2021 Mini Competition 2 Editorial

2021-03-27

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

M2121
Add to Cart

Subtask 1

Subtask 1 (17%): `N, A[i] ≤ 3000`

- For a customer pressing the buttons optimally, he should always press "`+X`" when adding `X` won't exceed the amount he wanted
- By that, we know that buying `A[i]` unit require a minimum of `(A[i] / X + A[i] % X)` button presses

- We can exhaust all possible values of `X` (which is [1 - A_{max}]) and calculate the total button presses required → take the minimum

Subtask 1

- We can exhaust all possible values of X (which is [1 - A_{max}]) and calculate the total button presses required → take the minimum
- Minimize `(A[1]/X + A[1]%X) + (A[2]/X + A[2]%X) + … + (A[N]/X + A[N]%X)` by choosing optimal X

Score: 17

Time Complexity: `O(`A_{max}`* N)`

Subtask 2

Subtask 2 (25%): `A[i] ≤ 3000`

> "We can exhaust all possible values of X (which is [1 - A_{max}]) and calculate the total button presses required → take the minimum"

- We try to optimize the above step, knowing that same value of `A[i]` requires same amount of button presses, we can first transform array `A` into a counting array `C[i]`, indicating the frequency of `i` in `A` is `C[i]`
- Then we could just loop through 1 to A_{max} for each X instead of looping through 1 to `N`

Subtask 2

- Then we could just loop through 1 to A_{max} for each X instead of looping through 1 to N
- Minimize `(1/X + 1%X) * C[1] + (2/X + 2%X) * C[2] + … +` $(A_{max}/X + A_{max}\%X) * C[A_{max}]$ by choosing optimal X

Score: 42

Time Complexity: $O(N + A_{max}^2)$

Full Solution

- How could we further optimize the solution?
- Try to rewrite the formula into a more useful form
- $(1/X + 1\%X) * C[1] + (2/X + 2\%X) * C[2] + \ldots + (A_{max}/X + A_{max}\%X) * C[A_{max}]$

Full Solution

- $(1/X + 1\%X) * C[1] + (2/X + 2\%X) * C[2] + \ldots + (A_{max}/X + A_{max}\%X) * C[A_{max}]$

	$(1/X + 1\%X) * C[1]$	$(2/X + 2\%X) * C[2]$	···	$((X-1)/X + (X-1)\%X) * C[X-1]$
$(X/X + X\%X) * C[X]$	$((X+1)/X + (X+1)\%X) * C[X+1]$	$((X+2)/X + (X+2)\%X) * C[X+2]$	···	$((2X-1)/X + (2X-1)\%X) * C[2X-1]$
···	···	···	···	···
$((A_{max}-1)/X + (A_{max}-1)\%X) * C[A_{max}-1]$	$(A_{max}/X + A_{max}\%X) * C[A_{max}]$			

Full Solution

- `(1/X + 1%X) * C[1] + (2/X + 2%X) * C[2] + … + (`A_{max}`/X + `A_{max}`%X) * C[`A_{max}`]`

`(0 + 0) * C[0]`	`(0 + 1) * C[1]`	`(0 + 2) * C[2]`	⋯	`(0 + X-1) * C[X-1]`
`(1 + 0) * C[X]`	`(1 + 1) * C[X+1]`	`(1 + 2) * C[X+2]`	⋯	`(1 + X-1) * C[2X-1]`
⋯	⋯	⋯	⋯	⋯
`(k + 0) * C[kX]`	`(k + 1) * C[kX+1]`	`(k + 2) * C[kX+2]`		`(k + X-1) * C[(k+1)X-1]`

*setting those C[i] out of range as C[i] = 0

Full Solution

`(0 + 0) * C[0]`	`(0 + 1) * C[1]`	`(0 + 2) * C[2]`	⋯	`(0 + X-1) * C[X-1]`
`(1 + 0) * C[X]`	`(1 + 1) * C[X+1]`	`(1 + 2) * C[X+2]`	⋯	`(1 + X-1) * C[2X-1]`
⋯	⋯	⋯	⋯	⋯
`(k + 0) * C[kX]`	`(k + 1) * C[kX+1]`	`(k + 2) * C[kX+2]`		`(k + X-1) * C[(k+1)X-1]`

- For each interval of `[iX - (i+1)X-1]` (let `a = iX`, `b = (i+1)X-1`), the number of presses contributed is

  `(i * C[a] + i * C[a+1] + i * C[a+2] + … + i * C[b]) +`

  `(0 * C[a] + 1 * C[a+1] + 2 * C[a+2] + … + (X-1) * C[b])`

Full Solution

- For each interval of `[iX - (i+1)X-1]` (let `a = iX, b = (i+1)X-1`), the number of presses contributed is

  `(i * C[a] + i * C[a+1] + i * C[a+2] + … + i * C[b]) +`

  `(0 * C[a] + 1 * C[a+1] + 2 * C[a+2] + … + (X-1) * C[b])`

- This can be calculated in O(1) by using 2 partial sum array

  `P[i] = C[0] + C[1] + … + C[i]`

  `P2[i] = 0 * C[0] + 1 * C[1] + ... + i * C[i]`

  Then, turning the above formula into

  `i * (P[b]-P[a-1]) + (P2[b]-P2[a-1]) - C[a] * (P[b]-P[a-1])`

*Handle case where a = 0 carefully

Full Solution

- So, for each X, we loop through the multiples of X, calculate the interval contribution by previous formula, and sum them all up.
- It may seem that by looping through multiples of X for each X we need $O(A_{max}^2)$ time, but actually the bound is lower because of the sum of harmonic series can be approximated by $\ln N$

  $1 + 1/2 + 1/3 + \ldots + 1/N \approx \ln N$

Score: 100

Time Complexity: $O(N + A_{max} \ln A_{max})$

Subtask 3

- Subtask 3 (22%): `N, A[i] ≤ 10^5`
- If you somehow think of all of the above (looping only the multiples, using partial sum) but not using counting array, the interval contribution of `[iX - (i+1)X-1]` (`a = iX, b = (i+1)X-1`) can be calculated alternatively
- Sort and precompute partial sum P[i] = A[1] + ... + A[i]. First use binary search (lower/upper bound) to locate `a, b` in the array (pnt_a, pnt_b), then interval contribution = $i * (pnt_b - pnt_a + 1) + P[pnt_b] - P[pnt_a - 1] - a * (pnt_b - pnt_a + 1)$

Subtask 3

- Sort and precompute partial sum P[i] = A[1] + ... + A[i]. First use binary search (lower/upper bound) to locate `a`, `b` in the array (pnt_a, pnt_b), then interval contribution = $i * (pnt_b - pnt_a + 1) + P[pnt_b] - P[pnt_a - 1] - a * (pnt_b - pnt_a + 1)$

- In other way, total presses = `Total Sum - [(a-i) * (pntb - pnta + 1)]` for every interval

Score: 64

Time Complexity: $O(N \ln N + A_{max} \ln A_{max} \ln N)$

Can be viewed as "How many presses has been reduced by pressing '+X' instead of '+1'"

Alternative Subtask 3 (Caution: Much Maths)

- Another way to tackle this problem is to formulate a function to calculate the total number of presses that varies solely on X, let call that f(X)
- The function f(X) can be separated into 2 parts, f(X) = a(X) * X + b(X)
- Suppose N = 1,

$$f(X) = A_1/X + A_1\%X$$
$$= A_1/X + (A_1 - A_1/X*X)$$
$$= -(A_1/X) * X + (A_1/X + A_1)$$

a(X) b(X)

Alternative Subtask 3 (Caution: Much Maths)

$a(X) = -(A_1/X)$

$b(X) = (A_1/X + A_1)$

- We could show that $a(X)$ and $b(X)$ only change its value for $O(\text{sqrt}(A_1))$ times
- For $X <= \text{sqrt}(A_1)$, A_1/X can take on at most $\text{sqrt}(A_1)$ value obviously
- For $X > \text{sqrt}(A_1)$, since $1 <= A_1/X <= \text{sqrt}(A_1)$, A_1/X can take on at most $\text{sqrt}(A_1)$ value as well

Check out [Codeforces 1263C] for more information

Alternative Subtask 3 (Caution: Much Maths)

- Generalizing the function to consider all N elements

$a(X) = -(A_1/X)-(A_2/X)-...-(A_N/X)$

$b(X) = (A_1/X + A_1)+(A_2/X + A_2)+...+(A_N/X + A_N)$

- We could see that each element contribute to $O(\text{sqrt}(A_i))$ changes to the function $f(X)$, precompute all those changes.
- Then, the function $f(X)$ can be calculated from $f(X-1)$ in $O(1)$ time, which yield us the answer by finding the minimum $f(X)$

Score: 64

Time Complexity: $O(N * \text{sqrt}(A_{max}) + A_{max})$

M2122
Tournament of Elementalists

Subtask 1-2

1	8	$N = 1$
2	10	$N = 2$ Your favourite origin is Ocean.

Some case handlings...

You can write all the cases and study them one by one

Subtask 3

3	15	Your favourite origin is Ocean. Exactly **ONE** participating Elementalist comes from Woodland.

Observation:

- You need not be worried about Infernos, as you can always beat them
- If there's no Ocean, it's a "No" case
- If there's an Inferno, we can eliminate the only Woodland and "Yes"
  - We can put them to face each other in the first phase
  - e.g. `IW`...

Subtask 4-6

From now on, without loss of generality, we can assume that:

- “Your favourite origin is Ocean.”
  - You can shift the numbers to make it “true”
  - Before outputting the answers, shift them back

```
int N, cnt[3], rotations;
vector<int> ans;

char rotated_origin(int num) {
  const static string origins = "OIW";
  return origins[(num + rotations) % 3];
}

int main() {
  // ...
  rotations = fav == "Ocean" ? 0 : (fav == "Inferno" ? 1 : 2);
  for (int i = 0; i < rotations; i++) {
    swap(cnt[0], cnt[1]);
    swap(cnt[1], cnt[2]);
  }
  // putting 0, 1, 2 inside ans, where:
  // 0 is our favourite
  // 0 beats 1, 1 beats 2, 2 beats 0
  for (const int elm : ans) {
    cout << rotated_origin(elm);
  }
}
```

Subtask 4

If, for whatever reason, you think sorting them is a good idea...

`O...I...W...`

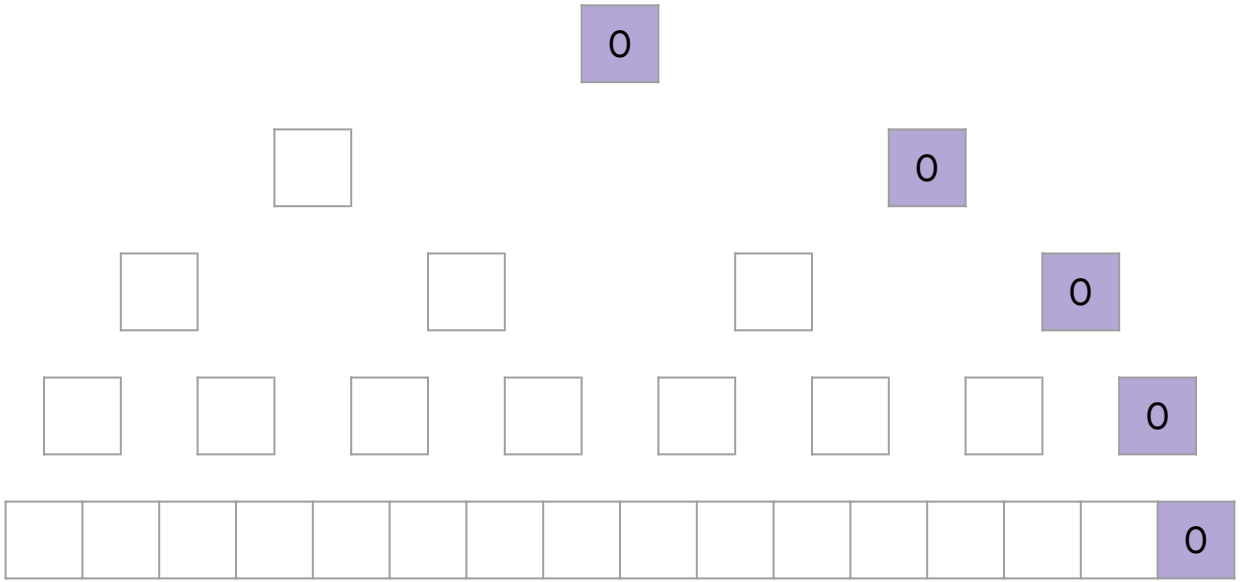
This works perfectly for Subtask 4, why?

The number of Woodlands < half

- There must exist one Inferno that can take care of all the Woodlands
- e.g. `O_______IWWWWWWW`

Full Solution

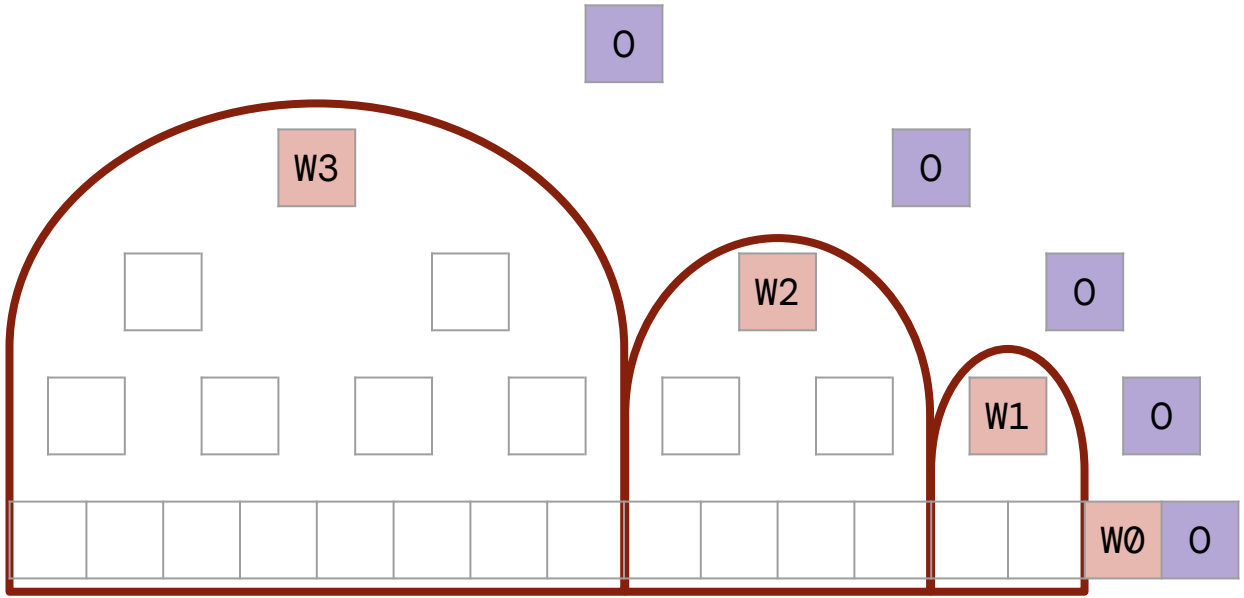
Without loss of generality, put 1 Ocean (our favourite) at the last spot



Full Solution

The W0, W1, ... are the winners from their respective subtrees

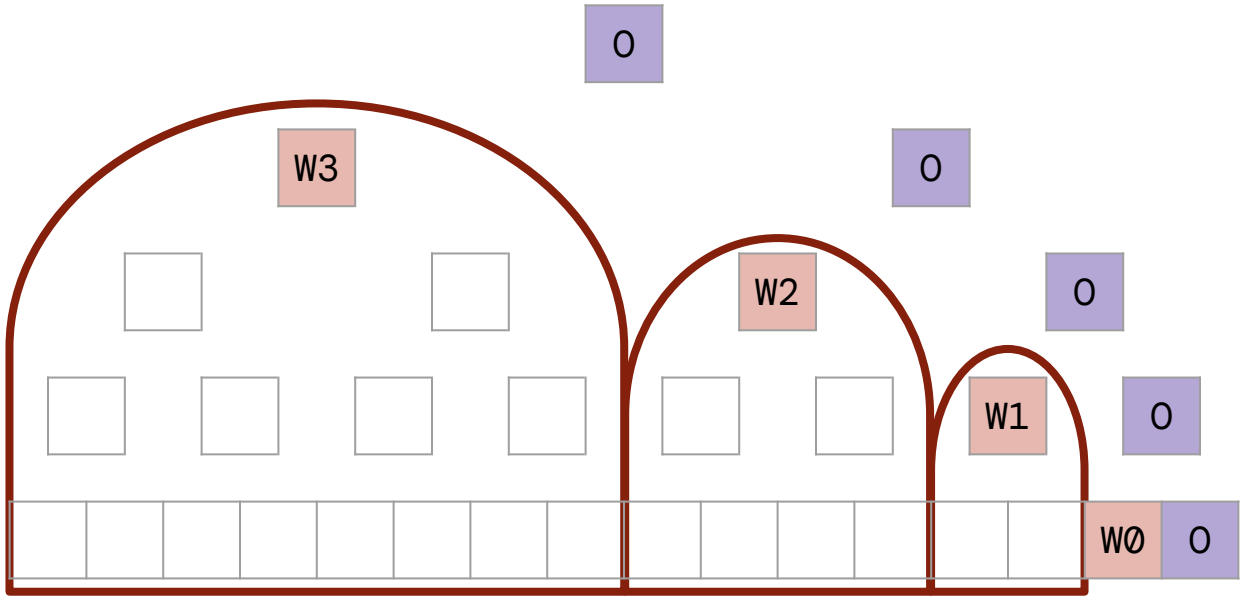
They can only be Ocean or Inferno



Full Solution

How to destroy the Woodlands?

We can greedily use 1 Inferno to eliminate 2^K-1 Woodlands

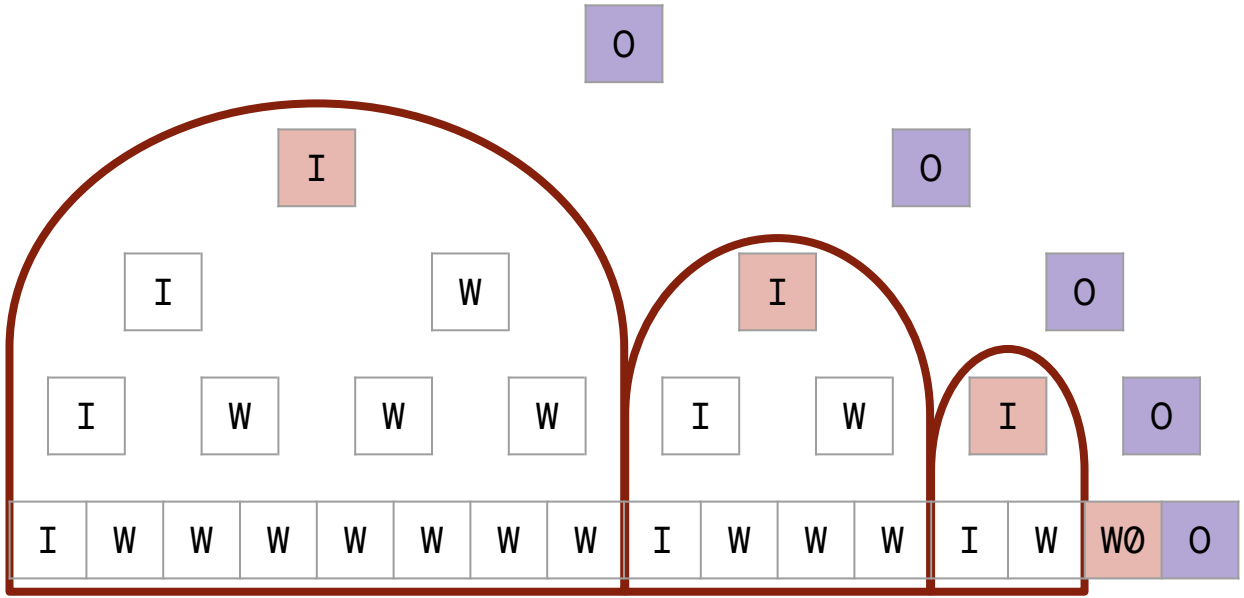


Full Solution

How to destroy the Woodlands?

We can greedily use 1 Inferno to eliminate 2^K-1 Woodlands

Example:

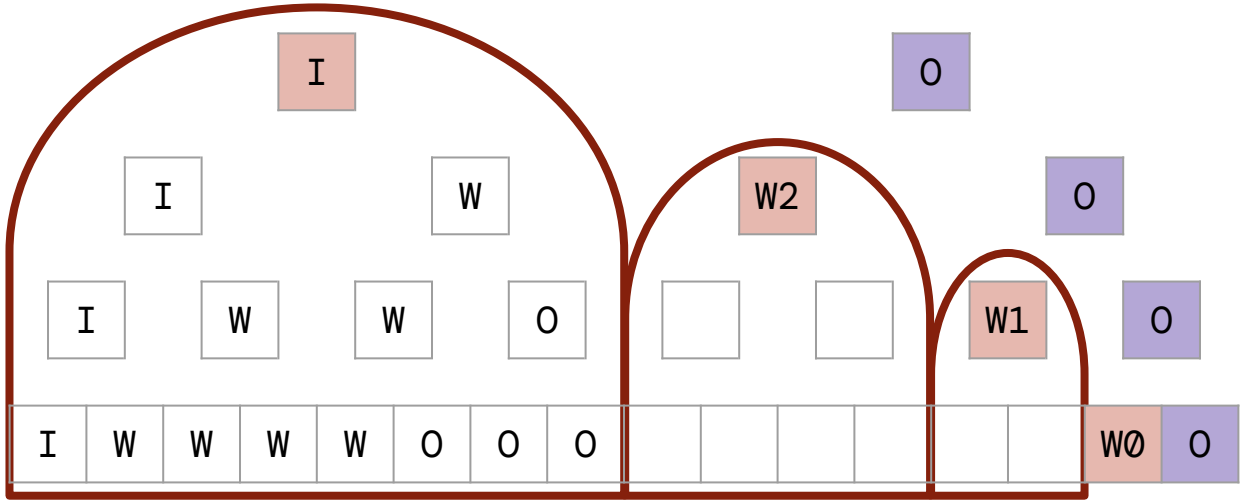


Full Solution

- We can start from the largest subtree (greedily)
- If there are still Woodlands remaining, first put an Inferno
- Put as many Woodlands as possible
- If having empty spots, we can still put Oceans

O

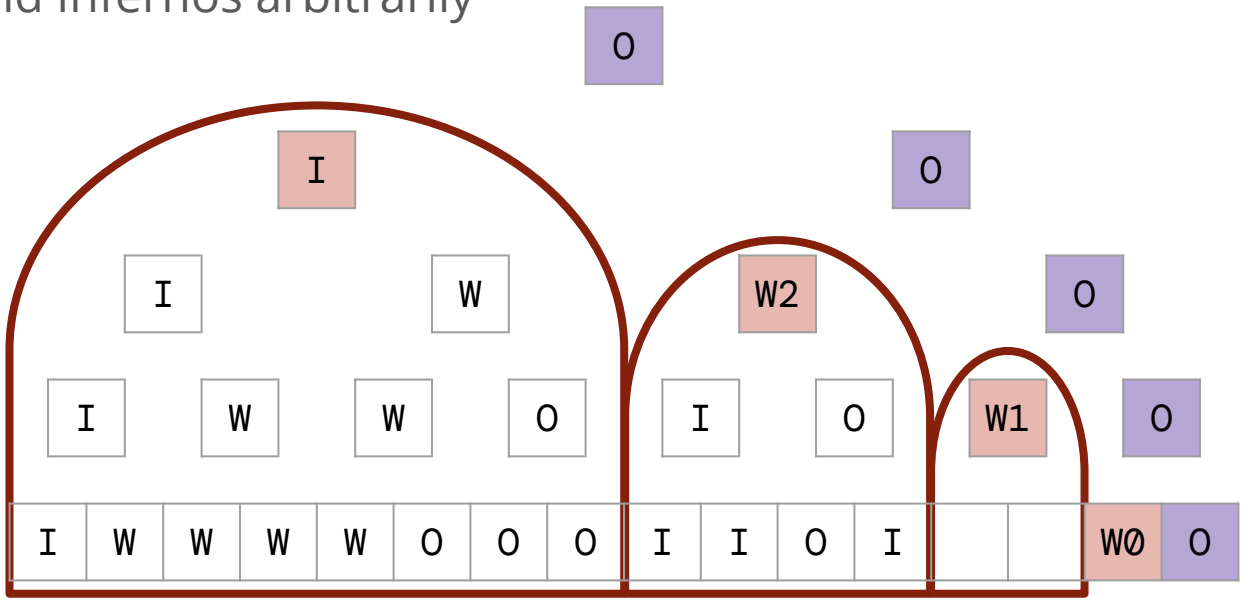
Example:



Full Solution

- We can start from the largest subtree (greedily)
- If no Woodlands remain,
  we can put Oceans and Infernos arbitrarily

Example:



Full Solution

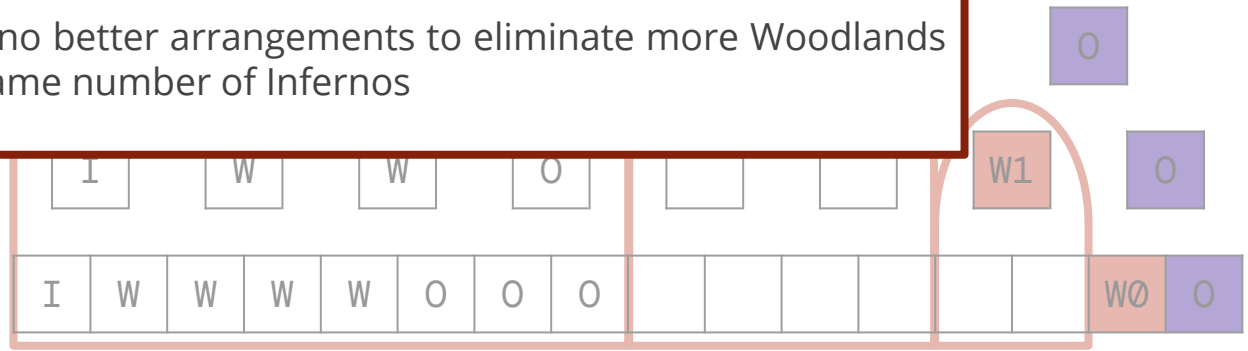
- We can start from the largest subtree (greedily)
- If there are still Woodlands remaining, first put an Inferno
- Put as many Woodlands as possible
- If having em

If no more Infernos at this step, it's a "No" case

This is true because, we have already best utilized the Infernos to maximize the number of Woodlands eliminated

There are no better arrangements to eliminate more Woodlands with the same number of Infernos

Example:



M2123
Pui Pui

Subtask 1

$1 \quad 5 \quad T = 0$

T = 0, It takes no time to stop at a station.

It's mean Poteto can eat as much carrots as he can at each station.

Simulate the whole trip, be careful of the bound of the hunger level. (0 to M)

Time Complexity: O(N)

Subtask 2

2 20 $N \leq 20$

N is small in this subtask, so we may try some kind of exhaustion.

Notice that for each station, Poteto can choose either pass the station or spend T to stop.

But how much carrots should he eat?

Observation 1

Notice that the amount of carrots Poteto ate wouldn't affect the total time spent.

Once Poteto decided to stop, he should eat as much carrots as he can.

Subtask 2

We can exhaust all possibilities of Potato stop at a station or not.

For each possibility, simulate the whole trip.

Be careful of the Hunger level again.

Time Complexity: $O(N*2^N)$

Subtask 3

3	35	$N \leq 200$ $M \leq 200$

Seems there are cube time complexity solutions of this subtask.

Let's think in this direction.

Subtask 3

Can define something like dp[i] = the shortest time travel from 1 to i?

Seems not so possible since Poteto may had M different hunger level in each station.

We can't use dp[i] to compute dp[i+1] easily since different hunger level may affect the answer.

So add another dimension instead.

Subtask 3

Let define dp[i][j] = the shortest time travel from 1 to i with the hunger level is equal to j when **arriving** station i. I.e after H = min(H+1,M), before pass or stop.

For passing the station **i-1**, dp[i][j] = dp[i-1][j-1] + (j-1).

Be careful for dp[i][M], dp[i-1][M] + M should be consider too.

For stopping at the station **i-1**, we need to consider how much carrots he ate.

Subtask 3

In case we didn't have observation 1, we may try to consider all amount of carrots ate at the station before.

Let k = the hunger level at the station i-1, and x = the carrots he ate.

To achieve hunger level j at the station i, k - x = j - 1

Notice that 0 <= x <= c[i-1], k = j - 1 + x,

j - 1 <= k <= j - 1 + c[i-1]

Subtask 3

For passing the station **i-1**, dp[i][j] = dp[i-1][j-1] + (j-1).

For stopping at the station **i-1**, dp[i][j] = min(dp[i-1][k] + T + j-1)
for all j - 1 <= **k** <= j - 1 + c[i-1].
Be careful the bound of hunger level again.

The answer = min(dp[N][j]) for all 1 <= j <= M.
Time Complexity: $O(N*M^2)$

Full Solution

We have observation 1. Can we this to reduce the time complexity?

Remind our transitional formula:

For stopping at the station **i-1**, dp[i][j] = min(dp[i-1][k] + T + j-1)

for all j - 1 <= **k** <= j - 1 + c[i-1].

We can only consider dp[i-1][j - 1 + c[i-1]]

Full Solution

We have observation 1. Can we this to reduce the time complexity?

Remind our transitional formula:

For stopping at the station **i-1**, dp[i][j] = min(dp[i-1][k] + T + j-1)

for all j - 1 <= **k** <= j - 1 + c[i-1].

~~We can only consider dp[i-1][j - 1 + c[i-1]]~~ **WA!**

Full Solution

Consider dp[i][1], the valid k is **1** - 1 <= **k** <= **1** - 1 + c[i-1]

I.e. 0 <= k <= c[i-1]

Actually, for those k hunger level in station i-1, the maximum amount of carrots Poteto can eat is k.

I.e. all those dp[i-1][k] satisfy observation 1, we need consider them all.

Full Solution

Let change the definition of dp[i][j]

Let dp[i][j] = the shortest time travel from i to N with the hunger level is equal to j when **arriving** station i.

For passing station i, dp[i][j] = dp[i+1][j+1] + j

Be careful dp[i][M] = dp[i+1][M] + M.

Full Solution

For stopping at station i, we can use observation 1, which eat as much carrots as Poteto can.

I.e. dp[i][j] = dp[i+1][**max(0, j-c[i])** + 1] + **max(0, j-c[i])** + T

The answer will be dp[1][M]

Time complexity: O(N*M)

M2124
Division Assignment

Subtask 1

$1 \le N, Q \le 5000$

We can simply carry out the queries.

Remember to use 64 bit integer to compute the sum.

O(QN)

Subtask 2

There is no type 2 query

What can we make good use of the constraints?

Subtask 2

Since there is no type 2 query (+= 1), it means that the numbers will only remain the same (for K = 1) or be reduced by at least a half (for K ≥ 2) for every query.

For **K = 1**, we simply **skip the division** and output the sum directly.

For K ≥ 2, we directly modify the numbers and compute the sum. All numbers will be reduced to 0 within $O(\log \max\{A_i\})$ queries.

After that the sum will always be 0.

$O(Q + N \log \max\{A_i\})$

Full Solution

No additional constraints

How can we deal with both queries?

Observation 1

Inspired by the previous subtask, we try to think of the **range** of the numbers.

Consider an array A with numbers [1, 2, 4, 8, 16, 32]. After a type 1 query with K = 2, the numbers become [0, 1, 2, 4, 8, 16].

The **range** of A is reduced by at least a half!

Observation 2

Consider the array [0, 1, 2, 4, 8, 16] again. After a type 2 query, the array becomes [1, 2, 3, 5, 9, 17].

The bound is shifted to the right by 1, but the **range** remains **unchanged**!

Full Solution

Instead of maintaining the original array, we maintain every **distinct** element in the range and its **frequency**. The total sum is computed first.

Whenever we have a type 2 query, we can compute the new sum by (old sum + N). We record the change (denoted by D) by D = D + 1.

Whenever we have a type 1 query with K ≥ 2 (**K = 1 is simply no change**), we can compute the new value for every original distinct element by ((old value + D) / K). We also update the sum with the new value and its frequency. This can be done by array + sorting / set / map.

Full Solution

As the range is reduced by at least a half for $K \geq 2$, it will be reduced to $O(1)$ after $O(\log (A_{max} - A_{min}))$ type 2 ($K \geq 2$) queries.

Overall: $O(Q + N \log N \log (A_{max} - A_{min}))$ (using set/map)
$O(Q + N \log (A_{max} - A_{min}))$ (using array and sorting)

Full Solution 2

Using observation 1, we can come up with another solution.

Find the **min** and the **max** of the array first. For type 2 query, we record the change(D = D + 1) directly as the previous full solution.

For type 1 query, we check if **(min + D) - (min + D) / K = (max + D) - (max + D) / K**. If not, we still directly update the values.

If so, then what does it mean?

Full Solution 2

If **(min + D) - (min + D) / K = (max + D) - (max + D) / K**, it means that the **difference** between the **new** value and the **old** value are the **same** for all elements!

In this case, we can treat the **division** as **subtraction**, which can be handled similarly as type 2 query.

Since the range **won't be increased** and it will be reduced to O(1) after O(log (A_{max} - A_{min})) type 2 query as mentioned before, this solution fits the time limit.

O(Q + N log (A_{max} - A_{min}))