TWGSS OI Team

2024/25 Mini Contest IV

Editorial

Prepared by Lo Ting Wai *Presented by* Fok Kin Wing Wong Ho Ki

15 May 2025

Fun Fact

The topic of Mini Contest IV is Minecraft!

The problem name of each problems can be found on Minecraft wiki.

Can you guess where they come from?

Statistics

	Problem	Attempts	Full	Max	Mean	Std Dev	LQ	Median	UQ	Subtasks	First Solve	
TC2541	Getting Started	21	10	10	9.524	2.13	10	10	10	2 | 20 · 3 | 20 · 5 | 20	Fok Kin Wing Nip Kit Fung	00:01:34
TC2542	Taking Inventory	20	20	20	16.8	6.608	20	20	20	3 | 19 · 5 | 16 · 7 | 17 · 5 | 16	Nip Kit Fung	00:03:00
TC2543	Time to Mine!	20	40	40	10.1	11.549	0	9.5	14	5 | 10 · 9 | 9 · 7 | 4 · 8 | 4 · 11 | 1	Wong Ho Ki	00:16:16
TC2544	Hot Topic	9	80	80	22.333	31.864	0	7	53.5	7 | 5 · 8 | 3 · 12 | 3 · 13 | 2 · 14 | 2 · 16 | 2 · 10 | 2	Wong Ho Ki	00:45:14
TC2545	We Need to Go Deeper	7	140	47	13.857	17.78	0	0	32	11 | 3 · 7 | 3 · 14 | 1 · 17 | 1 · 18 | 0 · 9 | 0 · 12 | 1 · 10 | 0 · 16 | 0 · 26 | 0		
TC2546	Time to Strike!	7	160	160	33.714	52.415	9	9	33	9 | 6 · 7 | 3 · 6 | 1 · 21 | 1 · 17 | 2 · 14 | 1 · 21 | 1 · 32 | 1 · 33 | 1	Wong Ho Ki	02:21:20
		21	**450**	**357**	**60.571**	**74.364**	**27.5**	**44**	**65.5**			

Getting Started

TC2541

Problem Statement

Given a message in the format of `Press` S `to start.`

Output the button to start the game.

There is a total of 4 different buttons, named `A`, `B`, `X`, `Y`.

If S is `Any Button`, output any button scores 100%.

Easy?

Subtask 1

$S = \texttt{A}$

Oh, there is only one possible value for S.
The input is definitely `Press A to start.`

What should we output?
Of course `A`!

Open Foundation Courses' slides to learn how to output `A`.

Expected Score	2	Cumulative	2

Subtask 2

$S \neq$ `Any Button`

The input is `Press` S `to start.`

As $S \neq$ `Any Button`, the output must be S.

How to get S?

Use `cin` twice!

Check Foundation Courses' slides to learn how to input string.

Expected Score	5	Cumulative	5

Full Solution

S can now be `Any Button`.

Simply output S doesn't work.
When S is `Any Button`, we should output one of `A`, `B`, `X`, `Y` instead.

However, directly output `A` also doesn't work.
The solution fail when $S \neq$ `A`.

We need another tool.

Full Solution

Obviously, there are two cases with separate solutions.
When S is `Any Button`, we would output one of `A`, `B`, `X`, `Y`.
Otherwise, we would output S instead.

To do case branching, we can utilise the power of `if`.
When the second string is `Any`, we know S would be `Any Button`.
Hence, output one of `A`, `B`, `X`, `Y` or otherwise output S.

Check Foundation Courses' slides to learn how to branch.

Expected Score	**10**	Cumulative	10

Alternative Solution

If you are smart enough, you can observe that the first character of `Any Button` is `A`, a valid answer.

Instead of outputting S, as the first character of S is the answer.

We can simply output $S[0]$.

You can also get the whole sentence by `getline(cin, S)`.

Then output $S[6]$.

Takeaway

Branching is a powerful tool in C++.

You can separate your solutions into multiple pieces and select the suitable one using branching.

Branching into smaller cases often result in easier solutions.

Or you would need an observation to fit everything into one solution.

Cases branching and solve them separately!

Taking Inventory

TC2542

Problem Statement

Given N rows and M columns of integers.

Check if all integers are 0.

Output `Yes` if all integers are 0.

Otherwise, output `No`.

Harder but still simple.

Subtask 1

$N = M = 1$

There is 1 row and 1 column of integer. How many integers are there?
There is only 1!

Simply input the integer and check if the integer is zero.
Use if to output `Yes` or `No`.

Remember to input N and M too although you know they are both 1.
Check Foundation Courses' slides to learn more about data type.

Expected Score	3	Cumulative	3

Subtask 2

$1 \leq N, M \leq 2$

Oh, N, M are quite small.

How many cases are there?

(N, M) can only be one of $(1,1), (1,2), (2,1), (2,2)$.

Only four cases!

Using the knowledge from the last problem, we should utilise the power of branching!

Let's separate into four cases!

Subtask 2

We have solved the cases where $N = M = 1$ in Subtask 1.

How about $N = M = 2$?

According to the problem statement, we should determine if all of the integers are 0.

We would output `Yes` if all of the integers are 0 or `No` if not.

How to chain the conditions together?

Use `&&`!

Check Foundation Courses' slides to learn more about Boolean operations.

Expected Score	8	Cumulative	8

Subtask 3

$N = 1$

This time, M can be as large 2000.
It is not quite possible to write 2000 if to separate cases.
(You could if you spend a lot of time)

Ideally, we would like to check each integer whether it is 0.
Notice that we are doing repeated operations again and again.

We need a better tool.

Subtask 3

Let's introduce looping!

Using looping, you can do repeated operations multiple times.

In this subtask, you can either use a for loop or while loop.

We can use a classical for loop to loop M times.

Each checking if the next integer is 0.

Just like Subtask 2, we can `&&` all conditions and output `Yes` or `No` at the end of the code.

Check Foundation Courses' slides to learn more about looping.

Subtask 3

`NewPlayer` $= True$

For i in $1 \ldots M$:

    Input $A_{1,i}$

    `NewPlayer` = `NewPlayer` && $(A_{1,i} = 0)$

If `NewPlayer`:

    Output `Yes`

Else:

    Output `No`

Expected Score	**10**	Cumulative	15

Full Solution

There can be N rows in total.

As N is as large as 2000, it is not quite possible to copy 2000 Subtask 3.

(You could if you spend a lot of time)

Notice that we are doing repeated operations again and again.

Similarly, we can just use another for loop to loop N times.

Each loop run the Subtask 3's loop.

Using nested loops, we can solve this problem easily.

Full Solution

`NewPlayer` $= True$

For i in $1 \ldots N$:

    For j in $1 \ldots M$:

        Input $A_{i,j}$

        `NewPlayer` = `NewPlayer` && $(A_{i,j} = 0)$

If `NewPlayer`:

    Output `Yes`

Else:

    Output `No`

Expected Score	**20**	Cumulative	20

Alternative Solution

Some contestants cleverly make an observation.

When there is a non-zero integer, we always output `No` at the end.

Instead of inputting other integers, we can immediately output `No` when we see a non-zero integer.

If we didn't output `No` previously, output `Yes` at the end.

We can make use of `return 0;` to terminate the code after output `No`.

Common Mistakes

A few contestants use `break;` instead of `return 0;`

There is a key difference between `break;` and `return 0;`

When using `break;`, the current for loop will be exited but the code will not.

When using `return 0;`, the whole code terminates immediately.

Using `break;` instead of `return 0;` may cause you to output multiple `No`. You will receive a verdict of “Wrong Answer” instead.

Also, think carefully of what the Boolean variable is storing for first solution.

Takeaway

For-loop is an essential tool in C++.

In most problems, you have to loop for multiple times to solve the problem.

You should learn and familiar with for loop and nested for loop.

They will help you in most of the cases.

Maintaining variables are useful when using for loop.

They can transfer data through for loop.

Update the data for each loop.

Time to Mine!

TC2543

Problem Statement

Given four parameters, T, D, M, C, and you have a new pickaxe initially.

You can mine 1 stone per M seconds with a cost of 1 durability of the pickaxe.

Each pickaxe has a durability of D initially. The pickaxe will be broken when its durability drops to zero.

You can craft a pickaxe using 3 stones and C seconds.

You need to output the maximum number of stones you can get using at most T seconds.

Observation

You will only craft a new pickaxe when your original one is broken.

Notice that crafting a new pickaxe will spend you 3 stones and C seconds.

As time is stone, crafting a pickaxe cannot benefit us.

When the pickaxe is broken, time is not stone anymore.

You can choose to wait till the remaining time is past or craft a new pickaxe.

However, you may not benefit by crafting a new pickaxe because of its cost.

Subtask 1

$1 \leq T \leq D \leq 10^9$
$M = 1$

Notice that you can mine at most 1 stone per second.

As $T \leq D$, your first pickaxe never runs out of durability.

If you can mine 1 stone per second and there is a total of T seconds.

How many stones can you mine?

It is just T stones!

Expected Score	5	Cumulative	5

Subtask 2

$1 \leq T \leq D \leq 10^9$

M is no longer 1.

How many stones can you mine?

By some magical operations, you can find that the answer is $\lfloor \frac{T}{M} \rfloor$!

Expected Score	**14**	Cumulative	14

Subtask 3

$1 \leq T \leq 2000$

T is quite small now.

We can loop through T without getting “Time Limit Exceeded”.

By the observation, we would mine stone whenever we can and craft pickaxe when we do not have one.

Is it correct?

Subtask 3

Crafting a pickaxe may not benefit.

Instead, we have to invest 3 stone to craft a new pickaxe.

Hence, if we cannot mine at least 3 stone back, crafting a pickaxe would lose us stone.

To prevent losing stone accidentally, we can use a variable to save the best record we have achieved before.

Output the best record instead of the last record to ensure the best result.

Subtask 3

`BestResult` $= 0$

`CurrentDurability` $= D$

`CurrentStone` $= 0$

While $T > 0$:

    If `CurrentDurability` $== 0$ && $T \geq C$ && `CurrentStone` ≥ 3:

        `CurrentDurability` $= D$

        T `-=` C

        `CurrentStone` `-=` 3

Subtask 3

Else If `CurrentDurability` > 0 && $T \geq M$:

`CurrentDurability` $-= 1$

$T -= M$

`CurrentStone` $+= 1$

Else:

Break

`BestResult` $=$ max(`BestResult`, `CurrentStone`)

Output `BestResult`

Expected Score	7	Cumulative	21

Subtask 4

$C = 10^6$

T can be as large as 10^9 now, simply loop will cause “Time Limit Exceeded”.

However, as C is quite big too, the number of “crafting pickaxe” operation is still small enough to loop through.

Can we skip the “mining” operations?

Subtask 1!

Subtask 4

`BestResult` $= 0$

`CurrentDurability` $= D$

`CurrentStone` $= 0$

While $T > 0$:

    If `CurrentDurability` $== 0$ && $T \geq C$ && `CurrentStone` ≥ 3:

        `CurrentDurability` $= D$

        T `-=` C

        `CurrentStone` `-=` 3

Subtask 4

Else If `CurrentDurability` > 0 && $T \geq M$:

`NumberOfStone` = min(`CurrentDurability`, $\left\lfloor \frac{T}{M} \right\rfloor$)

T −= $M \times$ `NumberOfStone`

`CurrentStone` += `NumberOfStone`

`CurrentDurability` −= `NumberOfStone`

Else:

Break

`BestResult` = max(`BestResult`, `CurrentStone`)

Output `BestResult`

Subtask 4

Combining the idea from Subtask 3 and Subtask 2.

It can be solved in 0.2 seconds.

O(	T/C	)	Expected Score	**15**	Cumulative	29

Full Solution

C can be small that there are 10^9 “crafting pickaxe” operations.

Can we skip it too?

Notice that almost all pickaxe we made will be broken.

We are repeating “craft” → “mine” → “broken” again and again.

Except for the first pickaxe that we don’t need to craft and the last pickaxe may not be broken at last.

We can merge “craft” → “mine” → “broken” into one operation.

Full Solution

Let call the new operation “craftmine”.

Step 1 - craft:

Spend C seconds

Lose 3 stones

Step 2 - mine until pickaxe is broken:

Spend $D \times M$ seconds

Gain D stones

Each “craftmine” operation spends $C + D \times M$ seconds in return of $D - 3$ stones.

Full Solution

The process of mining stones would become:

Mine until the first pickaxe is broken

Do some “craftmine” operations (possibly zero)

Craft the last pickaxe and mine until time runs out (optional)

Step 1 - Mine until the first pickaxe is broken:

$T \mathrel{-}= D \times M$

Answer = D

Carefully handle the case where $T < D \times M$.

Full Solution

Step 2 - Do some “craftmine” operations (possibly zero):

Recall that each “craftmine” operation spends $C + D \times M$ seconds in return of $D - 3$ stones.

If $D - 3$ is negative, we definitely will not want to do any operations.

Otherwise, we would do as many as possible greedily. (No reason to not do)

How many operations can we do at most?

By some magical operations, we can do at most $\lfloor \frac{T}{C+D \times M} \rfloor$ operations!

Full Solution

Step 3 - Craft the last pickaxe and mine until time runs out (optional):

If we decide to do so, we will first spend C seconds and 3 stones.

How many stones can we get after crafting the last pickaxe?

By some magical operations, we can get $\lfloor \frac{T-C}{M} \rfloor$ stones!

So totally, we will get $\lfloor \frac{T-C}{M} \rfloor - 3$ stones

If $\lfloor \frac{T-C}{M} \rfloor - 3 > 0$, we will craft and mine or otherwise we will stop.

Expected Score	**40**	Cumulative	40

Takeaway

For loop is an essential tool in C++.

However, for loop may not be able to help to get “Accepted” all the time.

It is limited to “time limit” and you can only loop about 10^9. (Usually less)

To reduce the time complexity, advanced coding skills have to be applied.

One simple techniques is to use math to skip loops.

Sometimes, you can derive a formula to calculate the correct answer instead of looping multiple times.

Hot Topic

TC2544

Problem Statement

There are N furnaces in a row where the i^{th} furnace has an efficiency of A_i.
(it can smelt A_i stones in a second using a coal)

If k consecutive furnaces are chosen, the efficiency of each furnace will multiply by c_k. You can use arbitrary consecutive furnaces and fuel them evenly using at most M coals.

Given Q events where the j^{th} event changes c_{K_j} to C_j. Find the maximum number of stones you can smelt.

Notice that the events affect the value of c_{K_j} permanently.

Subtask 1

$N = 1$
$Q = 1$

There is only one furnace.

Obviously, we won't choose 0 furnace.

Hence, we must choose the only furnace.

Just apply the formula to calculate the answer!

Answer $= (A_1 \times c_1) \times M$

Expected Score	7	Cumulative	7

Subtask 2

$N = 2$

There are two furnaces.

We can now choose to choose the first furnace only or the second furnace only or both furnace.

Which one is better?

Use max function to find the answer.

Subtask 2

For i in $1 \ldots Q$:

Input K_i, C_i

$c_{K_i} = C_i$

Output $\max((A_1 \times c_2 + A_2 \times c_2) \times \lfloor \frac{M}{2} \rfloor,$

$\max((A_1 \times c_1) \times M, (A_2 \times c_1) \times M))$

O(	Q	)	Expected Score	**8**	Cumulative	15

Subtask 3

$1 \leq N, Q \leq 50$

N, Q is quite small, let's try to brute force all possibilities.

Let's say we try to select furnace l to furnace r.

There are total of $r - l + 1$ furnaces selected.

Let $k = r - l + 1$

We can smelt $(A_l \times c_k + A_{l+1} \times c_k + \cdots + A_r \times c_k) \times \lfloor \frac{M}{k} \rfloor$ stones using this configuration.

Just loop all configuration and save the best result!

Subtask 3

For l in $1 \ldots N$:

For r in $l \ldots N$:

Tmp $= 0$

$k = r - l + 1$

For x in $l \ldots r$:

Tmp $+= A_x \times c_k$

Ans $= \max(\text{Ans}, \text{Tmp} \times \lfloor \frac{M}{k} \rfloor)$

O(QN^3)	Expected Score	19	Cumulative	27

Subtask 4

$1 \leq N, Q \leq 500$

O(QN^3) are too slow now but N, Q is still small.

If we can remove one N in the time complexity, O(QN^2) will be fast enough.

If we continue to brute force all possible ranges of furnaces, we have spent O(N^2) for each query.

We have to calculate the number of stones we can smelt using furnace l to furnace r in O(1).

Can we do that?

Observation 1

Let's say we try to select furnace l to furnace r.

Let $k = r - l + 1$

Instead of saying we can smelt $(A_l \times c_k + A_{l+1} \times c_k + \cdots + A_r \times c_k) \times \lfloor \frac{M}{k} \rfloor$ stones

We can say that we can smelt $(A_l + A_{l+1} + \cdots + A_r) \times c_k \times \lfloor \frac{M}{k} \rfloor$ stones.

Can we calculate $A_l + A_{l+1} + \cdots + A_r$ in O(1)?

Partial Sum!

Subtask 4

Precompute ps

For l in $1 \ldots N$:

For r in $l \ldots N$:

$k = r - l + 1$

Ans = max(Ans, $(ps[r] - ps[l-1]) \times c_k \times \lfloor \frac{M}{k} \rfloor$)

O(	$\boldsymbol{QN^2}$	)	Expected Score	**32**	Cumulative	40

Subtask 5

$1 \leq Q \leq 5000$

N, Q are quite big now.

It is impossible for us to loop through each ranges.

But is it necessary for us to loop all ranges for each query?

The answer is no!

Notice that for each k, $c_k \times \lfloor \frac{M}{k} \rfloor$ is constant.

$(ps[r] - ps[l-1]) \times c_k \times \lfloor \frac{M}{k} \rfloor$ is maximised when $ps[r] - ps[l-1]$ is.

Subtask 5

We can precompute the largest $ps[r] - ps[l-1]$ for each k.

Saving the result in an array.

When we need to get the maximum $ps[r] - ps[l-1]$ for each k.

We can just access the array in $O(1)$.

As we precompute the array at the beginning, it is outside the for-loop of Q and only run once, greatly reduce the time complexity.

Subtask 5

Precompute MaxSum:

For l in $1 \ldots N$:

For r in $l \ldots N$:

$$\text{MaxSum}[r-l+1] = \max(\text{MaxSum}[r-l+1], ps[r] - ps[l-1])$$

For k in $1 \ldots N$:

$$\text{Ans} = \max(\text{Ans}, \text{MaxSum}[k] \times c_k \times \lfloor \frac{M}{k} \rfloor)$$

O($N^2 + QN$)	Expected Score	46	Cumulative	54

Subtask 6

$c_i = 1$

$C_i \leq C_j$ for $i < j$

c_k is always increasing!

QN can be as large as 10^9, we cannot loop through each k for each query.

However, for each query, only one c_k is changed.

It is unnecessary to check N possible k and we can check $k = K_i$ only.

As c_k is increasing, the answer is also increasing and thus we can use max to maintain the answer.

Subtask 6

For k in $1 \ldots N$:

$$\text{Ans} = \max(\text{Ans}, \text{MaxSum}[k] \times c_k \times \lfloor \frac{M}{k} \rfloor)$$

For i in $1 \ldots Q$:

$$c_{K_i} = C_i$$

$$\text{Ans} = \max(\text{Ans}, \text{MaxSum}[K_i] \times c_{K_i} \times \lfloor \frac{M}{K_i} \rfloor)$$

O($N^2 + Q$)	Expected Score	**16**	Cumulative	70

Full Solution

Sometimes, c_k may decrease and cause the answer to decrease too.

We need to update c_k while maintain the maximum answer fast.

Is there a data structure supporting query max and update fast?

Yes!

Both `set`, `map` or `priority_queue` work!

They support both functions in just $O(\lg N)$.

$O(\ N^2 + Q \lg N\ )$	Expected Score	**80**	Cumulative	80

Takeaway

Simply brute force can give you decent amount of scores.

Try brute force when the constraints are small.

Most likely there will be subtasks for brute force solution.

To improve your solution, you have to make observation and make use of advanced skills you learn.

Try to follow the subtasks, they may guide you to improve your solution!

We Need to Go Deeper

TC2545

Problem Statement

Given a connected graph of N nodes and M bidirectional weighted edges.

Find the shortest path to go from node 1 to any node with water and then go to any node with lava.

A simple multi-source, multi-sink shortest path problem.

Although most of you haven't learnt about graph, you can still score well in this problem.

Subtask 1&2

Simple graph with only two nodes.

Travelling to another node cost l_1 seconds.

If there is both water and lava in node 1, answer is 0

If there is both water and lave in node 2, answer is l_1

Otherwise, we would first go to node 2 then back to node 1, answer is $2 \times l_1$

If you can understand the problem, you can probably solve this yourself.

Expected Score	**18**	Cumulative	18

Subtask 3

The graph is a star with centre of node 1.

In a star, you can travel to any node using a most 2 edges.
Therefore, there is only a few cases you have to consider.

Carefully loop through all cases and you can get the shortest path.

Let's look at the cases.

Subtask 3

Case 1:

There are water and lava at node 1->shortest path$= 0$

Case 2:

There is only water at node 1->shortest path to the node which contain lava

Case 3:

There is neither water or lava at node 1.

We would either go to a node with both water and lava or first go to a node with water then a node with lava. There will only involve 2 distinct edges.

Expected Score	**25**	Cumulative	32

Subtask 4

The graph is a chain.

There is only one way to travel to a node with water. Keep travelling to node with larger id.

Once you get to the node with water. There is only two way to travel to a node with lava. Keep travelling to node with smaller id or keep travelling to node with larger id.

Expected Score	28	Cumulative	49

Subtask 5&6

There is only one node with water and only one node with lava.

Sadly, from this subtask, you must learn something about graph to solve it.

Let node u and node v be the node with water and lava respectively.

We must travel from node 1 to node u to node v.

The answer is shortest path from 1 to u plus that from u to v.

Using DFS / BFS causes time complexity of O(NM) and Dijkstra's algorithm for faster time complexity.

O($\frac{NM}{(N+M)\lg M}$)	Expected Score	27	Cumulative	76

Subtask 7

$W = L$

We can observe that when there is water in a node, there is also lava.

Going to any one of the node with water complete the path.

Use Dijkstra's algorithm once and find the minimum shortest path one from all nodes with water.

O($(N+M)\lg M$)	Expected Score	**12**	Cumulative	88

Subtask 8

$1 \leq N, M \leq 1000$

Similar to subtask 6, but now there are multiple nodes with water and multiple nodes with lava but limited by N.

We can loop through all nodes with water as the first target, then add the shortest path to any nodes with lava from that node (Subtask 7).

O($N^2 \lg M + NM \lg M$)	Expected Score	**39**	Cumulative	98

Subtask 9

The graph is a tree.

There is a special property of tree, there is a unique path from any node u to any node v.

Similar to Subtask 4, but now there is multiple way to travel to a node with lava. However, for each way, you can save the shortest path to lava as there is only one direction.

It can be precompute by the first DFS and solve it in the second DFS.

Subtask 9

Func dfs(u, par):

    If $L[u] =$ '1':

        ShortestPathFrom$[u] = 0$

    Else:

        ShortestPathFrom$[u]$ =Inf

    For v connected to u with weight w:

        If $v \neq par$:

            ShortestPathFrom$[u] =$ min(ShortestPathFrom$[u]$,
                dfs(v, u)+w)

    Return ShortestPathFrom$[u]$

Subtask 9

Func solve(u, par, shortestPathToPar):

    Set<int> shortestPath = {shortestPathToPar}

    For v connected to u:

        If $v \neq par$:

            shortestPath.insert(ShortestPathFrom[v])

    If $W[u] =$ '1':

        Ans = shortestPathTo[u] +*shortestPath.begin()

    Else:

        Ans = Inf

Subtask 9

For v connected to u with weight w:

If $v \neq par$:

shortestPath.erase(ShortestPathFrom[v])

Ans = min(Ans, dfs(v, u, $*$shortestPath.begin()+w))

shortestPath.insert(ShortestPathFrom[v])

Return Ans

O($N \lg N$)	Expected Score	**65**	Cumulative	114

Full Solution

We can separate the graph into two layer.

Each node u is separated into two parts.

One is node u without visiting node with water.

One is node u after visiting node with water.

Edges should also be duplicated and carefully build the new graph.

Then you can run Dijkstra's algorithm once to find the shortest path to node with lava after visiting node with water.

$O((N+M)\lg M)$	Expected Score	**140**	Cumulative	140

Takeaway

We usually face some problems that we haven’t learnt before.

We can try to solve subtasks as some subtasks are not related to the full solution and don’t require advanced skills.

When we face difficult problems, we can try to convert it into known problems and we can solve it using classical skills.

For example, rebuild the graph is a common technique.

Time to Strike!

TC2546

Problem Statement

There are N types of monsters and M waves of monsters.

In each wave, you can choose either run away or kill all monsters.

If you run away, you will receive damage from all monsters.

If you fight all K monsters, there will be K turns of battle.

In each turn, you will receive damage from all alive monsters and then you can choose to kill any one of the monsters.

After K turns, you will enter the next wave, again and again.

You have H health points and your health points cannot become non-positive at any moment. Find the maximum number of monsters you can kill or output -1 if you cannot survive.

Expected Score	9	Cumulative	9

Subtask 1

$N = M = K_1 = 1$

There is one type of monsters and one monster in the only wave.

No matter you run away or fight with the monster, you must receive damage from the monster exactly one time. By greedy, you will kill the monster.

If your health point drops to zero or less after receiving the damage, output -1 or the answer is always 1.

Subtask 2

$N = M = 1$

There is only one type of monsters and only one wave of monsters.

There are two cases to be considered, run away or fight against them.

If we run away, we will receive $K_1 \times A_1$ damage.

If we fight against them, we will receive $((K_1) + (K_1 - 1) + (K_1 - 2) + \cdots + 1) \times A_1$ damage

Just check if your health point is enough to survive after you select run away and fight against them.

O(K_1)	Expected Score	**16**	Cumulative	16

Observation 1

No matter we run away or fight against all, we must receive damage from all monsters once.

Hence, we can first decrease our health points without choosing run or not.

If we want to kill monsters, we have to take some more damage from them.

If your health point drops to zero or less before choosing run or not, we cannot survive and we should output -1.

Subtask 3

$N = 1$
$K_1 = K_2 = \cdots = K_M$

After decreasing the health point for each wave, notice that choosing to fight against monsters in each wave is identical.

We can choose to K_1 monsters by taking $((K_1 - 1) + (K_1 - 2) + \cdots + 1) \times A_1$ damage.

Use division to check the how many monsters you can kill.

O(K_1)	Expected Score	**22**	Cumulative	22

Observation 2

When there is more than multiple types of monsters appear in the same wave, which monster should we kill first.

By greedy, we would like to kill the monster with higher damage.

Killing the one with highest damage can reduce the damage taken in the future.

We can sort the monsters by their damage and kill them one by one.

Subtask 5

$M = 1$

There is only one wave but may consist of monsters of different types.

As there are multiple types of monsters, we have to convert the names of the monsters to the damage of the monsters.

To convert a string to an integer, we can use map in C++.

Apply the observation to find the correct order.

O($K_1 \lg M$)	Expected Score	33	Cumulative	39

Subtask 6

$K_1 = K_2 = \cdots = K_M = 2$

By the observation, we can find the cost of fighting all monsters in each wave.

Let C_i be the cost of killing all monsters in the i^{th}.

As each wave consists of two monsters, we would like to kill monsters in more waves.

Greedily, we would like to spend less cost to gain 2 kill count.

O($M \lg N + M \lg M$)	Expected Score	**20**	Cumulative	53

Subtask 7

$1 \leq M \leq 10$

After calculating the cost of each wave, we face a problem that we don't know which waves we should fight against all monsters.

However, M is quite small so we can brute force all combinations.

There are only $2^M \leq 1024$ cases.

$O(\ M \lg N + M \lg M + M \times 2^M\ )$	Expected Score	**54**	Cumulative	74

Subtask 4

$N = 1$

After calculating the cost of each wave, we face a problem that we don't know which waves we should fight against all monsters.

Actually, the problem is identical to the classical Knapsack DP.

Knapsack DP is an advanced skill in oi.

Check HKOI training's slides for more information.

$O(\ MN \lg N + MH\ )$	Expected Score	**43**	Cumulative	95

Subtask 8

$1 \leq N, M \leq 100$

Similar to Subtask 4.

But you must use map to convert string to integer.

$O(MN \lg N + MM \lg M + MH)$	Expected Score	**48**	Cumulative	127	

Full Solution

Suppose $M \times H$ can be as large as 10^8.

We should not be able to store 10^8 integers.

We have to reduce the memory complexity of the solution.

We can use rolling DP instead of normal DP.

You can also use short (2-byte integer) instead of integer to store.

(Maybe the cases are too weak that you will not exceed the memory using Subtask 8 solution)

$O(M \lg N + M \lg M + MH)$	Expected Score	**160**	Cumulative	160

Takeaway

Not all problems can be solved easily.

However, you can still get lots of marks.
Try doing subtasks even you feel the problem is hard.