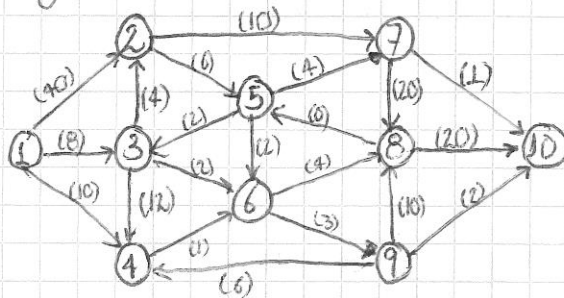


## Taller 2 - Algoritmos

Jorallan Campo Pongel

1)



a) Dijkstra:

- 1) Iniciar distancias con valor  $\infty$ , excepto la del nodo inicial ( $= 0$ )
- 2)  $a$  = nodo inicial
- 3) Visitar todos los vecinos de  $a$  y los marcamos al visitarlos ( $V_i$ )
- 4) Calcular distancia del nodo actual a sus vecinos.  $dT(V_i) = D_a + d(a, V_i)$ . (Distancia total igual a distancia acumulada + distancia del nodo actual al vecino)
- 5) Si la distancia calculada es menor que la almacenada, se actualiza como nueva distancia tentativa.
- 6) Se marca como completado el nodo  $a$ .
- 7) Se toma un nuevo nodo inicial (el de menor distancia en  $D$ ) y se repite desde el paso 3 mientras existan nodos no marcados.

Al terminar el algoritmo  $D$  tendrá las distancias entre los nodos.

b) Bellman - Ford

- 1) Iniciar distancias con valor  $\infty$ , excepto el nodo inicial.
- 2) Crear diccionario de Dist. Finales y de padres
- 3) Visitar cada arista del grafo  $n-1$  veces ( $n = \#$  de nodos)
- 4) Comprobar que no haya ciclos negativos.
- 5) Se obtiene una lista de los vértices ordenados por la ruta más corta.

c) Floyd - Warshal: Comparar todos los posibles caminos a través de un grafo entre cada par de vértices

- 1) Definir matrices de vértices  $C, D$
- 2) Inicializar constante  $k=1$
- 3) Seleccionar fila y columna  $k$  de  $C$  y para  $i \neq j \neq k$ :
- 4) Si  $(C_{ik} + C_{kj}) < C_{ij} \Rightarrow D_{ij} = D_{kj} \wedge C_{ij} = C_{ik} + C_{kj}$
- 5) Si no  $\Rightarrow$  No hay cambios.
- 6) Si  $k \leq n \Rightarrow k++$ ; else  $\Rightarrow$  Stop.
- 7)  $C$ , al final, contiene los costos óptimos para ir de un vértice a otro y  $D$  tiene los penúltimos vértices de los caminos óptimos entre dos nodos.



## 2) Problem set 6

- PROGRAMMING A REDUCTION

[illegible]

- **REDUCTION: K-CLIQUE TO DECISION**

Class 10  
Position Set 8

1. Quiz: Programming a Reduction

2. Quiz: Reduction & Clap to Death...

3. Quiz: Polynomial vs. Exponential

4. Quiz: Prime Factors for Cows

5. Quiz: NP vs. P vs. NP

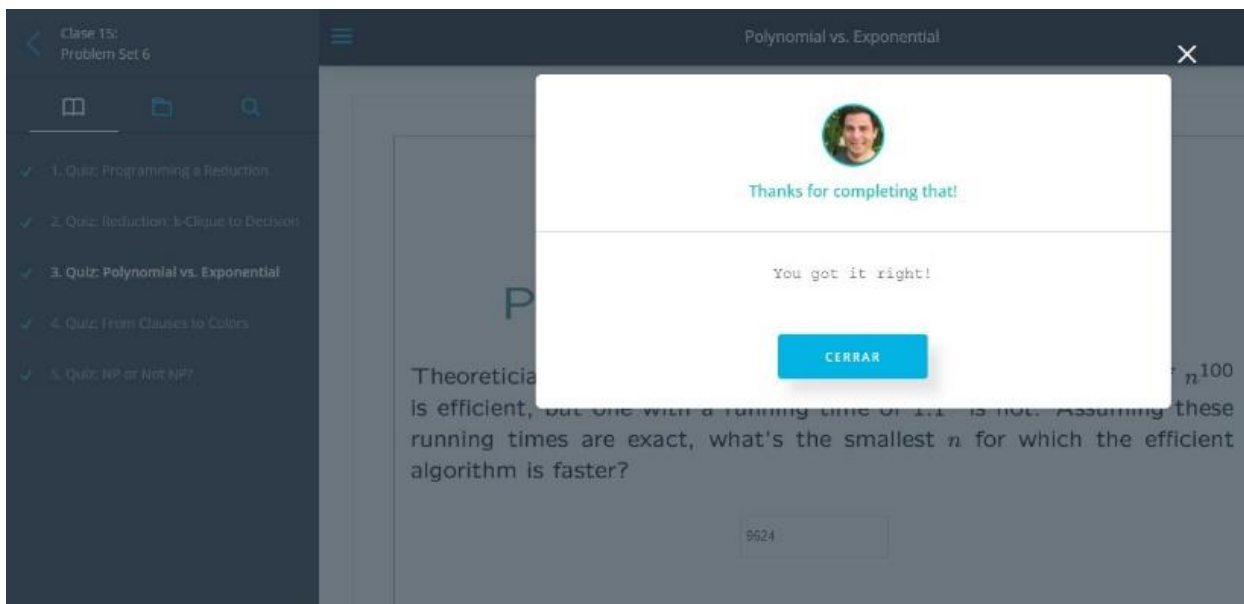
```

1 // Decision problems are often just as hard as actually representing an answer.
2 // One has a 0-1 string and he must give a solution to the 0-1 class decision
3 // problem, which
4 // is input, and
5 // is the graph.
6 // which takes a
7 // so will also be
8
9 // Return a list
10
11 import itertools
12 def is_subset(s):
13     for i in range(1, len(s)):
14         return s[i]
15     return s
16 if s == s:
17     return s
18 return s
19
20 # Check if the s
21 def is_subset(s):
22     for i in range(1, len(s)):
23         return s[i]
24
25 # Return a list
26
27 # Return a list
28
29 # Return a list
30
31 # Return a list
32
33
34
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
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
96
```



## NP or Not NP? That is the Question

Select all the problems below that are in NP. Hint: Think about whether or not each one has a short accepting certificate.

- ☒ **Connectivity:** Is there a path from  $x$  to  $y$  in  $G$ ?
- ☒ **Short path:** Is there a path from  $x$  to  $y$  in  $G$  that is no more than  $k$  steps long?
- ☐ **Fewest colors:** Is  $k$  the absolute minimum number of colors with which  $G$  can be colored?
- ☒ **Near Clique:** Is there a group of  $k$  nodes in  $G$  that has at least  $s$  pairs that are connected?
- ☒ **Partitioning:** Can we group the nodes of  $G$  into two groups of size  $n/2$  so that there are no more than  $k$  edges between the two groups.
- ☐ **Exact coloring count:** Are there exactly  $s$  ways to color graph  $G$  with  $k$  colors?

### 3)FINAL TEST

- BIPARTITE

Class 17:  
Final Assessment

📖

📁

🔍

✓ 1. Quiz: Bipartite

✓ 2. Quiz: Feel the Love

✓ 3. Quiz: Weighted Graph

✓ 4. Quiz: Finding the Best Flight

✓ 5. Quiz: Constantly Connected


✓ 6. Quiz: Distance Oracle (I)

✓ 7. Quiz: Distance Oracle (II)

✓ 8. Quiz: Finding a Favor

Bipartite

✕



Thanks for completing that!

Correct!

CERRAR

```
39 - def test():
40
41 - def make_link(G, n1, n2):
42 -     if n1 not in G:
43         G[n1] = set()
44         (G[n1])[n2]
45 -     if n2 not in G:
46         G[n2] = set()
47         (G[n2])[n1]
48     return G
49
50
51 - def test():
52     edges = [(1, 2), (1, 3), (2, 3)]
53     G = {}
54     for n1, n2 in edges:
55         make_link(G, n1, n2)
56     g1 = bipartite(G)
57     assert (g1 ==
58             g1 ==
59             edges = [(1, 2), (1, 3), (2, 3)]
60             G = {}
61             for n1, n2 in edges:
62                 make_link(G, n1, n2)
63             g1 = bipartite(G)
64             assert g1 == None
65
66
```

Correct!

- Feel the love

Class 17:  
Final Assessment

1. Quiz: Bipartite

2. Quiz: Feel the Love

3. Quiz: Weighted Graph

4. Quiz: Finding the Best Flight

5. Quiz: Constantly Connected

6. Quiz: Distance Orade (I)

7. Quiz: Distance Orade (II)

8. Quiz: Finding a Favor

Feel the Love

11 #code from https:  
12 #used by: Joacamp  
13  
14 import heapq  
15 from collections  
16  
17 def feel\_the\_love  
18 # return a pa  
19 # with 'I' as  
20 # or None if  
21 path = dijkst  
22 if not j in p  
23 return No  
24  
25 node\_a, node\_  
26 path\_a = path  
27 path\_b = (dij  
28  
29 return path\_a  
30  
31 def max\_weight\_edge(G, i):  
32 max\_so\_far = -float('inf')  
33 edge = None  
34 reachable = dijkstra\_path(G, i)  
35 for node in G:  
36 for neighbor in G[node]:  
37 if (G[node])[neighbor] > max\_so\_far and node in reachable:  
38 max\_so\_far = (G[node])[neighbor]  
39 edge = node, neighbor  
40

Thanks for completing that!

Correct!

CERRAR

## • Weighted Graph

←

Clase 17:  
Final Assessment

📖

📁

🔍

✓ 1. Quiz: Bipartite

✓ 2. Quiz: Feel the Love

✓ 3. Quiz: Weighted Graph

✓ 4. Quiz: Finding the Best Flight

✓ 5. Quiz: Constantly Connected


✓ 6. Quiz: Distance Oracle (I)

✓ 7. Quiz: Distance Oracle (II)

✓ 8. Quiz: Finding a Favor

Weighted Graph

✕



Thanks for completing that!

Correct!

CERRAR

```
1 # In lecture, we
2 # where edges were
3 # books they appear
4 # with edges between
5 # number of comic
6 #
7 # In this assignment
8 # comic book characters
9 # that a randomly
10 # the characters.
11 #
12 #code from https://
13 #used by: Joacamp
14 import itertools
15 from collections
16 import cPickle
17
18 marvel = cPickle
19 characters = cPickle
20
21 def create_weighted_graph(bipartiteG, characters):
22     G = defaultdict(dict)
23     for char_a, char_b in itertools.combinations(characters, 2):
24         a_books = set(bipartiteG[char_a])
25         b_books = set(bipartiteG[char_b])
26
27         inter_book_num = float(len(a_books.intersection(b_books)))
28         if inter_book_num == 0:
29             continue
30
31
32
33
34
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
```

Correct!

## • Constantly Connected

←

Clase 17:  
Final Assessment

📖

📁

🔍

✓ 1. Quiz: Bipartite

✓ 2. Quiz: Feel the Love

✓ 3. Quiz: Weighted Graph

✓ 4. Quiz: Finding the Best Flight

✓ 5. Quiz: Constantly Connected


✓ 6. Quiz: Distance Oracle (I)

✓ 7. Quiz: Distance Oracle (II)

✓ 8. Quiz: Finding a Favor

Constantly Connected

✕



Thanks for completing that!

Correct!

CERRAR

```
1 # Design and Implement
2 # graph and then
3 # graph" for any
4 #
5 # "process_graph"
6 # you can store a
7 # global variable
8 #
9 #code from https://
10 #used by: Joacamp
11 global G = {}
12
13 def process_graph
14     global global
15     global G = G
16     for node in G
17         to_visit
18
19         while to
20             new_node = to_visit.pop()
21
22             global G[node][new_node] = 1
23             global G[new_node][node] = 1
24
25             for x in global G[new_node]:
26                 if x not in global G[node]:
27                     to_visit += [x]
28
29 # When being graded, "is_connected" will be called
30 # many times so this routine needs to be quick
31
32
33
34
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
```

- Distance Oracle(I)

Clase 17:  
Final Assessment

1. Quiz: Bipartite

2. Quiz: Feel the Love

3. Quiz: Weighted Graph

4. Quiz: Finding the Best Flight

5. Quiz: Constantly Connected

6. Quiz: Distance Oracle (I)

7. Quiz: Distance Oracle (II)

8. Quiz: Finding a Favor

Distance Oracle (I)

23 # that node  
24 #  
25 - def create\_labels  
26 # BFS for the  
27 labels = {root  
28 frontier = [m  
29 while frontie  
30 cparent =  
31 for child  
32 if ch  
33 0  
34 label  
35 weigh  
36 label  
37 # mak  
38 for a  
39 1  
40 front

Thanks for completing that!

Correct!

CERRAR

Correct!

REINICIAR EL EXAMEN

EJECUTAR LA PRUEBA

ENVIAR RESPUESTA

Write your solution to work on weighted graphs. Note that the test given, all the edges have a weight of one - which isn't particularly interesting.

## Distance Oracle(II)

Clase 17:  
Final Assessment

1. Quiz: Bipartite

2. Quiz: Feel the Love

3. Quiz: Weighted Graph

4. Quiz: Finding the Best Flight

5. Quiz: Constantly Connected

6. Quiz: Distance Oracle (I)

7. Quiz: Distance Oracle (II)

8. Quiz: Finding a Favor

Distance Oracle (II)

1 #  
2 # This is the sam  
3 # only having to  
4 # create labels f  
5 #  
6 # In the shortest  
7 # interview, each  
8 # nodes in the ne  
9 # have the proper  
10 #  
11 # (1) for any pa  
12 # have at least  
13 #  
14 # (2) the shorte  
15 #  
16 # Given a graph G  
17 # create such lab  
18 # each label shou  
19 #  
20  
21 #code from https://github.com/WentaoZero/Intro-to-Algorithms  
22 #used by: Joacampora  
23 #  
24 # create\_labels takes in a tree and returns a dictionary, mapping each  
25 # node to its label  
26 #  
27 # a label is a dictionary mapping another node and the distance to  
28 # that node  
29 #  
30 def create\_labels(treeG, node):

Thanks for completing that!

Correct!

CERRAR



- Finding a Favor

Class 17:  
Final Assessment

📖

📁

🔍

✓ 1. Quiz: Bipartite

✓ 2. Quiz: Feel the Love

✓ 3. Quiz: Weighted Graph

✓ 4. Quiz: Finding the Best Flight


✓ 5. Quiz: Constantly Connected

✓ 6. Quiz: Distance Oracle (I)

✓ 7. Quiz: Distance Oracle (II)

✓ 8. Quiz: Finding a Favor

Finding a Favor



Thanks for completing that!

Correct!

CERRAR

```
6 #
7 # Write a function
8 # the probability
9 #
10
11 #
12 # Provided are two
13 # discussed in class
14 #
15 # You should manipl
16 # the given imple
17 # version (heap o
18 #
19 #code from https:
20 #used by: Joacamp
21
22 # code for heap o
23 from heap import
24 from operator imp
25 from collections import defaultdict
26 from math import log, exp
27
28 def reform_graph(G):
29     new_graph = defaultdict(dict)
30     for node in G:
31         for neighbor in G[node]:
32             new_graph[node][neighbor] = log(G[node][neighbor]) * -1
33     return new_graph
34
35 def maximize_probability_of_favor(G, v1, v2):
```

4) Tira de longitud  $n$  cubierta con fichas  $C_1$  y  $C_2$

a) Subestructura Óptima: El problema se puede solucionar solucionando el mismo problema con un  $n$  menor.

b) Ecuación Recursiva.

$$P_n = \begin{cases} \min(P_2, P_3); & \text{si } n \leq 2; \\ \min(2P_2, P_3); & \text{si } n = 3 \\ \min(P_i + P_{n-i}); & 1 \leq i \leq n-1; \text{ si } n > 3 \end{cases}$$

c) Programa en Python: \*

d) Tabla para  $C_2 = 5, C_3 = 7, n = 10$ .

$n$	1	2	3	4	5	6	7	8	9	10
Cubrir(5, 7, n)	5	5	7	10	12	14	17	19	21	24

5) Tablero  $3 \times n$  cubierto con fichas

c) Recurrencias:

$$A_n = D_{(n-1)} + C_{(n-1)}$$

$$C_n = A_{n-1}$$

$$D_n = D_{n-2} + 2 \cdot C_{n-1}$$

d)  $B_n = E_n = 0$  (No es posible que resulte una forma del tablero que representen)

e) Python \*

f)  $D_n$  para  $n = 10; 50; 100$

$$n = 10 \\ D_n = 203$$

$$n = 50 \\ D_n =$$

$$n = 100 \\ D_n =$$

$$n = 20 \\ D_n = 38651$$

\*Punto 4-c: By <https://es.scribd.com/document/358405261/Grafos-Complejidad-Computacional-Programacion-Dinamica>

```
1 def cubrir(C2, C3, n, r):
2     r[0] = 0
3     if n == 1 or n == 2:
4         q = min(C2, C3)
5     elif n == 3:
6         q = min(2 * C2, C3)
7     if i in r and (n - i) in r:
8         q = min(q, r[i] + r[n - i])
9     else:
10        q = min(q, cubrir(C2, C3, i, r) + cubrir(C2, C3, n - i, r)
11              )
12    r[n] = q
13    return q
```

\*Punto 5-e: By <https://es.scribd.com/document/358405261/Grafos-Complejidad-Computacional-Programacion-Dinamica>

```
1 def A(N):
2     if N == 0:
3         return 0
4     if N <= 1:
5         return 1
6     print "Haciendo A"
7     return D(N - 2) + C(N - 1)
8 def C(N):
9     if N == 0: return 0
10    if N <= 2: return 1
11    return A(N - 1)
12 def D(N):
13    if N == 0: return 0
14    if N <= 2: return 3
15    print "haciendo D"
16    return D(N - 2) + 2*A(N-1)
17
```