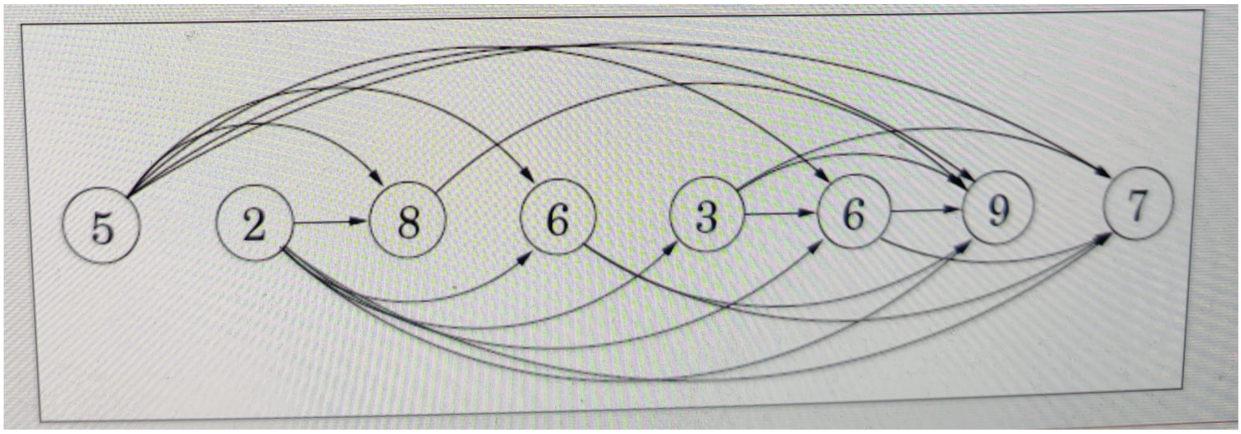
Longest increasing subsequence

A subsequence of a given sequence is a sequence that can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements. A longest increasing subsequence is a special subsequence where the elements of the subsequence are strictly increasing. In this problem we need to find the length of the longest such increasing subsequence. Lets try to solve this using the dynamic programming strategy, we will first try to express the problem as a DAG of the subproblems. Lets try to build the DAG on this rule: Let all the elements of the sequence be the nodes of the graph. Lets then try adding edges between node i and j if and only if $i < j$ and $a_i < a_j$. The DAG for a sequence of $\{5, 2, 8, 6, 3, 6, 9, 7\}$ would be something like this :



We will try to compute the longest increasing subsequence ending at every node, the answer would thus be the maximum of all the subsequences ending at node i from 1 to length of the sequence. The toposorted order in this case would just be the order of the elements in the sequence given. To formalise the algorithm, we can say that an LIS that ends at a node i can be represented as $dp[i]$ and the answer of $dp[i]$ would be 1 plus the maximum of all $dp[j]$ such that $j < i$ and $a_j < a_i$. After getting $dp[i]$ for all i from 1 to length of the sequence. The LIS would be the maximum of all $dp[i]$.

Let $dp[i]$ represent the length of the longest path / increasing subsequence ending at node i.

Helper code :

```
int lis(vector<int> const& a) {
    int n = a.size();
    vector<int> dp(n, 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                dp[i] = max(dp[i], dp[j] + 1);
        }
    }

    int ans = dp[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, dp[i]);
    }
    return ans;
}
```

This algorithm can actually be improved more to $O(nlogn)$ complexity, an outline for this is as follows :

The improvement happens in the second loop, we can try maintaining another array, c such that $c[i]$ always stores the last element value in an LIS of length i until the processed index, we can thus just binary search for this i value with current element value and determine the LIS value at this index.