalSortUtil(5, visited, Stack);
210
211     while (Stack.empty() == false) {
212         cout << Stack.top() << " ";
213         Stack.pop();
214     }
215 }
216
217 }
218
219 int main()
220 {
221
222     Graph g(6);
223     g.addEdge(5, 2);
224     g.addEdge(5, 0);
225     g.addEdge(4, 0);
226     g.addEdge(4, 1);
227     g.addEdge(2, 3);
228
229     topologicalSortUtil(5, visited, Stack);
230
231     while (Stack.empty() == false) {
232         cout << Stack.top() << " ";
233         Stack.pop();
234     }
235 }
236
237 }
238
239 int main()
240 {
241
242     Graph g(6);
243     g.addEdge(5, 2);
244     g.addEdge(5, 0);
245     g.addEdge(4, 0);
246     g.addEdge(4, 1);
247     g.addEdge(2, 3);
248
249     topologicalSortUtil(5, visited, Stack);
250
251     while (Stack.empty() == false) {
252         cout << Stack.top() << " ";
253         Stack.pop();
254     }
255 }
256
257 }
258
259 int main()
260 {
261
262     Graph g(6);
263     g.addEdge(5, 2);
264     g.addEdge(5, 0);
265     g.addEdge(4, 0);
266     g.addEdge(4, 1);
267     g.addEdge(2, 3);
268
269     topologicalSortUtil(5, visited, Stack);
270
271     while (Stack.empty() == false) {
272         cout << Stack.top() << " ";
273         Stack.pop();
274     }
275 }
276
277 }
278
279 int main()
280 {
281
282     Graph g(6);
283     g.addEdge(5, 2);
284     g.addEdge(5, 0);
285     g.addEdge(4, 0);
286     g.addEdge(4, 1);
287     g.addEdge(2, 3);
288
289     topologicalSortUtil(5, visited, Stack);
290
291     while (Stack.empty() == false) {
292         cout << Stack.top() << " ";
293         Stack.pop();
294     }
295 }
296
297 }
298
299 int main()
300 {
301
302     Graph g(6);
303     g.addEdge(5, 2);
304     g.addEdge(5, 0);
305     g.addEdge(4, 0);
306     g.addEdge(4, 1);
307     g.addEdge(2, 3);
308
309     topologicalSortUtil(5, visited, Stack);
310
311     while (Stack.empty() == false) {
312         cout << Stack.top() << " ";
313         Stack.pop();
314     }
315 }
316
317 }
318
319 int main()
320 {
321
322     Graph g(6);
323     g.addEdge(5, 2);
324     g.addEdge(5, 0);
325     g.addEdge(4, 0);
326     g.addEdge(4, 1);
327     g.addEdge(2, 3);
328
329     topologicalSortUtil(5, visited, Stack);
330
331     while (Stack.empty() == false) {
332         cout << Stack.top() << " ";
333         Stack.pop();
334     }
335 }
336
337 }
338
339 int main()
340 {
341
342     Graph g(6);
343     g.addEdge(5, 2);
344     g.addEdge(5, 0);
345     g.addEdge(4, 0);
346     g.addEdge(4, 1);
347     g.addEdge(2, 3);
348
349     topologicalSortUtil(5, visited, Stack);
350
351     while (Stack.empty() == false) {
352         cout << Stack.top() << " ";
353         Stack.pop();
354     }
355 }
356
357 }
358
359 int main()
360 {
361
362     Graph g(6);
363     g.addEdge(5, 2);
364     g.addEdge(5, 0);
365     g.addEdge(4, 0);
366     g.addEdge(4, 1);
367     g.addEdge(2, 3);
368
369     topologicalSortUtil(5, visited, Stack);
370
371     while (Stack.empty() == false) {
372         cout << Stack.top() << " ";
373         Stack.pop();
374     }
375 }
376
377 }
378
379 int main()
380 {
381
382     Graph g(6);
383     g.addEdge(5, 2);
384     g.addEdge(5, 0);
385     g.addEdge(4, 0);
386     g.addEdge(4, 1);
387     g.addEdge(2, 3);
388
389     topologicalSortUtil(5, visited, Stack);
390
391     while (Stack.empty() == false) {
392         cout << Stack.top() << " ";
393         Stack.pop();
394     }
395 }
396
397 }
398
399 int main()
400 {
401
402     Graph g(6);
403     g.addEdge(5, 2);
404     g.addEdge(5, 0);
405     g.addEdge(4, 0);
406     g.addEdge(4, 1);
407     g.addEdge(2, 3);
408
409     topologicalSortUtil(5, visited, Stack);
410
411     while (Stack.empty() == false) {
412         cout << Stack.top() << " ";
413         Stack.pop();
414     }
415 }
416
417 }
418
419 int main()
420 {
421
422     Graph g(6);
423     g.addEdge(5, 2);
424     g.addEdge(5, 0);
425     g.addEdge(4, 0);
426     g.addEdge(4, 1);
427     g.addEdge(2, 3);
428
429     topologicalSortUtil(5, visited, Stack);
430
431     while (Stack.empty() == false) {
432         cout << Stack.top() << " ";
433         Stack.pop();
434     }
435 }
436
437 }
438
439 int main()
440 {
441
442     Graph g(6);
443     g.addEdge(5, 2);
444     g.addEdge(5, 0);
445     g.addEdge(4, 0);
446     g.addEdge(4, 1);
447     g.addEdge(2, 3);
448
449     topologicalSortUtil(5, visited, Stack);
450
451     while (Stack.empty() == false) {
452         cout << Stack.top() << " ";
453         Stack.pop();
454     }
455 }
456
457 }
458
459 int main()
460 {
461
462     Graph g(6);
463     g.addEdge(5, 2);
464     g.addEdge(5, 0);
465     g.addEdge(4, 0);
466     g.addEdge(4, 1);
467     g.addEdge(2, 3);
468
469     topologicalSortUtil(5, visited, Stack);
470
471     while (Stack.empty() == false) {
472         cout << Stack.top() << " ";
473         Stack.pop();
474     }
475 }
476
477 }
478
479 int main()
480 {
481
482     Graph g(6);
483     g.addEdge(5, 2);
484     g.addEdge(5, 0);
485     g.addEdge(4, 0);
486     g.addEdge(4, 1);
487     g.addEdge(2, 3);
488
489     topologicalSortUtil(5, visited, Stack);
490
491     while (Stack.empty() == false) {
492         cout << Stack.top() << " ";
493         Stack.pop();
494     }
495 }
496
497 }
498
499 int main()
500 {
501
502     Graph g(6);
503     g.addEdge(5, 2);
504     g.addEdge(5, 0);
505     g.addEdge(4, 0);
506     g.addEdge(4, 1);
507     g.addEdge(2, 3);
508
509     topologicalSortUtil(5, visited, Stack);
510
511     while (Stack.empty() == false) {
512         cout << Stack.top() << " ";
513         Stack.pop();
514     }
515 }
516
517 }
518
519 int main()
520 {
521
522     Graph g(6);
523     g.addEdge(5, 2);
524     g.addEdge(5, 0);
525     g.addEdge(4, 0);
526     g.addEdge(4, 1);
527     g.addEdge(2, 3);
528
529     topologicalSortUtil(5, visited, Stack);
530
531     while (Stack.empty() == false) {
532         cout << Stack.top() << " ";
533         Stack.pop();
534     }
535 }
536
537 }
538
539 int main()
540 {
541
542     Graph g(6);
543     g.addEdge(5, 2);
544     g.addEdge(5, 0);
545     g.addEdge(4, 0);
546     g.addEdge(4, 1);
547     g.addEdge(2, 3);
548
549     topologicalSortUtil(5, visited, Stack);
550
551     while (Stack.empty() == false) {
552         cout << Stack.top() << " ";
553         Stack.pop();
554     }
555 }
556
557 }
558
559 int main()
560 {
561
562     Graph g(6);
563     g.addEdge(5, 2);
564     g.addEdge(5, 0);
565     g.addEdge(4, 0);
566     g.addEdge(4, 1);
567     g.addEdge(2, 3);
568
569     topologicalSortUtil(5, visited, Stack);
570
571     while (Stack.empty() == false) {
572         cout << Stack.top() << " ";
573         Stack.pop();
574     }
575 }
576
577 }
578
579 int main()
580 {
581
582     Graph g(6);
583     g.addEdge(5, 2);
584     g.addEdge(5, 0);
585     g.addEdge(4, 0);
586     g.addEdge(4, 1);
587     g.addEdge(2, 3);
588
589     topologicalSortUtil(5, visited, Stack);
590
591     while (Stack.empty() == false) {
592         cout << Stack.top() << " ";
593         Stack.pop();
594     }
595 }
596
597 }
598
599 int main()
600 {
601
602     Graph g(6);
603     g.addEdge(5, 2);
604     g.addEdge(5, 0);
605     g.addEdge(4, 0);
606     g.addEdge(4, 1);
607     g.addEdge(2, 3);
608
609     topologicalSortUtil(5, visited, Stack);
610
611     while (Stack.empty() == false) {
612         cout << Stack.top() << " ";
613         Stack.pop();
614     }
615 }
616
617 }
618
619 int main()
620 {
621
622     Graph g(6);
623     g.addEdge(5, 2);
624     g.addEdge(5, 0);
625     g.addEdge(4, 0);
626     g.addEdge(4, 1);
627     g.addEdge(2, 3);
628
629     topologicalSortUtil(5, visited, Stack);
630
631     while (Stack.empty() == false) {
632         cout << Stack.top() << " ";
633         Stack.pop();
634     }
635 }
636
637 }
638
639 int main()
640 {
641
642     Graph g(6);
643     g.addEdge(5, 2);
644     g.addEdge(5, 0);
645     g.addEdge(4, 0);
646     g.addEdge(4, 1);
647     g.addEdge(2, 3);
648
649     topologicalSortUtil(5, visited, Stack);
650
651     while (Stack.empty() == false) {
652         cout << Stack.top() << " ";
653         Stack.pop();
654     }
655 }
656
657 }
658
659 int main()
660 {
661
662     Graph g(6);
663     g.addEdge(5, 2);
664     g.addEdge(5, 0);
665     g.addEdge(4, 0);
666     g.addEdge(4, 1);
667     g.addEdge(2, 3);
668
669     topologicalSortUtil(5, visited, Stack);
670
671     while (Stack.empty() == false) {
672         cout << Stack.top() << " ";
673         Stack.pop();
674     }
675 }
676
677 }
678
679 int main()
680 {
681
682     Graph g(6);
683     g.addEdge(5, 2);
684     g.addEdge(5, 0);
685     g.addEdge(4, 0);
686     g.addEdge(4, 1);
687     g.addEdge(2, 3);
688
689     topologicalSortUtil(5, visited, Stack);
690
691     while (Stack.empty() == false) {
692         cout << Stack.top() << " ";
693         Stack.pop();
694     }
695 }
696
697 }
698
699 int main()
700 {
701
702     Graph g(6);
703     g.addEdge(5, 2);
704     g.addEdge(5, 0);
705     g.addEdge(4, 0);
706     g.addEdge(4, 1);
707     g.addEdge(2, 3);
708
709     topologicalSortUtil(5, visited, Stack);
710
711     while (Stack.empty() == false) {
712         cout << Stack.top() << " ";
713         Stack.pop();
714     }
715 }
716
717 }
718
719 int main()
720 {
721
722     Graph g(6);
723     g.addEdge(5, 2);
724     g.addEdge(5, 0);
725     g.addEdge(4, 0);
726     g.addEdge(4, 1);
727     g.addEdge(2, 3);
728
729     topologicalSortUtil(5, visited, Stack);
730
731     while (Stack.empty() == false) {
732         cout << Stack.top() << " ";
733         Stack.pop();
734     }
735 }
736
737 }
738
739 int main()
740 {
741
742     Graph g(6);
743     g.addEdge(5, 2);
744     g.addEdge(5, 0);
745     g.addEdge(4, 0);
746     g.addEdge(4, 1);
747     g.addEdge(2, 3);
748
749     topologicalSortUtil(5, visited, Stack);
750
751     while (Stack.empty() == false) {
752         cout << Stack.top() << " ";
753         Stack.pop();
754     }
755 }
756
757 }
758
759 int main()
760 {
761
762     Graph g(6);
763     g.addEdge(5, 2);
764     g.addEdge(5, 0);
765     g.addEdge(4, 0);
766     g.addEdge(4, 1);
767     g.addEdge(2, 3);
768
769     topologicalSortUtil(5, visited, Stack);
770
771     while (Stack.empty() == false) {
772         cout << Stack.top() << " ";
773         Stack.pop();
774     }
775 }
776
777 }
778
779 int main()
780 {
781
782     Graph g(6);
783     g.addEdge(5, 2);
784     g.addEdge(5, 0);
785     g.addEdge(4, 0);
786     g.addEdge(4, 1);
787     g.addEdge(2, 3);
788
789     topologicalSortUtil(5, visited, Stack);
790
791     while (Stack.empty() == false) {
792         cout << Stack.top() << " ";
793         Stack.pop();
794     }
795 }
796
797 }
798
799 int main()
800 {
801
802     Graph g(6);
803     g.addEdge(5, 2);
804     g.addEdge(5, 0);
805     g.addEdge(4, 0);
806     g.addEdge(4, 1);
807     g.addEdge(2, 3);
808
809     topologicalSortUtil(5, visited, Stack);
810
811     while (Stack.empty() == false) {
812         cout << Stack.top() << " ";
813         Stack.pop();
814     }
815 }
816
817 }
818
819 int main()
820 {
821
822     Graph g(6);
823     g.addEdge(5, 2);
824     g.addEdge(5, 0);
825     g.addEdge(4, 0);
826     g.addEdge(4, 1);
827     g.addEdge(2, 3);
828
829     topologicalSortUtil(5, visited, Stack);
830
831     while (Stack.empty() == false) {
832         cout << Stack.top() << " ";
833         Stack.pop();
834     }
835 }
836
837 }
838
839 int main()
840 {
841
842     Graph g(6);
843     g.addEdge(5, 2);
844     g.addEdge(5, 0);
845     g.addEdge(4, 0);
846     g.addEdge(4, 1);
847     g.addEdge(2, 3);
848
849     topologicalSortUtil(5, visited, Stack);
850
851     while (Stack.empty() == false) {
852         cout << Stack.top() << " ";
853         Stack.pop();
854     }
855 }
856
857 }
858
859 int main()
860 {
861
862     Graph g(6);
863     g.addEdge(5, 2);
864     g.addEdge(5, 0);
865     g.addEdge(4, 0);
866     g.addEdge(4, 1);
867     g.addEdge(2, 3);
868
869     topologicalSortUtil(5, visited, Stack);
870
871     while (Stack.empty() == false) {
872         cout << Stack.top() << " ";
873         Stack.pop();
874     }
875 }
876
877 }
878
879 int main()
880 {
881
882     Graph g(6);
883     g.addEdge(5, 2);
884     g.addEdge(5, 0);
885     g.addEdge(4, 0);
886     g.addEdge(4, 1);
887     g.addEdge(2, 3);
888
889     topologicalSortUtil(5, visited, Stack);
890
891     while (Stack.empty() == false) {
892         cout << Stack.top() << " ";
893         Stack.pop();
894     }
895 }
896
897 }
898
899 int main()
900 {
901
902     Graph g(6);
903     g.addEdge(5, 2);
904     g.addEdge(5, 0);
905     g.addEdge(4, 0);
906     g.addEdge(4, 1);
907     g.addEdge(2, 3);
908
909     topologicalSortUtil(5, visited, Stack);
910
911     while (Stack.empty() == false) {
912         cout << Stack.top() << " ";
913         Stack.pop();
914     }
915 }
916
917 }
918
919 int main()
920 {
921
922     Graph g(6);
923     g.addEdge(5, 2);
924     g.addEdge(5, 0);
925     g.addEdge(4, 0);
926     g.addEdge(4, 1);
927     g.addEdge(2, 3);
928
929     topologicalSortUtil(5, visited, Stack);
930
931     while (Stack.empty() == false) {
932         cout << Stack.top() << " ";
933         Stack.pop();
934     }
935 }
936
937 }
938
939 int main()
940 {
941
942     Graph g(6);
943     g.addEdge(5, 2);
944     g.addEdge(5, 0);
945     g.addEdge(4, 0);
946     g.addEdge(4, 1);
947     g.addEdge(2, 3);
948
949     topologicalSortUtil(5, visited, Stack);
950
951     while (Stack.empty() == false) {
952         cout << Stack.top() << " ";
953         Stack.pop();
954     }
955 }
956
957 }
958
959 int main()
960 {
961
962     Graph g(6);
963     g.addEdge(5, 2);
964     g.addEdge(5, 0);
965     g.addEdge(4, 0);
966     g.addEdge(4, 1);
967     g.addEdge(2, 3);
968
969     topologicalSortUtil(5, visited, Stack);
970
971     while (Stack.empty() == false) {
972         cout << Stack.top() << " ";
973         Stack.pop();
974     }
975 }
976
977 }
978
979 int main()
980 {
981
982     Graph g(6);
983     g.addEdge(5, 2);
984     g.addEdge(5, 0);
985     g.addEdge(4, 0);
986     g.addEdge(4, 1);
987     g.addEdge(2, 3);
988
989     topologicalSortUtil(5, visited, Stack);
990
991     while (Stack.empty() == false) {
992         cout << Stack.top() << " ";
993         Stack.pop();
994     }
995 }
996
997 }
998
999 int main()
1000 {
1001
1002     Graph g(6);
1003     g.addEdge(5, 2);
1004     g.addEdge(5, 0);
1005     g.addEdge(4, 0);
1006     g.addEdge(4, 1);
1007     g.addEdge(2, 3);
1008
1009     topologicalSortUtil(5, visited, Stack);
1010
1011     while (Stack.empty() == false) {
1012         cout << Stack.top() << " ";
1013         Stack.pop();
1014     }
1015 }
1016
1017 }
1018
1019 int main()
1020 {
1021
1022     Graph g(6);
1023     g.addEdge(5, 2);
1024     g.addEdge(5, 0);
1025     g.addEdge(4, 0);
1026     g.addEdge(4, 1);
1027     g.addEdge(2, 3);
1028
1029     topologicalSortUtil(5, visited, Stack);
1030
1031     while (Stack.empty() == false) {
1032         cout << Stack.top() << " ";
1033         Stack.pop();
1034     }
1035 }
1036
1037 }
1038
1039 int main()
1040 {
1041
1042     Graph g(6);
1043     g.addEdge(5, 2);
1044     g.addEdge(5, 0);
1045     g.addEdge(4, 0);
1046     g.addEdge(4, 1);
1047     g.addEdge(2, 3);
1048
1049     topologicalSortUtil(5, visited, Stack);
1050
1051     while (Stack.empty() == false) {
1052         cout << Stack.top() << " ";
1053         Stack.pop();
1054     }
1055 }
1056
1057 }
1058
1059 int main()
1060 {
1061
1062     Graph g(6);
1063     g.addEdge(5, 2);
1064     g.addEdge(5, 0);
1065     g.addEdge(4, 0);
1066     g.addEdge(4, 1);
1067     g.addEdge(2, 3);
1068
1069     topologicalSortUtil(5, visited, Stack);
1070
1071     while (Stack.empty() == false) {
1072         cout << Stack.top() << " ";
1073         Stack.pop();
1074     }
1075 }
1076
1077 }
1078
1079 int main()
1080 {
1081
1082     Graph g(6);
1083     g.addEdge(5, 2);
1084     g.addEdge(5, 0);
1085     g.addEdge(4, 0);
1086     g.addEdge(4, 1);
1087     g.addEdge(2, 3);
1088
1089     topologicalSortUtil(5, visited, Stack);
1090
1091     while (Stack.empty() == false) {
1092         cout << Stack.top() << " ";
1093         Stack.pop();
1094     }
1095 }
1096
1097 }
1098
1099 int main()
1100 {
1101
1102     Graph g(6);
1103     g.addEdge(5, 2);
1104     g.addEdge(5, 0);
1105     g.addEdge(4, 0);
1106     g.addEdge(4, 1);
1107     g.addEdge(2, 3);
1108
1109     topologicalSortUtil(5, visited, Stack);
1110
1111     while (Stack.empty() == false) {
1112         cout << Stack.top() << " ";
1113         Stack.pop();
1114     }
1115 }
1116
1117 }
1118
1119 int main()
1120 {
1121
1122     Graph g(6);
1123     g.addEdge(5, 2);
1124     g.addEdge(5, 0);
1125     g.addEdge(4, 0);
1126     g.addEdge(4, 1);
1127     g.addEdge(2, 3);
1128
1129     topologicalSortUtil(5, visited, Stack);
1130
1131     while (Stack.empty() == false) {
1132         cout << Stack.top() << " ";
1133         Stack.pop();
1134     }
1135 }
1136
1137 }
1138
1139 int main()
1140 {
1141
1142     Graph g(6);
1143     g.addEdge(5, 2);
1144     g.addEdge(5, 0);
1145     g.addEdge(4, 0);
1146     g.addEdge(4, 1);
1147     g.addEdge(2, 3);
1148
1149     topologicalSortUtil(5, visited, Stack);
1150
1151     while (Stack.empty() == false) {
1152         cout << Stack.top() << " ";
1153         Stack.pop();
1154     }
1155 }
1156
1157 }
1158
1159 int main()
1160 {
1161
1162     Graph g(6);
1163     g.addEdge(5, 2);
1164     g.addEdge(5, 0);
1165     g.addEdge(4, 0);
1166     g.addEdge(4, 1);
1167     g.addEdge(2, 3);
1168
1169     topologicalSortUtil(5, visited, Stack);
1170
1171     while (Stack.empty() == false) {
1172         cout << Stack.top() << " ";
1173         Stack.pop();
1174     }
1175 }
1176
1177 }
1178
1179 int main()
1180 {
1181
1182     Graph g(6);
1183     g.addEdge(5, 2);
1184     g.addEdge(5, 0);
1185     g.addEdge(4, 0);
1186     g.addEdge(4, 1);
1187     g.addEdge(2, 3);
1188
1189     topologicalSortUtil(5, visited, Stack);
1190
1191     while (Stack.empty() == false) {
1192         cout << Stack.top() << " ";
1193         Stack.pop();
1194     }
1195 }
1196
1197 }
1198
1199 int main()
1200 {
1201
1202     Graph g(6);
1203     g.addEdge(5, 2);
1204     g.addEdge(5, 0);
1205     g.addEdge(4, 0);
1206     g.addEdge(4, 1);
1207     g.addEdge(2, 3);
1208
1209     topologicalSortUtil(5, visited, Stack);
1210
1211     while (Stack.empty() == false) {
1212         cout << Stack.top() << " ";
1213         Stack.pop();
1214     }
1215 }
1216
1217 }
1218
1219 int main()
1220 {
1221
1222     Graph g(6);
1223     g.addEdge(5, 2);
1224     g.addEdge(5, 0);
1225     g.addEdge(4, 0);
1226     g.addEdge(4, 1);
1227     g.addEdge(2, 3);
1228
1229     topologicalSortUtil(5, visited, Stack);
1230
1231     while (Stack.empty() == false) {
1232         cout << Stack.top() << " ";
1233         Stack.pop();
1234     }
1235 }
1236
1237 }
1238
1239 int main()
1240 {
1241
1242     Graph g(6);
1243     g.addEdge(5, 2);
1244     g.addEdge(5, 0);
1245     g.addEdge(4, 0);
1246     g.addEdge(4, 1);
1247     g.addEdge(2, 3);
1248
1249     topologicalSortUtil(5, visited, Stack);
1250
1251     while (Stack.empty() == false) {
1252         cout << Stack.top() << " ";
1253         Stack.pop();
1254     }
1255 }
1256
1257 }
1258
1259 int main()
1260 {
1261
1262     Graph g(6);
1263     g.addEdge(5, 2);
1264     g.addEdge(5, 0);
1265     g.addEdge(4, 0);
1266     g.addEdge(4, 1);
1267     g.addEdge(2, 3);
1268
1269     topologicalSortUtil(5, visited, Stack);
1270
1271     while (Stack.empty() == false) {
1272         cout << Stack.top() << " ";
1273         Stack.pop();
1274     }
1275 }
1276
1277 }
1278
1279 int main()
1280 {
1281
1282     Graph g(6);
1283     g.addEdge(5, 2);
1284     g.addEdge(5, 0);
1285     g.addEdge(4, 0);
1286     g.addEdge(4, 1);
1287     g.addEdge(2, 3);
1288
1289     topologicalSortUtil(5, visited, Stack);
1290
1291     while (Stack.empty() == false) {
1292         cout << Stack.top() << " ";
1293         Stack.pop();
1294     }
1295 }
1296
1297 }
1298
1299 int main()
1300 {
1301
1302     Graph g(6);
1303     g.addEdge(5, 2);
1304     g.addEdge(5, 0);
1305     g.addEdge(4, 0);
1306     g.addEdge(4, 1);
1307     g.addEdge(2, 3);
1308
1309     topologicalSortUtil(5, visited, Stack);
1310
1311     while (Stack.empty() == false) {
1312         cout << Stack.top() << " ";
1313         Stack.pop();
1314     }
1315 }
1316
1317 }
1318
1319 int main
```



```
#include <vector>

#include <climits>

#include <cstring>

using namespace std;

bool isSafe(vector<vector<int>> &mat, vector<vector<bool>> &visited, int x, int y)

{

    return (x >= 0 && x < mat.size() && y >= 0 && y < mat[0].size()) &&

        mat[x][y] == 1 && !visited[x][y];

}

void findShortestPath(vector<vector<int>> &mat, vector<vector<bool>> &visited,

    int i, int j, int x, int y, int &min_dist, int dist)

{

    if (i == x && j == y)

    {

        min_dist = min(dist, min_dist);

        return;

    }

    visited[i][j] = true;

    if (isSafe(mat, visited, i + 1, j)) {

        findShortestPath(mat, visited, i + 1, j, x, y, min_dist, dist + 1);

    }

    if (isSafe(mat, visited, i, j + 1)) {

        findShortestPath(mat, visited, i, j + 1, x, y, min_dist, dist + 1);

    }

}
```



```
}

if (isSafe(mat, visited, i - 1, j)) {
    findShortestPath(mat, visited, i - 1, j, x, y, min_dist, dist + 1);
}

if (isSafe(mat, visited, i, j - 1)) {
    findShortestPath(mat, visited, i, j - 1, x, y, min_dist, dist + 1);
}

visited[i][j] = false;
}

int findShortestPathLength(vector<vector<int>> &mat, pair<int, int> &src,
    pair<int, int> &dest)
{

    if (mat.size() == 0 || mat[src.first][src.second] == 0 ||
        mat[dest.first][dest.second] == 0) {
        return -1;
    }

    int M = mat.size();
    int N = mat[0].size();
    vector<vector<bool>> visited;
    visited.resize(M, vector<bool>(N));
    int min_dist = INT_MAX;

    findShortestPath(mat, visited, src.first, src.second, dest.first, dest.second,
```

```
min_dist, 0);
```

```
if (min_dist != INT_MAX) {  
    return min_dist;  
}
```

```
return -1;  
}
```

```
int main()
```

```
{  
    vector<vector<int>> mat =  
    {  
        { 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },  
        { 0, 1, 1, 1, 1, 1, 0, 1, 0, 1 },  
        { 0, 0, 1, 0, 1, 1, 1, 0, 0, 1 },  
        { 1, 0, 1, 1, 1, 0, 1, 1, 0, 1 },  
        { 0, 0, 0, 1, 0, 0, 0, 1, 0, 1 },  
        { 1, 0, 1, 1, 1, 0, 0, 1, 1, 0 },  
        { 0, 0, 0, 0, 1, 0, 0, 1, 0, 1 },  
        { 0, 1, 1, 1, 1, 1, 1, 1, 0, 0 },  
        { 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },  
        { 0, 0, 1, 0, 0, 1, 1, 0, 0, 1 },  
    };  

```

```
pair<int, int> src = make_pair(0, 0);
```

```
pair<int, int> dest = make_pair(7, 5);
```

```
int min_dist = findShortestPathLength(mat, src, dest);
```

```

if (min_dist != -1)
{
    cout << "The shortest path from source to destination "
        "has length " << min_dist;
}
else {
    cout << "Destination cannot be reached from a given source";
}

return 0;
}

```

Output :-

The screenshot shows a web browser window with the OnlineGDB compiler. The code in the editor is as follows:

```

11 #include <string>
12 using namespace std;
13
14 bool isSafe(vector<vector<int>> &mat, vector<vector<bool>> &visited, int x, int y)
15 {
16     return (x >= 0 && x < mat.size() && y >= 0 && y < mat[0].size()) &&
17         mat[x][y] == 1 && !visited[x][y];

```

The console output shows:

```

input
The shortest path from source to destination has length 12
...Program finished with exit code 0
Press ENTER to exit console.

```

Evaluation Grid :

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Student Performance (Conduct of experiment) objectives/Outcomes.		12
2.	Viva Voce		10
3.	Submission of Work Sheet (Record)		8
	Total		30