Representação de ponto

Ponto: vetor de dimensão apropriada

Representação de ponto

Ponto: vetor de dimensão apropriada

Ficar nos inteiros enquanto for possível

Representação de ponto

Ponto: vetor de dimensão apropriada

Ficar nos inteiros enquanto for possível

```
#define X 0
#define Y 1
#define DIM 2 /* dimensão do espaço */

/* tipo ponto inteiro */
typedef int tPointi[DIM];

/* tipo ponto real */
typedef double tPointd[DIM];
```


Representação de polígono

Polígono: vetor ou lista ligada de pontos

Representação de polígono

Polígono: vetor ou lista ligada de pontos

Qual das duas opções escolher? Depende...

Representação de polígono

Polígono: vetor ou lista ligada de pontos

Qual das duas opções escolher? Depende...

Com vetor...

```
/* número máximo de pontos em um polígono */
#define PMAX 1000

/* tipo polígono de pontos inteiros */
typedef tPointi tPolygoni[PMAX];

/* tipo polígono de pontos reais */
typedef tPointd tPolygond[PMAX];
```


Cálculos de área

Triângulo

Cálculos de área

Triângulo

```
int Area2 (tPointi a, b, c) {
   return a[X]*b[Y]-a[Y]*b[X]+a[Y]*c[X]
         -a[X]*c[Y]+b[X]*c[Y]-c[X]*b[Y];
}
```

Cálculos de área

Triângulo

```
int Area2 (tPointi a, b, c) {
   return a[X]*b[Y]-a[Y]*b[X]+a[Y]*c[X]
         -a[X]*c[Y]+b[X]*c[Y]-c[X]*b[Y];
}
```

Com menos multiplicações e em pseudocódigo:

Area2(a, b, c)
1 **devolva** $(a[X] - c[X]) * (b[Y] - c[Y]) -$
$(a[Y] - c[Y]) * (b[X] - c[X])$

Cálculos de área

Triângulo

Area2(a, b, c)
1 **devolva** $(a[X] - c[X]) * (b[Y] - c[Y]) -$
$(a[Y] - c[Y]) * (b[X] - c[X])$

Polígono

Cálculos de área

Triângulo

Area2(a, b, c)
1 **devolva** $(a[X] - c[X]) * (b[Y] - c[Y]) -$
$(a[Y] - c[Y]) * (b[X] - c[X])$

Polígono

AreaPoly2(n, P)
1 $s \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** $n - 2$ **faça**
3 $s \leftarrow s+$Area2$(P[0], P[i], P[i + 1])$
4 **devolva** s

Segmentos e pontos

Ponto c à esquerda da reta dada por $\vec{ab}$

Left(a, b, c)
1 **devolva** Area2$(a, b, c) > 0$

Segmentos e pontos

Ponto c à esquerda da reta dada por $\vec{ab}$

Left(a, b, c)
1 **devolva** Area2$(a, b, c) > 0$

Ponto c à esquerda ou sobre a reta dada por $\vec{ab}$

LeftOn(a, b, c)
1 **devolva** Area2$(a, b, c) \geq 0$

Segmentos e pontos

Ponto c à esquerda da reta dada por $\vec{ab}$

Left(a, b, c)
1 **devolva** Area2$(a, b, c) > 0$

Ponto c à esquerda ou sobre a reta dada por $\vec{ab}$

LeftOn(a, b, c)
1 **devolva** Area2$(a, b, c) \geq 0$

Pontos a, b e c são colineares

Collinear(a, b, c)
1 **devolva** Area2$(a, b, c) = 0$

Interseção de segmentos

Interseção própria entre ab e cd

IntersectProp(a, b, c, d)
1 **se** Collinear(a, b, c) **ou** Collinear(a, b, d)
ou Collinear(c, d, a) **ou** Collinear(c, d, b)
2 **então devolva** FALSO
3 **devolva** Xor(Left(a, b, c),Left(a, b, d))
e Xor(Left(c, d, a),Left(c, d, b))

A rotina Xor devolve o ou exclusivo entre duas expressões booleanas.

Interseção de segmentos

Ponto c está no segmento ab

Between(a, b, c)
1 **se não** Collinear(a, b, c)
2 **então devolva** FALSO
3 **se** $a[X] \neq b[X]$ ▷ ab não é vertical
4 **então devolva** $a[X] \leq c[X] \leq b[X]$ **ou** $b[X] \leq c[X] \leq a[X]$
5 **senão devolva** $a[Y] \leq c[Y] \leq b[Y]$ **ou** $b[Y] \leq c[Y] \leq a[Y]$

Interseção de segmentos

Ponto c está no segmento ab

```
Between(a, b, c)
1  se não Collinear(a, b, c)
2        então devolva FALSO
3  se a[X] ≠ b[X]          ▷ ab não é vertical
4        então devolva a[X] ≤ c[X] ≤ b[X] ou b[X] ≤ c[X] ≤ a[X]
5        senão devolva a[Y] ≤ c[Y] ≤ b[Y] ou b[Y] ≤ c[Y] ≤ a[Y]
```

Interseção entre ab e cd

```
Intersect(a, b, c, d)
1  se IntersectProp(a, b, c, d)
2        então devolva VERDADE
3  devolva Between(a, b, c) ou Between(a, b, d)
           ou Between(c, d, a) ou Between(c, d, b)
```


Dentro ou fora?

Candidata a diagonal $p[i]p[j]$ está no interior do polígono?

Está no cone das arestas vizinhas do polígono?

```
InCone(n, P, i, j)
1  u ← i − 1 (mod n)
2  w ← i + 1 (mod n)
3  se LeftOn(P[u], P[i], P[w])        ▷ P[i] é convexo
4        então devolva Left(P[i], P[j], P[u]) e
                       Left(P[j], P[i], P[w])
5        senão devolva não (LeftOn(P[i], P[j], P[w]) e
                            LeftOn(P[j], P[i], P[u]))
```


Teste de diagonal

Quase uma diagonal...

```
Diagonalie(n, P, i, j)
1  para k ← 0 até n − 1
2        kp ← k + 1 (mod n)
3        se k ≠ i e k ≠ j e kp ≠ i e kp ≠ j
4              então se Intersect(P[i], P[j], P[k], P[kp])
5                    então devolva FALSO
6  devolva VERDADE
```


Teste de diagonal

Quase uma diagonal...

```
Diagonalie(n, P, i, j)
1  para k ← 0 até n − 1
2        kp ← k + 1 (mod n)
3        se k ≠ i e k ≠ j e kp ≠ i e kp ≠ j
4              então se Intersect(P[i], P[j], P[k], P[kp])
5                    então devolva FALSO
6  devolva VERDADE
```

Diagonal de fato...

```
Diagonal(n, P, i, j)
1  devolva InCone(n, P, i, j) e Diagonalie(n, P, i, j)
```


Teste de diagonal

Quase uma diagonal...

```
Diagonalie(n, P, i, j)
1  para k ← 0 até n − 1
2        kp ← k + 1 (mod n)
3        se k ≠ i e k ≠ j e kp ≠ i e kp ≠ j
4              então se Intersect(P[i], P[j], P[k], P[kp])
5                    então devolva FALSO
6  devolva VERDADE
```

Diagonal de fato...

```
Diagonal(n, P, i, j)
1  devolva InCone(n, P, i, j) e Diagonalie(n, P, i, j)
```

Tempo de execução: $\Theta(n)$

Triangulação em $O(n^4)$

```
Triang-n4(n, P)
 1  se n > 3
 2        então i ← 0        j ← 2
 3              enquanto não Diagonal(n, P, i, j) faça
 4                    j ← j + 1
 5                    se j = n
 6                          então i ← i + 1
 7                                j ← i + 2
 8              imprima {i, j}
 9              n1 ← j − i + 1
10              n2 ← n − n1 + 2
11              P1[0 . . n1 − 1] ← P[i . . j]
12              P2[0 . . n2 − 1] ← P[0 . . i] · P[j . . n − 1]
13              Triang-n4(n1, P1)
14              Triang-n4(n2, P2)
```


Triangulação em $O(n^4)$

```
Triang-n4(n, P)
 1  se n > 3
 2        então i ← 0        j ← 2
 3              enquanto não Diagonal(n, P, i, j) faça
 4                    j ← j + 1
 5                    se j = n
 6                          então i ← i + 1
 7                                j ← i + 2
 8              imprima {i, j}
 9              n1 ← j − i + 1
10              n2 ← n − n1 + 2
11              P1[0 . . n1 − 1] ← P[i . . j]
12              P2[0 . . n2 − 1] ← P[0 . . i] · P[j . . n − 1]
13              Triang-n4(n1, P1)
14              Triang-n4(n2, P2)
```

Pior caso: $\Theta(n^3)$ chamadas a Diagonal

Triangulação em $O(n^3)$: use orelhas!

```
PontaDeOrelha(n, P, i)
1  j ← (i − 1) mod n
2  k ← (i + 1) mod n
3  devolva Diagonal(n, P, j, k)
```

Triangulação em $O(n^3)$: use orelhas!

```
PontaDeOrelha(n, P, i)
1  j ← (i − 1) mod n
2  k ← (i + 1) mod n
3  devolva Diagonal(n, P, j, k)

Triang-n3(n, P)
1  se n > 3
2      então i ← 0
3            enquanto não PontaDeOrelha(n, P, i) faça
4                i ← i + 1
5            m, P' ← Remove(n, P, i)
6            Triang-n3(m, P')
7            imprima {(i − 1) mod n, (i + 1) mod n}
```

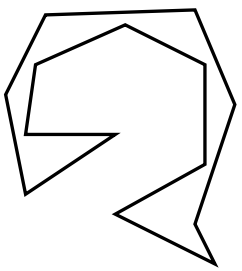

Triangulação em $O(n^3)$: use orelhas!

```
PontaDeOrelha(n, P, i)
1  j ← (i − 1) mod n
2  k ← (i + 1) mod n
3  devolva Diagonal(n, P, j, k)

Triang-n3(n, P)
1  se n > 3
2      então i ← 0
3            enquanto não PontaDeOrelha(n, P, i) faça
4                i ← i + 1
5            m, P' ← Remove(n, P, i)
6            Triang-n3(m, P')
7            imprima {(i − 1) mod n, (i + 1) mod n}
```


Pior caso: $\Theta(n^2)$ chamadas a Diagonal

Exemplo ruim



Triangulação em $O(n^2)$

```
Triang-n2(n, P)
 1  MarcaOrelha(n, P)
 2  enquanto n > 3 faça
 3      v2 ← P
 4      enquanto não orelha[v2] faça
 5          v2 ← prox[v2]
 6      v1 ← prev[v2]
 7      v3 ← prox[v2]
 8      imprima {v1, v3}
 9      prox[v1] ← v3
10      prev[v3] ← v1
11      n ← n − 1
12      P ← v3
13      orelha[v1] ← PontaDeOrelha(n, P, v1)
14      orelha[v3] ← PontaDeOrelha(n, P, v3)
```


Triangulação em $O(n^2)$

```
Triang-n2(n, P)
 1  MarcaOrelha(n, P)
 2  enquanto n > 3 faça
 3      v2 ← P
 4      enquanto não orelha[v2] faça
 5          v2 ← prox[v2]
 6      v1 ← prev[v2]
 7      v3 ← prox[v2]
 8      imprima {v1, v3}
 9      prox[v1] ← v3
10      prev[v3] ← v1
11      n ← n − 1
12      P ← v3
13      orelha[v1] ← PontaDeOrelha(n, P, v1)
14      orelha[v3] ← PontaDeOrelha(n, P, v3)
```

Pior caso: $\Theta(n^2)$ chamadas a Diagonal

Orelhas com listas ligadas

```
PontaDeOrelha(n, P, v)
1  u ← prev[v]
2  w ← prox[v]
3  devolva Diagonal(n, P, u, w)
```


Orelhas com listas ligadas

```
PontaDeOrelha(n, P, v)
1  u ← prev[v]
2  w ← prox[v]
3  devolva Diagonal(n, P, u, w)
```


```
MarcaOrelha(n, P)
1  v ← P
2  para i = 1 até n faça
3      u ← prev[v]
4      w ← prox[v]
5      orelha[v] ← Diagonal(n, P, u, w)
6      v ← w
```

Orelhas com listas ligadas

```
PontaDeOrelha(n, P, v)
1  u ← prev[v]
2  w ← prox[v]
3  devolva Diagonal(n, P, u, w)

MarcaOrelha(n, P)
1  v ← P
2  para i = 1 até n faça
3      u ← prev[v]
4      w ← prox[v]
5      orelha[v] ← Diagonal(n, P, u, w)
6      v ← w
```


Diagonal também teria que ser reescrita para listas ligadas.