

Użytkownik

[Strona główna](#)[Pomoc](#)[Zmiana hasła](#)[Wylogowanie](#)

Student

[Przedmioty](#)

## Dynamic sparse table

W celu przechowywania wartości indeksowanych rzadkimi kluczami z pewnego uniwersum  $N$  dobrym rozwiązaniem wydają się być tablice haszowane. Minusem ich zastosowania może być kiepski czas dostępu w pesymistycznym przypadku. Innym sposobem poradzenia sobie z tym zadaniem jest zastosowanie drzewa trie, które charakteryzuje się lepszym oszacowaniem przypadku pesymistycznego.

W naszym zadaniu drzewo trie opisane jest dwoma parametrami:  $n$  i  $k$ . Budowane jest w następujący sposób: korzeń zawiera  $n$  węzłów potomnych a każdy węzeł potomny posiada kolejne  $k$  węzłów potomnych. Wstawianie klucza  $x$  zaczynamy od korzenia i jeśli jest on wolny (każdy węzeł przechowuje maksymalnie jeden klucz i ewentualnie skojarzoną z nią wartość – w naszym przypadku będzie to jedynie klucz) to zapisujemy wstawianą wartość do węzła i kończymy operację. W przypadku kiedy korzeń jest już zajęty szukamy dla niego miejsca wybierając kolejny węzeł potomny równy  $\text{div} = x \bmod n$  a następnie  $\text{div} = \text{div} \bmod k$ , aż natrafimy na wolne miejsce. Wyszukiwanie wykonujemy analogicznie rozpoczynając od korzenia i przeglądając kolejne węzły o indeksach  $\bmod n$  a następnie  $\bmod k$  aż do znalezienia wartości  $x$  albo trafienia na pusty węzeł potomny (brak klucza w drzewie). Kasowanie jest trochę trudniejsze i wymaga odszukania węzła  $p$  z daną wartością klucza oraz, w przypadku kiedy jest to węzeł wewnętrzny, kandydata do skasowania. Kandydata  $q$  możemy wyszukiwać na różne sposoby jednakże zawsze musi to być węzeł zewnętrzny (nie posiadający potomstwa). My zastosujemy prostą strategię szukania zawsze na lewo. Jeśli znajdziemy kandydata to przepisujemy wartość w miejsce usuwanego klucza  $x$  w węźle  $p$  i usuwamy węzeł  $q$ .

Więcej informacji na [Storing a sparse table](#).

#####  
To store values indexed by sparse keys from a certain universe  $N$ , **hash tables** seem to be a good solution. A downside of using them may be poor access time in the **pessimistic case**. Another way to handle this task is to use a **trie** tree, which offers a better pessimistic time estimate.

In our task, the trie is described by two parameters:  $n$  and  $k$ . It is built in the following way: the root has  $n$  child nodes, and each of those has  $k$  more child nodes. To insert a key  $x$ , we start from the root. If the root is free (each node holds at most one key and possibly a value — in our case only the key), we store the value and finish the operation. If the root is already occupied, we search for a place by selecting the next child node using  $\text{div} = x \bmod n$ , and then  $\text{div} = \text{div} \bmod k$ , repeating until we find a free spot. Searching is done similarly — starting from the root and following nodes with indices  $\bmod n$  and then  $\bmod k$  until we find the value  $x$  or reach an empty child node (meaning the key is not in the tree). Deleting is a bit harder and requires finding the node  $p$  with the given key. If it's an internal node, a deletion candidate must be selected. The candidate  $q$  can be searched for in different ways but must always be a **leaf node** (one with no children). We use a simple strategy of always searching to the **left**. If a candidate is found, its value is copied to the node  $p$  (where key  $x$  was stored), and then node  $q$  is deleted.

More information at [Storing a sparse table](#).

#####

Użytkownik

[Strona główna](#)[Pomoc](#)[Zmiana hasła](#)[Wylogowanie](#)

Student

[Przedmioty](#)

simple strategy of always searching to the **left**. If a candidate is found, its value is copied to the node **p** (where key **x** was stored), and then node **q** is deleted.

More information at [Storing a sparse table](#).

#####

## Wejście

Pierwszy wiersz zawiera liczbę **n** określającą liczbę przypadków testowych (komendy **I**, **L**, **D** oraz **P** – wstaw, wyszukaj, usuń i przeglądaj), każdy w nowej linii. W kolejnym wierszu podane zostaną dwie liczby minimum i maksimum określające jak wielkie liczby mogą pojawić się na wejściu. W kolejnym wierszu kolejne dwie liczby to parametry **n** i **k** naszego drzewa trie. Następnie, po pustej linii wystąpi **n** przypadków testowych, o których była mowa na początku. Komenda:

- **I x** – powoduje wstawienie klucza **x** do drzewa. Jeśli już taki tam się znajduje to wypisuje jedynie informację **x exist**, w przeciwnym wypadku niczego nie wypisuje.
- **L x** – wyświetla informację **x exist** w przypadku odzyskania klucza oraz **x not exist** w przeciwnym przypadku.
- **D x** – kasuje klucz **x** z drzewa. W przypadku niepowodzenia (brak klucza) wypisuje informację: **x not exist**. W przypadku poprawnego wstawienia do drzewa niczego nie wyświetla. Do skasowania wybierany jest potomny węzeł zewnętrzny położony najbardziej po lewej węzła zawierającego klucz **x** albo węzeł zawierający **x** – jeśli nie posiada on potomków.
- **P** – wyświetla zawartość drzewa w kolejności **preorder**. Jeżeli podczas operacji kasowania (**D x**) nie będzie wybierany pierwszy węzeł potomny najbardziej po lewej, będący węzłem zewnętrznym, to kształt drzewa będzie się różnił, co wpłynie na kolejność preorder i ostatecznie na wynik.

## Wyjście

Kolejne linie zawierające informacje będące rezultatem użycia komend (**I**, **L**, **D**, **P**) albo brak linii w przypadku braku takiej informacji (w przypadku poprawnego dodania bądź skasowania **x**).

#####

## Input

The first line contains the number **n**, specifying the number of test cases (commands **I**, **L**, **D**, and **P** – insert, lookup, delete, and print), each on a new line. The next line contains two numbers: **minimum** and **maximum**, defining the possible range of input values. The following line contains two more numbers: **n** and **k**, the parameters of our trie. Then, after a blank line, there are **n** test cases as mentioned earlier. Command:

- **I x** – inserts key **x** into the trie. If it already exists, it prints **x exist**; otherwise, it prints nothing.
- **L x** – prints **x exist** if the key is found, otherwise **x not exist**.
- **D x** – deletes key **x** from the trie. If deletion fails (key not found), it prints **x not exist**. If deletion is successful, it prints nothing. The deleted node is either the **leftmost external child** of the node containing **x**, or the node itself if it has no children.

18:01

Przedmiot

Treść

Zgłaszanie

Wynik

Kod



Użytkownik

[Strona główna](#)[Pomoc](#)[Zmiana hasła](#)[Wylogowanie](#)

Student

[Przedmioty](#)

albo brak linii w przypadku braku takiej informacji (w przypadku poprawnego dodania bądź skasowania x).

#####

## Input

The first line contains the number **n**, specifying the number of test cases (commands **I**, **L**, **D**, and **P** – insert, lookup, delete, and print), each on a new line. The next line contains two numbers: **minimum** and **maximum**, defining the possible range of input values. The following line contains two more numbers: **n** and **k**, the parameters of our trie. Then, after a blank line, there are **n** test cases as mentioned earlier. Command:

- **I x** – inserts key **x** into the trie. If it already exists, it prints **x exist**; otherwise, it prints nothing.
- **L x** – prints **x exist** if the key is found, otherwise **x not exist**.
- **D x** – deletes key **x** from the trie. If deletion fails (key not found), it prints **x not exist**. If deletion is successful, it prints nothing. The deleted node is either the **leftmost external child** of the node containing **x**, or the node itself if it has no children.
- **P** – prints the contents of the tree in **preorder** order. If during deletion (**D x**) the leftmost external node is not chosen, the tree structure will differ, affecting preorder traversal and final output.

## Output

Each line contains the result of the commands (**I**, **L**, **D**, **P**), or no line in case no output is expected (e.g., successful insertion or deletion x).

#####

## Przykład/Example

### Wejście

29

0 511

8 4

I 3

I 31

I 150

I 190

I 130

I 174

I 81

I 30

I 203

P

L 32

L 30

L 150

D 150

L 150

I 150

L 150

D 3

L 3

I 3

18:02

Przedmiot

Treść

Zgłaszanie

Wynik

Kod



Użytkownik

[Strona główna](#)[Pomoc](#)[Zmiana hasła](#)[Wylogowanie](#)

Student

[Przedmioty](#)

## Output

Each line contains the result of the commands (**I**, **L**, **D**, **P**), or no line in case no output is expected (e.g., successful insertion or deletion x).

#####

## Przykład/Example

### Wejście

29

0 511

8 4

I 3

I 31

I 150

I 190

I 130

I 174

I 81

I 30

I 203

P

L 32

L 30

L 150

D 150

L 150

I 150

L 150

D 3

L 3

I 3

L 3

D 3

L 3

D 30

L 30

I 30

L 30

D 30

L 30

### Wyjście/Input

3 81 130 203 150 174 190 30 31

32 not exist

30 exist

150 exist

150 not exist

150 exist

3 not exist

3 exist

3 not exist

30 not exist

30 exist

30 not exist

## Testy/Tests

[1](#), [2](#), [3](#), [4](#).