Anar Rzayev Introduction to Algorithms ID: 20190788

Homework #6

1) a) **True** To show that CLIQUE $\in NP$, for a given graph $G=(V,E)$, we use the set $V' \subseteq V$ of vertices in the clique as a certificate for G. We can check whether V' is a clique in polynomial time by checking whether, for each pair $u,v \in V'$, the edge (u,v) belongs to E. Using the fact that 3-CNF-SAT problem is a NP-complete problem, and given condition that 3-CNF-SAT $\in P$ (alternatively, it's polynomial-time solvable) then, from Theorem 34.4 in CLRS $\Rightarrow$ **P=NP** becomes true. Since CLIQUE $\in NP$ was valid, as proved at the begining, then **CLIQUE $\in P$** $\checkmark$ ■

b) **False** Let $L_1 = L_2 = \emptyset$. Surely, $\emptyset \leq_p \emptyset$ by a reduction that is e.g. the identity function but $\emptyset$ can not be NP-complete (because none of the languages in NP, except for $\emptyset$ itself, are reducible to $\emptyset$) Since $P \subseteq NP$, it could be that L_1 and L_2 are in P. Then, both are equally hard but neither is NP-complete (since $P \neq NP$ by assumption) $\checkmark$ ■

c) True If $L_2 \in P$, then considering that $L_2 \in NP$-complete and Theorem 34.4 from CLRS $\Rightarrow$ L_2 is a NP-complete problem which is polynomial-time solvable, then $P = NP$ and since $L_1 \in NP$-complete, it also means $L_1 \in NP$ or equivalently, $L_1 \in P \Rightarrow$ ☒ Thus, $L_2 \notin P$ ✓ ■ (+)

2) a) We first show that $CSP \in NP$. Suppose, we are given a natural number k and m strings $s_1, s_2, \ldots, s_m$ where $s_i \in \{0,1\}^n$ for each $i = 1, 2, \ldots, m$. The certificate we choose is the string $t \in \{0,1\}^n$ itself. For each j, the algorithm affirms that $d(t, s_j) \leq k$ by going through consequentially each of n digits of t and s_j (respectively), and calculates # of cases when x^{th} ($x = 1, 2, \ldots, n$) digit of t and s_j were different. Finding (when x^{th} digits were different) (summing up the required cases) the value for $d(t, s_j)$; algorithm checks whether it's $\leq k$ or not. For each $j = 1, 2, \ldots, m \Rightarrow$ we perform sequential comparison of n digits to compare $d(t, s_j)$ and k, thus giving us a total of $O(mn)$ time for verification. Since $m = O(n^c)$ for some constant c, it's easily seen that we can verify the certificate in $m\,O(n) = O(n^{c+1})$ time, (for some c) or equivalently, in polynomial time with respect to given natural number n

Hence, $CSP \in NP$ ☒ ✓ ■

6) If ϕ has a satisfying assignment, then take the satisfying assignment for $(x_1, x_2, \dots, x_n)$ where 1 indicates as True, and 0 indicates as False. Concretely, for each $i = 1, 2, \dots, n$ of satisfying assignment $(x_1, x_2, \dots, x_n)$ in ϕ,

if $x_i = \text{True} \Rightarrow b_i = 1$
$x_i = \text{False} \Rightarrow b_i = 0$

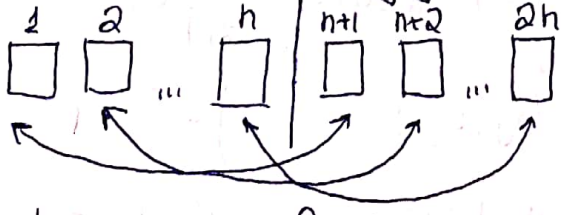
Thus, we will create $b_1, b_2, \dots, b_n$, consisting of digits 1 and 0

Then, construct new variables $c_1, c_2, \dots, c_n$ for which

if $b_j = 1 \Rightarrow c_j = 0$
$b_j = 0 \Rightarrow c_j = 1$

Getting the numbers $c_1, c_2, \dots, c_n$, they will also be consisting of $\{0, 1\}$

Now, construct the string $t = b_1 b_2 \dots b_n c_1 c_2 \dots c_n$, and we will prove that $d(t, s_i) \leq n+1$ for all $i = 1, \dots, 6n+m$ where $t \in \{0,1\}^{2n} \Rightarrow$ Before starting, we have to define some terminology for the explanation to be easily understood:



For each $1 \leq j \leq \frac{2n}{2}$, we divide our string into 2 parts (length $= n$) where j and $j+n$ are called "neighbors". For instance, $(1, n+1), (2, n+2), \dots, (n, 2n)$ are neighbors in a string of length $= 2n$. Note that, in the string t, the neighbors (b_j, c_j) are never equal, i.e. If one of them is zero, then the other one should be equal to $1 \Rightarrow$ Now, let's divide into some parts

For $i=1,\dots,2n$, according to definition, we can observe that

$g_i = \underbrace{1\,1\dots1}_{i-1}\,0\,\underbrace{1\,1\dots1}_{2n-i}$ or $g_i = 1\,1\dots1\,0\,1\dots \mid 1\,1\dots1\,1\,1\dots$ for $i\le n$

$g_i = 1\,1\dots1\,1\dots \mid 1\,1\dots0\,1\dots$ for $i\ge n+1$

In other words, the neighbors for g_i $(i=1,2,\dots,2n)$ will be $(1,1),(1,1),\dots,(0,1),(1,1),\dots,(1,1)$ where i^{th} place indicates the occurrence of "0" in either part of g_i

i^{th} place ↑ or it can be $(1,0)$

1^{st} neighbor → $(1,1)$
⋮
i^{th} neighbor → $(0,1)$ or $(1,0)$
⋮
n^{th} neighbor → $(1,1)$

it's easily noticable that except i^{th} neighbor, all the remaining tuples have identical numbers, whereas our string t would have different values

If 1^{st} neighbor → (b_1,c_1)
⋮ in the string t
n^{th} neighbor → (b_n,c_n)

since b_j and c_j are different, $d(b_j,1)+d(c_j,1)=1$ would be valid. $(c_j\neq i)$

Because $b_j=1 \Rightarrow c_j=0$ and $d((b_j,c_j),(1,1))=1$ or $b_j=0\Rightarrow c_j=1$ and $d((b_j,c_j),(1,1))=1$, $d(b_j,1)+d(c_j,1)=d(0,1)=1$. Thus, except i^{th} neighbor, all the $(n-1)$ neighbors will yield $(n-1)$ # of different bit pairs. (except $j\neq i$, $d(b_jc_j,11)=d(b_j,1)+d(c_j,1)=1$)

For the i^{th} neighbor, since $(1,0)$ and $(0,1)$ are opposite digits

It follows the idea with the fact that b_i and c_i are also in opposite digits. Meaning that, $d(b_i c_i, 01) \leq 2$, since $(b_i = 1, c_i = 0)$ can be true, or just $b_i = 0, c_i = 1$. Certainly, it can also be zero (depending on where 0 was placed). But for the upper bound $\Rightarrow d(b_i, 1) + d(c_i, 0) \leq 2$ and $d(b_i, 0) + d(c_i, 1) \leq 2 \Rightarrow$ This gives us 2 # of different bit pairs

In total, this yields that $d(t, s_i) \leq (n-1) + 2 = n+1$ for $i = 1, 2, \ldots, 2n$. In the similar way, from the definition neighbors for s_i $(i = 2n+1, \ldots, 4n)$ will be $(0,0), \ldots, (1,0), \ldots, (0,0) \Rightarrow$ 1st neighbor $\rightarrow (0,0)$ (with $(0,1)$)

i^{th} neighbor $\rightarrow (1,0)$ or $(0,1)$

n^{th} neighbor $\rightarrow (0,0)$

Except the i^{th}, digits are same, and considering (b_j, c_j) have different values, if we use the same strategy as in previous case $\Rightarrow d(b_j, 0) + d(c_j, 0) = 1$ as one of $\{b_j, c_j\}$ is zero $\Rightarrow$ For the i^{th}, it's possible that b_i and c_i will match exactly i^{th} neighbor or not. Therefore, we can assert that $d(b_i, 1) + d(c_i, 0)$ and $d(b_i, 0) + d(c_i, 1) \leq 2$

Combining the $(n-1)$ # of different bits from each neighbor except i, and at most 2 different bits from i^{th} one, we get that $d(t, s_i) \leq (n-1)+2 = n+1$ for $i = 2n+1, \dots, 4n$

Consider $i = 4n+1, \dots, 6n$, where each s_i is symmetric with respect to its neighbors. From the definition $b_j = c_j$ for $j = 1, 2, \dots, n$ and this means $(s_{3n+1} = \overline{s_{4n+1}}, s_{3n+2} = \overline{s_{4n+2}}, \dots, s_{4n} = \overline{s_{5n}})$ values for $x_1', x_2', \dots, x_n'$ are not important; since $s_i = \overline{s_{i-n}}$ for $i = 3n+1, \dots, 6n$, we have to only remember that each pair has equal values, from the construction of s_i in the $i = 4n+1, \dots, 5n \Rightarrow$ As we mentioned in the early cases, string t's neighbor tuples had distinct values $\Rightarrow$

1^{st} neighbor $\rightarrow (x_1', x_1')$

2^{nd} neighbor $\rightarrow (x_2', x_2')$

$\vdots$

n^{th} neighbor $\rightarrow (x_n', x_n')$

(b_1, c_1)
(b_2, c_2)
$\vdots$
(b_n, c_n)

$d((b_i c_i), (x_j', x_j')) = d(b_i, x_j') + d(c_i, x_j') = 1$, since exactly one of $\{b_i, c_i\}$ will be equal to x_j'

Following this scheme for each neighbor, $d(t, s_i)$ will output exactly n different bits and that will give $d(t, s_i) = n \leq n+1$ for each $i = 4n+1, \dots, 6n$

Now, we have to solve for the case $i = 6n+1, \dots, 6n+m$

Considering the case for $i=6n+1,\dots,6n+m$, take a string $s_i = x_1' x_2' \dots x_n' y_1' y_2' \dots y_n'$ where there are exactly 3 # of digit 1's. In fact, each clause C_{i-6n} includes 3 literals, where they are represented as variables x_j or $\neg x_j$. We took the satisfying assignment $b_1, b_2, \dots b_n$, and it's obvious that one of these assignments should make the clause C_{i-6n} to be True. In fact, we just need at least one assignment from b_j's to guarantee that clause C_{i-6n} will be True. For our s_i, there are 3 literals playing role in C_{i-6n}, and according to definition, this implies there will be exactly 3 # of digit 1's in the string s_i. When it comes to neighbors, there can be 2 possibilities for s_i:

(i)

(1,0) or (0,1)
(1,0) or (0,1)
(1,0) or (0,1)
} can be given in any order from left to right neighbor

(0,0)
(0,0)
⋮
(0,0)
} n-3 remaining neighbors

(ii)

(1,0) or (0,1) → in any order for left & right
(1,1) → indicating that x_i and $\neg x_i$ are both present

(0,0)
(0,0)
⋮
(0,0)
} n-2 remaining neighbors

These are possible neighbors for each clause presented as string s_i. Moreover, we know that our string $t = b_1 \dots b_n c_1 \dots c_n$ contained satisfying assignments, and in clause C_i-C_n, we have to have at least a literal making the clause True. Since ordering for t and s_i was done in sequential order, starting from X_1's situation, ending in X_n's $\Rightarrow$ we can be sure that there exist a pair from t such that it appears on s_i in the correct place, as it was on t, and therefore, when we compare it, these cases happen:

i) One of (1,0) or (0,1) neighbor in s_i will match exactly same with that of t's, where order was kept sequential & correct

Thus, in $d(s_i, t)$, this neighbor will have no contribution

For the $(n-3)$ neighbors of (0,0), we'll compare them with (b_i, c_i) with b_i, c_i - different $\Rightarrow d((b_i, c_i), (0,0)) = d(b_i, 0) + d(c_i, 0) = 1$ as we had in previous parts.

This'll guarantee $(n-3)$ # of different bits. Now, for remaining 2 neighbors, it's possible that correspondent pair neighbor for this comparison in string t can have totally different values (maybe they'll not be used for our 3-CNF as presented in string t) for each of these 2 neighbors. For the maximal bound, we can achieve at most

4 different bits $\begin{pmatrix} (x_i', y_i'), (x_j', y_j') \\ (\neg x_i', \neg y_i'), (\neg x_j', \neg y_j') \end{pmatrix}$ additionally from these neighbors. Henceforth, at the maximum, # of different bits $= d(s_i, t) \leq (n-3) + 4 = n+1 \Rightarrow d(s_i, t) \leq n+1$ ✓

ii) In this scenario, our matching neighbors from s_i and t can be either applied on $\begin{cases} (1,0) \text{ or } (0,1) \text{ neighbor of } s_i \\ (1,1) \text{ neighbor of } s_i \end{cases}$ The $(n-2)$ remaining neighbors of $(0,0)$ will be compared with (b_j, c_j) where b_j, c_j - different digits. As always, $d(b_j, 0) + d(c_j, 0) = 1$ since exactly one of them is 1. Overall, we obtain $(n-2)$ # of different bit pairs, and

- If $(1,1)$ of s_i was intertwined with $(0,1)$ or $(1,0)$ of string t in correct order, then # of different bit pairs will be 1. Similarly, remaining $(0,1)$ or $(1,0)$ neighbor in s_i can be completely different with that of string t (worst case, $(1,0)$ or $(0,1)$). Adding the last, we get $d(s_i, t) \leq n-2+1+2 = n+1$ ✓

- If $(1,0)$ or $(0,1)$ of s_i was intertwined with $(1,0)$ or $(0,1)$ of string t in that order, then there would be no contribution to $d()$ from here since they match. $(1,1)$ and any other (b_j, c_j) gives exactly 1 different pair of bit as exactly one

of b_j and c_j is equal to 1. In total, $d(s_i, t) \leq (n-2) + 1 = (n-1) < n+1$ ✓

Hence, in all cases, we proved that $d(t, s_i) \leq n+1$ for all $i = 6n+1, \dots, 6n+m$ ✓

Now, combining all what we did from the beginning of this part, we showed that after constructing t as outlined at the beginning, we found $d(t, s_i) \leq n+1$ for all i (considering that ϕ has a satisfying assignment) ✓ ■ ⊕

c) From the given condition, there exist $t \in \{0,1\}^{2n}$ such that $d(t, s_j) \le n+1$ for all $j = 1, \dots, 2n, 2n+1, \dots, 4n, \dots, 6n+m \Rightarrow$ If all digits of t were equal to zero, then $d(t, s_1) = d(\underbrace{00\dots0}_{2n}, 0\underbrace{11\dots1}_{2n-1}) = 2n-1 \le n+1 \Rightarrow n \le 2$. For now, assume that $n \ge 3$ is true (we'll consider the "$n \le 2$" part later)

So, if all digits of t were equal to zero $\Rightarrow 2 \ge n \ge 3$ (※)

Hence, there exist a digit with 1 on the string t. Similarly if all digits of t were equal to 1, then $d(t, s_{2n+1}) = d(\underbrace{11\dots1}_{2n}, 1\underbrace{00\dots0}_{2n-1}) = 2n-1 \le n+1 \Rightarrow n \le 2$ [※] Hence, there exist a digit with 0 on the string t.

Assume digit 1 appears on the i^{th} place, and digit 0 appears on the j^{th} place of string t $(1 \le i, j \le 2n)$

$t = \square\square\dots\boxed{1}\dots\boxed{0}\dots$ and $d(t, s_j) = d(t, 1\dots0\dots1) \le n+1$

$t = x_1' x_2' \dots x_{2n}'$ and $s_j = \underbrace{11\dots1}_{j-1}\underbrace{0}_{=}\underbrace{1\dots1}_{2n-j} \Rightarrow t = x_1' \dots x_{j-1}' \underbrace{0}_{=} x_{j+1}' \dots x_{2n}'$

j^{th} places for s_j and t are identical, whereas remaining numbers in s_j are equal to 1 $\Rightarrow$ from the definition, $d(t, s_j)$ indicates different # of bits, and j^{th} places will not contribute to $d(t, s_j)$ since they are identical

From the definition, $S_{2n+j} = \underbrace{00\ldots0}_{j-1}\underline{1}\underbrace{00\ldots0}_{2n-j}$, where j^{th} place has digit $\underline{1}$, and for string $t \Rightarrow$ $t = x'_1 \ldots x'_{j-1}\,\underline{0}\,x'_{j+1}\ldots x'_{2n}$ where j^{th} place shows digit $0 \Rightarrow$ since $d(S_{2n+j}, t) \le n+\underline{1}$ and this indicates different # of bits in S_{2n+j} & $t \Rightarrow$ this will be $d\left(\underbrace{00\ldots0}_{j-1}\,\underbrace{00\ldots0}_{2n-j}, x'_1 \ldots x'_{j-1}\,x'_{j+1}\ldots x'_{2n}\right) + \underline{1} \le$ $\le n+1$, where latter just indicates how many digits $\underline{1}$ are there among $x'_1, \ldots, x'_{j-1}, x'_{j+1}, \ldots, x'_{2n}$. Alternatively, we don't have to count matching zeros since they do not contribute to "distance" function, only $\underline{1}$'s contribute for the $d(00\ldots0, x'_1 \ldots x'_{2n}) \Rightarrow d\left(\underbrace{00\ldots0}_{2n-1}, x'_1 \ldots x'_{j-1}\,x'_{j+1}, \ldots, x'_{2n}\right)$ $=$ # of digit $\underline{1}$ among $\{x'_1 \ldots, x'_{j-1}, x'_{j+1}, \ldots, x'_{2n}\} =$

$=$ # of digit $\underline{1}$ in $x'_1 \ldots x'_{j-1}\,0\,x'_{j+1}\ldots x'_{2n} =$ # of digit $\underline{1}$ in $t \le$ $\le n \Rightarrow$ **# of digit $\underline{1}$ in string $t \le n$** (*) Likewise, consider $S_i = \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{i-1}\,0\,\underbrace{\underline{1}\ldots\underline{1}}_{2n-i}$, where i^{th} place has digit 0, and for string $t \Rightarrow$ digit $\underline{1}$ appears on the i^{th} place for $t \Rightarrow$ $t = x'_1 \ldots x'_{i-1}\,\underline{1}\,x'_{i+1}\ldots x'_{2n}$; we know $d(t, S_i) \le n+\underline{1}$, and similarly $d\left(\underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{i-1}\,\underbrace{\underline{1}\ldots\underline{1}}_{2n-i}\right) \Rightarrow d\left(\underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{i-1}\,\underbrace{\underline{1}\ldots\underline{1}}_{2n-i}, x'_1 \ldots x'_{i-1}\,x'_{i+1}\ldots x'_{2n}\right) + 1 \le n+\underline{1}$

Since digit $\underline{1}$ has no contribution for $d(\underline{1}\,\underline{1}\ldots\underline{1}, x_1' \ldots x_{2n}')$ among the $\{x_1', \ldots, x_{i-1}', x_{i+1}', \ldots, x_{2n}'\}$, we no longer need how many digit $\underline{1}$'s are there among $\{x_1', \ldots, x_{i-1}', x_{i+1}', \ldots, x_{2n}'\}$ to determine $d(t, s_i) \Rightarrow$ we have to know how many digits 0 are there, so that it can contribute to $d(t, s_i) \Rightarrow d(\underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{2n-1}, x_1' \ldots x_{i-1}' \; x_{i+1}' \ldots x_{2n}')$ is equal to $\Rightarrow$ how many digits 0 are there among $\{x_1', \ldots, x_{i-1}', x_{i+1}', \ldots, x_{2n}'\} \Rightarrow d(t, s_i) = d(t, \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{i-1} 0 \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{2n-i}) =$

$$= d\left(x_1' \ldots x_{i-1}' \underline{1}\, x_{i+1}' \ldots x_{2n}', \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{i-1}\, 0 \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{2n-i}\right) =$$

$$= \underline{1} + d\left(x_1' \ldots x_{i-1}' x_{i+1}' \ldots x_{2n}', \underbrace{\underline{1}\,\underline{1}\ldots\underline{1}}_{2n-1}\right) = \underline{1} + \#\text{ of digit 0 's among } \{x_1', \ldots, x_{i-1}', x_{i+1}', \ldots, x_{2n}'\}$$

$= \underline{1} +$ # of digit 0 in the $x_1' \ldots x_{i-1}' \underline{1}\, x_{i+1}' \ldots x_{2n}' =$

$= \underline{1} +$ # of digits 0 in the string $t \leq n+\underline{1} \Rightarrow$ # of digit 0 in string $t \leq n$

Since string t is composed of digits 0 and $\underline{1} \Rightarrow$ (# of digit 0 in string t) + (# of digit 1 in string t) = (# of digits in string t) $= 2n$, and if (# of digit 0 in string t) is not equal to $n \Rightarrow$ (# of digit 0 in string t) $\leq n-\underline{1}$, or $2n = ($# of digit 0$) + ($# of digit $\underline{1}$) \leq (n-1) + n = 2n-\underline{1} < 2n$ ⊗

Hence, our assumption was wrong ⇒ # of digit 0 in string $t = n$ ✗ ✗ ✗ So,

# of digit 1 in string $t = n = 2n - n$ ✗ ✗

# of digit $0 = n$ ✗
# of digit $1 = n$ ✓

∴ (+)

Note: It was accepted that $n \geq 3$, and thus, we don't have to worry about the case $n \leq 2$ ✓

d) Assume to the sake of contrary, that there exist $1 \le j \le n$ such that $b_j = c_j$. If $\underline{b_j = c_j = 1}$, then consider the number

$s_{4n+j} = \underbrace{1\,1\,\dots\,1}_{j-1}\,0\,\underbrace{1\,1\,\dots\,1}_{n-j} \,\big|\, \underbrace{1\,1\,\dots\,1}_{j-1}\,0\,\underbrace{1\,1\,\dots\,1}_{n-j}$ from the given definition

We have that $t = b_1 \dots b_{j-1}\,1\,b_{j+1} \dots b_n \,|\, c_1 \dots c_{j-1}\,1\,c_{j+1} \dots c_n$ with $d(t, s_{4n+j}) \le n+1$. Since the underlined digits will contribute to $d(t, s_{4n+j})$ by 2, we can remove them and get

$d\left(b_1 \dots b_{j-1}\, b_{j+1} \dots b_n \,|\, c_1 \dots c_{j-1}\, c_{j+1} \dots c_n,\ \underbrace{1\,1\,\dots\,1}_{n-1}\ \underbrace{1\,1\,\dots\,1}_{n-1}\right) + 2 \le$

$\le n+1$

As we can see, $d\left(b_1 \dots b_{j-1}\, b_{j+1} \dots b_n\, c_1 \dots c_{j-1}\, c_{j+1} \dots c_n,\ \underbrace{1\,1\,\dots\,1}_{2n-2}\right) \le$

$\le n-1$, and from previous part, we found that t has exactly n 1s and n 0s. Since we counted 1's twice at the begining, $t = b_1 \dots b_{j-1}\,1\,b_{j+1} \dots b_n\, c_1 \dots c_{j-1}\,1\,c_{j+1} \dots c_n$ has n # of 1's and n # of 0's

It means there are $(n-2)$ # of 1's and n # of 0's among $\{b_1, \dots, b_{j-1}, b_{j+1}, \dots, b_n, c_1, \dots, c_{j-1}, c_{j+1}, \dots, c_n\}$

Because each zero among these numbers will contribute to $d\left(b_1 \dots b_n\, c_1 \dots c_n,\ \underbrace{1\,1\,\dots\,1}_{2n-2}\right)$ and none of digit 1's has anything to do with comparison of $\underbrace{1\,1\,\dots\,1}_{2n-2}$ (because we need for $d(\dots)$ distinct digits) we have to put # of digit 0's among these numbers as a value

$n-1 \geq d(b_1 \dots b_{j-1} b_{j+1} \dots b_n c_1 \dots c_{j-1} c_{j+1} \dots c_n, \underbrace{1\,1 \dots 1}_{2n-2}) =$

$=$ # of digit 0's among $\{b_1, \dots, b_{j-1}, b_{j+1}, \dots, b_n, c_1, \dots, c_{j-1}, c_{j+1}, \dots, c_n\}$

$=$ # of digit 0's in the string $b_1 \dots b_{j-1} \underline{\underline{1}} b_{j+1} \dots b_n c_1 \dots c_{j-1} \underline{\underline{1}} c_{j+1} \dots c_n$

$=$ # of digit 0's in the string $t = n > n-1$ ※ Hence, it was not true assumption; $b_j = c_j = 0$. However, we'll use mostly same approach to yield a contradiction as before

Consider the number $s_{5n+j} = \underbrace{0\,0 \dots 0}_{j-1} \underline{\underline{1}} \underbrace{0 \dots 0}_{n-j} \,|\, \underbrace{0\,0 \dots 0}_{j-1} \underline{\underline{1}} \underbrace{0 \dots 0}_{n-j}$ from given definition, where $t = b_1 \dots b_{j-1} \underline{0} b_{j+1} \dots b_n c_1 \dots c_{j-1} \underline{0} c_{j+1} \dots c_n$

From previous part, we know that t has exactly n # of $\underline{1}$'s and n # of 0's, thus we know $d(s_{5n+j}, t) \leq n+1$

there are exactly n # of digit $\underline{1}$'s among $\{b_1, \dots, b_{j-1}, b_{j+1}, \dots, b_n, c_1, \dots, c_{j-1}, c_{j+1}, \dots, c_n\}$

and there are already 2 pairs (j^{th} and $(n+j)^{th}$ pairs) which are distinct $\Rightarrow n+1 \geq 2 + d\left(\underbrace{0 \dots 0}_{j-1} \underbrace{0 \dots 0}_{n-j} \underbrace{0 \dots 0}_{j-1} \underbrace{0 \dots 0}_{n-j}, b_1 \dots b_{j-1} b_{j+1} \dots b_n c_1 \dots c_{j-1} c_{j+1} \dots c_n\right)$

or $n-1 \geq d\left(\underbrace{0\,0 \dots 0}_{2n-2}, b_1 \dots b_{j-1} b_{j+1} \dots b_n c_1 \dots c_{j-1} c_{j+1} \dots c_n\right)$. As we observe, digit $\underline{1}$ among $\{b_1, \dots, b_n, c_1, \dots, c_n\}$ will be the main and only contributor, because digit 0's appearence among these numbers do not affect $d()$ value, as we are evaluating # of different bits

$n-1 \gtrdot d\left(\underbrace{00\ldots0}_{2n-2}, b_1 \ldots b_{j-1} b_{j+1} \ldots b_n c_1 \ldots c_{j-1} c_{j+1} \ldots c_n\right) =$

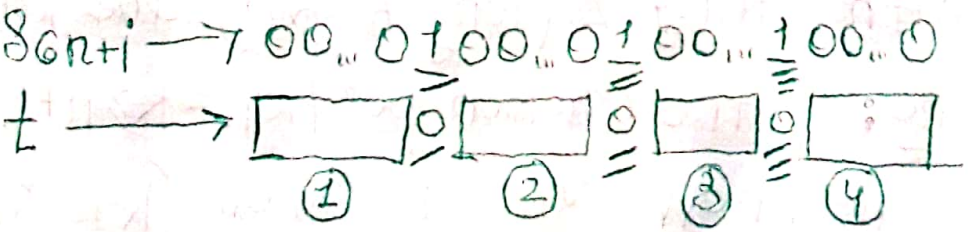
$=$ # of digit 1's among $\{b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_n, c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n\}$ $=$ # of digit 1's on the string $b_1 \ldots b_{j-1} \underline{0} b_{j+1} \ldots b_n \; c_1 \ldots c_{j-1} \underline{0} \; c_{j+1} \ldots c_n$

$=$ # of digit 1's on the string $t = n \gtrdot n-1$ ⊗ Therefore, the initial assumption was wrong $\Rightarrow$ $b_j \neq c_j$ for each $1 \leq j \leq n$

e) Using the previous t constructed in former parts, we will show that t and s_{6n+i} (for each $i = 1, \ldots, m$) have at least a digit 1 in a common position.

We have shown in part b) that since we use 3 literals per clause, and each of them displays where digit 1 is placed on the string s_{6n+i}. Assume to the sake of contrary, t and s_{6n+i} do not have common digit 1. As string s_{6n+i} has exactly 3 # of digits 1

# of digits 1 = 3 in the string s_{6n+i}



Bottom position of those digits 1 in the string t should be zero, otherwise there would be a common digit 1 on the same position of s_{6n+i} and t. Now, from part c), we know there are exactly n # of digits 1 and $(n-3)$ # of digits zero on the remaining places

①, ②, ③, ④ => Since no more digits 1 will appear on the positions of ①, ②, ③, ④ in the string S_{n+i}, all of these values will be zero and in order to find $d(S_{n+i}, t)$, we will compare # of digits 1 in the string t with correspondent zeros in S_{n+i} (0's in string t give no contribution for d() since they compare 0 & 0)

$d(S_{n+i}, t) = 3 + (\#\text{ of digit 1's in string } t) = 3 + n \leq n+1$, where $n+3$ came from the result in part c), and $n+1$ was given condition [X] Hence, since we also proved that if $t = b_1 \dots b_n c_1 \dots c_n$, then $b_j \neq c_j$ for all $1 \leq j \leq n$ (part d)),

[X] t and S_{n+i} (for $i = 1, \dots, m$) have at least a digit 1 in a common position

we can implement this idea to formulate why ϕ has a satisfying assignment. For each S_{n+i} ($i = 1, \dots, m$), we take the common digit 1 with string t (length $= 2n$), and look up to the following: if the common position $= k \geq n+1$, using the part d), we know value at position $(k-n)$ will be zero, or $x_{k-n} = \text{False}$. Otherwise, common position $= k \leq n$, then we just assign $x_k = \text{True}$. Pictographically I can explain why this works

As $t = b_1 \ldots b_n c_1 \ldots c_n$ where $b_k \neq c_k$ for each $1 \leq k \leq n$, we showed that there exist a common positioned digit 1 with each term s_{cnt+i} $(i = 1, \ldots, m)$, or alternatively, it means there exist an assignment to each clause C_i so that each of them will become true to make ϕ a satisfying assignment. The key role here is to assign values so that $x_i = \text{True}$ and $\neg x_i = \text{True}$ will not become feasible. Since we proved part d), it will allow us to change the True/False value of an assignment in each clause such that collision of such terms will not happen, as we go through each clause C_j $(j = 1, \ldots, m)$ $(x_i = \text{True}, x_i = \text{False})$

- For each string $s = b_1 \ldots b_n c_1 \ldots c_n$, b_j corresponds to x_j, and c_j corresponds to $\neg x_j$ (*)

Using this rule, we'll look up to which position digit 1 was assigned: - if $c_i = 1$, then let $x_i = \text{False}$
if $b_i = 1$, then let $x_i = \text{True}$

Because of $b_k \neq c_k$, and common positioned digit 1, this will allow us to iteratively assign x_j values to T/F, at the same time keeping track of each clause to be satisfiable (True) in the sense that no collision will happen ⇒ This will make ϕ satisfiable (True)