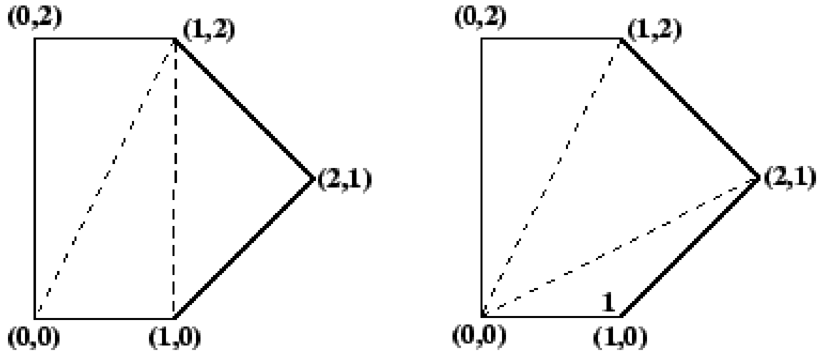
Minimum Cost Polygon Triangulation

A triangulation of a convex polygon is formed by drawing diagonals between non-adjacent vertices (corners) such that the diagonals never intersect. The problem is to find the cost of triangulation with the minimum cost. The cost of a triangulation is sum of the weights of its component triangles. Weight of each triangle is its perimeter (sum of lengths of all sides)

See following example taken from this source.



Two triangulations of the same convex pentagon. The triangulation on the left has a cost of 8 + 2√2 + 2√5 (approximately 15.30), the one on the right has a cost of 4 + 2√2 + 4√5 (approximately 15.77).

This problem has recursive substructure. The idea is to divide the polygon into three parts: a single triangle, the sub-polygon to the left, and the sub-polygon to the right. We try all possible divisions like this and find the one that minimizes the cost of the triangle plus the cost of the triangulation of the two sub-polygons.

```
Let Minimum Cost of triangulation of vertices from i to j be minCost(i, j)
If j <= i + 2 Then
  minCost(i, j) = 0
Else
  minCost(i, j) = Min { minCost(i, k) + minCost(k, j) + cost(i, k, j) }
                  Here k varies from 'i+1' to 'j-1'

Cost of a triangle formed by edges (i, j), (j, k) and (k, j) is
  cost(i, j, k)  = dist(i, j) + dist(j, k) + dist(k, j)
```

Following is C++ implementation of above naive recursive formula.

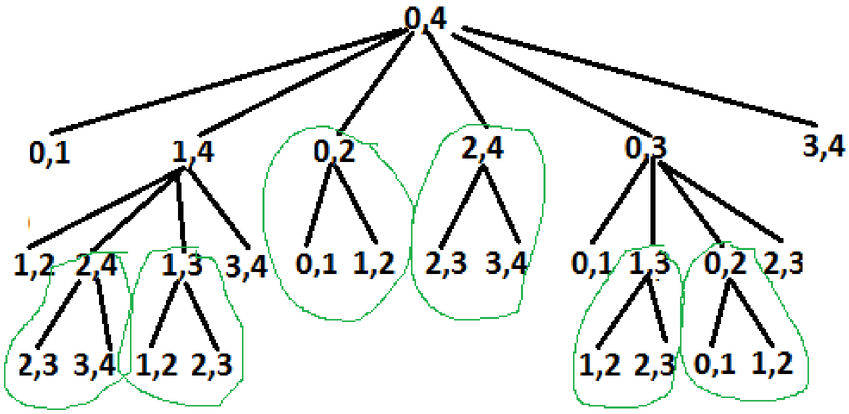
```
// Recursive implementation for minimum cost convex polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}
```

```
// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A recursive function to find minimum cost of polygon triangulation
// The polygon is represented by points[i..j].
double mTC(Point points[], int i, int j)
{
   // There must be at least three points between i and j
   // (including i and j)
   if (j < i+2)
      return 0;

   // Initialize result as infinite
   double res = MAX;

   // Find minimum triangulation by considering all
   for (int k=i+1; k<j; k++)
        res = min(res, (mTC(points, i, k) + mTC(points, k, j) +
                        cost(points, i, k, j)));
   return  res;
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTC(points, 0, n-1);
    return 0;
}
```

Output:

```
15.3006
```

The above problem is similar to Matrix Chain Multiplication. The following is recursion tree for mTC(points[], 0, 4).



Recursion Tree for recursive implementation. Overlapping subproblems are encircled.

It can be easily seen in the above recursion tree that the problem has many overlapping subproblems. Since the problem has both properties: [Optimal Substructure]() and [Overlapping Subproblems](), it can be efficiently solved using dynamic programming.

Following is C++ implementation of dynamic programming solution.

```
// A Dynamic Programming based program to find minimum cost of convex
// polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
```

```
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A Dynamic programming based function to find minimum cost for convex
// polygon triangulation.
double mTCDP(Point points[], int n)
{
   // There must be at least 3 points to form a triangle
   if (n < 3)
      return 0;

   // table to store results of subproblems.  table[i][j] stores cost of
   // triangulation of points from i to j.  The entry table[0][n-1] stores
   // the final result.
   double table[n][n];

   // Fill table using above recursive formula. Note that the table
   // is filled in diagonal fashion i.e., from diagonal elements to
   // table[0][n-1] which is the result.
   for (int gap = 0; gap < n; gap++)
   {
      for (int i = 0, j = gap; j < n; i++, j++)
      {
          if (j < i+2)
             table[i][j] = 0.0;
          else
          {
              table[i][j] = MAX;
              for (int k = i+1; k < j; k++)
              {
                double val = table[i][k] + table[k][j] + cost(points,i,j,k);
                if (table[i][j] > val)
                     table[i][j] = val;
              }
          }
      }
   }
   return  table[0][n-1];
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTCDP(points, n);
    return 0;
}
```

Output:

```
15.3006
```

Time complexity of the above dynamic programming solution is $O(n^3)$.

Please note that the above implementations assume that the points of covnvex polygon are given in order (either clockwise or anticlockwise)

Exercise:
Extend the above solution to print triangulation also. For the above example, the optimal triangulation is 0 3 4, 0 1 3, and 1 2 3.

Sources:
http://www.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture12.html
http://www.cs.utoronto.ca/~heap/Courses/270F02/A4/chains/node2.html

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming, geometric algorithms

Writing code in comment? Please use **ideone.com** and share the link here.

rverma • a month ago

Nice problem and i have another solution which has TC O(n^2) and auxillery space is O(n).
Approach is
start from counter clockwise
e.g take table as temp 2d array of colum are n-1,row=n;
for gap=0
table[i][gap]=0.0
for gap=1

i=0,0 j=1,0 k=2,1
table[i][gap]=table[i][gap-1]+cost(p,i,j,k,n); (Mean for ith point(0 point) take gap=1)

for gap=2
i=0,0 j=2,1 k=1,2 i=0,j=i+gap,k=j+1
table[i][gap]=table[i][gap-1]+cost(p,i,j,k,n);(table[i][gap-1] is ith point has cost at gap-1 +cost(p,i,j,k,n) is current triangle cost)=cost of triangle of for ith point at gap=2

for(gap=0;gap<n-1;gap++){ for(i="0;i<n;i++){" if(gap="">0){
j=i+gap
k=j+1

see more

Reply • Share ›

anonymous123 • 2 months ago

A really nice problem and a tricky solution. Can you please also explain how is the condition -' Diagonals should never intersect' handled in this code.

Reply • Share ›

Sanket Patel → anonymous123 • 2 months ago

Your choice of subproblems for a given k for DP[i][j] implies that diagonals don't intersect.

Polygons p[i..k] and p[k+1..j] are disjoint.

Reply • Share ›

amit gupta • 4 months ago

Cost of a triangle formed by edges (i, j), (j, k) and (k, j) is
cost(i, j, k) = dist(i, j) + dist(j, k) + dist(k, j)
This is wrong ...
It should be

Cost of a triangle formed by edges (i, j), (j, k) and (k, i) is
cost(i, j, k) = dist(i, j) + dist(j, k) + dist(k, i)

2 • Reply • Share ›

coder • 5 months ago

isn't there an error in the original pseudo-code where the condition j < i+2 is given instead as j <= i+2

Reply • Share ›

codem • 6 months ago

Really good problem. I will use this on my website www.findyourcoaching.com

• Reply • Share ›

flop coder • 7 months ago

O(n^2) solution:

http://ideone.com/I8WQqg

• Reply • Share ›

RK- An Unproven Theorem → flop coder • 6 months ago

Can you please share your code? It's not showing on ideone. Or please share the algorithm, how to solve this problem with the complexity O(n^2)?

• Reply • Share ›

flop coder → RK- An Unproven Theorem • 6 months ago

Dunno what's wrong with that ideone link. Anyways, the algo is as follows.

Take the above picture as example .Start denoting points from the lower left corner and travel counter clockwise as 0,1,2...etc. Let there be n number of points.

```
def successor i:
if i==0:
return (n-1)
else:
return (i+1)

for p in 0,1,...,(n-1):
sum=0
for i in 0,1,....(n-1):
if i != p and (i+1) != p:
sum = sum + length(p,i) + length(i,successor(i)) + length(successor(i),p)
check if sum is greater than previous sum
```

• Reply • Share ›

Kenneth • 7 months ago

Very interesting problem and thanks for your sharing solution.
Here is my DP solution, which is actually the same idea with yours:

http://ideone.com/tDLKMI

• Reply • Share ›

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding "extern" keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

    - lt_k

      i need help for coding this function in java...

      Java Programming Language · 1 hour ago

    - Piyush

      What is the purpose of else if (recStack[*i])...

Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago