Dynamic Programming
& Bipartite Matching
Problem Set 5 – CS6515 (Spring 2025)

- This problem set is due on **Thursday February 20th**.
- Submission is via Gradescope.
- Your solution must be a typed pdf (e.g., via LaTeX, Markdown, etc. Anything that allows you to type math notation) – no handwritten solutions.
- Please try to make your solutions as concise and readable as possible. Most problems will have solutions that are no more than a page long. Consider using bullet points and adding space to break up large paragraphs into smaller chunks.
- There are 3 problems. Each problem is graded with 20p. **There is +1p bonus per problem** for stating (i) how long it took you to solve that problem, and (ii) how long it took you to type the answer.

Remark on algorithm descriptions To make things easier for the TAs to grade, please use a combination of plain English and mathematical notation. Do not write code (C, Java, etc.). For example, you can say something like "$x := \max_{i=0,\ldots,n-1}$ `A[i]`" or "Find the maximum value in the array A" instead of writing a for-loop that computes the maximum.

13 DP-flix

An online streaming platform has a marathon event for your favorite drama series. All episodes are streamed in sequence without any interruption. You have to pay the streaming platform for each watched episode, but to encourage longer viewing sessions, the platform has a special offer. If you watch k consecutive episodes, you only pay $\max(1, k-1)$.

For example, if there are 20 episodes in total, and you watch episodes 1,2,5,6,7,8,20, then you watch 3 contiguous blocks: [1,2], [5...8], and [20]. So this will cost max(2-1,1)+max(4-1,1)+max(1-1,1)=1+3+1=5.

There is also some online rating webpage that provides a rating for each episode. The rating of the ith episode is $R[i]$.

Let n be the total number of episodes, R an array of episode ratings, and K the amount of money you are willing to spend. What is the maximum total rating of the episodes you can watch?

13.1 Theory option

Solve this problem via dynamic programming.

(a) Give the array and describe what an entry means. (E.g., $T[i]$ or $T[i, j]$ is...)

(b) Formulate the recursion as compact as possible with mathematical notation using your array. Explain briefly why it's correct.

(c) Which cells of the array form the base case, and what are their initial values?

(d) Which cell (or cells) contain the answer?

(e) What is the time complexity to fill the array and return the solution?

For full points, the algorithm should run in $O(nK)$ time.

13.2 Programming option

Solve this problem via dynamic programming and submit your program to gradescope.

Write a program that receives an array R (the ratings for n episodes), and an integer K (the amount of money you are willing to spend). The program should return the maximum total rating of the episodes you can watch.

Input There are code templates on Piazza that handle reading the input.

In case you do not want to use the templates, the input will be of the following format. The first line is n and k, separated by space. The next line are n numbers, separated by space, specifying the rating of each episode. Example:

```
7 3
1 3 20 19 4 8 10
```

Here $n = 7$ episodes, $K = 3$ are the available funds, and the ratings for the episodes are $[1, 3, 20, 19, 4, 8, 10]$.

The correct output would be 60, via 3+20+19+8+10.

Grades: Gradescope will automatically test your submission. You can resubmit as often as you want (until the due date) and you can see on gradescope how many test cases you currently passed.

The automated system will give partial credit 12/20 when the submission passes the first 6 test cases (which corresponds to a correct algorithm with $O(n^2K)$ time complexity) and full credit when passing all test cases (which corresponds to a correct algorithm with $O(nK)$ time complexity).

While the tentative grade is based on the number of passed test cases, the final grade is given by a TA to verify whether the submission solves the actual problem. E.g., if your code passed because you hardcoded the correct output, then you will receive 0 points. (We need this policy because we make the test cases and expected outputs public to help you find any errors in your algorithm.)

14 Transport Distance

Consider a binary tree $T = (V, E)$ with root $r \in V$ where each edge $e \in E$ has an associated length $l_e \in \mathbb{R}_{>0}$, and each vertex $v \in V$ holds a quantity of goods $g_v \in \mathbb{R}_{>0}$. We are also given a parameter $k \leq |V|$. Your task is to select k vertices, to serve as distribution centers that minimizes the total transport distance from all vertices to their nearest distribution center. The goods can only move in the upward direction towards the root (i.e., all edges are directed upwards).

For a set of k distribution centers and $v \in V$, let $d_v \in V$ be the nearest distribution center to node v. Then the "total transport distance" in the graph is

$$\sum_{v \in V} \text{dist}(v, d_v) \cdot g_v$$

We want to design a DP algorithm to solve this problem by answering the following questions. For full points, we are looking for an $O(|V|^2k^2)$-time algorithm. You may assume that for any pair $u, v \in V$, you can get $\text{dist}(u, v)$ in $O(1)$ time.

(a) Give the array and describe what an entry means. (E.g., $T[i]$, $T[i, j]$ or $T[i, j, k]$ is ...)

(b) Formulate the recursion as compact as possible with mathematical notation using your array. Explain briefly why it's correct

(c) Which cells of the array form the base case, and what are their initial values?

(d) Which cell (or cells) contain the answer?

(e) What is the time complexity to fill the array and return the solution?

Hint: Often for DP on trees, the table $T[u, ...]$ only stores information about the sub-tree rooted at $u \in V$. However, for this exercise it might help to add another index $v \in V$ to the table (i.e., $T[u, ..., v]$) that represents the closest distribution center higher up in the tree, above the subtree.

15 Approximate Matching

Let $G = (V, E)$ be a bipartite graph with m edges and n vertices. In class we have seen an algorithm that finds the maximum matching in time $O(mn)$ by repeatedly finding augmenting paths. We now want to construct an algorithm that finds a (2/3)-approximation in much faster $O(m)$ time. Proof question (a) below serves as a hint for finding such an algorithm.

Problem:

(a) Let M be a matching with the property that every remaining augmenting path has length > 3 (so length ≥ 5, since augmenting paths have odd length). Prove that $|M| \geq (2/3)k$ where k is the size of a maximum matching.

(b) Construct an $O(m)$ time algorithm that finds a (2/3)-approximate maximum matching. Analyze its time complexity. (You do not need to give proof of correctness, but please add a short explanation why the algorithm correctly eliminates all length ≤ 3 augmenting paths, to help the grading TA understand your algorithm.)

Hint for a: Let M' be the maximum matching. What does $H := M' \Delta M$ look like? Does H contain long paths? What does that tell us about the size of M vs M'?

Hint for b: If repeated BFS/DFS ends up too slow, try a different approach. What structure does an augmenting path of length 3 have?