时间限制：C/C++ 3秒，其他语言6秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Given a, b, c, p, q and n, please calculate

$$\sum_{i=0}^{n}\sum_{1\le cj\le ai+b} i^p j^q$$

输入描述:

The first and only line contains 6 integers a, b, c, p, q, n ($1 \le n, c \le 10^9$, $0 \le a, b, \le 10^9$, $0 \le p, q \le 50$).

输出描述:

Output one line which contains a single integer. As the output can be very large, please output it modulo 998244353.

示例1

输入 复制

```
1 1 1 1 1 1
```

输出 复制

```
3
```

示例2

输入 复制

```
1 1 4 5 1 4
```

输出 复制

```
1267
```

示例3

输入 复制

```
8 9 3 8 9 3
```

输出 复制

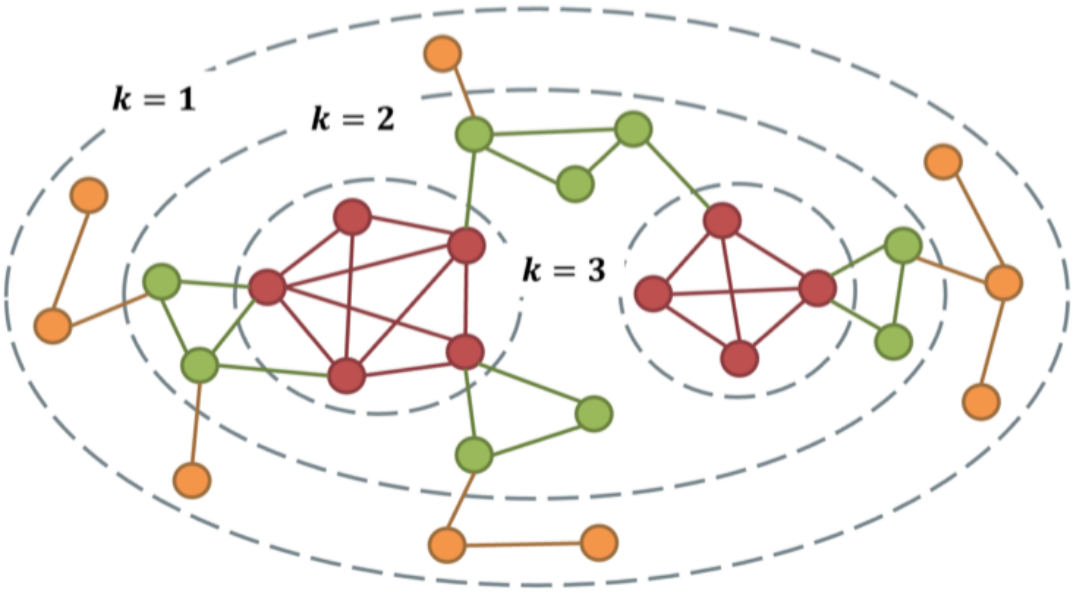
```
811685801
```

时间限制：C/C++ 2秒，其他语言4秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Cuber QQ is interested in locating the best k-degree subgraph in the graph G=(V(G),E(G)).

A subgraph S is a k-degree subgraph of G, if (i) each vertex $v \in S$ has at least k degree in S, (ii) S is connected; and (iii) S is maximal, i.e., any supergraph of S is not a k-degree except S itself.



Then Cuber QQ defines the score of a subgraph S. Before the definition of the score, he firstly defines

- n(S): the number of vertices in subgraph S, i.e., n(S) = |V (S)|;
- m(S): the number of edges in subgraph S, i.e., m(S) = |E(S)|;
- b(S): the number of boundary edges in subgraph S, $b(S) = |\{(u,v)|(u,v) \in E(G), u \in V(S), v \notin V(S), v \in V(G)\}|$;

Then he defines the score of a subgraph is $score(S) = M \cdot m(S) - N \cdot n(S) + B \cdot b(S)$, where M,N,B are given constants.

The higher the score of the subgraph, the better Cuber QQ thinks it is. You are asked to find the best k-degree subgraph in the graph G. If there are many k-degree subgraph with the same score, you should maximize k.

输入描述:

> The first line contains two integers n and m ($1 \le n, m \le 10^6$), where n and m are the number of nodes and the number of edges in the graph, respectively.
>
> The second line contains three integers M,N and B ($-10^9 \le M, N, B \le 10^9$), where the score of a subgraph is $score(S) = M \cdot m(S) - N \cdot n(S) + B \cdot b(S)$.
>
> Each of the next m lines contains two integers u and v ($1 \le u, v \le n,\ u \ne v$), meaning that there is an edge between vertices u and v.
>
> The given graph is an undirected graph without any loops and multiple edges.

输出描述:

> The only line contains two space-separated integers, which are k and score of the best k-degree subgraph. You should make sure that k>0 .

示例1

输入 复制

```
3 3
1 1 2
1 2
2 3
3 1
```

输出 复制

```
2 0
```

时间限制：C/C++ 3秒，其他语言6秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Cuber QQ got a table of infinite size. He will consider the table rows numbered from top to bottom $-\infty$ through $+\infty$, and the columns numbered from left to right $-\infty$ through $+\infty$. Then he'll denote the cell in row x and column y as (x, y).

Initially, there are n trucks at cell $(0, a_i)$ respectively. The goal is that each of the trucks have to go to one of the n destinations, located at cell $(1,0),(2,0),\cdots(n-1,0),(n,0)$. Cuber QQ has to make sure that there are no two trucks at the same destination. In another word, any two paths of the trucks are non-intersecting (no common points).

Each truck can go from cell (x, y) to one of two cells (x + 1, y) and (x, y - 1). And the truck can't have any chance of meeting any other trucks along the way.

Now can you help Cuber QQ to find the number of ways that trucks can achieve this goal.

输入描述:

> The first line contains one integer n ($1 \le n \le 5 \cdot 10^5$), where n is the number of trucks.
>
> The second line contains n integers a_i ($0 \le a_i \le 10^6$, $a_i < a_{i+1}$), which are the trucks' initial positions.

输出描述:

> Print the number of ways modulo 998244353 on the only line of output.

示例1

输入 复制

```
3
1 2 3
```

输出 复制

```
4
```

时间限制：C/C++ 1秒，其他语言2秒
空间限制：C/C++ 524288K，其他语言1048576K
Special Judge, 64bit IO Format: %lld

题目描述

Cuber QQ has challenged you with the task to construct a **binary tree with root 1**, and no more than 5000 nodes, along with a divide-and-conquer strategy, such that the recursive depth d of your divide-and-conquer strategy is at least 2 levels less than the strategy based on sizes of subtrees.

The recursive depth d of a function is defined as the maximum number of functions ever appeared in the call stack (see the return value of the pseudo-code below for a formal definition).

The divide-and-conquer strategy based on subtree sizes is shown as below:

$\mathrm{DivideAndConquer}(T)$:

$\quad$ **If** $|T| = 1$ **then return** 1;

$\quad u = T.\mathrm{root}$;

$\quad$ **For** node $v \in T$:

$\quad\quad$ **If** $u = T.\mathrm{root}$ **or** $\max\{|T_v|, |T| - |T_v|\} < \max\{|T_u|, |T| - |T_u|\}$

$\quad\quad\quad$ **or** $(\max\{|T_v|, |T| - |T_v|\} = \max\{|T_u|, |T| - |T_u|\}$ **and** $\min\{v, v.\mathrm{parent}\} < \min\{u, u.\mathrm{parent}\})$

$\quad\quad\quad$ **or** $(\max\{|T_v|, |T| - |T_v|\} = \max\{|T_u|, |T| - |T_u|\}$ **and** $\min\{v, v.\mathrm{parent}\} = \min\{u, u.\mathrm{parent}\}$ **and** $\max\{v, v.\mathrm{parent}\} < \max\{u, u.\mathrm{parent}\})$

$\quad\quad\quad$ **then** $u = v$;

$\quad$ Cut u from its parent and divide tree T into T_u and $T \setminus T_u$;

return $\max\{\mathrm{DivideAndConquer}(T_u), \mathrm{DivideAndConquer}(T \setminus T_u)\} + 1$;

Note: In this pseudo-code, T_u denotes a subtree in T whose root is u.

输入描述:

```
No input.
```

输出描述:

In the first line, output an positive integer n ($1 \le n \le 5000$), which is the node number of your binary tree.

In the following n-1 lines, output u_i and v_i in the i-th line, denoting an edge connecting u_i and v_i.

Cuber QQ will delete the edge one by one in the order that you give in the output, starting from (u_1, v_1), and recursively handle the subtrees.

时间限制：C/C++ 1秒，其他语言2秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

There are n cities in the volcano country, numbered from 1 to n. The city 1 is the capital of the country, called Eyjafjalla. A large volcano is located in the capital, so the temperature there is quite high. The temperature of city i is denoted by t_i.

The n cities are connected by n-1 undirected roads. The i-th road connects city u_i and city v_i. If u_i is closer to the capital than v_i, then $t_{ui} > t_{vi}$. The capital has the highest temperature.

The coronavirus is now spreading across the whole world. Although the volcano country is far away from the mess, Cuber QQ, the prime minister of the country, is still designing an emergency plan in case some city getting infected. Now he wants to know if a virus whose survival temperature range is [l, r] breaks out in city x, how many cities will be infected. The assumption is that, if the temperature of one city is between l and r inclusive, and it is connected to another infected city, it is infected too.

输入描述:

The first line contains an integer n ($1 \le n \le 10^5$), representing the number of cities.

In each of the next n-1 lines, there are two integers u,v ($1 \le u, v \le n,\ u \ne v$) per line, denoting that a road connecting u and v.

The next line contains n space-separated integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 10^9$), representing the temperature in each city.

The next line contains one integer q ($1 \le q \le 10^5$), representing the number of queries.

The next q lines contain one query each line. Each query is 3 space-separated integers x,l,r ($1 \le x \le n, 1 \le l \le r \le 10^9$), representing a query that a virus whose survival temperature range is [l, r] breaks out in city x.

输出描述:

For each query, output one integer representing the number of cities that will be infected. As a special case, if $t_x \notin [l, r]$, please output 0.

示例1

输入 复制

```
4
1 2
1 3
2 4
10 8 7 6
3
1 10 10
2 6 7
3 7 10
```

输出 复制

```
1
0
3
```

说明

> For the third query, the virus will first land in city 3, and then infect city 1, and then infect 2, so 3 cities will be infected.

时间限制：C/C++ 5秒，其他语言10秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Order execution is a key component in financial investment, which is the step to adjust the portfolio in real. For example, in stock investment, the investor might want to adjust the position (held amount) of its held instrument (stock) at the beginning of the day, and he/she expects the adjustment is done by the end of the day. If the desired position goes down, he/she needs to sell some stocks; if it goes up, he/she needs to buy some. So he makes an "order" at the very beginning of the day, and expects the order to be "executed" in this day.

The timing in which he/she buys/sells the stock has an impact on the profit. For example, if the stock price goes down, selling it in the morning is better than afternoon. The problem that on which time slots the investor should do the liquidation/acquisition (buying/selling) to obtain the desired position is called order execution. In a typical setting of such problem, only past market information can be observed, and it can be reduced into a sequential decision making problem, but we are doing something different in this problem. Instead of trying to predict the "unpredictable" market, for the scope of this task, you can given the market information of the whole time horizon (one day in our case). The task is to find out the best timing to execute an intraday order, so that the profit is maximized.

We set this task on China A-shares market, which are the stock shares of mainland China-based companies that trade on the two Chinese stock exchanges, the Shanghai Stock Exchange (SSE) and the Shenzhen Stock Exchange (SZSE). In our setting, the finest-grained execution step duration is one minute. A-shares market has 240 minutes (9:30-11:30, 13:00-15:00) every day, which means you will have 240 chances. In each minute, you can decide to buy/sell some shares. The cash change is the deal price per share on that minute times the traded shares. Note that, if you have a "SELL" order, **you cannot buy any share on that day**; if the order is in "BUY" direction, you cannot sell any.

The shares you can buy/sell on a particular minute is limited by the market volume on that minute. This limitation should be carefully computed so that the impact on market price is taken into account even during the simulation. For simplicity, we give a "safe" volume for each minute, which is the maximum number of shares that is available for trade in that minute. Also, on A-shares market, the minimum trading unit is "one lot", which is 100 shares, meaning that the shares to trade on a particular minute must be a multiple of 100.

If on one minute, the trading algorithm gives a non-zero volume, i.e., it desires to buy/sell some shares (a multiple of 100 shares of course), the exchange will make the deal, and also charge for the deal. This includes the handling fee, government tax and etc. The charge is 0.03% of the trade value (which is the cash you pay/earn for the trading itself), **rounded to cents** (cents means 0.01 yuan). If the charge is less than 5 Chinese yuan, it will be 5 yuan.

输入描述:

The first line of input is one integer T ($1 \leq T \leq 5\,000$), stating that there will be T test cases.

In each case, we show an order along with the market information on that day, which has 241 lines.

In the first line, there is a word BUY or SELL, followed by a space and a number a ($100 \leq a \leq 5\,000\,000$, a is a multiple of 100), which is the shares of the order to execute.

The following 240 lines are the market information from 9:30 to 15:00. Each line has two number, separated by space. The first is price p_i ($0.01 \leq p_i \leq 3550$), rounded to cents, and the second is the maximum number of shares availble for trading on that minute v_i ($0 \leq v_i \leq 5\,000\,000$).

It is guaranteed that the orders can all be completely executed.

The sum of a from all test cases does not exceed $5 \cdot 10^7$.

输出描述:

Output one number of each test case, which is the maximum cash balance (cash on account at the end of the day minus cash at the beginning), rounded to cents.

示例1

输入 复制

```
1
BUY 101600
3.07 4734400
3.04 1343900
3.04 637400
3.03 658100
3.03 471100
3.03 574000
3.04 646400
3.04 359100
3.02 1290200
3.02 231200
3.02 292800
3.01 276300
3.01 906600
3.02 614902
3.02 263900
3.02 298998
3.02 112000
3.03 466902
3.03 428000
3.03 209100
3.03 100500
3.02 534300
3.02 755800
3.01 194400
2.99 2646800
3.00 480100
3.01 267200
3.00 103900
3.01 211600
3.00 288200
3.01 306900
3.01 203600
3.00 190500
3.00 77900
3.00 171700
3.01 241800
3.01 161700
3.01 72300
3.00 166400
3.00 261300
3.01 322100
3.02 399800
3.02 134600
3.02 104700
3.02 165700
3.01 252300
3.01 248800
3.00 126200
3.01 323500
3.01 149800
3.01 67700
3.01 131400
3.01 237900
3.01 169900
```

```
3.02 112600
3.02 245800
3.02 108100
3.01 89200
3.02 51100
3.02 112500
3.01 36400
3.02 186700
3.02 106200
3.02 94600
3.02 58600
3.01 170800
3.01 69300
3.01 69200
3.02 38000
3.01 65100
3.02 139200
3.02 147900
3.02 56800
3.02 79500
3.01 52400
3.02 54100
3.02 102000
3.02 43700
3.01 174800
3.02 89800
3.01 150200
3.02 839200
3.02 20200
3.02 126600
3.01 15700
3.02 148800
3.02 185700
3.01 86200
3.01 51100
3.01 52600
3.02 73900
3.02 47600
3.01 814900
3.00 107000
3.00 108000
3.00 200300
3.01 138300
3.00 1272300
3.00 82200
3.00 153000
2.99 165900
2.99 103500
3.00 836500
2.99 157700
3.00 407900
2.99 249700
2.98 1284300
2.98 312000
2.98 110200
2.97 1881453
2.98 738200
```

```
2.97 913500
2.97 723100
2.97 732653
2.97 237800
2.96 328800
2.96 1358047
2.95 531700
2.96 766200
2.97 186553
2.99 1096800
2.99 819547
2.98 198400
2.98 195700
2.99 850400
2.98 424700
2.99 184700
3.00 309700
2.99 322600
2.99 95000
2.99 52700
2.99 39100
3.00 88400
2.99 147600
2.99 121300
2.99 114500
2.98 133200
2.98 284300
2.98 220200
2.98 78700
2.98 33300
2.99 12700
2.98 79300
2.99 115700
2.99 135100
2.98 117000
2.99 27800
2.98 18200
2.99 99400
2.98 43000
2.98 76000
2.98 120000
2.99 55400
2.98 411400
2.99 201100
2.98 24600
2.98 52400
2.98 38700
2.99 149500
2.98 134000
3.00 646900
3.00 44300
2.99 104000
2.99 95300
2.99 75800
2.99 59400
2.99 83700
2.99 76700
```

```
2.99 50200
3.00 91500
2.99 308500
2.99 235600
2.99 77100
2.99 38900
2.99 43100
2.99 433600
2.99 53100
2.98 332200
2.99 69900
2.99 37000
2.98 75400
2.98 41200
2.98 5400
2.98 41000
2.99 37800
2.98 51200
2.98 31100
2.99 33000
2.99 114100
2.99 127300
2.98 502600
2.98 268300
2.99 24700
2.98 64100
2.98 74700
2.98 221400
2.98 90600
2.97 149800
2.98 254000
2.98 71700
2.98 77800
2.99 280200
2.98 30500
2.97 171400
2.97 193900
2.97 109300
2.98 337700
2.97 171400
2.98 193300
2.97 88800
2.98 42900
2.97 267300
2.98 104700
2.98 39200
2.98 71700
2.98 348500
2.98 104100
2.98 211000
2.98 231800
2.98 64600
2.99 63000
2.98 109100
2.99 54000
2.98 63800
2.98 124100
```

```
2.98 481300
2.99 125300
2.98 196100
2.98 382300
2.98 92900
2.98 162000
2.98 291400
2.99 318000
2.99 339200
2.98 602700
2.98 719100
2.99 395000
2.99 516033
2.99 516033
2.99 516033
```

输出 复制

```
-299809.92
```

时间限制：C/C++ 2秒，其他语言4秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Here you are given a 1-rooted tree with n vertices, with a glass ball staying on every vertex, respectively.

For every vertex u with parent v, u's altitude (namely height) is higher than v by 1 unit. Thus a ball on u can roll along edge (u, v) to v. A ball rolling along an edge takes 1 second.

All balls start rolling at the same time. There are some special vertices called ``storage vertices". Not every vertex is storable. Besides the root, which must be storable, the probability of a vertex being storable is p. If a ball rolls to a storage vertex, it breaks, and it will be taken up from the tree. If a ball is initially on a storage vertex, it breaks immediately.

Two balls rolling to the **same** vertex at the **same** time is called a **collision**, without considering the type of the vertex. If collision occurs, it will be a terrible incident. Not only the two balls will break, but also the whole system, and all the rolling cannot continue any more.

If collisions occur, you lose and your score is 0. Otherwise your score is $\sum_{i=1}^{n} f(i)$, where f(u) denotes the number of edges that the ball initially on u come across during the whole rolling process. Now please calculate the expect score module $998\,244\,353$.

输入描述:

> The first line contains two integers n, p ($1 \leq n \leq 5 \times 10^5$, $0 \leq p < 998\,244\,353$), where p is the probability modulo $998\,244\,353$. To explain this, we denote p as an irreducible fraction $\frac{a}{b}$, where a and b are integers and $b \not\equiv 0 \pmod{M}$. Then $p \equiv a \cdot b^{-1} \pmod{998\,244\,353}$. In other words $p \cdot b \equiv a \pmod{998\,244\,353}$.
>
> The second line contains n - 1 space-separated integers $a_2, a_3, \ldots, a_n$, where a_i denotes the parent vertex of i.

输出描述:

> Output one line contains one integer denoting the expect score.

示例1

输入 复制

```
3 499122177
1 1
```

输出 复制

```
499122177
```

示例2

输入 复制

```
8 1919810
1 1 1 4 5 1 4
```

输出 复制

```
495380040
```

时间限制：C/C++ 1秒，其他语言2秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

Digits 2, 3 and 6 are happy, while all others are unhappy. An integer is happy if it contains only happy digits in its decimal notation. For example, 2, 3, 263 are happy numbers, while 231 is not.

Now Cuber QQ wants to know the n-th happy positive integer.

输入描述:

> The first and only line of input contains a positive integer n ($1 \leq n \leq 10^9$).

输出描述:

> The first and only line of output contains the n-th happy positive integer.

示例1

输入 复制

```
680
```

输出 复制

```
326623
```

时间限制：C/C++ 1秒，其他语言2秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

In the block-chain, miners compete for proposing a valid block to append it to the current blockchain, which is incentivized by rewards. The chance of winning the competition usually depends on the resource controlled by miners, e.g., computation power. Despite its popularity, PoW (Power of Work) incurs massive energy consumption as the mining competition relies on computation power. To eradicate the waste of computation resources, PoS (Power of Stake) protocols are invented, where the competition depends on staking power instead.

There is a PoS incentive model that uses the multi-lottery PoS incentive policy. Stakeholders of ML-PoS blockchains create a valid block if a candidate block satisfies the condition that $Hash(\text{time}, \cdots) < D \cdot \text{stake}$, where time represents the timestamp when the candidate block is generated, D is a pre-determined mining difficulty and stake is the value of stakes possessed. Since $Hash(\cdot)$ is uniformly distributed in the range $[0, 2^{256} - 1]$, the event that a candidate block becomes valid satisfies Bernoulli distribution with a success probability of $\frac{D \cdot \text{stake}}{2^{256}}$. Thus, if a miner possesses more stakes, he/she is more likely to create a new block successfully. Moreover, miners will try at different timestamps until a candidate block becomes valid. The trials are independent of timestamps following the same Bernoulli distribution.

Now, Cuber QQ considers the two-miner scenario where miner A and miner B are competing for proposing blocks in the network. Initially, the **resource (stake)** share of miner A (resp. B) is a (resp. b). Without loss of generality, a and b are normalized such that a+b=1. We refer to T_A (resp. T_B) as the number of timestamps miner A (resp. B) has checked until A (resp. B) meets the first success timestamp. It is easy to see that T_A and T_B follow geometric distributions with probability parameters $p_A = \frac{D \cdot a}{2^{256}}$ and $p_B = \frac{D \cdot b}{2^{256}}$, respectively, i.e., $\Pr[T_A = t] = (1 - p_A)^{t-1} p_A$, Furthermore, if A wins the next block, A finds a valid block with fewer timestamps than B such that $T_A < T_B$ or A has a chance of 50% when $T_A = T_B$ to break the tie.

Each turn, the miner who successfully creates a new block will be rewarded an incentive w, so the chance that A can win a block not only depends on the initial staking power (i.e., a) but also relies on previous mining outcomes. Specifically, the probability of A proposing a new block is determined by A's current staking power, including the initial stakes and the earned stakes. For example, if a miner is ``lucky" to mine some blocks, his/her expected rewards will be improved in the future as the volume of her stakes increases. In the following, we show that ML-PoS still preserves expectational fairness. We also assume both miners A and B do not perform additional action after a mining game starts.

Cuber QQ uses λ_A (resp. λ_B) to denote the fraction of the total reward that miner A (resp. B) received (obviously, $\lambda_A + \lambda_B = 1$). Two miners are competing for n blocks in total. He now wants to know $\mathbb{E}[n \cdot \lambda_A]$, i.e., the reward of miner A (**number of the blocks that miner A will win**) in expectation.

输入描述:

> The first and only line contains three space-separated positive integers n, w, x and y ($1 \le n, w < 998\,244\,353, 0 \le x \le y < 998\,244\,353, y > 0$). Then $a = \frac{x}{y}$ and $b = 1 - a = 1 - \frac{x}{y}$.

输出描述:

> Output one integer --- the reward of miner A (number of the blocks that miner A will win) in expectation modulo 998244353.
>
> Formally, let M=998~244~353. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \mod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

示例1

输入 复制

```
1 1 0 1
```

输出 复制

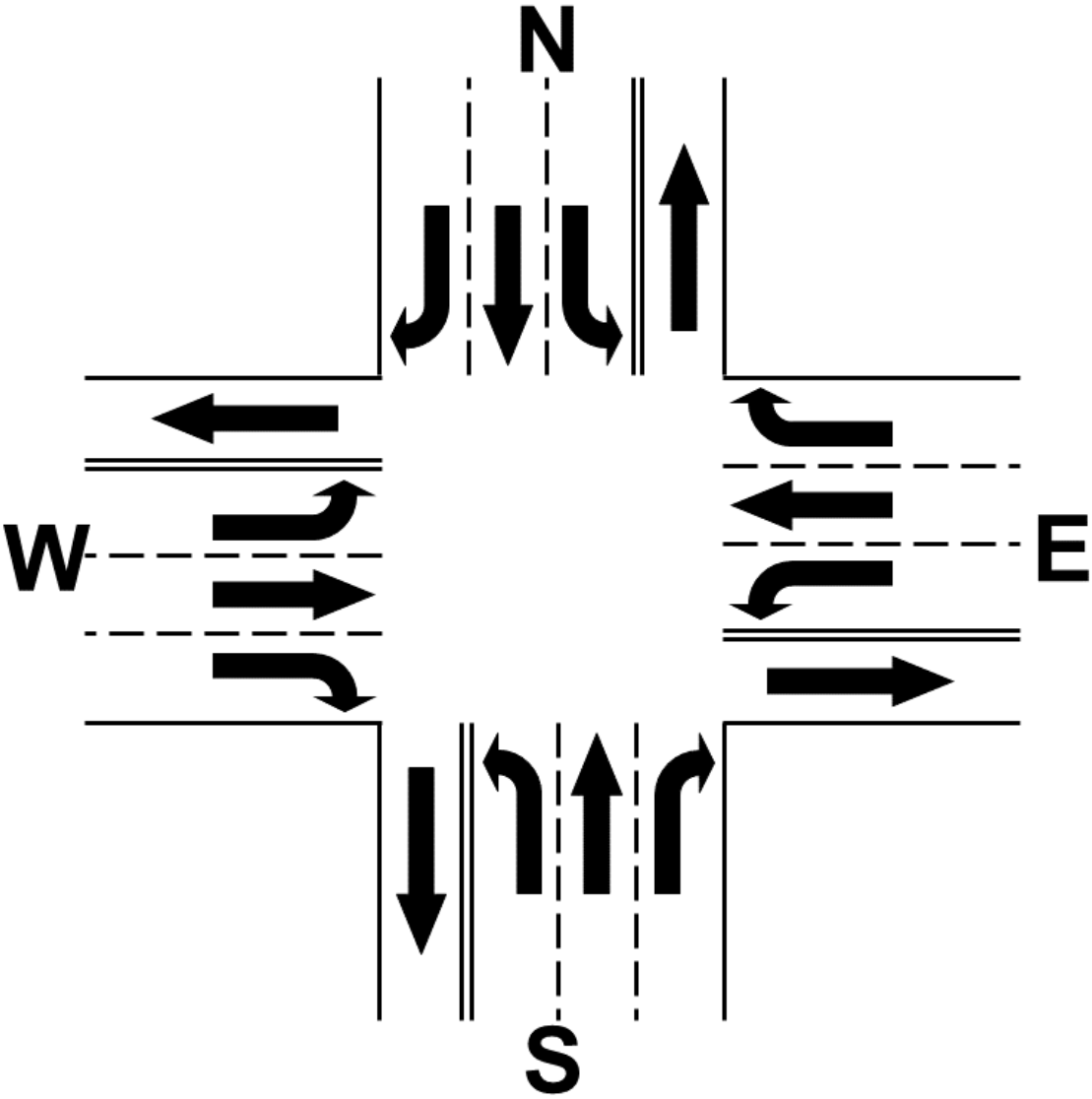
```
0
```

时间限制：C/C++ 4秒，其他语言8秒
空间限制：C/C++ 524288K，其他语言1048576K
64bit IO Format: %lld

题目描述

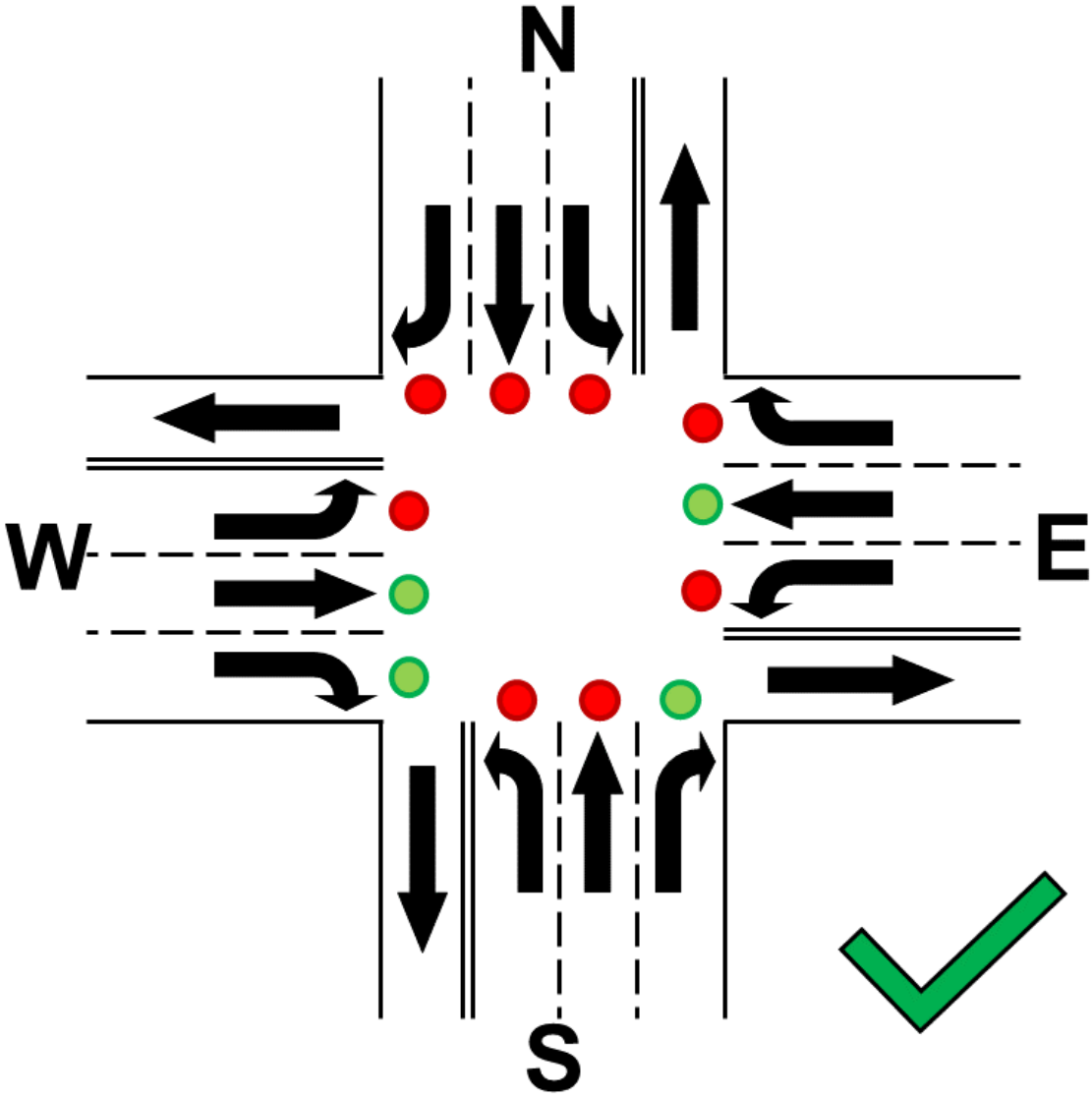
Controlling traffic lights on crossing roads is one of the most essential applications of scheduling algorithms. Elaborating on those details will be a long story. In short, for this task, you will control the traffic lights to resolve a already-happened traffic jam.

The following figure shows a typical example of a crossing road in a Right-hand Traffic (RHT, meaning that drivers keep to the right side of the road). If we do not consider cases where drivers turn around, there are 12 directions: N->E, N->S, N->W, E->S, E->W, E->N, S->W, S->N, S->E, W->N, W->E and W->S.

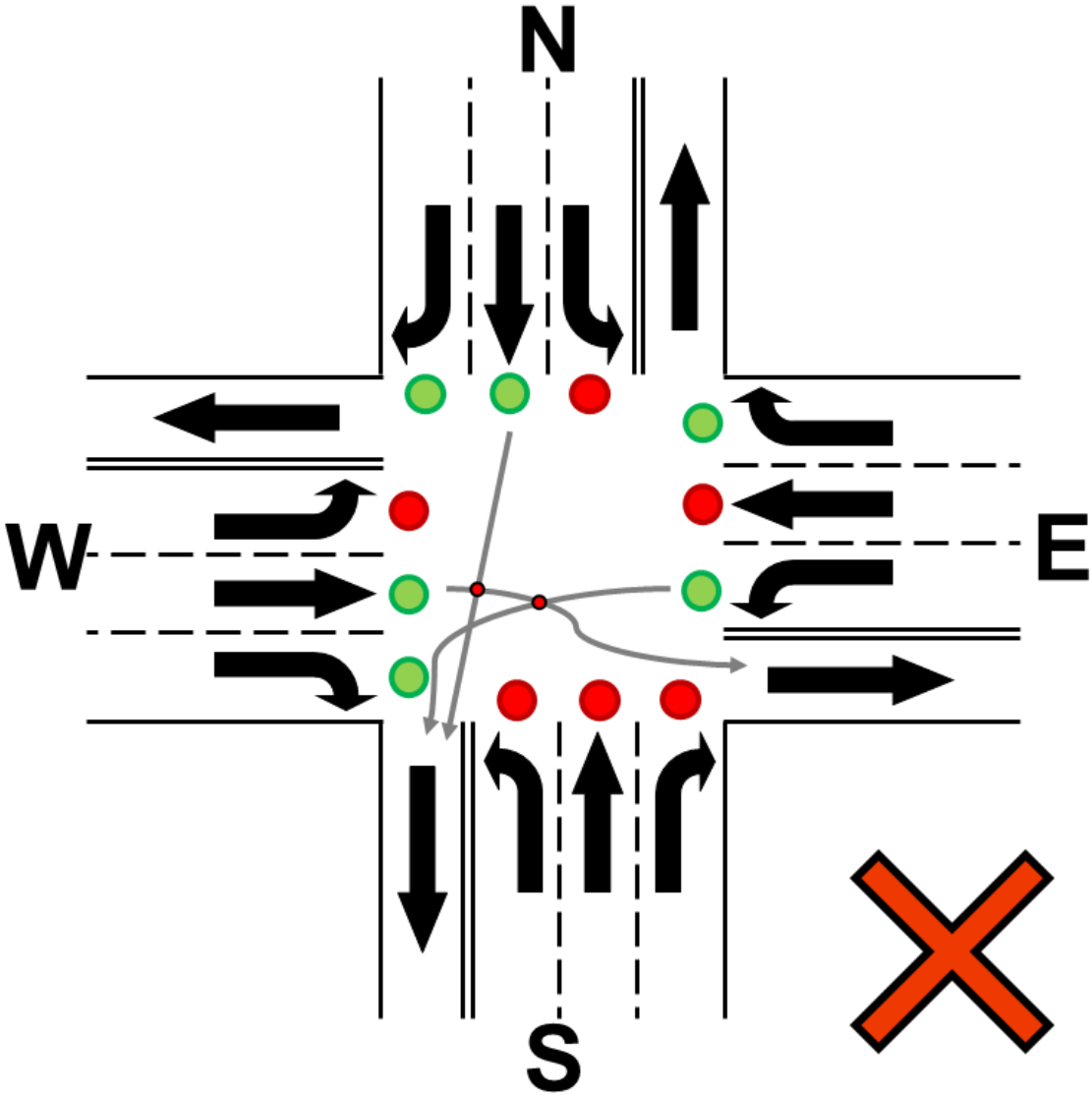


We have the situation here that there is already a traffic jam. The number of cars driving in the 12 directions are already known respectively. Each car needs 1 second to pass the crossing road to reach the destination. We assume there is no more traffic jam after they manage to drive through the crossing road. There will be no more cars entering this jam either.

Your task is to assign a appropriate traffic light for each second, so that the traffic jam is resolved (all the cars successfully pass the crossing roads) as quickly as possible. The assignment should also guarantee no potential collision between cars. The following figure show a valid example.



Here is a bad case. It is illegal because there can be two collisions as marked in the figure, but we allow that two cars run into the S direction simultaneously and we do not penalize for this.



输入描述:

The first line of input is a number T ($1 \leq T \leq 100$), which is the number of test cases.

For each test case, there will be 4 lines, representing a matrix. Each line is 4 integers separated by space, representing one row of the matrix.

The j-th integer on i-th line $c_{i,j}$ ($c_{k,k} = 0,\ 0 \leq c_{i,j} \leq 100$) denotes number of cars attempting to drive from the i-th direction to j-th direction, where the i-th direction means N, E, S, W for i=1,2,3,4 respectively. For example, $c_{3,2}$ corresponds to S->E.

输出描述:

For each test case, output one number, which is the minumum number of seconds to resolve the traffic jam.

示例1

输入 复制

```
2
0 1 1 1
0 0 0 0
0 0 0 0
0 0 0 0
0 3 1 4
```

```
1 0 5 9
2 6 0 5
3 5 8 0
```

输出 复制

```
1
17
```

说明

```
For example 1, there are only 3 cars running from the same direction, and they can go simultaneously.

For example 2, here is what could happen during the 17 seconds.

1. N->W, W->S, E->W
2. E->N, E->W, S->W, W->S
3. N->W, E->S, E->W
4. S->E, E->S, E->W
5. W->S, E->S, W->N
6. E->S, N->W, W->N
7. W->S, E->S, W->N
8. W->S, N->E, S->W
9. N->W, N->E, S->W
10. W->S, N->E, S->W
11. S->E, E->W, W->E
12. E->W, W->S, W->E
13. S->E, E->W, W->E
14. S->E, E->W, W->E
15. S->E, E->W, W->E
16. S->E, N->S, S->N
17. S->N, S->W, W->S
```