You should be able to attempt and solve Problems 1 and 2 even before the lecture; Problem 3 should be doable, but it'd make sense to wait until after the lecture, and otherwise at the very least require you to have a look at the lecture notes.

Problem 5 is quite hard (see below): you do not *have* to solve it, but then it'd be good to have a quick look at the solutions to get some idea about the argument (ignoring the annoying details). Same for Problem 8.

Problem 7 is "fun", but optional: it illustrates the use of the Probabilistic Method covered at the end of the lecture.

The problems marked with a $(\star)$ are on the difficult side for their level (Warm-up, Problem solving, Advanced). Those with $(\star\star)$ are very difficult (or time-consuming).

Warm-up

Problem 1. Check your understanding: how many independent random bits are necessary (and sufficient) to generate a uniformly random integer in $\{1, \dots, n\}$? To generate a uniformly random subset $S \subseteq \{1, \dots, n\}$?

Solution 1. Generally speaking, $\lceil \log_2 k \rceil$ bits of randomness are necessary (and sufficient) to generate a uniformly random element from a domain of size k. This implies the answer to the first question is $\lceil \log_2 n \rceil$, and the answer to the second is $\lceil \log_2(2^n) \rceil = n$.

Problem 2. Let X, Y be independent Bernoulli random variables with parameter $1/2$ (that is, independent, uniformly random bits), and set $Z = X \oplus Y$. Show that Z is a uniformly random bit, and that X, Y, Z are pairwise independent but not independent.

Solution 2. To see that Z is uniformly distributed, observe that $Z = 0$ if, and only if, $X = Y$, which means

$$\begin{aligned}\Pr[\,Z = 0\,] &= \Pr[\,X = 0, Y = 0\,] + \Pr[\,X = 1, Y = 1\,] \\ &= \Pr[\,X = 0\,] \cdot \Pr[\,Y = 0\,] + \Pr[\,X = 1\,] \cdot \Pr[\,Y = 1\,] \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}\end{aligned}$$

where we used that X, Y are independent and uniformly distributed. For pairwise independence: we already know that X, Y are pairwise independent (that's our assumption), so it only remains to show that X, Z are pairwise independent (by symmetry, Y, Z will be pairwise independent by the same argument). We can do it

as follows: for every $b, b' \in \{0,1\}$,

$$\begin{aligned}\Pr\big[\, X = b, Z = b' \,\big] &= \Pr\big[\, X = b, X \oplus Y = b' \,\big] \\ &= \Pr\big[\, X = b, b \oplus Y = b' \,\big] \\ &= \Pr\big[\, X = b, Y = b \oplus b' \,\big] \\ &= \Pr[\, X = b \,] \cdot \Pr\big[\, Y = b \oplus b' \,\big] \\ &= \frac{1}{2} \cdot \frac{1}{2} = \Pr[\, X = b \,] \cdot \Pr\big[\, Z = b' \,\big]\end{aligned}$$

where we relied again on the independence of X and Y (and, for the last step, on X, Z both being uniformly distributed).

This proved pairwise independence of X, Y, Z. To wee why they are not independent, note for instance that

$$\begin{aligned}\Pr[\, X = 0, Y = 0, Z = 1 \,] &= \Pr[\, X = 0, Y = 0, X \oplus Y = 1 \,] = \Pr[\, X = 0, Y = 0, 0 = 1 \,] \\ &= 0 \\ &\neq \frac{1}{8} = \Pr[\, X = 0 \,] \cdot \Pr[\, Y = 0 \,] \cdot \Pr[\, Z = 1 \,]\end{aligned}$$

Problem 3. We have seen in class the definition of a family of pairwise independent hash functions, also called a *strongly universal hash family* from $\mathcal{X}$ to $\mathcal{Y}$: $\mathcal{H}$ is such a family if, for every distinct $x, x' \in \mathcal{X}$ and every $y, y' \in \mathcal{Y}$, we have

$$\Pr_{h \sim \mathcal{H}}\big[\, h(x) = y, h(x') = y' \,\big] = \frac{1}{|\mathcal{Y}|^2}$$

where the probability is over the uniformly random choice of $h \in \mathcal{H}$. We now introduce a related (but weaker) concept: $\mathcal{H}$ is a *universal hash family* from $\mathcal{X}$ to $\mathcal{Y}$ if, for every distinct $x, x' \in \mathcal{X}$,

$$\Pr_{h \sim \mathcal{H}}\big[\, h(x) = h(x') \,\big] \leq \frac{1}{|\mathcal{Y}|}$$

Show that every strongly universal hash family is a universal hash family. *(Note: the converse is not true, see for instance Problem 8.)*

Solution 3. Suppose $\mathcal{H}$ is a *strongly* universal hash family from $\mathcal{X}$ to $\mathcal{Y}$. Then, for any two distinct $x, x' \in \mathcal{X}$,

$$\Pr_{h \sim \mathcal{H}}\big[\, h(x) = h(x') \,\big] = \sum_{y \in \mathcal{Y}} \Pr_{h \sim \mathcal{H}}\big[\, h(x) = y, h(x') = y \,\big] = \sum_{y \in \mathcal{Y}} \frac{1}{|\mathcal{Y}|^2} = \frac{1}{|\mathcal{Y}|}$$

where we used the guarantee provided by strongly universal hashing for the second equality. Note that in the end, we get an equality ($= \frac{1}{|\mathcal{Y}|}$), instead of the slightly relaxed requirement of a universal hash family: $\leq \frac{1}{|\mathcal{Y}|}$ would have been enough.

Problem solving

Problem 4. ($\star$) Give a randomised algorithm which, on input a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, runs in time $O(m(n+m))$ and outputs a cut (A, B) such that $c(A, B) \geq \frac{m}{2}$ with probability at least 0.99.

Solution 4. We would like to use what we saw in class, namely, that since $\mathbb{E}[c(A,B)] \geq \frac{m}{2}$ for our "basic" randomised algorithm, we have

$$\Pr\Big[\, c(A,B) \geq \frac{m}{2} \,\Big] > 0$$

and then repeat the algorithm many times (and check the output each time) to amplify the probability of success. Unfortunately, "strictly positive probability" is not enough: maybe this probability could be arbitrarily close to zero, and how many times would we need to run the basic algorithm to amplify this?! Now, imagine that we have proved instead, for some not-too-small $p > 0$ (we'll show $p = \Omega(1/m)$).

$$\Pr\Big[\, c(A,B) \geq \frac{m}{2} \,\Big] \geq p \tag{1}$$

where the probability is over the choice of a random cut as before: that is, $\geq p$ instead of > 0. That'd be enough! Then, just repeating the process (with independent, fresh randomness) a total of $O(1/p)$ times, we'd have that with probability at least 99% *one* of the these random cuts has value at least $m/2$. Indeed (think of the geometric distribution!) the probability that something which happens with probability p does not happen in $T = 5/p$ independent attempts is at most

$$(1-p)^T \leq e^{-pT} = e^{-5} < 1/100$$

(where we used $1 - x \leq e^{-x}$). And if one of the cuts obtained has at least $m/2$ edges, we can easily detect it: counting the number of edges of a given cut takes $O(m)$ time. So overall, choosing a uniformly random cut takes time $O(n)$ (n random bits to set, one per vertex), checking whether the result has value at least $m/2$ takes time $O(m)$), and for $O(1/p)$ repetitions we end up with a total running time of $O(\frac{1}{p}(m+n))$.

So it "suffices" to prove this stronger statement (1) for $p = \Omega(1/m)$. To do so, let's start by what we proved earlier, $\mathbb{E}[c(A,B)] = \frac{m}{2}$, and try to squeeze more of it. For convenience, define $\delta := \Pr\big[\, c(A,B) \geq \frac{m}{2} \,\big]$ (and observe that a cut cannot have more than m edges, so $\Pr[\, c(A,B) \geq \ell \,] = 0$ for $\ell > m$)

$$\begin{aligned}
\frac{m}{2} = \mathbb{E}[c(A,B)] &= \sum_{\ell=1}^{\infty} \Pr[\, c(A,B) \geq \ell \,] = \sum_{\ell=1}^{m} \Pr[\, c(A,B) \geq \ell \,] \\
&= \sum_{\ell=1}^{\lceil m/2 \rceil - 1} \underbrace{\Pr[\, c(A,B) \geq \ell \,]}_{\leq 1} + \sum_{\ell=\lceil m/2 \rceil}^{m} \underbrace{\Pr[\, c(A,B) \geq \ell \,]}_{\leq \Pr\left[\, c(A,B) \geq \frac{m}{2} \,\right] = \delta} \\
&\leq (\lceil m/2 \rceil - 1) \cdot 1 + (m - \lceil m/2 \rceil + 1) \cdot \delta = (\lceil m/2 \rceil - 1) + (\lfloor m/2 \rfloor + 1) \cdot \delta
\end{aligned}$$

and so

$$\Pr\Big[\, c(A,B) \geq \frac{m}{2} \,\Big] = \delta \geq \frac{\frac{m}{2} - (\lceil \frac{m}{2} \rceil - 1)}{\lfloor \frac{m}{2} \rfloor + 1} \geq \frac{\frac{1}{2}}{\lfloor \frac{m}{2} \rfloor + 1} \geq \frac{\frac{1}{2}}{\frac{m}{2} + 1} = \frac{1}{m+2} = \Omega\left(\frac{1}{m}\right)$$

as we wanted (where we used for the second inequality the fact that $k/2 - \lceil k/2 \rceil$ is either 0 (for k even) or $-1/2$ (for k odd), so always at least $-1/2$).

Problem 5. ($\star\star$) We will prove Fact 22.2 from the lecture notes:

> There exists an explicit family of pairwise independent hash functions $\mathcal{H} \subseteq \{h\colon [n] \to \{0,1\}\}$ with $|\mathcal{H}| = 2^{\lceil \log(n+1) \rceil}$.

To do so, suppose for simplicity that $n+1$ is a power of 2, i.e., $n = 2^k - 1$ for some integer k. We will identify an integer $1 \leq x \leq n$ with its binary representation $x \in \{0,1\}^k$ (note that this representation is *not* the all-zero vector, as $x \neq 0$). Define $\mathcal{H} = \{h_S\}_{S \subseteq \{0,1\}^k}$, where, for a given set $S \subseteq \{0,1\}^k$,

$$h_S(x) = \bigoplus_{i \in S} x_i, \qquad x \in \{0,1\}^k$$

that is, $h_S(x)$ is the sum, modulo 2, of the bits of x that are indexed by S.

a) What is the size $|\mathcal{H}|$ of $\mathcal{H}$?

b) How many random bits does it take to draw a hash function h from $\mathcal{H}$? Argue such a hash function can be drawn, stored, and evaluated (on any input x) efficiently.

c) Show that $\mathcal{H}$ is a family of pairwise independent hash functions.

Solution 5.

a) This is 2^k: the number of distinct subsets of $[k] = \{1, 2, \ldots, k\}$

b) This takes $\log_2(2^k) = k$ independent uniform random bits. Generate a random k-bit string s and store it: this corresponds to a subset $S \subseteq [k]$ in the natural way ($i \in S$ if, and only if, $s_i = 1$). To evaluate h_S given this $s \in \{0,1\}^k$, it suffices to observe that we can then rewrite

$$h_S(x) = \bigoplus_{i \in S} x_i = \bigoplus_{i=1}^{k} s_i x_i$$

so h_S can be evaluated on x in $O(k)$ time by computing $x \wedge s$ (the bitwise AND of x and s), and then looking at the parity (XOR) of the resulting k-bit string.

c) Fix two distinct $x, x' \in \{0,1\}^k$, and any two $y, y' \in \{0,1\}$. We will look at the subsets of $\{0,1\}^k$ corresponding to x and x': that is, define

$$A \coloneqq \{\ i \in [k] \ :\ x_i = 1, x'_i = 0\ \} = x \setminus x'$$
$$B \coloneqq \{\ i \in [k] \ :\ x_i = 0, x'_i = 1\ \} = x' \setminus x$$
$$C \coloneqq \{\ i \in [k] \ :\ x_i = 1, x'_i = 1\ \} = x \cap x'$$

Since $x \neq x'$, at least one of A, B must be non-empty. Without loss of generality, assume $A \neq \emptyset$: whether B is empty or not, we don't know. Similarly, C could be empty as well, but if B is empty then C must be non-empty: otherwise x' would be the all-zero string, but we know $x' \neq 0$.
Given a subset $S \subseteq [k]$ corresponding to a string $s \in \{0,1\}^k$, we can rewrite (check it!) $h_S(x) = \bigoplus_{i \in A} s_i \oplus \bigoplus_{i \in C} s_i$, and $h_S(x') = \bigoplus_{i \in B} s_i \oplus \bigoplus_{i \in C} s_i$. Importantly, since A, B, C are disjoint subsets of $[k]$ and s is chosen uniformly at random in $\{0,1\}^k$, the 3 random variables

$$\bigoplus_{i \in A} s_i, \quad \bigoplus_{i \in B} s_i, \quad \bigoplus_{i \in B} s_i$$

are independent! (But, if $B = \emptyset$ for instance, then $\bigoplus_{i \in B} s_i = 1$ regardless of s. So we will have to do an annoying distinction of cases.)

Let's start: **first case**, if A, B are *both* non-empty

$$\begin{aligned}
&\Pr_S\left[\, h_S(x) = y, h_S(x') = y' \,\right] \\
&\quad = \Pr_s\left[\bigoplus_{i\in A} s_i \oplus \bigoplus_{i\in C} s_i = y, \bigoplus_{i\in B} s_i \oplus \bigoplus_{i\in C} s_i = y'\right] \\
&\quad = \sum_{y''\in\{0,1\}} \Pr_s\left[\bigoplus_{i\in A} s_i \oplus \bigoplus_{i\in C} s_i = y, \bigoplus_{i\in B} s_i \oplus \bigoplus_{i\in C} s_i = y', \bigoplus_{i\in C} s_i = y''\right] \\
&\quad = \sum_{y''\in\{0,1\}} \Pr_s\left[\bigoplus_{i\in A} s_i = y \oplus y'', \bigoplus_{i\in B} s_i = y' \oplus y'', \bigoplus_{i\in C} s_i = y''\right] \\
&\quad = \sum_{y''\in\{0,1\}} \Pr_s\left[\bigoplus_{i\in A} s_i = y \oplus y''\right] \Pr_s\left[\bigoplus_{i\in B} s_i = y' \oplus y''\right] \Pr_s\left[\bigoplus_{i\in C} s_i = y''\right] \\
&\quad = \sum_{y''\in\{0,1\}} \frac{1}{2}\cdot\frac{1}{2}\cdot \Pr_s\left[\bigoplus_{i\in C} s_i = y''\right] \\
&\quad = \frac{1}{4}
\end{aligned}$$

where for the 4th equality we used independence of the 3 random variables, and for the last we used the fact that $A, B \neq \emptyset$ (so $\bigoplus_{i\in A} s_i$, $\bigoplus_{i\in B} s_i$ are uniformly distributed) and that probabilities sum to one.
Now, **second case**, if $A \neq \emptyset$ but B is empty (then C cannot be empty), then $\bigoplus_{i\in B} s_i = \bigoplus_{i\in\emptyset} s_i = 1$, and

$$\begin{aligned}
\Pr_S\left[\, h_S(x) = y, h_S(x') = y' \,\right] &= \Pr_s\left[\bigoplus_{i\in A} s_i \oplus \bigoplus_{i\in C} s_i = y, \bigoplus_{i\in B} s_i \oplus \bigoplus_{i\in C} s_i = y'\right] \\
&= \Pr_s\left[\bigoplus_{i\in A} s_i \oplus \bigoplus_{i\in C} s_i = y, \bigoplus_{i\in C} s_i = y'\right] \\
&= \Pr_s\left[\bigoplus_{i\in A} s_i = y \oplus y', \bigoplus_{i\in C} s_i = y'\right] \\
&= \Pr_s\left[\bigoplus_{i\in A} s_i = y \oplus y'\right] \cdot \Pr\left[\bigoplus_{i\in C} s_i = y'\right] \\
&= \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}
\end{aligned}$$

here using again the fact that $\bigoplus_{i\in A} s_i, \bigoplus_{i\in C} s_i$ are independent, but also that they are both uniformly distributed since A, C are noth nonempty.

Remark: if we were only asked to prove the weaker statement that $\mathcal{H} = \{h_S\}_S$ is a universal hash family (not *strongly* universal), this would be much easier! Try it.

Problem 6. In an (undirected) graph $G = (V, E)$, a *triangle* is a triple of vertices u, v, w such that the 3 edges $(u, v), (v, w), (u, w)$ exist in E. In a *directed* graph $G = (V, \vec{E})$, an oriented triangle is a cycle of length 3: namely, a triple of vertices u, v, w such that the 3 directed edges $(u \to v), (v \to w), (w \to u)$ exist in $\vec{E}$.

Given as input an undirected graph G, we want to give an orientation to each edge $e \in E$ (that is, convert G into a *directed* graph) while maximising the number of oriented triangles in the resulting directed graph.

a) Give a randomised algorithm whose output has an *expected* number of oriented triangles at least $1/4$ the maximum possible number $\mathrm{OPT}(G)$.

b) Convert your algorithm into a *deterministic* (efficient) algorithm achieving the same approximation guarantee.

Solution 6.

a) Consider the "obvious" randomised algorithm which, for edge edge $e = (u, v) \in E$, picks one of the two orientations $(u \to v)$ or $(v \to u)$ uniformly at random, independently of other edges. Let T be set of undirected triangles in the input graph G: for every $t = (u, v, w) \in T$, denote by X_t the indicator random variable of whether the triangle t becomes a valid oriented triangle. This happens if all 3 orientations are consistent: $u \to v \to w \to u$ or $u \leftarrow v \leftarrow w \leftarrow u$ (two options). Since each edge's orientation is chosen independently and the triangle has 3 edges, each of the two options has probability $(1/2)^3$, and so we get, for every $t \in T$,

$$\mathbb{E}[X_t] = \Pr[\, t \text{ becomes an oriented triangle}\,] = 2 \cdot \left(\frac{1}{2}\right)^3 = \frac{1}{4}$$

The total number of oriented triangles we get is $\sum_{t \in T} X_t$ (the number of undirected triangles successfully becoming oriented triangles), and so by linearity of expectation the expected number of oriented triangles we get is

$$\mathbb{E}\left[\sum_{t \in T} X_t\right] = \sum_{t \in T} \mathbb{E}[X_t] = \sum_{t \in T} \frac{1}{4} = \frac{|T|}{4} \geq \frac{\mathrm{OPT}(G)}{4}$$

the last inequality since the optimal number of oriented triangles cannot be larger than the number of *undirected* triangles in the original graph (we cannot "create" triangles by orienting edges).

b) Let's derandomise the "basic" randomised algorithm above. To do so, we have two options: the first is to note that we don't need full independence of our random choices for the orientation of the edges, but just *3-wise* independence: indeed, all we need is that, for each triangle, the choices of orientation for the 3 edges involved are independent, and then we get the $(1/2)^3$ probability in our analysis. But as mentioned in class, to get m bits that are 3-wise independent, we only need $O(\log m)$ truly random bits *[more generally, for m k-wise independent bits we only need $O(k \log m)$ uniformly random bits]*, so we can use the brute-force approach seen in class to go over all random seeds and check the result for each (in time $O(n^3)$, since there are at most $\binom{n}{3} = O(n^3)$ triangles): this gives an algorithm running in time $O(2^{O(\log m)} \cdot n^3) = \text{poly}(n)$ time (recall that $m \leq n^2$, so $\log m = O(\log n)$).
The second option is to use the *method of conditional expectations*: to do so, we will fix an (arbitrary) ordering of the edges, say $e_1, e_2, \ldots, e_m$, and sequentially choose their orientation. Letting N_T be the random variable denoting the number of oriented triangles and $X_1, \ldots, X_m \in \{\uparrow, \downarrow\}$ denoting the orientation chosen for the corresponding edges, what we want is to be able to efficiently compute

$$\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \uparrow], \mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \downarrow]$$

so that, at step $i+1$ we can choose the orientation of edge e_{i+1} which maximises the conditional expectation of N_T given the choices already made. If we can do this, then since

$$\begin{aligned}\mathbb{E}[N_T \mid X_1, \ldots, X_i] &= \frac{1}{2}\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \uparrow] + \frac{1}{2}\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \downarrow] \\ &\leq \max(\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \uparrow], \mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \downarrow])\end{aligned}$$

(the first equality since our original algorithm choose $X_{i+1} = \uparrow$ and $X_{i+1} = \downarrow$ both with probability $1/2$), by picking the best choice of the two we'll have

$$\begin{aligned}\frac{\text{OPT}(G)}{4} &\leq \mathbb{E}[N_T] \\ &\leq \mathbb{E}[N_T \mid X_1] \\ &\leq \cdots \\ &\leq \mathbb{E}[N_T \mid X_1, \ldots, X_i] \\ &\leq \mathbb{E}[N_T \mid X_1, \ldots, X_{i+1}] \\ &\leq \cdots \\ &\leq \mathbb{E}[N_T \mid X_1, \ldots, X_m]\end{aligned}$$

and that last value is not random anymore, it's just what our derandomised algorithm will have given us after choosing the orientation of all m edges deterministically.

So how to compute $\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \uparrow]$ efficiently? After having chosen $X_1, \ldots, X_i$, from all the original "possible triangles" T, we only need to consider the ones involving the current edge e_{i+1} (since our choice will not affect the others). Among this set of triangles T_{i+1}, we have: (1) the ones in which 0 edges have been oriented so far; (2) the ones in which 1 edge has been oriented; (3) the ones in which 2 edges have been oriented, *and they are no longer possible because the orientation was bad*, ; (3) the ones in which 2 edges have been oriented, *and they are still possible because the orientation was consistent*. We only care about (1), (2), and (4), since the triangles in (3) are "dead" regardless of our choice for X_{i+1}.

- Each triangle t in case (1) will still be possible regardless of our choice for X_{i+1}: so $\mathbb{E}[X_t \mid X_1, \ldots, X_{i+1}] = \frac{1}{2^2} = \frac{1}{4}$ (once we choose X_{i+1}, the remaining two edges only form an oriented triangle if they both take the orientation consistent with our choice)
- A triangle t in case (2) only remains possible if X_{i+1} is consistent with the already chosen edge in that triangle: so $\mathbb{E}[X_t \mid X_1, \ldots, X_{i+1}]$ is either 0 (our choice is not consistent with that edge, the triangle dies) or $1/2$ (our choice is consistent, so the triangle will survive if the remaining last edge is also consistent)
- A triangle t in case (4) survives for sure if X_{i+1} is consistent with the 2 already chosen edges in that triangle, and dies otherwise: so $\mathbb{E}[X_t \mid X_1, \ldots, X_{i+1}]$ is either 0 or 1.

In time $O(n^3)$, for each of the two choices of $X_{i+1} \in \{\uparrow, \downarrow\}$ we can go through all the triangles in T, see which ones are in which case, and sum the corresponding value for the resulting conditional expectation $\mathbb{E}[X_t \mid X_1, \ldots, X_{i+1}]$ for that triangle (either $0, 1/4, 1/2$, or 1). This gives us $\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \uparrow]$ and $\mathbb{E}[N_T \mid X_1, \ldots, X_i, X_{i+1} = \downarrow]$, and we pick the choice for X_{i+1} leading to the biggest value of the two.

Problem 7. $(\star)$ Given a 2-colouring $c\colon E \to \{\text{red}, \text{blue}\}$ of a graph $G = (V, E)$, a *monochromatic triangle* is a triple of vertices $(u, v, w) \in V^3$ such that the edges $(u, v), (v, w), (u, w)$ exist (are in E) and $c(u, v) = c(v, w) = c(u, w)$ (they have the same colour). Show that, for every n, there exists is a 2-colouring of the complete graph K_n with *at most* $\frac{n^3}{24}$ monochromatic triangles. Give an efficient (polynomial-time) deterministic algorithm which, on input n, finds such a 2-colouring.

Solution 7. *(Sketch).* This is similar to the previous exercise: pick a uniformly random 2-colouring by choosing the colour of edge independently: since this is the complete graph, there are exactly $\binom{n}{3}$ triangles, so after colouring at random the expected number of monochromatic triangles will be $2 \cdot \frac{1}{2^3} \cdot \binom{n}{3} = \frac{n(n-1)(n-2)}{24} \leq \frac{n^3}{24}$. This can again be derandomised with either of the two approaches (3-wise independence and brute-force search; or method of conditional expectations).

Advanced

Problem 8. Fix a prime number $p \geq 2$ and an integer $n \geq 1$. For a given $a = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$, define the function $h_a \colon \mathbb{Z}_p^n \to \mathbb{Z}_p$ by

$$h_a(x) = \sum_{i=1}^{n} a_i x_i \bmod p, \qquad x \in \mathbb{Z}_p^n$$

and let $\mathcal{H} = \{h_a\}_{a \in \mathbb{Z}_p^n}$.

a) How many bits does it take to fully specify a function $h \in \mathcal{H}$? And an arbitrary function $f \colon \mathbb{Z}_p^n \to \mathbb{Z}_p$?

b) Show that $\mathcal{H}$ is a universal hash family (see Problem 3); that is, for every $x, x' \in \mathbb{Z}_p^n$,

$$\Pr_{h \sim \mathcal{H}}\left[\, h(x) = h(x') \,\right] = \frac{1}{p}$$

c) Is it a strongly universal hash family?

Solution 8.

a) This takes $n \lceil \log_2 p \rceil$ bits for the first, $p^n \lceil \log_2 p \rceil$ for the second (from $\log_2(|\mathbb{Z}_p|^{|\mathbb{Z}_p^n|}) = |\mathbb{Z}_p^n| \log_2 |\mathbb{Z}_p| = p^n \log_2 p$).

b) First, note that, for any fixed two distinct $x, y \in \mathbb{Z}_p^n$ and $a \in \mathbb{Z}_p^n$,

$$h_a(x) = h_a(y) \Leftrightarrow \sum_{j=1}^{n} a_j (x_j - y_j) = 0 \bmod p$$

It is therefore sufficient to show the statement for $x \in \mathbb{Z}_p^n \setminus \{0\}$ and $y = 0$. Fix such a x, and let m be the number of its non-zero components; since any i such that $x_i = 0$ does not contribute to the value of $h_a(x)$, and n is arbitrary, we can, without loss of generality, suppose that all x_i's are non-zero and (up to a permutation of the indices) work with the quantity $\left(\sum_{j=1}^{m} a_j x_j\right) \bmod p$ (where $m \geq 1$ by assumption on x being non-zero). We will prove by induction on $m \geq 1$ the following (stronger) claim (where every equality is taken mod p):

$$\forall q \in \mathbb{Z}_p, \qquad \Pr\left[\left(\sum_{j=1}^{m} a_j x_j\right) = q \bmod p\right] = \frac{1}{p}$$

Base case ($m = 1$) For any $q \in Z_p$, $\Pr[\, a_1 x_1 = q \,] = \Pr\left[\, a_1 = q x_1^{-1} \,\right] = \frac{1}{p}$ (as $x_1 \neq 0$, and a_1 is drawn uniformly at random).

Induction step Suppose $m \geq 2$, and fix an arbitrary $q \in \mathbb{Z}_p$.

$$\begin{aligned}
\Pr\left[\left(\sum_{j=1}^{m} a_j x_j \right) = q \right] &= \sum_{r=0}^{p-1} \Pr\left[\sum_{j=1}^{m-1} a_j x_j = q - r,\ a_m = r \right] \\
&= \sum_{r=0}^{p-1} \Pr\left[\sum_{j=1}^{m-1} a_j x_j = q - r \right] \cdot \Pr[\, a_m = r \,] \\
&\qquad \text{(independence)} \\
&= \sum_{r=0}^{p-1} \Pr\left[\sum_{j=1}^{m-1} a_j x_j = q - r \right] \cdot \frac{1}{p} \quad (a_m \text{ is uniform}) \\
&= \sum_{r=0}^{p-1} \frac{1}{p} \cdot \frac{1}{p} \qquad \text{(induction hypothesis)} \\
&= p \cdot \frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

which proves the claim.

c) No: Consider taking $x = 0$, $y \neq 0$: then regardless of x', y', we have

$$\Pr[\, h(0) = y, h(x') = y' \,] = 0$$

Interestingly, adding a_0 would fix this issue and give a strongly universal hash family: that is, having h_a defined as $h_a(x) = a_0 + \sum_{i=1}^{n} a_i x_i$, where $a \in \mathbb{Z}_p^{n+1}$ (try to prove it!)