Problem 1 (0 points).

Assume that a simple regular language supports $*$ and ?, where ? matches any single character, and $*$ matches any multiple characters (can match nothing as well). Given a string S over alphabest Σ and a simple regular expression E (a string over $\Sigma \cup \{*\} \cup \{?\}$), describe an algorithm that decides whether S matches E.

Solution. Let $F(i,j)$ denote whether $E[1 \cdots i]$ matches $S[1 \cdots j]$, $1 \leq i \leq |E|$, $1 \leq j \leq |S|$. The recursion considers the following three cases:

1. If $E[i] \in \Sigma$, then $F(i,j) = 1$ if and only if $E[i] = S[j]$ and $F(i-1, j-1) = 1$. Formally, $F(i,j) = \delta(E[i] = S[j])$ AND $F(i-1, j-1)$, where $\delta(E[i] = S[j]) = 1$ if $E[i] = S[j]$ and 0 otherwise.
2. If $E[i] =?$, then $F(i,j) = F(i-1, j-1)$.
3. If $E[i] = *$, then $F(i,j) = 1$ if $F(i-1,j) = 1$ (i.e., $*$ matches nothing of S), or $F(i, j-1) = 1$ (i.e., $*$ matches $S[j]$ now and can match additional characters). Formally, $F(i,j) = F(i-1,j)$ OR $F(i,j-1)$.

The *pseudo-code*, which includes three steps, is as follows.

Initialization:

$F(0,0) = 1$
$F(0,i) = 0$, for any $1 \leq i \leq |S|$
$F(i,0) = 1$ if the first i characters of E are all $*$, and $F(i,0) = 0$ otherwise, for any $1 \leq i \leq |S|$

Iteration:

```
for i = 1 ··· |E|:
  for j = 1 ··· |S|:
    if E[i] ∈ Σ: F(i,j) = δ(E[i] = S[j]) AND F(i-1, j-1)
    if E[i] =?: F(i,j) = F(i-1, j-1)
    if E[i] = *: F(i,j) = F(i, j-1) OR F(i-1, j)
  endfor
endfor
```

Termination: $F(|E|, |S|)$ gives whether E matches S.

Running time: $O(|E| \cdot |S|)$.

Problem 2 (0 points).

Given a positive integer X with n digits, try to remove k ($k \leq n$) digits from X so that the number consisting of the remaining digits is as small as possible. You may assume that X has no leading zero and you are not allowed to change the relative order of the digits.

Solution: We may use the idea of "monotonic (increasing) stack". We use a stack to store the candidate digits in the resulting number. We scan X from its highest digit to its lowest digit: when we examine $X[i]$, i.e., the ith digit of X, we keep poping from the stack until the top digit of the stack is no larger than $X[i]$, or the stack is empty, or we have popped k times in total; we then push $X[i]$ to the stack. If there is more than $n-k$ numbers in the stack, we continuously pop until there is exactly $n-k$ numbers in the stack. The digits in the stack consist our final number.

Correctness. The algorithm keeps three invariants. First, all digits in the stack are non-decreasing (from bottom to top); this is guaranteed by the condition when new digit is pushed to the stack. Second, the digits in the stack keeps the original order, which is guaranteed by the property of stack (first-in-first-out). Third, the number of digits in the stack plus the number of un-examined digits is always at least $n-k$; this is again guaranteed by the condition we keep poping.

Based on above invariants, we first show that, when we examin $X[i]$ and the pop operation happens, say p is poped, then p cannot be in the resulting number. Suppose conversely that p is in the resulting (smallest) number $X' = x_1 x_2 \cdots p \cdots x_m$. If $X[i]$ is not in X', then we can replace p with $X[i]$. Because $p > X[i]$, we get a smaller number, a contradiction. If $X[i]$ is in X', then we can remove all numbers from p to $X[i]$ (not include $X[i]$) from it and add back the same number of digits after $X[i]$ that are not in X'– such digits exists, because of the 3rd invariant above; this will again give us a smaller number, a contradiction. This proves that during the examining process, it is optimal to pop. After all digits are examined, all digits in the stack are sorted and kept their original order (first two invariants above), and therefore it is again optimal to keep the bottom $n-k$ digits.

Problem 3 (0 points).

Recall from the class we talked about the sequence alignment problem where we use dynamic programming to calculate the edit distance between two strings. Given two strings A and B have the same length of n characters, and a threshold k, $0 < k < n$, design an algorithm to check whether the edit distance between A and B is less than k. Your algorithm should run in $O(nk)$ time.

Solution: Instead of calculating the value for *all* the cells in the $n \times n$ dynamic programming matrix M, here, we only need to calculate the value in the diagonal with the width of $2k-1$ (Figure 1) using exactly the same recursion we introduced during lecture. We can prove that, $M[n,n] < k$ if and only if the edit distance between A and B is less than k.

Proof. Suppose that $M[n,n] < k$, then we can find a path from $(0,0)$ to (n,n) within the diagonal area, whose corresponding alignment gives an edit distance of $M[n,n]$ (recall that there is a one-to-one correspondence between an alignment and such a path). On the other hand, if the edit distance between A and B is less than k, then its corresponding path must be completely within the diagonal area (with width of $2k-1$). This is because any cell, say (i,j), outside the diagonal area indicate the edit distance between $A[1 \cdots i]$ and $B[1 \cdots j]$ is at least k, which implies the edit distance between A and B is at least k as well, as we have $d(A,B) \geq d(A',B')$ for any prefix A' of A and any prefix B' of B.

Problem 4 (0 points).

You are given an undirected connected graph $G = (V,E)$ satisfying that $|E| = |V| + 5$. Each edge $e \in E$ has weight $w(e)$ and you may assume that all weights are distinct. Design an $O(|V|)$ time algorithm to find the Minimum Spanning Tree of G, and prove the correctness of your algorithm.

Solution: Since $|E| = |V| + 5$, we should remove 6 edges to obtain a spanning tree. We can prove that in a cycle the edge with maximum cost cannot appear in the minimum spanning tree (see below; also check the cycle-property in the textbook). Armed with this statement, we can traverse the graph to find a cycle using BFS, and then find and remove the edge with maximum cost in this cycle, which costs $O(|E|)$ time. Repeating this process 6 times will yield a minimum spanning tree. Clearly, the running time of this algorithm is $O(|E|) = O(|V|)$.

Now we prove that the edge with maximum cost in a cycle cannot appear in the minimum spanning tree by contradiction. Suppose that $e = (u,v)$ is the edge with maximum cost in cycle C, and e appears in the

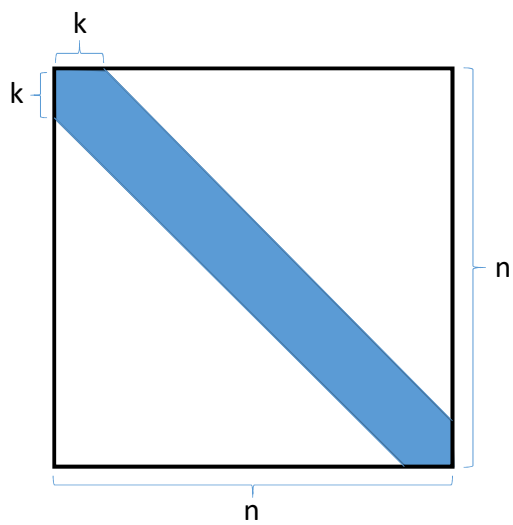

Figure 1: Diagonal area required to fill in the dynamic programming matrix when only edit distance of less than k is considered.

minimum spanning tree T. Now we remove e in T, which will split T into two isolated components, and u and v are in different components. Consider another path in C that connects u and v (not edge e). There must be at least one edge e' that links two nodes which belong to differenct components. We can add e' to T to obtain a spanning tree that has smaller total cost than T, since the cost of e' is smaller than that of e. This contradicts with the assumption that T is the minimum spanning tree.

Problem 5 (0 points).

Let $X = \{a_1, a_2, \cdots, a_n\}$, where $a_i \in \mathbb{R}^d$, $1 \leq i \leq n$. Let S be the set of all subsets of X that are linearly independent. Prove that (X, S) forms a matroid.

Solution: To prove that (X, S) forms a matroid, we need to verify that they satisfy both hereditary property and exchange property.

Let $A \in S$. Let $A_1 \subset A$ be any subset of A. By definition all vectors in A are linearly independent. Then vectors in A_1 are linearly independent too. This prove that (X, S) satisfies the hereditary property.

Let $A \in S$, $B \in S$ and $|A| < |B|$. We need to prove that there exists $x \in B \setminus A$ such that $A \cup \{x\} \in S$. We now prove it by contradiction: suppose conversely that no such x exists, i.e., for any $x \in B \setminus A$, we assume $A \cup \{x\} \notin S$. Let F_A be the linear space spanned by the vectors in A. Let F_B be the linear space spanned by the vectors in B. We have that $\dim(F_A) = |A|$ and $\dim(F_B) = |B|$ as vectors in A are linearly independent and vectors in B are linearly independent. Let $x \in B \setminus A$. The assumption that $A \cup \{x\} \notin S$ implies that $x \in F_A$. Further, for any $x \in B$ we have $x \in F_A$. This therefore gives that $F_B \subset F_A$, which is a contradiction to the fact that $\dim(F_B) = |B| > |A| = \dim(F_A)$.

Remark: This matroid is called "vector matroid", one of the basic examples of matroid. Matroid is a generalization of vector space, and the terminology of "independent subset" used for matroid (elements in S) is borrowed from the "linearly independence" in vector space.