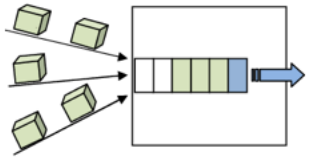
Simulation of Processing Network Packets



Everything you do on the Internet involves packets. For example, every Web page that you receive comes as a series of packets, and every e-mail you send leaves as a series of packets. Networks ship data around in small packets.

In this assignment, you'll simulate the processing network packets arrives at a computer with one processor. Packets arrive in some order. For each packet number i, assume that you know the time when it arrived $\mathbf{T_i}$ and the time it takes the processor to process it $\mathbf{P_i}$. The processer processes the incoming packets in the order of their arrival. If the processor started to process some packet, it doesn't stop until it finishes the processing of this packet, and the processing of packet **i** takes exactly $\mathbf{P_i}$ milliseconds.

Assume that the computer has a buffer with size **S (in packets)** to store the network packets, and each packet has the same size. If the buffer is full when a packet arrives, it will be dropped and won't be processed at all. If several packets arrive at the same time, they are first all stored in the buffer. Some of them might be dropped because of the limited buffer size. Please note that even if multiple packets could arrive at the same time, they still must come through a single pipeline in a certain order. In this assignment, we assume the order is already known and specified in the input data.

The computer processes the next available packet from the buffer as soon as it finishes processing the previous one. If there are no packets on the buffer, the processor stays idle.

You are asked to write a C++ program to simulate this process. You'll read the packets as inputs to your program. Here is the input format:

- The first line of the input contains the size **S** of the buffer and the number **n** of incoming network packets.
- Each of the next n lines contains two numbers. i^{th} line contains the time of arrival $\mathbf{T_i}$ and the processing time $\mathbf{P_i}$ of the i^{th} packet.
- Assume those n lines are already sorted in terms of the time of arrival $\mathbf{T_i}$ so the computer will process those packets in the order specified by the input, even if two packets arrive at the same time.
- All the numbers in the input are non-negative integers.

Sample Input:

```
1 2
0 1
0 1
```

As seen in the sample input given above, the network buffer is given as 1, and the number of packets is 2 (S = 1 , n=2). The next two lines gives the time of arrival and processing time for each of two packets. The first packet arrived at time 0, and the second packet also arrived at time 0. Each takes 1 millisecond to be processed.

After reading the input from the user, your program will generate an output to summarize the moment of time when the processor began processing the packet, or -1 if the packet was dropped.

Sample Input:

```
1 2
0 1
0 1
```

Sample Output:

```
0
-1
```

As seen in the output, the first packet started processing at time 0, and the second one is dropped, because the buffer has size 1 and it was full after it took the first packet. So, the second packet wasn't processed at all.

More Sample Runs:

Sample Run-1: There are no packets, the processor is idle

Input:

```
1 0
```

Output:

```
-2
```

Sample Run-2: The only packet arrived at time 0, and computer started processing it immediately.

Input:

```
1 1
0 1
```

Output:

```
0
```

Sample Run-3: The first packet arrived at time 0, the computer started processing it immediately and finished at time 1. The second packet arrived at time 1, and the computer started processing it immediately.

Input:

```
1 2
0 1
1 1
```

Output:
0
1

Sample Run-4: The first packet arrived at time 0, the second and third packets arrived at time 1. The computer started processing the first packet at 0, second packet at time 1, and third one at time 3 after spending 2 milliseconds for the second packet.

Input:
1 3
0 1
1 2
1 1
Output:
0
1
3

You are provided with a started code. The class is partially implemented, and your task is to implement the rest of it. You are asked to implement and use a **Queue** data structure in your solution. Make sure you implement **Queue** with dynamic arrays (not vectors), and make sure to design **Queue** class as a template class.

Starter Code:

```
#include <iostream>
#include "Packet.h"
#include "Response.h"
#include "Buffer.h"

using namespace std;
//…
int main() {
    int bufferSize;
    cin >> bufferSize;
    vector<Packet> requests = readPackets(); //read packets from user

    //create buffer with the given size
    Buffer buffer(size);

    //process the packets
    vector<Response> responses = processPackets(requests, &buffer);

    //print responses
    printResponses(responses);

    return 0;
}
```

//…

HOW TO EVALUATE: The following rubric describes how your work will be evaluated.

Correctness (90 points)

- [90] Program is correct in object-oriented design and function; meets specification
- [75] Program output is correct but elements of specification missing, e.g. variable/method declarations.
- [45] Part of the specification has been implemented, e.g. one out of two required
- subprograms.
- [20] Program has elements of correct code but does not assemble/compile.

Readability (10 points)

- [10] Programmer name and assignment present. Sufficient comments to illustrate program logic. Well-chosen identifiers.
- [7] Programmer name present, most sections have comments. Fair choice of identifiers
- [5] Few comments, non-meaningful identifiers
- [0] No programmer name. No comments. Poor identifiers