DFS LAB – ASSIGNMENT 3

MTech(CS) I year 2018–2019

Deadline: 26 October, 2018

Total: 60 marks

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs18xx-assign3-progy.c`

   IMPORTANT: Insert a single alpha-numeric string of your choice, 6-8 characters long, in the name given above as shown in the examples below. Think of this string as something like a security password, except that you are not required to remember the string. Examples: `cs1840-assign3-x19jdh4-prog1.c`, `ppo03wws-cs1840-assign3-prog2.c`, `cs1840-assign3-prog2-jsiwm7de.c`

2. To submit a file, go to the directory containing the file and run the following command from your account on the server (IP address: 192.168.64.35)

   ```
   cp -p name-of-your-file ~dfslab/2018/assign3/cs18xx/
   ```

   If you want to submit your files from a computer with a different IP address, run

   ```
   scp -p name-of-your-file \
   mtc18xx@www.isical.ac.in:/user1/perm/pdslab/2018/assign3/cs18xx/
   ```

   (enter your password when prompted).

   To submit all `.c` and `.h` files at one go, use
   `cp -p *.c *.h ~dfslab/2018/assign3/cs18xx/` or similar.

NOTE: Unless otherwise specified, all programs should take the required input from stdin, and print the desired output to stdout.

Q1. Consider that you have a dictionary (not necessarily following a lexicographic order) of synonyms and antonyms. They are provided as a sequence of word pairs in a separate input file. Write a program to store these word pairs into some data structures such that given a pair of words one can efficiently determine whether they are synonyms, antonyms or neither. Note that, synonyms are both symmetric and transitive in nature. On the other hand, antonyms are symmetric but antonym of an antonym is effectively a synonym. [20 marks]

Input Format

Input word pairs, which are synonyms or antonyms of each other, will be provided in a separate file (name to be taken from command-line arguments) as follows.

Each line of input will contain a character marker followed by a pair of words where both the words are synonyms or antonyms of each other. The integer markers '+' or '-' denote whether the relation is synonym or antonym, respectively. The query words will be taken as command-line arguments.

Output Format

Output is to be printed on the standard output as SYNONYM, ANTONYM or NEITHER depending on whether the query words are synonyms, antonyms or neither, respectively.

Command-line Arguments Considering the query words as `word1` and `word2`:

```
./prog1 <dictionary_filename> <word1> <word2>
```

Sample Input 0

```
Contents of dictionary.txt:

+ enormous gigantic
+ analysis examination
+ scrutiny scan
+ vast enormous
- cold hot
+ examination scrutiny

Input from command-line:
./prog1 dictionary.txt scan analysis
```

Sample Output 0

```
Output on the terminal:
SYNONYM
```

Sample Input 1

```
Contents of input.txt:

+ sweep brush
- good wicked
+ organize arrange
- virtuous wicked
+ scrutiny scan

Input from command-line:
./prog1 input.txt virtuous good
```

Sample Output 1

```
Output on the terminal:
SYNONYM
```

Sample Input 2

```
Contents of input.txt:

+ pupil student
+ beautiful pretty
- pretty ugly
- silly rational
+ goggle stare

Input from command-line:
./prog1 input.txt beautiful ugly
```

Sample Output 2

```
Output on the terminal:
ANTONYM
```

Q2. Consider a pair of queues Q_0 and Q_1, each containing a list of tasks / jobs / processes.[1] These tasks are to be executed by a machine M that can handle / execute one process at a time. Each job in Q_0 and Q_1 takes no more than 6 time units to be completed by M. A *scheduling algorithm* decides the order in which M is going to execute the given processes. Consider the following scheduling algorithm.

(a) First, pick a queue Q_i ($i = 0$ or 1), and allocate **6 units** of time for executing jobs from that queue. After these 6 units of time are over, Q_{1-i} and Q_i will be allocated time slots alternately.

(b) The duration of a slot given to Q_0 or Q_1 is either **6 units** or **8 units**. If the total number of processes executed so far from Q_i is less than the total number of processes executed so far from Q_{1-i}, then Q_i is allocated 8 units; otherwise, it is given 6 units.

(c) During the time slot given to a queue, processes from that queue are executed *atomically*, i.e., a process is scheduled if it can be completed within the time remaining in the current slot for that queue.

(d) For a given queue, processes are usually selected in first-come-first-served (FCFS) order. However, if the execution time for the next process in FCFS order is more than the time remaining in the current slot for a particular queue (say Q_i), then either of the following can happen.

[1] These terms will be used interchangeably for this problem.

- If the cumulative execution time of this process and its immediate successor in FCFS order is more than 8 units, then out of the two processes, the one with higher execution time is demoted to the end of the queue Q_i. If both have equal execution time, the one with the lower process id is demoted to the end of the queue.
- If not, the process is ignored in the current round.

Write a program to find out how the processes get scheduled for a total of **40 units** when (i) Q_0 and (ii) Q_1 is scheduled first, and determine which of these 2 options is better. The option that results in the maximum number of processes being **completed** within the first 40 units is regarded as the better option; if this number is the same for both options, the one with less idle time[2] is regarded as better. See below for details regarding the output format. [25 marks]

Input Format: The first line of input will contain two integers, n_0 and n_1, specifying the number of processes in the queues Q_0 and Q_1, respectively. This will be followed by $n_0 + n_1$ more lines. Each of these remaining lines contains a process id (a string), and its execution time (an integer).

Output Format: Your program should generate three lines of output. The first line should contain the total number of processes completed with the first 40 units, and the total idle time during this period, for option (i) above. The second line should contain the same information for option (ii). The third line should state `Option 1 is better than Option 2`, or `Option 2 is better than Option 1`, or `Both Option 1 and 2 are equally good`, as the case may be.

Sample Input 0

```
7 7
P1 5
P2 2
P3 3
P4 6
P5 5
P6 3
P7 2
W1 4
W2 3
W3 1
W4 6
W5 5
W6 3
W7 4
```

Note: The processes 1-7 are in Q_0 and 8-14 are in Q_1.

[2]Idle time is the time during which M does not execute any process and remains idle. This happens, for example, when the execution time of the next scheduled process exceeds the remaining time in the slot for the current queue, and the process is therefore not executed.

Sample Output 0

```
9 4
9 7
Option 1 is better than Option 2
```

Explanation for Sample Output 0:

Scheduling for option (i) (first 40 units):

$P1(5) \rightarrow idle(1) \rightarrow W1(4) \rightarrow W2(3) \rightarrow W3(1) \rightarrow P2(2) \rightarrow P3(3) \rightarrow idle(3) \rightarrow W4(6) \rightarrow P5(5) \rightarrow P6(3) \rightarrow stop$

Scheduling for option (ii) (first 40 units):

$W1(4) \rightarrow idle(2) \rightarrow P1(5) \rightarrow P2(2) \rightarrow idle(1) \rightarrow W2(3) \rightarrow W3(1) \rightarrow idle(4) \rightarrow P3(3) \rightarrow P5(5) \rightarrow W5(5) \rightarrow W6(3) \rightarrow stop$

For option 1: Q_0 is scheduled first. Process P1 (time: 5 units) is scheduled for execution in slot 1 ($t = 0$ to 6). P2 has execution time of 2 units; hence it cannot be scheduled in the 1st slot. Now check for process P2 and P3. Both together have cumulative execution time of 5, so no change. Idle time: 1 unit.

For the 2nd slot, since the number of processes previously executed in queue Q_1 (0) is less than that of Q_0 (1), so set time quantum to 8 units ($t = 6$ to 14). W1 (time: 4 units), W2 (time: 3 units) and W3 (time: 1 unit) get executed covering the full 8 units.

For the 3rd slot, time quantum is again set to 8 units ($t = 14$ to 22). P2 (time: 2 units) and P3 (time: 3 units) executes. P4 has execution time of 6 units, hence cannot be scheduled. Check P4 and P5. Cumulative execution time is 11 units. P4 has higher execution time than P5; hence, it gets demoted to the end of the queue. Idle time: 3 units.

Status of queue Q_0 after the third slot:

Processes	Execution time
P5	5
P6	3
P7	2
P4	6

For the 4th slot, since an equal number of processes (3) have been executed from both Q_0 and Q_1, the time quantum is set back to 6 units ($t = 22$ to 28). W4 (time: 6 units) get scheduled.

For the 5th slot, again the time quantum is set to 8 units ($t = 28$ to 36). P5 (time: 5 units) and P6 (time: 3 units) get executed.

For the 6th slot, again the time quantum is set to 8 units ($t = 36$ to 44). W5 (time: 5 units) and W6 (time: 3 units) get executed.

But since W5 completes at time $t = 41$, we cannot consider it to be a program completed within the first 40 units. Thus, we write a stop after P6 (3 units). **But the last 4 units of time cannot be considered as idle time either**, since M was executing W5 during this time.

For option 2: Q_1 is scheduled first. Process W1 (time: 4 units) is scheduled for execution in slot 1 ($t = 0$ to 6). W2 has execution time of 3 units; hence it cannot be scheduled in the 1st slot. Now check for process W2 and W3. Both together have a cumulative execution time of 4, so no change. Idle time: 2 units.

For the 2nd slot, since number of processes previously executed in queue Q_0 is less than that of Q_1, so set time quantum to 8 units ($t = 6$ to 14). P1 (time: 5 units) and P2 (time: 2 units) get executed. P3 cannot be executed in this slot. Check for P3 and P4. Their cumulative execution time is $9(> 8)$ units. Since P4 has higher execution time, it gets demoted to end of queue Q_0. Idle time: 1 unit.

Status of queue Q_0 after the 2nd slot:

Processes	Execution time
P3	3
P5	5
P6	3
P7	2
P4	6

For the 3rd slot, time quantum is again set to 8 units ($t = 14$ to 22). W2 (time: 3 units) and W3 (time: 1 unit) executes. W4 has execution time of 6 units, hence cannot be scheduled. Check W4 and W5. Cumulative execution time is 11 units. W4 has higher execution time than W5, hence gets demoted to end of queue Q_1. Idle time: 4 units.

Status of queue Q_1 after the third slot:

Processes	Execution time
W5	5
W6	3
W7	4
W4	6

For the 4th slot, the time quantum is set to 8 units ($t = 22$ to 30). P3 (time: 3 units) and P5 (time: 5 units) get scheduled.

For the 5th slot, again time quantum is set to 8 units ($t = 30$ to 38). W5 (time: 5 units) and W6 (time: 3 units) get executed.

For the 6th slot, again time quantum is set to 8 units ($t = 38$ to 46). P6 (time: 3 units) is chosen, but its completion overshoots $t = 40$, so we stop.

Sample Input 1

```
7 10
P1 2
P2 6
P3 6
P4 2
P5 3
```

```
P6 4
P7 1
W1 5
W2 1
W3 1
W4 2
W5 4
W6 2
W7 6
W8 2
W9 3
W10 3
```

Sample Output 1

```
12 5
11 3
Option 1 is better than Option 2
```

Explanation:

Scheduling for option (i) (first 40 units):

$P1(2) \rightarrow idle(4) \rightarrow W1(5) \rightarrow W2(1) \rightarrow W3(1) \rightarrow idle(1) \rightarrow P3(6) \rightarrow P4(2) \rightarrow W4(2) \rightarrow W5(4) \rightarrow P5(3) \rightarrow P6(4) \rightarrow P7(1) \rightarrow W6(2) \rightarrow stop$

Total no. of process executed : 12
Total Idle Time : 5 units

Scheduling for option (ii) (first 40 units):

$W1(5) \rightarrow W2(1) \rightarrow P1(2) \rightarrow P2(6) \rightarrow W3(1) \rightarrow W4(2) \rightarrow idle(3) \rightarrow P3(6) \rightarrow P4(2) \rightarrow W5(4) \rightarrow W6(2) \rightarrow P5(3) \rightarrow stop$

Total no. of process executed : 11
Total Idle time : 3 units

Q3. The *min-max* heap is a kind of non-linear data structure that is a combination of min-heap and max-heap, such that both minimum and maximum elements can be looked up in constant times. Moreover, insertion and deletion each take logarithmic times just like a normal heap does in the worst case. The details of a min-max heap may be found at: `http://cglab.ca/~morin/teaching/5408/refs/minmax.pdf`.

Write a program that implements a min-max heap. You may assume that the data elements are all of type `int`, and will be provided via standard input. The program will return the m_i-th maximum or minimum element to the users as asked in the command-line arguments.

Note: Interested readers may refer to the following link for more materials related to *double-ended priority queues*: `https://www.cise.ufl.edu/~sahni/dsaaj/enrich/c13/double.htm`.

Input Format

The data items of the min-max heap will be taken as inputs from the standard input file. The program will also take $m_1, m_2, \ldots, m_k$ as command-line arguments, where $k > 0$ and each m_i is a positive or negative integer. For each m_i, your program should print the m_i-th maximum element if m_i is positive, and should print the $|m_i|$-th minimum element if m_i is negative.

Output Format

The returned data items will be printed (in the respective order) on the standard output.

Sample Input 0

```
Input via standard input:

23 76 29 7 18 44 91

Input from command-line:
./prog3 2 -2 3
```

Sample Output 0

```
Output on the terminal:
76 18 44
```