Problem Tutorial: “Yes, Prime Minister”

Obviously $r > 0, r \geq l$. If $l \leq 0$, then $\sum_{i=l}^{r} i = \sum_{i=-l+1}^{r} i$ and $r \geq -l+1 > 0$. Therefore, for any interval $[l_1, r_1]$ there must exist an interval $[l_2, r_2]$ where $\sum_{i=l_1}^{r_1} i = \sum_{i=l_2}^{r_2} i$ and $r_2 \geq l_2 > 0$.

$$\sum_{i=l}^{r} i = \frac{(l+r)(r-l+1)}{2}$$

When $l > 0$, if $r - l \geq 2$, then at least one of $\frac{l+r}{2}$ and $\frac{r-l+1}{2}$ is an integer greater than 1, which means $\sum_{i=l}^{r} i$ is the product of two factors greater than 1. So intervals with a prime summation must have a length of no more than 2.

Precalculate all prime numbers in $[2, 2.001 \times 10^7]$. Both of Eratosthenes sieve and Euler sieve are OK.

If $x \leq 0$, then candidate answers are:

- $[-y+1, y]$ where y is the minimum integer satisfying $y \geq 1 - x$ and y is a prime number.
- $[-z+2, z]$ where z is the minimum integer satisfying $z \geq 2 - x$ and $2z - 1$ is a prime number.

If $x > 0$, then candidate answers are:

- $[x, x]$。
- $[x, x+1]$。
- $[x-1, x]$。
- $[-y+1, y]$ where y is the minimum integer satisfying $y \geq x$ and y is a prime number.
- $[-z+2, z]$ where z is the minimum integer satisfying $z \geq x$ and $2z - 1$ is a prime number.

Binary search is adequate with a time complexity of $O(T \log(|x|) + |x|)$. You can also optimize it to $O(T + |x|)$.

Problem Tutorial: “Might and Magic”

The answer is equal to the maximum damage you do to your enemy when his HEALTH is infinite. Considering that your turn of survival $t = \lceil \frac{H0}{Damage} \rceil$ has $O(\sqrt{H_0})$ different values, you can enumerate t and D_0 you must cost.

Then we have a conclusion: you must distribute all remain attribute points to physical attack(A_0) or to magical attack(P_0, K_0). We state the proof as follows:

Let’s fix the times you make physical and magical attack $a, b(a + b = t)$. If $b > K_0$ then we will do nothing in $b - K_0$ turns.

If we distribute x points to A_0, then our physical damage function $P(x) = \begin{cases} aC_p & x <= D_1 \\ aC_p(x - D_1) & x > D_1 \end{cases}$ is a convex function.

If we distribute x points to P_0, K_0, then our magical damage function $M(x) = \begin{cases} C_m(x - \lfloor \frac{x}{2} \rfloor)\lfloor \frac{x}{2} \rfloor & 2x <= b \\ C_m(x - b)b & 2x > b \end{cases}$ is also a convex function.

When we distribute x points to A_0 and $N - D_0 - x$ to P_0, K_0, the total damage function $F(x) = P(x) + M(N - D_0 - x)$ is a convex function too, which means its maximum value is $\max(F(0), F(N - D_0))$.

In physic precedence cases, distribute all $N - D_0$ points to A_0.

In magic precedence cases, the optimal distribution can be solved by calculating the maximum value of $D(K_0) = C_m K_0(N - D_0 - K_0) + C_p(t - K_0)(0 \le K_0 \le \min(t, N - D_0))$.

Time complexity is $O(T\sqrt{H_0})$.

Problem Tutorial: "0 tree"

Let's define a new vertex weight $b_i = \sum_{e=\langle i,v\rangle} b_e$. We find that $\forall e \in E, b_e = 0 \iff \forall i \in V, b_i = 0$. It can be proved from leaves to root.

When performing operation x y w, we can notice that no weight will change except weight of endpoints a_x, a_y, b_x, b_y.

Define $dis(x, y)$ as the minimal number of edges travelling from x to y. When $dis(x, y)$ is even, the operation x y w is equivalent to:

$$a_x \leftarrow a_x \bigoplus w; \quad a_y \leftarrow a_y \bigoplus w; \quad b_x \leftarrow b_x + w; \quad b_y \leftarrow b_y - w$$

When $dis(x, y)$ is odd, the operation x y w is equivalent to:

$$a_x \leftarrow a_x \bigoplus w; \quad a_y \leftarrow a_y \bigoplus w; \quad b_x \leftarrow b_x + w; \quad b_y \leftarrow b_y + w$$

Color the tree(a bipartite graph too) with $0, 1$. There exists no solution if any of these conditions is satisfied:

- $\bigoplus_{col_i=0} a_i \ne \bigoplus_{col_i=1} a_i$。
- a_i, b_i have different parity because a_i, b_i's parity always change synchronously.
- $\bigoplus_{col_i=k} a_i > -\sum_{col_i=k} b_i$. Operations between vertices with same color do no effect on the XOR sum of a_i or sum of b_i. So operations between vertices with different color must change both weight to zero. Several non-negative integers' sum is greater or equal to their XOR sum.

Let $A = \bigoplus_{col_i=0} a_i = \bigoplus_{col_i=1} a_i, B = \sum_{col_i=0} b_i = \sum_{col_i=1} b_i$. First change A, B to 0. It can be done by choosing any two vertices x, y with different colors and making 3 operations x y w with $w = A, -(A + B)/2, -(A + B)/2$.

For any two vertices x, y with same color, we can make two operations x y w, without changing a_i, letting:

$$b_x \leftarrow b_x + 2w; \quad b_y \leftarrow b_y - 2w;$$

We can also make two operations x y w followed by one operation y x 2w, without changing b_i, letting:

$$a_x \leftarrow a_x \bigoplus 2w; \quad a_y \leftarrow a_y \bigoplus 2w;$$

Reduce any of both vertex weight to zero will cost $n - 2$ operations. Then use the solution mentioned above to reduce the others to zero. $2n - 4$ or $3n - 6$ operations are needed according the way you choose. In total you will make at most $3n - 3$ or $4n - 5$ operations, satisfying the limit of $4n$.

When $n = 1$ we may need special check.

Complexity: $O(n)$

Problem Tutorial: "Decomposition"

Consider constructing a Hamiltonian cycle in Fig. 1 left: we put a blue vertex at the center and $n-1$ black vertices uniformly on a circle around the center. Starting from the blue vertex, we visit black vertices in a zigzag pattern shown in Fig. 1 and return to the blue vertex in the end. We can repeat rotating the circle by $2\pi \cdot \frac{2}{n-1}$ radius to generate $\frac{n-1}{2}$ distinct Hamiltonian cycles (see Fig. 1 middle and right). We can show that concatenating these cycles would result in an Euler tour of the complete graph, and every $n-2$ continuous vertices in the Euler tour are pairwise different. Thus partitioning the Euler tour sequentially into paths would yield a satisfying answer.

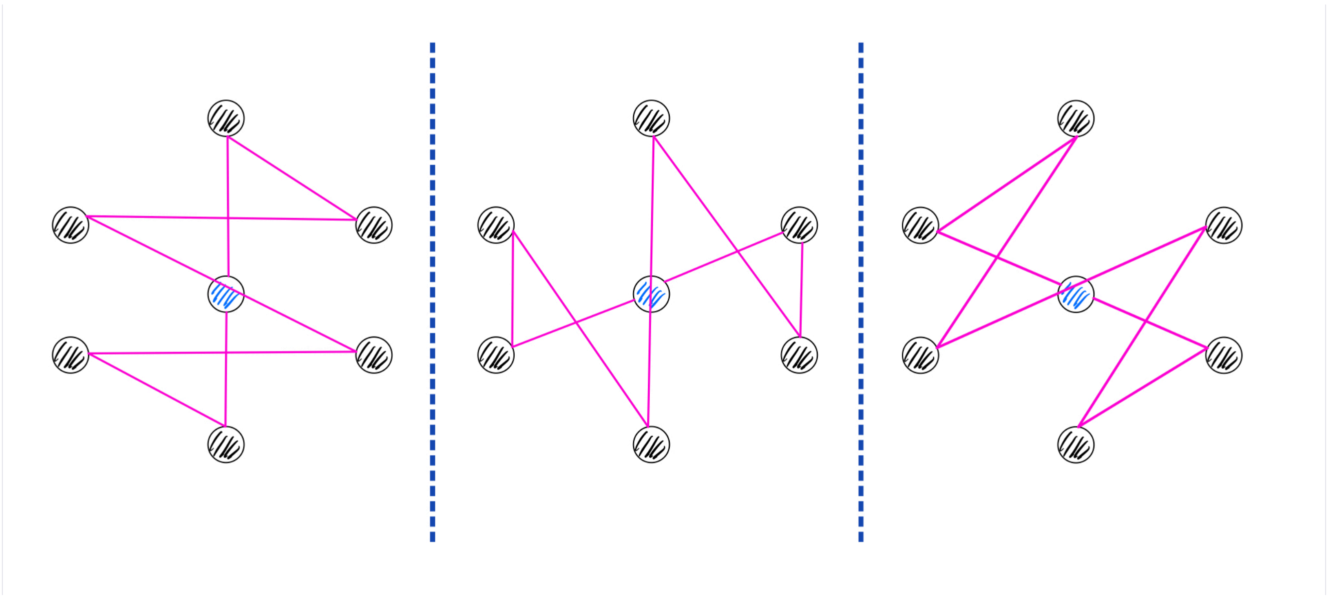
Figure 1: Construction

More formally, we can treat black points as elements of $\mathbb{Z}_{n-1}$ and label them clockwise. We construct the zigzag pattern by selecting a starting value $v(0 \le v < \frac{n-1}{2})$ and then adding (in $\mathbb{Z}_{n-1}$) $+1, -2, +3, -4, \ldots, +(n-2)$ to the current value. The set of added values is exactly $\{\pm 1, \pm 2, \ldots, \pm \frac{n-1}{2} - 1, \frac{n-1}{2}\}$ modulo $n-1$, covering all possible distance between vertices. As in undirected graphs, distances $\pm i (1 \le i < \frac{n-1}{2})$ can be paired together and there are only $\frac{n-1}{2}$ edges with distance $\frac{n-1}{2}$, the $\frac{n-1}{2}$ starting values will guarantee that every undirected edge would be covered exactly once, resulting in an Euler tour. In our construction, one vertex is visited again in at least $n-1$ steps, thus every $n-2$ continuous vertices in the Euler tour are pairwise distinct.

Remark: the construction of the Euler tour can also be applied to partition complete graphs into perfect matchings.

Problem Tutorial: "Median"

For each $i \in [1, n]$, if $a_i > 0$ we call it a good object, otherwise we call it a bad object.

For two common objects of indexes x and y $(x < y)$, they are in the same group if and only if $\forall i \in [x, y], i$ is bad object. We can get all groups of common objects and sizes of those groups easily.

1. Two common objects from different groups can be paired 2. A common object i can be paired with a precious object j if $j \le i$

An arrangement of objects exists if and only if you can find a method to pair them such that all common objects are paired(not necessary for precious objects). So we know that the arrangement is impossible if and only if there is a group of common objects T such that $|T| > P + Q$

Here suppose the indexes of objects in T forms the segment $[l, r]$, P is the total number of common objects and Q is total number of precious objects with index i such that $i \le l$

Problem Tutorial: "The Struggle"

The algorithm complexity of this question is $O(n \log n)$, where $n = \max_{(x,y) \in E} \max(x, y)$. Consider using the FWT algorithm for calculation. We consider xor convolution which can only process one at a time a$[x \times 2^n, x \times 2^n + 2^n - 1] \times [y \times 2^n, y \times 2^n + 2^n - 1]$ square. We first process all the largest squares that are all inside the ellipse, and then process the next largest squares, and so on...

But the complexity of this algorithm is $O(n \log^2 n)$, which is not fast enough. Consider optimizing this algorithm. The method is to perform FWT from the bottom up, and calculate the squares that need to be calculated at each layer. After calculating the inner product of FWT array, we should not calculate the inverse FWT, but should "accumulate" it on the result array. (See author's solution for better understanding)

One issue in the complexity analysis of this question is to prove that the sum of the side lengths of all squares is $O(n \log n)$. This fact can be proved on the condition that the border function is a monotone function, and the boundary of the ellipse can be split into four monotone functions. The idea of the proof is to see that the y-intervals corresponding to each x-interval must be a constant plus some "extra" intervals, and for x-coordinate intervals of the same size, the total length of the "extra y-intervals" cannot exceed n. Since there is only $\log n$ sizes for x-intervals, the proof is done.

Problem Tutorial: "Power Station of Art"

First of all, we observe that the operation is reversible, which means that we only need to operate the first graph. Then we found that for the graph to be equivalent, all connected components has to be equivalent.

Consider a connected component. First of all, the multiset of the numbers in this component in the two graphs must be the same. Then we can find that the operation can be stated in another way: swap the numbers and colors of the two vertices, and then flip both colors. The result of this operation and the original operation will be the same.

In this case, if a number is swapped an odd number of times, its color will be flipped, and if it is swapped an even number of times, the color will not be flipped. Then we will discuss the situation according to the situation:

- Case One: The graph is a bipartite graph. Then we can color the graph in black and white, and we observe that for a number, if we know the initial black or white color of the node and the color of the number, after we swap this number to any position, the color of the number is fixed. Moreover, since the graph is connected, the numbers can be arbitrarily arranged on all nodes. Then the requirement in this case is to divide the number into two multisets according to (node color$\hat{}$number color). The two multisets of the two graphs must be the same.

- Case Two: The graph is not a bipartite graph. In this case, there will be an odd cycle in the component. The requirement in this case is that the multisets of all numbers in the input two graphsmust be the same, and the parity of the number of colors cannot be changed. This is because we can think of the graph as a bipartite graph with some extra edges. We can construct a scheme for any odd ring so that all the numbers do not move, and only the colors of two positions are reversed, regardless of those two positions, without regard to the color of the nodes. The scheme is: for the points on the odd ring numbered sequentially from 1 to n, first perform n-1 operations to exchange the numbers on 1 to node n, and then perform another operation to exchange the numbers on nodes 1 and n, and then Use n-2 operations to exchange the numbers on node n to node 2.

Problem Tutorial: "Command and Conquer: Red Alert 2"

Consider a sniper with a range of k, then the area that can be attacked is a cube. Then we might as well restate the problem, saying that the area that a sniper can attack from (x, y, z) is all points (x', y', z') that satisfies

$x \le x' \le x+2k, y \le y' \le y+2k, z \le z' \le z+2k$. Then we will find that if the coordinates of a certain dimension of our sniper are smaller than the minimum coordinates of the dimension of the remaining points, we should move the sniper to the minimum coordinates of that dimension, because this will only make the targets for sniping increase. When our three dimensions are all the smallest coordinates of that dimension, the sniper must eliminate all enemy forces with the smallest coordinates in one of the three dimensions before the sniper can continue to move. At this time, we can greedily choose the enemy with the smallest k needed to kill. The complexity is $O(n \log n)$.

Problem Tutorial: "Typing Contest"

To avoid float number calculation, let's multiply f by 100. Now, the speed of 1st student is calculated by $\frac{1}{10000}s_1 \times (10000 - f_1f_2 - f_1f_3 - \cdots - f_1f_k)$. For convenience, the coefficient $\frac{1}{10000}$ will be omitted in the following discussion.

The speed of student 1 is $s_1 \times (10000 - f_1(\sum_{i \in S} f_i - f_1))$, S means the set of selected students here. If $\sum_{i \in S} f_i$ is fixed, Then speed of each student in the set is decided. So let's enumerate $\sum_{i \in S} f_i = F$, the problem is changed to: the weights of students are f_i,values of students are $s_i \times (10000 - f_i(F - f_i))$, solve the 01 backpack problem that takes as much value as possible within total weight F.

Actually, we don't need to enumerate F from 1 to $\sum_{i=1}^{n} f_i$. In fact, enumerating F from 1 to $100\sqrt{n+2}$ is enough. The following is the proof, and total time complexity is $O(5000 \times n^2)$。

Proof: suppose we've selected k objects, for any object t , we have (here f_i is multiplied by 100):

$$s_t(10000 - f_t(\sum_{i=1}^{k} f_i - f_t)) > 0$$

Then:

$$f_t \sum_{i=1}^{k} f_i - f_t^2 < 10000$$

For $t = 1, 2, \cdots, k$ add them (We call the equation below A):

$$(\sum_{i=1}^{k} f_i)^2 - \sum_{i=1}^{k} f_i^2 < 10000k$$

Since:

$$f_t \sum_{i=1}^{k} f_i - f_t^2 < 10000$$

Therefore:

$$f_t^2 \sum_{i=1}^{k} f_i - f_t^3 < 10000 f_t$$

For $t = 1, 2, \cdots, k$ add them (We call the equation below B):

$$\sum_{i=1}^{k} f_i^2 < 10000 + \frac{\sum_{i=1}^{k} f_i^3}{\sum_{i=1}^{k} f_i} \le 20000$$

consider A and B, we have:

$$(\sum_{i=1}^{k} f_i)^2 < \sum_{i=1}^{k} f_i^2 + 10000k < 10000k + 20000$$

So:

$$(f_1 + f_2 + \cdots + f_k) < 100\sqrt{k+2} \le 100\sqrt{n+2}$$

Problem Tutorial: "Array"

Notice: For some implementations, maybe you need to manage the special cases when $B_i < i$ or $B_1 = n+1$ manually.

For an array A, let C_i be the rightmost position x in A such that $x < i$ and $A_x = A_i$. If there is no identical number in front of A_i, let $C_i = 0$

If we find the array C that satisfies the given array B, we can construct the array A

C has to satisfy the following conditions(for convenience, let $B_0 = B_1$):

1. $\forall i \in [1,n]$,if $i = 1$ or $B_i > B_{i-1} \wedge B_i \neq n+1$,then C_{B_i} **must be** $i-1$. Let those B_i be **special positions**

2. For all non-special positions i, C_i **can be** set to x if and only if $x < i$ and $B_x > i$

3. There **can be** more than one i that $C_i = 0$

4. $\forall i \in [1,n], B_i = n+1$, there **must be** zero or one x such that $C_x = i-1$

5. $\forall i \in [1,n], B_i < n+1$, there **must be** exactly one x such that $C_x = i-1$

Those are all conditions needed, let's find the array C

For any set S of **special positions** $S = \{r_1, r_2, r_3 ......r_k\}, k = |S|$, call S a **proper set** if and only if $max\{C_x | x \in S\} + 1 < min\{x | x \in S\}$

Let T be the set of all special positions, then $S \subset T$

Since $\forall x, y \in T, x < y \to C_x < C_y$,for any proper set S, the number of different elements appeared in A is no less than $|S| + 1$

the largest proper set S can be found by simple prefix sum in $O(n)$

that means we can find $P = max\{|S| + 1\}$

Another observation is that $\forall i \in [1,n]$ such that $B_i < n+1$, the number of different elements appeared in A is no larger than $B_i - i + 1$. Let $Q = min\{B_i - i + 1 | i \in n[1,n], B_i < n+1\}$

Theorem: the desired C exists if and only if $P \le Q$

necessity: Let M be the number of different elements appeared in A, then we've known that $P \le M \le Q$

sufficiency: let's construct C using algorithm below:

```
//here "al" can be any integer between P and Q, in std.cpp al=P
//if i is a special position, vis[i]=c[i], otherwise vis[i]=-1
int l,r,cnt,seq[200005];
B[n+1]=n+1;
l=r=1;
cnt=2;
for(int i=1;i<=n;i++)
```

```
{
        if(vis[i]!=-1)C[i]=vis[i];
        else
        {
                if(cnt<=a1)C[i]=0,cnt++;
                else C[i]=seq[l++];
        }
        if(B[i+1]==B[i])seq[r++]=i;
}
```

you can prove that given $P \le Q$, this algorithm always gives a desired array C

Then, you can construct corresponding A easily from C

Problem Tutorial: "Game"

From the position (x, y), if you can't get to special points within three steps, then the SG status in (x, y) and $(x + 1, y - 1)$ are the same by simple case analysis.

There are only $O(n)$ special points, so only $O(n)$ points can't be deduced by $(x + 1, y - 1)$. Call those points bad points. So for any query (x, y), we just need to go to the farthest $(x + k, y - k)$ such that $(x + k, y - k)$ is a bad point. For bad points, we only need to bruteforce two kinds of possible choice.

If you always memorize the answers of queries done, you can get the total time complexity $O((n + q)\log(n + q))$