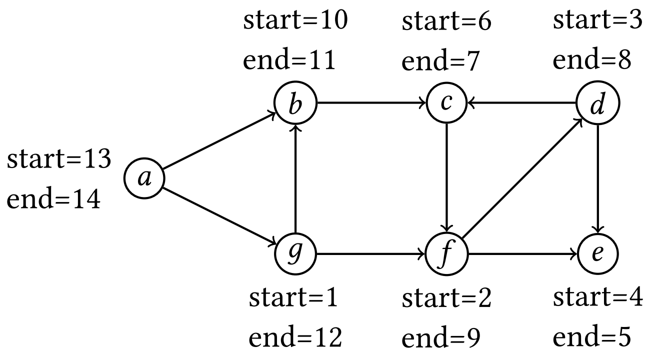
1. **(4 marks)** Consider the following graph with the start and end times for a DFS marked next to it. Classify all the edges as tree, forward, backward or cross.



Solution:

- Tree edges: (g, f), (f, d), (d, e), (d, c), (g, b)
- Back edges: (c, f)
- Forward edges: (f, e)
- Cross edges: (b, c), (a, g), (a, b)

2. **(4 marks)** Your friend has given you two recursive algorithms for a computational problem you are trying to solve.

- The first algorithm divides the instance of size n into two instances, one of size $3n/4$ and another of size $7n/8$, and combines the solutions in $\Theta(n)$ time.
- The second algorithm divides the instance of size n into two instances, one of size $n/6$ and another of size $n/4$ and combines the solutions in time $\Theta(n)$

Which is the asymptotically better algorithm, and why?

Solution:

- The first recurrence is

$$T(n) = T(6n/8) + T(7n/8) + \Theta(n).$$

The total time taken in level i is at most $\left(\frac{13}{8}\right)^i n$; you can show this by induction. So the total running time for this algorithm is

$$T(n) = n \sum_{i=0}^{\log_{8/7} n} \left(\frac{13}{8}\right)^i .$$

The total time taken is $n^{1+\epsilon}$

- The second recurrence is

$$T(n) = T(n/6) + T(n/4) + \Theta(n).$$

The total time taken in level i is at most $\left(\frac{5}{12}\right)^i n$. The total time taken is at most n.

The second algorithm is better.

3. **(4 marks)** Suppose you are given an array $A[1, 2, \ldots, n]$ of distinct numbers that represent the elements of a binary search tree in *postorder*. Give an efficient algorithm to count the number of elements in the left and right subtrees of the root of this binary search tree.

Solution: Since the array is given in postorder, the element $A[n]$ is the root of the binary search tree. There is a position i in the array such that $A[1, 2, \ldots, i]$ are all numbers strictly less than $A[n]$, and all the numbers $A[i+1, \ldots, n-1]$ are all numbers strictly larger than $A[n]$. We need to find the position i, and we can get the number of elements in the right and left subtrees. For this, we just need to perform a binary search (even though the array is not sorted). For each k, if $A[k] > A[n]$, then the position i must lie before k, otherwise at k or after it. Thus we can get an $O(\log n)$-time algorithm.

4. **(4 marks)** In the game of *Behind Enemy Lines*, you are given a battlefield represented as an $n \times n$ board, where each cell contains either a number, a mine, or a star (which denotes your destination behind enemy lines). There is exactly one cell with a star. You start the game from some cell other than a mine or a star, and try to reach the cell with the star in a finite number of steps.

If you are in a cell with a number k, you must move k steps to the right, left, top or bottom with the following constraints.

- If while moving the k steps, you move over a cell with a mine, you will be killed.
- If after moving k steps, you are next to a cell with a mine, you will be killed.
- If there are no k cells left in a particular direction, then you cannot move in that direction.

Give an efficient algorithm to check if there is a cell in the board such that you will never reach behind enemy lines if you start from that cell.

Solution: Create a graph G where the vertices are the cells (i, j). If the number in cell $(i, j) = k$, create edges as follows:

- If there is no mine between the cells (i, j) to $(i + k, j)$, and there is no mine in $(i + k + 1, j)$, $(i + k, j - 1)$ or $(i + k, j + 1)$ put an edge from (i, j) to $(i + k, j)$
- If there is no mine between the cells (i, j) to $(i - k, j)$, and there is no mine in $(i - k - 1, j)$, $(i - k, j - 1)$ or $(i - k, j + 1)$ put an edge from (i, j) to $(i - k, j)$.

- If there is no mine between the cells (i, j) to $(i, j + k)$, and there is no mine in $(i + 1, j + k)$, $(i - 1, j + k)$ or $(i, j + k + 1)$ put an edge from (i, j) to $(i, j + k)$
- If there is no mine between the cells (i, j) to $(i, j - k)$, and there is no mine in $(i + 1, j - k)$, $(i - 1, j - k)$ or $(i, j - k - 1)$ put an edge from (i, j) to $(i, j - k)$

Now, if there is a path from a cell (i, j) to the cell with a star, it must be a valid path. To check if there is a cell from which there is no path to reach behind enemy lines do the following: find $\text{rev}(G)$ and find $\text{reach}(star)$ in $\text{rev}(G)$. No vertex in $V - \text{reach}(star)$ will have a path to behind enemy lines.

5. **(4 marks)** Karl has recently received some royalty for a book that he wrote and wants to organize a party for his friends. He has n friends, but not all of his friends are friends amongst themselves. If two of Karl's friends, Max and Georg are not friends amongst themselves, then both will not attend the party together, only at most one will. But, if Max and Georg are friends amongst themselves, then one will not attend the party without the other. Karl is particular that his friend Friedrich attends the party. He wants to know if it is possible to organize a party with these constraints; he does not care how many people actually come to the party apart from Friedrich. Design an efficient algorithm to help Karl decide if this party can be arranged.

Solution: We can write this as an instance of the 2-SAT problem that we saw in the problem set. We have boolean variables f_i for each friend of Karl, and let f_1 denote the variable corresponding to Friedrich. Now $f_i = 1$ denotes if i is coming to the party. The constraints can be expressed as follows:

- i and j are not friends: $(f_i \Rightarrow \overline{f}_j)$ (both cannot come together)
- i and j are friends: $f_i \Leftrightarrow f_j$

Once you construct this formula, set $f_1 = 1$ (since Friedrich must come), and get a new 2-SAT formula. Check if this new formula is satisfiable using the algorithm in the problem set.