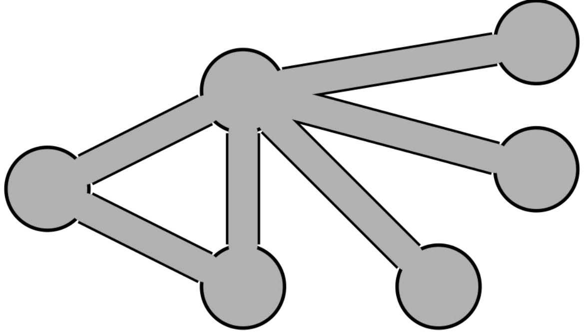
Min Vertex Cover Problem

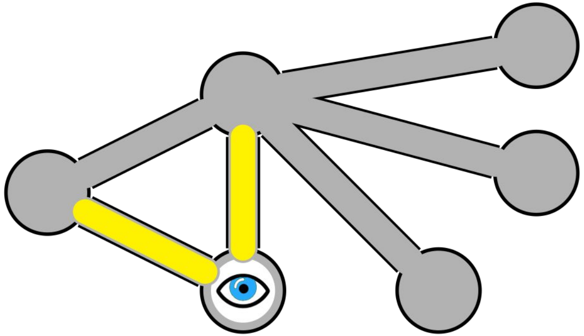
By: Mason and Chingiz

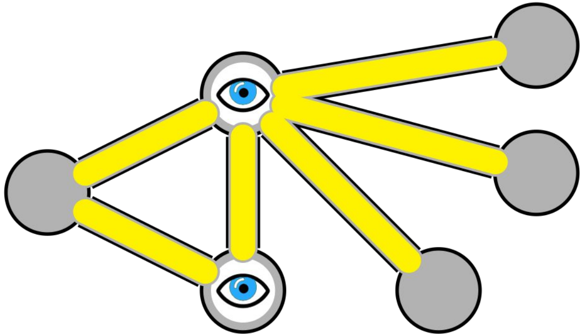
What is a Vertex Cover

A vertex cover of an undirected graph is a subset of its vertices where:

- For each edge (u, v) of the graph, either u or v is in the vertex cover.
- This set of vertices should be incident to all edges of the graph.
- The minimum vertex cover is the smallest subset of vertices (there can be more than one correct solution)



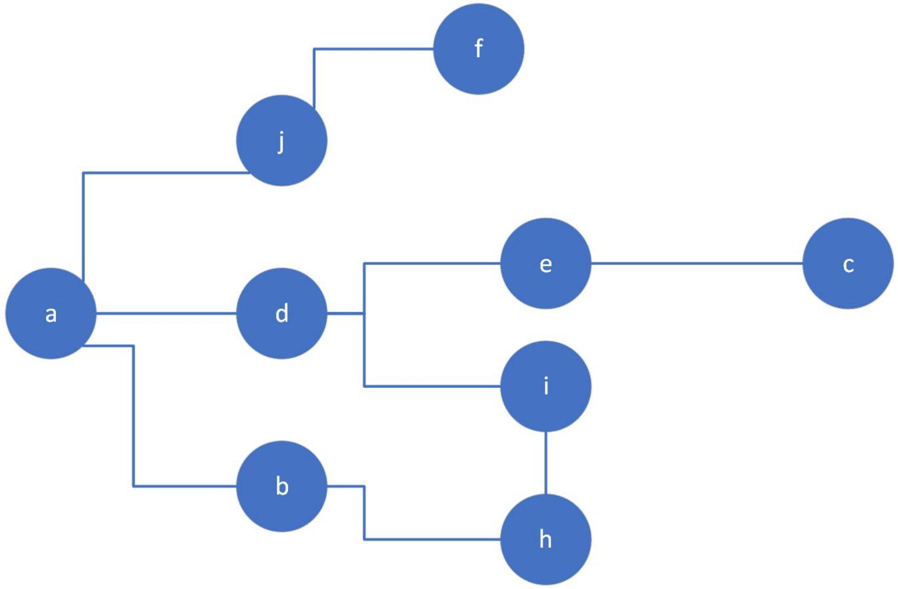




Decision vs Optimization

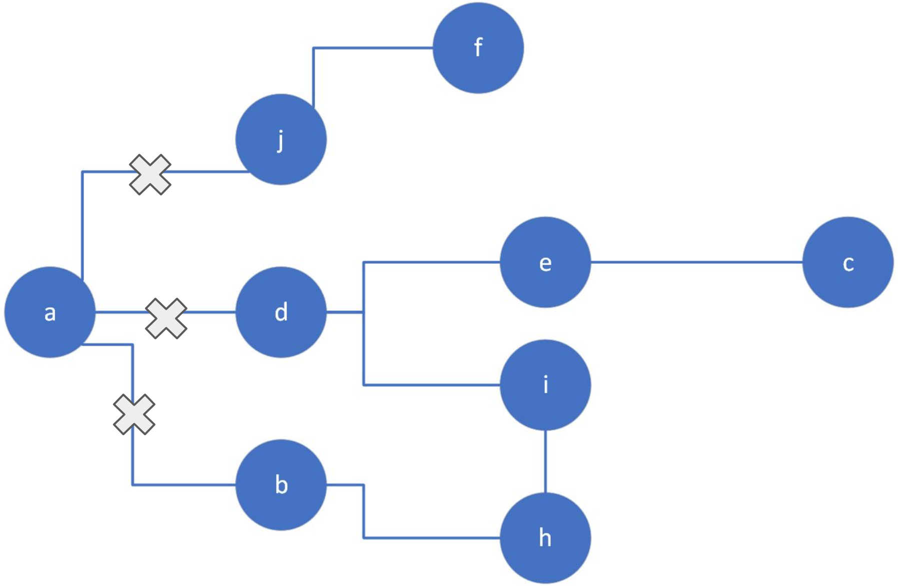
- Decision: Given an undirected graph G and an integer k. Is there a vertex cover of size ≤ k.
  - If there are zero edges, then the minimum vertex cover is a set containing nothing
  - If there are edges, some set of vertices exists that is incident to all of the edges
  - The set of all vertices contains all edges, so it is a valid vertex cover
- Optimization: For any given graph, is it possible to find the minimum vertex cover
  - From our decision problem, we know that a valid vertex cover exists for any graph
  - If we test all possible combinations of vertices, at least one will be the minimum vertex cover

Example Input and Output



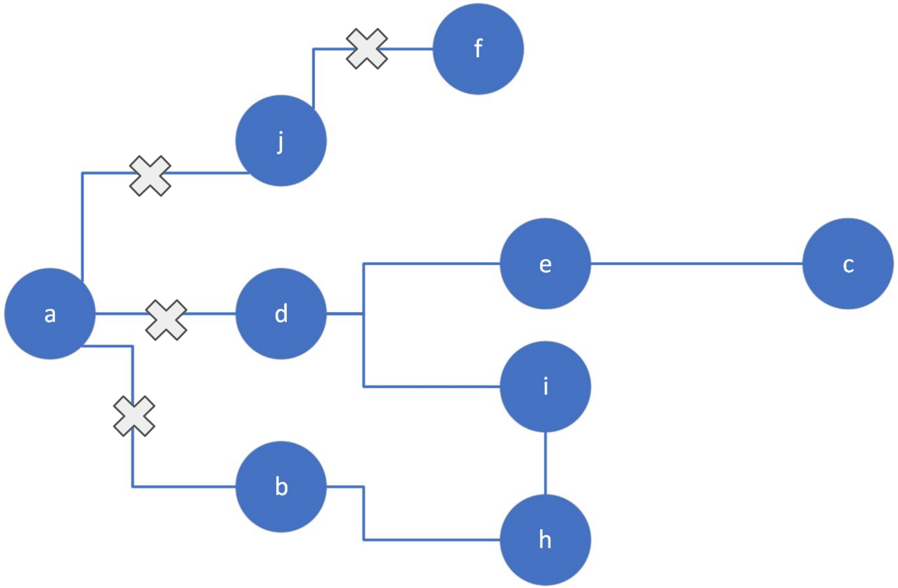
Output: a j d e h

Example Input and Output



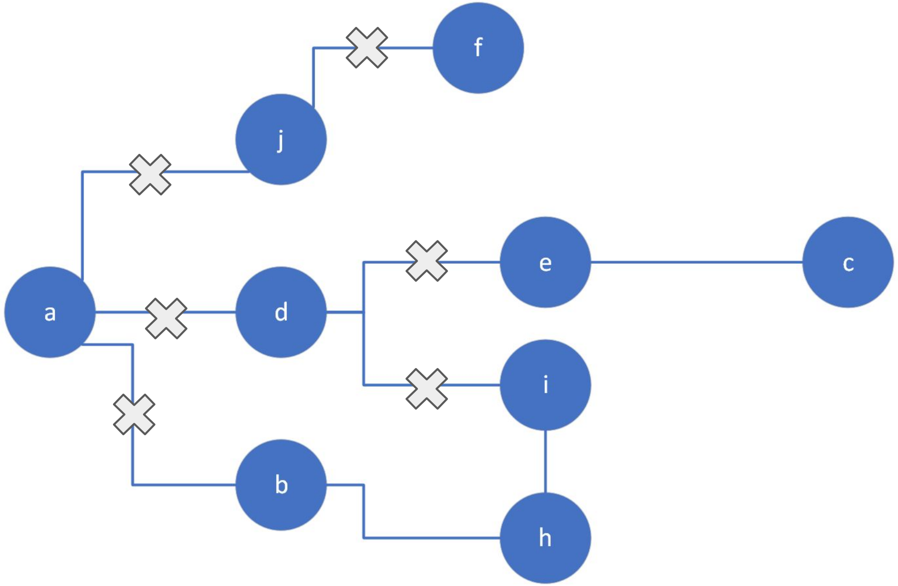
Output: **a** j d e h

Example Input and Output



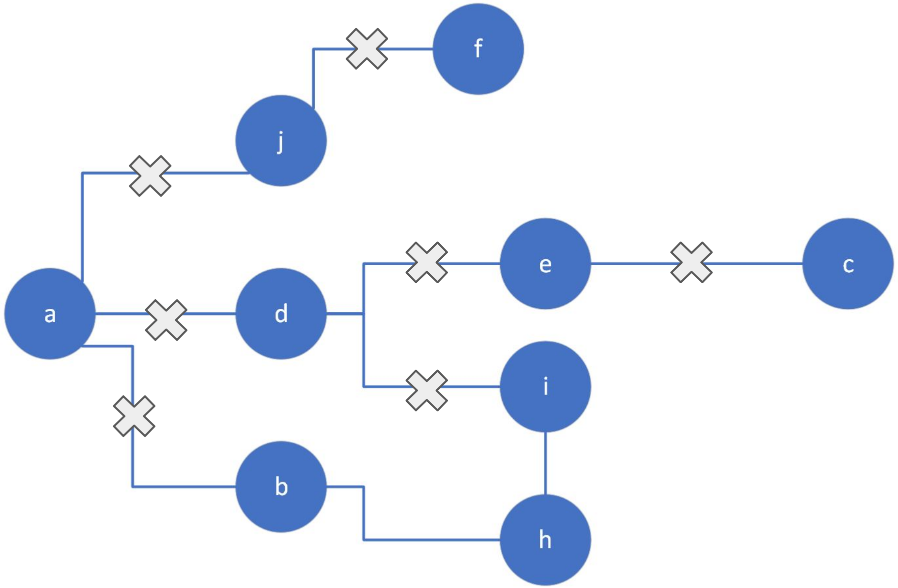
Output: a **j** d e h

Example Input and Output



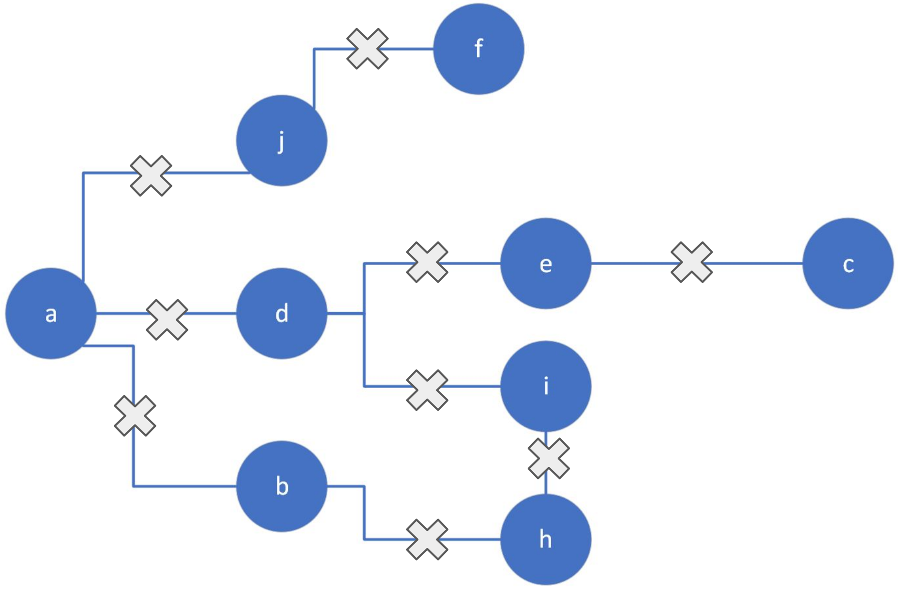
Output: a j **d** e h

Example Input and Output



Output: a j d **e** h

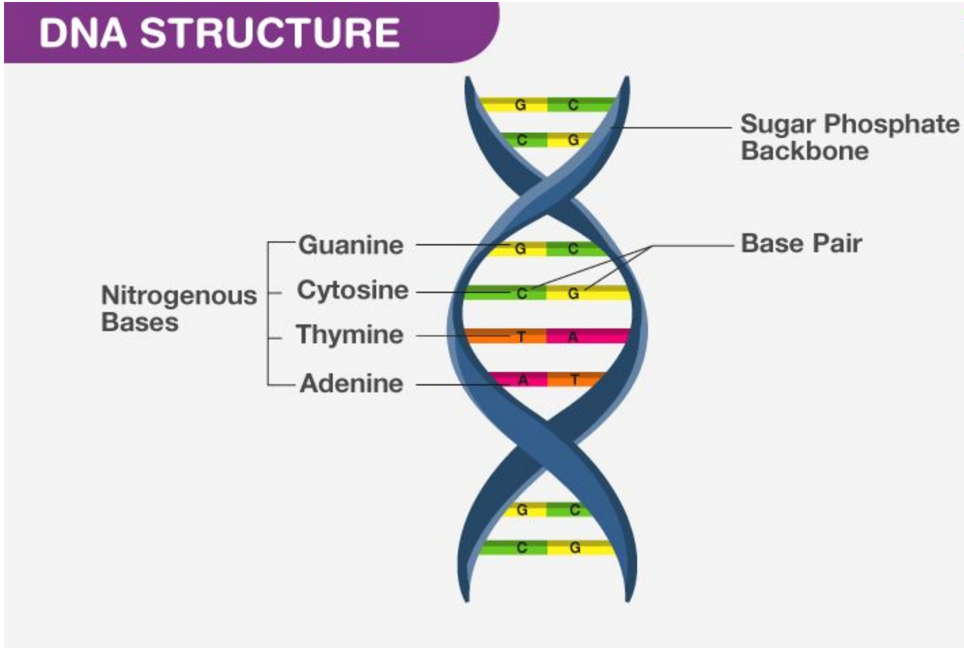
Example Input and Output



Output: a j d e **h**

Applications of Min Vertex Cover

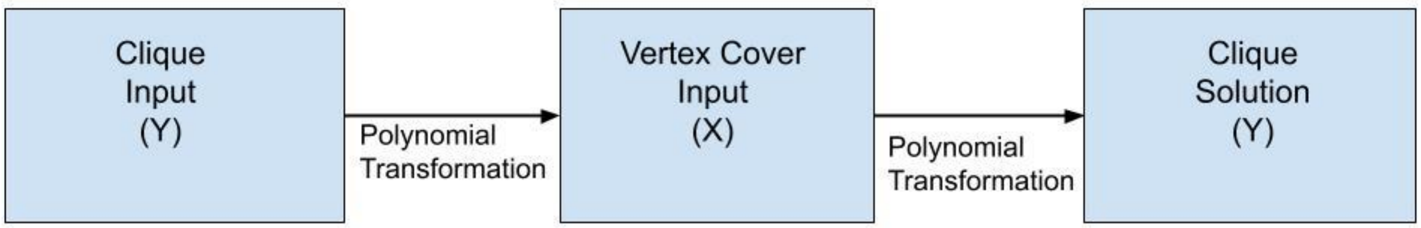
- Automating design of thousands of nonrepetitive parts for engineering stable genetic systems
  - Engineered genetic systems are prone to failure when their genetic parts contain repetitive sequences
  - Multiple parts may have the same repetitive sequence
  - Each genetic part is represented as a vertex
  - The minimum vertex cover represents the smallest number of genetic parts that have at least one shared repetitive sequence
  - After removing the vertex cover, we are left with all of our nonrepetitive genetic parts



https://www.nature.com/articles/s41587-020-0584-2

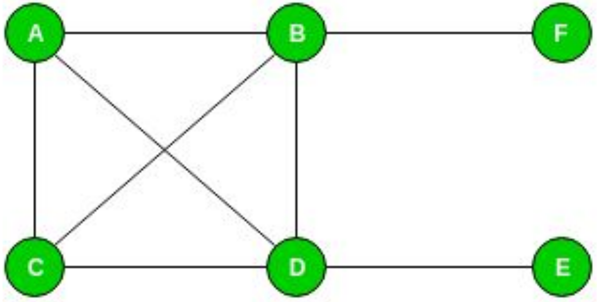
NP Complete Reduction 1/2

- As a reduction we take the Clique problem (which is NP complete) and reduce it to the Vertex Cover problem
    - *"In computer science, the clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph."*

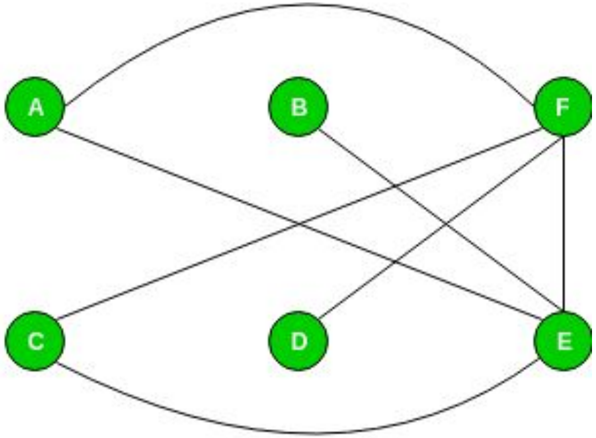


NP Complete Reduction 2/2

- *Consider the graph G’ which consists of all edges not in G, but in the complete graph of G*
- *There is a clique of size k in graph G if and only if there is a vertex cover of size V - k in G’*
- *Any instance of the clique problem can be reduced to an instance of the vertex cover problem, so vertex cover is NP hard*
- *Since vertex cover is in both NP and NP hard, it is NP complete*



G
V' ={A, B, C, D}



G'
V'' ={E, F}

Our Exact Solution

- We call our certifier for each subset of vertices
- Each subset starts with small combinations and grows until it covers the whole graph
- For every vertex of the subset remove all of its edges
- If there are no more edges than we have found the first vertex cover, so we break out of the loop

```python
def vertex_cover(graph):
    min_cover = None

    def check_cover(i):
        for combo in itertools.combinations(graph, i):
            g_clone = {u:set([v for v in graph[u]]) for u in graph}

            for v in combo:
                if v not in g_clone:
                    continue
                for u in g_clone[v]:
                    g_clone[u].remove(v)
                    if not g_clone[u]:
                        g_clone.pop(u)
                g_clone.pop(v)
            if not g_clone:
                return combo

        return None

    for i in range(len(graph) + 1):
        min_cover = check_cover(i)
        if min_cover != None:
            break

    print(" ".join(min_cover))
```

Our Exact Solution Runtime

- Our solution's runtime is $O(2^n)$
- In the worst case our outer for loop will call check_cover() n times where n is the number of vertices
- The loop that iterates through all possible combos of n vertices will run 2^n times in the worst case

```python
def vertex_cover(graph):
    min_cover = None

    def check_cover(i):
        for combo in itertools.combinations(graph, i):
            g_clone = {u:set([v for v in graph[u]]) for u in graph}
            for v in combo:
                if v not in g_clone:
                    continue
                for u in g_clone[v]:
                    g_clone[u].remove(v)
                    if not g_clone[u]:
                        g_clone.pop(u)
                g_clone.pop(v)
            if not g_clone:
                return combo

        return None

    for i in range(len(graph) + 1):
        min_cover = check_cover(i)
        if min_cover != None:
            break

    print(" ".join(min_cover))
```

Analyzing the Runtime

- The operations that are executed the most and take up the most runtime are:
  - Dictionary/list comprehension for creating graph copies
  - Pop/remove for removing vertices/edges from our graph

```
a i r c k o g f n q l
         36777985 function calls in 21.559 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   21.559   21.559 {built-in method builtins.exec}
        1    0.000    0.000   21.559   21.559 cs412_minvertexcover_exact.py:1(<module>)
        1    0.000    0.000   21.559   21.559 cs412_minvertexcover_exact.py:13(main)
        1    0.000    0.000   21.559   21.559 cs412_minvertexcover_exact.py:25(vertex_cover)
       12    7.704    0.642   21.558    1.797 cs412_minvertexcover_exact.py:28(check_cover)
   648795    6.928    0.000   11.225    0.000 cs412_minvertexcover_exact.py:30(<dictcomp>)
 12975900    4.297    0.000    4.297    0.000 cs412_minvertexcover_exact.py:30(<listcomp>)
 16871868    1.802    0.000    1.802    0.000 {method 'remove' of 'set' objects}
  6281170    0.827    0.000    0.827    0.000 {method 'pop' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
       39    0.000    0.000    0.000    0.000 {built-in method builtins.input}
       76    0.000    0.000    0.000    0.000 {method 'setdefault' of 'dict' objects}
       76    0.000    0.000    0.000    0.000 {method 'add' of 'set' objects}
       38    0.000    0.000    0.000    0.000 {method 'split' of 'str' objects}
        1    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
        1    0.000    0.000    0.000    0.000 <frozen codecs>:281(getstate)
        1    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

Test Case Generation 1/2

- For our test generation our loop index is the chance (percentage) of connecting 2 vertices with an edge.
- There are 2 parameters, one for the number of vertices (vertex_count) and the other for how much to increase the chance by (step_count)
- The loop exits after using an edge chance that is over 100 (100%)
- We use the same parameters and loop for running all our test cases

```
for (( i=5 ; i<=150 ; i=i+$step_count ));
do
    let x=$file_count y=1 file_count=x+y
    if ! python generate_test.py $vertex_count $i $file_count; then
        exit
    fi
    if [ "$i" -gt 100 ]; then
        exit
    fi
done
```

```
for (( i=5 ; i<=150 ; i=i+$step_count ));
do
    let x=$file_count y=1 file_count=x+y
    file="test_cases/input"$file_count".txt"

    time (python cs412_minvertexcover_exact.py < $file >> $output) 2>> times.txt

    if [ "$i" -gt 100 ]; then
        exit
    fi
done
```

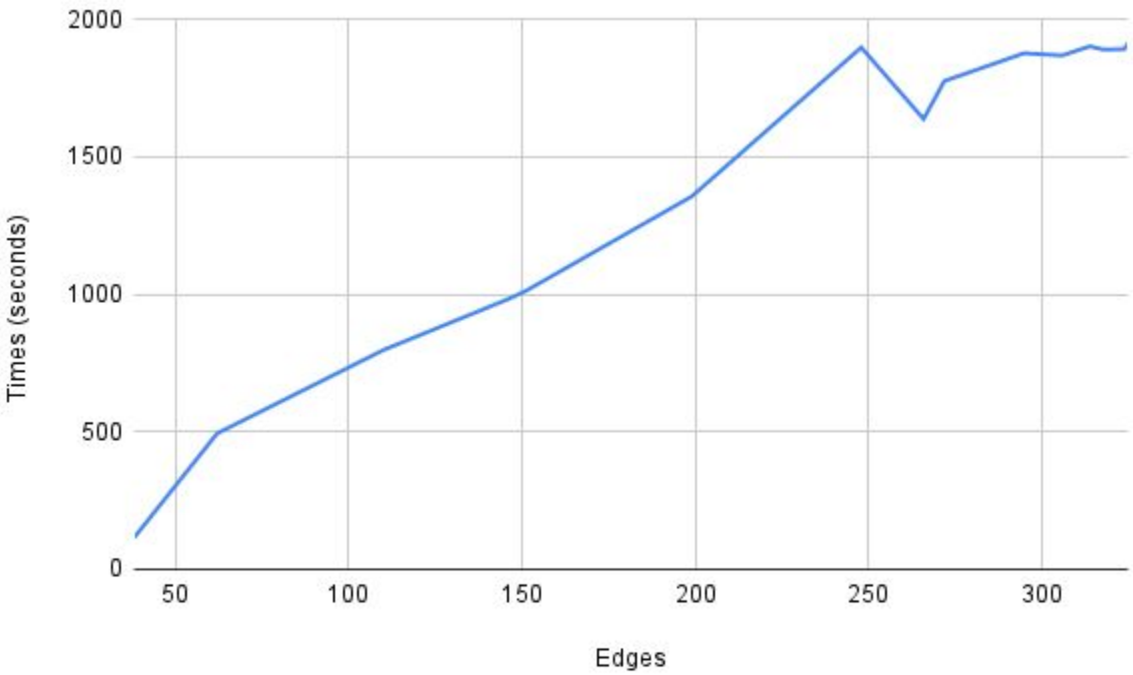
Test Case Generation 2/2

- We generate a graph using the vertex count and edge chance from our bash script
- All edges are randomly generated using the edge chance
- We then use the graph to create input file

```python
def generate_graph(vertices, edge_chance, file_num):
    # Graph object
    graph = {v:[] for v in vertices}
    edge_count = 0

    for u in graph:
        for v in graph:
            if u == v or u in graph[v]:
                continue
            percent = random.random()
            if edge_chance >= percent:
                graph[u].append(v)
                graph[v].append(u)
                edge_count += 1
```

Test Size vs Runtime



Sources

https://www.geeksforgeeks.org/introduction-and-approximate-solution-for-vertex-cover-problem/

https://commons.wikimedia.org/wiki/File:Couverture_de_sommets.svg

https://byjus.com/biology/properties-of-dna/

https://www.nature.com/articles/s41587-020-0584-2

https://www.geeksforgeeks.org/proof-that-vertex-cover-is-np-complete/#