Instructions.

1. Solution may not be submitted by students in pairs.

2. You may submit a pdf of the homework, either printed or hand-written and scanned, as long as it is easily readable.

3. If your solution is illegible not clearly written, it might not be graded.

4. Unless otherwise stated, you should prove the correctness of your answer. A correct answer without justification may be worth less.

5. If you have discussed any problems with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually encouraged. However, you must write your whole solution yourself.

6. Unless otherwise specified, all questions have the same weight.

7. You may refer to data structures or their properties studied in class without having to repeat details, and may reference theorems we have studied without proof. If your answer requires only modifications to one of the algorithms, it is enough to mention the required modifications, and the effect (if any) on the running time and on other operations that the algorithm performs.

8. In general, a complete solution should contain the following parts:

   (a) A high level description of the data structures (if needed). E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...

   (b) A clear description of the main ideas of the algorithm. You may include pseudocode in your solution, but this may not be necessary if your description is clear.

   (c) Proof of correctness (e.g. show that your algorithm always terminates with the desired output).

   (d) A claim about the running time, and a proof showing this claim.
1. We are adding an deleting keys into/from a an array $A[1..m]$. If a new keys $x$ is inserted, and the number of keys $n$ is $\leq m$, then we insert $x$ at $A[j+1]$ where $j$ is the index where the previous key was inserted. Similarly we delete the keys in a reverse order.

If $A[1..m]$ is full when an insert is needed, then we create a new array $A[(1 + \alpha)m]$ and copy the keys from $A$ into the new array. This operation is called expansion. The reverse operation (shrink) takes place when $n < m \beta$. Here $\alpha$ and $\beta$ are parameters given to you. An example might be $\alpha = 1$ and $\beta = 0.25$.

Show that if $0 < \beta < 1$ and $0 < \alpha$, then any sequence of $n$ insertion and deletions (in any arbitrary order) takes $O(n)$ time. Here $\alpha$ and $\beta$ are constants.

Hint: start from the case $\alpha = 1$ and $\beta = 0.25$.

Another hint: Remember that you could inject a constant amount of $$ also for each deletion.

2. Let $G(V, E)$ be a given graph, let $s$ be a vertex in $V$, and and let $k$ be a number (not known to you) with the property that for every pair of vertices $u, v \in V$, the shortest path $u \rightarrow v$ contains at most $k$ vertices.

Suggest a modification of Bellman-Ford algorithm, such that the output is the same as the original algorithm,(the distance $\delta(s, v)$ for every $v \in V$) but the running time is improved to $O(km)$. Prove the correctness of your modification.

3. You are given a directed graph $G(V, E)$ where every vertex $v_i \in V$ is associated with a weight $w_i > 0$. The length of a path is the sum of weights of all vertices along this path. Given $s, t \in V$, suggest an $O((n + m) \log n)$ time algorithm for finding the shortest path from $s$ to $t$.

As usual, $n = |V|$ and $m = |E|$.

4. Suggest an algorithm that receives a graph $G(V, E)$ with weights assigned to its edges. It $G$ is a DAG, the algorithm finds the longest simple path in the graph. If $G$ is not a DAG, the algorithm should report so. Analyze the running time and prove correctness.

5. Let $G(V, E)$ be an undirected graph with positive weights on its edges. Assume that edges are given in an increasing order of weights. So $E = \{e_1 \ldots e_m\}$ where

$$0 < w(e_1) \leq w(e_2) < \cdots < w(e_m).$$

Given a real value $r$, let $G_r$ denote the graph obtained from $G$ be removing from $E$ every edge whose weight is strictly larger than $r$. That is, $G_r$ contains only the edges whose weight is $r$ or smaller than $r$.

For example, $G_{w(e_1) - 0.00001}$ contains no edges, $G_{w(e_2)}$ contains 5 edges, and $G_{w(e_m)}$ contains all the edges of $E$.

We say that $\beta$ is the critical value of $G(V, E)$ if $G_\beta$ is connected, by $G_{\beta-0.00001}$ is not connected.

Suggest an $O(m \log m)$ time algorithm for finding $\beta$. 
6. Your computer needs to run a list of tasks \( T_1 \ldots T_n \). (Jobs) For each task \( T_i \) the executable of the code is given to you. Also provided are a list of pairs \((i, j)\) such that task \( T_j \) needs input from \( T_i \).

Explain how to determine (as efficient as possible) if we could perform these tasks, and none of the tasks has to wait for an input from another task.

7. You are given a terrain. (You know - mountains, valleys etc). To specify a terrain, we give a grid of points - latitude and longitude, and for each point \((x_i, y_j)\) we specify the elevation \( h(x_i, y_j) \) above sea level of this point.

Explain how you’d find, as fast as possible, the most efficient path between every pair of grid points of the terrain. Efficiency here is measured by the energy needed to drive the path. It is computed as follows:

The energy \( w(p, q) \) to drive from \( p \) to a neighboring grid points \( p \) and \( q \) is given by the formula

\[
 w(p, q) = \begin{cases} 
 0.1 \times \text{(Euclidean distance}(p, q) + 2 \times (h(q) - h(p))) & \text{if } h(q) > h(p) \\
 0.1 \times \text{Euclidean distance}(p, q) + 0.5 \times (h(q) - h(p)) & \text{if } h(q) \leq h(p) 
\end{cases}
\]

Where \( h(p) \) is the elevation (height) of \( p \) above sea level. The rational is that if you are using an electric vehicle then you are spending some amount of energy going uphill, but your car uses this some portion of this energy driving downhill, in order to charge its batteries.

The Euclidean distance between \((p.x, p.y)\) and \((q.x, q.y)\) is \( \sqrt{(p.x - q.x)^2 + (p.y - q.y)^2} \).

8. Let \( G(V, E) \) be a graph with positive weight \( w(u, v) \) given for every \((u, e) \in E\). Let \( s \) be a vertex of \( V \). Suggest an \( O(mn) \) time algorithm that computes a matrix \( C[i, j] \) (for \( 1 \leq i, j \leq n \)) such that \( C[i, j] \) is the length of the shortest from \( s \to v_i \) that contains no more than \( j \) edges.

9. Let \( S = \{s_1 \ldots s_n\} \) be a set of vertical segments in the plane. See Figure 1. Each segment \( s_i \) is given by its \( x \)-coordinate \( x_i \) and two values \( y_i, Y_i \) where \((x_i, y_i)\) is the lower point of the segment \( s_i \) and \((x_i, Y_i)\) is the upper point of this segment. Also given to each segment a unique color. Each segment is either blue, orange or black.

You are also given a list \( E \) of pairs of endpoints of the segments \((p_i, q_j)\), such that \((p_i, q_j) \in E\) if and only if \( p_i \) sees \( q_j \). That is, the segment connecting them does not cross any segment of \( S \).

Suggest an algorithm that in \( O(n + m) \) time finds the shortest path that starts at an endpoint of a blue segment, ends at an endpoint of orange segment, and does not cross any other segment (from any color).

Here \( m = |E| \).

Hints: use these stages

(a) assume first that you aim for \( O(m \log n) \) time, and that the start point and end point of the optimal path are given.

(b) Assume multiple endpoints are given, but single point is unique. Aim for \( O(m \log n) \).
(c) This step is tricky. Assume multiple start points, but still aim for $O(m \log n)$.
(d) Use geometric observations (and prove it) to obtain $O(m)$

Figure 1: Left: An example of the input to the problem, and a possible path $\pi$ from a point on a blue segment to a point on an orange segment. This path does not cross any input segment. Right - the same, but now the path crosses one of the segments

10. Suggest a practical approach for answering shortest path queries between street addresses in the USA. Each such query specifies a source address and a destination address. Your goal is to pre-process the road map into a reasonable size data-structure that would enable answering such queries efficiently.

To clarify possible doubt: Assume the number of houses is about $125M$, so a data structure of size proportional to all roughly $(125M)^2$ is not considered reasonable.

There is no single correct solution, but your answer must fit the realistic limitations of small devices.

11. Explain how from the output of Warshall-Floyd algorithm you could deduce if a graph contains a negative cycle.

12. You are given the output of Warshall-Floyd algorithm for a graph $G(V, E)$, and you are also given 2 vertices $v_i, v_j \in V$. Explain how you could find the shortest path $v_i \rightarrow v_j$ in optimal time. Prove the optimality of your proposal.