London Taught Course Centre

2020 examination

Graph Theory

Answers

1 This question is closely related to the Mycielski construction; with sufficient Googling it can likely be found.

The chromatic number of G_n is $n+2$ for each n. To prove the upper bound (inductively) observe G_0 has a 2-colouring. Now given $n \geq 1$ and an $(n+1)$-colouring of G_{n-1}, we extend this to a colouring of G_n as follows. We colour each v_i with the same colour that $u_i \in G_{n-1}$ has. Then we assign colour $n+2$ to w. Another way to do this is to colour each v_i with colour $n+2$ and then colour w with colour 1.

Now we prove the lower bound inductively. It is obvious that we need 2 colours to colour G_0. Now suppose $n \geq 1$ and we have a proper $(n+1)$-colouring c of G_n. Consider the copy of G_{n-1} in G_n. For each colour $1 \leq i \leq n+1$, there must exist a vertex in G_{n-1} which is assigned colour i by c, otherwise we have a colouring of G_{n-1} with less than $n+1$ colours. Furthermore, for each $1 \leq i \leq n-1$ there must exist a vertex u_{t_i} of G_{n-1} whose colour is i and whose neighbours contain all colours other than i. Otherwise, we could one by one recolour every vertex of colour i to a colour not in its neighbourhood; since the vertices of colour i form an independent set, this recolouring does not affect the colours in the neighbourhood of any colour i vertices. That would give us a proper colouring of G_{n-1} with less than $n+1$ colours, a contradiction.

Now the vertex v_{t_i} must have colour i, since its neighbourhood amongst the u_j is the same as that of u_{t_i} and so contains all colours $[n+1] \setminus \{i\}$. Finally w is adjacent to each vertex v_{t_i} for $1 \leq i \leq n+1$, so w cannot have any colour in $[n+1]$, a contradiction.

A standard mistake here is to try to argue the lower bound in reverse: if we have a colouring of G_n with $n+1$ colours, then there has to be a colour on w, so there are n colours left for the v_i vertices, and we can copy the colouring of the v_i vertices to the u_i vertices for a colouring of G_{n-1} with only n colours. Sounds reasonable, but the last step is false: you can't do that so simply, as the proper colouring example where all the v_i vertices have the same colour shows. The problem is that when you recolour one u_i you change what colours are available at its neighbours; if you try to recolour a neighbour as well, then you are in trouble. To make this work, you need to find some way to do the recolouring without recolouring some u_i and u_j which are adjacent; one way to do it is to specify that you will only recolour vertices u_i which have the same colour as w.

2 (a) A is a real symmetric matrix since $ij \in E(G)$ if and only if $ji \in E(G)$. All real symmetric matrices have only real eigenvalues. Most people obviously knew this is a standard fact of linear algebra, but still gave a proof — don't bother, next time. This is the kind of thing you should simply quote.

(b) This is an easy induction on k which everyone did well.

(c) It's fairly easy to do $O(n^3)$ time; for example run breadth first search from each vertex (worst case n^2 time) and take the maximum result. That is worth 2 points.

There is a way to improve, which the question slightly points you in the direction of: try to view this as a matrix multiplication problem, using (b). The problem is the same as finding the smallest k such that for any i, j there is a walk of length at most k from i to j. This is (again trivially) the same thing as finding the smallest k such that $(A+I)^k$ has no zero entries. If $(A+I)^{n-1}$ has zero entries, then the graph is not

connected, so the diameter is ∞. You can easily check connectivity in $O(n^2)$ time, so from now on let's assume G is connected hence has diameter at most $n-1$.

There are now two tricks you need to find. One is 'standard' and often taught: if you are trying to guess a number between 1 and n and you can always find out if your guess is too large, too small or just right, then the way to do this efficiently is binary search. That is, guess $n/2$, and either you're right, or you have only half the range to search: you will get to the right answer in about $\log n$ steps (which is the best you can hope for; obviously it is much better than a linear search where you might need to guess every single number before finding the answer at the end).

The other is a standard trap, which everyone fell into (everyone always does, the first time) — if you're going to do any computational work, remember it! It's easy to find statements like 'you can multiply $n \times n$ matrices in time $O(n^{2.373\ldots})$' (a theorem of Le Gall) on the internet, but this statement is missing details, and it's important here. What is true is that this time bound holds for matrices with bounded integer entries, or (with a polylog factor, but that is almost irrelevant in the time complexity) with integer entries which grow at most polynomially in n. But the statement is not true without some such bound — and that's actually pretty obvious: if you want to calculate with an $n \times n$ integer matrix where lots of entries are as big as say 2^n, then you need something like n^3 time just to write the matrix down; in fact Le Gall's algorithm will not even do this well.

And in this problem, that matters. Suppose the graph you're working with contains a clique with $n/2$ vertices, plus some other things so that the diameter is still linear (say a path of $n/2$ vertices). There are $(n/2)!$ paths of length $n/2$ in this clique, and even more walks which start at any given vertex. That means by the time you compute $(A+I)^{n/2}$, you're dealing with a matrix in which $n^2/4$ entries are all at least $(n/2)!$; even writing this matrix down takes more than cubic time!

The trick to notice here is that we do not need to keep track of how many walks there are between a given pair of vertices. It's enough to be able to distinguish between 0 and ≥ 1. So after we do each matrix multiplication, we can afford to reduce our matrices back to $\{0,1\}$-matrices, by replacing integers bigger than 1 with 1. That keeps the numbers from blowing up.

Let's now sketch the algorithm that works. Let M_k be the matrix we get from $(A+I)^k$ by replacing all integers bigger than 1 with 1. What we want is to find the smallest k such that M^k has no zero entries. Let $t := \lceil \log n \rceil$.

We first compute M_{2^s} for $s = 0, \ldots, t-1$ by repeated squaring; we store all these matrices. That is, we have $M_1 = A + I$, and given M_{2^s} we can get $M_{2^{s+1}}$ by squaring M_s then reducing the result to an $\{0,1\}$-matrix.

Now we do binary search. We start with the 'possible interval' J_0 being 1 up to 2^t, and the 'start matrix' S_0 being the identity. We check whether the matrix $S_0 \times M_{2^{t-1}}$ (which is one of our stored matrices) contains a zero entry or not, and accordingly we change our 'possible interval' to J_1 being either 1 to 2^{t-1} or $2^{t-1}+1$ to 2^t, and start matrix S_1 to respectively S_0 or $S_0 \times M_{2^{t-1}}$ reduced to a $\{0,1\}$-matrix. Then we continue like this. In step k, we are given J_{k-1} and S_{k-1}; we check if $S_{k-1} \times M_{2^{t-k}}$ has zeroes or not, and update J_k and S_k correspondingly.

Note that we are careful that any time we do a matrix multiplication, what we multiply are $\{0,1\}$-matrices. We end up doing $O(\log n)$ matrix multiplications, plus the same number of checks whether the matrix has zero entries (which take $O(n^2)$ time).

So the time complexity is $O\big(n^x(\log n)\big)$ where n^x is the time required to multiply two $n \times n$ $\{0,1\}$-matrices, since it is known (and obvious!) that this x is at least 2. The current best we know for x is roughly 2.373..., a theorem of Le Gall; that works out to 4 points for the question. As far as I know, this is the current best we know for the diameter problem. However, it's often conjectured (though not by everyone!) that matrix multiplication can be done in time $O(n^{2+\varepsilon})$ for every $\varepsilon > 0$. So maybe the question is worth infinitely many points..!

3 (a) This is a straightforward application of Chernoff's inequality and the union bound. The probability that a given (X, Y) violates the desired condition is at most $\exp\big(-\frac{1}{3}p|X||Y|\big) \leq \exp\big(-C\eta^2 n/3\big)$, while there are at most 2^{2n} choices for (X, Y). Choosing $C \geq 12\eta^{-2}$, by the union bound there are no such pairs.

Note that this probability bound is fairly sharp — if $p = o(n^{-1})$ then the statement becomes false (and that is not too hard to prove).

(b) This is the Sparse Regularity Lemma, due independently to Kohayakawa and Rödl. It can be proved by a (actually rather easy) modification of arguments in notes; it's also easy to Google. Briefly, the modification one should make to the notes proof is (i) define a p-energy of a partition in terms of the p-density defined in the question, and (ii) check (this requires a little work to do properly!) that we only need to care about partitions in which all parts have size at least ηn, where $\eta > 0$ is a (very very small!) constant depending on ε but not n, so that for such partitions by 3(a) the p-energy cannot exceed 4.

(c) This is false, and the argument was shown in a different context in lectures. The expected number of triangles in $G(n,p)$ is at most $p^3n^3 = n^{0.3}$, so with high probability Γ has at most $n^{0.4}$ triangles (Markov's inequality). We can simply remove one edge of each triangle from Γ to obtain a triangle-free graph G. By a small variant of (a), with high probability Γ is $(\frac{\varepsilon}{2}, p)$-regular with $d_p(V_i, V_j) = 1 \pm \frac{\varepsilon}{2}$ on the given partition, and (by simple counting) removing $n^{0.4}$ edges to form G at worst changes the $\frac{\varepsilon}{2}$ to ε. So this gives us a graph G which satisfies the conditions of the question, but doesn't have any triangles at all, let alone $\frac{1}{1000}p^3n^3$ of them.

(d) This is true, but (moderately) hard to prove. A student who tries to Google this should find a reference easily enough though. This particular statement is (more or less!) a theorem of Kohayakawa, Łuczak and Rödl, though reproved several times in more generality since then.

For both the last two parts, a standard mistake is to try to mimic the proof of the Counting Lemma from lectures. This does not work (and for (c) it can't as it would prove a false statement!); the place where it goes wrong is that in the proof of the Counting Lemma, we pick a typical v in V_1 and look at its neighbours in V_2 and V_3. These are two large sets (on the order of n vertices, by choice of ε much bigger than εn) and so ε-regularity tells us how many edges go between the sets, i.e. how many triangles there are at v. If we try

to do this in this sparse setting, then we will be able to show that a typical vertex in V_1 has neighbourhoods of the 'right' size in V_2 and V_3, but the 'right size' is on the order of pn, which is much smaller than εn: (ε, p)-regularity doesn't tell us anything about edges between such small sets.

As usual with this course, this exam was supposed to highlight a few important features and standard traps (i.e. you should hopefully learn from it), and to make you suffer (at least a bit) if you tried to do it without using the internet. The reason for the last is that if you try to do research without first finding out what other people have done, you can waste a lot of time trying to prove false statements, or repeating others' work. Find out what is out there before trying to solve a problem. You might not want to look at someone else's solution right away — you can learn a lot by trying to find your own — but it's a good idea to know whether the problem you are looking at is solved or not.