# CS 6363 Design and Analysis of Algorithms
# Fall 2019 - Homework #6 Solutions

## Problem # 1 (#34.5 – 5) Set-partition problem

(1) Set-Partition $\in$ NP

Simply guess $I \subseteq \{1, 2, ..., n\}$

```
1. S_1 ← 0, S_2 ← 0
2. for i ← 0 to n do
3.       if i ∈ I then S_1 ← S_1 + a_i
4.            else S_2 ← S_2 + a_i
5. return YES ⇔ S_1 = S_2
```

(2) Set-Partition $\in$ NP-Hard

Subset-Sum $\in$ NP-Complete
Instance: A set $S$ and an integer $k$
Question: Is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = k$?

We know that Subset-Sum $\in$ NP-Complete. For the proof, we show that Subset-Sum $\leq_p$ Set-Partition. Let integers $a_1, a_2, ..., a_n, b$ be an instance of Subset-Sum problem. We construct an instance of Set-Partition problem as follows:
$a_1, a_2, ..., a_n, a_{n+1}, a_{n+2}$, where $a_{n+1} = b + 1$, $a_{n+2} = \sum_{i=1}^{n} a_i + 1 - b$
It is easy to see that the construction can be done in polynomial time.
Now, we need to show that

$$\exists I \subseteq \{1, 2, ..., n\} : \sum_{i \in I} a_i = b \iff \exists I' \subseteq \{1, 2, ..., n, n+1, n+2\} : \sum_{i \in I'} a_i = \sum_{i \notin I'} a_i$$

($\Rightarrow$): Let $I' = I \cup \{n + 2\}$. We have following two:

- $\sum_{i \in I'} a_i = \sum_{i \in I} a_i + a_{n+2} = b + \sum_{i=1}^{n} a_i + 1 - b = \sum_{i=1}^{n} a_i + 1$
- $\sum_{i \notin I'} a_i = \sum_{i=1}^{n} a_i - b + a_{n+1} = \sum_{i=1}^{n} a_i - b + b + 1 = \sum_{i=1}^{n} a_i + 1$

Therefore, $\sum_{i \notin I'} a_i = \sum_{i \in I'} a_i$

($\Leftarrow$): Note that exactly one of $n + 1$ and $n + 2$ should be in $I$. Without loss of generality, let $n + 2 \in I$. Let $I' = I - \{n + 2\}$. Then:

$$\begin{aligned} \textstyle\sum_{i \in I'} a_i &= \textstyle\sum_{i \in I} a_i - a_{n+2} = \sum_{i \in I} a_i - \left(\sum_{i \in I} a_i + 1 - b\right) \\ &= \tfrac{1}{2} \textstyle\sum_{i=1}^{n+2} a_i - \sum_{i=1}^{n} a_i - 1 + b = \tfrac{1}{2} \left(2 \sum_{i=1}^{n} a_i + 2\right) - \sum_{i=1}^{n} a_i - 1 + b = b \end{aligned}$$

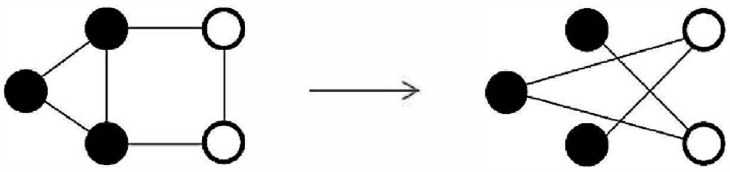
## Problem #2 (#34 – 1) Independent set

(a)

Decision Problem: Given a graph $G = (V, E)$, and an integer $k$, is there an independent set $V' \subseteq V$ of size $k$?

NP-Completeness Proof:

1. An instance of Independent-Set can be verified in polynomial time.
2. Let $\langle G_C = (V, E), k\rangle$ be an instance of Clique. Given $\langle G_C = (V, E), k\rangle$, we are going to build a graph $G_I$ to create an input $\langle G_I, k\rangle$ of Independent-Set as follows: $G_I$ is defined with the set $V$ and a set $\overline{E}$, where $\overline{E} = \{(u, v) | (u, v) \notin E\}$, that is, $G_I = \overline{G_C}$.
   Example:

   Now, we have to show that $G_C$ **has a clique of size** $k \Leftrightarrow G_I$ **has an independent set of size** $k$. This can be easily proved by the definitions of Clique and Independent-Set, and the property of the complement of graph. Let us first prove ($\Rightarrow$) direction. Since the set of vertices in Clique (black vertices in the example) is the set of $k$ vertices that are connected each other, by ***removing*** the edges connecting them we can get $k$ independent vertices. And connecting the $k$ vertices to remaining $(n - k)$ vertices (white vertices) using complement edges, we can make the remaining vertices to be connected to the newly generated independent vertices. This forms an Independent set of size $k$. This ($\Leftarrow$) direction can be proved easily following similar argument as ($\Rightarrow$) direction.

By 1 and 2, and we know that Clique is NP-Complete, it has been proved that Independent-Set is NP-Complete.

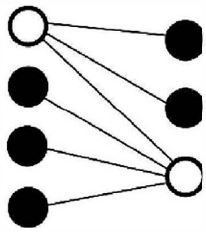
(b) We start from $k = n$ downto 1, give the query to the "black-box", "Is there an independent set $V' \subseteq V$ of size $k$?" If the "black-box" return "Yes", then we stop the algorithm, and the value $k$ represents the size of the independent set of $G$.

Now, we need to find the independent set. Let us call a vertex in the independent set 'black vertex', and a vertex not in the set 'white vertex'. We are going to exam each vertex. In the 1st exam, pick one vertex $v$ and remove the vertex and its incident edges from $G$. Then we will get a new subgraph $G_1$ of $G$, and call the "black-box" subroutine with the new subgraph $G_1$. If the subroutine returns "Yes" and the size of the independent set of $G_1$ is still $k$, then this means that the removed vertex $v$ was white vertex. In this case, we will start new exam with a new vertex. Otherwise, if the size of independent set of $G_1$ is smaller than $k$, then this means that the removed vertex $v$ was black vertex. In this case, we put the black vertex back into the subgraph $G_1$, because it was an element of the independent set.

We repeat this until we exam all vertices. Then, only white vertices will be deleted, and only black vertices will be left. These black vertices represent the independent set of graph $G$.

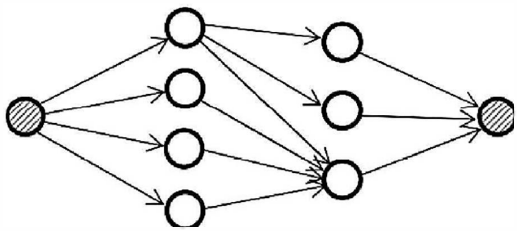
(c) If each vertex of a graph has degree 2, then the graph forms a cycle. So, we simply pick each alternate vertex on a cycle until the size of independent set becomes $\lfloor n/2 \rfloor$, where $n$ is the number of vertices in the graph. This algorithm runs in $O(n)$.

(d) Simply taking one side that has the larger number of vertices does not guarantee the maximum independent set. See the following example.



Now, let us use the result of Section 26.3. In page 735, there is a Corollary 26.11: *The cardinality of a maximum matching $M$ in a bipartite graph $G$ equals the value of a maximum flow $f$ in its corresponding flow network $G'$.*

Given a graph $G = L \cup R$ as a bipartite graph, where $L \cap R = \emptyset$, let us construct a new graph $G'$ adding a new vertex $s$ connected to each vertex $l_i \in L$, and another new vertex $t$ connected to each vertex $r_j \in R$. The capacity of each edge $(s, l_i)$ for all $i$, is assigned 1, and the capacity of each edge $(r_j, t)$ for all $j$, is also assigned 1. Assign $\infty$ capacity to the original edges in $G$. All the edges have directions from $s$ to $t$.



Then, we can find a maximum matching $M$ using the Ford-Fulkerson algorithm. We pick the vertices from $\{L \cup R\}$ that are not incident to the edges in $M$, mark those as 'black'. When we pick the 'black' vertices, we mark their incident vertices in $G$ as 'white'. These picked 'black' vertices are obviously independent. And, for each edge $(l, r) \in M$, we pick a vertex $l$ (or $r$) and mark it as 'black' if the vertex $r$ (or $l$) is currently marked as 'white'. Newly marked 'black' vertices must be independent to the previously marked 'black' vertices, because there is a 'white' vertex between them.

The running time for finding the maximum matching takes $O(VE)$. Finding the independent set takes $O(E) + O(E)$. Therefore, the total running time takes $O(VE) + O(E)$.

## Problem #3 (#34.4 – 6)

$A$: the algorithm that decides formula satisfiability
$\phi$: a given boolean formula with $n$ variables $x_1, x_2, ..., x_n$

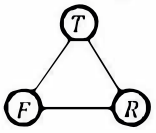
```
1.  Run A on φ.
2.  if A returns "No", then print "No assignments exist."
3.  else
4.      φ_0 ← φ
5.      for i = 1 to n
6.          φ' ← φ_{i-1} ∧ x_i
7.          Run A on φ'.
8.          if A returns "Yes", then φ_i ← φ' and x_i ← 1
9.          else if A returns "No", then φ_i ← φ_{i=1} ∧ ¬x_i and x_i ← 0
10. return the assignment ⟨x_1, x_2, ..., x_n⟩
```

## Problem #4 3-coloring

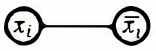
(1) 3-coloring is in NP because a coloring can be verified in polynomial time.

(2) 3-CNF-SAT $\leq_p$ 3-coloring
For the reduction, we will use following gadgets.

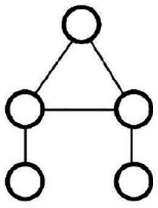
1. Palette gadget in which vertices are labeled with T, F, and R



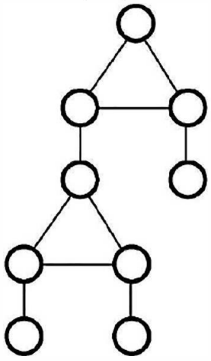

2. Variable gadget




3. OR-gadget



two OR-gadgets merged



Let $\phi = c_1 \wedge c_2 \wedge ... \wedge c_l$ be a 3CNF over variables $x_1, x_2, ..., x_m$, where each $c_i$ represents the clause. Then, we need to build a graph $G_\phi$. The graph $G_\phi$ consists of $2m + 6l + 3$ vertices; 2 vertices for each variable, 6 vertices for each clause, and 3 extra vertices.

E.g. (Fig. in next page): $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$, and $x_1$ =TRUE, $x_2$ =FALSE, $x_3$ =FALSE

Now, we need to show that **$\phi$ is satisfiable $\Leftrightarrow$ the graph $G_\phi$ is 3-colorable**.

($\Rightarrow$): Let us say that the three colors are T, F, and R. Let us color the palette with its labels. For each variable, (a) if the variable is TRUE in a satisfying assignment, then we color the vertex $x_i$ with T and the vertex $\overline{x_i}$ with F, (b) otherwise, if the variable is FALSE in the satisfying assignment, then we color the vertex $x_i$ with F and the vertex $\overline{x_i}$ with T. Here we know that each clause has one TRUE literal in the assignment, so we can color the vertices of OR-gadgets of that clause so that the vertex connected to the F vertex in the palette gadget is not colored F. Therefore, we can get a 3-colored graph.

($\Leftarrow$): We now start with a 3-colorable graph. Given a 3-colored graph, we can obtain a satisfying assignment by taking colors assigned to $x_i$ vertices in each variable gadget. Observe that neither vertex of the variable gadget can be colored R, because all variable vertices are connected to the R vertex in the palette gadget. Furthermore, if both of the bottom vertices of an OR gadget are colored F, the top vertex must be colored F, and hence, each clause contain a TRUE literal. Otherwise, the three bottom vertices that were merged with the variable vertices would be colored F, and then both top vertices would be colored F, but one of the top vertices is connected to the F vertex in the palette gadget.

By (1) and (2), it has been proved that 3-coloring is NP-Complete.