CONSTRUCTION PROBLEMS

VIPUL NAIK

Abstract. In this article, I describe the general problem of constructing configurations subject to certain conditions, and the use of techniques like "greedy algorithms" to construct such configurations

1. The pivotal role of construction problems

1.1. Construction, existence and enumeration. The general situation of combinatorics could be summarized as follows: We are given a general kind of configuration and we are asked to determine whether there are any configurations of that general kind that satisfy certain particular constraints. We could have:

(1) The *construction* problem which asks for an algorithm or method to construct a configuration satisfying the constraints
(2) The *existence* problem which simply asks for an answer to whether a configuration exists
(3) The *enumeration* problem which asks for the total number of configurations

Skill at solving the construction problem helps both with the existence problem and the enumeration problem, as follows:

- If we are able to actually construct *a* configuration, we can in particular prove that it exists.
- If we are able to devise an algorithm that constructs *all* configurations, and we can measure the number of times that algorithm outputs a configuration, then we have counted the number of configurations.

Thus, construction problems in combinatorics, apart from being important in their own right, also help with existential and enumeration problems.

2. Construction by induction

2.1. Construction by induction: the idea. Let C_n be the set of *all* configurations of "size" n. Then, construction by induction uses a listing of the elements of C_{n-1} to obtain a listing of the elements of C_n.

There are two possibilities for this. One is that we *first* pick a configuration of "size" $n-1$ (that is, an element of C_{n-1}) and then make some further choices and decisions to obtain a configuration of size n (that is, an element of C_n). Note that since we could make these choices in multiple ways, we may have more than one configuration of size n corresponding to the configuration of size $n-1$.

The other approach is to start off with a (as yet unspecified) configuration of size n, make some choices and decisions and then reach the situation where we have to choose a configuration of size $n-1$.

Let's look at each of these methods.

2.2. Top-down construction. In a *top-down* construction, we start with trying to construct the configuration of size n, make a few choices, and then reach down to the situation where we have to select a configuration of size $n-1$.

Let's take the example where the "configuration" of "size" n is an arbitrary permutation of the set $\{1, 2, \dots, n\}$. In this context, C_n is the set of all permutations on n letters, while C_{n-1} is the set of all permutations on $n-1$ letters.

A permutation here is described by a "word" with the first letter being where 1 goes, the second letter being where 2 goes, and so on. Thus, the elements of C_n are words of length n (with all letters distinct and drawn from 1 to n) while the elements of C_{n-1} are words of length $n-1$ (with all letters distinct and drawn from 1 to $n-1$).

Then, in order to describe a particular configuration of size n, we start out by specifying the first letter (that is, the image of 1 under the permutation). There are n choices for this. Once we have made

©Vipul Naik, B.Sc. (Hons) Math and C.S., Chennai Mathematical Institute.

this choice, we can strip off the first letter and get a word of length $n-1$ with distinct letters drawn from 1 to n, except the first letter.

The number of ways of choosing such words equals the numbers of ways of choosing words of length $n-1$ with letters drawn from 1 to $n-1$ (by simply relabelling). This corresponds to the cardinality of C_{n-1}, and we get:

$$|C_n| = nC_{n-1}$$

Top-down approaches are typically more *local* in nature since the initial local choices have to be made *before* fixing the configuration of size $n-1$. In other words, when following a top-down approach, we cannot *a priori* know the global impact.

In the case of permutations that is not a problem, because whatever choice we make for the first letter, the behaviour for the remaining $n-1$ letters is qualitatively similar.

2.3. **Bottom-up construction.** In *bottom-up* construction, we start by constructing the configuration of size $n-1$, and then from that we proceed to construct the configuration of size n.

That is, for very element $c \in C_{n-1}$ we try various ways of augmebnting c to obtain an element of C_n.

Bottom-up approaches are typically more *global* in nature since here the local choices have to be made *after* fixing the configuration (hence we can make them more intelligently).

3. A bottom-up problem: convolutions

3.1. **Problem statement.** Here's a (rather hard) problem which is best solved bottom-up:

Problem 1 (Romanian TST 2002). Consider words of length n in four letters: a, b, c and d. Define a convolution in a word as a contiguous repetition of the same block, for instance *aa* is a convolution and so is *abab*. Call a word *convolution-free* if it does not contain any convolutions. Prove that the number of convolution-free words of length n is at least 2^n.

3.2. **Exploration of this problem.** We first set this problem within the framework of "configurations" and "constraints":

- The *configurations* here are the strings of length n with letters drawn from a, b, c and d
- The *constraint* is that of being convolution-free, viz there is no convolution *anywhere* in the word.

We want to show that there are lots of configurations (namely at least 2^n) of length n.

In order to use induction, let's first try to understand how convolution-free words of length n are related to convolution-free words of smaller length. :

- Let w be a string of length n and v be a contiguous substring of w. Then any convolution in v gives a convolution in w.
- Hence, if w is convolution-free, so is every contiguous substring of w.
- In particular, if w is convolution-free, then the substring obtained by stripping off the last letter of w, is also convolution-free.

This means that to construct all the convolution-free words of length n, it suffices to first construct all convolution-free words of length $n-1$, and then try each of the four possibilities for the last letter.

3.3. **Some notation.** For our convenience, we'll let C_n denote the set of all convolution-free words of length n, and D_n denote the set of all words of length n whose initial segment of length $n-1$ is convolution-free.

To be able to manipulate strings (words) effectively, we'll define the following notation for substrings. Whenever $n \geq m+k-1$, define $i_{m,k}$ as the map that takes a string of length n and outputs the contiguous substring of length k starting at the m^{th} place. Then, the fact that every contiguous substring of a convolution-free word is convolution-free, can be expressed via the fact that $i_{m,k}$ maps C_n to inside C_k.

3.4. **How many will work?** Here's the question: given a convolution-free word of length $n-1$, how many possibilities are there for the last letter so that the new word is again convolution-free?

Answer: It could be as large as 3 and as low as 0.

For instance, if the word so far is *abc*, then any of the letters a, b, or d can be added.

On the other hand, if the word so far is *babcbabdbabcbab* then *none* of the letters works.

3.5. **What's the algorithm?** Let's look again at the procedure for obtaining convolution-free words of length n.

- First, list all the elements of C_{n-1}, viz all the convolution-free words of length $n-1$.
- Using this, list all the elements of D_n. Recall that D_n is the set of words of length n whose initial segment of length $n-1$ is convolution-free. $|D_n| = 4\,|C_{n-1}|$ because all we do is try appending each of the four letters to each element of C_{n-1}.
- Try every word that arises by attaching a letter at the end to a convolution-free word of length $n-1$. For this word, check that it is convolution-free.

If the initial segment of $n-1$ letters is convolution-free, then any convolution in the new word must have been *introduced* by the last letter – hence one of the blocks must contain the last letter. This forces the following picture:

The last k letters are a repeat of the k letters just before them (here $k \leq n/2$).

Put in symbols, this is the same as:

$$\forall\ w \in D_n \setminus C_n, \exists k \leq n/2 \text{ such that } i_{m-2k+1,k}(w) = i_{m-k+1,k}(w)$$

Let's think of it as a filter. We send to this filter an element in D_n. The filter checks, for each k, if the last k letters are a repeat of the k letters that come before them. If this happens, the word is "filtered out" by the filter.

Thus, to find out how many candidates get filtered in, we can calculate how many candidates get filtered out.

Let F_n denote the set of candidates that are filtered out by this process. In other words, $F_n = D_n \setminus C_n$. Then, we have:

$$F_n = \bigcup_{1 \leq k \leq n/2} F_{n,k}$$

where $F_{n,k}$ is the set of those candidates that are rejected because the last k letters are a repeat of the k letters before them.

In symbols:

$$F_{n,k} = \{w \in D_n | i_{m-2k+1,k}(w) = i_{m-k+1,k}(w)\}$$

3.6. **Cardinality bounding.** To get a bound on the size of F_n, it suffices to get a bound on the size of $F_{n,k}$ for each k. Let's look closely at a word $w \in F_{n,k}$.

By assumption, the initial segment of w of length $n-1$ is convolution-free, hence, since $k \geq 1$, the initial segment of w of length $n-k$ is convolution-free. Further, once we fix the first $n-k$ letters, the last k are anyway fixed (since they are a repeat of earlier letters). Hence, we have an injective map:

$$i_{1,n-k} : F_{n,k} \to C_{n-k}$$

which just strips the last k coordinates.

In particular:

$$|F_{n,k}| \leq |C_{n-k}|$$

Thus:

$$\begin{aligned} |F_n| &\leq \sum_k |F_{n,k}| \\ \implies |F_n| &\leq \sum_k |C_{n-k}| \\ \implies |C_n| &\geq |D_n| - \sum_k |C_{n-k}| \\ \implies t_n &\geq 4t_{n-1} - \sum_k t_{n-k} \end{aligned}$$

3.7. **The final step.**

Claim. For any $n \geq 2$, $t_n \geq 2t_{n-1}$

Proof. Let us assume that the result holds inductively. Then we have:

$$t_{n-1} \geq 2t_{n-2} \geq 2^2 t_{n-3} \dots$$

Thus in general:

$$t_{n-k} \leq \frac{t_{n-1}}{2^{n-1-k}}$$

Plugging this in the inequality for t_n, we get:

$$\begin{array}{rcl} t_n & \geq & 4t_{n-1} - t_{n-1}(1 + \frac{1}{2} + \frac{1}{2^2} \dots) \\ \implies t_n & \geq & 4t_{n-1} - 2t_{n-1} \\ \implies t_n & \geq & 2t_{n-1} \end{array}$$

We are using lots of loose inequalities here (for instance, the actual summation is only a finite one but we are extending it to an infinite summation so that we can apply the formula for summing a geometric progression). □

3.8. **Learning and value from this proof.** This proof brings out many ideas:

(1) The nature of the constraints and dependencies encourages a bottom-up construction: In the problem of finding convolution-free words, a top-down approach is less natural, because there are lots of dependencies (in fact, there are dependencies all over the place). Hence, it is hard to make one's *initial* choices wisely without knowing how the rest of the word looks like. A bottom-up approach, where one starts off with a convolution-free word of length $n-1$ and *then* tries to plug in the last letter, thus takes the dependencies into account more effectively.
(2) For any *specific* configuration of size $n-1$, we cannot predict how many configurations of size n we can construct from it. However, we can count (or at least bound) the *total* number of configurations that get rejected.
(3) When trying to prove a lower bound on a combinatorial quantity, it is sometimes more helpful to inductively prove a lower bound on the growth rate. For instance, in this problem, we started off with trying to prove $t_n > 2^n$, but found it easier to prove that $t_n \geq 2t^{n-1}$.

4. Greedy algorithms for existential problems

4.1. **The core idea.** Suppose we have a notion of configurations for every size n, and we are asked to determine whether there exists a configuration of that size n. Then, the top-down approach will say: make some initial choices, and then come down to the problem of constructing a configuration of size $n-1$.

Suppose now that there *does* exist a configuration of size n. Then there are two possibilities:

- Our initial choices are wise and they help us hit a configuration
- Our initial choices are unwise and they do not help us hit a configuration

On the other hand, suppose there does not exist a configuration of size n. Then there is only one possibility: our initial choices do not help us hit a configuration.

Thus, if our initial choices lead to a correct configuration, we have solved the existence problem positively. However, if our initial choices do *not* yield a correct configuration, we cannot be sure whether or not there exists a correct configuration. The problem is that our initial choices *may have been bad.*

This leads us to the question: Can we use some local heuristic to ensure that our initial choices are always the *best*, that is, that if there is a correct configuration, then there will also be a correct configuration yielded by our initial choices?

4.2. **The notion of local heuristics.** The Knight's Tour Problem is as follows: Given a starting position of a knight on a chessboard, make the knight "tour" the entire board, using knight's moves, visiting each square *exactly* once.

Here, at every stage ,we have to decide, based on local criteria, where the knight should move next. Can we evolve a local heuristic that will tell us what move of the knight is *best*? One possible local heuristic is Warnsdorff's rule, which works well for boards of smaller size. The idea of this rule is to always go to that neighbour which has the least number of neighbours.

The problem in general is to determine a suitable local heuristic that guarantees good results. Of course, an ideal local heuristic is one where we have the following guarantee:

> There exists a configuration if and only if there exists a configuration using the local heuristic.

Algorithms which use such smart local heuristics are termed greedy algorithms.

4.3. **Why Warnsdorff's rule works.** Let's look again at Warnsdorff's rule to understand *why* it works for boards of small size.

Warnsdorff's rule uses the local heuristic of giving priority to the *most lonely* square among the neighbours (viz the square with the minimum number of free neighbours). The reason why this is important is as follows. Suppose there is a neighbour (in the knight's move sense) with only one more free neighbour. If the knight does *not* move to this neighbour now, then the only way it can visit the neighbour in the future is through the *other* neighbour. But in that case, it can't come out.

In commonsense language: if you want everybody to survive, you have to help the most needy. In the same way, if you want to cover *every* square, you should give priority to the squares that have a chance of being blocked off or getting isolated.

We shall see the same idea in the problem in the next section.

5. An interesting problem

5.1. **The tiling problem.**

Problem 2 (IMO 2001 shortlist). Let $m < n$ be positive integers. Call a three-element subset T of $\mathbb{N}$ a tile if it is either of the form $\{x, x+m, x+m+n\}$ or of the form $\{x, x+n, x+m+n\}$. Prove that $\mathbb{N}$ can be expressed as a disjoint union of tiles.

5.2. **Thinking of the problem in terms of choices.** Call an expression of $\mathbb{N}$ as a disjoint union of tiles a **tiling**(defined) of $\mathbb{N}$. The idea is to view a tiling as a sequence of *choices* that we need to make.

Let's do this from start to end. Since 1 is a member of some tile, and since there is no natural number smaller than 1, that tile could be either of the form $\{1, 1+m, 1+m+n\}$ or of the form $\{1, 1+n, 1+m+n\}$. Thus, we have a *choice* for 1.

Once we have fixed some of the tiles, we have to proceed to decide tiles for the *smallest* untiled number. Let l denote the smallest untiled number at any stage. Then, to obtain a tile fitting l, we must either choose $\{l, l+m, l+m+n\}$ or $\{l, l+n, l+m+n\}$. Thus, at every stage, we have two choices. (It may so happen that one or both of them is not available to us for some values of l).

What *local heuristic* should we use to determine which choice to take at each stage, so that we always have at least one choice?

5.3. **Which one is preferable?** Let us call a tile of the form $\{x, x+m, x+m+n\}$ a **short tile**(defined) and a tile of the form $\{x, x+n, x+m+n\}$ a **long tile**(defined). The question: in the start, what tiles should we use: short tiles or long tiles?

Clearly, if we use the long tile first, we may run into a blockage issue. For instance, suppose we consider $m = 1$, $n = 2$. Then if we use the long tile at 1, we cannot fit any tile at 2.

The problem is very similar to that in the knight's tour:

- In the knight's tour, if we miss out on the vertices which have low degree, then they get trapped forever.
- In the tiling problem, if we indiscriminately use the long tile, we miss out on the small numbers and they get trapped forever.

Hence, it makes sense in general to use the short tile. What kind of results does this heuristic guarantee? Let's first precisely formulate the heuristic:

> At any stage, after having placed i tiles, let l be the least number that is not a member of any tile. Then, if the short tile can be fit to l, fit it. Otherwise fit the long tile to l.

5.4. **Proof that it works.** We need to show that at every stage, either the short tile or the long tile must work. So suppose, for a given l, the short tile doesn't work. Clearly, $l+m+n$ cannot be a member of any tile, because by our inductive construction, the smallest member of any tile so far has to be less than l.

We need to show that at every stage, we can choose either the short tile or the long tile. In fact, we will show that at every step, we *can* choose the long tile as long as we have been following the heuristic.[1]

Some observations:

(1) By the inductive construction, there is no tile that starts beyond l.
(2) The short tile and the long tile starting at any value j differ only in the presence of $j+m$ versus $j+n$. Thus, if we were unable to choose the short tile and managed to choose the long tile, the only reason could be that $j+m$ was already a member of some tile.

In order to show that we can fit a long tile starting at l, it suffices to show that $l+n$ has not yet been tiled. Now given that $n > m$, $l+n$ cannot be the middle member of an existing tile since that would show that there are existing tiles which have a least value at least l, contradicting our inductive construction (as per point (1) above). the only way that $l+n$ could be tiled is if it is the largest member of a tile, and hence it can be tiled only by a tile $\{l-m, l+n-m, l+n\}$. In particular, this means that at $l-m$, we *had* chosen the long tile, and thus, the short tile at $l-m$ was *already* blocked. This could happen only if $l = (l-m)+m$ was blocked, as per point (2) above. But this contradicts the assumption that l is the smallest unblocked value.

6. Another problem to consider

Problem 3 (IMO 2003, Problem 1)**.** Let A be a 101-element subset of the set $S = \{1, 2, \dots, 1000000\}$. Prove that there exist numbers $t_1, t_2, \dots, t_{100}$ in S such that the sets

$$A_j = \{x + t_j \mid x \in A\}, \qquad j = 1, 2, \dots, 100$$

are pairwise disjoint.

The *configuration* in this problem is a set S, and the constraint is that for distinct i and j, there cannot exist distinct x and y in A such that $x + t_i = y + t_j$.

6.1. **Making choices.** How do we *choose* our set of t_is? That is, how do we choose what is t_1, t_2, and so on?

Let us say that the t_is are in ascending order. Then, selecting t_1 first means selecting the smallest element of the set of t_is. Then, selecting t_2 means selecting the second smallest of all the t_is. And so on. At any stage, if we have already selected t_1, $t_2, \dots, t_i$, then selecting t_{i+1} means selecting the smallest among the other elements.

Now, we want to pick as many as 100 elements, so the idea is to try to pick the elements as *small* as possible. This means that having fixed $t_1, t_2, \dots, t_i$, we should try to pick for t_{i+1} the smallest number which is *feasible.* What is the smallest number that is feasible? Clearly, a number that has not been *blocked.*

Now, if we can obtain an upper bound M on the *number of values* that have been blocked, then the smallest unblocked number is at most $M+1$.

How many values are blocked? Clearly, numbers of the form $x + t_j - y$ where $x, y \in S$ and $1 \le j \le i$.

Thus, even assuming that all the blocked numbers are distinct, we get an upper bound of $i|S|^2$ on the number of blocked numbers and this does the trick.

Actually some smarter counting of the number of blocked numbers will show us that we can restrict ourselves to blockage of the form $t_j + (x - y)$ where $x > y$ because the other numbers would already have been blocked from some earlier source.

[1]This is analogous to saying that if we skimp whenever possible, we always have the flexibility to splurge; but if we splurge whne we don't need to, we may run into problems

7. A non-existence problem and some philosophy

7.1. The problem statement.

Problem 4 (Indian Selection Test 4, 2003)**.** Consider any partition of the numbers $\{1, 2, \dots, 3n\}$ into three sets A, B and C, each of size n. Prove that there exist $x \in A$, $y \in B$, and $z \in C$ such that one of x, y and z is the sum of the other two.

This problem superficially looks very much the opposite of earlier problems: all earlier problems required us to show that certain nice configurations exist, while this problem requires us to show that a nice configuration does *not* exist.

However, the techniques for solving the problems remain very similar: we try to use *blocking* to show that such a configuration cannot be attained.

7.2. Structure-finder and structure-avoider.

Many combinatorial situations can be modelled as follows. The aim is to construct a configuration that *avoids* a certain kind of structure. For instance, in the previous problem involving x_is and t_js, the aim was to construct a configuration that avoids a *collision* between the $x_i + t_j$s. Similarly, the famous Ramsey problems such as finding graphs where no m people know each other, and no n people know each other, we are trying to *avoid* certain subgraphs.

Configurations that avoid a structure can be modelled as follows:

- The *structure-avoider* is a player whose goal is to construct a configuration that avoids the structure
- The *structure-finder* is a player whose goal is to, given any configuration, find that structure in it

In this sense, both the structure-avoider and the structure-finder are *constructive.* The structure-avoider's constructive goal is to build the configuration, the structure-finder's constructive role is to find the structure.

For instance, in the problem of convolution-free words, the *structure* in question was a covolution within the word and the structure-avoiding configurations were the convolution-free words.

7.3. Coming back to the present problem.

In the present problem:

- The configurations are partitions of $\{1, 2, \dots, 3n\}$ into sets of size n each
- The structure (to be avoided) is a triple (x, y, z), each from a different element of the partition, such that one is a sum of the other two
- Thus, the structure-finder's goal is to, given any configuration, find a triple satisfying the above. The structure-avoider's goal is to construct a configuration such that the structure-finder cannot succeed.

In such a game, the structure-finder can best meet his/her goal by repeatedly asking the structure-avoider questions. At each stage, the structure-avoider has to make compromises and weird choices, and is finally pushed into a corner (viz a contradiction).

7.4. The game begins.

A quick run of the first few steps:

- Without loss of generality, asume $1 \in A$.
- Let k be the least element not in A. Thus, $1, 2, \dots, k-1 \in A$. Again, without loss of generality, assume $k \in B$.

We now have a slight picture. Clearly, the presence of these elements constrains choices for the structure-avoider. For instance:

- The structure-avoider cannot place consecutive elements in B and C, because $1 \in A$. More generally, the structure-avoider cannot place in B and C any two elements that are *close by* in the sense that they differ by a value less than k.
- The strucure-avoider cannot place elements which differ by k in A and C.

Together, these two facts tell us that:

If $m, m-1 \in C$, so are $m-k, m-k-1$.

And this forces an infinite descent, leading to a contradiction.

Thus, no two consecutive elements are in C.

This forces that for any $m \in C$, $m-1 \in A$. Thus, we have an *injective* map from C to A.

Moreover this injective map never takes the value 1 (because $2 \in A$ or $2 \in B$). Thus, we have an injective map from C to A that misses at least one value. Thus, $|A| > |C|$ contradicting the fact that all have equal size.

8. Techniques used: a summary

The summary provided here is only a summary of observations and techniques and general themes in the problems discussed here, *not* a summary of general techniques for construction problems.

(1) To construct all configurations of size n, we can use inductive construction. The idea behind inductive construction is to relate configurations of size n with configurations of size $n-1$. In some cases, we may also need to involve configurations of even smaller size (for instance, while finding convolution-free words of length n, we involved convolution-free words of length smaller than $n-1$ as well).

(2) Inductive construction could be done in two typical ways. One is the bottom-up way where we start with a configuration of size $n-1$ and augment in such a way as to get a configuration of size n. While augmenting we need to make sure that the new structure again satisfies the constraints.

(3) Top-down coonstructions first start from trying to build a onfiguration of size n, and then reduce it to building a configuration of size $n-1$. The idea is that we first make some local choices to reduce the problem to size $n-1$. These constructions work well when the constraints are sufficiently local in nature.

(4) When using inductive construction to get an exact or approximate count, it is important to keep in mind that oen only needs to count the *total* number of configurations that satisfy the constraints. It may so happen that for each configuration of size $n-1$, we may not be able to judge how many configurations of size n it will give rise to. However, we may still be able to determine the total number of configurations by finding out how many get rejected in total.

    We followed this approach for the problem of counting the number of convolution-free words.

(5) Top-down constructions are used in existential problems via the tool of greedy algoriths. A greedy algorithm constructs a configuration by making a local choice at every stage such that, if a configuration exists, then at each stage, the local choice gives rise to a valid configuration (the local choices are thus never bad).

(6) For problems that involve *covering everything* (by means of touring, tiling or other similar constructions) the natural choice for a greedy algorithm is one that gives top priority to covering the things that are likely to get isolated or left out.

    We followed this approach for the knight's tour problem, as well as for the touring problem.

(7) In some problems, we are required to construct configurations that avoid a certain kind of structure. These problems can be viewed in terms of a game between two constructive players. One player (the *structure-avoider*) is trying to construct a configuration that avoid the structure. The other player (the *structure-finder*) is trying to find the structure in the configuration chosen by the structure-avoider.

    We saw this in the problem of partitioning a set into three sets each of equal size.

(8) When the structure to be avoided is a small local structure (for instance, one number being the sum of the other two) then the structure-avoider can proceed by, at each stage, choosing the least element that avoids the structure. For this, one needs to keep track of those choices that are blocked and show that one can always find unblocked configurations?

Appendix A. More on convolution-free words

(1) Suppose the alphabet comprises m letters instead of 4. Mimic the proof for $m=4$ to find a good α for which it is true that the number of convolution-free words of length n is at least α^n.

(2) For a positive integer m, defien the convolution-free growth-rate of m (denoted $c(m)$) as $\liminf_{n\to\infty} e^{\log(t_{n,m})/n}$. The problem we have with us shows that $c(4) \geq 2$. Can we get a stronger lower bound of $c(4)$? For instance, can we show that $c(4) \geq 1+\sqrt{5}$? What are the upper bounds on $c(4)$?

(3) What are the answers to similar questions for convolution-free cyclic words? A cyclic word is a word written around a circle, and a convolution in a cyclic word is a repeating block of letters that is *not* self-overlapping.

Appendix B. More on tiling problems

(1) Prove that for any m and n, there exists a positive integer N such that the greedy tiling procedure tiles all the numbers from 1 to N. Further, show that the tiling is symmetric about $N/2$.
(2) Find an expression (or at least an upper bound) for N in terms of m and n.
(3) Using the fact that the tiling works for ntural numbers, prove that we can perform the tiling for integers, and for rational numbers.

Appendix C. More on sum-freeness

We make two definitions:

- A subset $S \subseteq \mathbb{N}$ is termed a **sum-free set**(defined) if the equation $x + y = z$ has no solutions for $x, y, z \in S$ (note that we allow solutions where $x = y$).
- A subset $S \subseteq \mathbb{N}$ is termed a **Sidon set**(defined) if the equation $x + y = z + w$ has no solutions for $x, y, z \in S$ with $x \neq z$, $x \neq w$.

(1) Prove that for any N, one can find a sum-free subset of $\{1, 2, \dots, N\}$ of size $N/2$ or $(N + 1)/2$ (depending on whether N is even or odd).
(2) Prove that for any N, one can find a Sidon subset of $\{1, 2, \dots, N\}$ of size at least $\sqrt{N/8}$.
(3) Prove that there exists a sum-free subset $S \subseteq \mathbb{N}$ with the property that for any N, $|S \cap \{1, 2, \dots, N\}| \geq \sqrt{(8N + 1)/4} - 1/2$.
(4) Prove that there exists a Sidon subset $S \subseteq \mathbb{N}$ with the property that for any N, $|S \cap \{1, 2, \dots, N\}| \geq N^{1/3}$. Can the right side be made a bit better?

Appendix D. More on partitioning the set

(1) Prove that if the set $\{1, 2, \dots, n\}$ is partitioned into three subsets each of which has size at least $n/4$, then we can pick one element from each set such that one of them is the sum of the other two.
(2) Prove that if A, B, C. D are four sets that form a partition of $\{1, 2, \dots, 4n\}$, each set having size n, we can pick x, y, z, w, one from each set, such that $x + y = z + w$.

Index