MAC 331 - LISTA (3)

DANIELA GONZALEZ FAVERO - 10277443

(1) Para a construção da árvore, basta guardar cada segmento num nó, ordenando segundo a x-coordenada do extremo do segmento que se encontra na reta $y=1$. É necessário construir uma árvore rubro-negra, afinal a busca precisará ser em $\lg n$. A construção de uma árvore rubro-negra a partir de um vetor desordenado custa $n \lg n$.

Para a busca na árvore, basta inserir o ponto procurado como um segmento de tamanho 0 na árvore (inserção custa $\lg n$) e buscar o predecessor e o sucessor na árvore (ambas as operações custam $\lg n$). Por fim, é necessário fazer um teste de esquerda para o ponto em relação ao predecessor e outro em relação ao sucessor (testes de esquerda têm custo constante).

(4)
```
MARCA-VIZINHOS (P1, P2)
  v ← P1
  seg ← 0
  repita  △ constrói conjunto de segmentos
    u ← next[v]
    se  v[X] < u[X]
        então  e[seg] ← v
               d[seg] ← u
        senão  e[seg] ← u
               d[seg] ← v
    v ← u
    seg = seg + 1
  até que  v = P1
  para  i ← 0 até seg  △ marca quais segmentos são arestas vizinhas
      viz[i] ← i+1
  viz[seg] ← 0
  △ Fazer o mesmo para P2
  devolva  e, d, viz
```

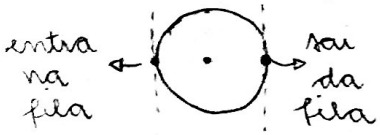
A função MARCA-VIZINHOS consome tempo linear (possui 2 laços que percorrem os dois polígonos).

```
INTERSECÇÃO-SH-VIZ(e, d, n, viz)
    (E, sgm, esq) ← FilaDeEventos(e, d, n)
    Crie(T)
    para p←1 até 2n faça
        i ← sgm[p]
        pred ← Predecessor(T, Ex[p], Ey[p])
        suc ← Sucessor(T, Ex[p], Ey[p])
        se esq[p]
            então Insere(T, i)
                se (pred ≠ NIL e Inter(e, d, i, pred) e viz[pred] ≠ i e viz[i] ≠ pred)
                ou (suc ≠ NIL e Inter(e, d, i, suc) e viz[suc] ≠ i e viz[i] ≠ suc)
                    então devolva verdade
            senão Remove(T, i)
                se (pred ≠ NIL e suc ≠ NIL e Inter(e, d, pred, suc) e viz[suc] ≠ pred
                e viz[pred] ≠ suc)
                    então devolva verdade
    devolva falso
```

A função INTERSECÇÃO-SH-VIZ tem a mesma complexidade do algoritmo original de Shamos e Hoey ($n \lg n$), pois só adicionamos condições de consulta constante (pesquisas no vetor viz).

```
INTERSECÇÃO-DE-POLÍGONOS(P1, P2, n)
    (e, d, viz) ← MARCA-VIZINHOS(P1, P2)
    devolva INTERSECÇÃO-SH-VIZ(e, d, n, viz)
```

(5) Para determinar se há dois discos que se intersectam na coleção, é necessário utilizar um algoritmo de linha de varredura similar ao de Shamos e Hoey. Primeiro ordenamos os discos pela y-coordenada do centro de cada um. Os pontos evento serão as extremidades em x dos discos:



Toda vez que uma circunferência for detectada pela linha de varredura, ela entra na fila de eventos e seu centro é comparado* com os centros da circunferência predecessora e da sucessora em y (que já estão na fila). Toda vez que uma circunferência sai da fila, o centro de suas sucessora e predecessora são comparados*

* A comparação se dá pela distância entre os dois centros ser menor do que a soma dos raios (que, neste caso, seria uma intersecção)

Cada iteração faz uma chamada a predecessor, sucessor, insere ou remove (operações que custam $\lg n$). Como há $2n$ pontos eventos, há $2n$ iterações. Portanto o algoritmo custará $n \lg n$.