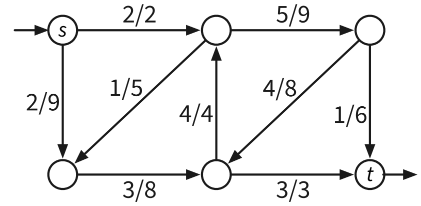
1. The figure below shows a flow network G, along with an (s,t)-flow f that is *not* a maximum flow. ***Clearly*** indicate the following structures in G:



   (a) An augmenting path for f.
   (b) The result of augmenting f along that path.
   (c) A maximum (s,t)-flow in G.
   (d) A minimum (s,t)-cut in G.

   (The answer booklet contains several drawings of G for you to annotate.)

2. A sequence of numbers $x_1, x_2, \ldots, x_\ell$ is ***sort-of-increasing*** if each element (except the first two) is larger than the *average* of the two previous elements; that is, for every index $i > 2$, we have $2x_i > x_{i-1} + x_{i-2}$. Describe an efficient algorithm to compute the length of the longest sort-of-increasing subsequence of a given array $A[1..n]$ of numbers.

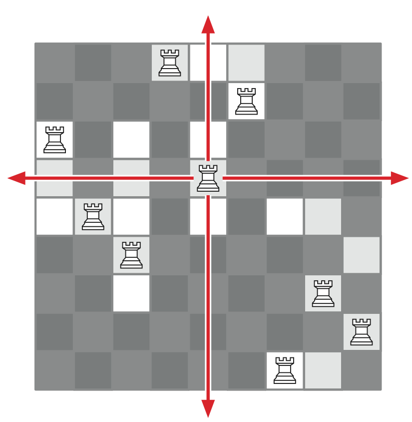
   For example, given the input array

   $$[\underline{3}, \underline{1}, \underline{4}, 1, \underline{5}, 9, 2, 6, \underline{5}, 3, 5, \underline{8}, 9, \underline{7}, 9, 3, 2, 3, \underline{8}, 4],$$

   your algorithm should return the integer 8, which is the length of the sort-of-increasing subsequence $\langle 3, 1, 4, 5, 5, 8, 7, 8 \rangle$.

3. Suppose you are given a chessboard with certain squares removed, represented as a two-dimensional boolean array *Legal*$[1..n, 1..n]$. A ***rook*** is a chess piece that attacks every square in the same row or column; that is, a rook on square (i,j) attacks every square of the form (i,k) or (k,j). Describe an algorithm to places as many rooks on the board as possible, each on a legal square, so that no two rooks attack each other.

   For example, given the 9×9 board shown below, your algorithm should return the integer 9. *The correct output is not always equal to the height of the board.*



4. An ***open-address*** hash table is implemented as an array, where wach entry either contains one hashed item or is empty. The hash function defines, for each item x, a ***probe sequence*** $h(x,0), h(x,1), \ldots$ of table locations. To insert item x, we examine locations in the hash table specified by the probe sequence of x until we find an empty location; then we insert x at that empty location.

```
INSERT(x):
  for i ← 0 to ∞
      j ← h(x, i)
      if T[j] is empty        ⟨⟨"probe j"⟩⟩
          T[j] ← x
          return j
```

For example, consider a table of size 6, with items already stored at indices 1, 4, and 6, as shown below:

a			c		b

If the probe sequence of a new item x is $1, 4, 4, 3, 5, 6, 2, \ldots$, then inserting x will require exactly four probes.

Suppose we insert a sequence of n items into an initially empty hash table of size $2n$, using an *ideal random* open-address hash function. That is, for all items x and all indices i, the addresses $h(x,i)$ are uniformly distributed in $\{1, 2, \ldots, 2n\}$ and *fully independent*.

(a) ***Prove*** that for all $1 \le k \le n$ and for all $m \ge 0$, the kth insertion requires more than m probes with probability at most 2^{-m}. *[Hint: Show that with probability at least* $1/2$*, each probe finds an empty location.]*

(b) ***Prove*** that for all $1 \le k \le n$, the kth insertion requires more than $2 \log_2 n$ probes with probability at most $1/n^2$. *[Hint: Use part (a).]*

(c) ***Prove*** that the maximum number of probes over all n insertions is more than $2 \log_2 n$ with probability at most $1/n$. *[Hint: Use part (b) and the union bound.]*

(d) What is the *exact* expected *total* number of probes for all n insertions? (A tight $O(\cdot)$ bound is worth significant partial credit.)

[Hint: You may be able to solve each part by assuming earlier parts.]

5. Suppose you are given a directed graph $G = (V, E)$ with positive integer edge capacities $c\colon E \to \mathbb{Z}^+$ and an integer maximum flow $f^*\colon E \to \mathbb{Z}$ from some vertex s to some other vertex t in G. Describe and analyze efficient algorithms for the following operations:

(a) INCREMENT(e): Increase $c(e)$ by 1 and update the maximum flow f^*.

(b) DECREEMENT(e): Decrease $c(e)$ by 1 and update the maximum flow f^*.

Both of your algorithms should be significantly faster than recomputing the maximum flow from scratch.

6. Suppose you are given an $n \times n$ array of 0s and 1s. Each row of the array is colored either red or blue, there are exactly k 1s in each row, and each column j of the matrix has a non-negative weight w_j. Your goal is to choose a subset of the columns that satisfy the following conditions:

   - In each red row, there are *at least one* 1s in the chosen columns.
   - In each blue row, there is *at least two* 1s in the chosen columns.
   - The sum of the weights of the chosen columns is as small as possible.

   Solving this problem *exactly* is NP-hard.

   (a) Write an integer linear program that ***exactly*** captures this problem. In particular, each solution of the integer linear program must describe a set of columns, and each set of columns must correspond to a solution of your integer linear program.

   You do not need to prove that your integer linear program is correct, but for partial credit, some justification is recommended.

   (b) Describe and analyze an efficient $(k/2)$-approximation algorithm for this problem. Remember to ***prove*** that your algorithm returns a valid solution, and ***prove*** that it achieves an approximation ratio of $k/2$. *[Hint: Use LP relaxation and rounding.]*

   Part (b) was broken; the correct approximation ratio from LP rounding is actually k, not $k/2$. **Everyone received full credit for this subproblem.**