You may work in groups of up to 4 students. Be sure to include your name and student number with your assignment. If your handwriting is possibly illegible, be sure to hand in your assignment in some typed form. Partners should only submit one copy to Markus.

When using results seen in class or the book do not rewrite the code. Instead just talk about how you use or modify these items to answer the questions.

1. For this question we have a set of people $X = \{x_1, \ldots, x_n\}$ who wish to attend a show at the New York Fashion Week. Because fashion is weird the people in $X$ have each decided to attend wearing an outfit comprised of various allergenic materials. Formally each person in $X$ has an outfit comprised of materials from the set $M = \{m_1, \ldots, m_k\}$ (their outfit may contain any subset of the $M$ materials) and a set of allergies also a subset of $M$. You may assume that the people in $X$ are studious and have each compiled a list $L_i \subseteq X$ of the people who are wearing outfits that will cause $x_i$ to have an allergic reaction (as the update below says $L_i$ now also contains people that are allergic to the clothes person $x_i$ is wearing).

The seating area for the show is split into two areas (split by the runway) call these $A_1$ and $A_2$. A person $x_i$ will have an allergic reaction iff they are seated in the same area as a person in $L_i$. You may assume each area is very large and can seat all of $X$ if needed.

Your job is to determine if it is possible to seat all of the people in $X$ without causing an allergic reaction, and if so what that seating is. You are given the set of people $X$ and the self compiled lists.

How will you do this? How long will your algorithm run for? Show that it is correct. If your algorithm runs too slowly you will not receive any marks.

Update: Because the people at NYFW are so studious they have converted their directed graph into an undirected one. That is for a person $x_i$ their list $L_i$ now contain a person $x_j$ if we cannot seat $x_i$ and $x_j$ together. Before $x_j \in L_i$ meant that $x_i$ would be endangered by $x_j$. Now it means $x_i$ would be endangered by $x_j$ or $x_i$ endangers $x_j$ or both. So now all the directed edges have become undirected edges. Furthermore you can assume the graph is connected.

Both of these changes make the problem easier to solve for you.

Update^2: The material set $M$ isn’t actually that useful for your solution. My solution didn’t make use of it (although I’m sure some solutions could). I think anything useful you could derive from it should be contained in the lists $L_i$.

Hint(s):
(a) First this is a graph problem, it’s pretty much in an adjacency list format already, treat it as such.

(b) To solve this you will need to make a minor modification to a graph algorithm we have seen in class.

(c) You need to show two things. First, briefly argue that if there is a valid seating arrangement your algorithm will report it. Second, briefly argue if there is no valid arrangement your algorithm will report this.

(d) Hint: For the second part you may use the fact that a valid seating arrangement exists iff there is no cycle of odd length.

An odd length cycle is a sequence of edges $C = (x_1, x_2), (x_2, x_3), \ldots, (x_{k-1}, x_k), (x_k, x_1)$ where $|C|$ is odd.
2. Consider the following two logical statements:

Statement 1: Statement 2 is True.

Statement 2: Statement 1 is False.

Obviously these two statements must form a contradiction. If Statement 1 is True we know Statement 2 must be True. But if Statement 2 is True that means Statement 1 is False. Thus we get that Statement 1 being True implies Statement 1 is False. You can also determine that if Statement 1 is False then it must be that Statement 1 is also True.

Given a sequence of $n$ statements, each of the form:

Statement i: Statement j is True (for $i \neq j$)

or

Statement i: Statement j is False (for $i \neq j$)

Determine if the statements form a contradiction. To do this you will need to formulate your problem as a graph problem. Carefully define what your nodes are and what your edges are. Prove your algorithm is correct. What is the runtime of your algorithm. If your algorithm runs too slowly you will not get any marks.

How to solve this:

(a) Convert Statement i: Statement j is True (for $i \neq j$) into its logical form. Hint it expresses something stronger than $i \Rightarrow j$. You should do the same for saying $j$ is False.

(b) If a chain of statements 1, ..., $n$ form a contradiction what does this mean from a logical perspective? Use your part (a) here.

(c) Convert the chain of statements into a directed graph, what are the nodes and edges? Hint a statement can be True or False how will you express this?

(d) Identify what it means for your graph to have a contradiction. For this you should translate your part (b) into a graph property.

(e) Show how one can test for the property in part (d) using an algorithm we have seen in class (hint you will need to modify this algorithm slightly).

To prove correctness briefly argue why if there is a contradiction in the graph your algorithm will find it (you will find it useful to use some theorems we saw that are related to the algorithm). For the running time report the time needed to convert the statements into graph form and the time to test for a contradiction.
3. A Mega Stack is a sequence of stacks, each with twice the storage of the previous. That is we have stacks $S_1, S_2, \ldots, S_m$ where $S_1$ can hold 1 element, $S_2$ can hold 2 element and $S_i$ can hold $2^{i-1}$ elements.

The only operation that is supported is MegaPush($x$). This simply pushes $x$ onto $S_1$, if $S_1$ is at capacity we first pop the element out of it to make space for $x$. We then push the element we removed from $S_1$ onto $S_2$. If $S_2$ is at capacity we repeat the process (pop the two elements on it and try to push them to $S_3$). In general when we try to push an element onto a full stack $S_i$ we first pop all of the elements from $S_i$ and try and push them onto $S_{i+1}$, we recursively repeat this process until all elements have been successfully pushed onto a stack.

The time to push or pop a single element is simply $O(1)$.

(a) Consider a Mega Stack with $n$ elements. Say we execute a MegaPush($x$), what is the worst case time of this operation. Do not use asymptotic notation, provide an exact number.

(b) Show the amortized cost of $n$ MegaPush($x$) operations (starting from an empty structure) is $O(\log(n))$.

Hint: While solvable using both the aggregate and accounting method, the accounting method provides a much easier solution.

4. Consider the following four definitions of the cost $\delta(p)$ of a path $p$. For this let the weight of an edge $(u, v)$ be $w_{(u,v)}$ and let $|p|$ be the number of edges in the path.

(a) $\delta(p) = \sum_{(u,v) \in p} w_{(u,v)}$
(b) $\delta(p) = \min_{(u,v) \in p} w_{(u,v)}$
(c) $\delta(p) = \max_{(u,v) \in p} w_{(u,v)}$
(d) $\delta(p) = (\sum_{(u,v) \in p} w_{(u,v)})/|p|$

Recall the Dijkstra’s Algorithm works by keeping a set $S$ of nodes (initially just the source node $s$) it has found the optimal path to. It then grows this set by one node by taking the node that is one step away from $S$ with the lowest cost path to it from $s$ via the nodes of $S$. For definition (a) above (the normal definition of the cost of a path) the algorithm is optimal, that is it always finds the lowest cost path (when costs are sums) to a node.

For the other three definitions determine if the algorithm is still optimal. For example definition (b) says the cost of the path is the cost of the smallest edge on that path. So now the algorithm will always pick the node that optimizes this new cost function at each step. You must determine if optimizing this new cost function at each step is indeed the correct way to find the lowest cost path (where the cost is the lowest weight edge on a path) to each node.

For the costs functions for which the algorithm is still optimal briefly explain why this is. For the ones that are no longer optimal give a graph for which the algorithm fails to find the optimal path. Indicate where the algorithm fails on this graph, always label your source node $s$. Assume we do not have negative edge weights.
5. (a) Consider the Binomial Heap structure we studied in class. Let us form two max heaps in the following way.
Heap 1 is built by inserting (in the following order) 5, 2, 6.
Heap 2 is built by inserting (in the following order) 7, 4, 8, 9.
Show the resulting heaps.
Next we perform a Union on the two heaps. Show the resulting heap.
Next we perform an extract max on the heap, show the resulting heap.
(b) Consider a Max Binomial Heap with 374 nodes.
What is the maximum number of nodes we need to search to find the maximum element? Provide exact numbers. Do not assume we are keeping a pointer to the largest element.
6. Consider the following Graph:

Figure 1: The graph $G$

For each of the following show the MST it generates and the order the edges are added in.

- Kruskal’s algorithm.
- Prim’s algorithm (with $F$ as the source node).