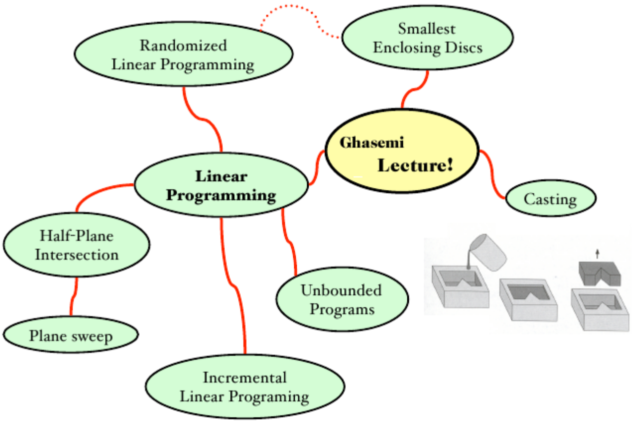
Linear Programming (Manufacturing with Molds)

M.M. Ghaseminya

Department of Computer Science,
Yazd University

1389-2

Lecture Mind Map



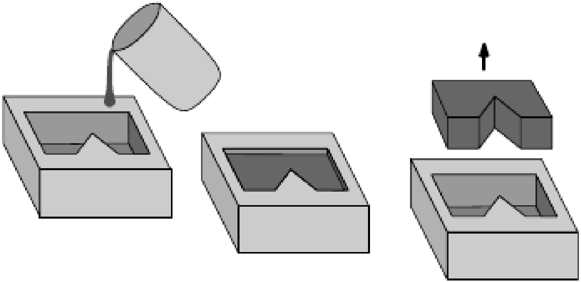
Computational Geometry - Håkan Jons

Introduction

- Computers play an important role in automated manufacturing, both in the design phase and in the construction phase.
- CAD/CAM facilities are a vital part of any modern factory.

Introduction

- Computers play an important role in automated manufacturing, both in the design phase and in the construction phase.
- CAD/CAM facilities are a vital part of any modern factory.



Assumptions

- **We** assume that the object to be constructed is polyhedral.
- **We** only consider molds of one piece, not molds consisting of two or more pieces.
- **We** only allow the object to be removed from the mold by a single translation.

we want to determine whether an object can be manufactured by casting

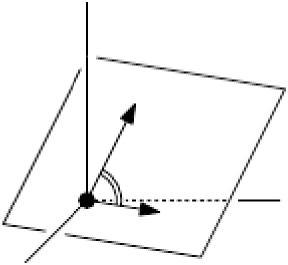
- we have to find a suitable mold for it!
- We call an object **castable** if it can be removed from its mold for at least one of these orientations.

Definition:

- **top facet**:
  One obvious restriction on the orientation is that the object must have a horizontal top facet.
- We call a facet of P that is not the top facet an ordinary facet. Every ordinary facet f has a corresponding facet in the mold, which we denote by f'.

Angles between 3D-vectors!

To get the angle between two vectors u and v, consider the plane they span and pick the smaller angle between the two in this plane. !



A removability criterion

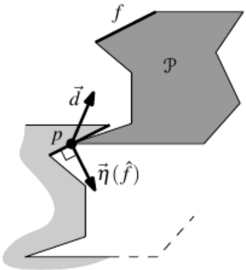
- Let P be casted piece and $\vec{d}$ a direction.
- P can be removed by a translation in direction $\vec{d}$ iff $\vec{d}$ makes an angle of at least 90° outward normal of every ordinary facet of P.

  Except the top-facet.
- **How** do we find such a direction?

A removability criterion

- Let P be casted piece and $\vec{d}$ a direction.
- P can be removed by a translation in direction $\vec{d}$ iff $\vec{d}$ makes an angle of at least 90° outward normal of every ordinary facet of P.

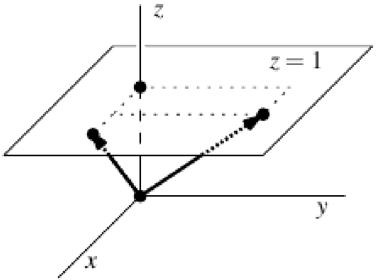
  Except the top-facet.
- **How** do we find such a direction?



Representing directions

- We assume translations in the positive z-direction only.
- The direction of a vector $(x, y, 1)$ is represented by the point $(x, y, 1)$ in the plane $z = 1$.

  So, every point in the plane $z = 1$ now represents a unique direction.



Finding a valid direction!

- A direction $\vec{d} = (d_x, d_y, d_z)$ makes an angle at least 90° with a outward normal $n = (n_x, n_y, n_z)$ of P iff the dot product $\vec{n}.\vec{d} \leq 0$.
- Notice that the inequality defines a half-plane on the plane $z = 1$.
  The line $n_x d_x + n_y d_y + n_z d_z = 0$ splits the plane in two parts, one of which contains all locally valid directions in which P can be translated.

Theorem (4.2)

Let P be a polyhedron with n facets. In $\mathcal{O}(n^2)$ expected time and using $\mathcal{O}(n)$ storage it can be decided whether P is castable.Moreover, if P is castable, a mold and a valid direction for removing P from it can be computed in the same amount of time.

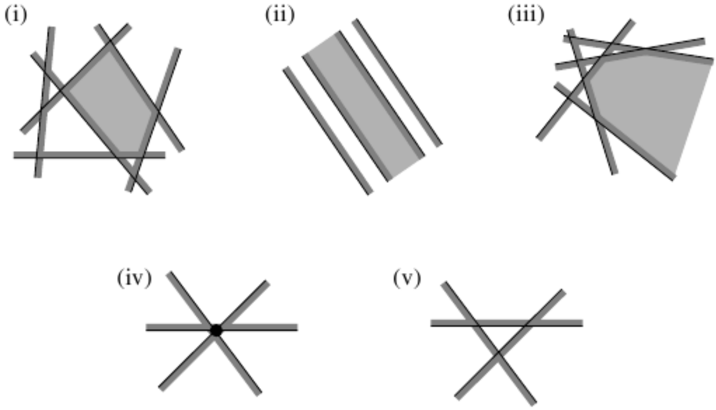
Half-Plane Intersection

- So, our practical problem has turned into the geometric problem of computing the intersection of half-planes.
- A half-plane in the plane (Euclidean, $2D$) is defined by a linear constraint in two variables.

  $a_i x + b_i y \leq 0$
- Given a bunch of half-planes, we consider the problem of finding all points (x, y) that satisfy all constraints.

  The intersection of n half-planes is a convex polygonal region bounded by at most n edges.

Examples of the intersection of half-planes



Computing the intersection

A Divide-and-Conquer algorithm is used:

Algorithm INTERSECTHALFPLANES(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.
1. **if** card(H) = 1
2. **then** $C \leftarrow$ the unique half-plane $h \in H$
3. **else** Split H into sets H_1 and H_2 of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.
4. $C_1 \leftarrow$ INTERSECTHALFPLANES(H_1)
5. $C_2 \leftarrow$ INTERSECTHALFPLANES(H_2)
6. $C \leftarrow$ INTERSECTCONVEXREGIONS(C_1, C_2)

Computing the intersection

- One version of **IntersectConvexRegions** was actually presented in Chapter 2!
  - corollary 2.7 the Intersection of two polygons with n vertices can be computed in $\mathcal{O}(nLogn + kLogn)$ times.
  - Some adjustment needed since we need to compute unbounded regions(not simple polygons)
  - Also in our case ,$k \leq n$ since the region are convex.

$$T(n) = \begin{cases} \mathcal{O}(1) & if n = 1 \\ \mathcal{O}(nLgn) + 2T(\frac{n}{2}) & if n > 1 \end{cases}$$

Total Running time of Algorithm

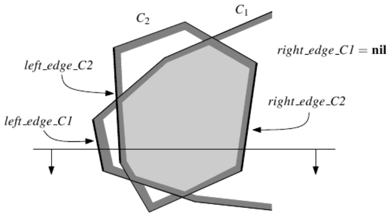
- There is in fact a faster version of IntersectConvexRegions that run in $\mathcal{O}(n)$ time .

  this implies that the complexity gets lowered from $\mathcal{O}(nlog^2n)$ to $\mathcal{O}(nlogn)$
- this version is based on planeSweep.
- while the sweep line is moved downward over the regions using vertices as event point, we keep track of the intersection with the boundaries of two regions.

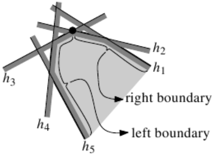
Plane Sweep Algorithm

- To simplify the description of the algorithm, we shall assume that there are no horizontal edges!
- we move a sweep line downward over the plane, and we maintain the edges of **C1** and **C2** ,**intersecting the sweep line.**
- C_1 and C_2 are ?
- So:we simply have pointers $left\_edge\_C_1$, $right\_edge\_C_1$,$left\_edge\_C_2$, and $right\_edge\_C_2$ to them.

- To simplify the description of the algorithm, we shall assume that there are no horizontal edges!
- we move a sweep line downward over the plane, and we maintain the edges of **C1** and **C2** ,**intersecting the sweep line**.
- C_1 and C_2 are ?
- So:we simply have pointers $left\_edge\_C_1$, $right\_edge\_C_1$,$left\_edge\_C_2$, and $right\_edge\_C_2$ to them.



New Plane Sweep Algorithm Object

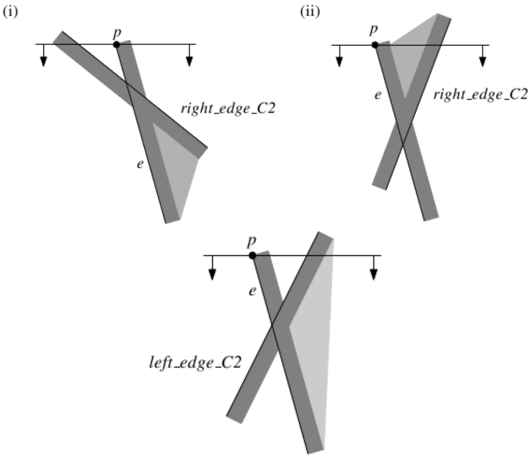


$\mathcal{L}_{\text{left}}(C) = h_3, h_4, h_5$

$\mathcal{L}_{\text{right}}(C) = h_2, h_1$

New Plane Sweep Algorithm Structure

1. Initialize.
2. Status of edges e.
3. Four functions needed.
4. The procedure that handles e will discover three possible edges that C might have.
   - The edge with p as upper endpoint.
   - The edge with $e \cap left\_edge\_C_2$ as upper endpoint.
   - The edge with $e \cap right\_edge\_C_2$ as upper endpoint.



(i)
p
right_edge_C2
e
(ii)
p
e
right_edge_C2
p
e
left_edge_C2

Constant time to handle an edge

The intersection of two convex polygons can be computed in time $\mathcal{O}(n)$

- We have to prove that it adds the half-planes defining the edges of C in the right order.
- Consider an edge of C, and let p be its upper endpoint.Two case occurred?

Theorem (4.3)

The intersection of two convex polygonal regions in the plane can be computed in $\mathcal{O}(n)$ time.

corollary 4.4

The common intersection of a set of n half-planes in the plane can be computed in $\mathcal{O}(nlogn)$ time and linear storage

Finally

By Looping through all n facets we can solve the castability Problem in $\mathcal{O}(n^2logn)$ time.

But. . .

But is this the Best Time?

corollary 4.4

The common intersection of a set of n half-planes in the plane can be computed in $\mathcal{O}(nlogn)$ time and linear storage

Finally

By Looping through all n facets we can solve the castability Problem in $\mathcal{O}(n^2logn)$ time.

But. . .

But is this the Best Time?

corollary 4.4

The common intersection of a set of n half-planes in the plane can be computed in $\mathcal{O}(nlogn)$ time and linear storage

Finally

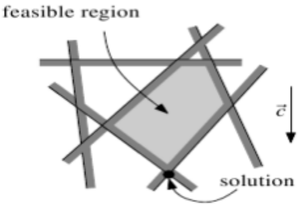
By Looping through all n facets we can solve the castability Problem in $\mathcal{O}(n^2logn)$ time.

But. . .

But is this the Best Time?

- The intersection of half-plane gives us all valid direction, but we just need one!
- Finding Just One Direction can be Done using Linear Programming.

$$
\begin{array}{ll}
\text{Maximize} & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\
\text{Subject to} & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leqslant b_1 \\
 & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leqslant b_2 \\
 & \vdots \\
 & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leqslant b_n
\end{array}
$$



Linear Programming Status

- H is a set of n two dimensional constraints.
- $f_{\vec{c}}(p) = C_x P_x + C_y P_y$ gives objective function
- **GOAL** find $p \in R^2$ so that $p \in \cap H$ and $f_{\vec{c}}(p)$ is maximized.
- Let C denote feasible region for $(H, \vec{c})$.

Types of solutions

i The linear program is infeasible.

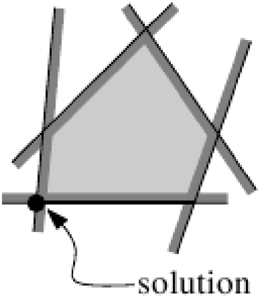
ii The feasible region is unbounded in direction $\vec{c}$.

iii The feasible region has an edge e whose outward normal points in the direction $\vec{c}$.

iv If none of the preceding three cases applies, then there is a unique solution, which is the vertex v of C that is extreme in the direction $\vec{c}$.

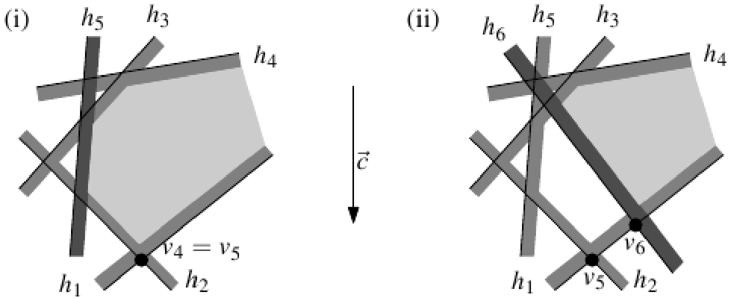
Adding Half-Planes

We Compute the Solution using an incremental algorithm in which the constraint are considered one at a time;



Lemma (4.5)

When considering another constraint (half-plane) the optimal solution point can only be effected in two ways:

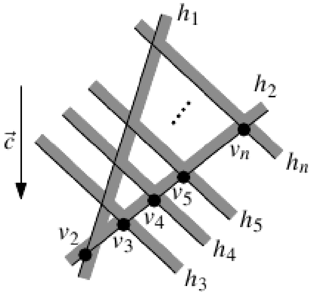


Adding Half-Planes

Lemma (4.6)

The change in case(ii) can be computed in $\mathcal{O}(i)$ when we consider constraint (half-plane) h_i

Figure: Worst Case



2DBOUNDEDLP

We can now describe the linear programming algorithm in more detail. As above, we use l_i to denote the line that bounds the half-plane h_i .

```
Algorithm 2DBOUNDEDLP(H, c⃗, m1, m2)
Input. A linear program (H ∪ {m1, m2}, c⃗), where H is a set of n half-planes,
   c⃗ ∈ ℝ², and m1, m2 bound the solution.
Output. If (H ∪ {m1, m2}, c⃗) is infeasible, then this fact is reported. Otherwise,
   the lexicographically smallest point p that maximizes f_c⃗(p) is reported.
    Let v0 be the corner of C0.
    Let h1, ..., hn be the half-planes of H.
    for i ← 1 to n
        do if v_{i−1} ∈ h_i
              then v_i ← v_{i−1}
              else v_i ← the point p on ℓ_i that maximizes f_c⃗(p), subject to the
                   constraints in H_{i−1}.
                   if p does not exist
                     then Report that the linear program is infeasible and quit.
    return v_n
```

Adding Half-Planes

Lemma (4.7)

A solution can be computed in $\mathcal{O}(n^2)$ time.

$$\sum_{i=0}^{n} \mathcal{O}(i)$$

Randomized Linear Programming

- There is a surprisingly simple way to reduce the time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

  start by the permuting the input randomly!

- This gets rid of the worst case!

Randomized Linear Programming

- There is a surprisingly simple way to reduce the time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

  start by the permuting the input randomly!

- This gets rid of the worst case!

Algorithm 2DRANDOMIZEDBOUNDEDLP$(H, \vec{c}, m_1, m_2)$

Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.

Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

1. Let v_0 be the corner of C_0.
2. Compute a *random* permutation $h_1, \ldots, h_n$ of the half-planes by calling RANDOMPERMUTATION$(H[1 \cdots n])$.
3. **for** $i \leftarrow 1$ **to** n
4.   **do if** $v_{i-1} \in h_i$
5.   **then** $v_i \leftarrow v_{i-1}$
6.   **else** $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
7.   **if** p does not exist
8.   **then** Report that the linear program is infeasible and quit.
9. **return** v_n

Yazd Univ.

Computational Geometry

Casting

Half-Plane Intersection

Incremental Linear Programming

Randomized Linear Programming

Unbounded Linear Programs

Smallest Enclosing Discs

RANDOMPERMUTATION Algorithm

Algorithm RANDOMPERMUTATION(A)
Input. An array $A[1\cdots n]$.
Output. The array $A[1\cdots n]$ with the same elements, but rearranged into a random permutation.

1. **for** $k \leftarrow n$ **downto** 2
2. **do** *rndindex* $\leftarrow$RANDOM(k)
3. Exchange $A[k]$ and $A[rndindex]$.

Lemma (4.8)

The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Why

- RANDOMPERMUTATION is run in $\mathcal{O}(n)$ time.
- Define Function x_i for adding new plane to H.
- So, total Time is $\sum \mathcal{O}(i) X_i = \mathcal{O}(n)$

Lemma (4.8)

The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Why

- RANDOMPERMUTATION is run in $\mathcal{O}(n)$ time.
- Define Function x_i for adding new plane to H.
- So, total Time is $\sum \mathcal{O}(i) X_i = \mathcal{O}(n)$

Unbounded Linear Programs

In the preceding sections we avoided handling the case of an unbounded linear program by adding two additional, artificial constraints.
This is not always a suitable solution!

Lets first

how we can recognize whether a given linear program (H, c) is unbounded.

If we denote the rays starting point as p, and its direction vector as d, we can parameterize ρ as follows:

$$\rho = \{p + \lambda \vec{d} : \lambda > 0\}$$

.

Lemma (4.9)

A linear program $(H, \vec{c})$ *is unbounded if and only if there is a vector* $\vec{d}$ *with* $\vec{d}.\vec{c} > 0$ *such that* $\vec{d}.\vec{\eta}(h) \geq 0$ *for all* $h \in H$ *and the linear program* $(H', \vec{c})$ *is feasible, where* $H' = h \in H : \vec{\eta}(h).\vec{d} = 0$*.*

2DRANDOMIZEDLP Algorithm

Algorithm 2DRANDOMIZEDLP($H, \vec{c}$)

Input. A linear program $(H, \vec{c})$, where H is a set of n half-planes and $\vec{c} \in \mathbb{R}^2$.

Output. If $(H, \vec{c})$ is unbounded, a ray is reported. If it is infeasible, then two or three certificate half-planes are reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

1. Determine whether there is a direction vector $\vec{d}$ such that $\vec{d} \cdot \vec{c} > 0$ and $\vec{d} \cdot \vec{\eta}(h) \geqslant 0$ for all $h \in H$.
2. **if** $\vec{d}$ exists
3. **then** compute H' and determine whether H' is feasible.
4. **if** H' is feasible
5. **then** Report a ray proving that $(H, \vec{c})$ is unbounded and quit.
6. **else** Report that $(H, \vec{c})$ is infeasible and quit.
7. Let $h_1, h_2 \in H$ be certificates proving that $(H, \vec{c})$ is bounded and has a unique lexicographically smallest solution.
8. Let v_2 be the intersection of ℓ_1 and ℓ_2.
9. Let $h_3, h_4, \ldots, h_n$ be a random permutation of the remaining half-planes in H.

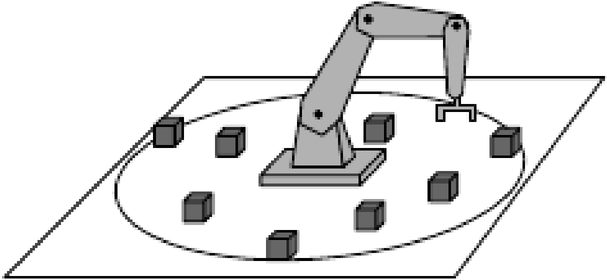
2DRANDOMIZEDLP Algorithm

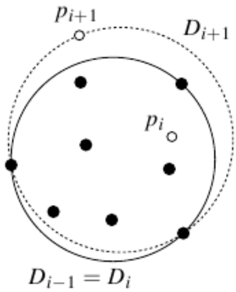
for $i \leftarrow 3$ **to** n
    do if $v_{i-1} \in h_i$
        then $v_i \leftarrow v_{i-1}$
        else $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
            if p does not exist
                then Let h_j, h_k (with $j,k < i$) be the certificates (possibly $h_j = h_k$) with $h_j \cap h_k \cap \ell_i = \emptyset$.
                    Report that the linear program is infeasible, with
return v_n

Unbounded Linear Programs

Theorem (4.10)

A 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.




p_{i+1}
D_{i+1}
p_i
$D_{i-1} = D_i$

Algorithm MINIDISC(P)

Input. A set P of n points in the plane.

Output. The smallest enclosing disc for P.

Compute a random permutation $p_1, \ldots, p_n$ of P.
Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
for $i \leftarrow 3$ **to** n
  do if $p_i \in D_{i-1}$
    then $D_i \leftarrow D_{i-1}$
    else $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \ldots, p_{i-1}\}, p_i$)
return D_n

MINIDISCWITHPOINT(P, q)

Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.

Output. The smallest enclosing disc for P with q on its boundary.

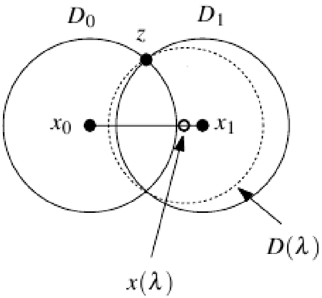
```
Compute a random permutation p_1, ..., p_n of P.
Let D_1 be the smallest disc with q and p_1 on its boundary.
for j ← 2 to n
    do if p_j ∈ D_{j-1}
        then D_j ← D_{j-1}
        else D_j ← MINIDISCWITH2POINTS({p_1, ..., p_{j-1}}, p_j, q)
return D_n
```

MINIDISCWITH2POINTS(P, q_1, q_2)

Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.

Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

```
Let D_0 be the smallest disc with q_1 and q_2 on its boundary.
for k ← 1 to n
    do if p_k ∈ D_{k-1}
          then D_k ← D_{k-1}
          else D_k ← the disc with q_1, q_2, and p_k on its boundary
return D_n
```


D_0
D_1
z
x_0
x_1
$D(\lambda)$
$x(\lambda)$