2CE339

Analysis and Design of Algorithm

Innovative Assignment

Triangulation

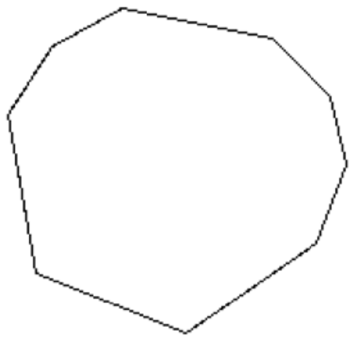
By 11bce078,12bce153,12bce157

Introduction

Triangulation means partition a polygon P into non-overlapping triangles using diagonals only. Triangulation of a polygon is the set of chords that divide the polygon into triangles such that no two chords intersect except possibly at a vertex. Let v0, v1, ..., vn-1 be the vertices of a convex polygon with n vertices(i.e. n-gon). This polygon can be divided into n-2 triangles by a set of n-3 non-crossing chords. This set of n-3 non-crossing chords is called a triangulation of the n-gon.

A triangulation of a polygon is a set of T chords of the polygon that divide the polygon into disjoint triangles. In the set T of chords is maximal: every chord not in T intersects some chord in T. The sides of triangles produced by the triangulation are either chords in the triangulation or sides of the polygon.

Every simple polygon admits a triangulation. Every triangulation of n-gon has exactlyn-2 triangles.



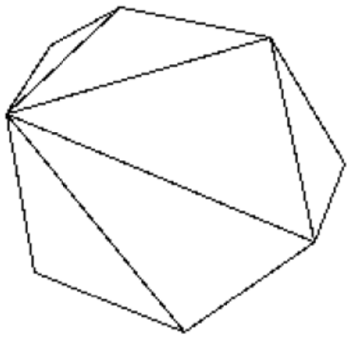


Fig. Convex Polygon Fig. Triangulation of the Polygon

Optimal Triangulation

The optimal polygon triangulation problem for a convex polygon is an optimization problem to find a triangulation with minimum total weight. It is known that this problem can be solved using the dynamic programming technique in O(n3) time using a work space of size O(n2).

An optimal triangulation is one that minimizes some cost function of the triangles. A common cost function is the sum of the lengths of the legs of the triangles, i.e.

w(Delta vivjvk) = |vivj| + |vjvk| + |vkvi|

Let T be the set of all triangulations of a convex n-gon and t(belongs to T) be a triangulation that is a set of n-3 non-crossing chords. The optimal polygon triangulation problem requires finding the triangulation t from the set T which has the minimium cost as per the cost function specified. So, if the cost function is the sum of the lengths of the legs of the triangle, then the optimal triangulation is given by

$$\min\{ \sum_{v_i v_j \in t} w_{i,j} \mid t \in T \},$$

Features of Triangulation

The number of triangulation is depends on the choice of triangulation.
Triangulation reduces the complex shape to collection of simpler shapes.
Triangulations are usually not unique.

Applications of Triangulation

The application of triangulation visibility, robotics, mesh generation, location etc..

Techniques for Solution

1. Greedy Strategy
2. Dynamic Programming Approach

Greedy Strategy

Always add shortest diagonal, consistent with previous selections.

Example:

If the polygon is P = (A, B, C, D, E),

Let A = (0, 0);

B = (50, 25);

C = (80, 30);

D = (125, 25);

E = (160, 0).

The edge lengths are BD = 75; CE < 86; AC < 86;

BE > 112; AD > 127.

Greedy algorithm puts BD, then forced to use BE, for total

weight = 187.

But the Optimal uses would be AC, CE, with total weight = 172.

Dynamic Program Approach

Consider the requirement of an optimal triangulation of a polygon <vi-1,vi,vi+1,…,vj>. Let t(i,j,) be the cost of an optimal triangulation of this polygon. So,

t(i,j)=0,if i=j

min*i*<=*k*<=*j*-1

{

t(*i*,*k*) + t(*k*+1,*j*) + cost (<*vi*-1, *vk*, *vj*>)

},if i < j

So,if we have a single line segment, that is, we're considering the "polygon" <*vi*-1, *vj*>, so *i*=*j*, then *t*(*i*, *j*) is just 0. Otherwise, we let *k* range from *i* to *j*-1, looking at the sum of the costs of all triangles <*vi*-1, *vk*, *vj*> and all polygons <*vi*-1, ..., *vk*> and <*vk*+1, ..., *vj*> and finding the minimum. Then *t*(1, *n*) is the cost of an optimal triangulation for the entire polygon <v0,v1,v2,…vn>.
So, up to n recursive calls are required here per invocation. Many redundant computations need to be done. Hence, this situation is perfectly suited for dynamic programming. Once the value of t(i,j) is found, its value is stored in a two-dimensional array so that its value need be computed again later. Its value can be directly used from this array.

Algorithm

Here, to find an optimal triangulation of a convex polygon with n+1 sides, the dynamic programming approach is used. Let the polygon be <v0,v1,v2,...vn>. A two-dimensional array memo_t of size n*n is used to store the values of t(i,j) that have already been computed. Each element of this array consists of a pair: the value representing the minimum cost and the index of the vertex of the polygon for which the minimum cost is found. Initially, all the values are set to -1 and as and when the values are computed, they are stored into this array. So to compute the optimal triangulation of the specified polygon, this algorithm is called with the values i=1 and j=n. To compute the value of any t(i,j), the array memo_t is first checked to see if the value has already been computed. If so, it is used. Otherwise its value is computed using the formula mentioned before. This value is stored in the array and then used. Thus, the dynamic programming approach reduces the number of redundant computations carried out in this problem.

Greedy Program

```
#include<stdio.h>

//#include<graphics.h>

#include<math.h>

#include<sys/types.h>


struct minimum

{

int c;

int k;

}***m;
```

```
struct point
{
int x;
int y;
}**v;


int t(int i,int j,int n)
{
int min = 1000,k,x,mink;
printf("t(%d,%d)\n",i,j);
        if( i == j)return 0;
        for (k=i;k<=j-1;k++)
        {
x =  cost ((i-1)%(n+1),k,j);
        if( x < min)
        {
        min = x;
        mink=k;
        }
        }
        x=t (i, mink,n)+t ((mink+1)%(n+1), j,n) + min;
        m[i][j]->c = min;
        m[i][j]->k=mink;
        return x;
}
int cost(int i,int j,int k)
{
int x,y,z;
x=((v[i]->x-v[j]->x)*(v[i]->x-v[j]->x)+(v[i]->y-v[j]->y)*(v[i]->y-v[j]->y))^(1/2);
y=((v[k]->x-v[j]->x)*(v[k]->x-v[j]->x)+(v[k]->y-v[j]->y)*(v[k]->y-v[j]->y))^(1/2);
z=((v[i]->x-v[k]->x)*(v[i]->x-v[k]->x)+(v[i]->y-v[k]->y)*(v[i]->y-v[k]->y))^(1/2);
printf("Cost is:%d\n",x+y+z);
return (x+y+z);
}

/*
void draw(int n)
{
int i,j;
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);
for(i=0;i<n-1;i++)
{
line(v[i]->x,v[i]->y,v[i+1]->x,v[i+1]->y);
}
line(v[n-1]->x,v[n-1]->y,v[0]->x,v[0]->y);
```

```
for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

if(i!=j)

{

line(v[i]->x,v[i]->y,v[j]->x,v[j]->y);

line(v[i]->x,v[i]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

line(v[j]->x,v[j]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

}

}

}

*/


/*

draw_t(int i,int j,int n)

{

        if(i != j)

        {

line(v[(i-1)%(n+1)]->x,v[(i-1)%(n+1)]->y,v[j]->x,v[j]->y);

line(v[(i-1)%(n+1)]->x,v[(i-1)%(n+1)]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

line(v[j]->x,v[j]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

        draw_t (i, m[i][j]->k,n);

        draw_t ((m[i][j]->k+1)%(n+1), j,n);
}

}

*/


int main()

{

//int gdriver=DETECT,gmode;

int n,i,j,weight=0;

int fd[2];

long int sttime,ftime,ttime;

printf("\n Greedy Approach \n");

printf("\n Enter the number of vertices :: ");

scanf("%d",&n);

m=(struct minimum ***)malloc(n*sizeof(struct minimum));

for(i=0;i<n;i++)

{

m[i]=(struct minimum **)malloc(n*sizeof(struct minimum));

}

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

m[i][j]=(struct minimum *)malloc(sizeof(struct minimum));

}
```

```
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
m[i][j]->c=-1;
}

v=(struct point **)malloc(n*sizeof(struct point));

for(i=0;i<n;i++)
{
printf("\n Enter the points x and y:");
v[i]=(struct point *)malloc(sizeof(struct point));
scanf("%d %d",&v[i]->x,&v[i]->y);
}
for(i=0;i<n;i++)
{
printf("The value x:%d y:%d\n",v[i]->x,v[i]->y);
}
fd[0]=creat("start",0666);
fd[1]=creat("end",0666);
system("date +%N > start");
weight = t (1, n-1,n-1);
```

```
system("date +%N > end");

read(fd[0],&sttime,sizeof(sttime));

read(fd[1],&ftime,sizeof(ftime));

ttime=ftime-sttime;

printf("The min weight is: %d\n",weight);

//initgraph(&gd,&gm,NULL);

//draw_t(1,n-1,n-1);

//getch();

return;

}
```

Note : Remove red color comment to see how the graph is look like. Graph can be seen by executing this program (remove comment) on turbo C. And this program we have executed on linux platform to see the execution time of both program.

Dynamic Program

```
#include<stdio.h>

//#include<graphics.h>

#include<math.h>


struct point

{

int x;

int y;

}**v;
```

```
struct minimum

{

int c;

int k;

}***m;


int t(int i,int j,int n)

{

int min = 1000,k,x,mink;

printf("t(%d,%d)\n",i,j);

        if( i == j)

        return 0;

        if (m[i][j]->c != -1) return(m[i][j]->c);

        for (k=i;k<=j-1;k++)

        {

        x = t (i, k,n) + t ((k+1)%(n+1), j,n)

                + cost ((i-1)%(n+1),k,j);

        if( x < min)

        {

        min = x;

        mink=k;

        }
        }
        m[i][j]->c = min;
        m[i][j]->k=mink;
        return min;
}

int cost(int i,int j,int k)
{
int x,y,z;
x=((v[i]->x-v[j]->x)*(v[i]->x-v[j]->x)+(v[i]->y-v[j]->y)*(v[i]->y-v[j]->y))^(1/2);
y=((v[k]->x-v[j]->x)*(v[k]->x-v[j]->x)+(v[k]->y-v[j]->y)*(v[k]->y-v[j]->y))^(1/2);
z=((v[i]->x-v[k]->x)*(v[i]->x-v[k]->x)+(v[i]->y-v[k]->y)*(v[i]->y-v[k]->y))^(1/2);
printf("Cost is:%d\n",x+y+z);
return (x+y+z);
}

/*
void draw(int n)
{
int i,j;
int gdriver=DETECT,gmode;
initgraph(&gd,&gm,NULL);
for(i=0;i<n-1;i++)
```

```
{
line(v[i]->x,v[i]->y,v[i+1]->x,v[i+1]->y);
}
line(v[n-1]->x,v[n-1]->y,v[0]->x,v[0]->y);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
if(i!=j)
{
line(v[i]->x,v[i]->y,v[j]->x,v[j]->y);
line(v[i]->x,v[i]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);
line(v[j]->x,v[j]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);
}
}
}
*/


/*
drawnew(int i,int j,int n)
          {
          if(i != j)
          {
line(v[(i-1)%(n+1)]->x,v[(i-1)%(n+1)]->y,v[j]->x,v[j]->y);
line(v[(i-1)%(n+1)]->x,v[(i-1)%(n+1)]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

line(v[j]->x,v[j]->y,v[m[i][j]->k]->x,v[m[i][j]->k]->y);

        drawnew (i, m[i][j]->k,n);

        drawnew ((m[i][j]->k+1)%(n+1), j,n);

}


getch();

}

*/


int main()

{

//int gdriver=DETECT,gmode;

int n,i,j,weight=0;

int fd[2];

long int sttime,ftime,ttime;

printf("\n Dynamic Programming ");

printf("\n Enter the number of vertices :: ");

scanf("%d",&n);

m=(struct minimum ***)malloc(n*sizeof(struct minimum));

for(i=0;i<n;i++)

{

m[i]=(struct minimum **)malloc(n*sizeof(struct minimum));
```

```
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
m[i][j]=(struct minimum *)malloc(sizeof(struct minimum));
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
m[i][j]->c=-1;
}
v=(struct point **)malloc(n*sizeof(struct point));
for(i=0;i<n;i++)
{
printf("\n Enter the points x and y: ");
v[i]=(struct point *)malloc(sizeof(struct point));
scanf("%d %d",&v[i]->x,&v[i]->y);
}
for(i=0;i<n;i++)
{
printf("The value x:%d y:%d\n",v[i]->x,v[i]->y);
}
fd[0]=creat("start",0666);
```

```
fd[1]=creat("end",0666);

system("date +%N > start");

weight = t (1, n-1,n-1);

system("date +%N > end");

read(fd[0],&sttime,sizeof(sttime));

read(fd[1],&ftime,sizeof(ftime));

ttime=ftime-sttime;

printf("The min weight is: %d\n",weight);

//initgraph(&gd,&gm,NULL);

//drawnew(1,n-1,n-1);

//getch();

return;

}
```

Note : Remove red color comment to see how the graph is look like. Graph can be seen by executing this program (remove comment) on turbo C. And this program we have executed on linux platform to see the execution time of both program.

Output

```
yamini@ubuntu:~$ gcc greedy.c

greedy.c: In function 'main':

greedy.c:95: warning: incompatible implicit declaration of built-in function 'malloc'

yamini@ubuntu:~$ time ./a.out
```

```
 Greedy Approach


 Enter the number of vertices :: 3


 Enter the points x and y:20

30


 Enter the points x and y:50

70


 Enter the points x and y:40

90

The value x:20 y:30

The value x:50 y:70

The value x:40 y:90

t(1,2)

Cost is:7000

t(1,-1074784776)

t(1,-1216451800)

t(1,15651904)

Segmentation fault


real	0m27.750s
```

```
user      0m0.000s

sys       0m0.040s

yamini@ubuntu:~$ time ./a.out


 Greedy Approach


 Enter the number of vertices :: 4


 Enter the points x and y:40

50


 Enter the points x and y:30

70


 Enter the points x and y:100

150


 Enter the points x and y:70

120

The value x:40 y:50

The value x:30 y:70

The value x:100 y:150

The value x:70 y:120

t(1,3)
```

```
Cost is:10400

Cost is:21200

t(1,-1074126152)

t(1,-1216029912)

t(1,15692864)

Segmentation fault


real	0m39.389s

user	0m0.000s

sys	0m0.016s

yamini@ubuntu:~$ time ./a.out


 Greedy Approach


 Enter the number of vertices :: 5


 Enter the points x and y:20

50


 Enter the points x and y:70

100


 Enter the points x and y:40

150
```

```
 Enter the points x and y:90
120


 Enter the points x and y:30
80
The value x:20 y:50
The value x:70 y:100
The value x:40 y:150
The value x:90 y:120
The value x:30 y:80
t(1,4)
Cost is:8000
Cost is:16400
Cost is:16000
t(1,-1078154504)
t(1,-1216640216)
t(1,14373952)
Segmentation fault


real	0m32.561s
user	0m0.000s
sys	0m0.016s
yamini@ubuntu:~$ gcc dynamic.c
```

```
dynamic.c: In function ‘main’:

dynamic.c:98: warning: incompatible implicit declaration of built-in function ‘malloc’

yamini@ubuntu:~$ time ./a.out


 Dynamic Programming

 Enter the number of vertices :: 3


 Enter the points x and y: 20

30


 Enter the points x and y: 50

70


 Enter the points x and y: 40

90

The value x:20 y:30

The value x:50 y:70

The value x:40 y:90

t(1,2)

t(1,1)

t(2,2)

Cost is:7000

The min weight is: 1000
```

```
real	0m23.071s

user	0m0.000s

sys	0m0.020s

yamini@ubuntu:~$ time ./a.out


 Dynamic Programming

 Enter the number of vertices :: 4


 Enter the points x and y: 40

50


 Enter the points x and y: 30

70


 Enter the points x and y: 100

150


 Enter the points x and y: 70

120

The value x:40 y:50

The value x:30 y:70

The value x:100 y:150

The value x:70 y:120

t(1,3)
```

```
t(1,1)

t(2,3)

t(2,2)

t(3,3)

Cost is:17200

Cost is:10400

t(1,2)

t(1,1)

t(2,2)

Cost is:25400

t(3,3)

Cost is:21200

The min weight is: 1000


real    0m45.307s

user    0m0.004s

sys     0m0.020s

yamini@ubuntu:~$ time ./a.out


 Dynamic Programming

 Enter the number of vertices :: 5


 Enter the points x and y: 20

50
```

```
 Enter the points x and y: 70
100


 Enter the points x and y: 40
150


 Enter the points x and y: 90
120


 Enter the points x and y: 30
80
The value x:20 y:50
The value x:70 y:100
The value x:40 y:150
The value x:90 y:120
The value x:30 y:80
t(1,4)
t(1,1)
t(2,4)
t(2,2)
t(3,4)
t(3,3)
t(4,4)
```

```
Cost is:13600

Cost is:10400

t(2,3)

t(2,2)

t(3,3)

Cost is:7600

t(4,4)

Cost is:8000

Cost is:8000

t(1,2)

t(1,1)

t(2,2)

Cost is:18800

t(3,4)

Cost is:16400

t(1,3)

t(1,1)

t(2,3)

Cost is:15600

t(1,2)

t(3,3)

Cost is:23600

t(4,4)

Cost is:16000
```

```
The min weight is: 1000

real	0m30.921s
user	0m0.004s
sys	0m0.016s
yamini@ubuntu:~$
```

Conclusion

Execution Time	Greedy	Dynamic
Real	27.750s	23.071s
User	0s	0s
System	0.040s	0.020s
Vertices : 3		
Real	39.3895s	45.307s
User	0s	0.004s
System	0.016s	0.020s
Vertices : 4		
Real	32.561s	30.921s
User	0.004s	0.004s
System	0.016s	0.016s
Vertices : 5		