

Assembling Bitcoin Solution

Group 7:

Raymond Feng
Hao-Chun Hsiao
Sophia Shi
Keer Feng
Samuel Lin

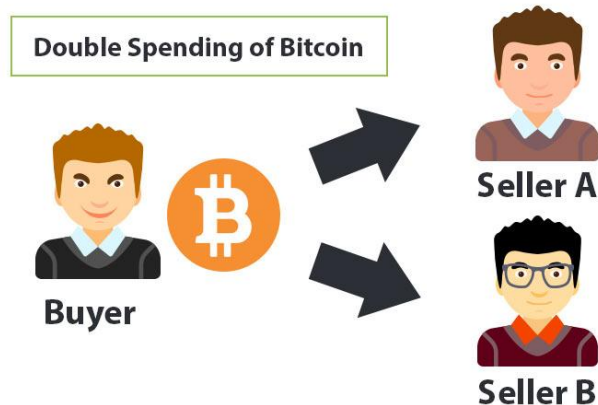
- Bitcoin and Independent Set Decision Problem
- NP-complete
- NP-hard
- Co-NP
- Greedy Algorithm
- Special Case with Tree Search
- Integer Linear Programming
- Bellman-Ford Algorithm
- Our Solution

Bitcoin and Independent Set Decision Problem

Bitcoin and Independent Set Decision Problem

Brief review of the Bitcoin situation:

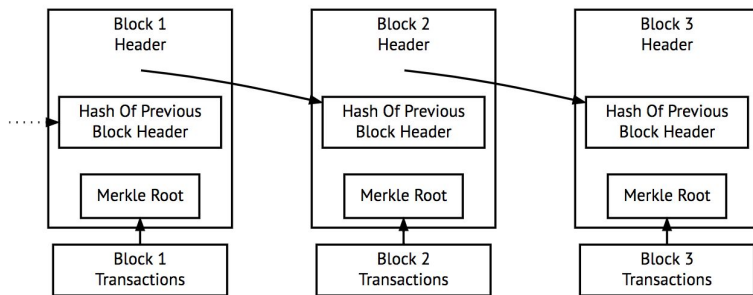
- The **double-spending problem**
- Bitcoin's solution
- Assembling a Bitcoin block
- Our Assumption
- Independent Set Decision Problem



Assembling a Bitcoin Block

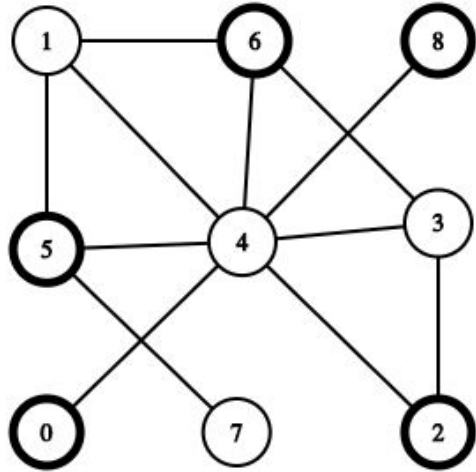
Miners provide a solution by pooling a block of transaction and a mathematical lottery is held to find the correct hash value. With the correct hash value the block will be added to the end of the chain, making sure that all transactions are in the right order.

1. they will go into a pool of pending transactions
2. sort transactions in the right order and try to lock on the “block chain”
3. To select which block is next, a kind of mathematical lottery is held.
4. The first miner to solve the problem will be rewarded with bitcoins



Simplified Bitcoin Block Chain

Independent Set Decision Problem



Given an undirected graph G

Is there a subgraph of k vertices, with no edges between any two vertices? (The answer is either “Yes” or “No”)

An the Independent set in this graph is

$\{0, 2, 5, 6, 8\}$

Our Assumption

According to Bitcoin, each block is around 1MB. Each transaction size will affect the amount of transaction that can be put into a block.

In our project, we will assume that all transaction will be equal. Thus, the number of transaction will be constant. By simply dividing $1\text{MB} / \text{size}(\text{transaction})$, we can get the number of transactions in a block.

In the case of Independent Set Decision Problem, the number of transactions in a block will be the size K of Independent set (subgraph) that we are trying to determine if it can be found in the graph.



NP-completeness

NP-completeness

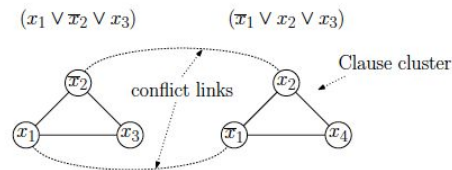
1) Independent Set Decision Problem is in NP:

Given a set we can easily check that there are no edges between any two vertices.

2) Independent Set Decision Problem is NP-Hard:

A polynomial time reduction exist when we reduce 3-SAT to Independent Set Decision Problem.

We can formulate a graph from a 3-SAT boolean formula by creating clusters for each clause then we create edges between all vertices associated with x to all vertices associated with $\sim x$.



Clause clusters for the clauses $(x_1 \vee x_2 \vee x_3)$ and $(x_1 \vee x_2 \vee x_4)$.



NP-hard

NP-Hard Version

Reminder ** NP-Hard Definition: “A decision problem H is NP-hard when for every problem L in NP, there is a polynomial time reduction from L to H .” at least as hard as the hardest problems in NP.

A Maximum Independent Set Problem (MIS) is an Independent set of the largest possible size for a given graph and this is NP-Hard.

Python and the packages NetworkX and matplotlib allows us to approximation of Maximum Independent Problem.

According to the documentation, `maximum_independent_set()` returns a maximum independent set for the given undirected Graph.

The approximation uses the framework of “subgraph-excluding” algorithm and the time complexity is $O(|n|/(\log|n|)^2)$.



Python

```
import networkx
from networkx.algorithms.approximation import independent_set
import matplotlib.pyplot as plt
```

```
# setting the width of the display
width= 8
# setting the height of the display
height= 10
#Dots per inch
DPI = 70
# configuration
plt.figure(figsize=(width, height), dpi=DPI)
```



Configuration of the output

```
#creates an empty graph
undirectedGraph = networkx.Graph()
```

```
#add vertices(nodes) to the graph
undirectedGraph.add_node(1)
undirectedGraph.add_node(2)
undirectedGraph.add_node(3)
undirectedGraph.add_node(4)
undirectedGraph.add_node(5)
undirectedGraph.add_node(6)
undirectedGraph.add_node(7)
undirectedGraph.add_node(8)
undirectedGraph.add_node(9)
undirectedGraph.add_node(10)
```



Set up nodes

```
#connect the vertex(nodes)
undirectedGraph.add_edges_from([(1, 4), (1, 2), (1,8) , (2,3) , (2,7) , (2,5 ),(7,6) , (6,4) , (4 ,5) , (8,9) , (8,10) , (7 , 10)])
labels = [ 1, 2, 3, 4, 5, 6 , 7 , 8 , 9 ,10]
```

```
#draw out the graph
networkx.draw_networkx(undirectedGraph, with_labels=labels)
plt.axis ("off")
plt.show()
```



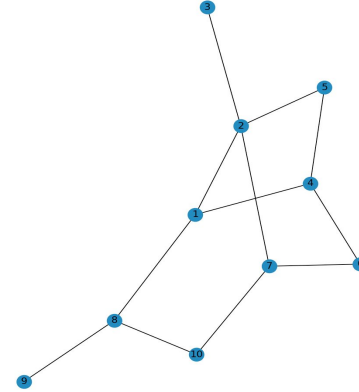
Set up graph

```
#Get the maximum independent set in the graph
set = independent_set.maximum_independent_set(undirectedGraph)
print(len(set))
print("This is the maximum independent set for the given undirectedGraph :")
print(set);
```



Approximate maximum independent set

Output



```
(base) dhcp-206-87-155-235:Independent set problem samuellin$ python3 MaxIndependentSet.py
6
```

```
This is the maximum independent set for the given undirectedGraph :
{1, 3, 5, 6, 9, 10}
```

Co-NP

Co-NP

NP: Is there an Independent Set of size k in $G(V,E)$

find Independent Set in $|G(V,E)| \geq k$

Co-NP: Is there not an independent set of size k in $G(V,E)$

All independent set $|G(V,E)| < K$ in the given graph

If there is an Independent Set of size k in $G(V,E)$ in NP the output will be “Yes”. On the other hand, the output of the Co-NP will be “No”. It is easy to check if the answers are correct or not.

For example, if we let $k = 3$ and there exist an Independent set of size k in NP. The question for Co-NP will produce “No”. It is saying that , “No, there is an independent set of size k ”. We can check by finding an Independent set of size 3 in the graph



Greedy Algorithm

Greedy Algorithm

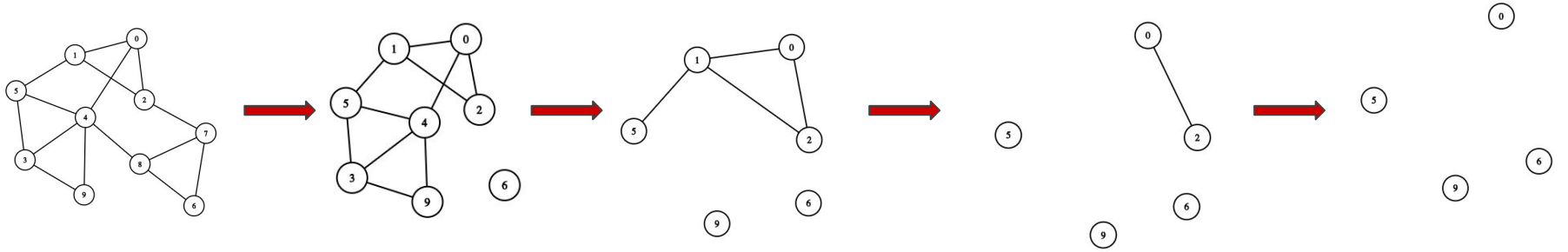
As we proved before, the maximum independent set is a NP-Hard problem, which generally indicates a set of unconnected two vertices with no edges in the conflict graph.

What happens when we run the Greedy algorithm?

when we try to run the Greedy algorithm, at each step, we choose a vertex with the minimum neighbours, select the vertex and discard all its neighbours, repeat the process until there is no connected vertex remaining and in the end we will get the independent set.



Greedy Algorithm Visualization



Graph1 : The undirected Graph with 9 vertices

Graph2: (** Select the vertex with least degree and remove the connecting vertices) Selected 6 and remove vertex 8 and 7

Graph3: Select vertex 9 and remove vertex 3 and 4

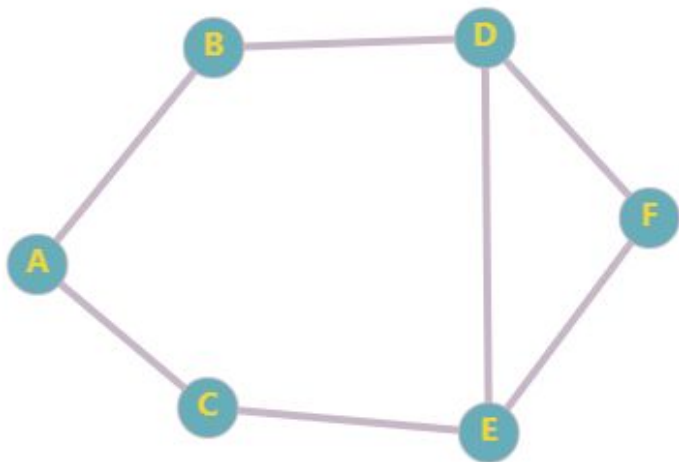
Graph4: Select vertex 5 and remove vertex 1

Graph5: Select vertex 0 and remove vertex 2

Greedy Algorithm

We can draw the random graph to suspect some counter-examples, because MIS is NP-hard while greedy algorithm running in the polynomial-time.

The following graph is the smallest graph that the Greedy algorithm can fail, showing 7 vertices and any sequences that lead Greedy to a non-optimal solution:

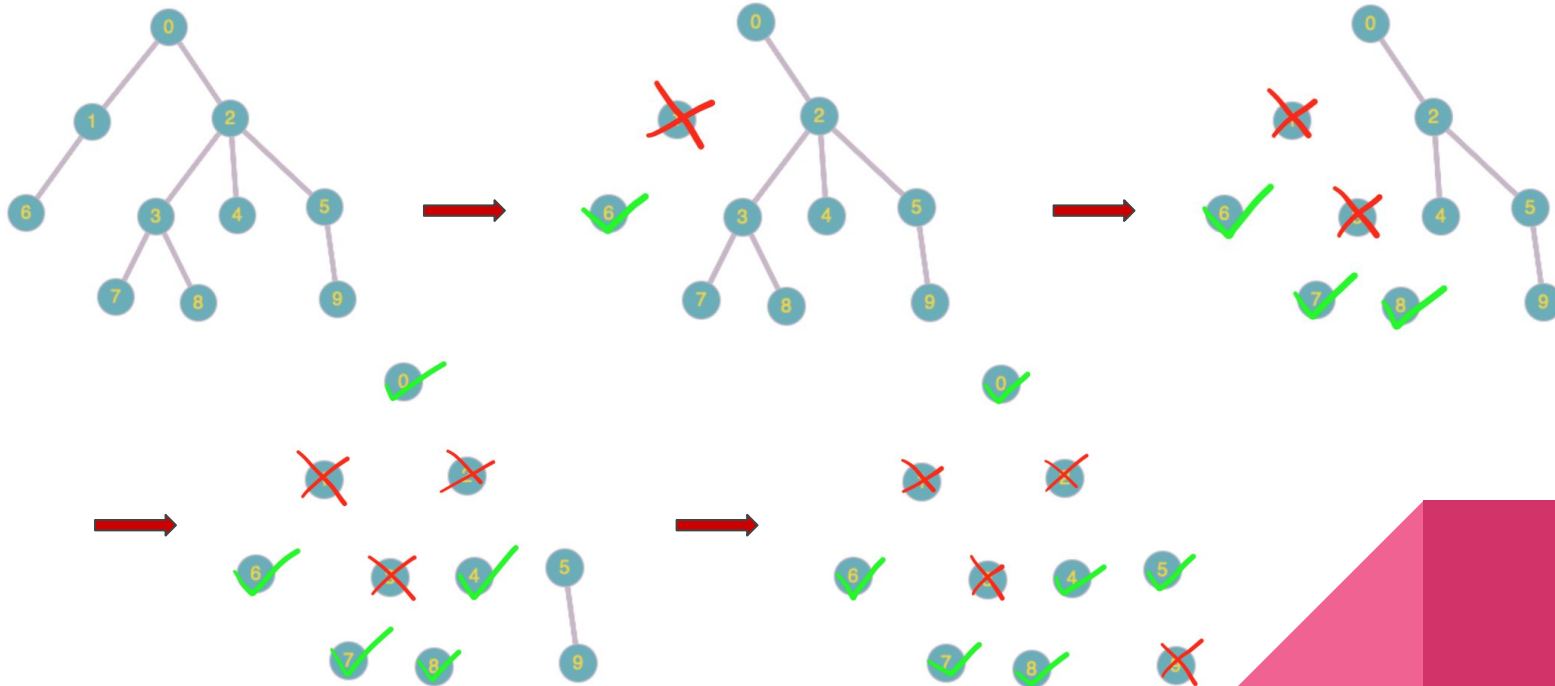


Ex: the smallest graph that Greedy can fail. Picking nodes from A leads to a solution of the size 2 while F gives a solution of size 3.

Special Case with Tree Search

Special Case: Tree Searching

Definition: A tree graph is an undirected graph in which any two vertices are connected by exactly one path



|Integer Linear Programming

Integer Linear Programming

Maximize $x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8$

$$x_0 + x_4 \leq 1 \quad x_1 + x_4 \leq 1$$

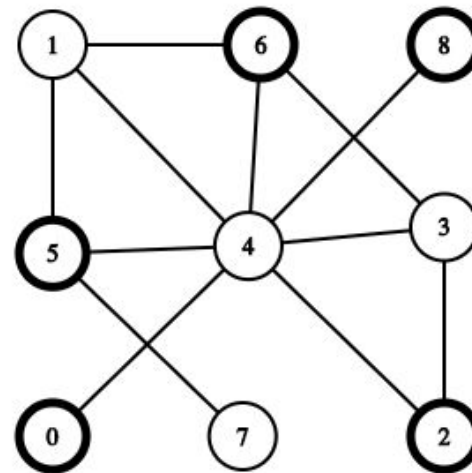
$$x_1 + x_5 \leq 1 \quad x_1 + x_6 \leq 1$$

$$x_2 + x_3 \leq 1 \quad x_2 + x_4 \leq 1$$

$$x_3 + x_4 \leq 1 \quad x_3 + x_6 \leq 1$$

$$x_4 + x_5 \leq 1 \quad x_4 + x_6 \leq 1$$

$$x_4 + x_8 \leq 1 \quad x_5 + x_7 \leq 1$$



$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in \{0, 1\}$ [0,1]

$$x_0 = 1 \quad x_1 = 0 \quad x_2 = 1 \quad x_3 = 0 \quad x_4 = 0 \quad x_5 = 0 \quad x_6 = 1 \quad x_7 = 1 \quad x_8 = 1$$

Input

```
1 from gurobipy import *
2
3
4 model = Model('Peter')
5
6 xk = model.addVars(9, vtype=GRB.BINARY, name='x')
7
8 model.addConstr(xk[0]+xk[4]<=1)
9 model.addConstr(xk[1]+xk[5]<=1)
10 model.addConstr(xk[2]+xk[3]<=1)
11 model.addConstr(xk[3]+xk[4]<=1)
12 model.addConstr(xk[4]+xk[5]<=1)
13 model.addConstr(xk[4]+xk[8]<=1)
14 model.addConstr(xk[1]+xk[4]<=1)
15 model.addConstr(xk[1]+xk[6]<=1)
16 model.addConstr(xk[2]+xk[4]<=1)
17 model.addConstr(xk[3]+xk[6]<=1)
18 model.addConstr(xk[4]+xk[6]<=1)
19 model.addConstr(xk[5]+xk[7]<=1)
20
21 model.setObjective(quicksum(xk), GRB.MAXIMIZE)
22
23 model.optimize()
24
25 # this can output the desired .lp file.
26 model.write('MIS_Raymond_ex.lp')
27
28 print('')
29 print('Solution:')
30
31 # Retrieve optimization result
32
33 solution = model.getAttr('X', xk)
34
```

Set up the model with 9 variables

Construct constraints on the nodes

Set up the objective function

Find solution

Output solution

Output

Optimize a model with 12 rows, 9 columns and 24 nonzeros

Variable types: 0 continuous, 9 integer (9 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+00]

Objective range [1e+00, 1e+00]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+00]

**Solution:

{ 0 , 1 , 2 , 7 , 8 } is the Independent set

Found heuristic solution: objective 5.0000000

Presolve removed 12 rows and 9 columns

Presolve time: 0.01s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.01 seconds

Thread count was 1 (of 4 available processors)

Solution count 1: 5

Optimal solution found (tolerance 1.00e-04)

Best objective 5.0000000000000e+00, best bound 5.0000000000000e+00, gap 0.0000%

Solution:

{0: 1.0, 1: 1.0, 2: 1.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 1.0, 8: 1.0}

LP Rounding

In case we did not get an integer solution, we round up all $x_i > 0.5$ to 1 and the rest to 0. Because for each edge $\{x_i, x_j\}$, x_i and x_j can not be both greater than 0.5, they will not be both round up to 1, hence we now have a solution to our independent set problem. Since linear programming is in P and this rounding algorithm can be done in $O(n)$ time, the solution we obtained is a polynomial time approximation.



Bellman-Ford Algorithm

Bellman-Ford Algorithm

“The goal of the algorithm is to minimize the number of excluded vertices for each vertex added to the path. The algorithm works in multiple rounds. To keep track of all paths, the algorithm maintains the lists *Exclude* , *Cost* , and *Previous* for each round. BMA algorithm can run up to $|V| - 1$ rounds. Each round is a new attempt to find a higher independence number. It will stop will stop at the round where no more vertices can be added to any path(independent set)”

Mostafa H. D.(2014, May 31). Maximum Independent Set Approximation Based on Bellman-Ford Algorithm

At round 0 the Exclude of a vertex is the set of vertices are connected to the vertex

The cost is the cardinality of each vertex's Exclude

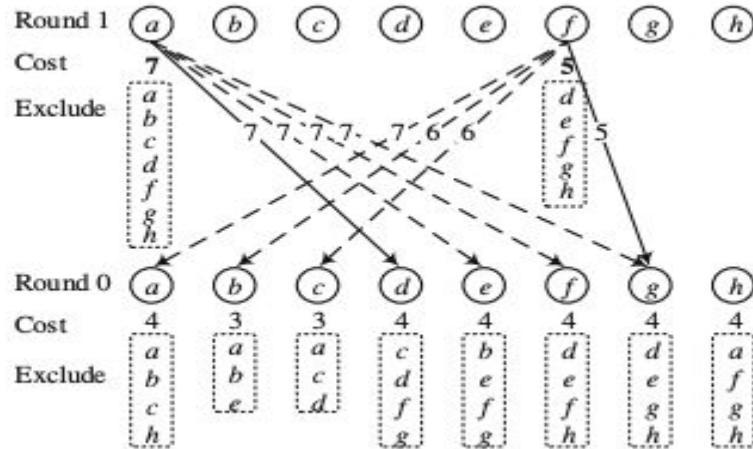
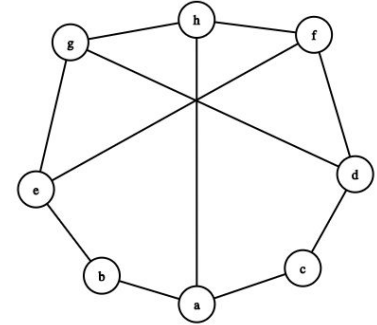
Previous contains the Excludes of all vertices in the previous round.



Bellman-Ford Algorithm

To explain we will give an example,

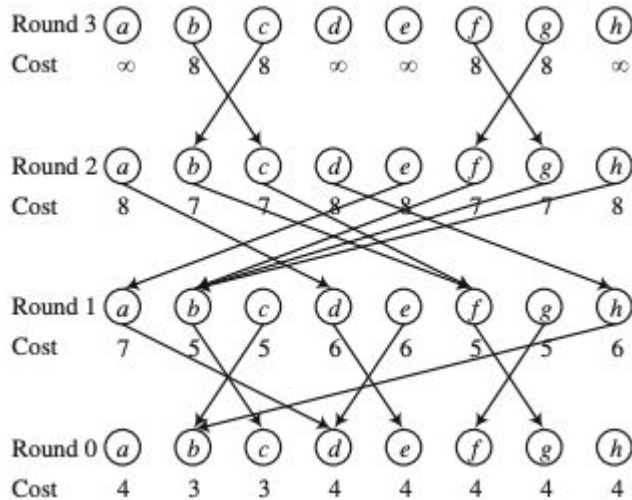
Given the graph on the right we can set up each round like this:



For each vertex we look at the vertices that do not have an vertices between them. We then look at the union of their Excludes and we choose the one with the lowest cost to become the next round of Exclude. We do this for all vertices.

Bellman-Ford Algorithm

This is a completion of all rounds in the graph



When a vertex's Exclude contains all the vertices in the graph we do not continue. For example, for vertex (a) the cost is 8 which means that all vertices from the graph is included.

To find a solution we look at the last vertices to reach full capacity. These vertices are {b, c, f, g}. For each of these vertices we can trace back down the arrows to Round 0 and the Independent set is {b, c, f, g}

The time complexity for this algorithm is:
 $O(n(n^2 - m))$

Our Solution

Our Solution

Instead of going through all the possible combinations of size k we can **reduce it**

- 1) Each row to represent a node
- 2) For each row, we will remove the nodes that has an edge between the selected node

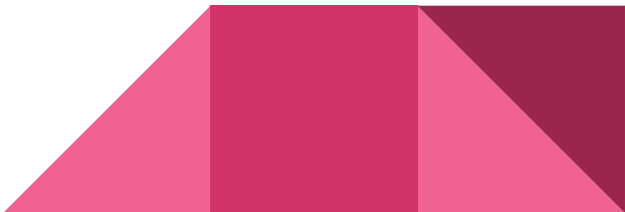
** For example , take node 0 as an example, we see that it has edges between vertex 1 and 2, we can remove those two nodes

leaving us with the set of nodes $[0, 3, 4, 5, 6, 7, 8]$

- 3) We do this for each row. We will get :

Vertex 0: $[0, 3, 4, 5, 6, 7, 8]$	Vertex 1: $[1, 2, 4, 6]$	Vertex 2: $[2, 3, 4, 5, 6, 8]$
Vertex 3: $[0, 1, 2, 3, 6, 7, 8]$	Vertex 4: $[0, 2, 4, 6, 7, 8]$	Vertex 5: $[0, 1, 2, 5, 6, 8]$
Vertex 6: $[0, 2, 3, 4, 5, 6]$	Vertex 7: $[0, 1, 3, 4, 4]$	Vertex 8: $[0, 1, 2, 3, 4, 5, 8]$

- 4) For each set we have to find all combinations of size k and check if it is independent



Our Solution

Take Vertex 0: [0 , 3 , 4 , 5 , 6 , 7 , 8] for example:

$$\text{Combinations, } {}_nC_r = \frac{7!}{3! \times (7-3)!} = \mathbf{35}$$

If K = 3 then for this set of 7 numbers there are 35 combinations of size 3

Using the java code from GeeksforGeeks

(<https://www.geeksforgeeks.org/print-all-possible-combinations-of-r-elements-in-a-given-array-of-size-n/>)

We are able to find all 35 combinations from the set

[0, 3, 4], [0, 3, 5], [0, 3, 6], [0, 3, 7], [0, 3, 8],
[0, 4, 5], [0, 4, 6], [0, 4, 7], [0, 4, 8], [0, 5, 6],
[0, 5, 7], [0, 5, 8], [0, 6, 7], [0, 6, 8], [0, 7, 8],
[3, 4, 5], [3, 4, 6], [3, 4, 7], [3, 4, 8], [3, 5, 6],
[3, 5, 7], [3, 5, 8], [3, 6, 7], [3, 6, 8], [3, 7, 8],
[4, 5, 6], [4, 5, 7], [4, 5, 8], [4, 6, 7], [4, 6, 8],
[4, 7, 8], [5, 6, 7], [5, 6, 8], [5, 7, 8], [6, 7, 8]]



We have to check each set
to see whether an
Independent set exist



Our Solution

To check to see if a set is Independent we can take $[0, 3, 4]$ and $[0, 3, 8]$ from the set of combinations that we got from the previous slide.

	0	1	2	3	4	5	6	7	8
0	0	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	1	0	0
2	1	1	0	0	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0
4	0	1	0	1	0	1	0	0	0
5	0	0	0	1	1	0	0	1	0
6	0	1	0	0	0	0	0	1	1
7	0	0	1	0	0	1	1	0	1
8	0	0	0	0	0	0	1	1	0

Set : $[0, 3, 4]$

1. We start at row 0 and see if column 3 and 4 is either 1 or 0.
2. If the value is 1 then that means there is an edge so it is not independent. We stop and check the next set
3. If the value is 0 we go to the next row which is 3 and we go back to step 1.

At row 3 column 4 there it value is 1. Thus, the set is **not Independent**

Set: $[0, 3, 8]$

Row 0 column 3 and 8 are zeros

Row 3 column 0 and 8 are zeros

Row 8 column 0 and 3 are zeros

The set is **Independent**

Java

```
20 Project:IndependentSet
9
10 /**
11  * @author Samuel Lin, A01079605
12  *
13  */
14
15 public class Driver {
16
17     /**
18      * @param args
19      */
20     public static void main(String[] args) {
21         // Create IndependentSet Object
22         IndependentSet indSet = new IndependentSet();
23
24         // Model the graph with a two-dimensional matrix
25         int[][] graph = { { 0, 1, 1, 0, 0, 0, 0, 0, 0 }, { 1, 0, 1, 0, 1, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1, 1, 0, 0, 0 },
26             { 0, 1, 0, 1, 0, 1, 0, 0, 0 }, { 0, 0, 0, 1, 1, 0, 0, 1, 0 }, { 0, 1, 0, 0, 0, 0, 0, 1, 1 }, { 0, 0, 1, 0, 0, 1, 1, 0, 1 },
27             { 0, 0, 0, 0, 0, 0, 1, 1, 0 } };
28
29         int size = 3;
30
31         // Check Decision
32         if (indSet.check(graph, size)) {
33             System.out.println("Yes, an Independent set of the given size exist in the graph");
34         } else {
35             System.out.println("No, an Independent set of the given size does not exist in the graph");
36         }
37     }
38 }
39
40 }
41
```



We set up the matrix with ones and zeros in a two-dimensional array

Java

```
1 |
3 * Project:IndependentSet[]
9
10 import java.util.ArrayList;[]
15
16 /**
17 *
18 */
19 public class IndependentSet {
20     ArrayList<Integer> IndependentSet = new ArrayList<Integer>();
21
22     // Constructor
23     public IndependentSet() {
24         super();
25         // TODO Auto-generated constructor stub
26     }
27
28 /**
29 *
30 * @param graph
31 *         - undirect graph in matrix form
32 * @param size
33 *         - size of Independent set
34 * @return "True", if size k Independent set exist. "False" , if size k independent set does not exist
35 */
36
37 // check to is there is a size k independent set in the given graph
38 public boolean check(int[][] graph, int size) {
39
40     // Empty Independent set
41     Object arr[] = {};
42
43     // Create an empty HashMap
44     Map<Integer, ArrayList<Integer>> map = new HashMap<Integer, ArrayList<Integer>>();
45
46     // Decision, either "Yes" or "No"
47     Boolean output = false;
48
49     // Put vertex numbers as the key and the values of each each is an empty ArrayList
50     for (int i = 0; i < graph.length; i++) {
51         map.put(i, new ArrayList<Integer>());
52     }
53
54     // Using a two for-loops we go through the graph and add the vertex with no edges to each set
55     for (int i = 0; i < graph.length; i++) {
56         for (int j = 0; j < graph[i].length; j++) {
57             if (graph[i][j] == 0) {
58                 map.get(i).add(j);
59             }
60         }
61     }
62 }
```

** The function, check() takes in the graph and the size of the Independent Set



For each row, we will remove the nodes that
has an edge between the selected node

Java

```
// We loop through each keySet and the value which represents the set of vertices with not edges, find combinations of given size and check if
// it is independent
for (int i = 0; i < map.keySet().size(); i++) {

    Object[] independentNumbers = map.get(i).toArray();
    arr = independentNumbers;

    System.out.println("Vertex " + i);
    System.out.println(Arrays.toString(independentNumbers));

    // Create Combination class
    Combination getCombination = new Combination();

    // We find all combinations of size k
    List<ArrayList<Integer>> allCombinations = getCombination.CreateCombination(arr, arr.length, size);

    System.out.println(" ");
    System.out.println("Possible combinations with given size:");
    System.out.println(allCombinations);
    System.out.println(" ");

    // loop through each combination, and check if it there is one that is independent
    for (int k = 0; k < allCombinations.size(); k++) {
        if (FesibleSet(graph, allCombinations.get(k), size)) {
            output = true;
        }
    }

    return output;
}

/**
 * @param graph - undirected graph
 * @param Independentset - vertices that are non-adjacent
 * @param size - size of Independent set
 * @return "True", if the set is independent . "False", if the set is not independent
 */
// Check to see if the set is Independent
public boolean FesibleSet(int[][] graph, ArrayList<Integer> Independentset, int size) {

    for (int i = 0; i < Independentset.size(); i++) {
        for (int j = 1; j < Independentset.size(); j++) {
            // if two vertices have an edge between we return false. Note*, 1 represents an edge in the matrix
            if (graph[Independentset.get(i)][Independentset.get(j)] == 1) {
                return false;
            }
        }
    }

    // return true there are no edges between the vertices in the set.
    return true;
}

}
```



Find all combinations of size k



Using two for-loops we can check to see if the set is independent or not

Output

Yes, an Independent set of the given size exist in the graph

OR

No, an Independent set of the given size does not exist in the graph



Reference

Reference(i)

Bernard, Z. (2018, November 10). Everything you need to know about Bitcoin, its mysterious origins, and the many alleged identities of its creator. Retrieved from

<https://www.businessinsider.com/bitcoin-history-cryptocurrency-satoshi-nakamoto-2017-12#dorian-nakamoto-nick-szabo-and-craig-wright-arent-the-only-ones-who-been-pinned-as-the-inventor-of-bitcoin-11>.

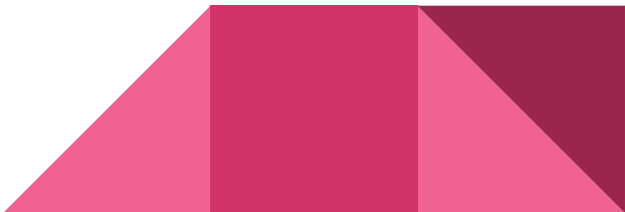
Bitcoin is NP-hard. (n.d.). Retrieved from <https://tsa-heidi.github.io/Bitcoin/>.

Bitcoin mining the hard way: the algorithms, protocols, and bytes. (n.d.). Retrieved from

<http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html>.

Bitcoin mining is NP-hard. (n.d.). Retrieved from <https://freedom-to-tinker.com/2014/10/27/bitcoin-mining-is-np-hard/>.

Branch Algorithm Kratsch, D. (n.d.). Retrieved from [https://www.sop.inria.fr/mascotte/seminaires/AGAPE/lecture_notes/Agape 09 - Dieter Kratsch - B3e](https://www.sop.inria.fr/mascotte/seminaires/AGAPE/lecture_notes/Agape%2009%20-%20Dieter%20Kratsch%20-%20B3e)



Reference(ii)

Fortney, L. (2019, September 20). Blockchain Explained. Retrieved from <https://www.investopedia.com/terms/b/blockchain.asp>.

Independent set (graph theory). (2019, November 13). Retrieved from [https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory)).


Installing¶. (n.d.). Retrieved from <https://matplotlib.org/2.0.2/users/installing.html>.

Independent set (graph theory). (2019, November 13). Retrieved from [https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory)).

Kettley, S. (2017, December 21). Bitcoin price: How many bitcoin are there and when will the popular crypto token run out? Retrieved from <https://www.express.co.uk/finance/city/895187/bitcoin-price-how-many-bitcoin-are-there-when-will-cryptocurrency-token-run-out>.

Mathieu Mari, *Study of greedy algorithm for solving Maximum Independent Set problem*, (CNAM • ENS Rennes, 2017), 1-2, 4-6.

Mount, D. (2019). *CMSC 451: Lecture 20 NP-Completeness: 3SAT and Independent Set*. [ebook] CMSC 451. Available at: <http://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect20-np-3sat.pdf> [Accessed 17 Oct. 2019].



Reference(iii)

Mostafa H. D.(2014, May 31). Maximum Independent Set Approximation Based on Bellman-Ford Algorithm

Semidoc.(2018, Jan 29). Greedy Algorithm for Maximum Independent Set. <https://semidoc.github.io/greedyMIS>

Studies Studio . (2018 , June 15). NP-completeness of Independent Set with Proof || By Studies Studio [Video file]. Retrieved from <https://www.youtube.com/watch?v=ub8qsgPamqE>

References. (. Retrieved from <http://www.cs.princeton.edu/~wayne/kleinberg-tardos>.

