Olympiad Combinatorics

Pranav A. Sriram

August 2014

Copyright notices

About the Author

Pranav Sriram graduated from high school at The International School Bangalore, India, and will be a Freshman at Stanford University this Fall.

1. Algorithms

Introduction

Put simply, an **algorithm** is a procedure or set of rules designed to accomplish some task. Mathematical algorithms are indispensable tools, and assist in financial risk minimization, traffic flow optimization, flight scheduling, automatic facial recognition, Google search, and several other services that impact our daily lives.

Often, an algorithm can give us a deeper understanding of mathematics itself. For instance, the famous Euclidean algorithm essentially lays the foundation for the field of number theory. In this chapter, we will focus on using algorithms to prove combinatorial results. We can often prove the existence of an object (say, a graph with certain properties or a family of sets satisfying certain conditions) by giving a procedure to explicitly construct it. These proofs are hence known as constructive proofs. Our main goals in this chapter will be to study techniques for designing algorithms for constructive proofs, and proving that they actually work.

In this chapter, and throughout the book, the emphasis will be on *ideas*. What can we *observe* while solving a given problem? How can disparate ideas and observations be pieced together cohesively to motivate a solution? What can we learn from the solution of one problem, and how may we apply it to others in the future? Each problem in this book is intended to teach some lesson - this may be a combinatorial trick or a new way of looking at problems. We suggest that you keep a log of new ideas and insights into combinatorial structures and problems that you encounter or come up with yourself.

Greedy Algorithms

Be fearful when others are greedy and greedy when others are fearful - Warren Buffet

Greedy algorithms are algorithms that make the best possible short term choices, hence in each step maximizing short term gain. They aren't always the optimal algorithm in the long run, but often are still extremely useful. The idea of looking at extreme elements (that are biggest, smallest, best, or worst in some respect) is central to this approach.

Example 1
In a graph $\boldsymbol{G}$ with n vertices, no vertex has degree greater than Δ. Show that one can color the vertices using at most Δ+1 colors, such that no two neighboring vertices are the same color.

Answer:
We use the following greedy algorithm: arrange the vertices in an arbitrary order. Let the colors be 1, 2, 3… Color the first vertex with color 1. Then in each stage, take the next vertex in the order and color it with the smallest color that has not yet been used on any of its neighbors. Clearly this algorithm ensures that two

adjacent vertices won't be the same color. It also ensures that at most Δ+1 colors are used: each vertex has at most Δ neighbors, so when coloring a particular vertex v, at most Δ colors have been used by its neighbors, so at least one color in the set {1, 2, 3, ..., Δ+1} has **not** been used. The minimum such color will be used for the vertex v. Hence all vertices are colored using colors in the set {1, 2, 3,..., Δ+1} and the problem is solved. ■

Remark: The "greedy" step here lies in always choosing the color with the smallest number. Intuitively, we're saving larger numbers only for when we really need them.

Example 2 [Russia 2005, Indian TST 2012, France 2006]
In a 2 x n array we have positive reals such that the sum of the numbers in each of the n columns is 1. Show that we can select one number in each column such that the sum of the selected numbers in each row is at most $(n+1)/4$.

0.4	0.7	0.9	0.2	0.6	0.4	0.3	0.1
0.6	0.3	0.1	0.8	0.4	0.6	0.7	0.9

Figure 1.1: 2xn array of positive reals, n=8

Answer:
A very trivial greedy algorithm would be to select the smaller number in each column. Unfortunately, this won't always work, as can easily be seen from an instance in which all numbers in the top row are 0.4. So we need to be more clever. Let the numbers in the top row in **non-decreasing order** be a_1, a_2, ..., a_n and the corresponding numbers in the bottom row be b_1, b_2, ..., b_n (in non-increasing order, since $b_i = 1 - a_i$). Further suppose that the sum of the numbers in the top row is less than or equal to that of the bottom row. The idea of ordering the variables is frequently used, since it provides some structure for us to work with.

Our algorithm is as follows: Starting from a_1, keep choosing the smallest remaining element in the top row as long as possible. In other words, select $a_1, a_2, \ldots, a_k$ such that $a_1 + a_2 + \ldots + a_k \leq \frac{n+1}{4}$ but $a_1 + a_2 + \ldots + a_k + a_{k+1} > \frac{n+1}{4}$. Now we cannot select any more from the top row (as we would then violate the problem's condition) so in the remaining columns choose elements from the bottom row. We just need to prove that the sum of the chosen elements in the bottom row is at most $\frac{n+1}{4}$. Note that a_{k+1} is at least the average of $a_1, a_2, \ldots, a_k, a_{k+1}$ which is more than $\frac{n+1}{4(k+1)}$.

Hence $b_{k+1} = (1 - a_{k+1}) < 1 - \frac{n+1}{4(k+1)}$. But b_{k+1} is the largest of the chosen elements in the bottom row. So the sum of the chosen elements in the bottom row cannot exceed $(1 - \frac{n+1}{4(k+1)}) \times (n-k)$. We leave it to the reader to check that this quantity cannot exceed $(n+1)/4$. ■

Remark: One of the perks of writing a book is that I can leave boring calculations to my readers.

Example 3

In a graph $\boldsymbol{G}$ with V vertices and E edges, show that there exists an induced subgraph $\boldsymbol{H}$ with each vertex having degree at least E/V. (In other words, a graph with average degree d has an induced subgraph with minimum degree at least $d/2$).

Answer:

Note that the average degree of a vertex is $2E/V$. Intuitively, we should get rid of 'bad' vertices: vertices that have degree $< E/V$. Thus a natural algorithm for finding such a subgraph is as follows: start with the graph $\boldsymbol{G}$, and as long as there exists a vertex with degree $< E/V$, delete it. However, remember that while deleting a vertex we are also deleting the edges incident to it, and in the process vertices that were initially not 'bad' may become bad in the subgraph formed. What if we end up with a graph with all vertices bad? Fortunately, this won't happen: notice that the *ratio* *of edges/vertices is strictly increasing* (it started at E/V and each time we deleted a vertex, less than E/V edges were deleted by the condition of our algorithm). Hence, it is impossible to reach a stage when only 1 vertex is remaining, since in this case the edges/vertices ratio is 0. So at some point, our *algorithm must terminate*, leaving us with a graph with more than one vertex, all of whose vertices have degree at least E/V. ■

Remark: This proof used the idea of monovariants, which we will explore further in the next section.

The next problem initially appears to have nothing to do with algorithms, but visualizing what it actually means allows us to think about it algorithmically. The heuristics we develop lead us to a very simple algorithm, and proving that it works isn't hard either.

Example 4 [IMO shortlist 2001, C4]
A set of three nonnegative integers $\{x, y, z\}$ with $x < y < z$ satisfying $\{z-y, y-x\} = \{1776, 2001\}$ is called a *historic* set. Show that the set of all nonnegative integers can be written as a disjoint union of historic sets.

***Remark**:* The problem is still true if we replace {1776, 2001} with an arbitrary pair of distinct positive integers $\{a, b\}$. These numbers were chosen since IMO 2001 took place in USA, which won independence in the year 1776.

Answer:
Let $1776 = a$, $2001 = b$. A historic set is of the form $\{x, x+a, x+a+b\}$ or $\{x, x+b, x+a+b\}$. Call these *small sets* and *big sets* respectively. Essentially, we want to cover the set of nonnegative integers using historic sets. To construct such a covering, we visualize the problem as follows: let the set of nonnegative integers be written in a line. In each move, we choose a historic set and cover these numbers on the line. Every number must be covered at the end of our infinite process, but no number can be covered twice (the

historic sets must be disjoint). We have the following *heuristics*, or intuitive guidelines our algorithm should follow:

Heuristic 1: At any point, the smallest number not yet covered is the most "unsafe"- it may get trapped if we do not cover it (for example, if x is the smallest number not yet covered but $x+a+b$ has been covered, we can never delete x). Thus in each move we should choose x as the smallest uncovered number.

Heuristic 2: From heuristic 1, it follows that our algorithm should prefer small numbers to big numbers. Thus it should prefer small sets to big sets.

Based on these two simple heuristics, we construct the following greedy algorithm that minimizes short run risk: in any move, choose x to be the **smallest number not yet covered**. Use the small set if possible; only otherwise use the big set. We now show that this simple algorithm indeed works:

Suppose the algorithm fails (that is, we are stuck because using either the small or big set would cover a number that has already been covered) in the (n+1)th step. Let x_i be the value chosen for x in step i. Before the (n+1)th step, x_{n+1} hasn't yet been covered, by the way it is defined. $x_{n+1} + a + b$ hasn't yet been covered since it is larger than all the covered elements ($x_{n+1} > x_i$ by our algorithm). So the problem must arise due to $x_{n+1} + a$ and $x_{n+1} + b$. Both of these numbers must already be covered. Further, $x_{n+1} + b$ must have been the largest number in its set. Thus the smallest number in this set would be $x_{n+1} + b - (a+b) = x_{n+1} - a$. But at this stage, x_{n+1} was not yet covered, so the small set should have been used and x_{n+1} should have been covered in that step. This is a contradiction. Thus our supposition is wrong and the algorithm indeed works. ■

Remark: In an official solution to this problem, the heuristics would be skipped. Reading such a solution would leave you thinking "Well that's nice and everything, but how on earth would anyone come up with that?" One of the purposes of this book is to show that Olympiad solutions don't just "come out of nowhere". By including heuristics and observations in our solutions, we hope that readers will see the motivation and the key ideas behind them.

Invariants and Monovariants

Now we move on to two more extremely important concepts: invariants and monovariants. Recall that a monovariant is a quantity that changes monotonically (either it is non-increasing or non-decreasing), and an invariant is a quantity that doesn't change. These concepts are especially useful when studying combinatorial processes. While constructing algorithms, they help us in several ways. Monovariants often help us answer the question "Well, what do we do now?" In the next few examples, invariants and monovariants play a crucial role in both constructing the algorithm and ensuring that it works.

Example 5 [IMO shortlist 1989]
A natural number is written in each square of an m x n chessboard. The allowed move is to add an integer k to each of two adjacent numbers in such a way that nonnegative numbers are obtained (two squares are adjacent if they share a common side). Find a necessary and sufficient condition for it to be possible for all the numbers to be zero after finitely many operations.

Answer:
Note that in each move, we are adding the same number to 2 squares, one of which is white and one of which is black (if the chessboard is colored alternately black and white). If S_b and S_w denote the sum of numbers on black and white squares respectively, then $S_b - S_w$ is an invariant. Thus if all numbers are 0 at the end, $S_b - S_w = 0$ at the end and hence $S_b - S_w = 0$ in the

beginning as well. This condition is thus necessary; now we prove that it is sufficient.

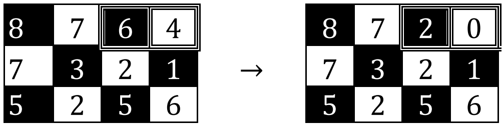

Figure 1.2: A move on the mxn board

Suppose a, b, c are numbers in cells A, B, C respectively, where A, B, C are cells such that A and C are both adjacent to B. If $a \leq b$, we can add $(-a)$ to both a and b, making a 0. If $a \geq b$, then add $(a-b)$ to b and c. Then b becomes a, and now we can add $(-a)$ to both of them, making them 0. Thus we have an algorithm for reducing a positive integer to 0. Apply this in each row, making all but the last 2 entries 0. Now all columns have only zeroes except the last two. Now apply the algorithm starting from the top of these columns, until only two adjacent nonzero numbers remain. These last two numbers must be equal since $S_b = S_w$. Thus we can reduce them to 0 as well. ■

The solution to the next example looks long and complicated, but it is actually quite intuitive and natural. We have tried to motivate each step, and show that each idea follows quite naturally from the previous ones.

Example 6 [New Zealand IMO Training, 2011]
There are $2n$ people seated around a circular table, and m cookies are distributed among them. The cookies can be passed under the following rules:
(a) Each person can only pass cookies to his or her neighbors
(b) Each time someone passes a cookie, he or she must also eat a cookie

Let A be one of these people. Find the least m such that no matter how m cookies are distributed initially, there is a strategy to pass cookies so that A receives at least one cookie.

Answer:
We begin by labeling the people A_{-n+1}, A_{-n+2}, ..., A_0, A_1, A_2, ..., A_n, such that $A = A_0$. Also denote $A_{-n} = A_n$. We assign weight $1/2^{|i|}$ to each cookie held by person A_i. Thus for example, if A_3 passes a cookie to A_2, that cookie's weight increases from $1/8$ to $1/4$. Note that A_3 must also eat a cookie (of weight $1/8$) in this step. Thus we see in this case the sum of the weights of all the cookies has remained the same. More precisely, if A_i has a_i cookies for each i, then the total weight of all cookies is

$$W = \sum_{i=-n+1}^{n} \frac{a_i}{2^{|i|}}$$

Whenever a cookie is passed towards A_0 (from $A_{\pm i}$ to $A_{\pm(i-1)}$ for i positive) one cookie is eaten and another cookie doubles its weight, so the total weight remains invariant. If a cookie is passed away from A, then the total weight decreases. Thus the total weight is indeed a monovariant.

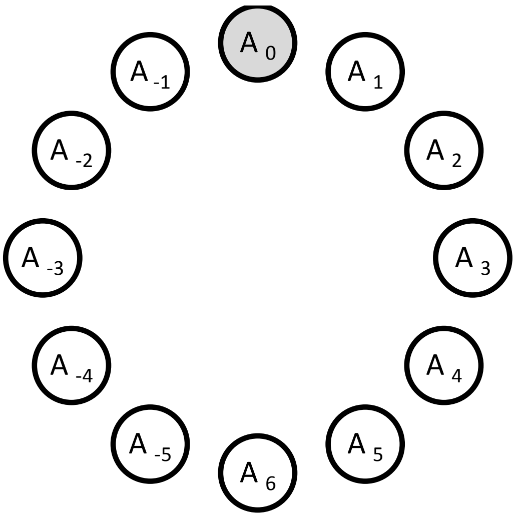

Figure 1.3: Labeling scheme to create a monovariant (n=5)

If $m < 2^n$, then if all the cookies are initially given to A_n, the initial total weight is $m/2^n < 1$. Therefore the total weight is always less than 1 (since it can never increase), so A_0 cannot receive a cookie (if A_0 received a cookie it would have weight 1). Thus we must have $m \geq 2^n$.

We now show that for $m \geq 2^n$, we can always ensure that A_0 gets a cookie. Intuitively, we have the following heuristic:

Our algorithm should never pass away from A_0, otherwise we will decrease our monovariant. Thus in each step we should pass towards A_0.

This heuristic, however, does not tell us which way A_n should pass a cookie, as both directions are towards A_0 (A_n and A_0 are diametrically opposite). This leads us to consider a new quantity in order to distinguish between the two directions that A_n can pass to. Let W_+ be the sum of the weights of cookies held by A_0, A_1, A_2, …., A_n and let W_- be the sum of the weights of cookies held by A_0, A_{-1}, A_{-2}, …., A_{-n}. Assume WLOG $W_+ \geq W_-$. Then this suggests that we should make A_n pass cookies only to A_{n-1} and that we should only work in the semicircle containing nonnegative indices, since this is the semicircle having more weight. Thus our algorithm is to make A_n pass as many cookies as possible to A_{n-1}, then make A_{n-1} pass as many cookies as possible to A_{n-2}, and so on until A_0 gets a cookie. But this works if and only if $W_+ \geq 1$: $W_+ \geq 1$ is certainly necessary since W_+ is a monovariant under our algorithm, and we now show it is sufficient.

Suppose $W_+ \geq 1$. Note that our algorithm leaves W_+ invariant. Suppose our algorithm terminates, that is, we cannot pass anymore cookies from any of the A_i's with i positive, and A_0 doesn't have any cookies. Then A_1, A_2, …., A_n all have at most 1 cookie at the end (if they had more than one, they could eat one and pass one and our algorithm wouldn't have terminated). Then at this point $W_+ \leq ½ + ¼ + ... + 1/2^n < 1$, contradicting the fact that W_+ is invariant and ≥ 1. Thus $W_+ \geq 1$ is a sufficient condition for our algorithm to work.

Finally, we prove that we indeed have $W_+ \geq 1$. We assumed $W_+ \geq W_-$. Now simply note that each cookie contributes at least $1/2^{n-1}$ to the sum $(W_+ + W_-)$, because each cookie has weight at least $1/2^{n-1}$ except for cookies at A_n. However, cookies at A_n are counted twice since they contribute to both W_+ and W_-, so they also contribute $1/2^{n-1}$ to the sum. Hence, since we have at least 2^n cookies, $W_+ + W_- \geq 2$, so $W_+ \geq 1$ and we are done. ■

The next example demonstrates three very useful ideas: monovariants, binary representation and the Euclidean algorithm. All of these are very helpful tools.

Example 7 [IMO shortlist 1994, C3]
Peter has 3 accounts in a bank, each with an integral number of dollars. He is only allowed to transfer money from one account to another so that the amount of money in the latter is doubled. Prove that Peter can always transfer all his money into two accounts. Can he always transfer all his money into one account?

Answer:
The second part of the question is trivial - if the total number of dollars is odd, it is clearly not always possible to get all the money into one account. Now we solve the first part. Let A, B, C with $A \leq B \leq C$ be the number of dollars in the account 1, account 2 and account 3 respectively at a particular point of time. If $A = 0$ initially, we are done so assume $A > 0$. As we perform any algorithm, the values of A, B and C keep changing. Our aim is to monotonically strictly decrease the value of *min* (A, B, C). This will ensure that we eventually end up with *min* $(A, B, C) = 0$ and we will be done. Now, we know a very simple and useful algorithm that monotonically reduces a number- the Euclidean algorithm. So let $B = qA + r$ with $0 \leq r < A$. Our aim now is to reduce the number

of dollars in the second account from B to r. Since $r < A$, we would have reduced min (A, B, C), which was our aim.

Now, since the question involves doubling certain numbers, it is a good idea to consider binary representations of numbers. Let $q = m_0 + 2m_1 + ... + 2^k m_k$ be the binary representation of q, where $m_i = 0$ or 1. To reduce B to r, in step i of our algorithm, we transfer money to account 1. The transfer is from account 2 if $m_{i-1} = 1$ and from account 3 if $m_{i-1} = 0$. The number of dollars in the first account starts with A and keeps doubling in each step. Thus we end up transferring $A(m_0 + 2m_1 + ... + 2^k m_k) = Aq$ dollars from account 2 to account 1, and we are left with $B - Aq = r$ dollars in account 2. We have thus succeeded in reducing min (A, B, C) and so we are done. ■

Now we look at a very challenging problem that can be solved using monovariants.

Example 8 [APMO 1997, Problem 5]
n people are seated in a circle. A total of nk coins have been distributed among them, but not necessarily equally. A *move* is the transfer of a single coin between two adjacent people. Find an algorithm for making the minimum possible number of moves which result in everyone ending up with the same number of coins.

Answer:
We want each person to end up with k coins. Let the people be labeled from 1, 2, ..., n in order (note that n is next to 1 since they are sitting in a circle). Suppose person i has c_i coins. We introduce the variable $d_i = c_i - k$, since this indicates how close a person is to having the desired number of coins. Consider the quantity

$$X = |d_1| + |d_1 + d_2| + |d_1 + d_2 + d_3| + ... + |d_1 + d_2 + ... + d_{n-1}|$$

Clearly $X = 0$ if and only if everyone has k coins, so our goal is to make $X = 0$. The reason for this choice of X is that moving a coin between person j and person $j + 1$ for $1 \leq j \leq n - 1$ changes X by exactly 1 as only the term $|d_1 + d_2 + \ldots + d_j|$ will be affected. Hence X is a monovariant and is fairly easy to control (except when moving a coin from 1 to n or vice versa). Let $s_j = d_1 + d_2 + \ldots + d_j$.

We claim that as long as $X > 0$ it is always possible to reduce X by 1 by a move between j and $j + 1$ for some $1 \leq j \leq n - 1$. We use the following algorithm. Assume WLOG $d_1 \geq 1$. Take the first j such that $d_{j+1} < 0$. If $s_j > 0$, then simply make a transfer from j to $j + 1$. This reduces X by one since it reduces the term $|s_j|$ by one. The other possibility is $s_j = 0$, which means $d_1 = d_2 = \ldots = d_j = 0$ (recall that d_{j+1} is the first negative term). In this case, take the first $m > i+1$ such that $d_m \geq 0$. Then $d_{m-1} < 0$ by the assumption on m, so we move a coin from m to $(m-1)$. Note that all terms before d_m were either 0 or less than 0 and $d_{m-1} < 0$, so s_{m-1} was less than 0. Our move has increased s_{m-1} by one, and has hence decreased $|s_{m-1}|$ by one, so we have decreased X by one.

Thus at any stage we can always decrease X by at least one by moving between j and $j + 1$ for some $1 \leq j \leq n - 1$. We have not yet considered the effect of a move between 1 and n. Thus our full algorithm is as follows: At any point of time, if we can decrease X by moving a coin from 1 to n or n to 1, do this. Otherwise, decrease X by 1 by the algorithm described in the above paragraph. ■

Sometimes while creating algorithms that monotonically decrease (or increase) a quantity, we run into trouble in particular cases and our algorithm doesn't work. We can often get around these difficulties as follows. Suppose we want to monotonically decrease a particular quantity. Call a position *good* if we can decrease the monovariant with our algorithm. Otherwise, call the position *bad*. Now create an algorithm that converts bad positions into good positions, without increasing our monovariant. We use the first algorithm when possible, and then if we are stuck in a bad position, use the second algorithm to get back to a good position. Then we can again use the first algorithm. The next example (which is quite hard) demonstrates this idea.

Example 9 [USAMO 2003-6]
At the vertices of a regular hexagon are written 6 nonnegative integers whose sum is 2003. Bert is allowed to make moves of the following form: he may pick a vertex and replace the number written there by the absolute value of the difference between the numbers at the neighboring vertices. Prove that Bert can make a sequence of moves, after which the number 0 appears at all 6 vertices.

Remark: We advise the reader to follow this solution with a paper and pen, and fill in the details that have been left for the reader. We first suggest that the reader try some small cases (with 2003 replaced by smaller numbers).

Answer:
Our algorithm uses the fact that 2003 is odd. Let the sum of a position be the sum of the 6 numbers and the maximum denote the value of the maximum of the 6 numbers. Let A, B, C, D, E, F be the numbers at the 6 vertices in that order. Our aim is to monotonically decrease the maximum. Note that the maximum can never increase.

We need two sub-algorithms:

(i) "Good position" creation: from a position with odd sum, go to a position with exactly one odd number
(ii) Monovariant reduction: from a position with exactly one odd number, go to a position with odd sum and strictly smaller maximum, or go to the all 0 position.

For (i), since (A + B + C + D + E + F) is odd, assume WLOG that A + C + E is odd. If exactly one of A, C, E is odd, suppose A is odd. Then make the following sequence of moves: B, F, D A, F (here we denote a move by the vertex at which the move is made). This way, we end up with a situation in which only B is odd and the rest become even (check this), and we are done with step (i). The other possibility is that all of A, C and E are odd. In this case make the sequence of moves (B, D, F, C, E). After this only A is odd (check this).

Now we are ready to apply step (ii), the step that actually decreases our monovariant. At this point, only one vertex contains an odd number; call this vertex A. Again we take two cases. If the maximum is even, then it is one of B, C, D, E or F. Now make moves at B, C, D, E and F in this order. (The reader should check that this works, that is, this sequence of moves decreases the maximum and ensures that the sum is odd). If the maximum is odd, then it is A. If C = E = 0, then the sequence of moves (B, F, D, A, B, F) leaves us with all numbers 0 and we are done. Otherwise, suppose at least one of C and E is nonzero so suppose C > 0 (the case E > 0 is similar). In this case, make the moves (B, F, A, F). The reader can check that this decreases the maximum and leaves us with odd sum.

Thus starting with odd sum, we apply (i) if needed, after which we apply (ii). This decreases the maximum, and also leaves us again with odd sum (or in some cases it leaves us with all 0s and we are done), so we can repeat the entire procedure until the maximum eventually becomes 0. ■

Miscellaneous Examples

Now we look at a few more problems involving moves that don't directly use monovariants or greedy algorithms. These problems can often be solved by algorithms that build up the required configuration in steps. Sometimes, the required algorithm becomes easier to find after making some crucial observations or proving an auxiliary lemma. But in lots of cases, all a combinatorics problem needs is patience and straightforward logic, as the next example shows. Here again the solution looks long but most of what is written is just intended to motivate the solution.

Example 10 [China 2010, Problem 5]
There are some (finite number of) cards placed at the points A_1, A_2, …, A_n and O, where $n \geq 3$. We can perform one of the following operations in each step:

(1) If there are more than 2 cards at some point A_i, we can remove 3 cards from this point and place one each at A_{i-1}, A_{i+1} and O (here $A_0 = A_n$ and $A_{n+1} = A_1$)
(2) If there are at least n cards at O, we can remove n cards from O and place one each at A_1, A_2, …, A_n.

Show that if the total number of cards is at least n^2+3n+1, we can make the number of cards at each vertex at least $n + 1$ after finitely many steps.

Answer:
Note that the total number of cards stays the same. We make a few observations:

(a) We should aim to make the number of cards at each A_i equal or close to equal, since if in the end some point has lots of cards, some other point won't have enough.

(b) We can make each of the A_i's have 0, 1 or 2 cards.
Proof: repeatedly apply operation (1) as long as there is a point with at least 3 cards. This process must terminate, since the number of coins in O increases in each step but cannot increase indefinitely. This is a good idea since the A_i's would now have a 'close to equal' number of coins, which is a good thing by observation a).

(c) From observation b), we see that it is also possible to make each of the A_i's have 1, 2, or 3 cards (from the stage where each vertex has 0, 1 or 2 cards, just apply operation (2) once). This still preserves the 'close to equal' property, but gives us some more flexibility since we are now able to apply operation 1.

(d) Based on observation c), we make each of the A_i's have 1, 2 or 3 cards. Suppose x of the A_i's have 1 card, y of the A_i's have 2 cards and z of the A_i's have 3 cards. The number of cards at O is then at least $(n^2+3n+1) - (x + 2y + 3z)$. Since $x + y + z = n$, $(x + 2y + 3z) = (2x + 2y + 2z) + z - x = 2n + z - x \leq 2n$ if $x \geq z$. Thus if $x \geq z$, O will have at least $(n^2+3n+1) - 2n = n^2+n + 1$ cards. Now we can apply operation (2) n times. Then all the A_i's will now have at least $n + 1$ cards (they already each had at least 1 card), and O will have at least $n^2 + n + 1 - n^2 = n + 1$ cards and we will be done.

Thus, based on observation d), it suffices to find an algorithm that starts with a position in which each of the A_i's have 1, 2, or 3 cards and ends in a position in which each of the A_i's have 1, 2, or 3 cards but the number of points having 3 cards is not more than the number of points having 1 card. This is not very difficult- the basic idea is to ensure that between any two points having 3 cards, there is a point containing 1 card. We can do this as follows:

If there are consecutive 3's in a chain, like $(x, 3, 3, \ldots, 3, y)$ with $(x, y \neq 3)$, apply operation (1) on all the points with 3 cards to get $(x + 1, 1, 2, 2, \ldots, 2, 1, y+1)$. Thus we can ensure that there are no adjacent 3's. Now suppose there are two 3's with only 2's between them, like $(x, 3, 2, 2, 2, \ldots, 2, 3, y)$ with $x, y \neq 3$. After doing operation (1) first on the first 3, then on the point adjacent to it that has become a 3 and so on until the point before y, we get the sequence $(x+1, 1, 1, \ldots, 1, y+1)$.

Thus we can repeat this procedure as long as there exist two 3's that do not have a 1 between them. Note that the procedure preserves the property that all A_i's have 1, 2 or 3 cards. But this cannot go on indefinitely since the number of coins at 0 is increasing. So eventually we end up with a situation where there is at least one 1 between any two 3's, and we are done. ■

Example 11 [IMO 2010, Problem 5]
Six boxes B_1, B_2, B_3, B_4, B_5, B_6 of coins are placed in a row. Each box initially contains exactly one coin. There are two types of allowed moves:

Move 1: If B_k with $1 \leq k \leq 5$ contains at least one coin, you may remove one coin from B_k and add two coins to B_{k+1}.
Move 2: If B_k with $1 \leq k \leq 4$ contains at least one coin, you may remove one coin from B_k and exchange the contents (possibly empty) of boxes B_{k+1} and B_{k+2}.

Determine if there exists a finite sequence of moves of the allowed types, such that the five boxes B_1, B_2, B_3, B_4, B_5 become empty, while box B_6 contains exactly $2010^{2010^{2010}}$ coins.
Note: $a^{b^c} = a^{(b^c)}$

Answer:

Surprisingly, the answer is yes. Let $A = 2010^{2010^{2010}}$. We denote by $(a_1, a_2, \ldots, a_n) \rightarrow (a_1', a_2', \ldots, a_n')$ the following: if some consecutive boxes have $a_1, a_2, \ldots, a_n$ coins respectively, we can make them have $a_1', a_2', \ldots, a_n'$ coins by a legal sequence of moves, with all other boxes unchanged.

Observations:
a) Suppose we reach a stage where all boxes are empty, except for B_4, which contains at least $A/4$ coins. Then we can apply move 2 if necessary until B_4 contains exactly $A/4$ coins, and then apply move 1 twice and we will be done. Thus reaching this stage will be our key goal.

b) Move 1 is our only way of increasing the number of coins. Since it involves doubling, we should look for ways of generating powers of 2. In fact, since A is so large, we should try to generate *towers* of 2's (numbers of the form 2^{2^2}).

Based on this, we construct two sub algorithms.

Algorithm 1: $(a, 0, 0) \rightarrow (0, 2^a, 0)$ for any positive integer a.
Proof: First use move 1: $(a, 0, 0) \rightarrow (a-1, 2, 0)$.
Now use move 1 on the middle box till it is empty: $(a-1, 2, 0) \rightarrow (a-1, 0, 4)$
Use move 2 on the first box to get $(a\text{-}2, 4, 0)$.
Repeating this procedure (that is, alternately use move one on the second box till it is empty, followed by move one on the first box and so on), we eventually get $(0, 2^a, 0)$.
Now, using this algorithm, we can construct an even more powerful algorithm that generates a large number of coins.

Algorithm 2: Let P_n be a tower of n 2's for each positive integer n (eg. $P_3 = 2^{2^2} = 16$). Then
$(a, 0, 0, 0) \rightarrow (0, P_a, 0, 0)$.
Proof: We use algorithm 1. As in algorithm 1, the construction is stepwise. It is convenient to explain it using induction.

We prove that $(a, 0, 0, 0) \rightarrow (a\text{-}k, P_k, 0, 0)$ for each $1 \le k \le a$. For $k = 1$, simply apply move 1 to the first box. Suppose we have reached the stage $(a\text{-}k, P_k, 0, 0)$. We want to reach $(a\text{-}\ (k+1), P_{k+1}, 0, 0)$. To do this, **apply algorithm 1** to get $(a\text{-}k, 0, 2^{P_k}, 0)$. Note that $2^{P_k} = P_{k+1}$. So now just apply move 2 to the first box and we get $(a\text{-}k\text{-}1, P_{k+1}, 0, 0)$. Thus by induction, we finally reach (for $k = a$) $(0, P_a, 0, 0)$.

With algorithm 2 and observation a), we are ready to solve the problem.

First apply move 1 to box 5, then move 2 to box 4, 3, 2 and 1 in this order:
(1, 1, 1, 1, 1, 1) → (1, 1, 1, 1, 0, 3) → (1, 1, 1, 0, 3, 0) → (1, 1, 0, 3, 0, 0) →(1, 0, 3, 0, 0, 0) → (0, 3, 0, 0, 0, 0).

Now we use algorithm 2 twice:
(0, 3, 0, 0, 0, 0) → (0, 0, P_3, 0, 0, 0) → (0, 0, 0, P_{16}, 0, 0).

Now we leave it to the reader to check that $P_{16} > A/4$ (in fact P_{16} is much larger than A). By observation a), we are done.

Remark: In the contest, several contestants thought the answer was no, and spent most of their time trying to prove that no such sequence exists. Make sure that you don't ever jump to conclusions like that too quickly. On a lighter note, in a conference of the team leaders and deputy leaders after the contest, one deputy leader remarked "Even most of us thought that no such sequence existed". To this, one leader replied, "That's why you are deputy leaders and not team leaders!"

We close this chapter with one of the hardest questions ever asked at the IMO. Only 2 out of over 500 contestants completely solved problem 3 in IMO 2007. Yup, that's right- 2 high school students in the entire world.

Example 12 [IMO 2007, Problem 3]
In a mathematical competition some competitors are friends; friendship is always mutual. Call a group of competitors a clique if each two of them are friends. The number of members in a clique is called its size. It is known that the size of the largest clique(s) is even. Prove that the competitors can be arranged in two rooms such that the size of the largest cliques in one room is the same as the size of the largest cliques in the other room.

Answer:
Let $\boldsymbol{M}$ be one of the cliques of largest size, $|\boldsymbol{M}| = 2m$. First send all members of $\boldsymbol{M}$ to Room A and all other people to Room B. Let $c(\mathrm{A})$

and c(B) denote the sizes of the largest cliques in rooms A and B at a given point in time. Since $\boldsymbol{M}$ is a clique of the largest size, we initially have $c(A) = |\boldsymbol{M}| \geq c(B)$. Now we want to "balance things out". As long as $c(A) > c(B)$, send one person from Room A to Room B. In each step, c(A) decreases by one and c(B) increases by at most one. So at the end we have $c(A) \leq c(B) \leq c(A) + 1$. We also have $c(A) = |A| \geq m$ at the end. Otherwise we would have at least m+1 members of $\boldsymbol{M}$ in Room B and at most m−1 in Room A, implying $c(B) - c(A) \geq (m+1) - (m-1) = 2$.

Clearly if $c(A) = c(B)$ we are done so at this stage the only case we need to consider is $c(B) - c(A) = 1$. Let $c(A) = k$, $c(B) = k+1$. Now if there is a competitor in B, who is also in $\boldsymbol{M}$ but is not in the biggest clique in B, then by sending her to A, c(B) doesn't change but c(A) increases by 1 and we are done. Now suppose there is no such competitor. We do the following: take each clique of size k+1 in B and send one competitor to A. At the end of this process, $c(B) = k$. Now we leave it to the reader to finish the proof by showing that c(A) is still k. (You will need to use the supposition that there is no competitor in B who is also in $\boldsymbol{M}$ but not in the biggest clique of B. This means that every clique in B of size (k+1) contains B∩M). ■

Exercises

1. **[Activity Selection Problem]**
   On a particular day, there are n events (say, movies, classes, parties, etc.) you want to attend. Call the events E_1, E_2, …, E_n and let E_i start at time s_i and finish at time f_i. You are only allowed to attend events that do not overlap (that is, one should finish before the other starts). Provide an efficient algorithm that selects as many events as possible while

satisfying this condition.
(Note: We have not defined what “efficient” here means. Note that this problem can be solved by simply testing all $2n$ possible combinations of events, and taking the best combination that works. However, this uses a number of steps that is exponential in n. By efficient, we mean a procedure that is guaranteed to require at most a number of steps that is polynomial in n).

2. **[Weighted Activity Selection]**
   Solve the following generalization of the previous problem: event E_i has now weight w_i and the objective is not to maximize the number of activities attended, but the sum of the weights of all activities attended.

3. **[Russia 1961]**
   Real numbers are written in an $m \times n$ table. It is permissible to reverse the signs of all the numbers in any row or column. Prove that after a number of these operations, we can make the sum of the numbers along each line (row or column) nonnegative.

4. Given $2n$ points in the plane with no three collinear, show that it is possible to pair them up in such a way that the n line segments joining paired points do not intersect.

5. **[Czech and Slovak Republics 1997]**
   Each side and diagonal of a regular n-gon ($n \geq 3$) is colored blue or green. A move consists of choosing a vertex and switching the color of each segment incident to that vertex (from blue to green or vice versa). Prove that regardless of the initial coloring, it is possible to make the number of blue segments incident to each vertex even by following a sequence of moves. Also show that the final configuration obtained is uniquely determined by the initial coloring.

6. **[Bulgaria 2001]**
Given a permutation of the numbers 1, 2, …, *n*, one may interchange two consecutive blocks to obtain a new permutation. For instance, 3 5 4 8 9 7 2 1 6 can be transformed to 3 9 7 2 5 4 8 1 6 by swapping the consecutive blocks 5 4 8 and 9 7 2. Find the least number of changes required to change *n*, *n*-1, *n*-2, …, 1 to 1, 2, …, *n*.

7. **[Minimum makespan scheduling]**
Given the times taken to complete *n* jobs, t_1, t_2, …, t_n, and *m* identical machines, the task is to assign each job to a machine so that the total time taken to finish all jobs is minimized. For example, if *n* = 5, *m* = 3 and the times are 5, 4, 4, 6 and 7 hours, the best we can do is make machine 1 do jobs taking 4 and 5 hours, machine 2 do jobs taking 4 and 6 hours, and machine 3 do the job taking 7 hours. The total time will then be 10 hours since machine 2 takes (4 + 6) hours.

Consider the following greedy algorithm: Order the jobs arbitrarily, and in this order assign to each job the machine that has been given the least work so far. Let T_{OPT} be the total time taken by the best possible schedule, and T_A the time taken by our algorithm. Show that $T_A/T_{OPT} \leq 2$; in other words, our algorithm always finds a schedule that takes at most twice the time taken by an optimal schedule. (This is known as a *2-factor approximation algorithm*.)

8. **[USAMO 2011-2]**
An integer is written at each vertex of a regular pentagon. A solitaire game is played as follows: a turn consists of choosing an integer m and two adjacent vertices of the pentagon, and subtracting m from the numbers at these vertices and adding 2*m* to the vertex opposite them. (Note that *m* and the vertices chosen can change from turn to turn). The game is said to be won at a vertex when the number 2011 is written at it and the

other four vertices have the number 0 written at them. Show that there is exactly one vertex at which the game can be won.

9. [Chvatal's set covering algorithm]

Let S_1, S_2, …, S_k be subsets of {1, 2, …, n}. With each set S_i is an associated cost c_i. Given this information, the minimum set cover problem asks us to select certain sets among S_1, …, S_k such that the union of the selected sets is {1, 2, …, n} (that is, each element is covered by some chosen set) and the total cost of the selected sets is minimized. For example, if $n = 4$, $k = 3$, $S_1 = \{1, 2\}$; $S_2 = \{2, 3, 4\}$ and $S_3 = \{1, 3, 4\}$ and the costs of S_1, S_2 and S_3 are 5, 6 and 4 respectively, the best solution would be to select S_1 and S_3.

Consider the following greedy algorithm for set cover: In each stage of the algorithm, we select the subset S_i which maximizes the value of $\frac{|S_i \cap C'|}{c_i}$, where C' denotes the set of elements not yet covered at that point. Intuitively, this algorithm maximizes (additional benefit)/cost in each step. This algorithm does not produce an optimal result, but it gets fairly close: let C_A be the cost of the selected sets produced by the algorithm, and let C_{OPT} be the cost of the best possible selection of sets (the lowest cost). Prove that $C_A/C_{OPT} \le H_n$, where $H_n = 1 + ½ + … + 1/n$. (In other words, this is an H_n-factor approximation algorithm.)

10. A ***matroid*** is an ordered pair (S, F) satisfying the following conditions:

(i) S is a finite set

(ii) F is a nonempty family of subsets of S, such that if A is a set in F, all subsets of A are also in F. The members of F are called ***independent*** sets

(iii) If A and B belong to F but $|A| > |B|$, then there exists an element $x \in B\backslash A$ such that $A \cup \{x\} \in F$.

For example, if S = {1, 2, 3, 4} and F = {∅, {1}, {2}, {3}, {4}, {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3,4}}, then you can easily verify that the above properties are satisfied. In general, note that if F contains all subsets of S with k or fewer elements for some $k \leq |S|$, {S, F} will be a matroid.

An independent set A is said to be ***maximal*** if there does not exist any element x in S such that $A \cup \{x\} \in F$. (In other words, adding any element to A destroys its independence.) Prove that ***all maximal independent sets have the same cardinality***.

11. Consider a matroid {S, F} where $S = \{a_1, \ldots, a_n\}$. Let element a_i have weight w_i, and define the weight of a set A to be the sum of the weights of its elements. A problem central to the theory of greedy algorithms is to find an independent set in this matroid of ***maximum weight***. Consider the following greedy approach: starting from the null set, in each stage of the algorithm add an element (that has not been selected so far) with the highest weight possible while preserving the independence of the set of selected elements. When no more elements can be added, stop.
Show that this greedy algorithm indeed produces a ***maximum weight independent set***.

12. [IMO Shortlist 2013, C3]
A crazy physicist discovered a new kind of particle which he called an imon. Some pairs of imons in the lab can be entangled, and each imon can participate in many entanglement relations. The physicist has found a way to perform the following two kinds of operations with these particles, one operation at a time.
(i) If some imon is entangled with an odd number of other imons in the lab, then the physicist can destroy it.
(ii) At any moment, he may double the whole family of imons in the lab by creating a copy I’ of each imon I. During this procedure, the two copies I’ and J’ become entangled if and

only if the original imons I and J are entangled, and each copy I' becomes entangled with its original imon I; no other entanglements occur or disappear at this moment.

Show that after a finite number of operations, he can ensure that no pair of particles is entangled.

13. **[Japan 1998]**

Let n be a positive integer. At each of $2n$ points around a circle we place a disk with one white side and one black side. We may perform the following move: select a black disk, and flip over its two neighbors. Find all initial configurations from which some sequence of such moves leads to a position where all disks but one are white.

14. **[Based on IOI 2007]**

You are given n integers $a_1, a_2, \ldots, a_n$ and another set of n integers $b_1, b_2, \ldots, b_n$ such that for each i, $b_i \leq a_i$. For each $i = 1, 2, \ldots, n$, you must choose a set of b_i distinct integers from the set $\{1, 2, \ldots, a_i\}$. In total, $(b_1 + b_2 + \ldots + b_n)$ integers are selected, but not all of these are distinct. Suppose k distinct integers have been selected, with multiplicities $c_1, c_2, c_3, \ldots, c_k$. Your ***score*** is defined as $\sum_{i=1}^{k} c_k(c_k - 1)$. Give an efficient algorithm to select numbers in order to ***minimize*** your score.

15. **[Based on Asia Pacific Informatics Olympiad 2007]**

Given a set of n distinct positive real numbers $S = \{a_1, a_2, \ldots, a_n\}$ and an integer $k < n/2$, provide an efficient algorithm to form k pairs of numbers $(b_1, c_1), (b_2, c_2), \ldots, (b_k, c_k)$ such that these $2k$ numbers are all distinct and from S, and such that the sum $\sum_{\mathrm{i}=1}^{\mathrm{k}} |\mathrm{b_i} - \mathrm{c_i}|$ is minimized.
Hint: A natural greedy algorithm is to form pairs sequentially by choosing the closest possible pair in each step. However, this doesn't always work. Analyze where precisely the problem in this approach lies, and then accordingly adapt this algorithm so that it works.

16. **[ELMO Shortlist 2010]**

You are given a deck of kn cards each with a number in {1, 2, …, n} such that there are k cards with each number. First, n piles numbered {1, 2, …, n} of k cards each are dealt out face down. You are allowed to look at the piles and rearrange the k cards in each pile. You now flip over a card from pile 1, place that card face up at the bottom of the pile, then next flip over a card from the pile whose number matches the number on the card just flipped. You repeat this until you reach a pile in which every card has already been flipped and wins if at that point every card has been flipped. Under what initial conditions (distributions of cards into piles) can you guarantee winning this game?

17. **[Russia 2005]**

100 people from 25 countries, four from each country, sit in a circle. Prove that one may partition them onto 4 groups in such way that no two countrymen, nor two neighboring people in the circle, are in the same group.

18. **[Saint Petersburg 1997]**

An *Aztec diamond of rank n* is a figure consisting of those squares of a gridded coordinate plane lying inside the square $|x| + |y| \le n+1$. For any covering of an Aztec diamond by dominoes, a move consists of selecting a 2x2 square covered by two dominoes and rotating it by 90 degrees. The aim is to convert the initial covering into the covering consisting of only horizontal dominoes. Show that this can be done using at most $n(n+1)(2n+1)/6$ moves.