1. ***k*-colour $\in$ *NP*:** Consider the following verifier:

   > On input (G,c), where c is an assignment of colours to the vertices of G, check that c uses no more than k colours and that every edge of G has endpoints with different colours.

   This runs in polytime: a linear-time loop over the edges of G and a linear-time loop over the values of c.

   Moreover, the verifier outputs True for some c iff there exists a colouring of G using no more than k colours.

   ***k*-colour is *NP*-hard:** For an arbitrary $k \geqslant 3$, we show 3-colour $\to_p$ k-colour.

   > On input G, output G' equal to G with the addition of $k-3$ new vertices $u_1, \dots, u_{k-3}$, with edges between every pair of new vertices and between every new vertex and every vertex in G.

   Clearly, G' can be computed in polytime from G: we add a constant number of new vertices and a linear number of new edges, with a simple, regular structure.

   Also, if G can be coloured using at most 3 colours, then G' can be coloured using at most k colours: use 3 colours for the vertices of G and $k-3$ other colours for $u_1, \dots, u_{k-3}$ (one colour for each vertex).

   Finally, if G' can be coloured using at most k colours, then $u_1, \dots, u_{k-3}$ must be assigned $k-3$ different colours (because $u_1, \dots, u_{k-3}$ forms a clique), and none of those colours can be assigned to any vertex in G (because each u_i is adjacent to every vertex of G). This means the vertices of G can be coloured using only the 3 remaining colours.

   Alternate Reduction

   Show that k-colour $\to_p$ $(k+1)$-colour for every $k \geqslant 3$, (then k-colour is *NP*-hard for every $k \geqslant 3$, by induction):

   > On input G, output G' equal to G with the addition of one new vertex v_0 together with edges between v_0 and every vertex in G.

   Clearly, G' can be computed in linear time from G.

   Also, if G can be coloured with at most k colours, then G' can be coloured using at most $k+1$ colours: we only need one more colour for v_0.

   Finally, if G' can be coloured using at most $k+1$ colours, then the colour assigned to v_0 must be different from the colour of every other vertex. This means the vertices of G are coloured using at most k colours.

2. **Search Problem:** The 3-colour-search problem can be defined as follows.

   - **In:** Undirected graph $G = (V, E)$.
   - **Out:** Assignment of colours $c : V \to \{c_1, c_2, c_3\}$ such that every edge of G has endpoints of different colours $(\forall (u,v) \in E, c(u) \neq c(v))$—special value nil if this is not possible.

   Algorithm: To show that 3-colour-search is polytime Turing-reducible to 3-colour, assume that algorithm CD(G) solves 3-colour (for every graph G, CD(G) = True iff G is 3-colourable). We write an algorithm CS to solve 3-colour-search.

   CS(G):

       if not CD(G): **return** nil

       # At this point, we know G is 3-colourable.

\# Assign a first arbitrary vertex to colour 1.
Pick a vertex $u_1 \in V$, and let $C_1 \leftarrow \{u_1\}$
\# Find every other vertex that can be assigned the same colour as u_1.
for all $v \in V - \{u_1\}$:
  if $(u_1, v) \notin E$ **and** $\text{CD}(G_{u_1 v})$:
    $C_1 \leftarrow C_1 \cup \{v\}$ # v and u_1 can be assigned the same colour
    $G \leftarrow G_{u_1 v}$ # continue with u_1 and v merged, since they have the same colour
\# At this point, G is the result of merging every vertex from C_1 with u_1.

\# Pick an arbitrary remaining vertex and assign it to colour 2.
Pick a vertex $u_2 \in V - \{u_1\}$, and let $C_2 \leftarrow \{u_2\}$
\# Find every other vertex that can be assigned the same colour as u_2.
for all $v \in V - \{u_1, u_2\}$:
  if $(u_2, v) \notin E$ **and** $\text{CD}(G_{u_2 v})$:
    $C_2 \leftarrow C_2 \cup \{v\}$ # v and u_2 can be assigned the same colour
    $G \leftarrow G_{u_2 v}$ # continue with u_2 and v merged, since they have the same colour
\# At this point, G is the result of merging every vertex from C_2 with u_2.

\# Since G was 3-colourable, all remaining vertices can be assigned colour 3.
$C_3 = V - \{u_1, u_2\}$

return C_1, C_2, C_3

Runtime: Algorithm CS makes $\mathcal{O}(n)$ calls to CD (where $n = |V|$), and merges at most two vertices (in time $\mathcal{O}(n+m)$) for each call to CD. The total running time is therefore $\mathcal{O}(n \cdot t(n,m) + n(n+m))$, where $t(n,m)$ is the running time of CD.

Correctness: The algorithm maintains the loop invariant that G is 3-colourable, and assigns the same colour to different vertices only once it has confirmed that this is possible, based on the following claim.

For every graph $G = (V, E)$ and any $u, v \in V$ such that $(u, v) \notin E$, there is a 3-colouring of G where u and v are assigned the same colour iff G_{uv} is 3-colourable.

Proof: If $G = (V, E)$ is a graph and $u, v \in V$ with $(u, v) \notin E$ and if there is a 3-colouring of G in which u and v are assigned the same colour, then the same colouring can be applied to G_{uv}. Any vertices adjacent in G_{uv} are also adjacent in G so will be assigned different colours.

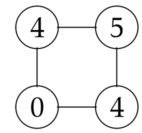
Conversely, if G_{uv} is 3-colourable, then the same colouring can be applied to the vertices of G, using the same colour for u and v. Again, adjacent vertices will be assigned different colours, because they are also adjacent in G_{uv}.

3. (a) Algorithm:

$A \leftarrow \{(i,j) : 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n\}$ # A is the set of every available intersection
$S \leftarrow \varnothing$ # S is the current selection of intersections
while $A \neq \varnothing$:
  pick $(i,j) \in A$ with the maximum value of $a_{i,j}$
  \# Add (i,j) to the selection, then remove it (and all adjacent intersections) from A.
  $S \leftarrow S \cup \{(i,j)\}$
  $A \leftarrow A - \{(i,j), (i-1,j), (i+1,j), (i,j-1), (i,j+1)\}$
return S

Counter-Example:

For the input on the right (where we've indicated the number of accidents prevented for each intersection), the algorithm returns the intersections "5" and "0," for a total of 5 accidents prevented. However, picking the other two intersections would prevent a total of 8 accidents instead.



(b) Note that the input to the problem can be represented as an undirected graph G, and valid selections of intersections are the same as independent sets in G.

Let $A(G)$ be the smallest number of accidents prevented over all independent sets returned by the greedy algorithm, and $M(G)$ be the *maximum* number of accidents prevented over *all* independent sets of G. We prove that for all inputs G, $A(G) \geqslant M(G)/4$—and that for at least one input G_0, $A(G_0) = M(G_0)/4$, showing that the approximation ratio for our algorithm is equal to 4.

Let S be any independent set returned by the algorithm, and T be any independent set with the maximum number of accidents prevented in G. For all $v \in T$, if $v \notin S$, then there is some intersection $v' \in S$ such that $(v', v) \in E$—otherwise, v could be added to S. Moreover, $a_{v'} \geqslant a_v$ because when v was removed from G by the algorithm, it was because of some adjacent intersection v' being added to S, which means that at that point, v' had the largest profit among all remaining intersections, including v.

Since no intersection in S has more than 4 neighbours, for all $v \in S$, there are at most 4 intersections $v_1, v_2, v_3, v_4 \in T$ such that $a_v \geqslant a_{v_1}$, $a_v \geqslant a_{v_2}$, $a_v \geqslant a_{v_3}$, $a_v \geqslant a_{v_4}$. In other words, for all $v \in S$, $4a_v \geqslant a_{v_1} + a_{v_2} + a_{v_3} + a_{v_4}$ (in the worst case) to "cover" all intersections in T. Hence, $4\sum_{v \in S} a_v \geqslant \sum_{u \in T} a_u$, i.e., $A(G) \geqslant M(G)/4$, as desired.

To show that $A(G)$ can be equal to $M(G)/4$, consider the input below. The algorithm will select the intersections that prevent $(p+1)+0+0+0+0 = p+1$ accidents while the maximum number of accidents prevented is obtained by picking the intersections $p+p+p+p = 4p$. This is not quite the desired factor of 4, though it can be made arbitrarily close to it by picking p large enough. To get a factor of 4 exactly, simply set the number of accidents prevented for the middle intersection to p. Then, the selection returned by the algorithm is no longer *guaranteed* to be sub-optimal, but it is *possible* that the algorithm will select the middle "intersection" first, and end up with a solution whose value is exactly 1/4 of the optimum. (Note that the size of this input can be made arbitrarily large by repeating the pattern multiple times, separated by rows and columns of intersections with 0 accidents prevented.)