TÉCNICO LISBOA

DEI
Departamento de Engenharia Informática

Advanced Algorithms

2024/2025

Project 1

Top-Down 2-3-4 Trees (version 1.3)

This project considers the problem of implementing a 2-3-4 tree to store fractions. The first delivery deadline for the project code is the 5th of March, at 12:00. The deadline for the recovery season is 9th July at 10:00, the system opens the 7th of July at 10:00. The deadline for the special season is 23th July at 10:00, the system opens the 21th at 10:00.

Students should not use existing code for the algorithms described in the project, either from software libraries or other electronic sources. They should also not share their implementation with colleagues, specially not before the special season deadline.

It is important to read the full description of the project before starting to design and implementing solutions.

The delivery of the project is done in the mooshak system.

1 2-3-4 Trees

The challenge is to implement a 2-3-4 tree data structure. A description of this data structure was given by Sedgewick [1998]. The implementation must strictly verify the specification given in this script, meaning that the resulting output must match exactly the one described in this document. Moreover the structure described in this script is designed to reduce the complexity of the implementation.

The input will consist of a sequence of commands, each one will correspond to an operation over the tree.

1.1 Description

Let us start by describing the basic building block of the 2-3-4 Tree, the `struct node234` that is used to store information about a node.

```
typedef struct node234* node234;

struct frac{ /* Stores a fraction */
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wlong-long"
  unsigned long long int a; /* Numerator */
  unsigned long long int b; /* Denominator */
#pragma GCC diagnostic pop
};

struct node234
{
  struct frac V[3]; /* Key values */
  node234 p[4]; /* Pointers */
};
```

Most fields are explained by the comments. The `struct frac` is used to store a fraction, where the `a` field stores the numerator and `b` store the denominator. These values will be supplied in the input and are always co-prime, meaning that their greatest common divisor is 1.

The `pragma` directives are necessary to suppress a compilation error with the given compilation flags due to the use of the `unsigned long long int` type. Using `pragma` directives to suppress other errors besides the ones triggered by the `-Wlong-long` option is strictly forbidden. The `unsigned long long int` type stores an unsigned 64 bit integer.

Concerning the `struct node234` the `V` field stores up to three fractions and the `p` field up to four pointers to `struct node234`. When there is no relevant information to store in the pointers they should contain `NULL`. Likewise the unused fractions should store 0 values for both `a` and `b`.

The name 2-3-4 refers to the number of valid pointers in each node, which is always one more than the number of fractions in the same node. Therefore a node with only one active fraction has two active pointers, a node with two active fractions has three active pointers and a node with three active fractions has four active pointers. The type of a node is the number of active pointers in that node. Hence nodes can be of type two, type three or type four. These nodes are arranged into a tree structure, with several properties. The tree is a sort of binary search tree, meaning that the inorder traversal of

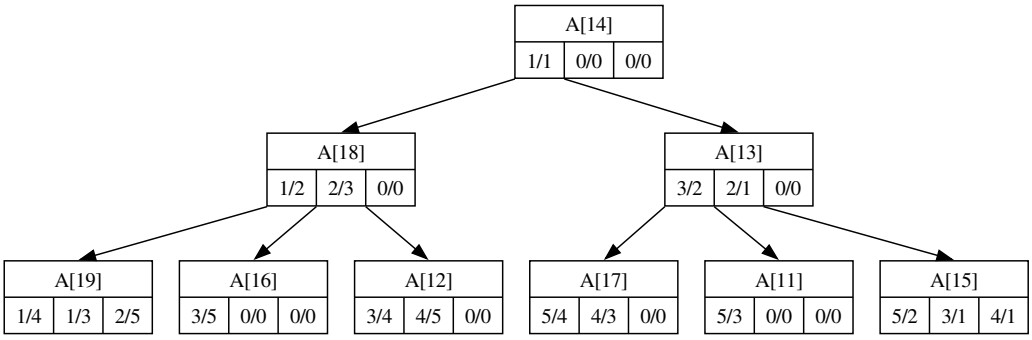

Figure 1: Example of a 2-3-4 Tree.

the nodes yields a sorted sequence of fractions, in increasing order. Hence, in particular, the fractions in a node are also in increasing order. Figure 1 shows an example of such a tree. The corresponding inorder sequence of factions is: $1/4 < 1/3 < 2/5 < 1/2 < 3/5 < 2/3 < 3/4 < 4/5 < 1/1 < 5/4 < 4/3 < 3/2 < 5/3 < 2/1 < 5/2 < 3/1 < 4/1$.

A crucial property of this tree is that it is balanced, all the sequences of pointers from the root to a `NULL` pointer have the same size. In the example of Figure 1 all these sequences contain exactly two pointers, excluding the `NULL` pointer, this coincides with the height of the tree. Hence this property can be re-stated as the fact that all the leaves of the tree are at the same distance from the root. In the operations that follow it is crucial the preserve this property.

To print the information related to a `node234` we will use the following code.

```
int ptr2loc(node234 v)
{
  int r;
  r = -1;
  if(NULL != v)
    r = ((size_t) v - (size_t) A) / sizeof(struct node234);

  return (int)r;
}

void showNode(node234 v)
{
  int f;
```

```
  int k;
  assert(NULL != v && "Trying to show NULL node.");

  printf("node: %d ", ptr2loc(v));
  k = 0;
  while(true){
    printf("%d ", ptr2loc(v->p[k]));
    if(3 == k) break;
    printf("%llu/%llu ", v->V[k].a, v->V[k].b);
    k++;
  }
  printf("\n");
}
```

This code assumes that there is a global variable `A` which contains all the necessary `struct node234`. This array of structs is allocated in the beginning of the program. Moreover the number of a node is given by its index in `A`, recall that in `C` indexes start at 0.

First let us make a brief description of the operations the implementation must support.

S `(node234)` Calls the `showNode` function for the node `A[i]`, where i is given as argument.

F `(frac)` Searches for the argument fraction in the tree. If the faction exists in the node at `A[i]` the function returns `i`. Otherwise, if the fraction is not found, the return is -1.

N Prints the fractions of the tree in inorder, separated by spaces. There should be a single space before the newline character.

P Prints the index of the node of the tree in Pre-order, separated by spaces. There should be a single space before the newline character. Note that this order is defined recursively as first the current node number and then, recursively, the pre-order of the nodes stored in the current pointers of the node, in increasing index order.

I `(frac)` Inserts the argument fraction into the tree. This function returns the index `i` when the given fraction is added to the node `A[i]`.

D `(frac)` Deletes the argument fraction from the tree. In case the given fraction does not exist in the tree the function returns -1. Otherwise the function the returns the index `i` of the leaf that lost a fraction due

to the procedure. Note that the lost fraction might be the argument fraction or its successor in the tree.

A line starting with this character is a comment and requires no interaction with the 2-3-4 tree, instead the whole line should be copied to the output.

X Terminates the execution of the program.

L Loads a tree configuration. This is given in the next lines of the input. If any of these lines starts with a `#` it is a comment line and can be ignored. The first line indicated the stack and the root. The next lines the information inside the nodes. This function is provided and is useful for debugging purposes. The configuration lines end with a line containing a single `X` character.

We can now discuss these operations in more detail. The first value that is given in the input is a value `n` that indicates how many `struct node234` instances will be necessary for the commands that follow. We therefore first alloc an array `A` containing `n` of these structs as follows:

```
scanf("%d", &n);

A = (node)calloc(n, sizeof(struct node234));
```

We therefore refer to each `node234` by its index in `A`, i.e., we will use `i` to represent the `node234` in `A[i]`. Hence the number 0 represents the first node. Note that the `calloc` function guarantees that the allocated memory is zeroed, which guarantees that the pointer fields are set to `NULL` and both the numerator and denominator of the fractions is 0.

In this initial configuration the tree is empty and the structs of A are stored in a stack configuration. To store this information we use the pointers `p[3]`. The nodes `A[i]` are pushed into this stack with `i` from 0 to $n-1$, in this order.

The find operation moves down the tree searching for the desired fraction. Hence for example to find the fraction 5/3 in the tree of Figure 1 we first compare 5/3 with 1/1, at the root (node `A[14]`). Since $1/1 < 5/3$ and there are no other active fractions at the root the search uses the pointer `p[1]` to node `A[13]`. Then we compare 5/3 with 3/2 and 2/1. Since $3/2 < 5/3 < 2/1$ the search uses the pointer `p[1]` to node `A[11]`. We finally find the argument fraction 5/3 in this node and therefore the return of the function is 11. Similarly if the argument fraction was 7/4 the search path would be the

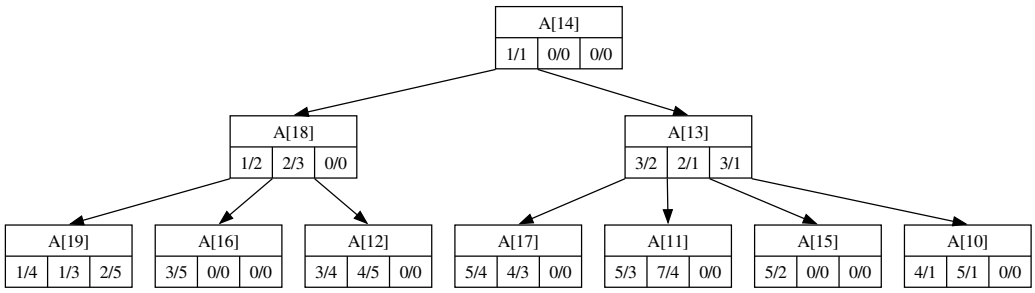

Figure 2: Example tree after inserting 7/4 and 5/1.

same, but the fraction would not be found and the function would return -1.

Note that a crucial step in executing this process is being able to compare two fractions to determine which one is largest. There are several possibilities to achieve determine this property. For approaches based on integers avoid procedures that involve overflow. The Euclidean algorithm is recommended. For approaches based on floating point representation consider the corresponding limits[1].

Now let us consider the insert function. Since there is room in the node `A[11]`, if we want to insert the fraction 7/4 we can simply add it to that node. The situation becomes trickier if after inserting 7/4 we further decide to insert 5/1. This time we would try to insert this fraction into node `A[15]`. However this node is full and therefore does not have room to support the new fraction. Hence it is necessary to create some room. Hence the fraction 3/1, which is the middle one of node `A[15]` is moved upwards to node `A[13]`. The fraction 5/2 is left in the same place, but the fraction 4/1 is moved to a new node that is taken from the stack of nodes. In this case it is `A[10]`. In general the largest fractions are always moved to whichever new node is grafted into the tree. The resulting tree is given in Figure 2.

This process of splitting nodes and moving fractions upwards to the parents is undesirable as we do not store upwards pointers and there is no point of storing the current traversed path. Instead we use the top-down approach described by Sedgewick [1998]. In this approach before moving the search to a node `A[i]` we guarantee that `A[i]` is not of type four. If this is the case we preemptively re-arrange the nodes to guarantee space in the target node. For example to insert the fraction 7/5 into this latter tree we preemptively

[1]https://wiki.sei.cmu.edu/confluence/display/c/FLP00-C.+Understand+the+limitations+of+floating-point+numbers

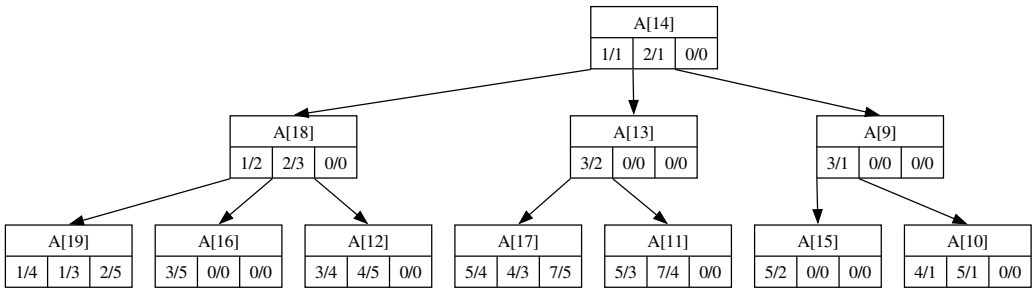

Figure 3: Example tree after inserting 7/5.

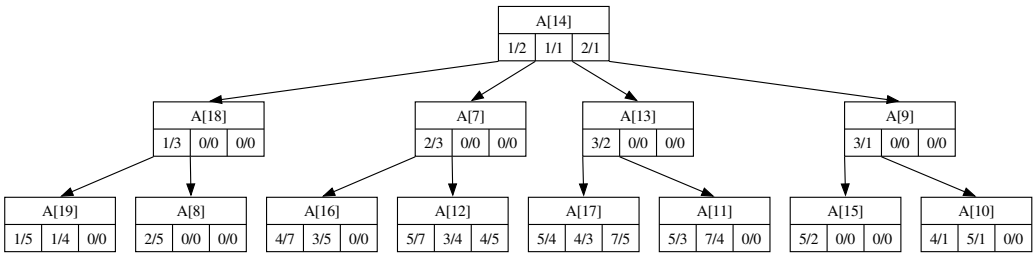

Figure 4: Example tree before splitting the root.

re-arrange the node `A[13]`, as it is a type four node in the path traversed by 7/5. The resulting tree is shown in Figure 3. In this case the fraction 2/1 is moved to the root and the fraction 3/1 is moved to the new node `A[9]`.

A special case occurs when the root itself is a node of type four, as the root has no parent node. In this case the division of the root yields a new root and the height of the tree increases. An example of such a case is given from the configuration of Figure 4 to the configuration of Figure 5. Note that in this particular case the division of the root node was not strictly necessary, as there was room in node `A[16]` to accommodate the argument fraction 5/8. However in the top-down approach we recommend this is a necessary side effect. It is simpler to increase the tree height, even if it is not strictly necessary, than to re-traverse the tree upwards if necessary. Also note that in this case we graft two new `struct node234` to the tree `A[6]` and `A[5]`. Both these nodes are poped from the node stack, where `A[6]` is poped before `A[5]`. Hence the first poped node becomes the new root and the second poped node its new child.

Whenever a node is split the program should print an information message with the following format `"Spliting node %d\n"`, where the number in question is the index `i` for node `A[i]`. As a quick recap we use a preemptive

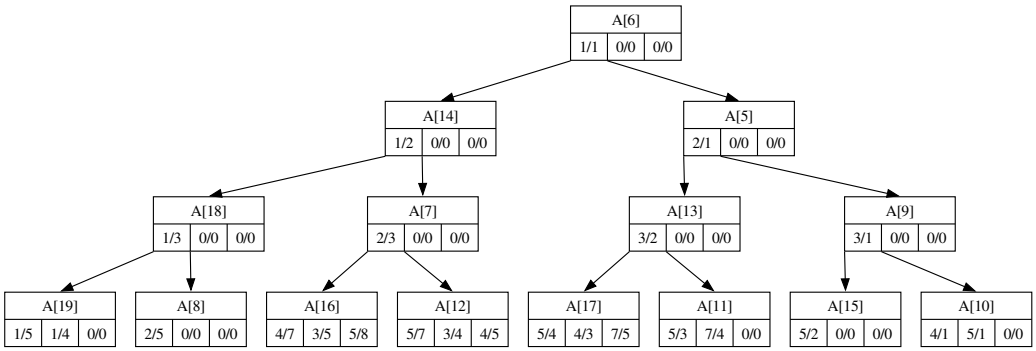

Figure 5: Example tree after inserting 5/8 on the tree of Figure 4.

top-down approach that modifies any type four node in the search path of an insertion. Such nodes are split in an appropriate way. For inner nodes we move the middle fraction to the parent node and the largest fraction (and the necessary pointers) to a new node.

For the root we first get a new node to become the new root and then proceed in a similar fashion.

Next we consider the delete operation. This operation is also implemented in with a top-down approach. In this case we seek to remove nodes of type two, from the search path. Because it could be the case that we end up deleting the only active fraction inside a node and this would be problematic. Consider for example the case of removing the fraction 2/5 from the tree of Figure 5. Since the root is of type two we preemptively join the nodes `A[14]`, `A[6]` and `A[5]`. This operation is reported with a message with the following format `"Joining nodes %d %d\n"`, in this case the values in question are 14 and 5. The fact that node `A[6]` is also involved in this process is because it is the root, which is not the case in general. Whenever there is a join operation the fractions move to the `struct node234` that initially contained the smaller fractions, in this case `A[14]`. Hence the structs `A[5]` and `A[6]` are pushed into the stack of structs, in this order, i.e., first the struct with the larger fraction 2/1 and then the former root. Next the operation proceeds to node `A[18]`, this node is also a type two node and therefore we join `A[18]` and `A[7]`, note that in this process we move 1/2 from `A[14]` to `A[18]`. Hence at this point `A[18]` contains $1/3 < 1/2 < 2/3$. Finally the operation proceeds to node `A[8]`. Again since this of type 2 some preemptive organization is required. In this case we can not join nodes `A[8]` and `A[16]`, because the resulting node would end up with five fractions $2/5 < 1/2 < 4/7 < 3/5 < 5/8$. We instead move these fractions around, so that `A[8]` ends up having two of

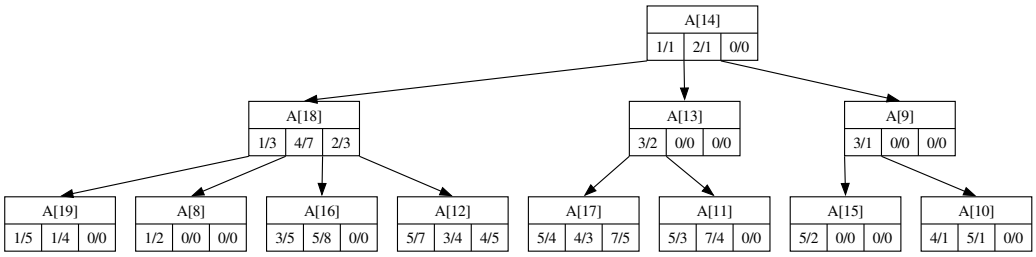

Figure 6: Example tree after deleting 2/5 from the tree of Figure 5.

them. Therefore we move 4/7 to `A[18]` and 1/2 to `A[8]`. At this point the node `A[8]` becomes a type three node and we can consider it. Finally we remove the fraction 2/7 from node `A[8]`, which ends up containing only 1/2. Hence the return result of this deletion operation is 8. This final configuration can be seen in Figure 6.

This particular deletion had the fortunate property that the fraction we wished to remove was actually part of a leaf, therefore it could simply be removed without it affecting the rest of the structure. Consider that now, from this last configuration we wish to remove the fraction 2/1, which is at the root of the tree in node `A[14]`. In this case we can not simply remove the faction as we need to separate the information between the pointers to `A[13]` and `A[9]`. On the other hand we can not keep the fraction 2/1 around for this purpose alone as all the fractions in the tree are counted as part of the sequence of elements. The solution is thus to replace 2/1 by its successor in the tree, i.e., the smallest fraction that is larger than 2/1. In this case this value is 5/2, which is conveniently located in a leaf, and as such can be "removed". Actually it is transferred to take the position of 2/1. In general the successor of a fraction contained in an internal node will always be in a leaf node.

Again we perform this computation in a top-down fashion. Hence the complete procedure is the following. First we search for 2/1 at the root, node `A[14]`. Because this node is not a leaf we do not immediately delete it. Instead we store a pointer to its position, so it can be removed once we find the successor. Hence the search now proceeds to node `A[9]`, where we are still searching for 2/1. Because this node is of type two some preemptive processing is required. We thus join the nodes `A[13]` and `A[9]`. This implies that the fraction 2/1 is actually pulled down to node `A[13]`, which now contains $3/2 < 2/1 < 3/1$. Again recall that we are searching for 2/1. It turns out that we again find it in node `A[13]`. We simply update our pointer and continue the search to node `A[15]`. Still searching for 2/1. Note that

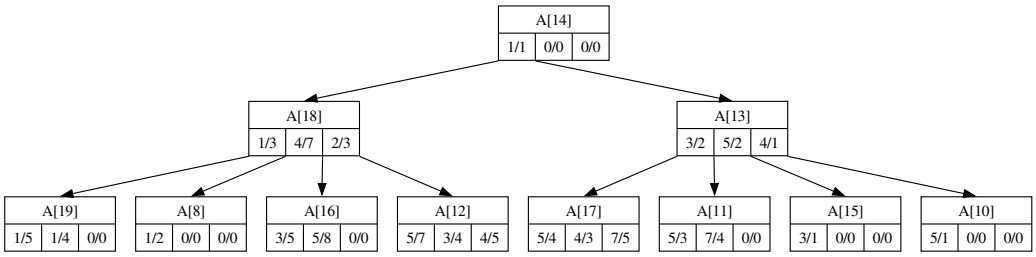

Figure 7: Example tree after deleting 2/1 from the tree of Figure 6.

node `A[15]` is also a type two node so we need to rearrange the fractions. This is done by leaving $5/2 < 3/1$ in `A[15]`, $3/2 < 2/1 < 4/1$ in `A[13]` and $5/1$ in `A[10]`. Finally the process reaches node `A[15]`, which is a leaf. We are currently looking for $2/1$ which is not in this node. However we do have a non `NULL` pointer to where we last found $2/1$ in the tree. Therefore we transfer $5/2$, which the smallest fraction of `A[15]` and the successor of $2/1$, to the location stored in the pointer. Thus $5/2$ ends up at node `A[13]`. The return value of this delete procedure is thus 15, since the successor fraction was moved from this node. This configuration is shown in Figure 7.

A final quick recap for the delete operations is that it also uses preemptive top-down approach that modifies any type two node in the search path of an deletion. Such nodes are joined with sibling nodes if both nodes are type two, otherwise the necessary fractions are simply transferred among them. In this process, when the delete path traverses the pointer `p[i]`, we preferably consider the sibling at `p[i+1]`. Only if such a sibling is non-existing do we consider the sibling at `p[i-1]`, which in this case mus necessarily exist. If the argument fraction is contained in an inner node it will be replaced by its successor fraction in the tree, otherwise it is simply deleted. The return value of this procedure is the index of a leaf of the tree, when the fraction is removed and -1 otherwise. Note that even if the argument fraction is not part of the tree the preemptive top-down approach may alter the tree structure.

1.2 Specification

To automatically validate the index we use the following conventions. The binary is executed with the following command:

```
./project < in > out
```

The file `in` contains the input commands that we will describe next. The output is stored in a file named `out`. The input and output must respect

the specification below precisely. Note that your program should **not** open or close files, instead it should read information from `stdin` and write to `stdout`. The output file will be validated against an expected result, stored in a file named `check`, with the following command:

```
diff out check
```

This command should produce no output, thus indicating that both files are identical.

The format of the input file is the following. The first line contains an integer `n`, which is the number of nodes that should be allocated to the array `A`.

The rest of the input consists in a sequence of commands. Each command consists in a letter that indicates the command to execute followed by the respective arguments, separated by spaces. Except for the `P`, `S`, `N`, `L` and `#` commands the other commands should print their return value. The commands should print the information indicated above.

The `X` terminates the execution of the program. Hence no other commands are processed after this command is found. Before terminating the program should describe the state of the date structure. First it should use the following instructions, where `S` and `root` are assumed global variables that point to the stack and root respectively.

```
printf("Final configuration:\n");
printf("S: %d ", ptr2loc(S));
printf("root: %d \n", ptr2loc(root));
```

Then it should print a list of lines that give the output of the `showNode` commands applied to all the indexes of `A` in increasing order. The final line printed by the problem should contain a single character `X`. Remember to free the array `A` now.

1.3 Sample Behaviour

The following examples show the expected `output` for the given `input`. These files are available on the course web page.

input 1

```
20
# Insert 1/1
I 1/1
S 19
# Insert 1/2
```

```
I 1/2
S 19
I 2/1
S 18
S 17
S 19
I 2/3
I 3/2
S 19
I 1/3
S 17
I 3/1
S 16
S 18
I 3/4
N
P
I 4/3
I 3/5
I 5/3
I 2/5
I 5/2
I 1/4
I 4/1
I 4/5
I 5/4
N
P
S 14
N
P
S 13
S 11
N
P
F 5/3
F 7/4
I 7/4
I 5/1
I 7/5
I 4/7
```

```
I 1/5
I 5/7
I 5/8
# Delete 2/5
D 2/5
# Delete 2/1
D 2/1
X
```

output 1

```
# Insert 1/1
19
node: 19 -1 1/1 -1 0/0 -1 0/0 -1
# Insert 1/2
19
node: 19 -1 1/2 -1 1/1 -1 0/0 -1
19
node: 18 -1 0/0 -1 0/0 -1 0/0 17
node: 17 -1 0/0 -1 0/0 -1 0/0 16
node: 19 -1 1/2 -1 1/1 -1 2/1 -1
Spliting node 19
19
17
node: 19 -1 1/2 -1 2/3 -1 0/0 -1
19
node: 17 -1 3/2 -1 2/1 -1 0/0 -1
17
node: 16 -1 0/0 -1 0/0 -1 0/0 15
node: 18 19 1/1 17 0/0 -1 0/0 -1
Spliting node 19
16
1/3 1/2 2/3 3/4 1/1 3/2 2/1 3/1
18 19 16 17
Spliting node 17
17
Spliting node 18
16
17
19
15
```

```
19
15
Spliting node 16
12
Spliting node 17
17
1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1 5/4 4/3 3/2 5/3 2/1 5/2 3/1 4/1
14 18 19 16 12 13 17 11 15
node: 14 18 1/1 13 0/0 -1 0/0 -1
1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1 5/4 4/3 3/2 5/3 2/1 5/2 3/1 4/1
14 18 19 16 12 13 17 11 15
node: 13 17 3/2 11 2/1 15 0/0 -1
node: 11 -1 5/3 -1 0/0 -1 0/0 -1
1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1 5/4 4/3 3/2 5/3 2/1 5/2 3/1 4/1
14 18 19 16 12 13 17 11 15
11
-1
11
Spliting node 15
10
Spliting node 13
17
16
Spliting node 19
19
Spliting node 18
12
Spliting node 14
16
# Delete 2/5
Joining nodes 14 5
Joining nodes 18 7
8
# Delete 2/1
Joining nodes 13 9
15
Final configuration:
S: 9 root: 14
node: 0 -1 0/0 -1 0/0 -1 0/0 -1
node: 1 -1 0/0 -1 0/0 -1 0/0 0
node: 2 -1 0/0 -1 0/0 -1 0/0 1
```

```
node: 3 -1 0/0 -1 0/0 -1 0/0 2
node: 4 -1 0/0 -1 0/0 -1 0/0 3
node: 5 -1 0/0 -1 0/0 -1 0/0 4
node: 6 -1 0/0 -1 0/0 -1 0/0 5
node: 7 -1 0/0 -1 0/0 -1 0/0 6
node: 8 -1 1/2 -1 0/0 -1 0/0 -1
node: 9 -1 0/0 -1 0/0 -1 0/0 7
node: 10 -1 5/1 -1 0/0 -1 0/0 -1
node: 11 -1 5/3 -1 7/4 -1 0/0 -1
node: 12 -1 5/7 -1 3/4 -1 4/5 -1
node: 13 17 3/2 11 5/2 15 4/1 10
node: 14 18 1/1 13 0/0 -1 0/0 -1
node: 15 -1 3/1 -1 0/0 -1 0/0 -1
node: 16 -1 3/5 -1 5/8 -1 0/0 -1
node: 17 -1 5/4 -1 4/3 -1 7/5 -1
node: 18 19 1/3 8 4/7 16 2/3 12
node: 19 -1 1/5 -1 1/4 -1 0/0 -1
X
```

input 2

```
20
# Close to 0 confusion
I 1/18446744073709550720
P
N
I 1/18446744073709551173
P
N
I 1/18446744073709551592
P
N
I 1/18446744073709550841
P
N
I 1/18446744073709550987
P
N
I 1/18446744073709551364
P
N
```

```
I 1/18446744073709550930
P
N
I 1/18446744073709550664
P
N
I 1/18446744073709551392
P
N
I 1/18446744073709551345
P
N
I 1/18446744073709550594
P
N
I 1/18446744073709551490
P
N
I 1/18446744073709551053
P
N
I 1/18446744073709551267
P
N
I 1/18446744073709550809
P
N
I 1/18446744073709550888
P
N
I 1/18446744073709550790
P
N
I 1/18446744073709550891
P
N
I 1/18446744073709550796
P
N
I 1/18446744073709550951
P
```

```
N
D 1/18446744073709551267
P
N
D 1/18446744073709550796
P
N
D 1/18446744073709550720
P
N
D 1/18446744073709551490
P
N
D 1/18446744073709551345
P
N
D 1/18446744073709550594
P
N
D 1/18446744073709550888
P
N
D 1/18446744073709551592
P
N
D 1/18446744073709550951
P
N
D 1/18446744073709551364
P
N
D 1/18446744073709550809
P
N
D 1/18446744073709551173
P
N
D 1/18446744073709550841
P
N
D 1/18446744073709550891
```

```
P
N
D 1/18446744073709551053
P
N
D 1/18446744073709550987
P
N
D 1/18446744073709551392
P
N
D 1/18446744073709550930
P
N
D 1/18446744073709550790
P
N
D 1/18446744073709550664
P
N
# Close to 1 confusion
# Common denominator
I 18446744073709551217/18446744073709551615
P
N
I 18446744073709551113/18446744073709551615
P
N
I 18446744073709551016/18446744073709551615
P
N
I 18446744073709551436/18446744073709551615
P
N
I 18446744073709551104/18446744073709551615
P
N
I 18446744073709550762/18446744073709551615
P
N
I 18446744073709551526/18446744073709551615
```

P
N
I 18446744073709550924/18446744073709551615
P
N
I 18446744073709550921/18446744073709551615
P
N
I 18446744073709551614/18446744073709551615
P
N
I 18446744073709551281/18446744073709551615
P
N
I 18446744073709551302/18446744073709551615
P
N
I 18446744073709550699/18446744073709551615
P
N
I 18446744073709551256/18446744073709551615
P
N
I 18446744073709551287/18446744073709551615
P
N
I 18446744073709551433/18446744073709551615
P
N
I 18446744073709550639/18446744073709551615
P
N
I 18446744073709550684/18446744073709551615
P
N
I 18446744073709551172/18446744073709551615
P
N
I 18446744073709550756/18446744073709551615
P
N

```
D 18446744073709551217/18446744073709551615
P
N
D 18446744073709551113/18446744073709551615
P
N
D 18446744073709551016/18446744073709551615
P
N
D 18446744073709551436/18446744073709551615
P
N
D 18446744073709551104/18446744073709551615
P
N
D 18446744073709550762/18446744073709551615
P
N
D 18446744073709551526/18446744073709551615
P
N
D 18446744073709550924/18446744073709551615
P
N
D 18446744073709550921/18446744073709551615
P
N
D 18446744073709551614/18446744073709551615
P
N
D 18446744073709551281/18446744073709551615
P
N
D 18446744073709551302/18446744073709551615
P
N
D 18446744073709550699/18446744073709551615
P
N
D 18446744073709551256/18446744073709551615
P
```

```
N
D 18446744073709551287/18446744073709551615
P
N
D 18446744073709551433/18446744073709551615
P
N
D 18446744073709550639/18446744073709551615
P
N
D 18446744073709550684/18446744073709551615
P
N
D 18446744073709551172/18446744073709551615
P
N
D 18446744073709550756/18446744073709551615
P
N
# Common numerator
I 18446744073709551615/18446744073709550932
P
N
I 18446744073709551615/18446744073709551206
P
N
I 18446744073709551615/18446744073709551314
P
N
I 18446744073709551615/18446744073709550909
P
N
I 18446744073709551615/18446744073709550851
P
N
I 18446744073709551615/18446744073709551521
P
N
I 18446744073709551615/18446744073709550987
P
N
```

```
I 18446744073709551615/18446744073709551337
P
N
I 18446744073709551615/18446744073709550936
P
N
I 18446744073709551615/18446744073709551382
P
N
I 18446744073709551615/18446744073709550906
P
N
I 18446744073709551615/18446744073709551346
P
N
I 18446744073709551615/18446744073709550993
P
N
I 18446744073709551615/18446744073709550609
P
N
I 18446744073709551615/18446744073709551067
P
N
I 18446744073709551615/18446744073709550606
P
N
I 18446744073709551615/18446744073709550977
P
N
I 18446744073709551615/18446744073709550687
P
N
I 18446744073709551615/18446744073709551293
P
N
I 18446744073709551615/18446744073709550941
P
N
D 18446744073709551615/18446744073709550932
P
```

```
N
D 18446744073709551615/18446744073709551206
P
N
D 18446744073709551615/18446744073709551314
P
N
D 18446744073709551615/18446744073709550909
P
N
D 18446744073709551615/18446744073709550851
P
N
D 18446744073709551615/18446744073709551521
P
N
D 18446744073709551615/18446744073709550987
P
N
D 18446744073709551615/18446744073709551337
P
N
D 18446744073709551615/18446744073709550936
P
N
D 18446744073709551615/18446744073709551382
P
N
D 18446744073709551615/18446744073709550906
P
N
D 18446744073709551615/18446744073709551346
P
N
D 18446744073709551615/18446744073709550993
P
N
D 18446744073709551615/18446744073709550609
P
N
D 18446744073709551615/18446744073709551067
```

```
P
N
D 18446744073709551615/18446744073709550606
P
N
D 18446744073709551615/18446744073709550977
P
N
D 18446744073709551615/18446744073709550687
P
N
D 18446744073709551615/18446744073709551293
P
N
D 18446744073709551615/18446744073709550941
P
N
# Close to infty confusion
I 18446744073709551126/1
P
N
I 18446744073709550798/1
P
N
I 18446744073709550824/1
P
N
I 18446744073709551449/1
P
N
I 18446744073709551336/1
P
N
I 18446744073709550898/1
P
N
I 18446744073709550633/1
P
N
I 18446744073709551357/1
P
```

```
N
I 18446744073709551217/1
P
N
I 18446744073709550814/1
P
N
I 18446744073709551572/1
P
N
I 18446744073709550642/1
P
N
I 18446744073709550895/1
P
N
I 18446744073709551282/1
P
N
I 18446744073709551454/1
P
N
I 18446744073709551536/1
P
N
I 18446744073709550744/1
P
N
I 18446744073709551355/1
P
N
I 18446744073709551045/1
P
N
I 18446744073709551150/1
P
N
D 18446744073709551126/1
P
N
D 18446744073709550798/1
```

```
P
N
D 18446744073709550824/1
P
N
D 18446744073709551449/1
P
N
D 18446744073709551336/1
P
N
D 18446744073709550898/1
P
N
D 18446744073709550633/1
P
N
D 18446744073709551357/1
P
N
D 18446744073709551217/1
P
N
D 18446744073709550814/1
P
N
D 18446744073709551572/1
P
N
D 18446744073709550642/1
P
N
D 18446744073709550895/1
P
N
D 18446744073709551282/1
P
N
D 18446744073709551454/1
P
N
```

```
D 18446744073709551536/1
P
N
D 18446744073709550744/1
P
N
D 18446744073709551355/1
P
N
D 18446744073709551045/1
P
N
D 18446744073709551150/1
P
N
# Common fraction confusion
I 18446743927680663697/18446743880436023772
P
N
I 18446744013580009517/18446743966335369352
P
N
I 18446743996400140353/18446743949155500236
P
N
I 18446743931975630988/18446743884730991051
P
N
I 18446743867551121623/18446743820306481866
P
N
I 18446743987810205771/18446743940565565678
P
N
I 18446743936270598279/18446743889025958330
P
N
I 18446743970630336607/18446743923385696562
P
N
I 18446744009285042226/18446743962040402073
```

```
P
N
I 18446743949155500152/18446743901910860167
P
N
I 18446743919090729115/18446743871846089214
P
N
I 18446743962040402025/18446743914795762004
P
N
I 18446743957745434734/18446743910500794725
P
N
I 18446743914795761824/18446743867551121935
P
N
I 18446743983515238480/18446743936270598399
P
N
I 18446744039349813263/18446743992105173026
P
N
I 18446743944860532861/18446743897615892888
P
N
I 18446744035054845972/18446743987810205747
P
N
I 18446743884730990787/18446743837486350982
P
N
I 18446743889025958078/18446743841781318261
P
N
D 18446743927680663697/18446743880436023772
P
N
D 18446744013580009517/18446743966335369352
P
N
```

```
D 18446743996400140353/18446743949155500236
P
N
D 18446743931975630988/18446743884730991051
P
N
D 18446743867551121623/18446743820306481866
P
N
D 18446743987810205771/18446743940565565678
P
N
D 18446743936270598279/18446743889025958330
P
N
D 18446743970630336607/18446743923385696562
P
N
D 18446744009285042226/18446743962040402073
P
N
D 18446743949155500152/18446743901910860167
P
N
D 18446743919090729115/18446743871846089214
P
N
D 18446743962040402025/18446743914795762004
P
N
D 18446743957745434734/18446743910500794725
P
N
D 18446743914795761824/18446743867551121935
P
N
D 18446743983515238480/18446743936270598399
P
N
D 18446744039349813263/18446743992105173026
P
```

```
N
D 18446743944860532861/18446743897615892888
P
N
D 18446744035054845972/18446743987810205747
P
N
D 18446743884730990787/18446743837486350982
P
N
D 18446743889025958078/18446743841781318261
P
N
X
```

output 2

```
# Close to 0 confusion
19
19
1/18446744073709550720
19
19
1/18446744073709551173 1/18446744073709550720
19
19
1/18446744073709551592 1/18446744073709551173 1/18446744073709550720
Spliting node 19
17
18 19 17
1/18446744073709551592 1/18446744073709551173 1/18446744073709550841 1/184467440
17
18 19 17
1/18446744073709551592 1/18446744073709551173 1/18446744073709550987 1/184467440
19
18 19 17
1/18446744073709551592 1/18446744073709551364 1/18446744073709551173 1/184467440
Spliting node 17
17
18 19 17 16
1/18446744073709551592 1/18446744073709551364 1/18446744073709551173 1/184467440
```

```
16
18 19 17 16
1/18446744073709551592 1/18446744073709551364 1/18446744073709551173 1/184467440
19
18 19 17 16
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
Spliting node 19
15
18 19 15 17 16
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
Spliting node 18
16
14 18 19 15 13 17 16
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
19
14 18 19 15 13 17 16
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
17
14 18 19 15 13 17 16
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
15
14 18 19 15 13 17 16
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Spliting node 16
16
14 18 19 15 13 17 16 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Spliting node 17
11
14 18 19 15 13 17 11 16 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Spliting node 13
16
14 18 19 15 13 17 11 10 16 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
11
14 18 19 15 13 17 11 10 16 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Spliting node 16
16
14 18 19 15 13 17 11 10 16 9 12
```

```
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Spliting node 11
11
14 18 19 15 13 17 11 8 10 16 9 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
15
14 18 19 15 17 13 11 8 10 16 9 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
16
14 18 19 15 17 13 11 8 10 16 9 12
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
Joining nodes 9 12
9
14 18 19 15 17 13 11 8 10 16 9
1/18446744073709551592 1/18446744073709551490 1/18446744073709551392 1/184467440
19
14 18 19 15 17 13 11 8 10 16 9
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
15
14 18 19 15 17 13 11 8 10 16 9
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
Joining nodes 13 10
9
14 18 19 15 17 13 11 8 16 9
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
Joining nodes 8 16
8
14 18 19 15 17 13 11 8 9
1/18446744073709551592 1/18446744073709551392 1/18446744073709551364 1/184467440
Joining nodes 19 15
19
14 18 19 17 13 11 8 9
1/18446744073709551392 1/18446744073709551364 1/18446744073709551173 1/184467440
11
14 18 19 17 13 11 8 9
1/18446744073709551392 1/18446744073709551364 1/18446744073709551173 1/184467440
19
14 18 19 17 11 13 8 9
1/18446744073709551392 1/18446744073709551173 1/18446744073709551053 1/184467440
8
14 18 19 17 13 11 8 9
```

```
1/18446744073709551392 1/18446744073709551173 1/18446744073709551053 1/184467440
Joining nodes 17 11
17
14 18 19 17 13 8 9
1/18446744073709551392 1/18446744073709551053 1/18446744073709550987 1/184467440
Joining nodes 18 13
Joining nodes 8 9
8
18 19 17 8
1/18446744073709551392 1/18446744073709551053 1/18446744073709550987 1/184467440
8
18 19 17 8
1/18446744073709551392 1/18446744073709551053 1/18446744073709550987 1/184467440
17
18 19 17 8
1/18446744073709551392 1/18446744073709550987 1/18446744073709550930 1/184467440
Joining nodes 17 8
17
18 19 17
1/18446744073709551392 1/18446744073709550930 1/18446744073709550790 1/184467440
19
18 19 17
1/18446744073709550930 1/18446744073709550790 1/18446744073709550664
Joining nodes 19 17
19
19
1/18446744073709550790 1/18446744073709550664
19
19
1/18446744073709550664
19


# Close to 1 confusion
# Common denominator
19
19
18446744073709551217/18446744073709551615
19
19
18446744073709551113/18446744073709551615 18446744073709551217/18446744073709551
```

```
19
19
18446744073709551016/18446744073709551615 18446744073709551113/18446744073709551
Spliting node 19
17
18 19 17
18446744073709551016/18446744073709551615 18446744073709551113/18446744073709551
19
18 19 17
18446744073709551016/18446744073709551615 18446744073709551104/18446744073709551
19
18 19 17
18446744073709550762/18446744073709551615 18446744073709551016/18446744073709551
17
18 19 17
18446744073709550762/18446744073709551615 18446744073709551016/18446744073709551
Spliting node 19
19
18 19 8 17
18446744073709550762/18446744073709551615 18446744073709550924/18446744073709551
19
18 19 8 17
18446744073709550762/18446744073709551615 18446744073709550921/18446744073709551
Spliting node 17
9
18 19 8 17 9
18446744073709550762/18446744073709551615 18446744073709550921/18446744073709551
Spliting node 18
17
14 18 19 8 13 17 9
18446744073709550762/18446744073709551615 18446744073709550921/18446744073709551
17
14 18 19 8 13 17 9
18446744073709550762/18446744073709551615 18446744073709550921/18446744073709551
Spliting node 19
19
14 18 19 11 8 13 17 9
18446744073709550699/18446744073709551615 18446744073709550762/18446744073709551
Spliting node 17
17
14 18 19 11 8 13 17 15 9
```

```
18446744073709550699/18446744073709551615 18446744073709550762/18446744073709551
15
14 18 19 11 8 13 17 15 9
18446744073709550699/18446744073709551615 18446744073709550762/18446744073709551
15
14 18 19 11 8 13 17 15 9
18446744073709550699/18446744073709551615 18446744073709550762/18446744073709551
19
14 18 19 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550699/18446744073709551
Spliting node 19
19
14 18 19 16 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
17
14 18 19 16 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Spliting node 18
16
14 18 19 16 10 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
17
14 18 19 16 10 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
17
14 18 19 16 10 11 8 13 17 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 8 17
8
14 18 19 16 10 11 8 13 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 10 13
9
14 18 19 16 10 11 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
8
14 18 19 16 10 11 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
16
14 18 19 16 11 10 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
```

```
9
14 18 19 16 11 10 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 16 11
16
14 18 19 16 10 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
16
14 18 19 16 8 10 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
9
14 18 19 16 10 8 15 9
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 15 9
15
14 18 19 16 10 8 15
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 18 10
15
18 19 16 8 15
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 16 8
16
18 19 16 15
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
16
18 19 16 15
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
Joining nodes 16 15
16
18 19 16
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
16
18 19 16
18446744073709550639/18446744073709551615 18446744073709550684/18446744073709551
19
18 19 16
18446744073709550684/18446744073709551615 18446744073709550756/18446744073709551
Joining nodes 19 16
19
19
```

```
18446744073709550756/18446744073709551615 18446744073709551172/18446744073709551
19
19
18446744073709550756/18446744073709551615
19


# Common numerator
19
19
18446744073709551615/18446744073709550932
19
19
18446744073709551615/18446744073709551206 18446744073709551615/18446744073709550
19
19
18446744073709551615/18446744073709551314 18446744073709551615/18446744073709551
Spliting node 19
16
18 19 16
18446744073709551615/18446744073709551314 18446744073709551615/18446744073709551
16
18 19 16
18446744073709551615/18446744073709551314 18446744073709551615/18446744073709551
19
18 19 16
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Spliting node 16
16
18 19 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
19
18 19 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
16
18 19 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Spliting node 19
19
18 19 8 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
```

```
Spliting node 18
15
14 18 19 8 10 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
19
14 18 19 8 10 16 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Spliting node 16
16
14 18 19 8 10 16 9 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
15
14 18 19 8 10 16 9 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
16
14 18 19 8 10 16 9 15
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Spliting node 15
11
14 18 19 8 10 16 9 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Spliting node 10
Spliting node 16
17
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
11
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
8
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
17
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
9
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
16
14 18 19 8 10 16 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
```

```
8
14 18 19 8 16 10 17 9 13 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
Joining nodes 10 13
15
14 18 19 8 16 10 17 9 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
15
14 18 19 8 16 10 17 9 15 11
18446744073709551615/18446744073709551521 18446744073709551615/18446744073709551
19
14 18 19 8 16 10 17 9 15 11
18446744073709551615/18446744073709551382 18446744073709551615/18446744073709551
Joining nodes 17 9
17
14 18 19 8 16 10 17 15 11
18446744073709551615/18446744073709551382 18446744073709551615/18446744073709551
Joining nodes 8 16
8
14 18 19 8 10 17 15 11
18446744073709551615/18446744073709551382 18446744073709551615/18446744073709551
17
14 18 19 8 10 17 15 11
18446744073709551615/18446744073709551382 18446744073709551615/18446744073709551
19
14 18 19 8 17 10 15 11
18446744073709551615/18446744073709551346 18446744073709551615/18446744073709551
Joining nodes 15 11
15
14 18 19 8 10 17 15
18446744073709551615/18446744073709551346 18446744073709551615/18446744073709551
Joining nodes 18 10
19
18 19 8 17 15
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709551
Joining nodes 8 17
8
18 19 8 15
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709551
15
18 19 8 15
```

```
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709551
8
18 19 8 15
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709550
Joining nodes 8 15
8
18 19 8
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709550
8
18 19 8
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709550
Joining nodes 19 8
19
19
18446744073709551615/18446744073709551293 18446744073709551615/18446744073709550
19
19
18446744073709551615/18446744073709550941
19


# Close to infty confusion
19
19
18446744073709551126/1
19
19
18446744073709550798/1 18446744073709551126/1
19
19
18446744073709550798/1 18446744073709550824/1 18446744073709551126/1
Spliting node 19
8
18 19 8
18446744073709550798/1 18446744073709550824/1 18446744073709551126/1 1844674407
8
18 19 8
18446744073709550798/1 18446744073709550824/1 18446744073709551126/1 1844674407
Spliting node 8
8
18 19 8 15
```

```
18446744073709550798/1 18446744073709550824/1 18446744073709550898/1 18446744073
19
18 19 8 15
18446744073709550633/1 18446744073709550798/1 18446744073709550824/1 18446744073
15
18 19 8 15
18446744073709550633/1 18446744073709550798/1 18446744073709550824/1 18446744073
8
18 19 8 15
18446744073709550633/1 18446744073709550798/1 18446744073709550824/1 18446744073
19
18 19 8 15
18446744073709550633/1 18446744073709550798/1 18446744073709550814/1 18446744073
15
18 19 8 15
18446744073709550633/1 18446744073709550798/1 18446744073709550814/1 18446744073
Spliting node 19
19
18 19 17 8 15
18446744073709550633/1 18446744073709550642/1 18446744073709550798/1 18446744073
Spliting node 18
Spliting node 8
8
14 18 19 17 10 8 11 15
18446744073709550633/1 18446744073709550642/1 18446744073709550798/1 18446744073
11
14 18 19 17 10 8 11 15
18446744073709550633/1 18446744073709550642/1 18446744073709550798/1 18446744073
Spliting node 15
16
14 18 19 17 10 8 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550798/1 18446744073
Spliting node 10
16
14 18 19 17 10 8 11 9 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550798/1 18446744073
19
14 18 19 17 10 8 11 9 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
15
14 18 19 17 10 8 11 9 15 16
```

```
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
8
14 18 19 17 10 8 11 9 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
11
14 18 19 17 10 8 11 9 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
Joining nodes 10 9
11
14 18 19 17 10 8 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
17
14 18 19 17 8 10 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
17
14 18 19 17 8 10 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
16
14 18 19 17 8 10 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
15
14 18 19 17 8 10 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
Joining nodes 17 8
17
14 18 19 17 10 11 15 16
18446744073709550633/1 18446744073709550642/1 18446744073709550744/1 18446744073
19
14 18 19 17 11 10 15 16
18446744073709550642/1 18446744073709550744/1 18446744073709550814/1 18446744073
15
14 18 19 17 10 11 15 16
18446744073709550642/1 18446744073709550744/1 18446744073709550814/1 18446744073
11
14 18 19 17 10 11 15 16
18446744073709550642/1 18446744073709550744/1 18446744073709550814/1 18446744073
17
14 18 19 17 11 10 15 16
18446744073709550642/1 18446744073709550744/1 18446744073709550895/1 18446744073
Joining nodes 15 16
15
```

```
14 18 19 17 10 11 15
18446744073709550642/1 18446744073709550744/1 18446744073709550895/1 1844674407
Joining nodes 18 10
19
18 19 17 11 15
18446744073709550744/1 18446744073709550895/1 18446744073709551045/1 1844674407
Joining nodes 17 11
17
18 19 17 15
18446744073709550744/1 18446744073709551045/1 18446744073709551150/1 1844674407
17
18 19 17 15
18446744073709550744/1 18446744073709551045/1 18446744073709551150/1 1844674407
15
18 19 17 15
18446744073709550744/1 18446744073709551045/1 18446744073709551150/1 1844674407
Joining nodes 17 15
17
18 19 17
18446744073709550744/1 18446744073709551045/1 18446744073709551150/1 1844674407
19
18 19 17
18446744073709551045/1 18446744073709551150/1 18446744073709551355/1
Joining nodes 19 17
19
19
18446744073709551045/1 18446744073709551150/1
19
19
18446744073709551150/1
19


# Common fraction confusion
19
19
18446743927680663697/18446743880436023772
19
19
18446743927680663697/18446743880436023772 18446744013580009517/1844674396633536
19
```

```
19
18446743927680663697/18446743880436023772 18446743996400140353/18446743949155500
Spliting node 19
19
18 19 17
18446743927680663697/18446743880436023772 18446743931975630988/18446743884730991
19
18 19 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
Spliting node 19
15
18 19 15 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
15
18 19 15 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
Spliting node 15
11
18 19 15 11 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
Spliting node 18
17
14 18 19 15 10 11 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
11
14 18 19 15 10 11 17
18446743867551121623/18446743820306481866 18446743927680663697/18446743880436023
19
14 18 19 15 10 11 17
18446743867551121623/18446743820306481866 18446743919090729115/18446743871846089
Spliting node 11
11
14 18 19 15 10 11 16 17
18446743867551121623/18446743820306481866 18446743919090729115/18446743871846089
11
14 18 19 15 10 11 16 17
18446743867551121623/18446743820306481866 18446743919090729115/18446743871846089
19
14 18 19 15 10 11 16 17
18446743867551121623/18446743820306481866 18446743914795761824/18446743867551121
16
```

```
14 18 19 15 10 11 16 17
18446743867551121623/18446743820306481866 18446743914795761824/18446743867551121
17
14 18 19 15 10 11 16 17
18446743867551121623/18446743820306481866 18446743914795761824/18446743867551121
Spliting node 11
11
14 18 19 15 10 11 8 16 17
18446743867551121623/18446743820306481866 18446743914795761824/18446743867551121
Spliting node 10
Spliting node 17
13
14 18 19 15 10 11 8 9 16 17 13
18446743867551121623/18446743820306481866 18446743914795761824/18446743867551121
Spliting node 19
19
14 18 19 12 15 10 11 8 9 16 17 13
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
19
14 18 19 12 15 10 11 8 9 16 17 13
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
Joining nodes 12 15
12
14 18 19 12 10 11 8 9 16 17 13
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
13
14 18 19 12 10 11 8 9 16 17 13
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
Joining nodes 17 13
17
14 18 19 12 10 11 8 9 16 17
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
Joining nodes 18 10
12
14 18 19 12 11 8 9 16 17
18446743867551121623/18446743820306481866 18446743884730990787/18446743837486350
19
14 18 19 12 11 8 9 16 17
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
16
14 18 19 12 11 9 8 16 17
```

```
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
11
14 18 19 12 11 9 8 16 17
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
16
14 18 19 12 11 9 8 16 17
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 16 17
16
14 18 19 12 11 9 8 16
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 12 11
12
14 18 19 12 9 8 16
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 18 9
12
18 19 12 8 16
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
8
18 19 12 8 16
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 8 16
8
18 19 12 8
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
12
18 19 12 8
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 12 8
12
18 19 12
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
12
18 19 12
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
12
18 19 12
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
Joining nodes 19 12
19
```

```
19
18446743884730990787/18446743837486350982 18446743889025958078/18446743841781318
19
19
18446743889025958078/18446743841781318261
19


Final configuration:
S: 19 root: -1
node: 0 -1 0/0 -1 0/0 -1 0/0 -1
node: 1 -1 0/0 -1 0/0 -1 0/0 0
node: 2 -1 0/0 -1 0/0 -1 0/0 1
node: 3 -1 0/0 -1 0/0 -1 0/0 2
node: 4 -1 0/0 -1 0/0 -1 0/0 3
node: 5 -1 0/0 -1 0/0 -1 0/0 4
node: 6 -1 0/0 -1 0/0 -1 0/0 5
node: 7 -1 0/0 -1 0/0 -1 0/0 6
node: 8 -1 0/0 -1 0/0 -1 0/0 16
node: 9 -1 0/0 -1 0/0 -1 0/0 11
node: 10 -1 0/0 -1 0/0 -1 0/0 13
node: 11 -1 0/0 -1 0/0 -1 0/0 17
node: 12 -1 0/0 -1 0/0 -1 0/0 8
node: 13 -1 0/0 -1 0/0 -1 0/0 15
node: 14 -1 0/0 -1 0/0 -1 0/0 9
node: 15 -1 0/0 -1 0/0 -1 0/0 7
node: 16 -1 0/0 -1 0/0 -1 0/0 14
node: 17 -1 0/0 -1 0/0 -1 0/0 10
node: 18 -1 0/0 -1 0/0 -1 0/0 12
node: 19 -1 0/0 -1 0/0 -1 0/0 18
X
```

input 3

```
20
# Loading ...
L
# This is the final config from T01
S: 9 root: 14
# With extra comments
node: 0 -1 0/0 -1 0/0 -1 0/0 -1
```

```
node: 1 -1 0/0 -1 0/0 -1 0/0 0
node: 2 -1 0/0 -1 0/0 -1 0/0 1
node: 3 -1 0/0 -1 0/0 -1 0/0 2
node: 4 -1 0/0 -1 0/0 -1 0/0 3
node: 5 -1 0/0 -1 0/0 -1 0/0 4
node: 6 -1 0/0 -1 0/0 -1 0/0 5
node: 7 -1 0/0 -1 0/0 -1 0/0 6
node: 8 -1 1/2 -1 0/0 -1 0/0 -1
node: 9 -1 0/0 -1 0/0 -1 0/0 7
# With extra comments
node: 10 -1 5/1 -1 0/0 -1 0/0 -1
node: 11 -1 5/3 -1 7/4 -1 0/0 -1
node: 12 -1 5/7 -1 3/4 -1 4/5 -1
node: 13 17 3/2 11 5/2 15 4/1 10
node: 14 18 1/1 13 0/0 -1 0/0 -1
node: 15 -1 3/1 -1 0/0 -1 0/0 -1
node: 16 -1 3/5 -1 5/8 -1 0/0 -1
node: 17 -1 5/4 -1 4/3 -1 7/5 -1
node: 18 19 1/3 8 4/7 16 2/3 12
node: 19 -1 1/5 -1 1/4 -1 0/0 -1
# With extra comments
X
# Loading finished
F 1/1
# Search 1/2
F 1/2
F 2/1
F 2/3
F 3/2
F 1/3
F 3/1
F 3/4
F 4/3
F 3/5
F 5/3
F 2/5
F 5/2
F 1/4
F 4/1
F 4/5
F 5/4
```

```
S 14
S 13
S 11
F 5/3
F 7/4
F 7/4
F 5/1
F 7/5
F 4/7
F 1/5
F 5/7
F 5/8
D 2/5
D 2/1
X
```

output 3

```
# Loading ...
# Loading finished
14
# Search 1/2
8
-1
18
13
18
15
12
17
16
11
-1
13
19
13
12
17
node: 14 18 1/1 13 0/0 -1 0/0 -1
node: 13 17 3/2 11 5/2 15 4/1 10
node: 11 -1 5/3 -1 7/4 -1 0/0 -1
```

```
11
11
11
10
17
18
19
12
16
-1
-1
Final configuration:
S: 9 root: 14
node: 0 -1 0/0 -1 0/0 -1 0/0 -1
node: 1 -1 0/0 -1 0/0 -1 0/0 0
node: 2 -1 0/0 -1 0/0 -1 0/0 1
node: 3 -1 0/0 -1 0/0 -1 0/0 2
node: 4 -1 0/0 -1 0/0 -1 0/0 3
node: 5 -1 0/0 -1 0/0 -1 0/0 4
node: 6 -1 0/0 -1 0/0 -1 0/0 5
node: 7 -1 0/0 -1 0/0 -1 0/0 6
node: 8 -1 1/2 -1 4/7 -1 0/0 -1
node: 9 -1 0/0 -1 0/0 -1 0/0 7
node: 10 -1 5/1 -1 0/0 -1 0/0 -1
node: 11 -1 5/3 -1 7/4 -1 0/0 -1
node: 12 -1 5/7 -1 3/4 -1 4/5 -1
node: 13 17 3/2 11 5/2 15 4/1 10
node: 14 18 1/1 13 0/0 -1 0/0 -1
node: 15 -1 3/1 -1 0/0 -1 0/0 -1
node: 16 -1 5/8 -1 0/0 -1 0/0 -1
node: 17 -1 5/4 -1 4/3 -1 7/5 -1
node: 18 19 1/3 8 3/5 16 2/3 12
node: 19 -1 1/5 -1 1/4 -1 0/0 -1
X
```

input 4

```
30
# I 1/1
I 1/1
# P
```

```
P
# N
N
# I 2/1
I 2/1
# P
P
# N
N
# I 1/2
I 1/2
# P
P
# N
N
# I 1/3
I 1/3
# P
P
# N
N
# I 2/5
I 2/5
# P
P
# N
N
# I 3/2
I 3/2
# P
P
# N
N
# I 3/1
I 3/1
# P
P
# N
N
# I 1/4
I 1/4
```

```
# P
P
# N
N
# I 4/1
I 4/1
# P
P
# N
N
# I 4/3
I 4/3
# P
P
# N
N
# I 5/1
I 5/1
# P
P
# N
N
# I 6/1
I 6/1
# P
P
# N
N
# I 5/3
I 5/3
# P
P
# N
N
# I 7/4
I 7/4
# P
P
# N
N
# F 7/2
```

```
F 7/2
# P
P
# N
N
# F 5/2
F 5/2
# P
P
# N
N
# I 7/2
I 7/2
# P
P
# N
N
# D 1/2
D 1/2
# P
P
# N
N
# D 4/1
D 4/1
# P
P
# N
N
# I 9/5
I 9/5
# P
P
# N
N
# I 5/2
I 5/2
# P
P
# N
N
```

```
# D 3/2
D 3/2
# P
P
# N
N
# D 2/1
D 2/1
# P
P
# N
N
# D 5/3
D 5/3
# P
P
# N
N
# D 5/2
D 5/2
# P
P
# N
N
# D 7/4
D 7/4
# P
P
# N
N
# D 3/1
D 3/1
# P
P
# N
N
# D 4/3
D 4/3
# P
P
# N
```

```
N
# D 7/2
D 7/2
# P
P
# N
N
# I 7/1
I 7/1
# P
P
# N
N
# I 3/8
I 3/8
# P
P
# N
N
# D 1/1
D 1/1
# P
P
# N
N
# I 8/1
I 8/1
# P
P
# N
N
# I 9/1
I 9/1
# P
P
# N
N
# D 9/5
D 9/5
# P
P
```

```
# N
N
# D 9/1
D 9/1
# P
P
# N
N
# D 10/1
D 10/1
# P
P
# N
N
# D 11/2
D 11/2
# P
P
# N
N
# D 9/1
D 9/1
# P
P
# N
N
# D 7/6
D 7/6
# P
P
# N
N
# D 2/7
D 2/7
# P
P
# N
N
# D 1/3
D 1/3
# P
```

```
P
# N
N
# D 6/1
D 6/1
# P
P
# N
N
# D 9/1
D 9/1
# P
P
# N
N
# I 7/6
I 7/6
# P
P
# N
N
# D 9/1
D 9/1
# P
P
# N
N
# D 1/3
D 1/3
# P
P
# N
N
# D 9/1
D 9/1
# P
P
# N
N
# I 9/1
I 9/1
```

```
# P
P
# N
N
# D 5/1
D 5/1
# P
P
# N
N
# I 10/1
I 10/1
# P
P
# N
N
# F 19/2
F 19/2
# P
P
# N
N
# I 11/1
I 11/1
# P
P
# N
N
# I 1/5
I 1/5
# P
P
# N
N
# I 1/3
I 1/3
# P
P
# N
N
# D 2/1
```

```
D 2/1
# P
P
# N
N
# I 15/2
I 15/2
# P
P
# N
N
# D 7/1
D 7/1
# P
P
# N
N
# I 12/1
I 12/1
# P
P
# N
N
# D 23/3
D 23/3
# X
X
```

output 4

```
# I 1/1
29
# P
29
# N
1/1
# I 2/1
29
# P
29
# N
```

```
1/1 2/1
# I 1/2
29
# P
29
# N
1/2 1/1 2/1
# I 1/3
Spliting node 29
29
# P
28 29 27
# N
1/3 1/2 1/1 2/1
# I 2/5
29
# P
28 29 27
# N
1/3 2/5 1/2 1/1 2/1
# I 3/2
27
# P
28 29 27
# N
1/3 2/5 1/2 1/1 3/2 2/1
# I 3/1
27
# P
28 29 27
# N
1/3 2/5 1/2 1/1 3/2 2/1 3/1
# I 1/4
Spliting node 29
29
# P
28 29 26 27
# N
1/4 1/3 2/5 1/2 1/1 3/2 2/1 3/1
# I 4/1
Spliting node 27
```

```
25
# P
28 29 26 27 25
# N
1/4 1/3 2/5 1/2 1/1 3/2 2/1 3/1 4/1
# I 4/3
Spliting node 28
27
# P
24 28 29 26 23 27 25
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 2/1 3/1 4/1
# I 5/1
25
# P
24 28 29 26 23 27 25
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 2/1 3/1 4/1 5/1
# I 6/1
Spliting node 25
22
# P
24 28 29 26 23 27 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 2/1 3/1 4/1 5/1 6/1
# I 5/3
27
# P
24 28 29 26 23 27 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 5/3 2/1 3/1 4/1 5/1 6/1
# I 7/4
Spliting node 27
21
# P
24 28 29 26 23 27 21 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 5/3 7/4 2/1 3/1 4/1 5/1 6/1
# F 7/2
-1
# P
```

```
24 28 29 26 23 27 21 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 5/3 7/4 2/1 3/1 4/1 5/1 6/1
# F 5/2
-1
# P
24 28 29 26 23 27 21 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 5/3 7/4 2/1 3/1 4/1 5/1 6/1
# I 7/2
Spliting node 23
25
# P
24 28 29 26 23 27 21 20 25 22
# N
1/4 1/3 2/5 1/2 1/1 4/3 3/2 5/3 7/4 2/1 3/1 7/2 4/1 5/1 6/1
# D 1/2
Joining nodes 28 23
Joining nodes 26 27
26
# P
24 28 29 26 21 20 25 22
# N
1/4 1/3 2/5 1/1 4/3 3/2 5/3 7/4 2/1 3/1 7/2 4/1 5/1 6/1
# D 4/1
22
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 3/2 5/3 7/4 2/1 3/1 7/2 5/1 6/1
# I 9/5
21
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 3/2 5/3 7/4 9/5 2/1 3/1 7/2 5/1 6/1
# I 5/2
25
# P
24 28 29 26 20 21 25 22
# N
```

```
1/4 1/3 2/5 1/1 4/3 3/2 5/3 7/4 9/5 2/1 5/2 3/1 7/2 5/1 6/1
# D 3/2
21
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 5/3 7/4 9/5 2/1 5/2 3/1 7/2 5/1 6/1
# D 2/1
25
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 5/3 7/4 9/5 5/2 3/1 7/2 5/1 6/1
# D 5/3
21
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 7/4 9/5 5/2 3/1 7/2 5/1 6/1
# D 5/2
25
# P
24 28 29 26 20 21 25 22
# N
1/4 1/3 2/5 1/1 4/3 7/4 9/5 3/1 7/2 5/1 6/1
# D 7/4
Joining nodes 21 25
21
# P
24 28 29 26 20 21 22
# N
1/4 1/3 2/5 1/1 4/3 9/5 3/1 7/2 5/1 6/1
# D 3/1
Joining nodes 28 20
21
# P
28 29 26 21 22
# N
1/4 1/3 2/5 1/1 4/3 9/5 7/2 5/1 6/1
# D 4/3
26
```

```
# P
28 29 26 21 22
# N
1/4 1/3 2/5 1/1 9/5 7/2 5/1 6/1
# D 7/2
Joining nodes 21 22
21
# P
28 29 26 21
# N
1/4 1/3 2/5 1/1 9/5 5/1 6/1
# I 7/1
21
# P
28 29 26 21
# N
1/4 1/3 2/5 1/1 9/5 5/1 6/1 7/1
# I 3/8
29
# P
28 29 26 21
# N
1/4 1/3 3/8 2/5 1/1 9/5 5/1 6/1 7/1
# D 1/1
26
# P
28 29 26 21
# N
1/4 1/3 3/8 2/5 9/5 5/1 6/1 7/1
# I 8/1
21
# P
28 29 26 21
# N
1/4 1/3 3/8 2/5 9/5 5/1 6/1 7/1 8/1
# I 9/1
Spliting node 21
22
# P
28 29 26 21 22
# N
```

```
1/4 1/3 3/8 2/5 9/5 5/1 6/1 7/1 8/1 9/1
# D 9/5
Joining nodes 26 21
26
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1 9/1
# D 9/1
22
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 10/1
-1
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 11/2
-1
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 9/1
-1
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 7/6
-1
# P
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 2/7
-1
# P
```

```
28 29 26 22
# N
1/4 1/3 3/8 2/5 5/1 6/1 7/1 8/1
# D 1/3
29
# P
28 29 26 22
# N
1/4 3/8 2/5 5/1 6/1 7/1 8/1
# D 6/1
26
# P
28 29 26 22
# N
1/4 3/8 2/5 5/1 7/1 8/1
# D 9/1
Joining nodes 26 22
-1
# P
28 29 26
# N
1/4 3/8 2/5 5/1 7/1 8/1
# I 7/6
Spliting node 26
26
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 5/1 7/1 8/1
# D 9/1
-1
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 5/1 7/1 8/1
# D 1/3
-1
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 5/1 7/1 8/1
```

```
# D 9/1
-1
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 5/1 7/1 8/1
# I 9/1
22
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 5/1 7/1 8/1 9/1
# D 5/1
22
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 7/1 8/1 9/1
# I 10/1
22
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 7/1 8/1 9/1 10/1
# F 19/2
-1
# P
28 29 26 22
# N
1/4 3/8 2/5 7/6 7/1 8/1 9/1 10/1
# I 11/1
Spliting node 22
21
# P
28 29 26 22 21
# N
1/4 3/8 2/5 7/6 7/1 8/1 9/1 10/1 11/1
# I 1/5
Spliting node 28
29
# P
```

```
24 28 29 26 20 22 21
# N
1/5 1/4 3/8 2/5 7/6 7/1 8/1 9/1 10/1 11/1
# I 1/3
Spliting node 29
25
# P
24 28 29 25 26 20 22 21
# N
1/5 1/4 1/3 3/8 2/5 7/6 7/1 8/1 9/1 10/1 11/1
# D 2/1
-1
# P
24 28 29 25 26 20 22 21
# N
1/5 1/4 1/3 3/8 2/5 7/6 7/1 8/1 9/1 10/1 11/1
# I 15/2
22
# P
24 28 29 25 26 20 22 21
# N
1/5 1/4 1/3 3/8 2/5 7/6 7/1 15/2 8/1 9/1 10/1 11/1
# D 7/1
22
# P
24 28 29 25 20 26 22 21
# N
1/5 1/4 1/3 3/8 2/5 7/6 15/2 8/1 9/1 10/1 11/1
# I 12/1
21
# P
24 28 29 25 20 26 22 21
# N
1/5 1/4 1/3 3/8 2/5 7/6 15/2 8/1 9/1 10/1 11/1 12/1
# D 23/3
-1
# X
Final configuration:
S: 27 root: 24
node: 0 -1 0/0 -1 0/0 -1 0/0 -1
node: 1 -1 0/0 -1 0/0 -1 0/0 0
```

```
node: 2 -1 0/0 -1 0/0 -1 0/0 1
node: 3 -1 0/0 -1 0/0 -1 0/0 2
node: 4 -1 0/0 -1 0/0 -1 0/0 3
node: 5 -1 0/0 -1 0/0 -1 0/0 4
node: 6 -1 0/0 -1 0/0 -1 0/0 5
node: 7 -1 0/0 -1 0/0 -1 0/0 6
node: 8 -1 0/0 -1 0/0 -1 0/0 7
node: 9 -1 0/0 -1 0/0 -1 0/0 8
node: 10 -1 0/0 -1 0/0 -1 0/0 9
node: 11 -1 0/0 -1 0/0 -1 0/0 10
node: 12 -1 0/0 -1 0/0 -1 0/0 11
node: 13 -1 0/0 -1 0/0 -1 0/0 12
node: 14 -1 0/0 -1 0/0 -1 0/0 13
node: 15 -1 0/0 -1 0/0 -1 0/0 14
node: 16 -1 0/0 -1 0/0 -1 0/0 15
node: 17 -1 0/0 -1 0/0 -1 0/0 16
node: 18 -1 0/0 -1 0/0 -1 0/0 17
node: 19 -1 0/0 -1 0/0 -1 0/0 18
node: 20 26 15/2 22 10/1 21 0/0 -1
node: 21 -1 11/1 -1 12/1 -1 0/0 -1
node: 22 -1 8/1 -1 9/1 -1 0/0 -1
node: 23 -1 0/0 -1 0/0 -1 0/0 19
node: 24 28 3/8 20 0/0 -1 0/0 -1
node: 25 -1 1/3 -1 0/0 -1 0/0 -1
node: 26 -1 2/5 -1 7/6 -1 0/0 -1
node: 27 -1 0/0 -1 0/0 -1 0/0 23
node: 28 29 1/4 25 0/0 -1 0/0 -1
node: 29 -1 1/5 -1 0/0 -1 0/0 -1
X
```

2 Grading

The mooshak system is configured for a total 40 points. The project accounts for 4.0 values of the final grade. Hence to obtain the contribution of the project to the final grade divide the number of points by 10. To obtain a grading in an absolute scale to 20 divide the number of points by 2.

Each test has a specific set of points. The first four tests correspond to the input output examples given in this script. These tests are public and will be returned back by the system. The tests numbered from 5 to 12 correspond to increasingly harder test cases, brief descriptions are given

by the system. Tests 13 and 14 are verified by the valgrind[2] tool. Test 13 checks for the condition `ERROR SUMMARY: 0 errors from 0 contexts` and test 14 for the condition `All heap blocks were freed -- no leaks are possible`. Test 15 to 17 are verified by the lizzard[3] tool, the test passes if the `No thresholds exceeded` message is given. Test 15 uses the arguments `-T cyclomatic_complexity=27`; test 16 the argument `-T length=150`; test 17 the argument `-T parameter_count=9 -T token_count=700`. To obtain the score of tests from 13 to 17 must it is necessary obtain the correct output, besides the conditions just described.

The mooshak system accepts the C programming language, click on `Help` button for the respective compiler. Projects that do not compile in the mooshak system will be graded 0. Only the code that compiles in the mooshak system will be considered, commented code, will not be considered for evaluation.

Submissions to the mooshak system should consist of a single file. The system identifies the language through the file extension, an extension `.c` means the C language. The compilation process should produce absolutely no errors or warnings, otherwise the file will not compile. The resulting binary should behave exactly as explained in the specification section. Be mindful that `diff` will produce output even if a single character is different, such as a space or a newline.

Notice that you can submit to mooshak several times, but there is a 10 minute waiting period before submissions. You are strongly advised to submit several times and as early as possible. Only the last version is considered for grading purposes, all other submissions are ignored. There will be **no** deadline extensions. Submissions by email will **not** be accepted.

3 Debugging Suggestions

There are several tools that can be used to help in debugging your project implementation. For very a simple verification a carefully placed `printf` command can prove most useful. Likewise it is also considered good practice to use the `assert` command to have your program automatically verify certain desirable properties. The flag `-D NDEBUG` was added to the gcc command of mooshak. This means that you may submit your code without needing to remove the `assert` commands, as they are removed by the pre-processor. Also if you wish to include code that gets automatically removed from the submission you can use `#ifndef NDEBUG`. Here is a simple example:

[2] `https://www.valgrind.org/`

[3] `https://github.com/terryyin/lizard`

```
#ifndef NDEBUG
      /* sprintf(fname, "tree%.3d.dot", fc); */
      /* fistr = fopen(fname, "w"); */
      /* vizShow(fistr, n); */
      /* fclose(fistr); */
      /* fc++; */
#endif /* NDEBUG */
```

The following functions may also prove helpful.

```
void
structLoad(void)
{
  int i;
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wlong-long"
  unsigned long long int j;
#pragma GCC diagnostic pop
  int k;
  char c;
  char *tok;
  size_t len = 1<<8;
  char *line = (char*)malloc(len*sizeof(char));

  line[0] = '#';
  while('#' == line[0]) getline(&line, &len, stdin);
  tok = strtok(line, " ");
  tok = strtok(NULL, " ");
  sscanf(tok, "%d", &i);
  S = NULL;
  if(-1 != i)
    S = &A[i];

  tok = strtok(NULL, " ");
  tok = strtok(NULL, " ");
  sscanf(tok, "%d", &i);
  root = NULL;
  if(-1 != i)
    root = &A[i];

  while(-1 != getline(&line, &len, stdin) &&
        'X' != line[0]){
```

```
    char *tok = strtok(line, " ");

    if(0 == strcmp("node:", tok)){
      tok = strtok(NULL, " ");
      sscanf(tok, "%d", &i);

      k = 0;
      while(true){
        tok = strtok(NULL, " ");
        sscanf(tok, "%llu", &j);
        A[i].p[k] = NULL;
        if(-1 != j)
          A[i].p[k] = &A[j];

        if(3 == k) break;

        tok = strtok(NULL, "/");
        sscanf(tok, "%llu", &j);
        A[i].V[k].a = j;
        tok = strtok(NULL, " ");
        sscanf(tok, "%llu", &j);
        A[i].V[k].b = j;
        k++;
      }
    }
  }
  free(line);
}

void
vizShow(FILE *f, int n)
{
  int i;
  node234 *Q = calloc(n+1, sizeof(node234)); /* Queue of nodes to print */
  int in = 0;
  int out = 0;

  fprintf(f, "digraph {\n");
  fprintf(f, "node [shape=record];");
  fprintf(f, "splines=false;\n");
  Q[in] = root;
```

```
in++;
while(NULL != Q[out]){ /* Non-empty Queue */
  i = ptr2loc(Q[out]);
  fprintf(f, "A%d [label=\"{<h> A[%d] |{<p0> %llu/%llu |<p1> %llu/%llu |<p2> %
          i, i,
          A[i].V[0].a, A[i].V[0].b,
          A[i].V[1].a, A[i].V[1].b,
          A[i].V[2].a, A[i].V[2].b
          );

  i = 0;
  while(NULL != Q[out]->p[i] && i < 4){
    Q[in] = Q[out]->p[i];
    in++;
    i++;
  }
  out++;
}

in = 0;
out = 0;
Q[in] = root;
in++;
while(NULL != Q[out]){ /* Non-empty Queue */
  i = 0;
  while(NULL != Q[out]->p[i] && i < 4){
    if(3 == i)
      fprintf(f, "A%d:p2:se -> A%d:h:n\n",
              ptr2loc(Q[out]),
              ptr2loc(Q[out]->p[i]));
    else
      fprintf(f, "A%d:p%d:sw -> A%d:h:n\n",
              ptr2loc(Q[out]), i,
              ptr2loc(Q[out]->p[i]));

    Q[in] = Q[out]->p[i];
    in++;
    i++;
  }
  out++;
}
```

```
  fprintf(f, "}\n");
  free(Q);
}
```

The `vizShow` function produces a description of the current state of your data structure in the `dot` language, see `https://graphviz.org/`. A code snippet of how to invoke this function was given with the `NDEBUG` macro example. To produce a `pdf` file with the corresponding image you may use the command `dot -Tpdf -O tree000.dot`.

The `structLoad` function can be used to load a configuration directly into the array `A`, without having to specify a sequence of commands that leads to that configuration.

For more complex debugging sessions it may be necessary to use a debugger, such as `gdb`, see `https://www.sourceware.org/gdb/`

The use of the `valgrind` tool, for memory verification is also highly recommended, see `https://valgrind.org/`

References

R. Sedgewick. *Algorithms in C*. Algorithms in C. Addison-Wesley, 1998. ISBN 9780201314526. URL `https://books.google.pt/books?id=Bf7XAAAAMAAJ`.