References

- Mini HW 10(handwritten)
- https://soptq.me/2020/06/26/3color-npc/
- Discussions with other B12.

Problem 1 - 2 SAT, or not 2 SAT, That Is the Question

(a) 2-SAT is in P

Solution.

Consider a graph G mentioned in the Hint section.

We can provide a proof and design our algorithm:

- All clauses can be satisfied. $\implies$ For all variables x, either x can't reach $\sim x$ or $\sim x$ can't reach x.

  Proof. Assume that both x and $\sim x$ can reach each other.
  Since for each clause $x \vee y$ the directed edges $(\sim x, y)$ and $(\sim y, x)$ represent that $\sim x \implies y$ and $\sim y \implies x$ respectively.
  Therefore, once both x can reach $\sim x$ and $\sim x$ can reach x then it means that $x \implies \sim x$ and $x \implies \sim x$, which leads to a contradiction because x can only be either 0 or 1. □

- For all variables x, either x can't reach $\sim x$ or $\sim x$ can't reach x. $\implies$ All clauses can be satisfied.

  Proof. Consider the following algorithm:

  - Find the SCC on the graph G, then build a new graph G', which is a DAG, with the SCCs.
  - Do the topological sort on graph G'.
  - Assign the variables with a later order literal, that is, if $ord[x] < ord[\sim x]$, then choose $\sim x$.

  The variables that we've assigned won't have any contradiction since we have already ensured that x and $\sim x$ are in different SCCs (they are not mutually reachable).
  The only case we need to consider is that $x \implies y$ and $\sim x \implies \sim y$ (W.L.O.G., assume that x is positive) so when we choose y, we must choose x or reversely.
  But that isn't a problem because if $ord[y] > ord[\sim y]$, then $ord[x] > ord[\sim x]$.
  Since the way we add the edges is symmetrical, that is, if $x \implies y$ then $\sim y \implies \sim x$. Therefore, x, y will be in a same component and $\sim x, \sim y$ will be in a same component that solve the case. □

Then, we have proven that:
All clauses can be satisfied. $\iff$ For all variables x, either x can't reach $\sim x$ or $\sim x$ can't reach x.
Hence, we can use the Kosaraju-Sharir algorithm to find all SCC, and we can check if each x and $\sim x$ are in different SCCs. If yes, then $\phi \in$ 2-SAT. If no, then $\phi \notin$ 2-SAT.
And the time complexity to find SCCs is $O(V + E)$, so that 2-SAT is in P.

□

(b) Conditions on MAX-2-SAT

(b-1) *Solution.*

Proof. To prove MAX-f-CondSAT is NP-hard. We can construct a reduction g from MAX-2-SAT, which is NP-complete, as follows:

- Just write down the original formula Φ in Φ'. We can do that in $O(N^2)$.
- Add another N^{1000} variables in Φ' with clauses $(b_i \vee \sim b_i)$, which will always be satisfied, so $N' = N + N^{1000}, k' = k + N^{1000}$. It's trivial that for every pair of variable (a_i, a_j), there is $|i-j| < N < f(N')$. We can do that in $O(N^{1000})$.

So we have the reduction $g(\Phi, k) = (\Phi', k')$, which can be done in polynomial time.
To prove its correctness, it's so trivial that the variables we added don't matter at all. Therefore, it's almost the same problem which leads to the correctness. □

Proof. To prove MAX-f-CondSAT is NP. We can just plug in all the truth values of the variables and verify the count, which is in polynomial time. □

Hence, MAX-f-CondSAT is NP-complete.

□

(b-2) *Solution.*
Consider $dp[k][m]$ to be the maximum number of satisfied clauses only the first k variables with the last $\log N$ bitmask m, which represents the values of variables from $\max\{0, k - \log N\}$ to k.

- The base case:
$$dp[0][0] = 0$$
- To simplify the computation, we can define a function $h(k, v, m)$, which means the increase of the number of satisfied clauses if $a_k = v$ and the last $\log N$ variables are given by the bitmask m:
$$h(k, v, m) = \sum_i [\text{ if clause } c_i \text{ will be satisfied right after setting } a_k = v]$$
- To calculate the dp values:
$$dp[i][j] = dp[i-1][j >> 1] + \max \begin{Bmatrix} h(i, j >> (\log N - 1), (j << 1) \& ((1 << \log N) - 1)), \\ h(i, j >> (\log N - 1), (j << 1) \& ((1 << \log N) - 1) + 1) \end{Bmatrix}$$

Since we have the number of states to be $N \times 2^{\log N} \in O(N^2)$, and the time complexity to compute $h(k, a, m)$ to be $O(N^2)$ (the number of clauses).
Therefore, the time complexity will be:
$$O(N^2) \times O(N^2) \in O(N^4)$$
Hence, MAX-f-CondSAT is P.

□

(c) 2-SAT with Generalized Boolen Values

(c-1) *Solution.*
We can construct a reduction f to 2-SAT, which is known P, as follows:

- Apply discretization on $[c]$, that is, store every value that appears in the formula and assign them with distinct numbers, then replace the original ones with the number we assigned. It's easy to do that in $O(N^2)$. After that, we can have a' taking value in $[2N]$, while b' is an element of $[2N]$, where each literal is of the form $a' = b'$. That is, c-GenSAT is the same as 2N-GenSAT.
- Consider constructing a new formula ϕ' in 2-SAT with $4N$ variables, which are some $\alpha_{(a'=x)}$ and $\sim \alpha = \alpha_{(a' \neq x)}$ where x is in $[2N]$.

- For each variable means not equal in ϕ', i.e. $\alpha_{(a'_i \neq x)}$, we add $2N-1$ clauses $\alpha_{(a'_i \neq x)} \vee \alpha_{(a'_i \neq y)}$ for all $x \neq y$. We can do that in total $O(N^3)$.
- For each clause $(a'_i = b'_i \vee a'_j = b'_j)$, we will add $(\alpha_{(a'_i \neq b'_i)} \vee \alpha_{(a'_j = b'_j)}$ and $(\alpha_{(a'_i = b'_i)} \vee \alpha_{(a'_j \neq b'_j)}$. We can do that in $O(N^2)$.

To prove its correctness:

- *Proof.* $\phi = 1 \implies f(\phi) = 1$ It's trivial that if we have a solution for ϕ, then we can plug it in to $f(\phi)$.
  Because we've ensured that every clause holds true for ϕ, it's still true for $f(\phi)$.
  Also, we know that either $a'_i \neq x$ or $a'_i \neq y$ for $x \neq y$ is true, since a'_i can't be both x, y. □
- *Proof.* $f(\phi) = 1 \implies \phi = 1$ It's always true that either a'_i will equals to a certain value v (the other will be false in 2-SAT) or will equal to nothing.
  That's because $\alpha_{(a'_i \neq x)} \vee \alpha_{(a'_i \neq y)}$ which means $a'_i = x \implies a'_i \neq y$) for all $x \neq y$.
  - if a'_i equals to a certain value v, then we just select it as the value in ϕ. (it will satisfied c-GenSAT as well because we satisfied all the clauses in ϕ specifying it in $f(\phi)$)
  - if a'_i equals to nothing, then it just means that it doesn't matter what a'_i is. (it doesn't appear in any clause)

  □

Therefore, we can do the reduction in $O(N^3)$, which is polynomial time reduction to 2-SAT.
Hence, c-GenSAT $\leq_p$ 2-SAT, c-GenSAT is P for all positive integers c.

□

(c-2) *Solution.*
To prove c-GenSetSAT is NP-hard:

Consider a reduction f from 3-colorability problem with graph G to c-GenSetSAT ($c \geq 3$) as follows:

- For each vertex v in G, we add the clause $(\vee v \in \{1,2,3\} \vee v \in \varnothing)$ to ϕ in c-GenSetSAT in $O(V)$. (make the value of $v \in \{1,2,3\}$)
- For each edge (u,v) in G, we add the following 3 clauses to ϕ in c-GenSetSAT $\in O(E)$ (make the color of u,v distinct):
  1. $(u \in \{1,2\} \vee v \in \{1,2\})$
  2. $(u \in \{2,3\} \vee v \in \{2,3\})$
  3. $(u \in \{1,3\} \vee v \in \{1,3\})$

To prove its correctness:

- *Proof.* $G = 1 \implies f(G) = 1$ If G is 3-colorable, we can simply assign the same value for variables in $f(G)$.
  It's trivial that value of each variable will be in $\{1,2,3\}$.
  Also, the value of every pair (u,v) satisfied that $u \neq v$, and by Pigeonhole Principle they will comply regulation of the clauses for edges. □
- *Proof.* $f(G) = 1 \implies G = 1$ If $f(G)$ is satisfiable, we can simply assign the same value for the color in G.
  The reasons are that the values are limited in $\{1,2,3\}$ and every (u,v) can't share a same value because if so then it won't satisfy the clauses for edges. □

Therefore, we can do the reduction f from 3-colorability problem to c-GenSetSAT in $O(V+E)$, which is a polynomial reduction.
So, c-GenSetSAT is NP-hard.
To prove c-GenSetSAT is NP:
We can just plug in the boolean values in the formula and check if it is satisfiable, which can be done in polynomial time.
So, c-GenSetSAT is NP.

Hence, c-GenSetSAT is NP-complete.

□

Problem 2 - Nathan and Packages Arranging

Working Hard, Working Smart

(a) *Solution.*
Consider an amortized cost of \$5 for each insertion.
In every operation, we use \$1 to put the package itself in the box.
And we can save \$4 for each package, and it will be used to do four operations as follows:

- Moving the package itself into a new box when resizing.
- Making room for itself in a new box.

- Moving one of the old packages (there isn't cash on it) into a new box when resizing.
- Making room for one of the old packages in a new box.

Since the old package can be always pair with a new package, the money on each package will always be greater or equal to zero. So the money in the account will always be non-negative.
Hence, by the account method, the time complexity would be:

$$5 \times N \in O(N)$$

□

(b) *Solution.*
Consider the situation $N = S + 1$, such that the box of size S is full when Nathan's trying to put the last package.
Since he needs to create a new box with size S^2, and hence needs at least S^2 units of time.
And it's trivial that $S^2 \notin O(N)$, so this strategy can't be executed in $O(N)$.

□

(c) *Solution.*
Consider an amortized cost of $\$(2N + 1)$ for each insertion.
In every operation, we use \$1 to put the package itself in the box.
And we can save $\$(2N)$ for each package, and it will be used to do four kinds of operations as follows:

- Moving the package itself into a new box when resizing. (use \$1)
- Making room for itself in a new box. (use \$1)
- Moving $N - 1$ of the old packages (there isn't cash on it) into a new box when resizing. (use $\$N - 1$)
- Making room for $N - 1$ of the old packages in a new box. (use $\$N - 1$)

Since every N of the old packages can be always pair with a new package, the money on each package will always be greater or equal to zero. So the money in the account will always be non-negative.
Hence, by the account method, the time complexity would be:

$$(2N + 1) \times N \in O(N^2)$$

□

(d) *Solution.*
By observation, the worst case would be that the manager appears every time right after a new package arrives. (because new package's put in the smallest box, so Nathan needs to move the package more often)
So the actual cost is:

$$c_i = \begin{cases} i, & \text{if } i = 2^k + 1, \text{ where } k \in \mathbb{N}^0 \\ 1, & \text{otherwise} \end{cases}$$

Then the total cost:

$$\begin{aligned} T(N) &= \sum_{i=1}^{N} c_i \\ &= \Theta(N) + \sum_{i=1}^{\lfloor \log_2 (N-1) \rfloor} 2^i \\ &= \Theta(N) + \Theta(2N) \\ &= \Theta(N) \end{aligned}$$

□

Work Life Balance

(e) *Solution.*
Notice that such data structure works similar to k-bit counter, so we can represent the box status with a binary number T.
Consider a potential function:

$$\Phi(T) = 2 \times T.onebits + 8 \times T.value$$

where $T.onebits$ = the number of the 1-bits in T and $T.value$ = the actual value of T.
We have designed this function so that $\Phi(D_0) = 0$ (trivial).
Then, we can compute the amortized cost:

- For an arrival (assume we have to carry for k times):

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= (2k + 2) + 2 \times (-k) + 8 \times 1 \\ &= 10 \\ &\in O(1) \end{aligned}$$

- For deliver (assume we have to deliver k packages, where $k \geq D_{i-1}.value$):

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k + 2(\log_2 D_{i-1}.value - 1) - 8k \\ &\leq 0 \end{aligned}$$

Hence, the amortized cost will be

$$O(1) \times N = O(N)$$

□