

# 2021\_DS\_Fall\_HW2

---

繳交日期：2021/11/06 11:59PM前

## Program Requirement

---

- Programming Language: C
- Execution Environment
  - CPU core: 1
  - Memory: 2 GB
  - Execution time limit: 1 second
  - C Compiler: `gcc`
    - Compiled with `-O2 -std=c11 -Wall`
  - C Standard: [C 11](#)
  - Read input from [stdin](#) and write your output to [stdout](#).
  - Use header file only from [C Standard Library](#).
- Submit your answer as a **C source code** file, no executable file.
  - Your program will be compiled by us.

## Homework #11 \*

---

Write a function, `insertLeft` and `insertRight`, that inserts a new node, `child`, as the left child of node `parent` in a threaded binary tree. The left child pointer of `parent` becomes the left child pointer of `child`.

## Input Format

---

Input consist  $n + 1$  line.

- First line contains two numbers  $n$  and  $r$ , each separated by whitespace.
  - $n$  represents the number of threaded binary tree insertion operation.
  - $r$  represents the node id of the root node.
- Following input consist of  $n$  line. Each line declares a threaded binary tree child node insertion.
  - Each line  $l$  contains three elements  $p_l, O_l, c_l$ , each element is separated by whitespace.
  - $p_l$  is the id number of the parent node during insertion.
  - $O_l$  is a string `left` or `right`. It represent the insertion direction (insert as left child or insert as right child).
  - $c_l$  is the id number of the newly inserted node.

## Output Format

---

Output a series of numbers, each number separated by whitespace.

These numbers describe the inorder traversal of the threaded binary tree.

## Technical Specification

- $1 \leq n \leq 10^3$ .
- $0 < r, c_0, c_1, \dots, c_n < 10^9$
- All node id numbers  $r, c_0, c_1, \dots, c_n$  are unique.
- Describe how your thread binary tree insertion works in the homework report.

## Sample Input

```
8 1
1 left 2
1 right 3
2 left 4
2 right 5
3 left 6
3 right 7
4 left 8
4 right 9
```

## Sample Output

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

## Homework #12 \*

[Programming Project]

Write a user-friendly, menu-driven program that allows the user to perform the following operations on min heaps.

1. Create a min heap.
2. Remove the key with the lowest value.
3. Change the priority of an arbitrary element.
4. Insert an element into the heap.

## Technical Specification

- Complete this programming project.
- No strict input/output or operation defined in this homework. Just write a user-friendly, menu-driven program that supports those 4 functional requirements.
- Demonstrate how to use your program in the homework report. Must cover all those 4 requirements.
- Describe how your min heap works in the homework report.

## Homework #13

Write a function `heightUnion` that uses the *hight rule* for union operations instead of the weighting rule. This rule is defined below:

**Definition[Height Rule]:**

If the height of tree  $i$  is less than that of tree  $j$ , then make  $j$  the parent of  $i$ .

Your function must run in  $O(1)$  time and should maintain the hight of each tree as a negative number in the *parent* filed of the root.

## Input Format

---

A line consists of a number *round*. There will be *round* testcases to be run.

In each testcase, a line with 2 integers *n*, *ops* are given. *n* is the number of elements of the whole set. Elements are labeled from 0 to (*n* − 1) . *ops* is the number of operations you need to do. *ops* operations will be given. There 3 operations as follows.

- `union a1 b1`
- `find a2`
- `same a3 b3`

(Note: If two sets with same height perform unioning, the first set `a1` be the parent)

## Output Format

---

- Each time you gets the operation `find`, output the root of the set.
- Each time you gets the operation `same`, output `true` if they are in the same set. Output `false` if they are not in the same set.

## Technical Specification

---

- $1 \leq n \leq 10^4$
- $1 \leq ops \leq 10^4$

## Sample Input

---

```
2
10 5
union 0 1
union 2 0
find 2
same 2 1
same 2 4
1000 1
find 999
```

## Sample Output

---

```
0
true
false
999
```

## Homework #14 \*

---

Experiment with function `weightedUnion` (Program 5.20) and `heightUnion` to determine which one produces better results when used in conjunction with function `collapsingFind` (Program 5.21).

## Technical Specification

---

- Perform the experiment.

- Write down your thought in the homework report.
- Remember to provide material(experiment result) to support your thought.

## Homework #15

---

Write an algorithm to construct the binary tree with given

- Preorder sequence and inorder sequence
- Postorder sequence and inorder sequence

## Input Format

---

Input consists of  $1 + 4m$  line.

- First line contains one number  $m$ , represent how many test dataset in the following input.
- Following input contains  $4m$  lines describe  $m$  test dataset, each test dataset composed of 4 lines.
  - First line of the test dataset contains one string  $S$ , it will be preorder-inorder or postorder-inorder.
  - Second-line of the test dataset contains one number  $n$ , represent how many nodes in the binary tree.
  - Third-line of the test dataset contains a series of numbers.  $i_0, i_1, \dots, i_n$ . It represent the preorder or postorder sequence of specific binary tree, The decision is based on what  $S$  is. If  $S$  is preorder-inorder, then it is preorder sequence. If  $S$  is postorder-inorder, then it is postorder sequence.
  - Forth-line of the test dataset contains a series of numbres.  $j_0, j_1, \dots, j_n$ . It represent the inorder sequence of specific binary tree.

## Output Format

---

The output should consist of  $m$  lines.

For each test dataset, output one line contains a series of numbers. Each is separated by one whitespace.

- If the  $S$  in test dataset is preorder-inorder, then output the **postorder** sequence of the reconstructed binary tree.
- If the  $S$  in test dataset is postorder-inorder, then output the **preorder** sequence of the reconstructed binary tree.

## Technical Specification

---

- $1 \leq m \leq 10^3$
- $1 \leq n \leq 10^3$
- $1 \leq i_0, i_1, \dots, i_n \leq n$ , all  $i_x$  in  $i_0, i_1, \dots, i_n$  are unique.
- $1 \leq j_0, j_1, \dots, j_n \leq n$ , all  $j_x$  in  $j_0, j_1, \dots, j_n$  are unique.

## Sample Input

---

```

2
preorder-inorder
7
1 2 3 4 5 6 7
3 2 4 1 6 5 7
postorder-inorder
10
5 6 4 7 3 8 2 10 9 1
5 4 6 3 7 2 8 1 9 10

```

## Sample Output

```

3 4 2 6 7 5 1
1 2 3 4 5 6 7 8 9 10

```

## Homework #16

Rewrite `dfs` so that it uses an adjacency matrix representation of graphs.

## Input Format

First line of the input consist of one number  $n$ , represent how many datasets in the following input.

Each dataset consists of  $m + 1$  line.

- First line of the dataset contains two numbers  $m, t$ .
  - $m$  represents the number of vertices in the given graph.
  - $t$  is a vertex index, it represents the entrypoint of the dfs traversal.
- The rest of the  $m$  lines in the dataset describe a  $m \times m$  matrix which describes an **undirected graph** in the adjacency matrix.

## Output Format

For each dataset, output one line.

Each line consist of  $n$  numbers. It represent the `dfs` visit order of the given graph.

## Technical Specification

- $1 < n < 100$
- $1 < m < 100$
- The given graph will be **undirected**.
- When there are multiple vertices available, always start from the vertex with the smallest index.

## Sample Input

```

2
6 0
0 1 0 0 1 0
1 0 1 0 1 0
0 1 0 1 0 0

```

```

0 0 1 0 1 1
1 1 0 1 0 0
0 0 0 1 0 0
7 0
0 0 1 0 1 0 0
0 0 1 1 0 0 0
1 1 0 0 0 1 0
0 1 0 0 1 1 1
1 0 0 1 0 0 1
0 0 1 1 0 0 1
0 0 0 1 1 1 0

```

## Sample Output

```

0 1 2 3 4 5
0 2 1 3 4 6 5

```

## Homework #17

Rewrite `bfs` so that it uses an adjacency matrix representation of graphs.

- When there are multiple vertices available, always start from the vertex with smallest index.

## Input Format

First line of the input consists of one number  $n$ . It represents how many datasets are in the following input.

Each dataset consists of  $m + 1$  line.

- First line of the dataset contains two numbers  $m, t$ .
  - $m$  represents the number of vertices in the given graph.
  - $t$  is a vertex index, it represents the entry point of the bfs traversal.
- The rest of the  $m$  lines in the dataset describe a  $m \times m$  matrix which describes an **undirected graph** in adjacency matrix.

## Output Format

For each dataset, output one line.

Each line consists of  $n$  numbers. It represents the `dfs` visit order of the given graph.

## Technical Specification

- $1 < n < 100$
- $1 < m < 100$
- The given graph will be **undirected**.
- When there are multiple vertices available, always start from the vertex with the smallest index.

## Sample Input

```

2
6 0

```

```

0 1 0 0 1 0
1 0 1 0 1 0
0 1 0 1 0 0
0 0 1 0 1 1
1 1 0 1 0 0
0 0 0 1 0 0
7 0
0 0 1 0 1 0 0
0 0 1 1 0 0 0
1 1 0 0 0 1 0
0 1 0 0 1 1 1
1 0 0 1 0 0 1
0 0 1 1 0 0 1
0 0 0 1 1 1 0

```

## Sample Output

```

0 1 4 2 3 5
0 2 4 1 5 3 6

```

## Homework #18

Write a C function that finds a minimum cost spanning tree using Kruskal's algorithm.

You may use the `union` and `find` functions from Chapter 5 and the `sort` function from Chapter 1 or the `min heap` functions from Chapter 5.

## Input Format

First line of input consists of two numbers  $V$  and  $E$ .  $V$  represents the vertices count.  $E$  represents the number of edges in this undirected graph.

The rest of the input contains  $E$  lines. Each line  $e$  contains three numbers  $u_e$ ,  $v_e$  and  $c_e$ . It declares an undirected edge from  $u_e$  vertex to  $v_e$  vertex with a cost  $c_e$  along with this path.

## Output Format

The output consists of one number,  $C$ . It represents the sum of all edge cost in the minimum spanning tree in terms of the given graph.

## Technical Specification

- $1 < V \leq 10^6$
- $V - 1 \leq E \leq \min(\frac{V(V-1)}{2}, 2 \times 10^6)$
- $1 < c_0, c_1, \dots, c_E < 10^6$

## Sample Input

```
7 9
0 1 28
0 5 10
1 2 16
5 4 25
6 4 24
1 6 14
3 4 22
2 3 12
3 6 18
```

## Sample Output

```
99
```

## Homework #19

Let  $T$  be a tree with root  $v$ . The edges of  $T$  are undirected. Edge in  $T$  has a nonnegative length. Write a C function to determine the length of the shortest paths from  $v$  to the remaining vertices of  $T$ . Your function should have complexity  $O(n)$ , where  $n$  is the number of vertices in  $T$ . Show that this is the case.

## Input Format

The input describes a tree topology in a graph way.

The first line of the input is a number  $V$ . It represents the vertices count. Each vertex has one id, it ranges from 1 to  $V$ .

The rest of the input contains  $V - 1 + 1$  lines.

- For the first  $V - 1$  lines, each line consists of three numbers  $s, t$  and  $c$ . It describes a edge between  $s$  vertex and  $t$  vertex in this undirected graph.
- The last line of input consists of one number  $v$ , which represents the root of the tree. You are going to calculate the shortest path to each tree child node from here.

## Output Format

The output consists of  $V$  lines.

Each line  $i$  consists of two numbers  $i$  and  $C_i$ .

- $i$  represents the index of vertex  $i$ .
- $C_i$  represents the cost to walk from vertex  $v$  to this vertex  $i$ .

## Technical Specification

- The graph in input has no cycle.
- Tree definition,  $\forall v, u \in T$ , there is only one path to connect  $v$  and  $u$ .
- $1 < V \leq 10^6$
- $1 \leq s, t \leq V$ , For each edge  $s \neq t$
- $1 \leq c \leq 500$

## Sample Input



```
10
1 2 110
1 3 150
1 4 100
2 5 50
2 6 80
3 7 120
3 8 150
4 9 200
4 10 400
1
```

## Sample Output

---

```
1 0
2 110
3 150
4 100
5 160
6 190
7 270
8 300
9 300
10 500
```

## Homework #20 \*

---

Compare the performance of leftist trees and min heaps under the assumption that the only operations to be performed are insert and delete min. For this, do the following:

- Create a random list of  $n$  elements and a random sequence of insert and delete-min operations of length  $m$ . The latter sequence is created such that the probability of an insert or delete-min operation is approximately 0.5. Initialize a min leftist tree and a min heap to contain the  $n$  elements in the first random list. Now, measure the time to perform the  $m$  operations using the min leftist tree as well as the min heap. Divide this time by  $m$  to get the average time per operation. Do this for  $n = 100, 500, 1000, 2000, \dots, 5000$ . Let  $m$  be 5000. Tabulate your computing times.
- Based on your experiments, make some statements about the relative merits of the two priority-queue schemes.

## Technical Specification

---

- Perform this experiment, tabulate the result in the homework report.
- Based on your experiments, make some statements about the relative merits of the two priority-queue schemes in the homework report.