Question 2

a) What are the possible solutions for the problem?

1. Using adjacency matrix to store the graph, then using Depth First Search to find out all possible solution from start to end for each day. For each solution, we find out the minimum weight, and choose the biggest weight then output.
2. Using adjacency list to store the graph. Since we want to find the maxmin path, we use a priority queue to determined which node to search next. From start, we find all its neighbors' and add then to the priority queue. We will choose the path that has largest weight, since we need maxmin. Then find all its neighbors' that are not visited yet and add then to the queue, update the maxmin value. Using priority queue, we always follow the path that has maxmin weight, so once we reach the end point, we get the maxmin result.

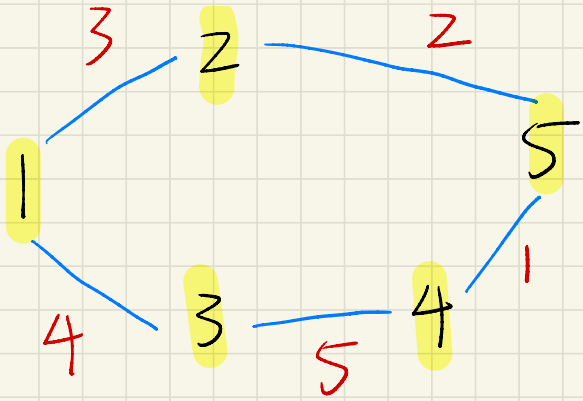
b) How do you solve this problem?

Adjacency list: we use `vector<vector<pair<int, int>>>` to store the graph. For node i, it all its neighbors are in `list[i-1]` The pair's first value denotes which node it connected to, and second value denotes the weight between them. At each day we simply change the value in the adjacency list.

Max min path: we initialized a `priority_queue<pair<int, int>>` and a `vector<bool>` denote whether the node has been visited or not. We first add start node to the queue, weight INT32_MAX. While the queue is not empty, we dequeue the node, since it is a priority queue, we always find the node that has largest min weight. We first mark it as visited, and find all its neighbors, if its neighbors are not visited yet, we enqueue then with min(current min , weight) and repeat the progress until we find the end point and return the current min, which is the max min path we find. See the draft in page two.

c) Why is your solution better than others?

My solution use adjacency list to store the graph, which saves memory. If we are using the adjacency matrix, when the graph is very large, it will lead to runtime error. Also, if we use the DFS to search all path that are possible from start to end will lead to time exceeded, since its not efficient. My solution always follows the path that has max min weight, once we reach the end point we get our result, no need to try all path. Once the path has smaller max weight the queue will move it behind, let us follow the max min path, which saves a lot of time.



start : 1

end : 5

max min = 2

(1→2→5 / 1→3→4→5)

initialized : queue{(Int32max, 1)}
current min, node

① Pop (Int32max, 1)

current min = Int32max

neighbor : 2 (w=3) bigger, 3 (w=4) bigger

queue{(4, 3), (3, 2)}
node, current min

② Pop (4, 3)

current min = 4, neighbor : 4 (w=5) smaller

queue{(4, 4), (3, 2)}

③ Pop (4, 4)

current min = 4, neighbor : 5 (w=1) bigger

queue{(3, 2), (1, 5)}

④ Pop (3, 2)

current min = 3, neighbor : 5 (w=2) smaller

queue{(2, 5), (1, 5)}

⑤ Pop (2, 5)

5 is end Point

So max min = current min = 2.

Done!

Question 3

a) What are the possible solutions for the problem?

1. Using adjacency matrix to store the graph, then using Depth First Search to find out all possible solution from start to end. For each solution, we first reverse the path we get, that is from end to start we calculate the health point from 0, and the spirit point is also start from 0, calculate the health point step by step and choose the minimum health point then output.
2. We calculate from the end, that is from end to start, our spirit point grows while we head to the start point; However, we don't care about the start point, we actually need a place for us to gain enough spirit point, so the hero can go anywhere he wants without deducting health points. So our goal becomes find a edge that the health point grows slow and stop to grow with minimum health point, then we can pass through that edge any times to gain enough spirits point, so then we will not loss any health point later. Start from end point, we find the path that has minimum weight, this path is a must to pass through, since it directly determined the minimum weight you pass through at spirit point 1, the minimum health point will be the weight of that edge. Then we add our spirit point by 1, and find the node of that edges all neighbor, to find the next minimum cost we need until our health point no longer increase, then we get the minimum health point.

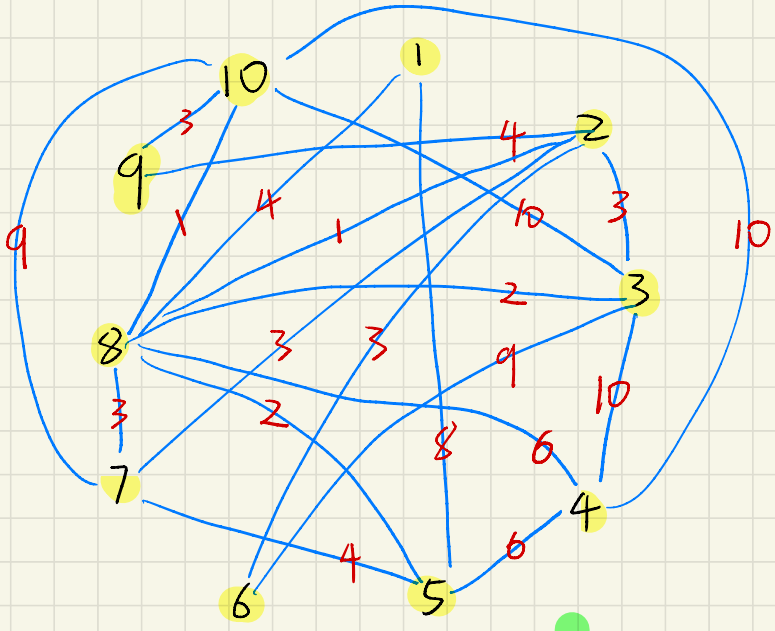
b) How do you solve this problem?

Adjacency list: we use `vector<vector<pair<int, int>>>` to store the graph. For node i, it all its neighbors are in `list[i-1]` The pair's first value denotes which node it connected to, and second value denotes the monster's attack power between them.

Find min hp: initialized spirit point and health point. We find all neighbors of the start point, and pick the smallest path to go, since it is a must to pass through, and update the start point at same time. Then we use a while statement, in the progress, we first update the health point with current path hp cost in last iteration we get, for the first iteration, it is the cost we get when initialized. Then add 1 to the spirit point, and find all neighbors of start point, pick the edge that has the minimum health point cost. We repeat this progress, until the health point is no longer increasing, that is our current path hp cost becomes zero, indicate that we can gain any spirit point from now on at this edge without increasing health point. So we return the current health point that is the result we get.

c) Why is your solution better than others?

My solution is efficient. There is no need to find out all possible solution from start to end using Depth First Search, which is time consuming, and may not get the correct answer since we can gain spirit point as much as we want to avoid losing health point.



Start Point: 5

end Point: 2

min HP: 1.

① neighbors of 2: 9 . 8 . 7 . 6 . 3

attack Power: 4 . 1 . 3 . 3 3

HP=0

SP=1

$HP = 0 + \lfloor \frac{1}{SP} \rfloor = 1.$

② neighbors of 8: 10 . 1 . 2 . 3 . 4 . 5 . 7

attack Power: 1 4 1 2 6 2 3

HP=1

SP=2

$HP = 1 + \lfloor \frac{1}{SP} \rfloor = 1$

↳ equals to 0, break.

at this edge we can gain as much sp as we want.

So min HP = 1.