Design and Analysis of Algorithms

PSet-3

13 November 2025

This is a compilation of solutions for PSet-3 of Design and Analysis of Algorithms class instructed by Prof. Siddharth Pritam. The solutions are of original design.

Table of contents

§1 Problem 1

Exercise

Following set of questions asks you to develop efficient algorithms to find optimal subsequences of various kinds.

(a) Let $A[1..m]$ and $B[1..n]$ be two arbitrary arraB. A common supersequence of A and B is another sequence that contains both A and B as subsequences. Describe an efficient algorithm to compute the length of the shortest common supersequence of A and B.

(b) Call a sequence $X[1..n]$ of numbers bitonic if there is an index i with $1 < i < n$, such that the prefix $X[1..i]$ is increasing and the suffix $X[i..n]$ is decreasing. Describe an efficient algorithm to compute the length of the longest bitonic subsequence of an arbitrary array A of integers.

(c) Call a sequence $X[1..n]$ of numbers oscillating if $X[i] < X[i+1]$ for all even i, and $X[i] > X[i+1]$ for all odd i. Describe an efficient algorithm to compute the length of the longest oscillating subsequence of an arbitrary array A of integers.

(d) Call a sequence $X[1..n]$ of numbers convex if $2X[i] < X[i-1] + X[i+1]$ for all i. Describe an efficient algorithm to compute the length of the longest convex subsequence of an arbitrary array A of integers.

(a)

Solution.

SHORTEST SUPERSEQUENCE

```
1  function SUPER(A, B):
2      n = length(A)
3      m = length(B)
4      dp = array of size (n + 1) × (m + 1)
5      for i from 0 to n:
6          dp[i][0] = i
7      for j from 0 to m:
8          dp[0][j] = j
9      for i from 0 to n:
10         for j from 0 to m:
11             if A[i] == B[j]:
12                 dp[i][j] = 1 + dp[i-1][j-1]
13             else:
14                 dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1])
15     return dp[n][m]
```

□

(b)

Solution.

LONGEST BITONIC

```
1  function BITONIC(A):
2      n = length(A)
3      dp1 = array of size n+1
4      dp2 = array of size n+1
5      lis = array of size n
6      lds = array of size n
7      dp1[0] = −∞
8      for i from 1 to n:
9          dp1[i] = ∞
10     for i from 0 to n-1:
11         a = A[i]
12         find largest l such that dp1[l − 1] < a
13         lis[i] = l
14         dp1[l] = a
15     dp2[0] = −∞
16     for i from 1 to n:
17         dp[i] = ∞
18     for i in n-1 down to 0:
19         a = A[i]
20         find largest l such that dp1[l − 1] < a
21         lds[i] = l
22         dp2[l] = a
23     ans = 0
24     for i from 0 to n-1:
25         ans = max(ans, lis[i] + lds[i] - 1)
26     return ans
```

□

(c)

Solution.

```
isSingleton :: [a] -> Bool
isSingleton [_] = True
isSingleton _ = False

monotone :: [Int] -> (Int -> Int -> Bool) -> ([Int],[Int])
monotone [] _ = ([],[])
monotone [x] _ = ([x],[])
monotone (x:xs) f = let (ans, res) = go xs x in (x:ans, res) where
    go [] _ = ([],[])
    go (y1:ys) y0 = if f y0 y1 then (y1:a,b) else ([],y1:ys) where (a,b) = go ys y1

monotones :: [Int] -> [[Int]]
monotones [] = []
monotones [x] = [[x]]
```

```
monotones (x:y:ys) = go (x:y:ys) (x < y) where
go [] _ = []
go lis flag
    | flag  = let (a,b) = monotone lis (<) in a:go b (not flag)
    | otherwise = let (a,b) = monotone lis (>) in a : go b (not flag)

oscillatingLength :: [Int] -> Int
oscillatingLength = (+ 1).length.monotones
```

> **Remark**
>
> This code is taken from my TA notes for the Haskell course. Hence, better than pseudo-code, it is actually working Haskell code!

□

(d)

Solution.

LONGEST CONVEX SUBSEQUENCE

```
1  function switch((a,b),(c,d)):
2      if a < c :
3          return True
4      if a == c:
5          if b > d:
6              return True
7      return False
8  function LCONVEXS(A):
9      pairs = []
10     for i in 1 to n:
11         pairs.insert((A[i],i))
12     sortBy(pairs, switch)
13     perm = []
14     for (x,j) in pairs:
15         perm.insert(j)
16     dp = n × n array with everything initialized as 1
17     for i in 1 to n
18         L = n array of empty queues
19         R = n array of empty queues
20         for j in 1 to n:
21             if perm[j] < i:
22                 L[i].insert(Perm[j])
23             if perm[j] > i:
24                 R[i].insert(Perm[j])
25         maximum = 2
26         while R[i] ≠ empty:
```

Longest Convex Subsequence

```
            r = R[i].pop
            while L[i] ≠ empty and X[i] − X[L[i].peek] ≤ X[r] − X[i]:
                l = L[i].pop
                maximum = max(maximum, dp[l][i] + 1)
            dp[i][r] = maximum
    return max(dp)
```

> **Remark**
>
> This answer was in discussion with Sayeed and hence our answers are probably similar. The cause of our discussion was to get an $O(n^2)$ algorithm.

□

§2 Problem 2

Exercise

Let P be a set of n points evenly distributed on the unit circle, and let S be a set of m line segments with endpoints in P. The endpoints of the m segments are not necessarily distinct; n could be significantly smaller than $2m$.

(a) Describe an algorithm to find the size of the largest subset of segments in S such that every pair is disjoint. Two segments are disjoint if they do not intersect even at their endpoints.

(b) Describe an algorithm to find the size of the largest subset of segments in S such that every pair is interior-disjoint. Two segments are interior-disjoint if their intersection is either empty or an endpoint of both segments.

(c) Describe an algorithm to find the size of the largest subset of segments in S such that every pair intersects.

(d) Describe an algorithm to find the size of the largest subset of segments in S such that every pair crosses. Two segments cross if they intersect but not at their endpoints.

We will represent each chord as a pair (l, r) where $l < r$ as well as by a letter l. Furthermore, $m \leq \binom{n}{2} = O(n^2)$.

Note, two chords intersect internally if and only if $l_1 < l_2 < r_1 < r_2$ or $l_2 < l_1 < r_2 < r_1$ and on the circle if and only if $l_1 = l_2$ or $r_1 = r_2$ or $l_1 = r_2$ or $l_2 = r_1$.

(a) Let $\text{dp}[i][j]$ be the largest subset of segments between the i^{th} and j^{th} division where $i < j$. Consider the recursive relation: If $(i, j) \notin C$ where C is the set of chords

$$\text{dp}[i][j] = \max([\text{dp}[i][k] + \text{dp}[k+1][j] \mid k \in [i, j]])$$

and if $(i, j) \in C$ where C is set of chords

$$\text{dp}[i][j] = \max(\text{dp}[i+1][j-1] + 1, \max([\text{dp}[i][k] + \text{dp}[k+1][j] \mid k \in [i, j]]))$$

with base case $\text{dp}[i][i] = 0$. We can compute this by going in order of $k = j - i$.

Our answer is clearly $\text{dp}[1][n]$.

Computing this takes $n + n - 1 * 2 + n - 2 * 3 + ... + 1 * n = \sum_{i=0}^{n}(n-i)(i+1) = O(n^3) = O(mn)$

(b) We will take the previous algorithm and make some changes

Let $\text{dp}[i][j]$ be the largest subset of segments between the i^{th} and j^{th} division where $i < j$. Consider the recursive relation: If $(i, j) \notin C$ where C is the set of chords

$$\text{dp}[i][j] = \max([\text{dp}[i][k] + \text{dp}[k][j] \mid k \in [i, j]])$$

and if $(i, j) \in C$ where C is set of chords

$$\text{dp}[i][j] = \max(\text{dp}[i][j-1] + 1, \text{dp}[i+1][j] + 1, \max([\text{dp}[i][k] + \text{dp}[k][j] \mid k \in [i, j]]))$$

with base case $\text{dp}[i][i] = 0$. We can compute this by going in order of $k = j - i$.

Our answer is clearly $\text{dp}[1][n]$.

Computing this takes $O(n^3) = O(mn)$ time.

§3 Problem 3

Exercise

Consider the following process. At all times you have a single positive integer x, which is initially equal to 1. In each step, you can either increment x or double x. Your goal is to produce a target value n.

For example, you can produce the integer 10 in four steps as follows:

$$1 \xrightarrow{+1} 2 \xrightarrow{*2} 4 \xrightarrow{+1} 5 \xrightarrow{*2} 10$$

Obviously, you can produce any integer n using exactly $n-1$ increments, but for almost all values of n, this is horribly inefficient.

Describe and analyze an algorithm to compute the minimum number of steps required to produce any given integer n.

Solution.

Claim 3.1. The greedy strategy of dividing even target by 2 and decrementing odd target by 1 produces the optimal transformation (we stop at 1).

Proof. We will prove this via induction.

(B) For $n = 1, 2, 3$, the strategy clearly works.

(S) Let this strategy work for all $n < N$. We will show that the strategy will work for N.

FTSOC, let the strategy fail at N. If N is odd, then our first move is forced and beyond that our strategy works. If N is even, if we let $f(x)$ denote the number of moves to produce x, then this implies

$$f\left(\frac{N}{2}\right) > f(N-1) = 1 + f(N-2) = 2 + f\left(\frac{N-2}{2}\right)$$

But $f\left(\frac{N}{2}\right) \leq f\left(\frac{N-2}{2}\right) + 1$ as we can decrement on the first move. Hence, the above inequality can never be true. Thus, we have a contradiction. Thus, our strategy works for all N.

Thus, by induction, our strategy works for all numbers. □

```
--| Implementation of the above described algorithm
stepsToProduce :: Int -> Int
stepsToProduce 1 = 0
stepsToProduce 2 = 1
stepsToProduce n = if even n
  then stepsToProduce (n `div` 2)
  else stepsToProduce (n-1)
```

□

§4 Problem 4

Exercise

In each of the following cases, prove that the given set system (S, I) satisfies the axioms of a matroid. That is, show that I is nonempty, hereditary (closed under subsets), and satisfies the augmentation (exchange) property.

(a) Let T be an $m \times n$ matrix over some field (for example, the real numbers). Let S be the set of columns of T, and define

$$I = \{A \subseteq S \mid \text{the columns in } A \text{ are linearly independent}\}.$$

Show that (S, I) is a matroid.

(b) Suppose (S, I) is a matroid. Define

$$I' = \{A' \subseteq S \mid S \setminus A' \text{ contains some maximal } A \in I\}.$$

That is, the maximal independent sets of (S, I') are precisely the complements of the maximal independent sets of (S, I). Show that (S, I') is a matroid.

(c) Let S be a finite set, and let $S_1, S_2, ..., S_k$ be a partition of S into nonempty, disjoint subsets. Define

$$I = \{A \subseteq S \mid |A \cap S_i| \leq 1 \forall i = 1, 2, ..., k\}.$$

Show that (S, I) is a matroid. (That is, I consists of all subsets that contain at most one element from each S_i.)

(a)

Proof. The empty set is trivially linearly independent.

If for the elements of $\{t_1, t_2, ..., t_n\} = T \in I$, $\nexists a_1, a_2, ..., a_n \in \mathbb{F}; a_1t_1 + a_2t_2 + ...a_nt_n = 0$ then $\nexists a_1, a_2, ..., a_{n-1} \in \mathbb{F}$ s.t $a_1t_1 + a_2t_2 + ... + a_{n-1}t_{n-1} = 0$ as otherwise keeping $a_n = 0$ would invalidate the first side.

FTSOC let there be $\{t_1, t_2, ..., t_n\} = T \in I$ and $\{s_1, s_2, ..., s_m\} = S \in I$ that are independent and $n > m$ but $\forall x \in T, S \cup \{x\}$ is not linearly independent. This implies that $\text{span}(T) = \text{span}(S)$ which given the linear independence of T, S implies $n = m$. But that is a contradiction. Thus, the exchange property must be satisfied. □

(b)

Proof. $\emptyset \in I'$ as $S \setminus \emptyset = S$ has to have an maximal $A \in I$ as all $K \in I \Rightarrow K \subseteq S$.

If $A' \in I'$, then there is a maximal $A \in I$ such that $A \subseteq S \setminus A' \Rightarrow \forall B \subseteq A, A \subseteq S \setminus B$ as $\forall B \subseteq A, S \setminus A \subseteq S \setminus B$.

If $A', B' \in I'$, implies there are maximal $A, B \in I$ such that $A \subseteq S \setminus A'$ and $B \subseteq S \setminus B'$. As I is a matroid, $|A| = |B|$ as otherwise, we can use the 3rd axiom to get a larger set, contradicting the maximality.

Let $|A'| > |B'|$. FTSOC, $\forall a \in A' \setminus B', B' \cup \{a\} \notin I'$.

This implies $\forall a \in A' \setminus B', B \not\subseteq S \setminus (B' \cup \{a\}) \Rightarrow B \not\subseteq S \setminus B' \setminus \{a\} \Rightarrow a \in B$.

Notice, $\forall a \in A, A \setminus \{a\} \in I$ and $\exists b \in B, A \setminus \{a\} \cup \{b\} \in I$.

This implies, we can exchange out $B \setminus \{a\} \cup \{a'\}$ where $a' \in A \setminus B \setminus \{a'\}$ and that will be maximal. Note, $A \setminus B \setminus \{a'\}$ can't be empty as otherwise we have a contradiction. Thus, we are done.

□

(c)

Proof. The empty set trivially satisfies the property and hence $\emptyset \in I$.

If $\forall i \in [k], |A \cap S_i| \leq 1 \Rightarrow \forall A' \subseteq A, \forall i \in [k], |A' \cap S_i| \leq |A \cap S_i| \leq 1$. Thus, $A \in I \Rightarrow \forall A' \subseteq A, A' \in I$.

Let $A = \bigcup_{i \in [k]} A \cap S_i \in I$ and $B = \bigcup_{i \in [k]} B \cap S_i \in I$ and $|A| > |B|$.

This implies, there is some $i \in [k]$ such that $|A \cap S_i| = 1$ but $|B \cap S_i| = 0$ as otherwise $|A| = |B|$. Thus, $B \cup (A \cap S_i) \in I$.

Thus, all three axioms are satisfied and we have a matroid. □

§5 Problem 5

Exercise

Let $G = (V, E)$ be a weighted, directed graph with a nonnegative weight function $w : E \to \{0, 1, ..., W\}$ for some nonnegative integer W.

Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(W|V| + |E|)$ time.

Modified Dijkstra's

```
1  function DIJKSTRA(s,G):
2      d = array of size |V| initialized at ∞
3      d[s] = 0
4      u = array of size |V| initialized at FALSE
5      p = array of size |V|
6      w = array of queues, of size W + 1
7      w[0].push(s)
8      pos = 0
9      num = 1
10     while num > 0:
11         while w[pos mod(W + 1)] is empty:
12             pos = pos + 1
13         cur = w[pos mod(W + 1)].pop
14         num = num - 1
15     if w[cur]:
16         continue
17     u[cur] = True
18     for t in cur.neighbors:
19         l = edge weight between cur and t
20         if d[t] > d[cur] + l:
21             p[t] = cur
22             d[t] = d[cur] + l
23             w[d[t] mod(W + 1)].push(t)
24             num = num + 1
25     return d
```

We use the idea that we can't really push vertexes who are more than W distance away from current to the queues. Hence, instead of keeping $W * V$ queues, we just remember the least distance we had and then update the $W + 1$ queues accordingly.

We take $O(W)$ time to search for the closest vertex. We make this search $O(|V|)$ times. Finally, the update step is done $O(|E|)$ times. Thus, the complexity is $O(W|V| + |E|)$ as required.

§6 Problem 6

Exercise

Prove that a binary tree that is not full cannot correspond to an optimal prefix code.

Proof. FTSOC, let an optimal prefix code have a non-full binary tree.

This implies there is a node with only one child subtree. We can move the subtree upwards, eliminating the node. This prefix tree has lower cost as the characters encoded by the subtree is now shorter by 1. This contradicts the optimality of the tree.

Thus, we have a contradiction. Thus, our initial assumption must be false. Thus, no optimal prefix code can correspond to an not full binary tree. □