# Lego Game (Assigment 14)

June 4, 2018

Student: Ghibuci Andrei

Teacher: Badica Costin

Assistant Teacher: Becheru Alexandru

Computer science in English

Group 1.2 A

1st Year

# Introduction

A depth-first search (DFS) is an algorithm for traversing a finite graph. DFS visits the child nodes before visiting the sibling nodes; that is, it traverses the depth of any particular path before exploring its breadth. A stack (often the program's call stack via recursion) is generally used when implementing the algorithm.

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG. For example, a topological sorting of the following graph is 5 4 2 3 1 0. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is 4 5 2 3 1 0. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).

# Problem statement

Alexander, which is a toddler, has received as a birthday gift a Lego game. How- ever, he was not able to construct the toy Lego, as there are many pieces. Alexan- der has observed that each piece is numbered with a number starting from 1 to 1000. The instructions of the game state the pieces that need to be assembled before each an every piece. Help Alexander to build his Lego toy by developing an application which determines the correct order in which the piece of Lego need to be assembled. he application should implement two different algorithms to help Alexander.

So, I had to make an orientated acyclic graph and tropological sort it. However, I wasn't able to make it sort from 1 to 1000, so i adapted it to sort a part of them. The problem was that I couldn't fix two bugs. First bug was when my graph was generated even( source node == destination node) and the second one was when my graph returned to the begining node too soon( source[1] - destination[2], destination[2] - source[1]).

# Pseudocode

First, we will use an adjacency matrix for the directed graph representation, being an easier method. And another two adjaceny vectors for the source and destination nodes.
We will use functions for each operation required by the application, with integer parameters, which will be called by the main program.
Here are the important functions employed by the program:

```
orientations (int oreintation[][], int i, int n)
1.      int source
2.      int destination
3.      srand(time(NULL))
4.      while i <- 1 to n execute
            4.1 source = random generation between 1 and 1000
            4.2 destination = random generation between 1 and 1000
            4.3 while adjancency_source[source] = 1 execute
                    4.3.1 source = random generation between 1 and 1000
            4.4 ajancency_source[source] = 1
            4.5 while adjancency_destination[destination] = 1 execute
                    4.5.1 destination = random generation between 1 and 1000
            4.6 adjancency_destination[destination] = 1
            4.7 orientation[source][destination] = 1
            4.8 i++
```

```
depth_first_search(int source)
1.      int destination
2.      int i
3.      reach[source] = 1
4.      for i <- 1 to 1000 do
       4.1 if orientation[source][i] = 1 and reach[i] = 0 then
                4.1.1 output source <- i
                4.1.2 counter++
                4.1.3 depth_first_search(i)
```

# Application design

The library contains the header ***functions.h*** which has all the function prototypes to compute the required operations. These are all of them:
—void orientations(int orientation[][], int i , int n)
—void depth first search(int node)

The source file ***functions.c*** has all the function implementations.
Function depth first search(int node) includes some steps, like:
1) We visit the starting node and print it,
2) We visit the nearest unvisited neighbor that can be reached and do the same for that one,
3) If we exhaust all our options for a node, we return back to the previous one, repeating step ***2***.

The adjacency matrix is stored as a 2D array called 'orientation'. This array is used by all of our functions. We will use a "source" node and a "destination" one, with the value '0' for unconected graph and 1 for orientation from source to destination. ***source*** represents the starting node from where we want to begin the traversal of the graph. ***n*** is simply the number of nodes. The function works recursively.

Function Orientations(int orientation[][1001], int i, int n) - works as follows:
1) We will random generate the source and the destination node,
2) Next we will check the adjancecy matrices of source and destination nodes to see if the random generation was allready generated ,
3) After we checked and assigned the random nodes we assign 1 to the orientation matrix which means the conection from the source node to destination node is available and checked .

The final source file, ***main.c*** itself, uses all of the functions from the ***functions.c***.

After calling all the functions, the main function will randomly generate a tropological sort between 1 and 1000 which represents the correct order of the Lego. As for the bugs as mentioned I tried to fix them and wasn't able to.

## Source Code

```
//-----------------------------functions.h-------------------------------

ifndef FUNCTIONS_H_INCLUDED
#define FUNCTIONS_H_INCLUDED

void orientations ( int orientation[][1001] , int i , int n);
void depth_first_search( int node );

#endif // FUNCTIONS_H_INCLUDED
```

```
//-----------------------------functions.c-------------------------------
#include "functions.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/**
* This vector is a representation of the current node and if this node
    was allready visited.
*/
int reach[1001];

/**
* This variable will be used to counter how many pieces were used in the
    sort.
*/
int counter = 0;

/**
* A matrix that retain the orientations between the nodes of the graph.
*/
int orientation[1001][1001];

/**
* Vector that retain if the source node allready appeared in the sort
*/
int adiacency_source[1001];

/**
* Vector that retain if the destination node allready appeared in the
    sort
*/
int adjacency_destination[1001];

/**
* Function that reads and makes the graph orientated.
```

```
*/
void orientations( int orientation [][1001] , int i , int n ){

    int source;
    int destination;

    srand(time(NULL));

    while( i <= n){

        source = rand()%1000 + 1;
        destination = rand()%1000 + 1;

        while( adjacency_source[source] == 1 ) //Randomly generate
            source node if it haven't allready occured.
             source = rand()%1000 + 1;
       adjacency_source[source] = 1;

        while(adjacency_destination[destination] == 1 )// Randomly
            generate destination node if it haven't allready occured.
             destination = rand()%1000 + 1;
        adjacency_destination[destination] = 1;

        orientation[source][destination] = 1; // Assigning the
            orietation from the source to node.
        i++;
       }
}

/**
* This function will do the tropological sort of the graph.
*/
void depth_first_search(int source){

     int destination;
     int i;
     reach[source]=1;

     for(i = 1;i <= 1000; i++)
        if(orientation[source][i] && !reach[i]){

           printf("\n %d->%d",source,i);// Printing the tropological
               sort.
           counter++;// Counting the number of pieces used.
           depth_first_search(i);
           }
}

//--------------------------------main.c--------------------------------
```

```c
#include "functions.h"
#include <stdio.h>
#include <stdlib.h>

/**
* Matrix that was used allready in the function orientations and
    explained.
*/
int orientation[1001][1001];

/**
*Variable that was used for counting the number of used pieces.
*/
int counter;

/**
* Main function of my project
*/
int main()
{
    int source;
    int destination;

    orientations(orientation, 1, 1000);
    depth_first_search(rand()%1000+1);

    printf("\n %d ", counter);

    return 0;
}
```

# Experiments and results

```
First Experiment:
1000 nodes
Starting node: 511
Pieces Used : 676
Tropological Sort :
511->506
506->622
622->986
986->562
562->729
729->976
976->685
685->678
678->424
424->353
353->546
546->411
411->75
75->571
571->398
398->612
612->788
788->437
437->130
130->963
963->91
91->115
115->818
818->826
826->136
136->807
807->478
478->652
652->857
857->973
973->553
553->701
701->234
234->863
863->756
756->387
387->53
53->854
854->423
423->727
727->493
493->899
```

```
899->885
885->327
327->737
737->811
811->391
391->11
11->274
274->714
714->375
375->558
558->590
590->797
797->12
12->4
4->734
734->214
214->721
721->650
650->665
665->560
560->994
994->175
175->830
830->408
408->455
455->909
909->592
592->331
331->314
314->535
535->828
828->654
654->853
853->469
469->25
25->956
956->138
138->141
141->207
207->122
122->939
939->877
877->472
472->522
522->577
577->799
799->787
787->540
540->95
95->180
```

```
180->252
252->279
279->266
266->709
709->931
931->428
428->172
172->705
705->232
232->895
895->216
216->468
468->741
741->657
657->176
176->57
57->247
247->218
218->444
444->954
954->121
121->431
431->246
246->90
90->611
611->941
941->840
840->168
168->498
498->969
969->583
583->171
171->948
948->570
570->450
450->621
621->503
503->712
712->849
849->846
846->587
587->98
98->667
667->38
38->415
415->873
873->51
51->964
964->396
396->317
```

```
317->333
333->183
183->674
674->959
959->618
618->221
221->212
212->832
832->866
866->322
322->981
981->740
740->273
273->62
62->458
458->166
166->211
211->526
526->77
77->362
362->257
257->524
524->915
915->512
512->46
46->692
692->835
835->329
329->951
951->888
888->361
361->482
482->624
624->109
109->600
600->249
249->297
297->975
975->842
842->970
970->920
920->263
263->215
215->120
120->949
949->356
356->148
148->702
702->579
579->201
```

```
201->708
708->5
5->716
716->655
655->301
301->283
283->81
81->377
377->3
3->922
922->984
984->957
957->160
160->443
443->449
449->489
489->719
719->344
344->633
633->118
118->623
623->94
94->289
289->784
784->372
372->608
608->287
287->241
241->802
802->230
230->378
378->881
881->494
494->615
615->549
549->161
161->298
298->271
271->812
812->629
629->995
995->52
52->547
547->192
192->661
661->892
892->193
193->574
574->809
809->154
```

```
154->23
23->958
958->591
591->236
236->292
292->367
367->731
731->328
328->613
613->658
658->36
36->924
924->41
41->258
258->580
580->805
805->593
593->61
61->785
785->54
54->531
531->996
996->806
806->564
564->823
823->47
47->770
770->744
744->324
324->859
859->345
345->649
649->780
780->673
673->746
746->280
280->101
101->476
476->500
500->227
227->107
107->240
240->764
764->332
332->217
217->244
244->461
461->371
371->906
906->79
```

```
79->149
149->132
132->194
194->738
738->277
277->89
89->599
599->833
833->142
142->750
750->769
769->484
484->433
433->186
186->988
988->491
491->831
831->554
554->879
879->838
838->42
42->474
474->262
262->893
893->700
700->720
720->688
688->874
874->203
203->27
27->627
627->771
771->382
382->92
92->572
572->852
852->543
543->946
946->926
926->773
773->697
697->518
518->792
792->291
291->872
872->261
261->516
516->426
426->908
908->856
```

```
856->992
992->960
960->901
901->589
589->275
275->735
735->595
595->97
97->164
164->358
358->439
439->33
33->602
602->363
363->430
430->816
816->878
878->687
687->417
417->887
887->584
584->766
766->453
453->173
173->418
418->153
153->707
707->238
238->169
169->349
349->537
537->393
393->604
604->783
783->753
753->309
309->517
517->789
789->432
432->127
127->556
556->841
841->684
684->913
913->929
929->189
189->670
670->58
58->225
225->30
```

```
30->388
388->870
870->985
985->1
1->847
847->413
413->436
436->989
989->145
145->475
475->732
732->462
462->233
233->268
268->29
29->585
585->337
337->134
134->364
364->253
253->594
594->359
359->569
569->798
798->155
155->206
206->219
219->559
559->405
405->198
198->717
717->796
796->616
616->967
967->267
267->900
900->794
794->777
777->282
282->943
943->269
269->601
601->982
982->573
573->542
542->447
447->406
406->695
695->610
610->754
```

```
754->187
187->286
286->752
752->538
538->736
736->536
536->284
284->485
485->745
745->776
776->588
588->614
614->243
243->321
321->26
26->747
747->373
373->952
952->477
477->758
758->545
545->603
603->32
32->821
821->104
104->210
210->576
576->341
341->945
945->181
181->307
307->167
167->124
124->69
69->507
507->72
72->795
795->979
979->827
827->530
530->380
380->669
669->177
177->728
728->311
311->401
401->814
814->916
916->384
384->195
```

```
195->897
897->942
942->20
20->551
551->715
715->761
761->755
755->668
668->404
404->190
190->864
864->204
204->786
786->868
868->93
93->682
682->446
446->222
222->567
567->822
822->44
44->965
965->144
144->646
646->235
235->824
824->191
191->810
810->643
643->778
778->937
937->568
568->129
129->100
100->343
343->205
205->70
70->239
239->354
354->947
947->386
386->921
921->8
8->351
351->442
442->159
159->165
165->596
596->73
73->950
```

```
950->74
74->925
925->582
582->394
394->131
131->793
793->495
495->200
200->914
914->889
889->641
641->40
40->689
689->660
660->938
938->112
112->676
676->119
119->883
883->653
653->76
76->907
907->202
202->762
762->515
515->66
66->923
923->250
250->790
790->791
791->837
837->102
102->765
765->726
726->933
933->927
927->834
834->113
113->628
628->869
869->85
85->487
487->607
607->370
370->894
894->414
414->803
803->128
128->999
999->340
```

```
340->690
690->178
178->464
464->260
260->904
904->24
24->813
813->158
158->15
15->903
903->110
110->397
397->918
918->686
686->898
898->910
910->199
199->638
638->742
742->185
185->326
326->152
152->318
318->126
126->725
725->779
779->248
248->224
224->366
366->285
285->251
251->303
303->105
105->617
617->302
302->265
265->163
163->223
223->365
365->402
402->473
473->497
497->264
264->631
631->640
640->441
441->335
335->421
421->972
972->281
```

```
 281->488
 488->699
 699->730
 730->480
 480->325
 325->336
 336->886
 886->63
 63->147
 147->706
 706->319
 319->316
 316->862
 862->998
 998->278
 278->775
 775->917
 917->672
 672->348
 348->323
 323->713
 713->642
 642->486
 486->991
 991->103
 103->808
 808->858
 858->748
 748->288
 288->87
 87->125
 125->179
 179->146
 146->465

2. 10 nodes
The tropological sort is :
 7->2
 2->4
 4->5
 5->8
 The number of pieces used : 4
```

## Conclusions

Personally, It was a challenging assigment for my programming skills. I tried hard to complete the tasks and I was a little dissapointed that I couldn't make the program work as it should have. However, I developed a software which I am proud of, I worked hard and I tried my best and I think these kind of tasks are a great choice to assign to students. It was a pleasure to take part in such assigment.

## References

1)https://www.geeksforgeeks.org/topological-sorting/

2)https://en.wikipedia.org/wiki/Depth-first-search

3)http://www.sharelatex.com