



**Tarea 4: Ciclos Hamiltonianos en grafos de distintas
cantidades de nodos utilizando C++**

Autor: John Serrano C.
Curso: Taller de Programación
Profesor: Pablo Román A.
Ayudante: Ricardo Hasbun M.

Tabla de contenidos

1. El problema	1
2. La solución	2
2.1. Clases de la solución	2
2.2. Explicación de la solución	2
3. Resultados	6
3.1. Pruebas y errores	6
3.2. Complejidad, eficiencia y heurísticas	7
4. Conclusiones	9
5. Bibliografía y Referencias	10
6. Anexos	11

1. El problema

Un grafo es un conjunto de nodos unidos, lo cual permite representar caminos o relaciones. En un grafo, donde se puede partir de un nodo inicial y recorrer todos los nodos, tal que solamente se recorra cada uno solamente una vez y luego se vuelva al nodo inicial, se denomina **Ciclo Hamiltoniano** al camino encontrado.

Por lo tanto, el objetivo principal es crear un algoritmo utilizando C++ y la Programación Orientada a Objetos para encontrar un Ciclo Hamiltoniano en un grafo con una variable cantidad de nodos. Para ello, el grafo se crea con una probabilidad P de que dos nodos tengan conexión entre si y cada vez que se descarta que un grafo tenga un Ciclo Hamiltoniano, se vuelve a crear otro con la probabilidad aumentada, hasta que se encuentre un grafo con un Ciclo Hamiltoniano.

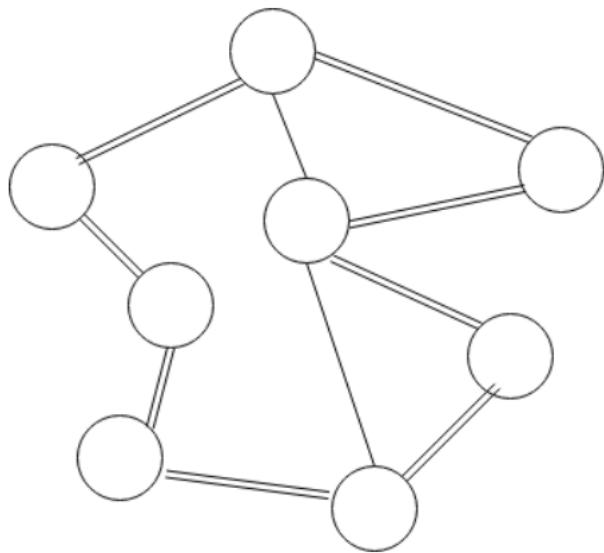


Figura 1: Un grafo que tiene un Ciclo Hamiltoniano.

2. La solución

2.1. Clases de la solución

Para resolver el problema planteado, se necesitan las siguientes clases:

- **Clase Node:** La clase “Node” representa a un nodo del grafo. Cada nodo tiene un numero asignado y un “valor” que corresponde a la cantidad de conexiones que tiene.
- **Clase Heap:** La clase “Heap” corresponde a la clase de la Cola de prioridad, la cual se utiliza como “Open list” para así guardar los nodos del grafo. Heap recibe un tamaño para poder funcionar.
- **Clase Hamiltonian:** “Hamiltonian” es la clase mas importante en esta tarea. En ella se crea el grafo a buscar y se tienen los métodos de solve para así buscar si el grafo generado tiene o no un Ciclo Hamiltoniano.

2.2. Explicación de la solución

Primero, se debe definir una cantidad de nodos que tendrá el grafo. El enunciado menciona principalmente, 15 nodos. Se utiliza el método **Hamiltonian** para crear una instancia de la clase Hamiltonian y se le pasa la cantidad de nodos. Internamente, Hamiltonian creará un vector de vectores de enteros inicializados en 0, el cual se convertirá en el grafo y un vector de enteros, el cual será el camino del Ciclo Hamiltoniano encontrado. Luego, con el método **createGraph** se crea el grafo, utilizando la probabilidad P. La probabilidad inicialmente parte con 0,03 y va aumentando a medida que se van descartando los grafos. La probabilidad P permite tener aleatoriedad a la hora de generar el grafo. Con el grafo ya generado, se dispone del método **printGraph** para poder ver si es que el grafo se generó correctamente.

Lo siguiente es aplicar el método **solve**. Este método, inmediatamente llama a otro método llamado **verify**, el cual se encarga de verificar que todos los nodos del grafo tengan un grado mayor o igual a 2. El grado de un nodo, nos permite saber cuantas conexiones un nodo tiene con otros nodos. Si un nodo tiene un grado menor a 2, eso implica inmediatamente

que el grafo no tiene un Ciclo Hamiltoniano, por lo que se retorna **false**. Si este es el caso, entonces se muestra un mensaje en color rojo de que no se encontró un Ciclo Hamiltoniano. En el caso contrario, es decir, todos los nodos tienen un grado mayor o igual a 2, se procede con el método **solveMain**, el cual es la segunda parte del método de la solución.

solveMain comienza creando dos Heaps, utilizando **Heap**. A ambas Heaps se les pasa la cantidad de nodos del grafo, la primera Heap es donde se guardarán los nodos del grafo y la segunda es una Heap temporal, donde se guardaran aquellos nodos descartados de tener una conexión con otro nodo. Luego, se tiene un ciclo for, donde se utiliza el método **getValue** para obtener el grado de todos los nodos. Aquel nodo con mayor grado es el nodo inicial del camino, por lo que se crea un nodo utilizando **Node**, cuyo número es el número del nodo que tiene el mayor grado y el valor es el grado encontrado. Luego se crean nodos para todos los otros nodos (a excepción por el inicial, ya que este ya fue incluido en el camino) y se agregan a la Heap principal. Mientras la Heap principal no este vacía, se saca un nodo de la Heap, el cual será el que tiene menor grado y se comprueba si es que el número de este nodo no se encuentra en el camino y si el nodo tiene una conexión con el nodo anteriormente agregado al camino. Si no es así, el nodo se guarda temporalmente en la segunda Heap. Esto es, debido a que puede que hayan otros nodos que si tengan conexión con ese Nodo que se sacó de la Heap y es importante no perderlo.

En el caso contrario, el número del nodo se agrega al camino y se aumenta el número del camino, luego se vuelven a agregar todos los nodos guardados en la Heap temporal a la Heap principal y se utiliza el destructor para limpiar la memoria de la Heap temporal. Se vuelve a crear otra y el ciclo se repite, hasta que la Heap principal quede vacía.

Una vez que la Heap principal queda vacía, se destruyen ambas Heaps y se comprueba si el número del camino es el mismo del tamaño de los nodos. Si es así, entonces se comprueba si es que el último nodo guardado en el camino tiene conexión con el primer nodo guardado. Si el resultado es **true**, entonces hemos encontrado el ciclo Hamiltoniano. Si el resultado es **false**, entonces el camino encontrado no es un Ciclo Hamiltoniano. Si el número del camino no es el mismo que la cantidad de nodos del grafo, o si sucedió algo que impedía que se siguiera generando el camino, se retorna **false**.

Si el resultado de **solveMain** fue **false**, entonces se muestra en rojo un mensaje

que dice que no se encontró un Ciclo Hamiltoniano. En el caso contrario, se llama al método **printSolution**, el cual muestra en color verde un mensaje que dice que se encontró el Ciclo Hamiltoniano y se procede a mostrar el camino encontrado.

```
Buscando con p = 0.166667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.2
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.233333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.266667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.3
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.333333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.366667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.4
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.433333
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 10 9 12 5 0 8 2 6 7 4 1 3 14 13 11 10
Iteracion completada en: 0.00152[s]

Tiempo promedio: 0.00152[s]
El promedio de la probabilidad es: 0.433333
```

Figura 2: Un ejemplo de como el programa muestra la solución.

La clase Main del programa permite ir repitiendo el proceso con distintas iteraciones. Notar que, cada iteración comienza con probabilidad 0,03 y continua hasta que se encuentre un Ciclo Hamiltoniano con una probabilidad específica. Se recomienda probar con 50 iteraciones.

La solución fue basada en el método de resolución de Backtracking que se puede encontrar en la página GeeksForGeeks [1], pero también se utilizaron muchas cosas de la Tarea 1 y 2 del curso, como por ejemplo, la idea de usar Heaps.

A continuación, se muestran algunos de los bocetos utilizados para idear la solución:

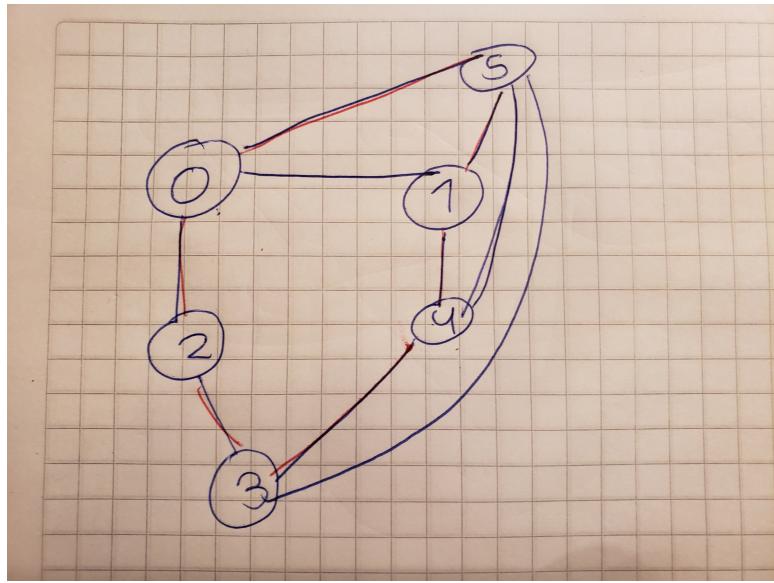


Figura 3: Boceto que muestra la idea de que solo algunos caminos son Ciclos Hamiltonianos.

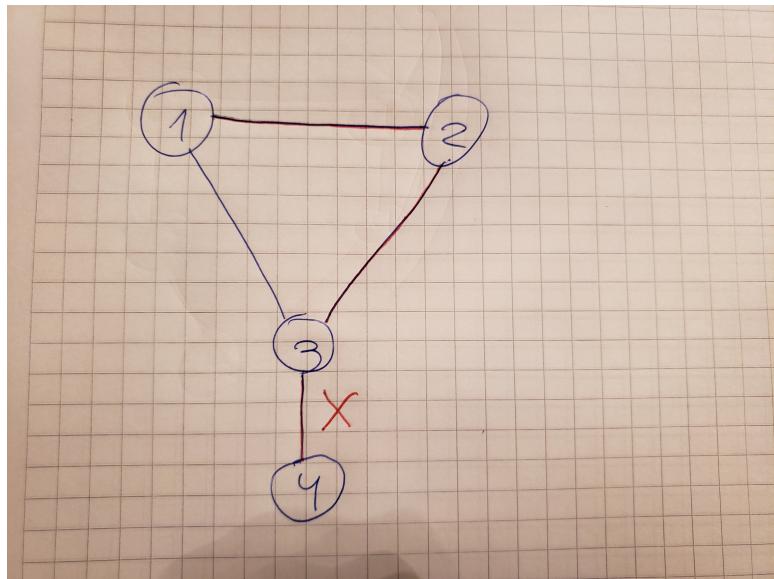


Figura 4: Boceto que muestra la idea de que es necesario que todos los nodos tengan grado mayor o igual a 2.

3. Resultados

3.1. Pruebas y errores

Inicialmente, se comenzó solo adaptando la solución de GeeksForGeeks. Con 15 nodos, no tomaba mucho tiempo completar 50 iteraciones, pero con 300 nodos era imposible. Fue ahí cuando se comenzaron a aplicar elementos e ideas de las tareas 1 y 2, principalmente el uso de las clases Node y Heap. Hubieron múltiples pruebas y varios errores, algunos de ellos fueron que debido a como funciona el programa, el grafo generado podía ser el mismo en cada iteración, por lo que se utilizó `rand(time(NULL))` en el método de `createGraph` para aplicar aleatoriedad y `usleep()` en main para así forzar a que el grafo generado fuera distinto en cada iteración. Un problema que lamentablemente no se pudo solucionar del todo, es que a veces, por temas de la Heap principal, esta se quedaba vacía y por lo tanto, el programa no encontraba el Ciclo Hamiltoniano cuando si lo había, lo cual también forzó a que la probabilidad promedio subiera un poco.

Aún así, el programa se logró optimizar bastante, de tal forma que ahora es posible realizar 50 iteraciones con 300 nodos, obteniendo un tiempo promedio entre 43 segundos y 60 segundos.

```
Buscando con p = 0.3
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.333333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.366667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.4
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.433333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.466667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.5
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 213 68 46 295 233 247 269 253 285 156 161 67 298 291 266 205 77 88 6 4 251 126 234 31 230 231 62 286 248 280 65 272 260 223 5 85 29 71 201 136 274 241 267 159 25 210 107 250 140 137 212 174 118 78 49 258 150 256 192 7 147 157 183 198 58 185 63 81 50 83 282 33 268 11 148 9 16 217 208 196 167 246 75 281 15 143 189 170 57 284 119 179 257 17 204 180 72 244 235 164 44 252 203 56 207 160 270 74 151 273 23 103 261 169 176 70 92 277 134 263 129 37 84 101 184 16 8 199 240 155 61 128 262 97 238 28 13 113 112 110 127 254 237 98 187 91 138 69 89 10 297 243 24 197 55 175 34 38 121 18 229 0 177 26 255 146 152 111 188 52 116 1 78 64 169 265 294 2 195 36 90 219 227 115 123 8 158 200 132 105 135 190 1 95 99 259 138 182 299 76 125 279 153 232 22 133 139 288 193 122 215 186 42 209 100 220 211 264 249 86 141 20 163 287 292 117 3 191 165 45 216 225 39 222 218 236 275 120 278 35 221 192 142 166 66 224 96 40 14 59 145 94 228 104 19 171 214 21 154 172 80 296 93 144 242 54 12 82 131 289 53 286 73 181 290 149 79 162 202 173 276 226 30 43 41 108 106 124 239 283 27 87 271 293 114 245 51 48 194 47 60 32 213
Solved in: 1.03642[s]

Tiempo total: 43.8279[s]
Tiempo promedio: 0.876558[s]
El promedio de la probabilidad es: 0.43
```

Figura 5: Tiempo promedio de 0.87 segundos al resolver 50 iteraciones de 300 nodos.

Para el programa principal, 50 iteraciones de 15 nodos, da un tiempo promedio usualmente menor 0.1 segundo, por lo que el programa es bastante eficiente y se optimizó de tal forma que el resultado principal se puede obtener muy rápido.

```

Iteracion Numero 50
Buscando con p = 0.0333333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.0666667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.1
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.133333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.166667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.2
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.233333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.266667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.3
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 12 8 11 13 9 0 1 5 14 4 3 7 6 10 2 12
Iteracion completada en: 0.000839[s]

Tiempo promedio: 0.0011795[s]
El promedio de la probabilidad es: 0.370667

```

Figura 6: Tiempo promedio de 0.0011 segundos al resolver 50 iteraciones de 15 nodos.

3.2. Complejidad, eficiencia y heurísticas

Como se mostró anteriormente, se tiene un algoritmo bastante eficiente y rápido, capaz de encontrar resultados incluso cuando la cantidad de nodos es muy grande. Aún así, la complejidad ronda cercano a N , ya que se tienen algunos ciclos for, pero debido a que se utilizaron vectores no fue necesario concatenar dobles ciclos for. El uso de vectores permitió optimizar de gran forma el programa en general, ya que algunas funciones como **count** permiten obtener la cantidad de veces que se repite un elemento sin la necesidad de iterar en el vector.

Algunas de las heurísticas utilizadas, fueron comenzar con el nodo de mayor grado

y luego seguir con el de menor grado, algo que ayudo a reducir la probabilidad promedio desde 0,6 a 0,35 aproximadamente, con 15 nodos. Otra heurística fue ir descartando grafos inmediatamente si es que alguno de los nodos no tenia un grado mayor o igual a 2, ya que eso implicaba que solo tenia camino hacia un lado o bien, era imposible llegar al nodo (En otras palabras, se tenia un nodo aislado).

Además de los Tests para cada clase, se dispone de un **Test de Eficiencia** el cual permite comprobar que el programa si es rápido cuando se tienen 300 iteraciones de 50 nodos. También es posible encontrar un Ciclo Hamiltoniano en un grafo de **1000 nodos**, lo cual se demora menos de 1 minuto, demostrando aún mas que la solución diseñada si es eficiente.

```
Buscando con p = 0.5666667
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 217 243 315 370 638 808 384 716 451 746 786 194 683 485 131 412 795 513 245 398 529 103 890 753 857 396 586 733 705 253 583 713 662 555
47 206 988 969 402 304 984 274 585 834 410 16 367 310 728 237 851 503 916 872 804 474 996 268 480 584 644 704 521 383 156 894 551 358 432 572 9 473 459 510 661
318 861 516 442 581 435 615 536 99 962 160 695 892 592 621 403 594 970 742 571 540 726 265 908 381 845 787 233 84 666 604 299 285 278 542 269 129 44 699 96 949 6
88 958 282 902 855 21 351 809 686 598 246 3 909 251 889 869 898 424 841 821 407 797 871 917 380 771 707 1 313 522 18 968 316 290 784 438 216 168 8 957 605 940 91
4 982 458 118 768 993 29 95 782 755 734 353 730 83 653 593 563 371 863 643 928 848 500 357 883 350 328 537 850 722 910 619 216 146 448 819 875 966 469 997 387 38
5 187 807 349 599 144 799 72 140 574 677 823 525 559 519 931 973 423 955 100 743 696 325 633 884 778 34 580 102 622 261 244 24 109 826 766 374 954 460 222 858 84
3 45 356 352 344 347 4 330 214 161 142 260 548 832 98 366 119 164 5 878 431 186 624 533 209 277 634 985 306 219 389 430 46 215 951 926 125 360 717 239 428 700 11
2 337 303 591 19 107 550 270 512 287 735 885 141 830 948 987 439 228 691 404 929 881 663 947 35 281 31 271 331 738 991 561 737 27 642 224 229 888 220 769 873 212
418 193 924 674 496 42 971 693 864 800 327 979 626 603 745 814 172 606 649 272 80 26 32 70 538 238 106 986 939 956 789 184 766 348 669 378 933 444 394 759 319 6
32 77 39 652 252 507 655 601 297 437 23 489 240 801 942 499 796 770 672 646 300 802 329 648 248 294 678 346 588 293 577 727 515 556 775 342 199 497 249 880 779 1
76 868 420 436 208 391 718 339 744 754 564 999 440 838 189 833 637 944 920 544 255 130 256 922 230 919 57 854 831 813 720 174 345 687 373 930 750 38 69 684 912 1
11 155 284 133 749 703 650 295 116 765 776 798 595 434 498 390 773 25 710 692 157 599 613 965 75 200 657 58 600 530 192 514 524 67 376 322 258 994 115 232 12 783
405 361 715 443 882 22 63 65 30 354 501 504 865 464 475 247 234 874 905 52 60 927 445 377 725 723 2 708 698 676 26 887 714 171 667 101 138 263 241 53 132 55 625
11 462 491 897 414 862 372 433 989 427 411 761 43 368 413 670 165 159 332 636 152 484 17 627 15 774 92 582 264 169 527 468 816 476 205 877 731 682 824 89 781 97
736 658 805 395 562 336 923 167 925 158 576 68 173 618 175 966 495 842 324 312 483 227 990 283 86 721 697 803 639 308 602 139 541 137 983 680 946 835 117 867 79
0 364 549 545 659 453 120 415 932 13 628 918 340 553 124 463 7 426 454 911 62 147 363 259 953 135 81 936 502 94 41 225 886 338 856 153 288 539 518 108 587 732 65
1 467 87 762 355 335 181 397 250 764 417 93 597 767 305 896 73 557 554 56 945 565 182 213 679 836 719 334 85 640 88 839 870 558 876 341 51 645 976 635 226 481 98
3 276 526 450 369 963 543 952 441 846 647 810 362 709 656 616 321 74 114 471 950 975 668 479 499 569 136 547 406 28 532 665 629 752 505 685 724 935 40 612 610 16
3 307 560 375 128 54 630 33 879 937 211 488 681 811 184 793 121 79 311 578 326 408 757 470 853 323 400 739 620 477 552 447 623 158 915 297 791 49 517 18 694 422
198 995 609 531 273 143 567 747 528 202 196 641 421 827 242 866 818 456 913 236 977 794 573 579 690 82 611 943 506 596 286 388 980 568 849 180 6 847 822 959 401
162 59 91 110 535 429 154 614 446 37 631 361 941 183 386 289 899 14 382 392 170 90 886 191 78 785 399 365 478 185 534 777 758 393 472 254 891 76 523 492 763 998
511 66 711 921 961 425 359 756 279 660 201 223 817 221 675 449 964 379 702 895 689 788 967 608 812 893 71 981 280 36 828 792 123 546 452 64 457 292 566 729 974 8
29 575 275 302 820 179 701 570 751 465 127 151 992 262 320 654 291 487 298 494 309 145 148 461 197 617 263 257 113 482 50 48 748 590 333 901 134 741 607 493 904
195 589 455 845 456 520 190 416 204 740 267 978 343 188 815 231 149 218 177 967 466 317 712 122 852 837 844 859 490 760 900 934 235 972 178 673 938 486 960 166 266 4
19 314 126 664 105 825 671 61 860 568 8 780 772 296 217
Iteracion completada en: 43.2201[s]
Tiempo promedio: 43.2201[s]
El promedio de la probabilidad es: 0.5666667
```

Figura 7: Un Ciclo Hamiltoniano de un grafo de 1000 nodos encontrado en menos de 1 minuto.

4. Conclusiones

Finalmente, a pesar de las complicaciones con la eficiencia, el tiempo y las probabilidades, se logró completar el objetivo principal que era diseñar un algoritmo tal que pueda encontrar un Ciclo Hamiltoniano en un grafo de una cantidad específica de nodos. Además, el algoritmo diseñado es eficiente, lo cual permite probar la solución con muchos casos de prueba y comparar distintos resultados.

Al ser esta la tarea final de la asignatura, se puede decir que se aprendió mucho a lo largo de todas las tareas, utilizando C++ y la Programación Orientada a Objetos y se espera que los conocimientos aprendidos en esta tarea y en la asignatura en general, sirvan en un futuro para otros desafíos.

5. Bibliografía y Referencias

- [1] GeeksForGeeks (2022). “Hamiltonian Cycle — Backtracking-6”. Artículo Online. Recuperado de: <https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/>
- [2] GeeksForGeeks (2019). “Visitor Design Pattern”. Artículo Online. Recuperado de: <https://www.geeksforgeeks.org/visitor-design-pattern/>

6. Anexos

```
Buscando con p = 0.166667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.2
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.233333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.266667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.3
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.333333
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.366667
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.4
No se encontro un ciclo Hamiltoniano.

Buscando con p = 0.433333
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 14 5 2 3 6 1 13 11 12 0 9 8 4 10 7 14
Iteracion completada en: 0.001593[s]

Tiempo promedio: 0.00108623[s]
El promedio de la probabilidad es: 0.356667
```

Figura 8: 100 iteraciones de grafos de 15 nodos resueltas en tiempo promedio 0,0010.

```
Numero de nodos del grafo: 15
Buscando con p = 0.43
0 0 0 1 1 0 0 1 0 0 1 1 0 0 1
0 0 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 0 0 1 1 0 0 1 0 0 1 0 1
1 0 0 0 1 0 1 0 0 0 0 0 0 1 1
1 1 0 1 0 0 0 0 1 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0 0 0 1 0 1
0 0 1 1 0 0 0 0 1 1 0 1 0 1 0
1 0 0 0 0 1 0 0 1 0 1 0 1 0 1
0 1 0 0 1 0 1 1 0 0 0 0 0 1 1
0 0 1 0 0 0 1 0 0 0 1 1 1 0 0
1 0 0 0 0 0 0 1 0 1 0 0 0 1 0
1 1 0 0 0 0 1 0 0 1 0 0 0 0 1
0 0 1 0 0 1 0 1 0 1 0 0 0 1 0
0 1 0 1 1 0 1 0 1 0 1 0 1 0 1
1 0 1 1 0 1 0 1 1 0 0 1 0 1 0
Se encontro un ciclo Hamiltoniano!
El ciclo Hamiltoniano es: 13 1 11 9 10 0 4 3 6 2 5 12 7 8 14 13
```

Figura 9: 1 iteración de un grafo de 15 nodos, imprimiendo el grafo y resuelta en menos de 0.1 segundos.