```
"""
Algorithms in computational Biology (INFO-F438)
Assignment 2 : Similarity-based genes prediction with split genes. Given several intervals of
substrings with a weight proportional to their similarity to the searched gene, find the
collection of substrings disjoints
and with the maximum weight.
"""

__author__ = "Charlotte Nachtegael"
__date__ = "22 mars 2016"


def chaining_exon(set_of_intervals):
    """
    :param set_of_intervals: list of weighted intervals in the form of tuple with the start
     point,the end point and the weight of the interval according to a similarity-based
     scoring.
    :return: The maximum score you can obtain by combining disjoint intervals of the set
    """
    # compute the points of interest
    end_points =[]
    all_points = []
    for interval in set_of_intervals:
        end_points.append(interval[1])
        if interval[0] not in all_points:
            all_points.append(interval[0])
        if interval[1] not in all_points:
            all_points.append(interval[1])
    all_points.sort()
    end_points.sort()

    # we calculate the score at each start/end points
    score = [0]*len(all_points)

    for i in range(len(all_points)):
        # if an interval ends there, calculate the best score for this point
        if all_points[i] in end_points:
            possible_next_score = [score[i-1]]
            intervals_to_study = find_interval(set_of_intervals, all_points[i])

            # look at all the intervals that end at this point
            for interval in intervals_to_study:
                index_start_interval = find_index_point(all_points, interval[0])
                score_of_interval = score[index_start_interval] + interval[2]
                possible_next_score.append(score_of_interval)
                set_of_intervals.remove(interval)

            score[i] = max(possible_next_score)

        else:
            score[i] = score[i-1]

    return score[-1]


def find_interval(set_of_intervals, end_point):
    """
    :param set_of_intervals: list of weighted intervals whose end was not yet reached
    in the form of tuple with the start point, the end point and the weight of the interval
    according to a similarity-based scoring.
```

```
    :param end_point: integer representing the end point of an interval
    :return: list of the intervals finishing at the end point given
    """
    intervals_with_end_point = []
    i = 0

    # search only until the end point, not above
    while i < len(set_of_intervals) and set_of_intervals[i][1] <= end_point:
        if set_of_intervals[i][1] == end_point:
            intervals_with_end_point.append(set_of_intervals[i])
        i +=1

    return intervals_with_end_point


def find_index_point(all_points, point):
    """
    :param all_points: list with all the start and end point of the intervals
    :param point: integer representing the point for which we want to know the position
    in the list of all points
    :return: the position of the point in the list
    """
    count = 0
    index = 0
    found = False
    while count < len(all_points) and not found:
        if all_points[count] == point:
            index = count
            found = True
        count += 1
    return index


if __name__ == '__main__':
    set_of_intervals = [] # insert your set of intervals here
    score = chaining_exon(set_of_intervals)
    print("Your score for this set of interval is", score)

# set_of_intervals =
[(1,5,4),(3,9,3),(4,10,3),(7,13,6),(10,17,10),(11,16,2),(14,19,1),(17,19,8)]
# score = 22
# set_of_intervals =
[(1,5,5),(2,3,3),(4,8,6),(6,12,10),(7,17,12),(9,10,1),(11,15,7),(13,14,0),(16,18,4)]
# score = 21
```