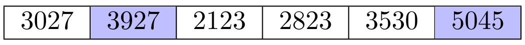
This assignment is for both COMP3027 and COMP3927 students.

Task 1: Social Distancing [30 marks]

You and your genius friend earned enough licensing fees from the deep quantum neural networks to retire. Finally, you can indulge in your lifelong dream of running a fleet of Mexican food buses called Nacho Bus-iness. You want to set them up along a street in downtown for Cinco de Mayo. However, due to social distancing requirements, the buses cannot be placed too close together. Your goal is to find a placement of the buses satisfying the social distancing requirement that maximizes total revenue.

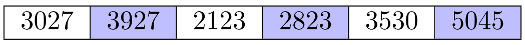
Formally, you are given as input an array R of n positive integers (not necessarily distinct) and the social distancing requirement $d \geq 1$. The integer $R[i]$ is the amount of revenue earned by a food bus placed at location i on the street. You can place as many buses as you wish, subject to the social distancing requirement: if a bus is placed at location i, then the nearest location that we can place another bus is $i-d$ and $i+d$; no other bus can be placed at locations $i-d+1, \ldots, i+d-1$. Your goal is to design an efficient dynamic programming algorithm to determine the maximum revenue that can be earned by a placement of food buses that satisfy the social distancing requirement. (Note that we are only interested in the revenue, not the actual placement.)

For example, consider the input $R = [3027, 3927, 2123, 2823, 3530, 5045]$ and $d = 3$. The optimal solution is to place buses at locations 1 and 5 for a total revenue of $3927 + 5045 = 8972$. For the same R and $d = 2$, the optimal solution is to place buses at locations 1, 3 and 5 for a total revenue of $3927 + 2823 + 5045 = 11795$. (Arrays start with index 0.)



3027	3927	2123	2823	3530	5045

Figure 1: Optimal solution for $d = 3$



3027	3927	2123	2823	3530	5045

Figure 2: Optimal solution for $d = 2$

(a) Define the subproblems needed by your algorithm.

What we expect to see: Definition of OPT(. . .), and its parameters

(b) Define the recurrence.

What we expect to see: a recurrence that relates each subproblem to "smaller" subproblems. We just want to see a clear description of the recurrence here. The justification for the recurrence goes into (e).

(c) Define the base cases.

What we expect to see: A description of the base cases and the values that OPT(..) takes on these base cases.

(d) Describe how the maximum revenue is obtained from $OPT(..)$.

What we expect to see: A description of how to obtain the optimal value to the original problem using the optimal values of the subproblems.

(e) Prove the correctness of your recurrence and base cases, and that the algorithm correctly returns the optimal value.

What we expect to see: The sufficiency of the base cases, i.e. the recurrence will always end up at a base case, the correctness of the recurrence and the values for the base cases, and that you correctly obtained the optimal revenue from OPT(..).

(f) Prove the time and space complexity of your algorithm.

What we expect to see: An analysis of the running time and space required to compute all the $OPT(..)$ values and to obtain the maximum revenue from $OPT(..)$

Task 2: Grid Climbing [70 marks]

Congratulations! You've made it to the finals of the World Grid Climbing competition. You start at the bottom-left corner of an $n \times m$ grid and are only allowed to climb rightwards or upwards. Each grid position (i, j) has a score $S[i, j]$. (The bottom-left position is $(0, 0)$ and the top-right is $(n-1, m-1)$.) You have an energy budget of B. Moving right costs c_r amount of energy and moving up costs c_u. The numbers B, c_r and c_u are non-negative integers. A climbing path consists of a sequence of rightwards and upwards moves and is feasible if the total energy cost is at most the budget. The goal is to find the maximum total score that can be obtained by a feasible climbing path. (Note that we are only interested in the score, not the actual path.)

For instance, suppose S is the following grid, $B = 10$, $c_u = 3$, and $c_r = 1$. Then, the optimal path is $(0, 0) \rightarrow (0, 1) \uparrow (1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \uparrow (2, 3)$ for a total score of $5 + 4 + 100 + 7 + 9 + 10 = 135$ and cost of $2 \times c_u + 3 \times c_r = 9$.

2	1	4	5
8	3	3	10
3	100	7	9
5	4	6	8

Figure 3: The grid S

2	1	4	5
8	3	3	**10**
3	**100**	**7**	**9**
5	**4**	6	8

Figure 4: The optimal path

For COMP3027: your algorithm should take $O(nmB)$ time and space for full marks. For COMP3927: your algorithm should take $O(nm)$ time and space for full marks; an $O(nmB)$ time and space algorithm is worth 50 marks.

(a) Define the subproblems needed by your algorithm.

What we expect to see: Definition of OPT(. . .), and its parameters

(b) Define the recurrence.

What we expect to see: a recurrence that relates each subproblem to "smaller" subproblems. We just want to see a clear description of the recurrence here. The justification for the recurrence goes into (e).

(c) Define the base cases.

What we expect to see: A description of the base cases and the values that OPT(..) takes on these base cases.

(d) Describe how the maximum total score is obtained from $OPT(..)$.

What we expect to see: A description of how to obtain the optimal value to the original problem using the optimal values of the subproblems.

(e) Prove the correctness of your recurrence and base cases, as well as how you obtain the optimal value from the subproblems.

What we expect to see: The sufficiency of the base cases, i.e. the recurrence will always end up at a base case, the correctness of the recurrence and the values for the base cases, and that you correctly obtained the optimal score from $OPT(..)$.

(f) Prove the time and space complexity of your algorithm.

What we expect to see: An analysis of the running time and space required to compute all the $OPT(..)$ values and to obtain the optimal score from $OPT(..)$.

(g) Implement your algorithm on Ed.

Submission details

- **Submission deadline is Wednesday 21 April, at 23:59.** Late submissions without special consideration will be subject to the penalties specified in the first lecture (20% per day). **Submissions later than Friday 23 April, 23:59 will not be accepted.**
- Submission time is the max of the submission times to Gradescope and Ed. For example, if you submit to Gradescope on time but your implementation on Ed is one day late, 20% will be deducted from the entire assignment.
- Submit your answers as a single document to Gradescope. Your work must be typed (no images of text, although you can use diagrams if you think it helps.) Please try to be reasonably concise (about 2 pages of A4)
- The implementation required for Task 2 should be done in Ed, and submitted via Ed.
- Your report will be subject to automatic and manual plagiarism detection systems. Remember, it's acceptable to discuss high level ideas with your peers, but you should not share the detail of your work, such as parts as the precise algorithms, examples, proofs, writing, or code.
- To facilitate anonymous grading, please do not write your name on your submission.

Exemplar for Knapsack

(a) Define the subproblems needed by your algorithm.

Consider the items in some arbitrary order. Define $\text{OPT}(i, w)$ to be the optimal solution to the knapsack problem consisting of the first i items and weight limit w for $0 \le i \le n$ and $0 \le w \le W$. When $i = 0$, this corresponds to a knapsack problem with no items.

(b) Define the recurrence.

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w_i > w, \\ \max\{v_i + \text{OPT}(i-1, w-w_i), \text{OPT}(i-1, w)\} & \text{otherwise} \end{cases}$$

(c) Define the base cases

The base cases are when $i = 0$. In this case, $\text{OPT}(0, w) = 0$ for all $0 \le w \le W$.

(d) Describe how to obtain the most value achieved by any feasible subset of items from the subproblems.

The most value achieved is in the subproblem $\text{OPT}(n, W)$.

(e) Prove the correctness of your recurrence, base cases, as well as how you obtain the optimal value to the original problem from $OPT(..)$.

The base cases are sufficient as each $\text{OPT}(i, w)$ depends on some $\text{OPT}(i-1, w)$. Thus, the recurrence will always end up at $\text{OPT}(0, w)$ for some w. We have $\text{OPT}(0, w) = 0$ since the most value that can be achieved with no items is 0.

We now prove the correctness of the recurrence. Let $S(i, w)$ be the optimal solution for the first i items with weight limit w. The recurrence for $\text{OPT}(i, w)$ is correct because:

- if $w_i > w$, then $S(i, w)$ cannot contain item i and so is a subset of the first $i - 1$ items with weight at most w, so $S(i, w) = S(i - 1, w)$ and $\text{OPT}(i, w) = \text{OPT}(i - 1, w)$
- if $w_i \le w$, then either item i is in $S(i, w)$ or not:

  - if item i is in $S(i, w)$, then the remaining items of $S(i, w)$ is a subset of the first $i - 1$ items with weight at most $w - w_i$ so $S(i, w) = \{i\} \cup S(i - 1, w - w_i)$ and so $\mathrm{OPT}(i, w) = v_i + \mathrm{OPT}(i - 1, w - w_i)$.
  - else, $S(i, w)$ is a subset of the first $i-1$ items with weight at most w, so $S(i, w) = S(i-1, w)$ and $\mathrm{OPT}(i, w) = \mathrm{OPT}(i - 1, w)$.

Finally, the optimal value for the original problem is $\mathrm{OPT}(n, W)$ as by definition of OPT, this is the most value that can be achieved by a subset of all n items with a weight limit of W.

(f) Prove the time and space complexity of your algorithm.

- Space complexity. The space required by our algorithm is dominated by the number of subproblems, which is $O(nW)$ as we have a subproblem $\mathrm{OPT}(i, w)$ for each $0 \leq i \leq n$ and $0 \leq w \leq W$.
- Time complexity. The base cases take time $O(W)$ as there are $W + 1$ of them and setting them to 0 takes constant time. For the recursive cases, there are $O(nW)$ iterations and each iteration takes $O(1)$ time since it only requires a constant number of comparisons, arithmetic operations and array lookups. Thus, the recursive cases takes time $O(nW)$. Returning the optimal value takes $O(1)$ time for an array lookup. Thus, the total time is $O(W) + O(nW) + O(1) = O(nW)$.