Dynamic Programming Group Project

Alperen Ugus, Mackenzie Hull, Joseph Lovato

November 4th, 2019

1. Algorithms

1.1. Recursive Definition

Define Max_Events(p, t) to be the maximum number of events viewed in the set of events E, where p is the current position of the telescope and t is the current time. Assumed indexing from 1.

$$
\begin{aligned}
\mathit{Max\_Events}(p, t) = \max\{ & \\
& \mathit{max\_events}(p + 1, t - 1), \\
& \mathit{max\_events}(p, t - 1), \\
& \mathit{max\_events}(p - 1, t - 1) \\
& \} + \mathit{viewable}
\end{aligned}
$$

Where viewable is defined as:

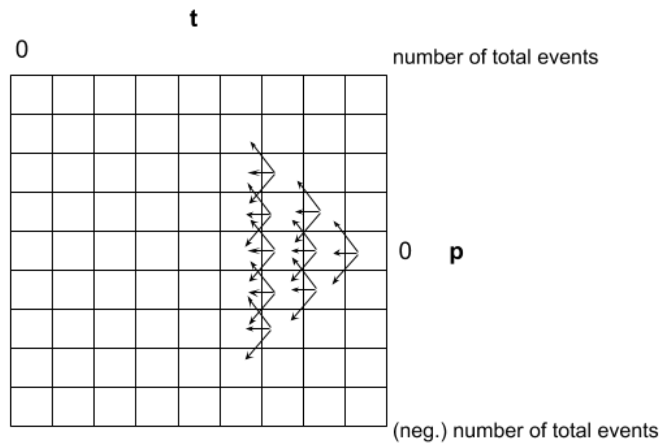
$$
\begin{aligned}
&\text{If } p = E[t] \text{ then } \mathit{viewable} = 1 \\
&\text{Else, } \mathit{viewable} = 0
\end{aligned}
$$

With basis cases of:

$$
\begin{aligned}
&\mathit{Max\_Events}(p \text{ where } |p| > t\,, t) = -\infty \\
&\mathit{Max\_Events}(0, 0) = 0 \\
&\mathit{Max\_Events}(p \mathrel{!=} 0, 0) = -\infty
\end{aligned}
$$

Explanation: Our function takes in position and time as parameters. It returns the maximum number of events that can be visited in the given time by moving the telescope position +1, 0, and -1, which is added to viewable. Viewable is a variable that stores 1 or 0 depending on if the telescope is viewing an event. Our first base case describes when the current event is farther away from 0 than there is time left. The second base case describes when we have reached the beginning and are at the valid starting position. The third base case describes when we are at time 0, but not in position 0, which is invalid.

1.2. Dynamic Programming Algorithm



Our DP table is two-dimensional, with one axis representing time (t) with a range from 0 to the total number of events, and the other axis representing position (p) with a range from the negative total number of events to the positive total number of events.

Each coordinate in the table is an object which can be described by:

Coordinate object (entry in table) contains:

- *Total Events*
- *Movement (-1, 0 or +1)*

Pseudo-code:

```
With a global list of events E

DP_max_events(p, t):
    Initialize table
    DP_recursive(p, t, table)
    trace_back(p, t, table)

DP_recursive(p, t, table):
    If coordinate (p, t) is already in table, return coordinate

    Basis Cases:
    If coordinate p > t, store coordinate in table as invalid entry
    If both p and t equal zero, store coordinate in table as valid entry
```

```
        If t = 0 but p != 0, store coordinate in table as invalid entry

        Set variable "viewable" to (1 or 0) depending on if telescope is viewing event
        at current time

        Find maximum total events of three adjacent coordinates with the recursive
        calls:
                Max (
                DP_recursive(p+1, t-1),
                DP_recursive(p, t-1),
                DP_recursive(p-1, t-1)
                )

        Add maximum result plus "viewable" to table at corresponding current
        position and time entries.
```

1.3. Traceback Algorithm

```
trace_back(p, t, table):
    Initialize the arrays that will store movements and events
    Grab the first coordinate (starting from the end) and append movement
    Using the movement value of the current coordinate, step backwards on the
    dp table until the beginning, while storing the current event (if viewable)
    and movements
    Reverse the movement and events arrays since our algorithm starts from the
    last event and returns them
```

2. Complexity

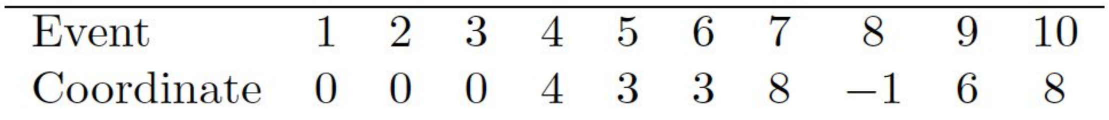
Because our DP algorithm must fill a fraction of the table defined by $n \; x \; 2n + 1$, the time complexity is O(n^2).

3. Implementation

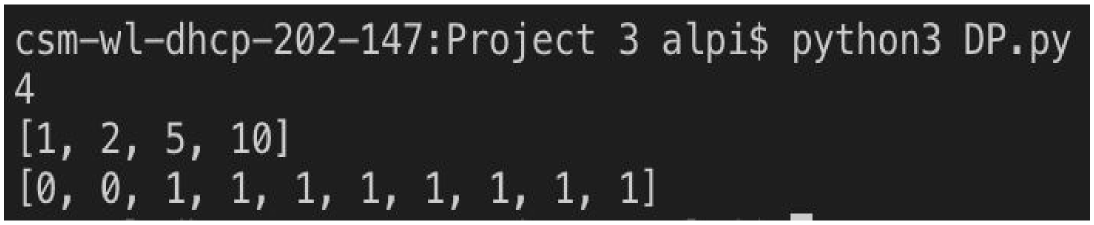
3.1. Code

```python
E = [0, 0, 0, 4, 3, 3, 8, -1, 6, 8]

def main():
    DP_max_events(E[len(E) - 1], len(E))

class Coordinate:
    def __init__(self, total_events, movement):
        self.total_events = total_events
        self.movement = movement

def DP_max_events(p, t):
    # Generate the DP table (t x 2*total-number-of-events)
    table = [[Coordinate(-100e10, -100e10) for x in range(t)] for y in
range(len(E)*2 + 1)]
    # Call recursive function
    num_events = DP_recursive(p, t, table)
    # Call traceback function
    events, movements = trace_back(p, t, table)
    # Print results
    print(num_events)
    print(events)
    print(movements)


def DP_recursive(p, t, table):
    p_index = p + (len(table) // 2)  # Scale position for table indexing
    t_index = t - 1  # Scale time for table indexing

    # If our entry is already in the table, just return it
    if table[p_index][t_index] is Coordinate(-100e10, -100e10):
        return table[p_index][t_index].total_events

    # BASE CASES:
    # If our position is farther away than we have time left --> invalid
    if abs(p) > t:
        table[p_index][t_index] = Coordinate(-100e10, 3)  # Add coordinate
to table
        return -100e10
    if t == 0:
        # If we're at position 0 and time 0 --> valid
        if p == 0:
            table[p_index][t_index] = Coordinate(0, 2)  # Add coordinate to
table
            return 0
        # If we're at time 0, but not at position 0 --> invalid
        else:
            table[p_index][t_index] = Coordinate(-100e10, 2)  # Add
coordinate to table
            return -100e10

    # If telescope is viewing event at current time, then viewable is 1
    viewable = 0
    if p == E[t_index]:
        viewable = 1

    # Define three possible moves, recursively
    increase = DP_recursive(p + 1, t - 1, table)
    stay = DP_recursive(p, t - 1, table)
    decrease = DP_recursive(p - 1, t - 1, table)

    # Find maximum of those three moves
    maximum = max(increase, stay, decrease)

    # Enter maximum move into table, then return
    if maximum == increase:
        table[p_index][t_index] = Coordinate(maximum + viewable, 1)
        return maximum + viewable
    if maximum == stay:
        table[p_index][t_index] = Coordinate(maximum + viewable, 0)
        return maximum + viewable
    if maximum == decrease:
        table[p_index][t_index] = Coordinate(maximum + viewable, -1)
        return maximum + viewable


def trace_back(p, t, table):
    # initialize arrays for events and movements
    events = []
    movements = []
    # add the initial (ending) time, because we're viewing an event
    events.append(t)

    counter = 0
    # Indices
    p_index = p + (len(table) // 2)  # Scale position for table indexing
    t_index = t - 1  # Scale time for table indexing
    # Currents
    p_curr = p_index
    t_curr = t_index

    # Grab the first coordinate, and append movement
    current_coor = table[p_index][t_index]
    movements.append(current_coor.movement*-1)

    # Iterate over all times
    while counter < len(table[0]) - 1:
        # update the current position and time
        t_curr -= 1
        p_curr = p_curr + current_coor.movement

        # step backward in the table, based on the movement
        current_coor = table[p_curr][t_curr]

        # add the current movement to the list, multiplied by -1
        # because we need to know where we're going vs. where we've been
        movements.append(current_coor.movement*-1)

        # if we're viewing an event at the current time, add that
        # unscaled time to our events list
        if (p_curr - (len(table) // 2)) == E[t_curr]:
            events.append(t_curr + 1)
        counter += 1

    # reverse both lists to compensate for backwards nature of algorithm
    events.reverse()
    movements.reverse()
    return events, movements
```

3.2. Demonstration



Event	1	2	3	4	5	6	7	8	9	10
Coordinate	0	0	0	4	3	3	8	−1	6	8



```
csm-wl-dhcp-202-147:Project 3 alpi$ python3 DP.py
4
[1, 2, 5, 10]
[0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
```

Here is the output of our algorithm given the sample coordinates of the events.

4. Appendices

4.1. Appendix 1: Recursive Definition Code

```python
def max_events(p, t):
    viewable = 0
    if abs(p) > t:
        return -100e10
    if t == 0:
        if p == 0:
            return 0
        else:
            return -100e10
    if p == E[t-1]:
        viewable = 1;
    increase = max_events(p + 1, t - 1)
    stay = max_events(p, t - 1)
    decrease = max_events(p - 1, t - 1)
    return max(increase, stay, decrease) + viewable
```