FIT2004 S1/2021: Assignment 2 - Dynamic Programming

DEADLINE: Friday 16th April 2021 23:55:00 AEST

LATE SUBMISSION PENALTY: 10% penalty per day. Submissions more than 7 calendar days late will receive 0. The number of days late is rounded up, e.g. 5 hours late means 1 day late, 27 hours late is 2 days late. For special consideration, please visit this page:
`https://www.monash.edu/connect/forms/modules/course/special-consideration`
and fill out the appropriate form. **Do not** contact the unit directly, as we cannot grant special consideration unless you have used the online form.

PROGRAMMING CRITERIA: It is required that you implement this exercise strictly using the **Python programming language** (version should not be earlier than 3.5). This practical work will be marked on the time complexity, space complexity and functionality of your program, and your documentation.
Your program will be tested using automated test scripts. It is therefore critically important that you name your files and functions as specified in this document. If you do not, it will make your submission difficult to mark, and you will be penalised.

SUBMISSION REQUIREMENT: You will submit a single python file, `assignment2.py`

PLAGIARISM: The assignments will be checked for plagiarism using an advanced plagiarism detector. In previous semesters, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT ACCEPTED. Please do not share your solutions partially or completely to others. If someone asks you for help, ask them to visit a consultation for help.

Learning Outcomes

This assignment achieves the Learning Outcomes of:

- 1) Analyse general problem solving strategies and algorithmic paradigms, and apply them to solving new problems;
- 2) Prove correctness of programs, analyse their space and time complexities;
- 4) Develop and implement algorithms to solve computational problems.

In addition, you will develop the following employability skills:

- Text comprehension
- Designing test cases
- Ability to follow specifications precisely

Assignment timeline

In order to be successful in this assessment, the following steps are provided as a **suggestion**. This is an approach which will be useful to you both in future units, and in industry.

Planning

1. Read the assignment specification as soon as possible and write out a list of questions you have about it.
2. Clarify these questions. You can go to a consultation, talk to your tutor, discuss the tasks with friends or ask in the forums.
3. As soon as possible, start thinking about the problems in the assignment.
    - It is strongly recommended that you **do not** write code until you have a solid feeling for how the problem works and how you will solve it.
4. Writing down small examples and solving them by hand is an excellent tool for coming to a better understanding of the problem.
    - As you are doing this, you will also get a feel for the kinds of edge cases your code will have to deal with.
5. Write down a high level description of the algorithm you will use.
6. Determine the complexity of your algorithm idea, ensuring it meets the requirements.

Implementing

1. Think of test cases that you can use to check if your algorithm works.
    - Use the edge cases you found during the previous phase to inspire your test cases.
    - It is also a good idea to generate large random test cases.
    - Sharing test cases **is** allowed, as it is not helping solve the assignment.
2. Code up your algorithm, (remember decomposition and comments) and test it on the tests you have thought of.
3. Try to break your code. Think of what kinds of inputs you could be presented with which your code might not be able to handle.
    - Large inputs
    - Small inputs
    - Inputs with strange properties
    - What if everything is the same?
    - What if everything is different?
    - etc...

Before submission

- Make sure that the input/output format of your code matches the specification.
- Make sure your filenames match the specification.
- Make sure your functions are named correctly and take the correct inputs.
- Make sure you zip your files correctly (if required)

Documentation (3 marks)

For this assignment (and all assignments in this unit) you are required to document and comment your code appropriately. This documentation/commenting must consist of (but is not limited to)

- For each function, high level description of that function. This should be a one or two sentence explanation of what this function does. One good way of presenting this information is by specifying what the input to the function is, and what output the function produces (if appropriate)
- For each function, the Big-O complexity of that function, in terms of the input. Make sure you specify what the variables involved in your complexity refer to. Remember that the complexity of a function includes the complexity of any function calls it makes.
- Within functions, comments where appropriate. Generally speaking, you would comment complicated lines of code (which you should try to minimise) or a large block of code which performs a clear and distinct task (often blocks like this are good candidates to be their own functions!).

can sort input by constraints so that we can terminate earlier

1 Athlete work schedule (8 marks)

You are an athlete, who mostly works as a personal trainer but also competes in sporting competitions. You can sometimes make more money from competing than from training, but you need time to prepare and recover, which is time that you cannot spend working as a trainer.

You want to maximise the amount of money you can earn by combining your job as a personal trainer with participating in competitions. To do this, you will write a function `best_schedule(weekly_income, competitions)`.

1.1 Input

`weekly_income` is a list of non-negative integers, where `weekly_income[i]` is the amount of money you will earn working as a personal trainer in week i.

`competitions` is a list of tuples, each representing a sporting competition. Each tuple contains 3 non-negative integers, `(start_time, end_time, winnings)`.

`start_time` is the is the week that you will need to begin preparing for this competition (i.e. the first week that you cannot do your regular job as a personal trainer, if you compete in this competition).

`end_time` is the last week that you will need to spend recovering from this competition (i.e. the last week that you cannot do your regular job as a personal trainer, if you compete in this competition).

`winnings` is the amount of money you will win if you compete in this competition.

1.2 Output

`best_schedule` returns an integer, which is the maximum amount of money that can be earned.

1.3 Example

```
regular_work = [3,7,2,1,8,4,5]
special_events = [(1,3,15),(2,2,8),(0,4,30),(3,5,19)]
print(best_events(regular_work, special_events))
>>> 42
```

In the above example, the optimal schedule is to work as a trainer in weeks 0 and 1 (earning 3+7), then compete in the (2,2,8) event in week 2 (earning 8), then compete in the (3,5,19) event from weeks 3 to 5 (earning 19), then work as a trainer in week 6 (earning 5).

1.4 Complexity

`best_schedule` should run in $O(Nlog(N))$ time and $O(N)$ space, where N is the total number of elements in `weekly_income` and `competitions` put together.

2 Sales itinerary (9 marks)

A salesperson lives on the coast. They travel to various cities along the coast to work. Sometimes they stay in a city for a day or a few days to work, and other times they simply pass through on their way to another city.

Since Covid-19, each city has instituted a policy of having travelers quarantine, but only if they want to stay in the city. If they are just passing through, they can continue on their way without quarantining. Also, each city asks visitors to quarantine for a different amount of time.

The salesperson has an idea of how much money they can make by working in each city, for each day. They need to decide which cities to travel to, and which cities to work in, in order to make the most money.

Each day, the salesperson can either work for the day in their current city (assuming they have finished quarantine), or they can travel to either adjacent city. Traveling always takes 1 day, and since the cities are along a coast, each city has two adjacent cities, except for two cities on the ends of the coast, which only have 1.

To solve this problem, you will write a function `best_itinerary(profit, quarantine_time, home)`.

2.1 Input

We think of the `n` cities as being numbered `0...n-1`. In one day, the salesperson can travel from city `i` to either city `i+1` or `i-1`. From city `0` they can only travel to city `1`, and from city `n-1` they can only travel to city `n-2`.

`profit` is a list of lists. All interior lists are length n. Each interior list represents a different day. `profit[d][c]` is the profit that the salesperson will make by working in city `c` on day `d`.

`quarantime_time` is a list of non-negative integers. `quarantime_time[i]` is the number of days city `i` requires visitors to quarantine before they can work there.

`home` is an integer between `0` and `n-1` inclusive, which represents the city that the salesperson starts in. They can start working in this city without needing to quarantine on the first day. If they leave and come back later, they will need to quarantine.

2.2 Output

`best_itinerary` returns an integer, which is the maximum amount of money that can be earned by the salesperson.

2.3 Example

```
profit = [
[6, 9, 7, 5, 9]
[4, 7, 3, 10, 9]
[7, 5, 4, 2, 8]
[2, 7, 10, 9, 5]
[2, 5, 2, 6, 1]
[4, 9, 4, 10, 6]
[2, 2, 4, 8, 7]
[4, 10, 2, 7, 4]]

quarantine = [3,1,1,1,1]
best_itinerary(profit, quarantine, 0)
>>> 39
best_itinerary(profit, quarantine, 1)
>>> 54
best_itinerary(profit, quarantine, 2)
>>> 47
best_itinerary(profit, quarantine, 3)
>>> 57
best_itinerary(profit, quarantine, 4)
>>> 51
```

2.4 Explanation of example

- h=0. The salesperson works in city 0 on the first day, then travels to city 1 on the second day, quarantines on the third day, then works in city 1 for the remaining days.
- h=1. The salesperson works in city 1 every day.
- h = 2. The salesperson works in city 2 on the first day, then travels to city 3 on the second day, quarantines on the third day, then works in city 3 for the remaining days.
- h=3. The salesperson works in city 3 every day.
- h=4. The salesperson works in city 4 for 3 days, then travels to city 3 on the fourth day, quarantines for a day, then works in city 3 for 3 days.

2.5 More examples

Since it is difficult to construct test cases for this question, I will be posting several additional test cases on the forums. As usual, students should feel free to create even more test cases and post them.

2.6 Complexity

`best_itinerary` should run in $O(nd)$ time and space, where n is the number of cities, and d is the number of days (i.e. the number of interior lists in `profit`). In the above example, n = 5, d = 8.

Warning

For all assignments in this unit, you may **not** use python **dictionaries** or **sets**. This is because the complexity requirements for the assignment are all deterministic worst case requirements, and dictionaries/sets are based on hash tables, for which it is difficult to determine the deterministic worst case behaviour.

Please ensure that you carefully check the complexity of each inbuilt python function and data structure that you use, as many of them make the complexities of your algorithms worse. Common examples which cause students to lose marks are **list slicing**, inserting or deleting elements **in the middle or front of a list** (linear time), using the `in` keyword to **check for membership** of an iterable (linear time), or building a string using **repeated concatenation** of characters. Note that use of these functions/techniques is **not forbidden**, however you should exercise care when using them.

These are just a few examples, so be careful. Remember, you are responsible for the complexity of every line of code you write!