GIST ALGORITHM MASTERS

2024 Contest

Official Solutions Editorial

2024 GIST Algorithm Masters TF

	Problem	Difficulty	Author
A	Printing GIST	Easy	Hyegeun Song @GIST invrtd_h
B	Matching Couple Ring	Easy	Jonghyeon Park @JNU belline0124
C	FOCUS	Normal	Hyegeun Song @GIST invrtd_h
D	Binary Grid	Normal	Hyegeun Song @GIST invrtd_h
E	Making ChatGPT	Hard	Hyegeun Song @GIST invrtd_h
F	Infinitely Popping Bubbles in a Bottle	Hard	Hyegeun Song @GIST invrtd_h Yoonsu Lee @JNU lys9546
G	Copying Money	Hard	Hyegeun Song @GIST invrtd_h Mingyu Ko @JNU jjkmk1013
H	Elementary Wizardry	Challenging	Hyegeun Song @GIST invrtd_h
I	Operating DevNight	Challenging	Hyegeun Song @GIST invrtd_h

Note

» We excluded cheating participants from the first solver statistics.

» This document was originally written in Korean. This document was machine translated from the Korean document and then reviewed by a non-English speaker.

A. Printing GIST

`#implementation, #string`

Difficulty — **Easy**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 115 Submitted, 103 Passed
    AC Ratio: 90.435%
  - First Solver: `riroan`, +1min

- Offline Onsite Contest
  - 128 Submitted, 51 Passed
    AC Ratio: 39.844%
  - First Solver: Geunseong Kim @JNU, +2min

A. Printing GIST

- ✓ Implement the formula presented in the problem as is.
- ✓ Most programming languages use 0–based index, but you should keep in mind that matrices use 1–based index.
- ✓ Implement the following equation:

$$Y = \left(y_{ij}\right), y_{ij} = x_{\lfloor \frac{i}{K} \rfloor \lfloor \frac{j}{K} \rfloor}$$

- ✓ Another way is to add 1 space of padding in the data, implement it as a 1–based index, and then cut the padding when output.

B. Matching Couple Ring

`#implementation`, `#hash-map`

Difficulty — **Easy**

- Author: Jonghyeon Park @JNU `belline0124`

- Online Open Contet
  - 111 Submitted, 77 Passed
    AC Ratio: 70.27%
  - First Solver: `riroan`, +2min

- Offline Onsite Contest
  - 165 Submitted, 41 Passed
    AC Ratio: 24.848%
  - First Solver: Geunseong Kim @JNU, +7min

B. Matching Couple Ring

- ✓ For each ring feature given as input, the goal is to find the ring feature that appears exactly twice.
- ✓ The correct solution is to create and utilize key-value pairs with the ring feature as the key and the set of the number of appearances of the ring feature and the name of the person wearing the ring as the values.
- ✓ Because the concept of key-value pairs and how to deal with them are less difficult, solving this problem without dealing with key-value pairs is more difficult. Therefore, we did not specifically block the processing of simple loop statements without using key-value pairs.

C. FOCUS

`#implementation, #bitmasking`

Difficulty — **Normal**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 75 Submitted, 51 Passed
    AC Ratio: 69.333%
  - First Solver: `edwardblue`, +5min

- Offline Onsite Contest
  - 113 Submitted, 36 Passed
    AC Ratio: 31.858%
  - First Solver: Jeonghoon Seong @GIST, +16min

- ✓ **Subtask 1**: Count the number of 3, and if there is an odd number, key 0 is in P_1, key 1 is in P_0, and all remaining keys remain in place. If there is an even number, all the keys are in place.
- ✓ **Subtask 2 & 3**: There are two solutions.
  - ✓ **Bitmasking solution**: Use the bit operation x & (1 << i) to store all information about which bit is turned on, and then swap only when the number of bits turned on is 2.
  - ✓ **Preprocessing solution**: After declaring an array arr of length 256, find the value of $2^i + 2^j$ for all $0 \le i < j \le 7$ and use $\mathrm{arr}[2^i + 2^j] = (i, j)$.

D. Binary Grid

#dp

Difficulty — **Normal**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 84 Submitted, 34 Passed
    AC Ratio: 40.476%
  - First Solver: `hyperbolic`, +9min

- Offline Onsite Contest
  - 81 Submitted, 20 Passed
    AC Ratio: 24.691%
  - First Solver: Geunseong Kim @JNU, +26min

D. Binary Grid

- ✓ **Subtask 1**: Complete search. Unfortunately, it is not much easier than solving **Subtask 3**.
- ✓ **Subtask 2**: Collect numbers by following the diagonal line.
- ✓ **Subtask 3**: If you store the maximum binary number $M = M[i][j]$ that can be obtained when reaching every cell, the following ignition equation is established. So DP can solve your problem.

  $$M[i][j] = 2 \times \max(M[i-1][j], M[i][j-1]) + D[i][j]$$

  ($D[i][j]$ is the number written in the row i and column j (0-based index))

E. Making ChatGPT

`#implementation, #graph_theory`

Difficulty — **Hard**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 128 Submitted, 18 Passed
    AC Ratio: 17.969%
  - First Solver: `iktk`, +25min

- Offline Onsite Contest
  - 199 Submitted, 7 Passed
    AC Ratio: 3.518%
  - First Solver: Geunseong Kim @JNU, +70min

- ✓ **Subtask 1:** You can implement the fingerprint almost as is.
  - ✓ The fingerprint states that whenever a letter c is given, all letters c within the training corpus are searched. However, since there are only 29 characters, the search results for all 29 characters are preprocessed. It is recommended to leave it there.
  - ✓ After preprocessing, each time a character comes in, the next character can be predicted in $O(1)$. Implementing this frees up subtask 1. Time complexity is $O(S + K)$. (S is the size of the training corpus)

E. Making ChatGPT

✓

Subtask 2: If the index exceeds a certain value, you need to determine that there is periodicity in the string.

- ✓ Let's assume that the length of the generated sentence s is 30 or more. 29 characters that can appear, so at least 2 of the 30 characters must be the same.
- ✓ Let's find this and say $s[i] = s[j]$. Then $s[i+1] = s[j+1]$, $s[i+2] = s[j+2]$, ..., etc. all hold true. Therefore, the string has periodicity, and the length of the string is infinite.
- ✓ If the length of the string is infinite, the Kth character can be found in $O(1)$ through modulo operation.

E. Making ChatGPT

- ✓ If the length of the string is finite, then by the above argument its length is less than or equal to 29. Therefore, if the input K is more than 30, print a dot (.).
- ✓ It should be noted that there may not be any special periodicity at the beginning of the string.
- ✓ For example, given a training corpus {[gisss]} containing 1 sentence, the generated string is [gisssssss⋯

✓ Time complexity is $O(S)$.

F. Infinitely Popping Bubbles in a Bottle

`#simulation`, `#ad-hoc`, `#(hashmap)`

Difficulty — **Hard**

- Authors: Hyegeun Song @GIST `invrtd_h`, Yoonsu Lee @JNU `lys9546`

- Online Open Contet
  - 25 Submitted, 11 Passed
    AC Ratio: 48%
  - First Solver: `hyperbolic`, +14min

- Offline Onsite Contest
  - 24 Submitted, 0 Passed
    AC Ratio: 0%
  - First Solver: –

F. Infinitely Popping Bubbles in a Bottle

- ✓ **Subtask 1**: Proceed with the simulation by scanning the entire array every second. Time complexity is $O(NT)$.
- ✓ **Subtask 2**: If you think about how high the bubbles created in the 1st zone can go, it doesn't go up that. Even if 100,000 bubbles are created in the first zone, the number of bubbles decreases to $1/5$ every time you go up one zone, so if you repeat this, the number of bubbles quickly becomes 0.

  Therefore, we find the area that the bubbles generated in the first area cannot reach, then exclude that area and run the simulation. At this time, since the number of non-excluded regions is about $O(\log M)$, the time complexity is $O(T \log M)$.

- **Subtask 3:** This task requires generalizing the solution of Subtask 2.
  - Let us assume that bubble generation occurs only once at location $y = Y$ at time $T = 0$.
  - List the (time, location) pairs affected by this bubble generation. Then:
  - $(0, Y) \setminus (1, Y-1), (1, Y), (1, Y+1) \setminus (2, Y-2), (2, Y-1), (2, Y), (2, Y+1), (2, Y+2) \setminus \cdots$
  - Likewise, since the number of bubbles decreases exponentially, there is a limit to the number of (time, position) pairs that can be affected. Its size is approximately $O(\log^2 M)$.
  - In areas not affected by bubble generation, the number of bubbles is always 0.

F. Infinitely Popping Bubbles in a Bottle

  - ✓ Therefore, you only need to manage $O(T \log^2 M)$ cells affected by T number of bubbles.
  - ✓ You can think of a way to use HashMap, etc. to manage only cells with a positive number of bubbles, excluding cells with 0 bubbles.
  - ✓ There is also a solution that does not use HashMap, but it runs much faster than the fixed solution.
- ✓ Time complexity is $O(T \log^2 M)$.

G. Copying Money

`#bellman-ford`, `#parametric-search`

Difficulty — **Hard**

- Authors: Mingyu Ko @JNU `jjkmk1013`, Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 27 Submitted, 5 Passed
    AC Ratio: 18.519%
  - First Solver: `aeren`, +58min

- Offline Onsite Contest
  - 0 Submitted, 0 Passed
    AC Ratio: 0%
  - First Solver: –

G. Copying Money

- ✓ Transactions can be modeled as graphs.
- ✓ If a customer changes item A to item B and loses c in the process, let's give an edge with weight c from A to B.
- ✓ On the contrary, if the customer gains c, let's give an edge with weight $-c$ from A to B.
- ✓ In the case of information transactions 1 and 2 where only one item is involved in the transaction process, let's fill in the empty space with item 0.

G. Copying Money

- ✓ Now, this problem is replaced with the problem of finding negative cycles.
- ✓ We can use the Bellman–Ford algorithm, which is an algorithm for finding negative cycles.
- ✓ However, if the customer does not have enough money, there are cases where a specific edge cannot be used, so to be exact, we will use an algorithm that slightly modifies Bellman–Ford. In the modified algorithm, edge propagation does not occur if the customer does not have enough money.
- ✓ Using the modified algorithm, we can determine whether the customer can reach a negative cycle with the initial funds of c. The validity of the algorithm can be proven by modifying the proof that the standard Bellman–Ford algorithm can legitimately determine the existence of negative cycles.

- ✓ Finally, the minimum amount of money needed to reach a negative cycle can be implemented using binary search. If you can reach a negative cycle with an initial fund of c, you can do it with $c+1$, and if you cannot reach a negative cycle with an initial fund of d, you cannot do it with $d-1$.
- ✓ The time complexity is (Bellman–Ford time complexity) $\times$ (Binary search time complexity) = $O(MN\log C)$.

Appendix: Let's prove the validity.

- ✓ **Argument.** The problem situation can be modeled as DP.
- ✓ Let $\mathrm{dp}[i][j]$ be the largest amount of money that can be held when reaching the item i using at most j exchanges. – Base case is defined as $\mathrm{dp}[0][0] = c, \mathrm{dp}[i][0] = -\infty (i \geq 1)$. c is the initial capital.
- ✓ In this case, the following formula holds:

$$\mathrm{dp}[i][j] = \max(\mathrm{dp}[i][j-1], \max_{v \in S}(\mathrm{dp}[v][j-1] + \mathrm{cost}[v][i]))$$

- ✓ In this case, the set S is the set of all vs that have an edge from v to i and that make $\mathrm{dp}[v][j-1] + \mathrm{cost}[v][i]$ greater than or equal to 0.

G. Copying Money

- ✓ **Claim.** If the DP table is updated after the algorithm has been performed for $n + 1$ rounds, then there is a positive cycle in this graph and the cycle is accessible from vertex 0. – The reason why we use $n + 1$ here is because the number of vertices in the graph is $n + 1$.

- ✓ **Proof.** The update of the value of vertex v means that one of the vertices with an edge to vertex v was updated in the previous round. If we repeat this backtracking, we will eventually form a path starting from vertex 0, which is the only vertex with a non-$-\infty$ value in the 0th round, and going to vertex v, which is updated in the $n + 1$th round. By the pigeonhole principle, this path must have a cycle. Since an update always increases a value, the value assigned to the vertex must increase after one round of this cycle. Therefore, this cycle is a positive cycle.

- ✓ **Argument.** If the DP table is not updated when the algorithm is performed for k rounds, the DP table will not be updated even when the algorithm is performed for $k + 1$ rounds.
- ✓ **Proof.** If no update occurs in round k, then $\mathrm{dp}[i][k] = \mathrm{dp}[i][k-1] \geq \max_{v \in S} \mathrm{dp}[v][k-1] + \mathrm{cost}[v][i]$ and vice versa. Now

$$\mathrm{dp}[i][k+1] = \max(\mathrm{dp}[i][k], \max_{v \in S}(\mathrm{dp}[v][k] + \mathrm{cost}[v][i]))$$

$$= \max(\mathrm{dp}[i][k-1], \max_{v \in S}(\mathrm{dp}[v][k-1] + \mathrm{cost}[v][i]))$$

$$= \mathrm{dp}[i][k-1] = \mathrm{dp}[i][k]$$

We get

- ✓ **Assertion.** When the DP table is not updated after the algorithm has been performed for $n + 1$ rounds, money duplication is impossible. Since the DP

G. Copying Money

table will never be updated in the future according to the above claim, the maximum amount of money that can be earned at each vertex is determined.

H. Elementary Wizardry

`#sorting`, `#greedy`

Difficulty — **Challenging**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 9 Submitted, 6 Passed
    AC Ratio: 66.667%
  - First Solver: `chmpro`, +70min

- Offline Onsite Contest
  - 0 Submitted, 0 Passed
    AC Ratio: 0%
  - First Solver: –

H. Elementary Wizardry

- ✓ **Subtask 1**: Use brute force. It looks like there are 100 million possible x–coordinates and 100 million possible y–coordinates for the top right point of the magic circle, but in reality, we only need to look at N^2 cases where the x–coordinate matches the x–coordinate of some magic circle and the y–coordinate matches the y–coordinate of some magic circle. Therefore, we can solve the problem with a time complexity of $O(N^3)$.
- ✓ **Subtask 2**: Since the y–coordinates of the points are all the same, we sort the points by x–coordinate and continue to increase the x–coordinates until the magic circle includes all the colored magic circles. The time complexity is the time complexity of sorting, $O(N \log N)$.

H. Elementary Wizardry

- ✓ **Subtask 3**: Let's extend the logic of **Subtask 2.** Let's say the magic circle is a part surrounded by four lines $x = 0, x = X, y = 0, y = Y$. When the value of Y is fixed, maximizing the power of the magic circle can be done using the same logic of **Subtask 2.** We keep increasing the value of X, and when we can no longer increase it, we try to decrease the value of Y.
- ✓ The reason why this algorithm is possible is that when the value of Y is fixed, the maximum value of X, $X_{\max}(Y)$, is a non-decreasing function.
- ✓ Each time we decrease the value of Y, we need to decide which point will be removed from the magic circle. Since the y coordinates of the magic circle are removed in order of increasing value, we can create a PQ that manages all the points in the magic circle and can pass with the time complexity $O(N \log N)$.

I. Operating DevNight

`#dijkstra`

Difficulty — **Challenging**

- Author: Hyegeun Song @GIST `invrtd_h`

- Online Open Contet
  - 20 Submitted, 5 Passed
    AC Ratio: 25%
  - First Solver: `chmpro`, +97min

- Offline Onsite Contest
  - 0 Submitted, 0 Passed
    AC Ratio: 0%
  - First Solver: –

- ✓ **Subtask 1**: If you run Dijkstra's algorithm once for a starting vertex, you can find out the length of the shortest path to all other vertices. Therefore, you can run Dijkstra's algorithm once for each of the K communication rooms as a starting point.
- ✓ When Dijkstra's algorithm is implemented in $O(M \log M)$, the time complexity is $O(KM \log M)$. Subtask 1 is sufficient because K is small.
- ✓ **Subtask 2**: After creating a path, sweep along the path and manage up to 2 communication rooms on both sides of each vertex. The second closest communication room is one of the 4, 2 on the left and 2 on the right of each conference room.

- ✓ **Subtask 3**: In addition to the solution explained in the editorial, there is one more solution for the reviewer. The reviewer's solution is probably easier. – Some prior knowledge of Dijkstra's algorithm is required.
- ✓ **Multisource Dijkstra**: Let's assume that there are multiple starting vertices and that you can start from any of them. Even in this situation, you can find the length of the shortest path to the destination vertex.
- ✓ **Proof**: You can prove this by replacing Multisource Dijkstra with the equivalent Standard Dijkstra problem. Let's randomly add a vertex and connect that vertex and all starting vertices with a unidirectional edge of length 0. Then, let's make the added vertex the new starting vertex.

- ✓ The problem requires the shortest path from each conference room to one of the communication rooms, but from the perspective of Multisource Dijkstra, it is more efficient to think of the shortest path from one of the communication rooms to each conference room. Therefore, from now on, we will call the communication room the starting vertex.

- ✓ **Shortest Path Tree:** A tree consisting of the shortest paths starting from the root node to each vertex.
- ✓ The Shortest Path Tree always exists and can be obtained during the process of running Dijkstra's algorithm. In each round of the algorithm, the vertex with the smallest assigned value among the vertices whose shortest path length is not yet determined is determined.
- ✓ At this time, the newly determined vertex may have been influenced by one of the vertices whose shortest path length has already been determined. If so, connect the influenced vertex and the influenced vertex.
- ✓ The shortest Path Tree is not unique. However, in this problem, people prefer communication rooms with smaller vertex numbers when the shortest path length is the same, so the Shortest Path Tree is unique.

- ✓ Multisource Dijkstra also has something similar to Shortest Path Tree.
  However, this time, it exists as a Forest (with multiple Trees) instead of a Tree.
  If you remove the vertex added in the proof above, you can obtain a Forest.
- ✓ At this time, **each forest has only one of several starting vertices (communication rooms) as its root node.**
- ✓ You can group the vertices based on which forest they belong to.

- ✓ Now, let's think about a set of vertices V_k that have the kth communication room as their root node.
- ✓ We need to find the closest communication room among the communication rooms that are **not** the kth communication room for all vertices in V_k.
- ✓ You can do this by running Dijkstra again with the vertices adjacent to V_k as the starting point. When you push the adjacent vertices to pq, you also include the length of the shortest path.
- ✓ For example, let's say that the vertices adjacent to V_k are vertices 10 and 11, and the distance from vertex 10 to the closest communication room is 100, and the distance from vertex 11 to the closest communication room is 110. Then, we first put (10, 100) and (11, 110) in PQ.

- ✓ We can prove that this algorithm works in the same way as the standard Dijkstra by creating a new vertex and connecting the starting vertices to that vertex, just like the proof of multisource Dijkstra. The distance of the newly created edges is the distance to the nearest communication room.
- ✓ We just need to run Dijkstra for all V_k.
- ✓ If we run dijkstra K times like this, the time complexity will be $O(KM\log M)$. However, if we think about it carefully, we do not need information about the entire graph when we run Dijkstra once. We only need information about V_k, its adjacent vertices, and the edges between the vertices.
- ✓ In this regard, if we do a good job of optimizing the size of the graph, we can reduce the time complexity to $O(M\log M)$.