Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 20

0-1 Knapsack Problem's Algorithm (using Dynamic Programming)

and

Optimal Weight Triangulation

Today Covered

- 0-1 knapsack problem
  - Algorithm
  - Generalizations and variations of the Problem
- Optimal Weight Triangulation
  - Definitions
  - Problem Analysis
  - Dynamic Solution
  - Algorithm using Dynamic Programming
  - Time Complexity
- Conclusion

Optimal Value: Entire Solution

Let W = 10

Final Solution: V[4, 10] = 90

Items selected = {2, 4}

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

V[i, w]	W = 0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0	0	0	0	40	40	40	40	40	50	50
i = 3	0	0	0	0	40	40	40	40	40	50	70
i = 4	0	0	0	50	50	50	50	90	90	90	90

Constructing Optimal Solution

- $V[i, j] = \max(V[i-1, j], v_i + V[i-1, j - w_i])$;
- $i = 4$

  $V[4, 10] = \max(70, 50 + 40) = 90$; Keep(4, 10) = 1
- $i = 3$

  $V[3, 10 - 3] = V[3, 7] = \max(40, 30) = 40$ Keep(3, 7) = 0
- $i = 2$

  $V[2, 7] = \max(10, 40) = 40$ Keep(2, 7) = 1
- $i = 1$

  $V[1, 7-4] = V[1, 3] = 0$ Keep(1, 3) = 0

Algorithm : Dynamic Programming

KnapSack (v, w, n, W)

for (i = 1 to n), V[i, 0] = 0;

for (j = 0 to W), V[0, j] = 0;

```
 for (i = 1 to n)
      for (j = 1 to W)
         if (w(i) ⩽ j)
                  V[i, j] = max(V[i-1, j], v_i + V[i-1, j – w_i]);
         else
                  V[i, j] = V[i-1, j];
```

Return V[n, W]

Time Complexity O(n.W)

Output Elements: Knapsack Algorithm

How do we use all values *keep[i, w],* to determine a subset S of items having the maximum value?

- If *keep[n, w]* is 1, then n $\in$ S, and we can repeat for *keep[n-1, W - w_n]*
- If *keep[n, w]* is 0, then n $\notin$ S and we can repeat for *keep[n-1, W]*
- Following is a partial program for this output elements

```
K = W;
for (i = n down to 1)
        if keep[i, K] = = 1
                output i
                K = K – wi
```

Complete: Dynamic Programming Algorithm

```
KnapSack(v, w, n, W)
for (w = 0 to W), V[0, w] = 0;  for (i = 1 to n), V[i, 0] = 0;
for (i = 1 to n)
        for (w = 1 to W)
            if ((w(i) ⩽ w) and (v_i + V[i-1,w – w_i] > V[i-1,w]))
                        V[i, w] = (v_i + V[i-1,w – w_i];
                        keep[i, w] = 1;
            else
                        V[i, w] = V[i-1,w];
                        keep[i, w] = 0;

K = W;
            for (i = n down to 1)
                        if keep[i, K] = = 1
                                    output i
                                    K = K – w_i
Return V[n, W]
```

1. Generalizations ($x_i \in \{0, 1\}$)

- Common to all versions are a set of n items, with each item $1 \leq j \leq n$ having an associated profit p_j and weight w_j.
- The objective is to pick some of the items, with maximal total profit, obeying that maximum total weight limit W.
- Generally, coefficients are scaled to become integers, and they are almost always assumed to be positive.
- The knapsack problem in its most basic form:

$$\text{maximize} \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0, 1\}, \quad \forall\ 1 \leq i \leq n$$

2. Specialization (weight = profit)

- If for each item the profit and weight are identical, we get the subset sum problem
- Often called the decision problem

$$\text{maximize} \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} p_i x_i \leq W$$

$$x_i \in \{0, 1\}, \quad \forall\ 1 \leq i \leq n$$

3. Generalizations (more than one objects)

- If each item can be chosen multiple times, we get the bounded knapsack problem.
- Suppose, weight of each item is at least 1 unit, then we can never choose an item more than W times.
- This is another variation in the basic form
- Now the problem will become

$$\text{maximize} \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0, 1, \ldots, W\}, \quad \forall\ 1 \leq i \leq n$$

4. Generalizations (K Classes)

- If the items are subdivided into k classes denoted N_i
- And exactly one item must be taken from each class
- We get the multiple choice knapsack problem
- In this case our optimized mathematical model is

$$\text{maximize} \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij}$$

$$\text{subject to} \sum_{i=1}^{n} \sum_{j \in N_i} w_{ij} x_{ij} \leq W \qquad \text{where} \sum_{j \in N_i} x_{ij} = 1$$

$$\forall\ 1 \leq i \leq k$$

$$x_{ij} \in \{0, 1\}, \quad \forall\ 1 \leq i \leq k, j \in N_i$$

5. Generalizations (more than one knapsacks)

- If there are n items, m knapsacks with capacities W_i
- We get the multiple knapsack problem

maximize $$\sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

subject to $$\sum_{j=1}^{n} w_j x_{ij} \le W_i \qquad 1 \le i \le m$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad 1 \le i \le n \qquad 1 \le j \le n$$

$x_{ij} \in \{0, 1\}$, $1 \le i \le m$ and $1 \le j \le n$

6. Item's Different Weight in Different Knapsack)

- If in the multiple knapsack problem, the weights are not the same in every container
- We are allowed to choose each item multiple times, we get multiple constrained knapsack problem

maximize $\sum_{j=1}^{n} p_j x_j$

subject to $\sum_{j=1}^{n} w_{ij} x_j \le W_i \qquad 1 \le i \le m$

$x_j \ge 0, \quad x_j \in Z$

Optimal Weight Triangulation

Why Polygon Triangulation?

- Finite element method is a technique for solving numerical problems e.g. stress or heat flow simulations of any kind of systems
- It involves dividing a shape into simple elements for example triangles
- Then formulating a set of linear equations describing relations between simulated quantities in each element, and solving these equations.
- The time and accuracy both, in solution, depend on the quality of dividing into triangles
- Generally, it is desired that triangles must be as close to equilateral as possible

Similarity: Optimal Polygon Triangulation, other Problem

- Optimal triangulation problem is very similar to matrix chain multiplication
- It is an excellent approach to make one to one corresponding between two problems and
- Then solving one problem based on the approach already used in the solution of the other problem
- This is what we are going to do in solving an optimal solution of the triangulation problem which is very popular in computational geometry
- Applications of this problem can be observed in many other areas where division of structures is required before performing computation over it.

Basic Concepts

Polygon

A set of finite piecewise-linear, closed curve in a plane is called a polygon

Sides

The pieces of the polygon are called its sides

Vertex

A point joining two consecutive sides is called a vertex

Interior

The set of points in the plane enclosed by a simple polygon forms interior of the polygon

Boundary

The set of point on the polygon forms its boundary

Exterior

The set of points surrounding the polygon form its exterior

Basic Concepts

Simple Polygon

A polygon is simple if it does not cross itself, i.e., if its sides do not intersect one another except for two consecutive sides sharing a common vertex.

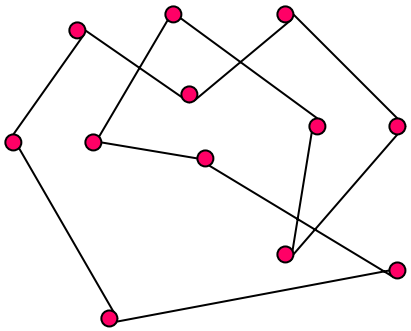
Subdivision of Polygon

A simple polygon subdivides the plane into its interior, its boundary and its exterior.

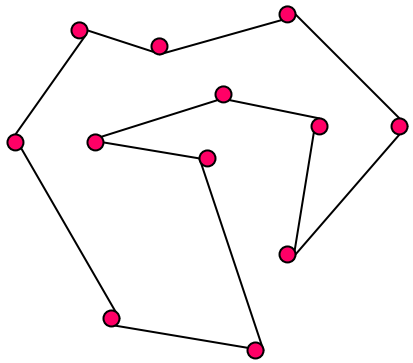
Convex Polygon

A simple polygon is convex if given any two points on its boundary or in its interior, all points on the line segment drawn between them are contained in the polygon's boundary or interior.

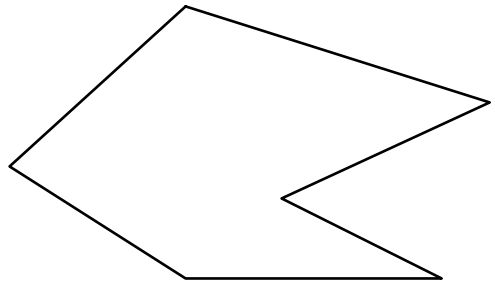
Polygons



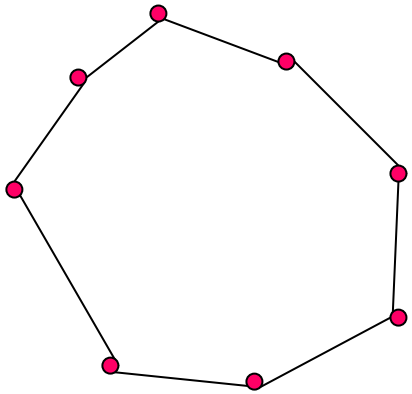
Polygon

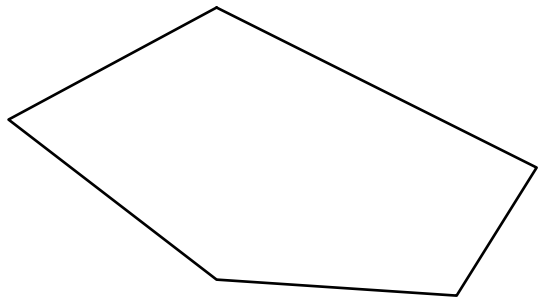


Simple Polygon



Simple Polygon

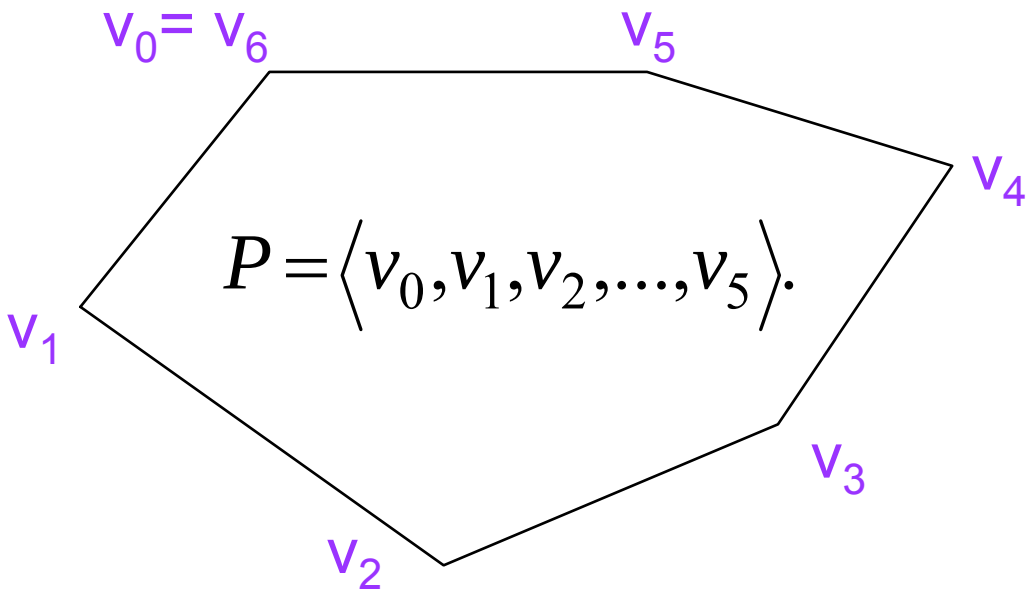




Convex Polygons

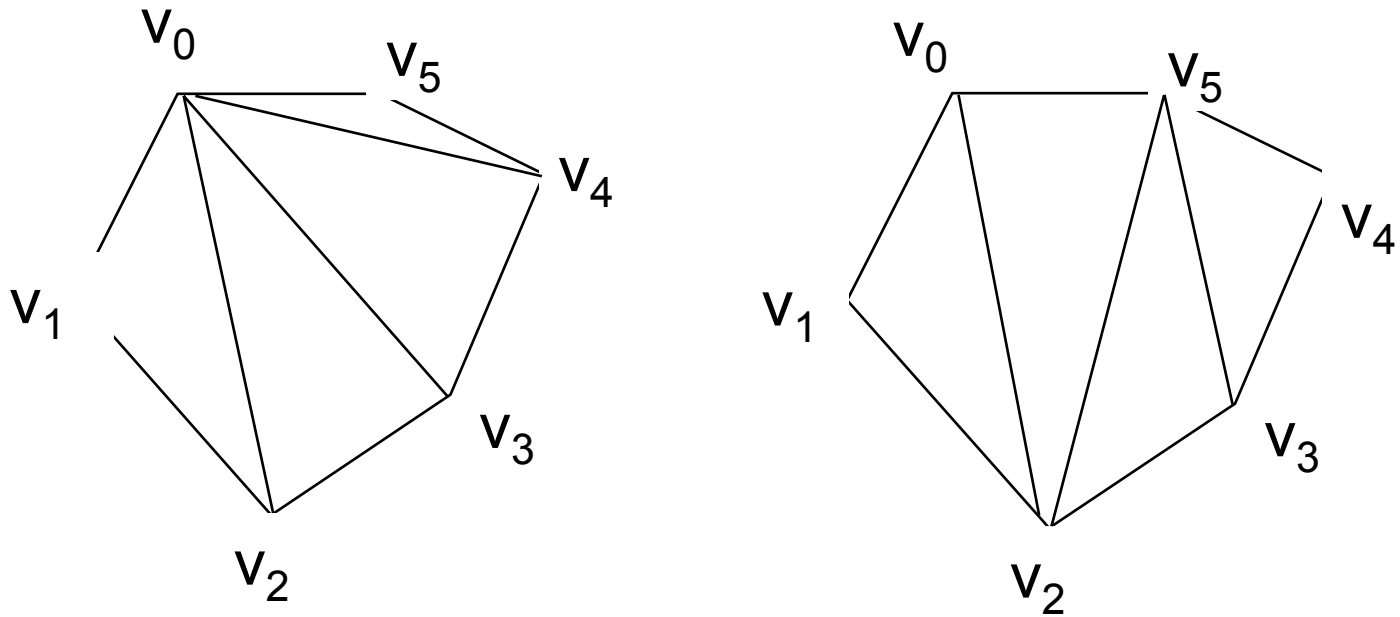
Labeling Convex Polygons

- For a convex polygon, it is assumed that its vertices are labeled in counterclockwise order $P = \langle v_0, v_1, v_2, \ldots, v_{n-1} \rangle$.
- We assume that indexing is done modulo *n*, so $v_0 = v_n$. and the above polygon P has *n* number of vertices



Chords in Polygons

- Given two non-adjacent vertices v_i, v_j of a convex polygon ($i < j$), the line segment $v_i v_j$ is called a chord
- For two non-adjacent vertices v_i and v_j of a simple polygon ($i < j$), line segment $v_i v_j$ is a chord if interior of the segment lies entirely in the interior of polygon
- Any chord subdivides a polygon into two polygons



Optimal Weight Triangulation Problem

- A triangulation of a convex polygon is a maximal set T of pair-wise non-crossing chords, i.e., every chord not in T intersects the interior of some chord in T
- It is easy to see that such a set subdivides interior of polygon into a collection of triangles, pair-wise disjoint

Problem Statement

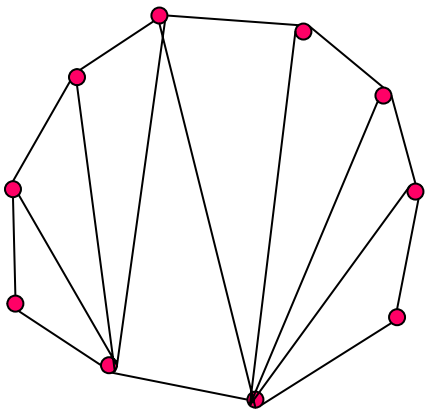
Given a convex polygon, determine a triangulation that minimizes sum of the perimeters of its triangles

Analysis

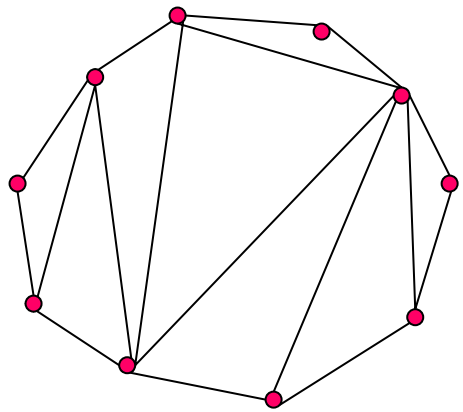
- Given three distinct vertices, v_i, v_j and v_k.
- Define a weight of associated triangle by a function $w(v_i, v_j, v_k) = |v_i\ v_j| + |v_j\ v_k| + |v_k\ v_i|$, where $|v_i\ v_j|$ denotes length of line segment (v_i, v_j).

Brute Force Triangulation: Triangulation

- In general, given a convex polygon, there are exponential number of possible triangulations.
- There are many criteria that are used depending on application. For example, you have to minimize the value of cable in designing such triangulation
- This suggests the optimal solution to this problem

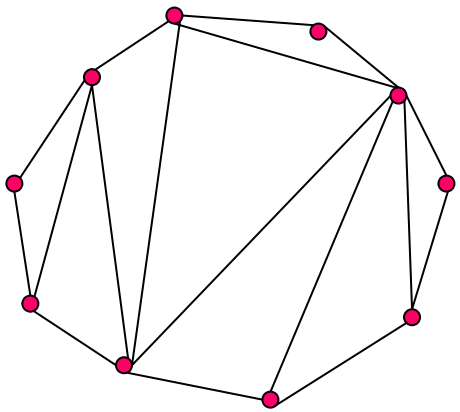


Triangulation

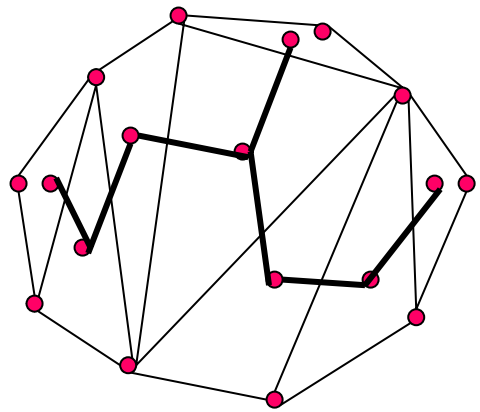


Dual Graphs

- Dual graph of a triangulation is a graph whose vertices are the triangles, and in which two vertices are adjacent if the corresponding both triangles share a common chord
- It is to be noted that dual graph is a tree. And hence algorithms for traversing trees can be used for traversing the triangles of that triangulation

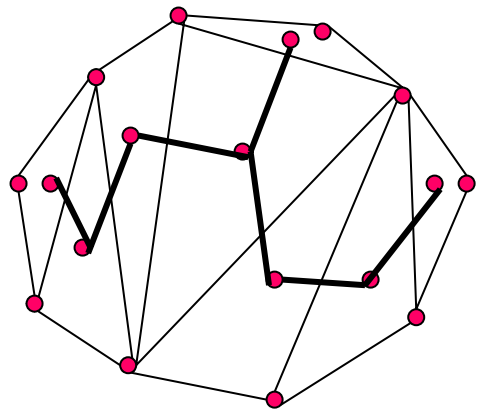


Dual Graph



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

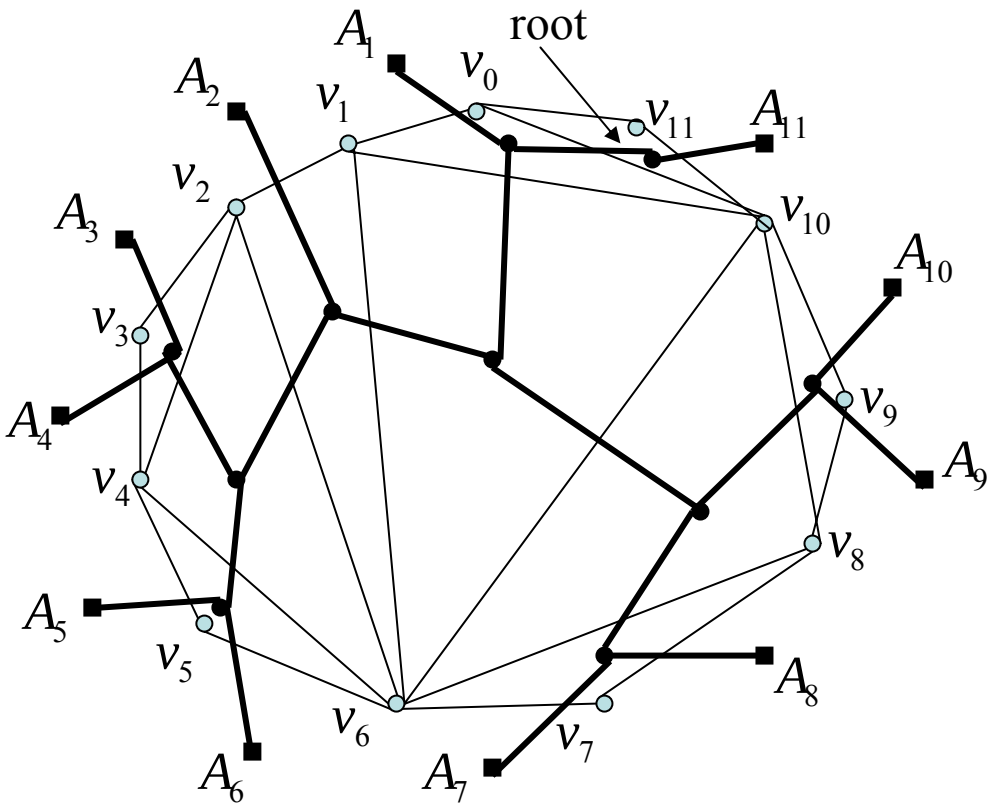
Contd..

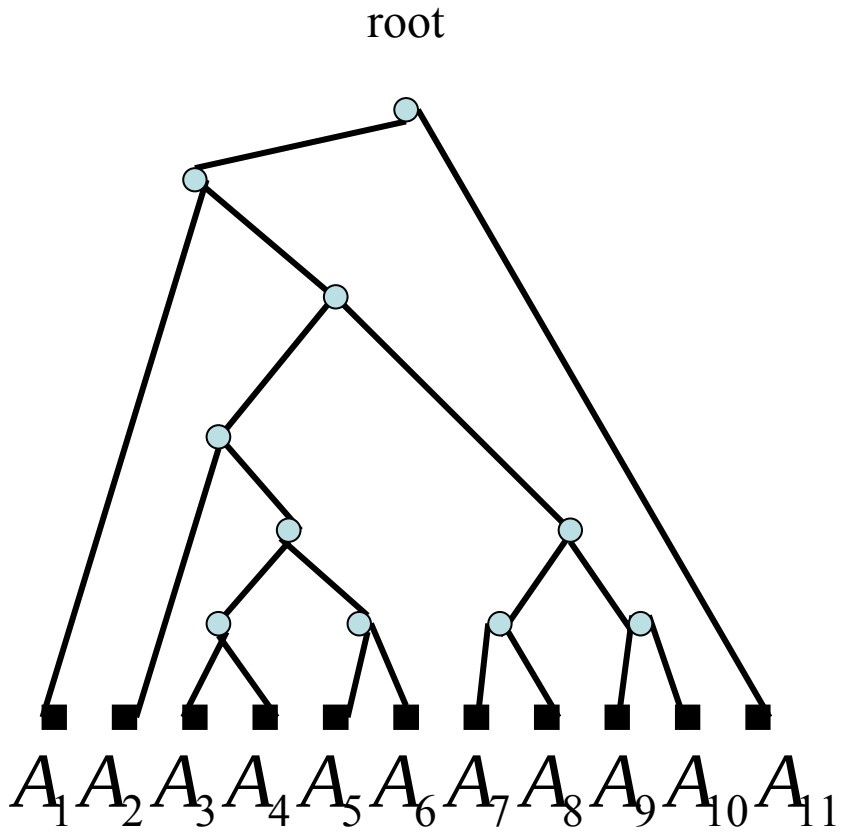
Claim

- 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





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, $P = \langle v_0, v_1, \ldots, v_n \rangle$ 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

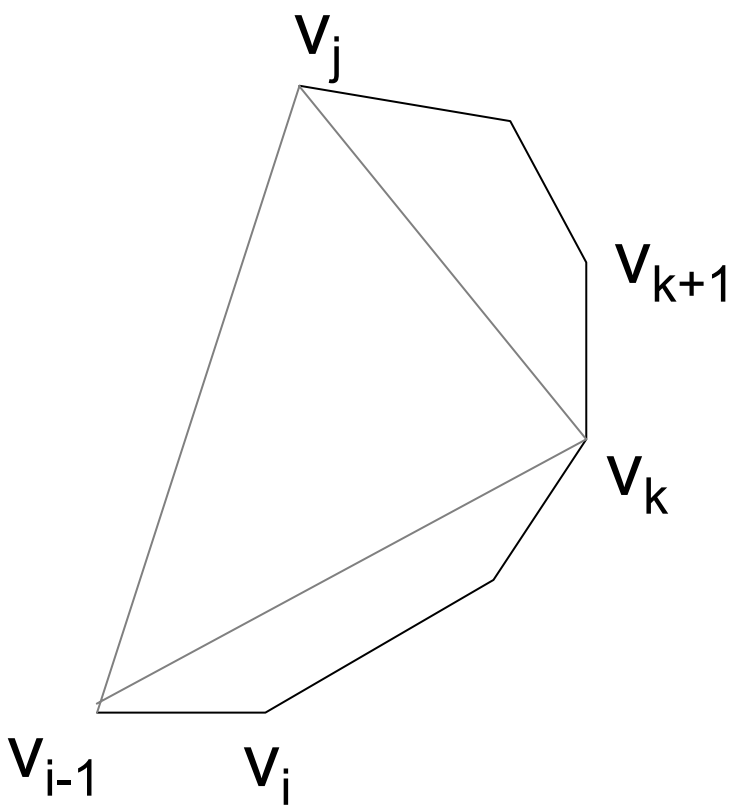
- Let t[i, j] = minimum weight triangulation for the sub-polygon $<v_{i-1}, v_i, \ldots, v_j>$, for $1 \leq i \leq j \leq n$
- We have start with v_{i-1} rather than v_i, to keep the structure as similar as matrix chain multiplication
- It is to be noted that if we can compute $t[i, j]$ for all i and j $(1 \leq i \leq j \leq n)$, 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 $\langle v_{i-1}, v_i, \ldots, v_j \rangle$.
- $t[i, j] = \min_k \{ t[i, k] + t[k+1, j] + w(\Delta v_{i-1} v_k v_j) \}$, $i < j$
- $t[i, i] = 0$

$i \leq k \leq j-1$



Optimal Substructure

- In general, to compute $t[i, j]$, consider the sub-polygon $<v_{i-1}, v_i, \ldots, v_j>$, 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 $i \leq k \leq j-1$.
- This subdivides the polygon into 2 sub-polygons $<v_{i-1},\ldots v_k>$ and $<v_{k+1},\ldots v_j>$, whose minimum weights are $t[i, k]$ and $t[k+1, j]$.
- It leads to following recursive rule computing $t[i, j]$

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

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

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 $i = j$.

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

```
function min_weight_tri(p[ ], n)
1.  for i ← 1 to n do
2.          t[i, i] ← 0;
3.  for l ← 2 to n do
4.          for i ← 1 to n – l+1 do
5.                  j ← i + l-1;
6.                  t[i, j] ← ∞;
7.                  for k ← i  to j - 1 do
8.                          q ← 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] ← min(t[i, j], q);
11.                                 v[i, j] ← k;
12.  return( t(1, n) );
```

Computational Cost

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

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

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

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

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