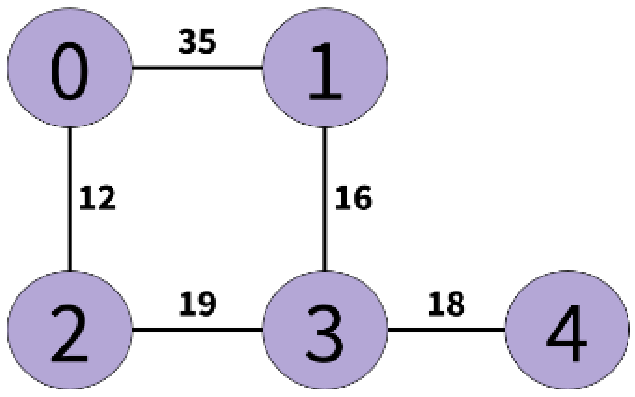
Can Cat Catch the Final?

It's finals week(s) at METU, and Cat is trying to get to her final on time. When Cat is on her way to the building, you call her to remind her that the exam is open book & notes. Turns out she forgot to print her notes and wants to get to the printer to print the slides. Also, Cat realizes that she left her ID in her dorm room, so she also wants to go to her dorm before the exam starts.

You know that Cat needs to follow the shortest route from her location to the exam building, but she also wants to stop by the printer and her dorm. You know how long it takes her to walk between buildings, and how much time she has left until the exam. So you offer your help to Cat to see if she can make it in time by writing a little program. Your program should decide the best thing for Cat to do to be on time for the exam. If she cannot be on time even without stopping by the dorm or the printer, your program should report that it is impossible.

There are **N** buildings in METU. Each building is numbered from **0** to **N-1**. Each building is connected to another building by a road and there are **M** roads on the campus. Each road can be walked in both directions. Moreover, walking each road takes Cat **T** minutes. Cat is currently in building **S**, the exam is in building **D**, the printer is in building **X**, and Cat lives in dorm building **Y**. There are **L** minutes left until the exam starts. So, you need to find the quickest route from **S** that ends in **D** which also visits **X** and **Y.** Additionally, Cat can visit a building (including S, D, X, and Y) *multiple times*.

Example graph of the campus:



L=65 (minutes left for the exam to start)
N=5 (number of buildings)
S=0, **D**=4 (Cat is in building 0, exam is in building 4)
X=1, **Y**=2 (printer is in building 1, dorm is building 2)
M=5 (number of roads)
Roads=[{0, 1, 35}, {0, 2, 12}, {1, 3, 16}, {2, 3, 19}, {3, 4, 18}]
^
T (time it takes Cat to walk that road)
PrintPath=0 (0: do not print the path / 1: print the path)

You need to write a function called *CanCatch*, which takes the number of buildings (***N***), information about the ***M*** Roads, the building that you are in (***S***), exam building (***D***), the building the printer is located (***X***), your dorm building (***Y***), and time left until the exam starts ***(L)***. The function should print whether or not you can catch the exam before it starts, and if you can catch the exam, the function should report the path you need to follow to catch it (you need to print the full path depending on the **PrintPath** parameter). This path may or may not include X and/or Y according to the time Cat has left for the exam:

- **If it is possible to visit both X and Y before the exam starts, you need to choose the shortest path.** The path should start from S, should go through both X and Y, and end in D. If order of visits to X and Y does not change the time cost*, she would rather stop by the printer (X) first, then stop by her dorm (Y).*
- **If it is impossible to visit both X and Y before the exam starts, you need to find the shortest path that goes through only one of them.** The path should start from S, should go through only one of X or Y, and end in D. You need to choose the path with the shortest time. *If Cat can stop by her dorm or the printer at the same time cost, she would rather stop by the printer (X).*
- **If passing through even one of X or Y is not possible timewise, you need to find the shortest path from S to D without visiting X or Y.**
- **If Cat cannot even make it to the exam without stopping by X or Y, you should report that it is impossible to catch the exam.**

Road information is given to you as a vector of **Road** structs. A **Road** struct contains two *buildings* on each end of the road and the *time* it takes to travel that road.

```
struct Road {
  std::pair<int, int> buildings;
  int time;
  Road(std::pair<int, int> bs, int t) : buildings(bs), time(t) {}
};
```

CanCatch's function declaration is:

```
void CanCatch(int n, std::vector<Road> roads, int s, int d, int x, int y, int l, int
printPath);
```

Inputs:
----

n = number of buildings **(N)**
roads = information about roads supplied with Road structs.
s = the building Cat is in **(S)**
d = the building of the exam **(D)**
x = the building the printer is located **(X)**
y = Cat's dorm building **(Y)**
l = time left until the exam starts **(L)**
printPath = whether to print the full path (1) or not (0)

Outputs:

You need to print whether Cat can make it in time to the exam or not, and if she can, whether she

can visit X and Y, the time it will take her to catch the exam as fast as possible, and the path she should take (if printPath == 1).

Possible output formats are:

```
YES BOTH <time of path>MINUTES
<path>

YES PRINTER <time of path>MINUTES
<path>

YES DORM <time of path>MINUTES
<path>

YES DIRECTLY <time of path>MINUTES
<path>
```

```
IMPOSSIBLE
```

The path should be printed in order by separating the building numbers in the path with '->'. An example path that goes through nodes 1, 2, 3, and 4 should be printed as follows:

```
1->2->3->4
```

Note that the last element in the path does not have '->' or a space after it and the output ends with a newline char ('\n').

*Hint: You can use the supplied **PrintRange** function to easily print elements of containers with iterators.*

Example IO:

Input: l=85, n=5, s=0, d=4, x=1, y=2, m=5, printPath=1
roads=[{0, 1, 35}, {0, 2, 12}, {1, 3, 16}, {2, 3, 19}, {3, 4, 18}]
 	
Output:`YES BOTH 81MINUTES`
`0->2->3->1->3->4`

Input: l=15, n=5, s=0, d=4, x=1, y=2, m=5, printPath = 1 roads=[{0, 1, 35}, {0, 2, 12}, {1, 3, 16}, {2, 3, 19}, {3, 4, 18}] Output:`IMPOSSIBLE`
Input: l=49, n=5, s=0, d=4, x=1, y=2, m=5, printPath = 0 roads=[{0, 1, 35}, {0, 2, 12}, {1, 3, 16}, {2, 3, 19}, {3, 4, 18}] Output*:*`YES DORM 49MINUTES`

Note: Road struct is represented as '{buildings.first, buildings.second, time}'