1 Study Group

List the names and SIDs of the members in your study group.

2 Three Partition

Given a list of positive numbers, $a_1, \ldots, a_n$, determine if we can partition $\{1, \ldots, n\}$ into 3 disjoint subsets, $I, J, K$ such that:

$$
\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{\sum_{i=1}^{n} a_i}{3}
$$

Devise and analyze a dynamic programming solution to the above problem that runs in time polynomial in $\sum a_i$ and $n$.

Please provide a 3-part solution.

**Solution: Main Idea:**

Our algorithm will consist of subproblems $A(i, j, k)$, each of which is true or false depending on whether it is possible to find two disjoint subsets $I_k$ and $J_k$ of $\{1, \ldots, k\}$ such that:

$$
\sum_{m \in I_k} a_m = i, \quad \sum_{l \in J_k} a_l = j.
$$

Using $w$ to denote $\sum_{i=1}^{n} a_i$, we may assume that $i, j$ are in the range from 0 to $w$ and that $k$ ranges from 0 to $n$. We compute the value of $A(i, j, k)$ using the following recursion:

$$
A(i, j, k) = A(i - a_k, j, k - 1) \lor A(i, j - a_k, k - 1) \lor A(i, j, k - 1)
$$

The base cases are

$$
A(i, j, 0) = \begin{cases} 
\text{True}, & \text{if } i = j = 0 \\
\text{False}, & \text{otherwise}
\end{cases}
$$

Our dynamic programming algorithm now starts with the base case where $k = 0$ and computes the value for $A(i, j, 0)$ for all combinations of $i$ and $j$. It then recursively computes the values of $A(i, j, k)$ for all combinations of $i$ and $j$ by using the previously computed values for $A(i, j, k - 1)$. Our algorithm finally returns $A(w/3, w/3, n)$.

**Runtime Analysis:**

Notice that each subproblem takes constant time to solve given solutions to all the previous subproblems. The runtime of our algorithm is bounded by $w^2 n$.

**Proof of Correctness:**

For the case $k = 0$, our algorithm trivially computes the right answers to the subproblems. Now assume that the values $A(i, j, k)$ are computed correctly for all combinations of $i, j$ and $k$ up to $l - 1$. Suppose we wish to compute the value of $A(i, j, l)$. Notice that if the algorithm
outputs True, there definitely exists a positive solution to the subproblem. If \( A(i-a_l, j, l-1) \) is True, our solution is simply \( \{l\} \cup I_{l-1}, J_{l-1} \) where \( I_{l-1}, J_{l-1} \) are solutions to \( A(i-a_l, j, l-1) \). Similarly, \( A(i, j-a_l, l-1) \) corresponds to adding \( l \) to \( I_{l-1} \), and \( A(i, j, l-1) \) corresponds to the solution \( I_{l-1}, J_{l-1} \). On the other hand, if \( A(i, j, l) \) is True, then there is some solution \( I_l, J_l \), and either \( l \in I_l \) or \( l \in J_l \) or \( I_l \not\subseteq I_l, J_l \). In either case, \( A(i, j, l) \) evaluates to True as one of \( A(i-a_l, j, l-1), A(i, j-a_l, l-1), A(i, j, l-1) \) have to be True. This proves the correctness of the algorithm.

3 Maximum weight independent set

Let \( G = (V, E) \) be a weighted graph, with nonnegative weights \( w(v) \) for each vertex \( v \in V \). A subset of nodes \( S \subset V \) is an independent set of \( G \) if there are no edges between them. Assuming that \( G \) is a tree, find a linear time algorithm for finding the maximum weight independent set in \( G \), i.e. an independent set \( S \) of \( G \) such that \( \sum_{v \in S} w(v) \) is maximized.

Please provide a 3-part solution.

Solution:

Main Idea:
Each vertex in the tree defines a subtree, and we define a subproblem for each subtree as follows.

\[
I(v) = \text{total weight of maximum weight independent set of subtree rooted at } v.
\]

Our goal is to compute \( I(r) \), where \( r \) is the root of the tree. Our recurrence relation is as follows.

\[
I(v) = \max \left\{ w(v) + \sum_{\text{grandchildren } u \text{ of } v} I(u), \sum_{\text{children } u \text{ of } v} I(u) \right\}.
\]

The children of \( V \) can only be in the maximum weight independent set of the subtree rooted at \( v \), if the set does not include \( v \). If not, then the maximum weight independent set consists of \( v \) and the union of the maximum weight independent sets of the subtrees of the grandchildren of \( v \).

Runtime Analysis:
The running time is \( O(n) \), where \( n \) is the number of vertices. Each vertex \( v \) is only processed 3 times: when the algorithm is processing \( v \), when it is processing \( v \)'s parent and when it is processing \( v \)'s grandparent.

Proof of Correctness:

Base Case: The base cases are the leaves of the tree. For each leaf \( v \), the maximum weight independent set consists of just \( v \), so \( I(v) = w(v) \).

Inductive Step: To compute \( I(v) \), if \( v \) is in the maximum weight independent set, then the children of \( v \) cannot be in the maximum weight independent set. Hence we take the union of \( v \) with the maximum weight independent sets of the subtrees rooted at the grandchildren of \( v \), so \( I(v) \) is \( w(v) \) plus \( I(u) \) for the grandchildren \( u \) of \( v \).
If $v$ is not in the maximum weight independent set, then the children of $v$ can be in the maximum weight independent set. The maximum weight independent set is the union of the maximum weight independent sets of the subtrees rooted at the children of $v$.

### 4 (Linear Programming) Minimum Spanning Trees

Consider the minimum spanning tree problem, where we are given an undirected graph $G$ with edge weights $w_{u,v}$ for every pair of vertices $u, v$.

An integer linear program that solves the minimum spanning tree problem is as follows:

Minimize $\sum_{(u,v) \in E} w_{u,v} x_{u,v}$

subject to $\sum_{(u,v) \in E : u \in S, v \in V \setminus S} x_{u,v} \geq 1$ for all $S \subseteq V$ with $0 < |S| < |V|$

$\sum_{(u,v) \in E} x_{u,v} \leq |V| - 1$

$x_{u,v} \in \{0, 1\}, \ \forall (u,v) \in E$

(a) Show how to obtain a minimum spanning tree $T$ of $G$ from an optimal solution of the ILP, and prove that $T$ is indeed an MST. Why do we need the constraint $x_{u,v} \in \{0, 1\}$?

(b) How many constraints does the program have?

(c) Suppose that we replaced the binary constraint on each of the decision variables $x_{u,v}$ with the pair of constraints:

\[ 0 \leq x_{u,v} \leq 1, \ \forall (u,v) \in E \]

How does this affect the optimal value of the program? Give an example of a graph where the optimal value of the relaxed linear program differs from the optimal value of the integer linear program.

**Solution:**

(a) $T = \{(u,v) \in E : x_{u,v} = 1\}$. The first constraint ensures that $T$ is connected (there is at least one edge crossing every cut). The second constraint ensures that $T$ is a tree. Moreover, every spanning tree $T$ is a feasible solution of the ILP. The objective is the weight of $T$, and so the optimum is the MST. We need $x_{u,v} \in \{0, 1\}$ because it’s not clear what you’d do with a fractional edge.

(b) There are $2^{|V|} + |E| - 1 = \Theta(2^{|V|})$ constraints.

(c) $v_{LP} \leq v_{ILP}$. The new linear program solution’s objective value $v_{LP}$ is at most the integer linear program’s objective value $v_{ILP}$, because every feasible solution of the ILP is a feasible solution of the LP.
One example is a cycle with 3 nodes, \( w_{u,v} = 1, \forall u, v \in E \). The optimal ILP formulation picks any two of the edges for a total objective cost of 2. The optimal LP formulation picks \( x_{u,v} = \frac{1}{2} \) for all edges, for a total objective cost of \( \frac{3}{2} \).

5 Jeweler

You are a jeweler who sells necklaces and rings. Each necklace takes 4 ounces of gold and 2 diamonds to produce, each ring takes 1 ounce of gold and 3 diamonds to produce. You have 80 ounces of gold and 90 diamonds. You make a profit of 60 dollars per necklace you sell and 30 dollars per ring you sell, and want to figure out how many necklaces and rings to produce to maximize your profits.

(a) Formulate this problem as a linear programming problem and find the solution (state the cost function, linear constraints, and all vertices except for the origin).

(b) Suppose instead that the profit per necklace is \( C \) dollars and the profit per ring remains at 30 dollars. For each vertex you listed in the previous part, give the range of \( C \) values for which that vertex is the optimal solution.

Solution:

(a) \( x = \) number of necklaces
\( y = \) number of engagement rings

Maximize: \( 60x + 30y \)

Linear Constraints:
\( 4x + y \leq 80 \)
\( 2x + 3y \leq 90 \)
\( x \geq 0 \)
\( y \geq 0 \)

Drawing a picture in 2 dimensions, we see that the vertices are \((x = 20, y = 0), (x = 15, y = 20), (x = 0, y = 30)\), and the objective is maximized at \((x = 15, y = 20)\), where \( 60x + 30y = 1500 \).

(b) There are lots of ways to solve this part. The most straightforward is to write and solve a system of inequalities checking when the objective of one vertex is at least as large as the objective of the other vertices. For example, for \((x = 15, y = 20)\) the system of inequalities would be \( C \cdot 15 + 30 \cdot 20 \geq C \cdot 20 \) and \( C \cdot 15 + 30 \cdot 20 \geq 30 \cdot 30 \). Doing this for each vertex gives the following solution:
\( (x = 0, y = 30) : C \leq 20 \)
\( (x = 15, y = 20) : 20 \leq C \leq 120 \)
\( (x = 20, y = 0) : 120 \leq C \)
One should note that there is a nice geometric interpretation for this solution: Looking at the graph of the feasible region, as \( C \) increases, the vector \((C, 30)\) starts pointing closer to the \(x\)-axis. The objective says to find the point furthest in the direction of this vector, so the optimal solution also moves closer to the \(x\)-axis as \( C \) increases. When \( C = 20 \) or \( C = 120 \), the vector \((C, 30)\) is perpendicular to one of the constraints, and there are multiple optimal solutions all lying on that constraint, which are all equally far in the direction \((C, 30)\).

6 Modeling: Tricks of the Trade

One of the most important problems in the field of statistics is the linear regression problem. Roughly speaking, this problem involves fitting a straight line to statistical data represented by points \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) on a graph. Denoting the line by \( y = a + bx \), the objective is to choose the constants \( a \) and \( b \) to provide the “best” fit according to some criterion. The criterion usually used is the method of least squares, but there are other interesting criteria where linear programming can be used to solve for the optimal values of \( a \) and \( b \). For each of the following criteria, formulate the linear programming model for this problem.

1. Minimize the sum of the absolute deviations of the data from the line; that is,

\[
\min \sum_{i=1}^{n} |y_i - (a + bx_i)|
\]

2. Minimize the maximum absolute deviation of the data from the line; that is,

\[
\min \max_{i=1 \ldots n} |y_i - (a + bx_i)|
\]

Solution:

(i) Given \( n \) data points \((x_i, y_i)\) for \( i = 1, 2, \ldots, n \), as well as variables \( a \) and \( b \), define new variables \( z_i = y_i - (a + bx_i) \) to denote the deviation of the \( i^{th} \) data point from the line \( y = a + bx \). We know that any number, positive or negative, can be represented by the difference of two positive numbers, so that \( z_i = z_i^+ - z_i^- \) where \( z_i^+ \geq 0 \) and \( z_i^- \geq 0 \) (so we are also introducing new variables \( z_i^+ \) and \( z_i^- \)).

Observe that minimizing \( |z_i| \) (which is non-linear) is equivalent to minimizing \( z_i^+ + z_i^- \) (which is linear) subject to the above three constraints.

Why is this the case? Let’s try a small example. We could represent the number 4 as 4 - 0, 5 - 1, 6 - 2, 100 - 96, etc. But notice that the pair that gives us the smallest sum is 4 and 0. And in general for \( z_i \), the \( z_i^+ \), \( z_i^- \) pair that gives us the smallest sum will always be either \( z_i^+ = z_i \) and \( z_i^- = 0 \) for positive \( z_i \), or \( z_i^+ = 0 \) and \( z_i^- = z_i \) for negative \( z_i \).

Hence

\[
\min \sum_{i=1}^{n} |y_i - (a + bx_i)|
\]
is equivalent to

\[
\text{Minimize } \sum_{i=1}^{n} z_i^+ + z_i^-
\]

subject to

\[
\begin{align*}
& \frac{y_i}{(a + bx_i)} = z_i^+ - z_i^- & \text{for } 1 \leq i \leq n \\
& z_i^+ \geq 0 & \text{for } 1 \leq i \leq n \\
& z_i^- \geq 0 & \text{for } 1 \leq i \leq n
\end{align*}
\]

(ii) By the same reasoning, we introduce variables \( z_i^+ - z_i^- = y_i - (a + bx_i) \), for \( z_i^+ \geq 0 \) and \( z_i^- \geq 0 \). Observe that minimizing \( |z_i| \) (which is non-linear) is equivalent to minimizing \( \max\{z_i^+, z_i^-\} \) (still non-linear) subject to the above constraints, which is equivalent to minimizing \( t \) such that \( t \geq z_i^+ \) and \( t \geq z_i^- \) (which is linear). In simpler words, the smallest possible upper bound on \( z_i^+ \) and \( z_i^- \) will always be precisely equal to the larger of the two. Hence

\[
\min \max_{i=1 \ldots n} |y_i - (a + bx_i)|
\]

is equivalent to

\[
\text{Minimize } t
\]

subject to

\[
\begin{align*}
& y_i - (a + bx_i) = z_i^+ - z_i^- & \text{for } 1 \leq i \leq n \\
& z_i^+ \geq 0 & \text{for } 1 \leq i \leq n \\
& z_i^- \geq 0 & \text{for } 1 \leq i \leq n \\
& z_i^+ \leq t & \text{for } 1 \leq i \leq n \\
& z_i^- \leq t & \text{for } 1 \leq i \leq n
\end{align*}
\]

Remark: The following is an alternative solution to (i):

\[
\text{Minimize } \sum_{i=1}^{n} t_i
\]

subject to

\[
\begin{align*}
& y_i - (a + bx_i) \leq t_i & \text{for } 1 \leq i \leq n \\
& y_i - (a + bx_i) \geq -t_i & \text{for } 1 \leq i \leq n
\end{align*}
\]

The following is an alternative solution to (ii):

\[
\text{Minimize } t
\]

subject to

\[
\begin{align*}
& y_i - (a + bx_i) \leq t & \text{for } 1 \leq i \leq n \\
& y_i - (a + bx_i) \geq -t & \text{for } 1 \leq i \leq n
\end{align*}
\]