# Homework #4

**Red Correction Time: 2023/12/22 21:00**

Due Time: 2023/12/26 14:20

Contact TAs: `ada-ta@csie.ntu.edu.tw`

## Instructions and Announcements

- There are **one programming problem** and **two hand-written problems**
- **Programming.** The judging system is located at `https://ada-judge.csie.ntu.edu.tw`. Please log in and submit your code for the programming problems (i.e., those containing "Programming" in the problem title) by the deadline. NO LATE SUBMISSION IS ALLOWED.
- **Hand-written.** For other problems (also known as the "hand-written problems"), you should upload your answer to **Gradescope**. For each sub-problem, please label (on Gradescope) the corresponding pages where your work shows up, **or you may get some penalty for each problem you did not label** . NO LATE SUBMISSION IS ALLOWED.
- **Collaboration policy.** Discussions with others are strongly encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (e.g., the Internet URL you consulted with or the people you discussed with, or **screenshot of the full conversation** if you are using GPT) on the **first** page of your solution to that problem and tag the corresponding page in Problem 0 on Gradescope. You may get zero points due to the lack of references.
- **Tips for programming problems.** Here are some C++ only tips. For this homework, you might want to check out sections about cin/cout, vectors, pairs, references, and range for. Reference is IMHO the most important since it rids you of C pointers. (Yeah!!!)
- **Instruction on explaining your algorithm with pseudocode.** If you decide to use pseudocode to describe your algorithm, please try to make your pseudocode human-readable (not C/C++ compilable), and your pseudocode should be less than 30 lines. Pseudocode without explanation or not human-readable may lead to a deduction of points for the corresponding problem. You can check out the textbook for some examples of easy-understanding pseudo code.

# Problem 1 - 2 SAT, or not 2 SAT, That Is the Question (Hand-Written) (30 points)

## (a) 2-SAT is in P (6 points)

A 2-CNF formula is a CNF formula where each clause has exactly 2 literals. For example:

$$\phi_1 = (a_1 \vee \sim a_2) \wedge (a_3 \vee a_1) \wedge (\sim a_1 \vee \sim a_1)$$

$$\phi_2 = (a_1 \vee a_1) \wedge (\sim a_1 \vee \sim a_1)$$

are both 2-CNF formulas.

$$\text{2-SAT (or 2-CNF-SAT)} := \{\phi | \phi \text{ is a 2-CNF formula with a satisfying assignment}\}$$

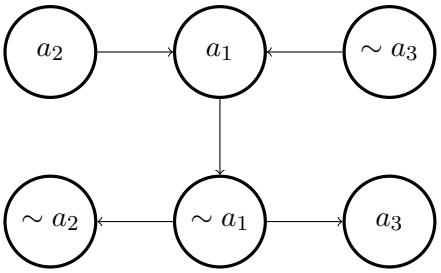
For example:
$\phi_1$ can be satisfied if $(a_1, a_2, a_3) = (0, 0, 1)$; therefore $\phi_1 \in$ 2-SAT.
$\phi_2$ does not have a satisfying assignment; therefore $\phi_2 \notin$ 2-SAT.

A 2-SAT problem is to determine whether a 2-CNF formula $\phi$ is in 2-SAT or not, and the size of this problem is defined as the number of clauses.

Prove that 2-SAT is in P.

### Hint

Let $\phi$ be a 2-CNF formula with variables $a_1, a_2, \ldots, a_n$. Consider a directed graph $G$ where $V(G) = \{a_1, a_2, \ldots, a_n, \sim a_1, \sim a_2, \ldots, \sim a_n\}$ is the set of all literals. For each clause $x \vee y$ of $\phi$, construct directed edges from $\sim x$ to $y$ and from $\sim y$ to $x$. For example, the following is the graph of $\phi_1$ (which is described above):




First prove that

$$\text{All clauses can be satisfied.}$$
$$\Updownarrow$$
$$\text{For all variables } x\text{, either } x \text{ can't reach } \sim x \text{ or } \sim x \text{ can't reach } x.$$

Then find an polynomial time complexity algorithm to check if $G$ satisfies the transformed condition, deducing that 2-SAT is in P.

## (b) Conditions on MAX-2-SAT

A MAX-2-SAT problem is to find the truth values of the variables such that the number of satisfied clauses in a given 2-CNF formula is maximized. The decision version of MAX-2-SAT problem is to

determine whether at least $k$ clauses in a 2-CNF formula can be satisfied. That is:

$$\text{MAX-2-SAT} := \{(\phi, k) | \phi \text{ is a 2-CNF formula with an assignment such that at least } k \text{ clauses can be satisfied}\}$$

It is known that MAX-2-SAT is in NP-complete, but some condition on the clauses may make the problem easier.

Let $f$ be a given single-variable function. The set $S_f$ is defined as the collection of all 2-CNF formulas whose variables can be numbered as $a_1, a_2, \ldots, a_n$ such that for all pairs of variables $(a_i, a_j)$ that are in the same clause, there is $|i - j| < f(n)$.
For example, let $f(x) := \sqrt{x}$.

$$\phi_3 = (b_1 \vee b_2) \wedge (\sim b_1 \vee b_3)$$

$$\phi_4 = (b_1 \vee b_2) \wedge (b_2 \vee b_3) \wedge (\sim b_3 \vee b_1)$$

One can number the variables in $\phi_3$ by letting $a_1 = b_2, a_2 = b_1, a_3 = b_3$, and $\phi_3$ becomes $(a_2 \vee a_1) \wedge (\sim a_2 \vee a_3)$. Since $|2 - 1| < \sqrt{3}, |2 - 3| < \sqrt{3}$, there is $\phi_3 \in S_{\sqrt{x}}$.
However, no matter how the variables in $\phi_4$ are numbered to $a_1, a_2, a_3$, there always exists a clause that contains both $a_1$ and $a_3$. Since $|1 - 3| \not< \sqrt{3}$, there is $\phi_4 \notin S_{\sqrt{x}}$.

Let's define the MAX-$f$-CondSAT problem, which is to find the truth values of the variables such that the number of satisfied clauses in a given $\phi \in S_f$ is maximized. The decision version of MAX-$f$-CondSAT problem is to determine whether at least $k$ clauses can be satisfied. That is:

$$\text{MAX-}f\text{-CondSAT} := \{(\phi, k) | \phi \in S_f \text{ has an assignment such that at least } k \text{ clauses can be satisfied}\}$$

### (b-1) (6 points)

Let $f(x) := x^{0.001}$. Prove that MAX-$f$-CondSAT is NP-complete.

#### Hint

Can you provide a reduction from a known NP-complete problem: MAX-2-SAT?

### (b-2) (6 points)

Let $f(x) := \log x$. Prove that MAX-$f$-CondSAT is P.

#### Hint

MAX-1-CondSAT can be easily done by dynamic programming in polynomial time complexity. How about MAX-2-CondSAT, MAX-3-CondSAT, ..., and so on? Can this dynamic programming method be generalized to MAX-$f$-CondSAT?

## (c) 2-SAT with Generalized Boolen Values

In a 2-SAT problem, all variables take values in $\{True, False\}$, but what if the variables take values in a set with size larger than 2? Will the problem become harder?

Let $c$ be a positive integer. Define the notation $[c] := \{1, 2, \ldots, c\}$.

## (c-1) (6 points)

The set $T_c$ is defined as the collection of all formulas $\phi$ that is the conjunction of some clauses, and each clause of $\phi$ is the disjunction of two literals, where each literal is of the form $a = b$. $a$ is a variable taking values in $[c]$, while $b$ is an element of $[c]$.
For example, let $c = 3$.

$$\phi_5 = (a_1 = 3 \vee a_2 = 2) \wedge (a_3 = 1 \vee a_2 = 3)$$

$$\phi_6 = (\sim (a_1 = 3) \vee a_2 = 2)$$

$\phi_5 \in T_c$, but $\phi_6 \notin T_c$ because $\sim (a_1 = 3)$ is not of the form $a = b$ for a variable $a$ and a constant $b$ taking values in $[c]$.

Let's define the $c$-GenSAT problem, which is to determine whether all clauses in a given $\phi \in T_c$ can be satisfied. That is:

$$c\text{-GenSAT} := \{\phi | \phi \in T_c \text{ has a satisfying assignment}\}$$

Prove that $c$-GenSAT is P for all positive integers $c$.

### Hint

Can you provide a reduction to a known P problem: 2-SAT?

## (c-2) (6 points)

The set $U_c$ is defined as the collection of all formulas $\phi$ that is the conjunction of some clauses, and each clause of $\phi$ is the disjunction of two literals, where each literal is of the form $a \in B$. $a$ is a variable taking values in $[c]$, while $B$ is a subset of $[c]$.
For example, let $c = 3$.

$$\phi_7 = (a_1 \in \{1, 2\} \vee a_2 \in \{3\}) \wedge (a_3 \in \{1, 2, 3\} \vee a_2 \in \emptyset)$$

$$\phi_8 = (a_1 \in \{3, 4\} \vee a_2 \in \{2, 3\})$$

$\phi_7 \in T_c$, but $\phi_8 \notin T_c$ because $\{3, 4\}$ is not a subset of $[c]$.

Let's define the $c$-GenSetSAT problem, which is to determine whether all clauses in a given $\phi \in U_c$ can be satisfied. That is:

$$c\text{-GenSetSAT} := \{\phi | \phi \in U_c \text{ has a satisfying assignment}\}$$

Prove that $c$-GenSetSAT is NP-complete for all integers $c \geq 3$.

### Hint

Can you provide a reduction from a known NP-complete problem: 3-colorability problem?

# Problem 2 - Nathan and Packages Arranging (Hand-Written) (25 points)

### Working Hard, Working Smart

Nathan's bubble tea buying problem had caused too much trouble for the villagers. Therefore, the head of the Small Circle Village, Arctan decides to drive him out of the village. Having no other choices, he then heads to Generous Town. In Generous Town, Nathan applied for a packages arranging job, the job involved putting packages into boxes. Simple as it may look, however, his manager demanded that he must put all packages into a single box. To make his work easier, he asked you to evaluate his strategy of arranging packages. He wanted to use an efficient way to arrange packages so that he could have his good days.

#### (a) (5 points)

Nathan designed a strategy: he starts with a box of size 1. Then, when a new package arrives, he checks if the box is full. If not, he simply spends 1 unit of time to put the package into the box. Otherwise, suppose the box has the size of $S$, he would spend $2S$ units of time to create a new box with the size of $2S$, then spend $S$ units of time to move all $S$ packages into the new box and spend 1 unit of time to put the new package into the new box. Given the strategy, please use **accounting method** to prove that it would take $O(N)$ units of time after $N$ arrivals of packages.

#### (b) (2 points)

Although the strategy above may look efficient, Nathan feels exhausted each time he creates a new box. Therefore, he decided to create a larger box each time to lessen the time of box creation. He now starts with a box of size 2, suppose that the box of size $S$ is full. Instead of making a new box with the size of $2S$ in the previous problem, he would make a box with the size of $S^2$, which costs $S^2$ units of time for him. Can this strategy process $N$ arrivals of packages in $O(N)$ units of time? If yes, analyze the time complexity of this strategy, otherwise explain why this strategy cannot be executed in $O(N)$.

#### (c) (4 points)

Given the strategy from (b), please use **accounting method** to prove that it would take $O(N^2)$ units of time after $N$ arrivals of packages.

#### (d) (7 points)

After working for a while, Nathan found out that his manager seldom appeared in the factory. Therefore, he decided to allow packages to be in different boxes and only put them into a single box when his manager showed up. His strategy is now as follow: make all box with the size of $2^k$, where $k$ are some integers $\geq 0$, before any package arrives **(you do not need to consider the time of creating any box)**. Then, when a package arrives, spend 1 unit of time to put it into the smallest non-full box. When the manager shows up, find the smallest box which can fit all the packages we currently have, and move all the packages that are not in it into it. Each move takes 1 unit of time. For example, the operations and the packages contained in each box is shown as below:

| Operation | Size-1 Box | Size-2 Box | Size-4 Box | Size-8 Box |
|---|---|---|---|---|
| package arrival | 1 | 0 | 0 | 0 |
| package arrival | 1 | 1 | 0 | 0 |
| package arrival | 1 | 2 | 0 | 0 |
| Manager appears | 0 | 0 | 3 | 0 |
| package arrival | 1 | 0 | 3 | 0 |
| package arrival | 1 | 1 | 3 | 0 |
| Manager appears | 0 | 0 | 0 | 5 |
| package arrival | 1 | 0 | 0 | 5 |
| package arrival | 1 | 1 | 0 | 5 |
| package arrival | 1 | 2 | 0 | 5 |
| package arrival | 1 | 2 | 1 | 5 |

Now, suppose the manager may show up at any moment, please use **aggregate method** to prove that it would take $O(N)$ units of time after $N$ arrivals of the package.

### Work Life Balance

Nathan always loves to play some games during his leisure time. One day, after enjoying playing a popular game suika game, he was inspired by the game and decided to apply it to his strategy.

Suppose that Nathan now has made an infinite amount of boxes with a size of $2^x$, where $x$ are some integers $\geq 0$. In addition, his manager no longer demanded to put all packages into a single box but asked him to do one more thing, picking any $k$ packages out of the box to deliver sometimes. Which $k$ items are chosen doesn't matter. However, delivering only a few packages one time would gain less profit, so the manager guaranteed that once a delivery request of $k$ packages is made, $k$ **must be greater or equal to half the number of packages currently stored in boxes.** With the inspiration from the game, he designed the following strategy:

- When a package arrives, spend 1 unit of time to put it into a new box with a size of 1.
- At any moment, if there are two boxes with the same size of $2^x$, spend 2 unit of time to put them into a box with a size of $2^{x+1}$. That is, spending 2 unit of time to "merge" two boxes of the same size into a bigger one.
- Then, when a delivery request of $k$ packages is made, Nathan would repeatedly take out two smaller boxes with size $2^{x-1}$ from a box with size $2^x$ until he could take out $k$ packages from $k$ boxes with size of 1, and take the $k$ packages out to deliver. That is, spending 2 unit of time to "split" one boxes into two boxes of the same size.
- Assume that you can only take out the content from a box of size 1, and it takes 1 unit of time.

Nathan would always spend the least amount of time performing a delivery operation and it can be proved that the status of boxes would be the same after the delivery operation if it is performed optimally. For example, the operations and their corresponding cost are shown as below:

| Operation | Total Cost | Put | Merge | Split | Take | Box status |
|---|---|---|---|---|---|---|
| package arrival | 1 | 1 | 0 | | | 1 |
| package arrival | 3 | 1 | 2 | | | 2 |
| deliver 2 packages | 4 | | | 2 | 2 | |
| package arrival | 1 | 1 | 1 | | | 1 |
| package arrival | 3 | 1 | 2 | | | 2 |
| package arrival | 1 | 1 | 0 | | | 1, 2 |
| package arrival | 5 | 1 | 4 | | | 4 |
| deliver 2 packages | 6 | | | 4 | 2 | 2 |
| package arrival | 1 | 1 | 0 | | | 1, 2 |
| package arrival | 5 | 1 | 4 | | | 4 |
| package arrival | 1 | 1 | 0 | | | 1, 4 |
| package arrival | 3 | 1 | 0 | | | 2, 4 |
| package arrival | 1 | 1 | 0 | | | 1, 2, 4 |
| package arrival | 7 | 1 | 6 | | | 8 |
| deliver 5 packages | 17 | | | 12 | 5 | 1, 2 |

Note that in the last operation, we split the box of size $8, 4, 2, 2, 4, 2$ sequentially.

### (e) (7 points)

Please use **potential method** to prove that the strategy would take $O(N)$ after $N$ arrivals of packages and several deliver operations.

#### Hint

- This data structure was inspired from $k$-bit counter, maybe you can start from the potential function from $k$-bit counter.
- Arrival operation could be $O(\log N)$, how should potential function handle it?
- The deliver operation may be troublesome, can you solve $k = 2^x$ first?
- Deliver operation could be $O(N)$, how should potential function handle it?
- What can we do with two different potential methods?
- Try some linear combinations!
- This data structure was inspired from $k$-bit counter, maybe you can start from the potential function from $k$-bit counter.

# Problem 3 - Kiwi's Multifunctional Can (15 points)

## Problem Description

*After sold out all cans produced in HW2,* ***Kiwi*** *feels bored with making normal kiwi cans. Instead of making boring cans, he decided to start up a new business, inventing cans with special features.*

**Kiwi** just invented a huge can, which is designed for storing many kiwis. Each kiwi in it has its own size. Since the can is too large, a user needs to manipulate it with a special machine. This machine supports four kinds of operations:

- `kiwi` $x$: Put a kiwi of size $x$ into the can.
- `wiki` $y$: Remove a kiwi of size $y$ from the can. Notice that there may be multiple kiwis of the same size $y$. In this case, **only one of them is removed**.
- `wiwi`: Assume that the sizes of all kiwis currently in the can are $x_1, x_2, \ldots, x_N$, find the minimum value of $\sum_{i=1}^{N} |x - x_i|$ where $x$ is an arbitrary real number.
- `kiki` $k$: Sort all kiwis in the can by their sizes in increasing order, and assume that their sizes are $x_1 \leq x_2 \leq \cdots \leq x_N$. Remove all kiwis from the can, "merge" them into $k$ kiwis, and push them back. Specifically, for $1 \leq i \leq k$, merge the $i, i+k, i+2k, \ldots$-th kiwi into a new kiwi of size $x_i + x_{i+k} + x_{i+2k} + \cdots$. More formally, for each $1 \leq i \leq k$, all kiwis with rank $j$ s.t. $j \equiv i \pmod{k}$ are merged into a new kiwi.

Unfortunately, the quality assurance engineers in **Kiwi**'s Kiwi Factory are on vacation. Please tell **Kiwi** what the machine should output for each `wiwi` operation, so that **Kiwi** can ensure the machine won't make any trouble in the product launch.

## Input

The first line contains a positive number $Q$, representing the number of operations you need to simulate.

There are $Q$ lines that follow. The $i$-th line is the instruction for the $i$-th operation, which is one of

- `kiwi` $x$
- `wiki` $y$
- `wiwi`
- `kiki` $k$

where $x, y, k$ are positive integers.

Before all these operations, the can is empty.

- $1 \leq Q \leq 5 \times 10^5$
- $1 \leq x \leq 10^9$
- $1 \leq y \leq 5 \times 10^{14}$
- $1 \leq k \leq Q$

- For a `wiwi` operation, it is guaranteed that the can is not empty.
- For a `wiki` $y$ operation, it is guaranteed that there is a kiwi of size $y$ in the can.
- For a `kiki` $k$ operation, it is guaranteed that $k$ is not greater than the number of kiwis in the can.

## Output

For each `wiwi` operation, output a line containing an integer, representing the minimum value of $\sum_{i=1}^{N}|x - x_i|$. It can be proven that the answer is an integer.

### Test Group 0 (0 %)

- Sample Input

### Test Group 1 (10 %)

- $Q \leq 1000$

### Test Group 2 (20 %)

- There is no `kiki` operation.

### Test Group 3 (15 %)

- For each `kiki` $k$ operation, if the number of kiwis in the can is $N$, it is guaranteed that $k \leq N/2$.

### Test Group 4 (55 %)

- No additional constraints

### Sample Input 1

```
13
kiwi 4
wiwi
kiwi 8
kiwi 7
wiwi
wiki 4
kiwi 6
wiwi
kiwi 3
kiki 2
wiwi
kiwi 10
wiwi
```

### Sample Output 1

```
0
4
2
4
4
```

### Sample Input 2

```
7
kiwi 1000000000
kiwi 1000000000
kiwi 1000000000
kiwi 1000000000
kiki 1
kiwi 1000000000
wiwi
```

### Sample Output 2

```
3000000000
```

### Sample Input 3

```
15
kiwi 3
wiwi
wiki 3
kiwi 2
kiwi 1
kiwi 5
kiwi 3
wiwi
kiwi 2
wiwi
wiki 5
wiwi
kiwi 2
wiki 3
kiwi 1
```

### Sample Output 3

```
0
5
5
2
```

### Hint

- Assume that you know the sizes of kiwis in the can are $x_1, x_2, \ldots, x_N$ in increasing order. What $x$ minimizes $\sum_{i=1}^{N} |x - x_i|$? (Try to draw this function with Desmos or Geogebra.)