A – Alice’s Astounding Alchemy

Luo Tsz Fung {pepper1208}

2025-07-08

Background

Problem Idea by `pepper1208`

Preparation by `pepper1208`, `rina__owo`

Problem Restatement

Find the number of pairs of number such that their sum is a square number.

Solution

Direct simulation is sufficient for **AC**.

Set up two for-loops to traverse all possible pairs of numbers.

Then, check if their sum is a square number.

- There are many ways to check if a number is a square number. One of the methods is to check whether the product of the square root of the number to itself is the original number or not.

Output the counter storing the number of valid pairs of numbers.

Time complexity: $O(N^2)$

Takeaways

- You must have a good sense to know which question is relatively easier and AC those questions first.
- You should AC these type of questions fast to gain advantages.

B – Binary Sequence

Luo Tsz Fung {`pepper1208`}

2025-07-08

Background

Problem Idea by `pepper1208`

Preparation by `pepper1208`, `rina__owo`

Problem Restatement

Given a sequence from 1 to N. Perform binary search on each element 1..N on the sequence. Find the number of occurrence of the variable M in the binary search where M is a given integer x.

Solution

Brute force is not applicable to AC this question.

Nevertheless, we can output all answers from smaller case to seek for any pattern(s).

Solution

Input:

10 10

1

…

10

Output:

1 4 2 1 10 2 1 5 2 1

Solution

Input:

```
4 4
1
…
4
```

Output:

```
1 4 2 1
```

Solution

Input:

```
5 5
1
…
5
```

Output:

```
2 1 5 2 1
```

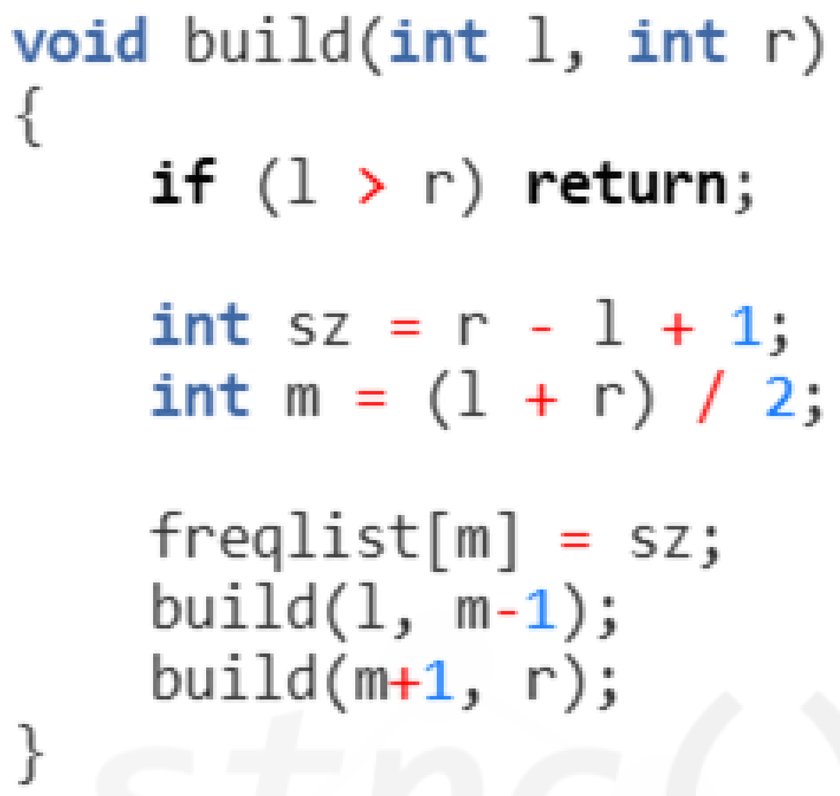
Solution

You can see that the answer of “10” consists of answer of “4” and answer of “5”.

Therefore, a recursive-like answer can be formed!

The proof is left for exercise.

Time complexity: O(N + Q) (pay attention to the size of the recursion tree)



```
void build(int l, int r)
{
	if (l > r) return;

	int sz = r - l + 1;
	int m = (l + r) / 2;

	freqlist[m] = sz;
	build(l, m-1);
	build(m+1, r);
}
```

Takeaways

- You should be aware of the constraints to design a suitable algorithm.
- Pattern-seeking is of paramount importance for ad-hoc problems.

C – Convolution Neural Network

Chin Ka Wang {rina__owo}

2025-07-08

Background

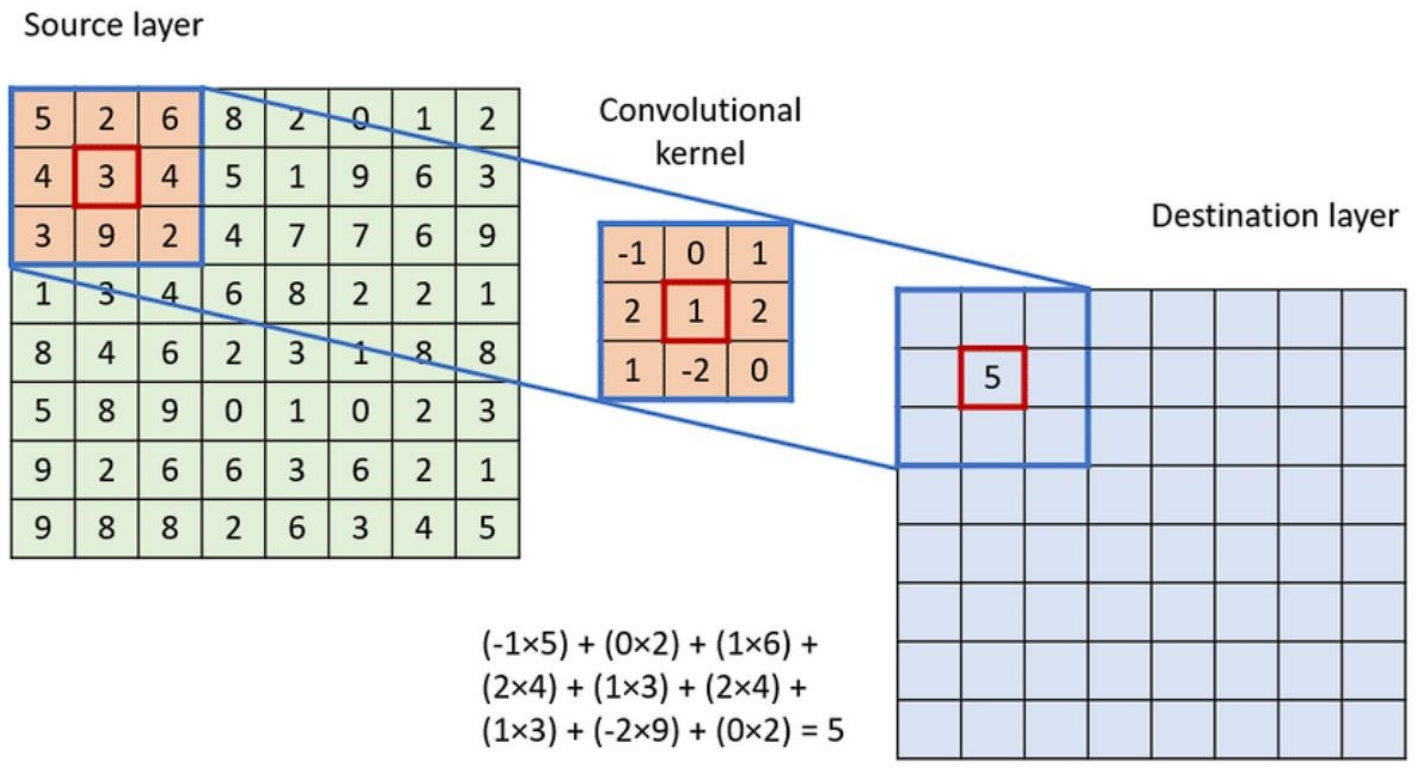
Problem Idea by `rina__owo`

Preparation by `rina__owo`, `pepper1208`

Problem Restatement

Given a gray scale image represented by a N * M number grid and a kernel represented by a K * K number grid. Do a convolution operation on the image by the kernel. Output the convolved image.

Solution



Source layer

5	2	6	8	2	0	1	2
4	3	4	5	1	9	6	3
3	9	2	4	7	7	6	9
1	3	4	6	8	2	2	1
8	4	6	2	3	1	8	8
5	8	9	0	1	0	2	3
9	2	6	6	3	6	2	1
9	8	8	2	6	3	4	5

Convolutional kernel

-1	0	1
2	1	2
1	-2	0

Destination layer

	5						

$$(-1\times5) + (0\times2) + (1\times6) + (2\times4) + (1\times3) + (2\times4) + (1\times3) + (-2\times9) + (0\times2) = 5$$

Solution

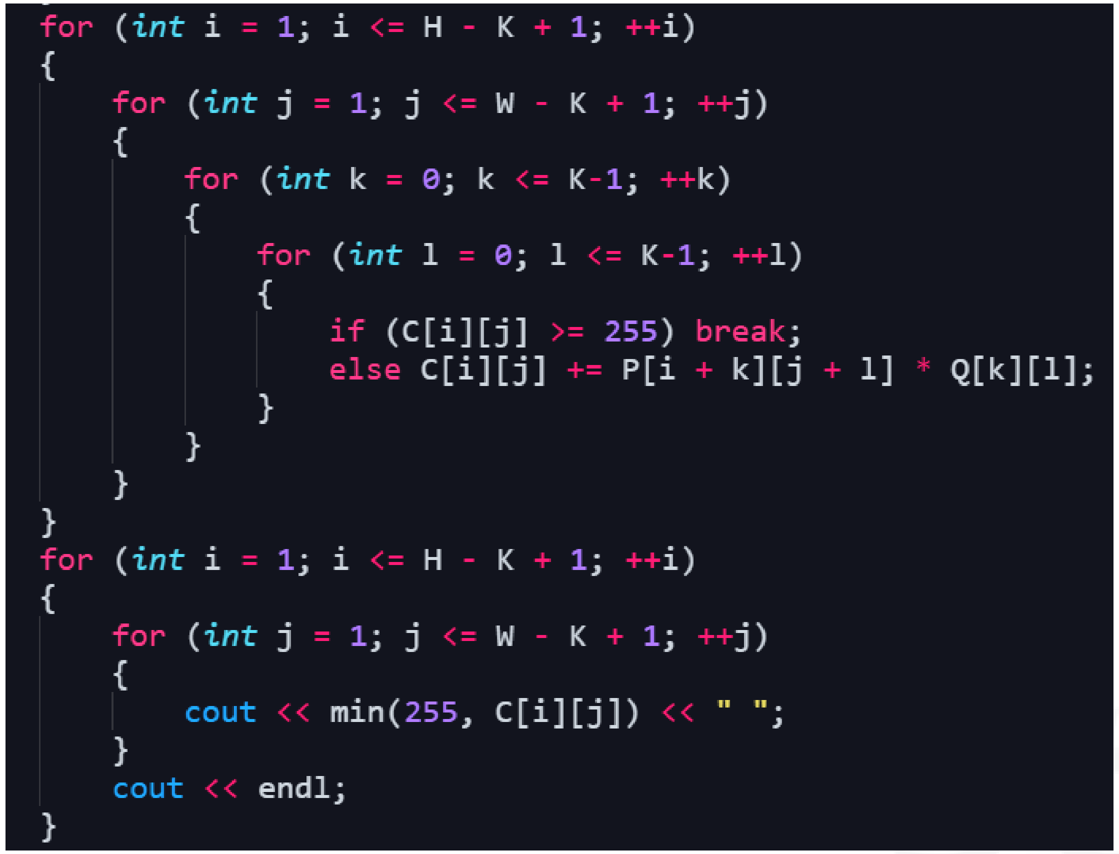
- The only difficulty in this question is to understand what the formula means.

$$C[x][y] = \min\left(255, \sum_{i=0}^{K-1}\sum_{j=0}^{K-1}(P[x+i][y+j] \times Q[i][j])\right)$$

Solution

- We can just completely follow the formula and use a four-layer for loop to simulate the operation.

Solution



```
for (int i = 1; i <= H - K + 1; ++i)
{
    for (int j = 1; j <= W - K + 1; ++j)
    {
        for (int k = 0; k <= K-1; ++k)
        {
            for (int l = 0; l <= K-1; ++l)
            {
                if (C[i][j] >= 255) break;
                else C[i][j] += P[i + k][j + l] * Q[k][l];
            }
        }
    }
}
for (int i = 1; i <= H - K + 1; ++i)
{
    for (int j = 1; j <= W - K + 1; ++j)
    {
        cout << min(255, C[i][j]) << " ";
    }
    cout << endl;
}
```

Solution

- Make sure to break the loop properly, or the result number might be added up to a very big number larger than the long long range, resulting in an abnormal of the result. (The number might become a negative number)

Takeaways

- Substitute numbers in the formula to understand it well.
- Break loops properly.

D – Decode IV

Chin Ka Wang {`rina__owo`}

2025-07-08

Background

Problem Idea by `rina__owo`

Preparation by `rina__owo`, `pepper1208`

Problem Restatement

Given a string consists of only digits, output the number of valid permutation of the string. The string is valid when the product of consecutive digits are always even.

Solution

- This is a simple math question requires knowledge of combination and permutation.

formula to find number of different permutations of n distinct objects taken r at a time is

$$P(n,r) = \frac{n!}{(n-r)!}$$

formula to find number of different combinations of n distinct objects taken r at a time is

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!\,(n-r)!}$$

Solution

- To ensure the product of consecutive digits are always even, the two digits must not be both odd.
- Thus, we can transform the question to first permutate even digits, then insert odd digits between the even digits. Don't forget that you can insert before the first even digit the after the last even digit. i.e. if there are n even digits and m odd digits, you can have n+1 place for inserting odd digits, i.e. P_m^{n+1}.
- It's not hard to realize that if number of odd digit > n+1. It's impossible to construct a valid string. In this case the answer is 0.

Solution

- As there might have repeat digits, we have to divide them to avoid repetitive calculation.
- Define k_n be the appearance of the digit n.
- The final formula of the answer would be:

$$\frac{(n+1)!\,P_m^{n+1}}{\prod_{i=0}^{9} k_n!}$$

- Build a function calculating factorial by recursion and we can get the answer by the formula. Then, we can AC the question.

Takeaways

- Some transformation of the question is often essential. It can greatly lower the difficulty of the question.
- If you are strong at math, please try more using math to solve OI questions. It can help you save lots of time and the time complexity of your code is often satisfactory.
- If you are not good at math, you can still try to brute force the solution to get partial scores. Don’t give up on these kind of questions.

E – Escape Byteland

Luo Tsz Fung {pepper1208}

2025-07-08

Background

Problem Idea by `pepper1208`

Preparation by `pepper1208`, `rina__owo`

Problem Restatement

Given a graph with the edge either “normal” or “hidden”.

You can pass through any normal roads, and any hidden roads afterwards.

Find the shortest path from node 1 to node N.

Solution

There are many approaches to **AC** this question!

One of the methods is building two graphs with either consists of normal roads only and hidden roads only.

Then, build all edges from a node in “normal graph” to the same node in “hidden graph”.

Afterwards, simply BFS and you can AC!

Time complexity: O(N + M)

Takeaways

- You need to have a flexible graph modeling skills.
- Draw some smaller cases will help you understand the question better!
- Be familiar with different common graph modelling techniques.

F – Flip Phone

Luo Tsz Fung {`pepper1208`}

2025-07-08

Background

Problem Idea by `pepper1208`

Preparation by `pepper1208`, `rina__owo`

Problem Restatement

Simulate the keyboard of a flip phone.

Solution

You can build an array to store the number of times pressing button(s) for each character.

Nevertheless, you can use a map to integrate the feature.

As there are only 26 lowercase letters, the space complexity is sufficient.

Time complexity: O(|S|)

Takeaways

- Easy simulation problem! You should quickly simulate the problem and gain advantages!
- Using various simple data structures helps you code the algorithm faster.

stpc();

G – Gathering

Chin Ka Wang {`rina__owo`}

2025-07-08

Background

Problem Idea by `rina__owo`

Preparation by `rina__owo`, `pepper1208`

Problem Restatement

There is a N*M grid, and you have to perform several operations. You may either asked to add a robot at a specific coordinate, remove a specific robot that already exists in the gird, or output a valid gathering spot.

A valid gathering spot is the spot where the sum of the Manhattan distance between each robot and the spot is the least.

Solution

- Observe the formula of Manhattan distance, you may find that it’s actually the sum of horizontal and vertical distance.
- For the sum of Manhattan distance to be the least, it’s not hard to figure out that it is equivalent to the sum of all horizontal and all vertical distance to be the least.
- As the x-coordinate and y-coordinate of the gathering spot will only affect total horizontal and vertical distance respectively, we can consider them respectively.

Solution

- Trivially, the tricks of considering horizontal total distance and vertical are actually the same.
- If you remember the Post Office Problem taught in DP(II), it's not hard to realize that:
  - The required coordinates are the median of the horizontal and vertical coordinates of the robots respectively.
  - If the required median is not an integer, the coordinate should be the nearest integer coordinate to the median (must be ± 0.5 to both x and y coordinate).

Solution

- To effectively find the medians online (having insertion, deletion or modification of data between queries), we can simply maintain two double-ended heap, one for horizontal coordinates and one for vertical. i.e. four priority queues.
- Every time we need to add or remove a robot, we can just insert or delete its x and y coordinate into the double-ended heaps resp.
- The middle element of the double-ended heap would be the median of all elements in the double-ended heap. O(1) time is needed for such a query.

Takeaways

- When seeing questions finding median or the k-smallest/k-greatest element, double-ended heaps are always used.
- Try recalling what you have learnt in questions you done before. Every questions you did is meaningful.

H – House Meeting

Chin Ka Wang {rina__owo}

2025-07-08

Background

Problem Idea by `rina__owo`

Preparation by `rina__owo`, `pepper1208`

Problem Restatement

Given N numbers numbered 1 to N in a specific order. You can swap any two numbers each time. Output at least how many swaps are needed to turn the initial order to another desired order.

Solution

- The main idea of solving this problem is related to group theory. However, it is simple to understand.

Solution

- Assume the initial order is completely different from the desired order. i.e. there are no numbers already match the desired order. If there are N numbers, it's not hard to observe that we need at most N – 1 swaps to turn the initial order to the desired order, since each swap can make a number to match its desired position, and the remaining numbers forms another "swapping question" of N - 1 numbers. When there are only two numbers left, only one swap is needed to simultaneously fix two numbers' position.

Solution

- However, the same situation might also happen when there are many numbers left, not only two numbers. Therefore, we need to find “cycle” of numbers. Which a “cycle” is defined as a set of subsequence of the given sequence that not such situation will occur.

Solution

Initial: A E B C F D G

Target: B C E F A G D

Loop 1: A → B → E → C → F → A (Required no. of swaps = 4, length of cycle = 5)

Loop 2: D → G → D (Required no. of swaps = 1, length of cycle = 2)

Solution

- As number of swaps needed in one cycle = length of cycle – 1

- Formula of calculating the answer can now be derived:
- Minimum no. of swaps needed to transform to target sequence = length of cycle – number of cycle

- Using simple loops can retrieve all needed information simply.

Takeaways

- Like solving the DP problems, breaking ad-hoc problems into smaller subproblems sometimes also give you inspiration of solving the whole problem.

I – Ice Cream

Luo Tsz Fung {`pepper1208`}

2025-07-08

Background

Problem Idea by `pepper1208`

Preparation by `pepper1208`, `rina__owo`

Problem Restatement

You want to eat N ice cream. 3 ice cream sticks can exchange a new ice cream.

Find the minimum number of ice cream you need to buy initially.

Solution

Brute force is not applicable to **AC** the question.

Note that when *N* increases, the minimum number of ice cream bought initially must increases.

Therefore, **monotonicity** occurs! → Binary search on answer!

Binary search on the minimum number of ice cream bought initially and find the answer!

Time complexity: O(log N)

Alternative solution

Why don't we output some small test cases and seek for any possible pattern(s)?

1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	3	4	5	5	6	7	7	8	9	9

We can find a great pattern and we can use a simpler approach to tackle this problem!

Alternative solution

```
long long a, s;
cin >> a;
if (a % 3 == 0) ++a;
s = a - a / 3;
cout << s << '\n';
```

Time complexity: O(1)

Takeaways

- Be aware of any monotonicity in the situation. Binary search (or binary search on answer) often helps!
- Mathematical approach may help to reduce time complexity!

A – Just Buy Some Stocks

Chin Ka Wang {rina__owo}

2025-07-08

Background

Problem Idea by `rina__owo`

Preparation by `rina__owo`, `pepper1208`

Problem Restatement

Given N initial capitals and M stocks, each stock have a cost and a profit.

Each time you buy a new stock, the total profit of all stocks you bought will add up to your saving. Find the maximum saving you can get.

+1

+1 +3

+1 +3 +7

+1 +3 +7 +12

Solution

- It’s not hard to observe that this question is extremely similar to a standard 0-1 knapsack problem.
- The only difference is that the profit we gain each time is not just the profit of the stock we buy, but the total profit of all stocks we have bought before.

Solution

- To tackle the "cumulative profit" part, we just need to open another array, where each cell in the array is attached to a corresponding state in the DP array, storing the corresponding cumulative profit on that state. Thus, the transitional formula becomes:

$$dp[i][j] = \max(dp[i-1][j], dp[i-1][j-c_i] + cp[i-1][j-c_i] + p_i)$$

- Here, the array cp stores the cumulative profit of each state. If the latter state is greater, update $cp[i][j]$ to be $cp[i-1][j-c_i] + p_i$, else update $cp[i][j]$ to be $cp[i-1][j]$.

Solution

- After rolling array optimization, the transitional equation becomes:

$$dp[i] = \max(dp[i], dp[i - c_i] + cp[j - c_i] + p_i)$$

- If the latter state is greater, update $cp[i]$ to be $cp[j - c_i] + p_i$.
- In this case, calculation order is from N to 1.

Solution

- However, these are not enough for us to get the answer.
- For the same stocks, different buying order might result in different answer.
- Our DP method didn’t consider this. We process the stocks with the input order, which is unsatifactory.

Solution

- Assume we have to buy four stocks with profit 1, 2, 3 and 4, what should be the best buying order?
- Trivially, we should buy the stocks in descending order of profit so that we can earn the greatest profit.

```
+4                    +1
+4 +3                 +1 +2
+4 +3 +2      >       +1 +2 +3
+4 +3 +2 +1           +1 +2 +3 +4
```

Solution

- Therefore, we just need to sort the stocks in descending order of profit. So that stocks with larger profit can be ensured to be processed first by our DP program.
- This idea is actually **Greedy Algorithm**!

Takeaways

- Sometimes question might require both greedy and DP. They are similar, don’t mix them up.
- Questions finding minimum/maximum are often DP problems. Do more similar questions to train your sense.