A sidebar indicates the "answer" for each step — it is content that is part of following the algorithm development process, separate from commentary about the process or the solution itself.

Assigning Compatible Groups

Specifications. State complete specifications for the problem. What is the problem? What do you start with (input) and what is the end result (output)? What are the legal input instances and the required output for each? For optimization problems, distinguish between legal solutions and optimal ones.

In a group of people, it is to be expected that some of them may not want to work with each other. Assuming that each person has at most d other people that they don't want to work with, divide the people into d+1 groups so that everyone is in exactly one group and no one is in a group with someone they don't want to work with.

Input: n people, and for each person, up to d other people they don't want to work with

Output: membership for each of the d+1 groups, or a labeling of the group each person belongs to

Legal solution: each person is in exactly one group, no one is in a group with someone they don't want to work with, and there aren't more than d+1 groups

Examples. If needed, give examples (specific inputs and the corresponding outputs) of typical and special cases to clarify the specifications.

(This example / clarification of the specifications comes about after the question of whether conflicts are mutual came up when considering special cases.)

Conflicts don't need to be mutual (person A can not want to work with person B without person B saying the same about person A), but each person can't be involved in more than d conflicts in any direction. The following is *not* a legal input because while each person only has one person they don't want to work with, each is involved in two conflicts: let $d = 1$ and $n = 3$, where person A doesn't want to work with B, B doesn't want to work with C, and C doesn't want to work with A. Three groups are required, but $d + 1 = 2$.

Targets. What are the time and space requirements for your solution?

Tactics. The time and space constraints may narrow down the algorithmic options and/or may guide you in particular directions. Consider both things you can and can't do.

Approaches. Consider specific iterative patterns — what would they look like if applied to this problem?

The input includes the people and the output involves the groups.

Process input: for each person, assign them to a group.

Produce output: for each group, assign its members.

Narrowing the search space isn't applicable, as this isn't a search problem.

Main steps. This is the core of the algorithm — the main loop, focusing on the loop body. What's being repeated?

Both process input and produce output approaches are plausible here, so pick whichever seems easiest to figure out as a starting point. With produce output, there's the additional question of when to stop assigning people to one group and move on to the next, while process input seems very straightforward — just (somehow) decide on a group for each person. If it is a tossup, go with process input. But the important thing here to pick something to try — you don't generally know in advance which approach will work (and both may be possible and lead to different algorithms), so just back up and try something else if you hit a dead end rather than worrying about avoiding the dead end in the first place.

So let's go with process input. How to assign a person to a group? Start simple and don't discard things that seem likely to be correct because they don't seem efficient enough — how to efficiently implement a particular idea is a separate task, and it may be possible to find a more efficient implementation than it initially appears. (Or maybe a different algorithmic approach *is* needed, but having an algorithm gives you a target to beat.)

```
for each person
  assign the person to a group that doesn't have someone they don't want to work with
```

Exit condition. Identify when the loop ends.

For each person — the loop ends when everyone has been assigned a group.

Setup. Whatever must happen before the loop starts (initialization, etc).

Initialize d+1 empty groups.

Wrapup. Whatever must happen after the loop ends to produce the final solution.

Nothing — the assignment of people to groups is the desired result.

Special cases. Make sure the algorithm works for all legal inputs — revise the previous steps to add handling for special cases as needed.

Is not wanting to work with someone mutual? That is, if person A doesn't want to work with person B, does person B also not want to work with person A? A small adjustment to the wording of the main steps can accommodate non-mutual preferences:

```
for each person
  assign the person to a group where there's no one they don't want to work
   with and no one doesn't want to work with them
```

Is this always possible with only d+1 groups? That's a correctness question and will be considered there.

Can there be fewer than d+1 groups? If exactly d+1 groups are needed, then a wrapup step can be added to split groups until there are d+1 groups.

Algorithm. Assemble the algorithm from the previous steps and state it.

`group(people,prefs)` —

Input: n people and for each person, up to d other people they don't want to work with

Output: the group assigned to each person

Legal solution: each person is in exactly one group, no one is in a group with someone they don't want to work with, and there at most d+1 groups

```
initialize d+1 empty groups
for each person
  assign the person to a group where there’s no one they don’t want to work
   with and no one doesn’t want to work with them
repeat until there are d+1 groups
  split a group with at least two people into two groups
```

Termination. Show that the loop eventually terminates.

These elements come from the process input pattern.

Measure of progress. The measure of progress is the number of people assigned to a group.

Making progress. Each iteration assigns a new person to a group, increasing the number of people assigned to a group by 1.

Reaching the end. The number of people assigned to groups keeps increasing, so eventually everyone has been assigned to a group.

Correctness. Show that the algorithm is correct.

Loop invariant. A loop invariant is a boolean statement about the state at the start of the loop body. It purpose is to ensure that the algorithm remains on track towards the desired solution.

This isn’t an optimization problem, so try a direct statement. For process input, this is to state that the solution so far is correct.

For the k people assigned to groups so far, no one is in more than one group, no one is in a group with someone they don’t want to work with, and there are at most d+1 groups.

Establish the loop invariant. Explain why the invariant is true after the setup and before the first iteration of the loop. Assuming the invariant is true before the first iteration, explain why it is still true after the first iteration and before the second.

For $k = 0$ (before the first iteration): the setup initializes d+1 empty groups. No one has been assigned to a group, so no one can be in more than one group or in a group with someone they don’t want to work with. And there aren’t more than d+1 groups because only that many groups were initialized.

For $k = 1$ (after the first iteration / before the second iteration): the loop body assigns the person to a single group and, since all groups are initially empty, this person is the first one in the group so that can’t be any conflicts. Also, since $d \geq 0$, $d+1 \geq 1$ so there’s at least one group for this person to be added to.

Maintain loop invariant. Assume that the loop invariant is true at the start of an iteration, and explain why the invariant is still true at the end of that iteration.

Assume that the first k people have been assigned to groups so that each person is assigned to exactly one group, no one is in a group with someone they don’t want to work with, and there are at most d+1 groups.

Show that the next iteration legally assigns person k+1 to a group:

The loop goes through people one at a time, so no one is assigned to more than one group.

Let g be the group that person $k + 1$ is assigned to. There aren’t any conflicts amongst people already in g (by the loop invariant), nor are there conflicts between person $k + 1$ and people in g (in either direction) — the algorithm assigns the current person to a group where there aren't conflicts.

The last piece is whether it is possible to find a group for person k+1 amongst the d+1 groups. (If not, more than d+1 groups would be needed.) Since there are at most d people that person k+1 doesn't want to work with, at most d different groups can contain those people so there has to be at least one group that person k+1 *doesn't* have a conflict with. But what if "doesn't want to work with" isn't necessarily mutual? Could there be someone in that final group that doesn't want to work with person k+1? Yes, for example: let $d = 1$ and $n = 3$, where person A doesn't want to work with B, B doesn't want to work with C, and C doesn't want to work with A. A and B must go into separate groups but then there's no where for C to go because C can't go with A and B can't go with C. So in order for there to be an assignment with at most d+1 groups, one person can't be involved in more than d conflicts regardless of the direction. So, having determined that person k+1 can't be involved in more than d conflicts, there's always at least one group that person k+1 can be added to without a conflict.

Final answer. Explain why, with the loop invariant being true and the exit condition being false when the loop completes, the wrapup steps lead to a correct result.

The loop invariant is that the first k people have been assigned to groups so that each person is assigned to exactly one group, no one is in a group with someone they don't want to work with, and there are at most d+1 groups.

The exit condition is that $k = n$ — all n people have been assigned a group. So that means that after the first $k = n$ people (i.e. all of them) have been assigned to groups so that each person is assigned to exactly one group, no one is in a group with someone they don't want to work with, and there are at most d+1 groups.

Finally, the wrapup step splits groups if exactly d+1 groups are required and fewer than d+1 have been found. This doesn't assign a second group to anyone, it just changes the one group assigned, so each person still is assigned to exactly one group. Splitting groups doesn't put people together that weren't already together, so it doesn't introduce a situation where two people who don't want to work together are put together — if there's a conflict, it had to have already existsed. Finally, we don't end up with more than d+1 groups because we stop splitting when there are d+1 groups.

So, the final result is a legal solution.

Implementation. Identify data structures and, as necessary, specific implementations of those data structures to efficiently support the algorithm. Also fill in any algorithmic steps that haven't been specified.

"Assign the person to a group where there's no one they don't want to work with and no one doesn't want to work with them" is clear enough a step to determine correctness, but more about how it is carried out is needed to determine efficiency.

"Assign the person to a group where there's no one they don't want to work with and no one doesn't want to work with them" could be addressed by going through each group in turn to determine whether there's a person in that group that either doesn't want to work with the new person or who the new person doesn't want to work with. With O(1)-per-element traversal of groups and group members (possible with an array indexed by group number and containing a List of members of each group) and O(1) lookup for "is there a conflict between A and B" (which can be done with a Map of Sets, where the Map's keys are people and the Sets are those that each person doesn't want to work with), we have O(k) to find a group for person k+1.

Time and space. Assess the running time and space requirements of the algorithm given the implementation identified.

- Initialize $d+1$ empty groups — an array of empty Lists. O(d)
- Build a Map of Sets with the "don't want to work with" info — assuming we can traverse the information provided in O(1) per element, this is O(nd) (n people, each with d conflicts) because inserting into a Map of Sets is O(1).
- "For each person" assumes an O(1)-per-element iteration through all of the elements. This is common.
- Determining which group to add a person to — O(k) with the conflict information stored in a Map of Sets.
- Adding a person to a group — adding to a List in any position. O(1)
- Splitting groups if there are fewer than $d+1$ groups — there aren't any rules about group size, so splitting off just one person is sufficient. Removing one element from a List and adding it to a new List is O(1), and this doesn't need to be done more than d+1 times for a total of O(d) work.

The setup time is O(nd), the main loop repeats n times and takes O(k) time per iteration where $k = 0..n-1$, and the wrapup is O(d) for a total of $O(nd + n^2 + d) = O(n^2)$.

Is this good? The slow part is taking O(k) time to find a group for each person. Can this be sped up?