Spring 2013 Programming Competition

Run by UPE at Rensselaer Polytechnic Institute

March 24, 2013
12:15 - 2:15

Sponsored By

1 Adjacent Number Sets

You are given a sequence of `n > 0` positive integers. Your goal is to organize these numbers into as many sets as possible such that the sum of the numbers in each set is equal across all sets. A set can only consist of numbers that are adjacent to each other. Not all numbers must be part of a set, and every number can only be part of at most one set.

Your input will first be an integer telling you how many numbers to read, and then a new number on each line. Your output will be a single integer representing the maximum number of sets with equal sum that can be formed.

1.1 Example

The sequence "3 4 5 1 1 7" can be split into three groups: "3 4", "5 1 1", and "7", which each total 7.

1.2 Single input

```
<number count>
<number1>
<number2>
```

1.3 Limits

```
<number of test cases> = 200
<number count> ≤ 100
```

1.4 Single output

```
<number of buckets>
```

1.5 Sample input

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

1.6 Expected output

```
3
2
```

2 Secret Destination

You need to meet up with a friend at a secret location. You don't know the location, but he gave you clues as to how to find it. You are given a list of pairs of cities, and each pair has a method of travel (`bus`, `plane`, `taxi`, or `train`), along with a cost. Not every method of travel between every city is possible.

The clues your friend has given you consist of simply a starting city and then a list of various methods of travel. You must figure out which city you will end up in if you begin at the starting city and take those methods of travel in their given order. If there are multiple cities that could be the final destination, you are instructed to choose the one that had the total route with the highest cost.

You will first be given an integer (`n`) describing how many location pairs are available. The next `n` lines will list two cities, a method of travel, and a cost as an integer. Note that you can travel **either direction** using this method of travel (i.e., A to B or B to A). Then you will be given the starting city and an integer on the same line that describes how many lines, each with one method of travel, will proceed.

You can assume there will only ever be one unique length for any given two cities and method of travel (e.g., you can't bus from NYC to Miami two different ways).

2.1 Example

If your possible routes are:

```
Phoenix <-> NYC via plane for cost 400
Phoenix <-> Miami via plane for cost 300
Miami <-> NYC via bus for cost 200
```

And your instructions say: `Phoenix plane bus`

Then your possible secret locations are either:

```
Phoenix -> NYC -> Miami (total cost 600)
Phoenix -> Miami -> NYC (total cost 500)
```

And you would print `Miami` as your solution.

2.2 Single input

```
<number of location pairs>
<city1> <city2> <method> <cost>
<city3> <city4> <method> <cost>
...
<starting city> <number of steps>
<step1>
<step2>
...
```

2.3 Limits

<number of test cases> = 200
<locations> $\leq$ 100
<steps> $\leq$ 40

2.4 Single output

<destination city>

2.5 Sample input

```
2
4
Phoenix NYC plane 400
Phoenix Miami plane 300
Miami NYC bus 200
NYC Redmond train 800
Phoenix 3
plane
bus
train
2
Alburquerque Washington plane 200
Omaha Washington taxi 100
Omaha 5
taxi
plane
plane
taxi
taxi
```

2.6 Expected output

```
Redmond
Washington
```

3 Fitness Program

You decide to try out a new fitness program. Instead of following a different step each day, this program lets you create your routine to fit your personal needs. The program offers `n` different workouts, and you have evaluated each workout for how valuable it would be to your fitness. The catch is, each workout requires you to complete one other specific workout first. The one exception to this is the warm-up, which is where you start your day. Some workouts do not lead into anything else, and that is because that is where you end your day. While you are free to complete a workout more than once, it will never contribute any value towards your fitness after the first time.

You have `M` days to boost your fitness, and you want to select the best workout schedule each day to do so. The first line of input consists of two integers: first the number of workouts to choose from, and second the number of days your program lasts. Each line after consists of three integers: the id of the workout (1 through `n`), the id of the workout that must be done first (or 0 if none), and the fitness value for that workout. You must output a list of the final workout from each day that you plan to do over the entire program, ordered by increasing id.

3.1 Example

```
Workout 1 is your warm-up and has value 10
Workout 2 requires workout 1 and has value 10
Workout 3 requires workout 1 and has value 10
Workout 4 requires workout 2 and has value 30
Workout 5 requires workout 2 and has value 25
Workout 6 requires workout 3 and has value 20
Workout 7 requires workout 3 and has value 15
```

If we had to plan two days, for the first day, we would select workouts 1, 2, and 4. For the second day, we would select 1, 3, and 6. The total fitness from these two days is 50 from the first day and 30 from the second. We would output `4 6`.

3.2 Single input

```
<number of workouts> <number of days>
<workout1> <dependency1> <value>
<workout2> <dependency2> <value>
...
```

3.3 Limits

```
<number of test cases> = 200
<number of workouts> ≤ 1000
<number of days> ≤ 1000
```

3.4 Single output

```
<final workout 1> <final workout 2> ...
```

3.5 Sample input

```
2
7 2
1 0 10
2 1 10
3 1 10
4 2 30
5 2 25
6 3 20
7 3 15
5 2
1 0 10
2 1 10
3 1 50
4 3 5
5 3 15
```

3.6 Expected output

```
4 6
2 5
```

4 A Hairy Situation

The last thing that Gilbert wants to worry about is shaving. He can spend his time doing so many better things, so a little facial hair doesn't bother him. However, he still has obligations throughout the week (meetings, interviews, etc.) and sometimes people are disappointed if he is not clean-shaven. When other people are disappointed, Gilbert is disappointed.

Gilbert wants to plan out when he shaves to maximize his happiness. Every time he shaves, he loses `M` hapiness, where M is a constant. On a given date, if he has an obligation, his happiness decreases by `c*g`, where `g` is the number of full days of hair growth, and `c` is an integer representing how much weight this obligation has. Gilbert always shaves in the morning, so shaving on the day of any obligation would mean `g` has a value of 0. If he did not shave the next day, `g` would increase to 1. Figure out which days Gilbert should shave.

Days are labeled starting at day 1 and increasing through day `n`. The first line of input in a problem will contain three integers. First, the number of days that you need to plan out (`n`). Second, the `M` constant for this problem, and third, how many obligations are coming up in this time period. Then each line after lists a day with an obligation and a `c` value for that obligation. Assume he always shaves on the first day (**DO** include this in your output).

4.1 Example

Assume Gilbert loses 30 happiness for shaving (`M = 30`), and you need to plan out only the next three days of shaving. His obligations are as follows:

```
Day 2: Disapproval of facial hair with a weight of 17
Day 3: Disapproval of facial hair with a weight of 16
```

If Gilbert shaves both days, his happiness decreases by `2*30 = 60`. If he shaves neither day, his happiness decreases by `17*1 + 16*2 = 49`. If he only shaves on day 2, his happiness decreases by `30 + 16*1 = 46`. And finally, if he only shaves on day 3, his happiness decreases by `17*1 + 30 = 47`. The best choice here is to shave on day 2 (in addition to the implicit shave on day 1), so your output would be `1 2`.

4.2 Single input

```
<number of days> <cost of shaving> <number of obligations>
<obligation1> <weight1>
<obligation2> <weight2>
...
```

4.3 Limits

<number of test cases> = 200
<number of days> $\leq$ 100
<number of obligations> $\leq$ 100

4.4 Single output

<day shaved> <day shaved> ...

4.5 Sample input

```
2
3 30 2
2 17
3 16
3 50 3
1 10
2 10
3 10
```

4.6 Expected output

```
1 2
1
```