Advanced Algorithms Analysis and Design

By

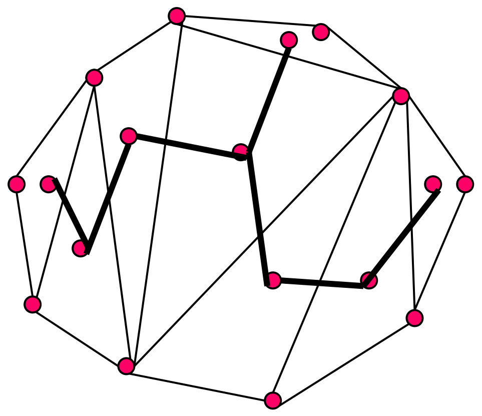
Nazir Ahmad Zafar

Lecture No 21

Optimal Weight Triangulation

Observations in Dual Graph

- Each internal node corresponds to one triangle
- Each edge between internal nodes corresponds to one chord of triangulation.
- Now for given n-vertex polygon
  - n-2 internal nodes which are in fact triangles and
  - n-3 edges which are chords



Proof of Lemmas

Lemma 1

- A triangulation of a simple polygon, with n vertices, has n-2 number of triangles.

Proof

Proof is done using mathematical induction

Basis Step

- Suppose that there three vertices, polygon will be a triangle, i.e. there are 3 - 2 = 1 number of triangles
- Hence statement is true for n = 3

Contd..

- If there are 4 vertices, polygon will be in fact a rectangle, divide it into two triangles. The result is true. Hence statement is true for n = 4

Inductive Hypothesis

- Let us suppose that statement is true for n = k, i.e., if there are k vertices then there are k-2 number of triangles

Claim

- Now we have to prove that if there are k+1 vertices there must be k+1-2 = k-1, number of triangles.

Contd..

- Since for k vertices there are k-2 triangles. Insert one more point at boundary of polygon
- In fact point will be inserted at boundary of one of the triangles. So the triangle will become rectangle. Divide it into two triangles. It will increase one more triangle in the division. Hence it becomes;
  $k – 2 + 1 = k - 1$, number of triangles.
- It proves the claim. Hence by mathematical induction it proves that for n number of vertices there are n - 2 number of triangles.

Another Lemma

Lemma 2

- A triangulation of a simple polygon, with n vertices, has n-3 chords.

Proof

- Proof not difficult, and it can be proved following the steps of proof in lemma 1.
- If there are three points, it will be triangle. To make a chord in a polygon, it requires at least four points.
- So you have to give proof for $n \geq 4$.

Contd..

Basis Step

- Suppose that there are four number of vertices, in this case polygon will be a rectangle, there must be 4 - 3 = 1 number of chords
- Hence statement is true for n = 4

Inductive Hypothesis

- Let us suppose that the statement is true for n = k, i.e., if there are k vertices then there are k - 3 number of chords of the polygon

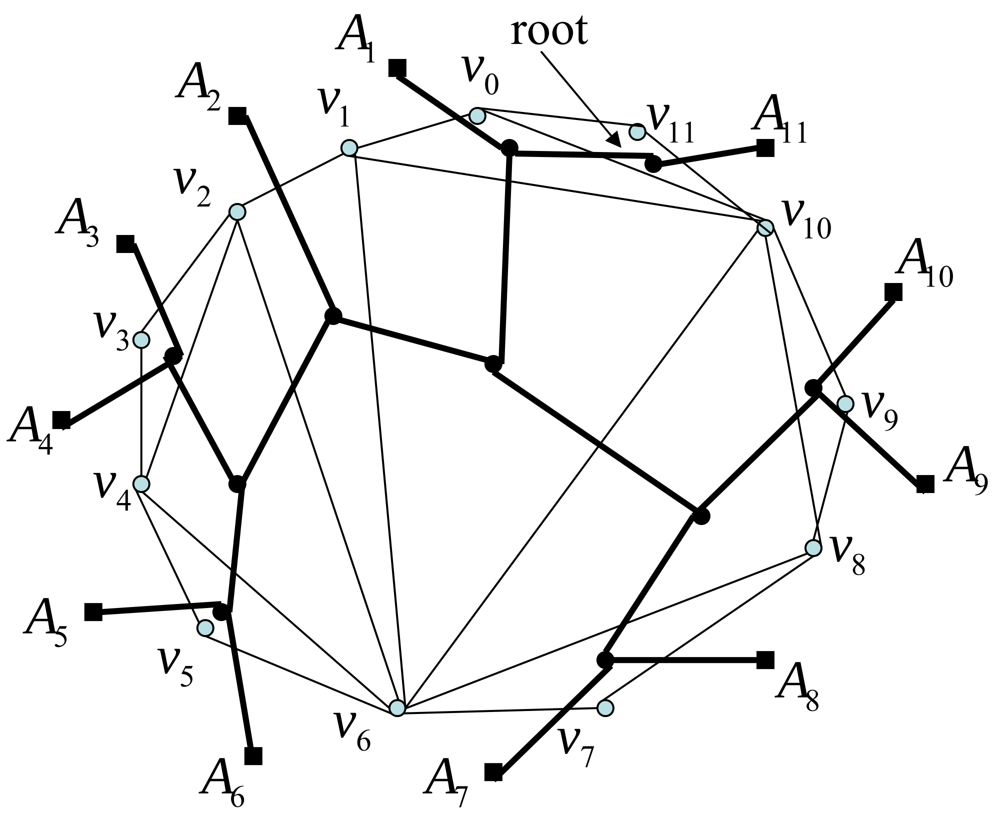
Contd..

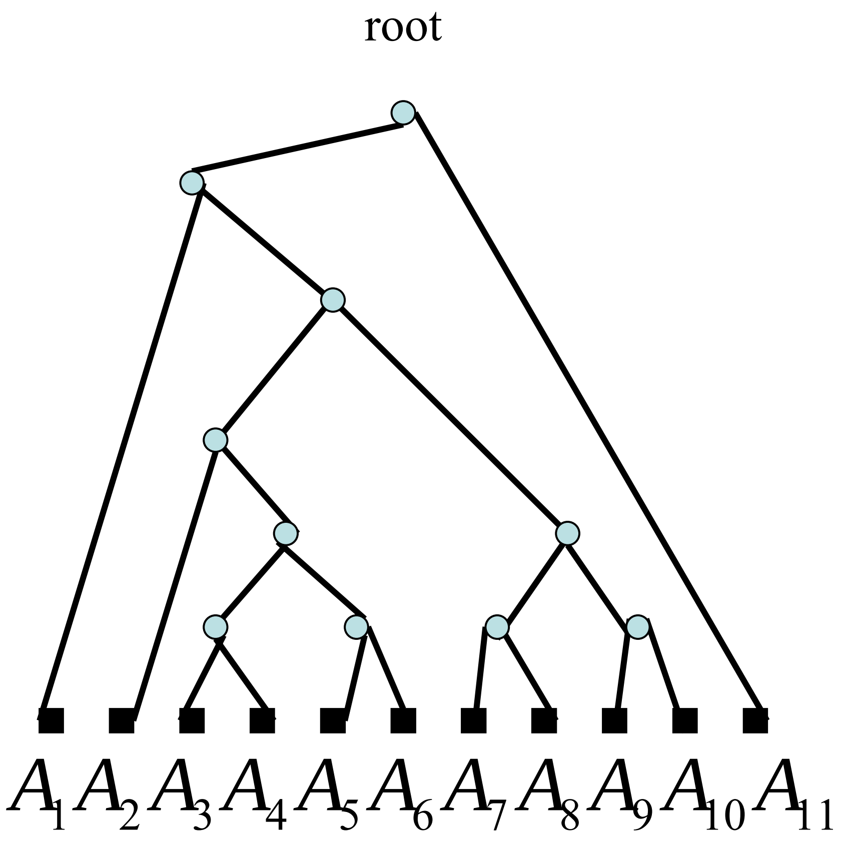
Claim

- Now we have to prove that if there are k+1 vertices there must be k+1-3 = k-2, number of chords.
- Since for k vertices there are k-3 chords. Insert one more point at boundary of polygon
- In fact point will be inserted at boundary of one of the triangles. So the triangle will become rectangle. Divide it into two triangles. It will increase one more chord in the division. Hence it becomes;
  k – 3 + 1 = k - 2, number of chords. Proved.

Correspondence to Binary Trees

- Relationship between optimal triangulation and chain matrix multiplication problem





root
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11}

Correspondence to Binary Trees

- In chain matrix multiplication, associated binary tree is the evaluation tree for the multiplication, where the leaves of the tree correspond to the matrices, and each node of the tree is associated with a product of a sequence of two or more matrices.
- Now let us consider an (n+1) sided convex polygon, $P = <v_0, v_1, \ldots, v_n>$ and fix one side of it as (v_0, v_n)
- Consider a rooted binary tree whose:

  root node = is the triangle containing side (v_0, v_n),

  internal nodes = are nodes of the dual tree, and

  leaves = are remaining sides of the tree.
- This partitioning of polygon is equivalent to a binary tree with n-1 leaves, and vice versa.

Dynamic Programming Solution

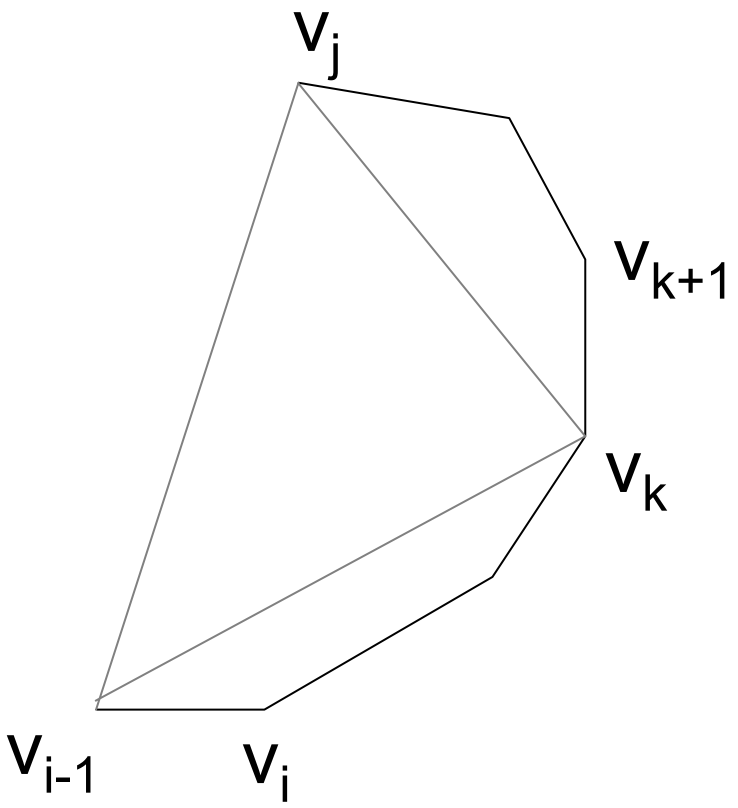
- Let t[i, j] = minimum weight triangulation for the sub-polygon $<v_{i-1}, v_i ,\ldots, v_j>$, for 1 ≤ i ≤ j ≤ n
- We have start with v_{i-1} rather than v_i, to keep the structure as similar as matrix chain multiplication
- It is to be noted that if we can compute $t[i, j]$ for all i and j $(1 \le i \le j \le n)$, then the weight of minimum weight triangulation of the entire polygon will be $t[1, n]$. Hence it is our objective function.
- For the base case

  $t[i, i]$ = 0, for line (v_{i-1}, v_i).

Optimal Substructure

- $t[i, j]$ = weight of an optimal triangulation of polygon $<v_{i-1}, v_i, \ldots, v_j>$.
- $t[i, j] = \min_k \{ t[i, k] + t[k+1, j] + w(\Delta v_{i-1} v_k v_j) \}, i < j$
- $t[i, i] = 0$

$i \leq k \leq j\text{-}1$



Optimal Substructure

- In general, to compute $t[i, j]$, consider the sub-polygon $<v_{i-1}, v_i ,\dots, v_j>$, where $i \leq j$.
- One of the chords of this polygon is the side (v_{i-1}, v_j).
- We may split this sub-polygon by introducing a triangle whose base is this chord, and whose third vertex is any vertex v_k, where $i \leq k \leq j\text{-}1$.
- This subdivides the polygon into 2 sub-polygons $<v_{i-1},\dots v_k>$ and $<v_{k+1},\dots v_j>$, whose minimum weights are $t[i, k]$ and $t[k+1, j]$.
- It leads to following recursive rule computing $t[i, j]$

$$t[i, i] = 0$$

$$t[i, j] = \min_{i \leq k \leq j-1} (t[i, k] + t[k+1, j] + w(v_{i-1}v_kv_j)) \text{ for } i < j$$

Algorithm

$t[i, j] = \min_{i < k < j} (t[i, k] + t[k, j] + w(v_i\ v_j\ v_k))$ if $i < j$;

$t[i, j] = 0$ if $i = j$.

```
function min_weight_tri(p[ ], n)
1.  for $i \leftarrow 1$ to n do
2.          $t[i, i] \leftarrow 0$;
3.  for $l \leftarrow 2$ to n do
4.          for $i \leftarrow 1$ to $n - l+1$ do
5.                  $j \leftarrow i + l-1$;
6.                  $t[i, j] \leftarrow \infty$;
7.                  for $k \leftarrow i$ to $j - 1$ do
8.                          $q \leftarrow t[i, k] + t[k+1, j] + w(v_i\ v_j\ v_k)$;
9.                          if $(q < t[i, j])$ then
10.                                 $t[i, j] \leftarrow \min(t[i, j], q)$;
11.                                 $v[i, j] \leftarrow k$;
12. return( t(1, n) );
```

t[1,1]	t[1,2]	. . .	t[1,n]
	t[2,2]	. . .	t[2,n]
		. . .	. . .
			t[n, n]

Computational Cost

$$T(n) = n + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (j-i) = \sum_{i=1}^{n} \sum_{k=1}^{n-i} k$$

$$T(n) = n + \sum_{i=1}^{n} \frac{(n-i)(n-i+1)}{2}$$

$$T(n) = n + \frac{1}{2} \sum_{i=1}^{n} (n^2 + n + 1) - \frac{1}{2} \sum_{i=1}^{n} ((1+2n)i - i^2)$$

$$= n + \frac{1}{2}(n^2 + n + 1) \sum_{i=1}^{n} 1 - \frac{1}{2}(1+2n) \sum_{i=1}^{n} i + \frac{1}{2} \sum_{i=1}^{n} i^2$$

$$= \Theta(n^3)$$

Longest Common Subsequence Problem

An Introduction

- In biological applications, we often want to compare the DNA of two (or more) different organisms.
- A part of DNA consists of a string of molecules called bases, where the possible bases are
  - adenine,
  - guanine,
  - cytosine, and
  - thymine.
- Represent each of the bases by their initial letters
- A part of DNA can be expressed as a string over the finite set {A, C, G, T}.

An Introduction

- For example, the DNA of one organism may be S1= CCGGTCGAGTGCGCGGAAGCCGGCCGAA,
- While the DNA of another organism may be S2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA.
- One goal of comparing two parts of DNA is to determine how "similar" two parts are, OR
- Measure of how closely related two organisms are.
- As we know that similarity is an ambiguous term and can be defined in many different ways.
- Here we give some ways of defining it with reference to this problem.

An Introduction: Similarity

- For example, we can say that two DNA parts are similar if one is a substring of the other. In our case, neither *S*1 nor *S*2 is a substring of the other. This will be discussed in string matching.
- Alternatively, we could say two parts are similar if changes needed to turn one to other is small.
- Another way to measure similarity is by finding third part *S*3 in which bases in *S*3 appear in both *S*1, *S*2
- Bases must preserve order, may not consecutively.
- Longer *S*3 we can find, more similar *S*1 and *S*2 are.
- In above, *S*3 is GTCGTCGGAAGCCGGCCGAA

What is a Subsequence?

- In mathematics, a subsequence of some sequence is a new sequence which is formed from original one by deleting some elements without disturbing the relative positions of the remaining elements.

Examples:

- < B,C,D,B > is a subsequence of

  < A,C,B,D,E,G,C,E,D,B,G > ,

  with corresponding index sequence <3,7,9,10>.
- < D, E, E, B > is also a subsequence of the same

  < A,C,B,D,E,G,C,E,D,B,G > ,

  with corresponding index sequence <4,5,8,10>.

Longest Common Subsequence

- The sequence Z = (B, C, A) is a subsequence of
  X = (A, B, C, B, D, A, B).
- The sequence Z = (B, C, A) is also a subsequence of
  Y = (B, D, C, A, B, A).
- Of course, it is a common subsequence of X and Y.
- But the above sequence is not a longest common subsequence
- This is because the sequence Z' = (B, D, A, B)
  is a longer subsequence of

  X = (A, B, C, B, D, A, B) and

  Y = (B, D, C, A, B, A)

Problem

Statement:

- In the longest-common-subsequence (LCS) problem, we are given two sequences

  $X = <x_1, x_2, . . . , x_m>$ and

  $Y = <y_1, y_2, . . . , y_n>$
- And our objective is to find a maximum-length common subsequence of X and Y.

Note:

- This LCS problem can be solved using brute force approach as well but using dynamic programming it will be solved more efficiently.

Brute Force Approach

- First we enumerate all the subsequences of
  $X = \langle x_1, x_2, \ldots, x_m \rangle$.
- There will be 2^m such subsequences.
- Then we check if a subsequence of X is also a subsequence of Y.
- In this way, we can compute all the common subsequences of X and Y.
- Certainly, this approach requires exponential time, making it impractical for long sequences.

Note:

- Because this problem has an optimal sub-structure property, and hence can be solved using approach of dynamic programming

Dynamic Programming Solution

Towards Optimal Substructure of LCS: Prefixes

- As we shall see, the natural classes of sub-problems correspond to pairs of “prefixes” of the two input sequences.
- To be precise, given a sequence $X = <x_1, x_2, ..., x_m>$, we define the ith prefix of X, for $i = 0, 1, ..., m$, as $X_i = <x_1, x_2, ..., x_i>$.

Examples,

If X = <A, B, C, B, D, A, B> then

- X_4 = <A, B, C, B> and
- X_0 is the empty sequence = < >

Theorem: Optimal Substructure of an LCS

- If $X = (x_1, x_2, \ldots, x_m)$, and $Y = (y_1, y_2, \ldots, y_n)$ be sequences and let us suppose that $Z = (z_1, z_2, \ldots, z_k)$ be a longest common sub-sequence of X and Y

- *Let,* $X_i = (x_1, x_2, \ldots, x_i)$, $Y_j = (y_1, y_2, \ldots, y_j)$ and
  $Z_l = (z_1, z_2, \ldots, z_l)$ are prefixes of X, Y and Z res.

1. if $x_m = y_n$, then $z_k = x_m$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

Proof of Theorem

Case 1

- On contrary suppose that $x_m = y_n$ but $z_k \neq x_m$,
- Then we could append $x_m = y_n$ to Z to obtain a common subsequence of X and Y of length k + 1, contradicting the supposition that Z is a LCS of X and Y.
- Thus, we must have $z_k = x_m = y_n$.
- Now, the prefix Z_{k-1} is a length-(k - 1) common subsequence of X_{m-1} and Y_{n-1}.

Now we wish to show that it is an LCS.

- Suppose, there is a common subsequence W of X_{m-1} and Y_{n-1} with length greater than k - 1.
- Appending $x_m = y_n$ to W gives common subsequence of X and Y whose length is greater than k, a contradiction.

Theorem: Optimal Substructure of an LCS

Case 2

- If $z_k \neq x_m$, then Z is a common subsequence of X_{m-1} and Y.
- If there were a common subsequence W of X_{m-1} and Y with length greater than k, then W would also be a common subsequence of X_m and Y, contradicting the assumption that Z is an LCS of X and Y.

Case 3

- The proof is symmetric to (2).