# Assignment-4

Rachit Parikh (CRS-2101)

Design and Analysis of Algorithms

**Disclaimer :** I declare that all the work presented in this assignment is my own work and I have only consulted the internet when it was absolutely necessary. Search problems definition is taken from [1].

---

## Q-1

### A

**Idea :** Instance $I$ would be the graph $G(V, E)$ and the solution $S$ would be a simple path of length $k$. We would just check whether this path is simple and contains $k$ vertices. Since this can be done in polynomial time of the instance $I$ we will say that this problem is in $\mathcal{NP}$.

To show that we can verify the evidence $S$ which is nothing but a path defined as $S = (v_1 e_{1,2} v_2 e_{2,3} \dots e_{k-1,k} v_k)$ where $e_{1,2}$ is edge between vertex $v_1$ and $v_2$. This is a simple path of length $k$. What we will do is just parse this evidence $S$ and keep a check whether any vertex was repeated or not. Hence our algorithm $\mathcal{C}(I, S)$ will return `YES` if the path $S$ is simple and contains $k$ vertices. Algorithm $\mathcal{C}$ is polynomial time of instance $I = V$, because any simple path can have at most $|V|$ vertices, hence the run time of the algorithm $\mathcal{C}$ is $\mathcal{O}(|V|)$ which is polynomial and hence the problem is in $\mathcal{NP}$.

### B

**Idea :** Instance is the number $k$ and the size of instance is $\mathcal{O}(\log k)$. Take a non-trivial divisor (divisor of $k$ other than $k$ and 1) as evidence. Since the divisor can be at most $k$, integer divison will take polynomial time to the size of input (because the size of divisor cannot be larger than that of $k$). This will prove that $k$ indeed is composite by definition.

Algorithm $\mathcal{C}$ will simply take the instance $I = k$ a non-trivial divisor $S = d$ where $1 < d < k$. This will ensure that size of $d$ is $\mathcal{O}(\log k)$. Since integer division takes polynomial time to size of input, our algorithm $\mathcal{C}(I, S)$ will simply return `YES` if the evidence $d$ is a non-trivial divisor of $k$. Since the runtime of our algorithm would be equal to integer division (which is polynomial), the proble is in $\mathcal{NP}$

## C

**Idea :** Instance is graph $G(V, E)$ and size $k$. Solution here would be a set $S$ which will be a vertex cover of size of $k$. We simply need to check for all edges $(u, v) \in E$ at least one of the vertex $u$ or $v$ is included in the set $S$. Hence the problem of verfiying whether there is a vertex cover of size $k$ can be done in polynomial time of the number of edges.

Algorithm $\mathcal{C}(I, S)$ will take instance $I = (G, k)$ where $G$ is the graph and $k$ is the size of vertex cover. The solution $S$ is simply a set of vertices of size $k$. Algorithm $\mathcal{C}$ just goes through all the edges $E$ and checks for each edge $(u, v)$ whether $u \in S$ or $v \in S$. If neither of the vertices are in $S$, then we can say that $S$ is not a vertex cover of $G$. If at least one of the vertices are present in $S$ then we continue the same procedure for other edges. The process terminates once we have checked all the edges. Thus the runtime of the algorithm $\mathcal{C}$ is $\mathcal{O}(E)$. Hence, the problem is in $\mathcal{NP}$.

## D

The search problem formulation of ILP can be stated as : "Given set of linear inequalities $\mathbf{Ax} \leq \mathbf{b}$ where $\mathbf{A}$ is a $m \times n$ matrix and $\mathbf{b}$ is an $m-$vector. There is an additional constraint where all coordinates of vector $\mathbf{x}$ have to be integers. We are given an objective function by $n-$vector $\mathbf{c}$ and a goal $k$. Now the problem is to find $\mathbf{x}$ such that $\mathbf{Ax} \leq \mathbf{b}$ and $\mathbf{c}.\mathbf{x} \leq k$".

This is very easy to verify in polynomial time, given we have the solution vector $\mathbf{s}$ and instance $I = (\mathbf{A}, \mathbf{b}, \mathbf{x})$. We simply need to multiply the matrix $\mathbf{A}$ with $\mathbf{x}$ which takes $\mathcal{O}(mn)$ time. Checking the constraints would just take $\mathcal{O}(m)$ time. Now to check the cost function it will take $\mathcal{O}(n)$ time. Hence the overall algorithm for checking takes $\mathcal{O}(mn)$ time. Thus the problem ILP at most $k$ is in $\mathcal{NP}$.

# Q-5

**Idea :** To prove that LONGEST PATH is $\mathcal{NP}$-complete we need to show an appropriate $\mathcal{NP}$-complete problem and reduce that problem to the LONGEST PATH problem. To do this we will use the HAMILTONIAN PATH problem.

First we prove that LONGEST PATH is in $\mathcal{NP}$. This can be done easily by taking the path $p = \{v_1, e_{1,2}, v_2, \ldots v_k, e_{k,k+1}, v_{k+1}\}$ and for this we just traverse through the path and check if any vertex was repeated or not. This can be done in $\mathcal{O}(k)$ time and $k \leq |V|$, hence polynomial in time of input. Thus LONGEST PATH is in $\mathcal{NP}$.
HAMILTONIAN PATH : Find a simple path between vertices $s$ and $t$ such that they cover all the vertices. Since we are reducing the HAMILTONIAN PATH problem to LONGEST PATH, we need a simple path from $s$ to $t$ such that it passes through all the vertices so the goal for LONGEST PATH problem would be $g = |V| - 1$. We need to prove for both 'yes' and 'no' instances.
First we convert instance of HAM-PATH to instance of LONGEST PATH. This can be done by taking an extra vertex $x$ and adding it to the graph between vertices $s$ and $t$ between which we are going to find the hamiltonian path. We add the edges $(s, x)$ and $(t, x)$ and create a new graph $G'$ which will be the instance for the LONGEST PATH problem. Along with the graph our goal in the LONGEST PATH problem here would be $g = |V|$ since one vertex has increased the simple path would now traverse through $|V|$ edges. We prove the following :

1. *'yes' instance of HAM-PATH* $\implies$ *'yes' instance of LONGEST PATH*
   If there exists a Hamiltonian path between vertices $s$ and $t$ (let's say it is $p$) then there exists a simple path of length $|V|$ in graph $G'$. We simply add the edge $(t, x)$ and the vertex $x$ to the path $p$ which creates a new simple path of length $|V|$.

2. *'no' instance of HAM-PATH* $\implies$ *'no' instance of LONGEST PATH*
   Instead of proving this directly we prove its contrapositive. We try to show that for a 'yes' instance of LONGEST PATH there exists a 'yes' instance for hamiltonian path. This can be shown easily just by removing the vertex $x$ from the graph as well as from the LONGEST PATH solution. Since the lenght of path was $|V|$ and we removed a vertex, the new length would be $|V|-1$ hence there is a path which contains every vertex of graph $G$. Since this path has all vertices on it, it will lead $s$ to $t$ hence a hamiltonian path exists.

# Q-2

**Idea :** We will be using VERTEX COVER to prove that HITTING SET is $\mathcal{NP}$-complete. We will reduce VERTEX COVER to HITTING SET.

We begin by proving that HITTING SET is in $\mathcal{NP}$. This is a decision problem which we will first convert into a search problem. The search problem version of decision problem will become "Find a set $H$ of size $b$ that intersects every set $S_i$". Since this is a search problem we can say that it has instance $I = (b, S_1, S_2, \dots S_n)$ and the solution for this problem would be a set $H$ of size $b$. We can verfiy this solution by just going over each set at a time and checking if there exists at least one element from each set $S_i$ that lies in $H$. If a set contains at least one common element with $H$, we stop our search. If no element exists in a set $S_i$ which is also in $H$ then we know $H$ is not a solution. Hence this whole search can be done in the linear time of total number of elements in each set, this is in $\mathcal{NP}$.

To prove it is in $\mathcal{NP}$-complete we will be using VERTEX COVER. The search problem formulation for VERTEX COVER can be stated as : "For a graph $G$, we need to find a set of $b$ vertices such that they touch every edge of the graph". Here the instance $I = (G, b)$. Now we convert this instance to that of HITTING SET problem. To define instance for the HITTING SET problem we take each edge $i$ as a set $S_i$ which contains as elements the vertices forming that edge. So if the edge $i$ is $(u, v)$ then $S_i = \{u, v\}$. This way we can form the sets $S_1, \dots S_{|E|}$. This instance conversion would take polynomial time of the input because the number of edges is an instance of the problem. Now we prove the following two things:

1. *'yes' instance of VERTEX COVER* $\implies$ *'yes' instance of HITTING SET*
   If there exists a vertex cover of size $b$, it means that there is a set of vertices such $V'$ of size $b$ such that they can cover all the edges. Another way of seeing this is that for each edge $e = (u, v)$, either $u \in V'$ or $v \in V'$. This means that either $H \cap S_i \neq \phi$. Since $H$ has size $b$ we have a yes instance for HITTING SET.

2. *'no' instance of VERTEX COVER* $\implies$ *'no' instance of HITTING SET*
   We will prove the converse and show that a 'yes' instance of VERTEX COVER exists if a 'yes' instance of HITTING SET is observed. If we have a 'yes' instance of HITTING SET then we know there is a set $H$ of size $b$ which will intersect each set $S_i$. This simply means there is no single edge which is not covered by the set $H$, thus $H$ is vertex cover by definition.

# Q-6

Assuming we have to find the shortest vertex cover, that would be our cost function $f$. Since for a single edge $e = (u, v)$ there should be at least one of the $u$ or $v$ inside the set cover, we will use indicator variables to describe whether a vertex is in vertex cover or not.

$$u_i = \begin{cases} 0 & \text{if } u_i \in S \\ 1 & \text{if } u_i \notin S \end{cases}$$

Here $S$ is the set cover that we are searching for. Since for each edge we need at least one vertex inside the set cover $S$, we also have the following constraints:

$$u_i + u_j \geq 1, \ \forall (u_i, u_j) \in E \tag{1}$$
$$u_i \in \{0, 1\}, \ \forall u_i \in V \tag{2}$$
$$\tag{3}$$

Finally our cost function $f$ subject to constraints above would be,

$$f(G) = \min \sum_{i=1}^{|V|} u_i$$

# Q-3

Since the problem $\Pi$ is in $\mathcal{NP}$, it is a search problem. Any search problem can be described by an algorithm $\mathcal{C}$ which will run in time polynomial of the instance $I$ and check for solution $S$.

Since $\mathcal{C}$ is a polynomial time, it can be rendered as a circuit whose inputs encode input to the algorithm. We will encode the input of the problem $\Pi$, (assume it to be $I$) as a polynomial time input for the circuit. Since we know that CIRCUIT SAT is in NP, we can check for the input in polynomial time of the input. Let the input to CIRCUIT SAT be $p(I)$ where $I$ is the input of the given problem $\Pi$, then it will take polynomial time $q(p(I))$ to check the instance $p(I)$.

This is verification part, now by brute force we can check for all the possible combination of input gates, whether the CIRCUIT SAT gives us a **true** instance. There will be total of $2^{p(I)}$ inputs, and each input can be checked by the CIRCUIT SAT in polynomial time. So the total time to solve a problem $\Pi$ would be $\mathcal{O}(q(n)2^{p(n)}) = \mathcal{O}(2^{p(n)})$, (here $n = I$).

# Q-4

## A

In the algorithm above, we are iterating through the edge list and removing incident edges to a corresponding edge. One thing to observe here is the loop invariant. The loop invariant is a property of the set $C$. Let us define $E_j$ as the set $E_0$ before the $j-$th iteration. The loop invariant here is "Before the $j-$th iteration the edges in $E_j$ are not incident to any edge in $C$".

Initialization : dBefore the first loop, $C$ is empty and hence the invariant is vacuously true.

Maintenence : Before the $j$th iteration we assume the property holds. Now, in the $j-$th loop we will be removing all the incident edges to the edge $(u, v) \in E_j$. Since we are already assuming that none of these edges are incident to the edges in $C$, and the edges incident to $(u, v) \in C$ are also removed, in the $j + 1$th iteration there would be no edge in $E_{j+1}$ that would be incident to any edge in $C$ (by induction and incident edge of $(u, v)$ are already removed).

Termination : At the end of loop the set $E_0$ becomes empty and we finally get a set $C$ which contains edges to which all edges in $E$ are incident with. This means that the vertices in these edges will cover all the edges of a graph. This implies we have found a set cover.

## B

At each iteration of the algorithm we are removing at least one edge from the set $E_0$ which contains $|E|$ edges. In the worst case we have a disconnected graph where each edge is not incident to any other edge of the graph and in this case also the algorithm would take $|E|$ steps to terminate. So the algorithm takes $O(|E|)$ time and hence is polynomial.

## C

For an edge $(u, v)$, at each iteration we are removing at most $deg(u) + deg(v) - 1$ number of edges from the edge list. For the optimal vertex cover $O$, we have $\sum_{u_i \in O} deg(u_i) \geq |E|$.

# References


[1] Dasgupta, S., Papadimitriou, C. and Vazirani, U., (2011). *Algorithms*. Boston: McGraw-Hill Higher Education.