Name: Yiyan Yang

zID: z5183946

Question 1

Question Part

You have a big pizza with n slices. Unfortunately, there's so much cheese on it that whenever you remove a piece from the pizza, some of the toppings fall off. Before you start eating you have analysed each slice i and determined that if you take the slice when there are k slices remaining (including that slice), then you will loose $a_i + bi \times k$ grams of topping. You would like to eat the pizza while loosing a little topping as possible. At any point, you may only eat a piece with an empty space next to it (except for the first piece, for which you may eat any one). The objective of this part is to obtain a dynamic programming algorithm to solve the following task (T): determine the minimal amount of topping lost as you eat the entire pizza.

Solution

Question 1.1

Define the subproblems used in a dynamic programming approach to solve task T.

Answer:

$P(i, j, m)$: Find the minimal amount of topping lost as you eat all the pizza from i to j, the numbers of slices have been eaten is m.

Notice: we only need two arguments to define a subproblem, but for clarity, we

Question 1.2

Express the recurrence relationship between subproblems and give the solution to the base case(s).

Answer:

First of all, we define a function to calculate the grams of topping lost when taking ith slice with k slices remaining $C(i, k)$.

For any $P(i, j, m)$, we assume that we have solved all the subproblems for all $m` < m$. Then we update $P(i-1, j, m+1)\ and\ P(i, j+1, m+1)$ if the cost of $P(i, j, m) + C(i-1, n-m)$ is smaller than the original cost(cost is defaulted by infinitely large).

The base case should be the case we have taken off one slice of pizza, which is $P(i, i, 1) = a_i$ for any $i\ that\ 0 \leq i \leq n$.

Question 1.3

Give the pseudo-code of an algorithm to solve task T.

Answer:

Helper functions:

$Cost(i, m)$ *where* i *is the slice that is going to be taken,* m *is the remaining number of slices.*
  $return\ a_i\ +\ b_i\ *\ m$

$Update(P(i, j\ , m))$
  $P(i\ -\ 1, j, m\ +\ 1)$
    $= min(\ P(i\ -\ 1, j, m\ +\ 1), P(i, j, m)\ +\ Cost(i\ -\ 1, n\ -\ m))$
  $P(i, j\ +\ 1, m\ +\ 1)$
    $= min(\ P(i, j\ +\ 1, m\ +\ 1), P(i, j\ , m)\ +\ Cost(j\ +\ 1, n\ -\ m))$

Algorithm:

$Set\ all\ P\ to\ infinitely\ large.$

$for\ i\ from\ 0\ to\ n - 1$
  $P(i, i, 1)\ =\ a_i$

$for\ m\ from\ 1\ to\ n - 1$
  $for\ i\ from\ 0\ to\ n$
    $Update(\ P(i, (i + m - 1)\%n, m + 1)\ )$

Question 1.4

What is the worst-case complexity of your algorithm?

Answer:

It's $O(n^2)$.

Because we have n starter slices, and for each m, we have 2 ways to update the total grams. In addition, m is an integer between 1 and n. Thus we need to compute $n * 2n\ =\ 2n^2$ times, which is $O(n^2)$.

In the code running aspect, the worst case is that each part of the array is executed, which is $n * n * 2\ =\ 2n^2$, namely, $O(n^2)$.

Question 2

Question Part

You have a row of n boxes. In each one sits a marble. Some of the marbles are coloured. For each coloured marble, there is a box of the same colour. All other marbles and boxes are black. Marbles are not necessarily sitting in a box of their own colour. All coloured marbles have a different colour from one another.
The cost of a configuration is defined as the total sum of the distances between each coloured marble and its corresponding box. For example, consider the following configuration with n = 8 boxes and marbles.

Solution

Question 2.1

What is the cost of the configuration displayed above?

The cost is $5 + 2 + 4 + 2 + 5 + 1 = 19$

Question 2.2

You wish to swap some marbles such that the cost is minimised. However, the following rules apply:

- Each swap involves one coloured marble and one a black marble.
- If two marbles are swapped, then all marbles sitting between them must remain in their original location.
- No marble may be swapped twice.

In the configuration above, what set of legal swaps leads to a configuration with the smallest cost? Answer using the following format: e.g., $\{(\mathbf{4},\mathbf{5}),(\mathbf{6},\mathbf{8})\}$.

$$\boldsymbol{Answer}:$$

$\{(7,8),(1,4)\}$ with cost 15.

Question 2.3

We would like to develop an $O(n^2)$ Dynamic Programming algorithm to determine the minimal possible cost reachable from an input starting configuration.

$Subquadratic$ algorithms (faster than $O(n^2)$) may exist but are not required.
Define the subproblems.

$\boldsymbol{SubProblems}$:

$\boldsymbol{P(i,j)}$: The minimum configuration cost while only considering first i marbles and first j black marbles.

Notice, if i_j is the index of current black marble we are considering, for all $P(i,j)\ that\ 0 < i < i_j$, the value is the configuration cost of $P(i, j-1)$ (and it's the original cost for the base case). Moreover, $P(i_j, j)$ stores the minimum configuration cost with jth marble swapped with any other coloured marble whose index is smaller than i_j.

If it's necessary to record the pairs we swapped, we also need to keep a record for $P(i,j)$ while updating the minimum configuration cost.

Question 2.4

Give the recurrence equation(s) linking the subproblems and give the solution to the base case(s).

$\boldsymbol{Answer}$:

Suppose we have m black marbles and index for jth black marble is i_j.

$\boldsymbol{Helper\ function}$:

$Swap(i,j)$ returns the cost reduced after swapping $ith\ and\ jth$ marbles. The return value can be calculated in $O(1)$ by comparing absolute differences d_i, d_j between i, j and the corresponding box of i respectively.

If $d_j < d_j$, return $\mathrm{d_i} - d_j$, otherwise, return 0.

$\boldsymbol{Recurrence}$

$$P(i_j, j) = \min\big(P(i_j, j),\ P(i-1, j-1) - Swap(i, i_j)\big) \qquad 1 \le i \le i_j, 1 \le j \le m$$

$$\mathrm{P}(i,j) = \min\Big(\mathrm{P}(i-1, j), \mathrm{P}(i, j-1) - \mathrm{Swap}\big(i, i_j\big)\Big) \qquad i_j < \mathrm{i} \le \mathrm{n}, 1 \le \mathrm{j} \le \mathrm{m}$$

$$P(n,m)\ is\ the\ expected\ answer.$$

$\boldsymbol{Base\ case}$:

$$P(i,j) = original\ cost \qquad 0 \le i \le n, 0 \le j \le m$$

We define $P(0,j) = original\ cost$ for any $j: 1 \le j \le m$ although we don't have marble with index 0.

The original cost can be calculated in linear time by adding up the absolute difference between each pair of coloured marble and its corresponding box.

Question 2.5

Prove that your approach leads to an algorithm with $O(n^2)$ complexity.

Answer:

We run the algorithm in the nested loops like this:

Calculate P(i, j) for all j from 0 to m
    Calculate P(i, j) for all i from 1 to n

The complexity of calculating $P(i, j)$ for all i with certain j is $O(n)$ while the complexity of choosing $P(i, j)$ for each j is m which is smaller than n. Thus, the overall complexity is $mn = O(mn) = O(n^2)$

Question 3

Question Part

You are given a set of points in the plane. Some of these points have line segments between them. These line segments do not overlap, except at endpoints. These line segments form a set of (potentially irregular) polygons. Polygons are not self-intersecting, and do not overlap, although they may share edges and/or vertices. You wish to colour as many of the polygons with one of two colours (say blue and red) such that no two polygons that share an edge have the same colour. Polygons may remain uncoloured. Two uncoloured polygons may share an edge.

We would like to solve this problem using integer linear programming (ILP).

Solution

Question 3.1

The first step in modelling of this problem is to represent it as a problem on a graph (V, E).

What would each vertex $v \in V$ represent? What would each edge $e \in E$ represent?

Answer:

v represents a point that can be connected to another point to form a segment, and it could be the vertex of certain polygons.

e represents a segment connected by two vertices who may have two adjacent polygons on both sides.

Question 3.2

List the variables for the ILP problem.

Answer:

$\boldsymbol{x} = < x_1, x_2, \dots, x_n >$ is the colouring state of each polygon, while n is the number of polygons.

$\boldsymbol{y} = < y_1, y_2, \dots, y_m >$ is the colouring state of the polygons adjacent to certain edge, $y_i = n$ means that there are n polygons coloured who have an edge e_i, while m is the number of edges.

Question 3.3

List the constraints.

Answer:

We assume that there are n vertices and m edges.

$0 \leq x_i \leq 1 \qquad 1 \leq i \leq n$

$0 \leq y_i \leq 1 \qquad 1 \leq i \leq m$

Question 3.4

State the objective function.

Answer:

Objective is to maximise

$$\sum_{i=1}^{n} x_i$$

In other words, to colour as many polygons as possible while keeping no adjacent coloured polygons exist.

Question 4

Question Part

You are stacking n bags onto a plane. Bags must be stacked in columns, up to h high. The plane's cargo hold is p columns wide. The amount of force that bag i placed in pile $q \in [1, p]$ applies to the plane is given by f_{iq}. You wish for the sum of these forces from all bags to be as small as possible. We aim at modeling this problem as a mathematical optimization problem.

Solution

Question 4.1

Is it better to model this problem using Linear Programming or Integer Linear Programming? Why? (One or two sentences in English should be sufficient to explain your reasoning.)

$\boldsymbol{Answer}$:

It is better to model this problem using Integer Linear Programming. The reason is that each bag can only be placed in one column while the number of bags and column can only be integers.

Question 4.2

List the variables for the problem.

$\boldsymbol{Answer}$:

$\boldsymbol{x} = < x_{11}, x_{12}, \dots, x_{1p}, x_{21}, x_{22}, \dots x_{2p}, \dots x_{np} >$

$x_{iq} = 1$ means put ith bag on the qth column.

Question 4.3

List the constraints.

Answer:

$$\sum_{i=1}^{n} x_{iq} < h \qquad 1 \le q \le p$$

$$\sum_{q=1}^{q} x_{iq} \le 1 \qquad 1 \le i \le n$$

$$\sum_{q=1}^{q} x_{iq} \ge 1 \qquad 1 \le i \le n$$

$$x_{iq} \ge 0 \qquad 1 \le i \le n, 1 \le q \le p$$

$$x_{iq} \le 1 \qquad 1 \le i \le n, 1 \le q \le p$$

Question 4.4

State the objective function.

Answer:

Objective is to minimise

$$\sum_{i=1}^{n} \sum_{q=1}^{p} f_{iq} x_{iq}$$