Part Purchasing

Objective

Give additional practice with dynamic memory in C.
Give practice with Queues in C.

Story

The ship arrived back at the harbor (mostly) in one piece. You have already walked through the ship and have noted the problems with the ship. You don't have the time (or parts) to repair the ship, so you will rely on a third party group to make the ship seaworthy again.

You fear that the repair crew will overcharge you. For this reason you want to write a program that will determine an estimate as to how much money the repair will cost. You are certain that they won't look for problems. The repair crew maintains a list of broken components that need to be fixed. The list begins empty, because they don't look for the problems directly. The repair crew will try to turn the ship on, and while some component is still broken, the repair crew will look for the most obvious component that is broken and add that component to the end of the list of broken components to fix.

Sometimes when a component is fixed it will reveal additional components that need to be fixed. Those revealed components will be added to the end of the list of components to fix assuming that they are still broken in the order they are revealed.

To fix a component a part will be required. If the part is available the repair crew will expend the part and the component will be fixed. If the required part is not available, then the repair crew will place an order for the part from some distribution center, but to charge you extra they will order an additional part from the distribution center claiming that it can save you money in the long run. For each part type you know the additional part type that will be ordered and the cost of order said parts. Since it takes a little bit of time to get the part once it has been ordered, the crew will place the component at the end of the list of components to fix. The good news is that by the time the next component in the list of components to fix is examined the 2 parts ordered will have arrived.

You want to know the sum of all the costs of all the parts ordered based on the repair crews repair procedure.

Problem

Given a list of parts pairs and their cost, the components that are broken in order of their obviousness, and the components that are revealed when a component is fixed. Determine the amount of money you will have to pay to repair the ship.

Input

Input will begin with a line containing 1 integer, ***P*** (1 ≤ ***P*** ≤ 100,000), representing the number of parts that can be used to repair the ship. The following ***P*** lines will each contain 2 integers, ***c*** and ***e***, (1 ≤ ***c*** ≤ 100,000; 1 ≤ ***e*** ≤ ***P***) representing the cost of ordering the part corresponding and the extra part that will be received at "no extra" cost.

The next line of input will contain a single integer, ***C*** (1 ≤ ***C*** ≤ 500,000), representing the number of components that need to be repaired. The most obvious broken component will always be the one with the lowest index. Following this will be C lines each beginning with 2 integers, ***p*** and ***n*** (1 ≤ ***p*** ≤ ***P***; 0 ≤ ***n*** ≤ ***C*** - 1), representing the part needed to repair the corresponding component and the number of broken components revealed upon repairing the component. On this same line will follow ***n*** integers all in the range of [1, ***C***] representing the components that are revealed when the corresponding component is repaired.

It guaranteed that the sum of all ***n***'s is less than 5,000,000

Output

Output should contain a single line with a single integer representing the price the repair crew will charge in costs for ordering all the parts.

Sample Input	**Sample Output**
5 10 3 5 2 25 2 30 5 25 1 4 4 2 4 3 2 1 1 1 0 3 0	65
3 5 2 10 3 15 3 6 1 3 2 6 3 2 1 2 3 1 5 1 1 2 2 0 3 5 5 4 3 2 1	35

Explanation

Case 1

There are 5 different parts.

- Purchasing the first part comes with an extra third part; the cost is 10 units(?).
- Purchasing the second part comes with an extra second part; the cost is 5 units(?).
- Purchasing the third part comes with an extra second part; the cost is 25 units(?).
- Purchasing the fourth part comes with an extra fifth part; the cost is 30 units(?).
- Purchasing the fifth part comes with an extra first part; the cost is 25 units(?).

There are 4 broken components

- The first component uses part 4 to be repaired. If the first component is repaired, then it reveals components 4 and 3 as broken.
- The second component uses part 2 to be repaired. If the second component is repaired, then it reveals component 1 as broken.
- The third component uses part 1 to be repaired and reveals no additional components.
- The fourth component uses part3 to be repaired and reveals no additional components.

The repair process starts with an empty list, there are no spare parts, and all components are broken (see image for visual representation).

Parts	1	2	3	4	5
Count	0	0	0	0	0

Component	1	2	3	4
Status	broken	broken	broken	broken

Component List	empty

The crew tries to start the ship and component 1's status becomes known

Parts	1	2	3	4	5
Count	0	0	0	0	0

Component	1	2	3	4
Status	known	broken	broken	broken

Component List	1

Component 1 needs part 4, but the count is 0. The repair crew purchases the part **(for 30 units)** and gets a part 5 for free. Then component 1 goes to the end of the component list.

Parts	1	2	3	4	5
Count	0	0	0	1	1

Component	1	2	3	4
Status	known	broken	broken	broken

Component List	1

Component 1 is then tried to fix again. This time the repair is successful, but when component 1 becomes functional part 4 and 3 become known. Component 4 is the front of the list and needs part 3, of which there are none.

Parts	1	2	3	4	5
Count	0	0	0	0	1

Component	1	2	3	4
Status	fixed	broken	known	known

Component List	4	3

Part 3 is ordered **(for 25 units)** which comes with an additional part 2, and component 4 is moved to the back of the list. Component 3 is now at the front of the list and needs a part 1 to be repaired. There are no part 1's.

Parts	1	2	3	4	5
Count	0	1	1	0	1

Component	1	2	3	4
Status	fixed	broken	known	known

Component List	3	4

A part 1 is ordered **(for 10 units)** which comes with an additional part 3. Component 4 is then attempted to be fixed again.

Parts	1	2	3	4	5
Count	1	1	2	0	1

Component	1	2	3	4
Status	fixed	broken	known	known

Component List	4	3

This time the part is availbed and the component is fixed. Repairing the component reveals no additional broken components. Component 3 is now the only component in the list

Parts	1	2	3	4	5
Count	1	1	1	0	1

Component	1	2	3	4
Status	fixed	broken	known	fixed

Component List	3

Component 3 repair is attempted. It requires a part 1.

Parts	1	2	3	4	5
Count	1	1	1	0	1

Component	1	2	3	4
Status	fixed	broken	known	fixed

Component List	3

The repair is completed successfully using part 1. No additional component is revealed.

Parts	1	2	3	4	5
Count	0	1	1	0	1

Component	1	2	3	4
Status	fixed	broken	fixed	fixed

Component List

The repair crew attempts to start the ship again. This time component 2 is revealed.

Parts	1	2	3	4	5
Count	0	1	1	0	1

Component	1	2	3	4
Status	fixed	known	fixed	fixed

Component List	2

Component 2 needs are part 2.

Parts	1	2	3	4	5
Count	0	1	1	0	1

Component	1	2	3	4
Status	fixed	known	fixed	fixed

Component List	2

The part is available, so the repair is accomplished with no further part purchases.

Parts	1	2	3	4	5
Count	0	0	1	0	1

Component	1	2	3	4
Status	fixed	fixed	fixed	fixed

Component List

The total of all the part purchases is therefore 30 + 25 + 10 for a total of **65 units**.

Case 2
The repair crew tries to start the engine, but component 1 is broken.

Component 1 is at the front of the list and needs part 1, but there are none.

Part 1 is purchased for **5 units**. The part comes with an extra part 2.

Component 1 goes back into the list.

Component 1 is at the front of the list again and needs part 1. This time there is a part 1 that can be used. The only part left is part 2.

Part 1 is fixed. The components revealed and added to the list are 2, 6, and 3.

Component 2 is at the front of the list and requires part 2. The part is available. The component is fixed and no parts remain.

Fixing component 2 reveals component 2 (which is weird), but since the part is fixed, nothing is added to the list.

Component 6 is at the front of the list. It needs part 3, but there is no part 3 available.

Part 3 is purchased for **15 units**. The part comes with an additional part 3. Component 6 is moved to the back of the list (behind 3).

Component 3 is now at the front of the list. It needs a part 3. It becomes fixed, and a single part 3 is left. Fixing the component reveals component 5, which is added behind component 6.

Component 6 is now at the front. It needs a part 3. It becomes fixed, and no parts are remaining. Fixing component 6 reveals component 5 (but it's not added since 5 is already in the list), component 4 (which is added to the list behind 5), component 3 (which is not added), component 2 (not added), and component 1 (not added).

Component 5 is at the front and requires part 2, which is not available. The part is purchased for **10 units**, and an extra part 3 is added. Component 5 is moved to the end of the list (behind component 4).

Component 4 is at the front and requires a part 1, which is not available. The part is purchased for **5 units**, and an extra part 2 is added. Component 4 is moved to the end of the list (behind component 5).

Component 5 is at the front and requires part 2, which is available. The component is fixed, and reveals nothing.

Component 4 is at the front and requires part 1, which is available. The component is fixed and reveals component 2, which is not added to the list because it is already fixed.

The total amount of money spent is 5 + 15 + 10 + 5 for a **total of 35**.

Hints

Many Arrays/Struct Arrays: My solution used several parallel arrays for each piece of information needed for the parts and components. You could also have a struct for Parts and a struct for Components and use an array of them.

Component List: The list of components to repair should be treated as a Queue. The Queue could hold merely the index of the component opposed to an entire component struct.

Component Status: Store the status of each component revealed or unrevealed in an array, so you can O(1) determine whether a newly revealed component after fixing a component should be added to the queue or not.

Dynamic Arrays: You should store the list of components revealed by fixing a component in a dynamic array.

Struct Suggestions: The structs you could use for the program could look like the following

```
struct Part {
   int count; // The number of this part available
   int extra_part; // The additional part that comes when purchasing
   int price; // The price of purchasing the part
};

struct Component {
   int known; // Is the component known to the repair crew
   int part_req; // The part required to fix
   int num_comp_rev; // Number of components revealed when fixed
   int * comp_rev; // Array of revealed components when fixed
};
```

Grading Criteria

- Good comments, whitespace, and variable names
    - 15 points
- Use standard Input/output
    - 5 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
    - 5 points
- Begin the “repair process” only after reading in all the information
    - 5 points
- Track the number of each available part
    - 5 points
- Repair Process
    - Loop over all the components in order and when one is found to be broken add it to the repair “list” and begin trying to repair it
        - 5 points
    - When “repairing” a component loop until the repair “list” is empty
        - 5 points
    - Handle for each component what happens when the needed part is and is not available
        - 5 points

No points will be awarded to programs that do not compile using “gcc -std=gnu11 -lm”.

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use a Queue. ***Without this, your program will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.