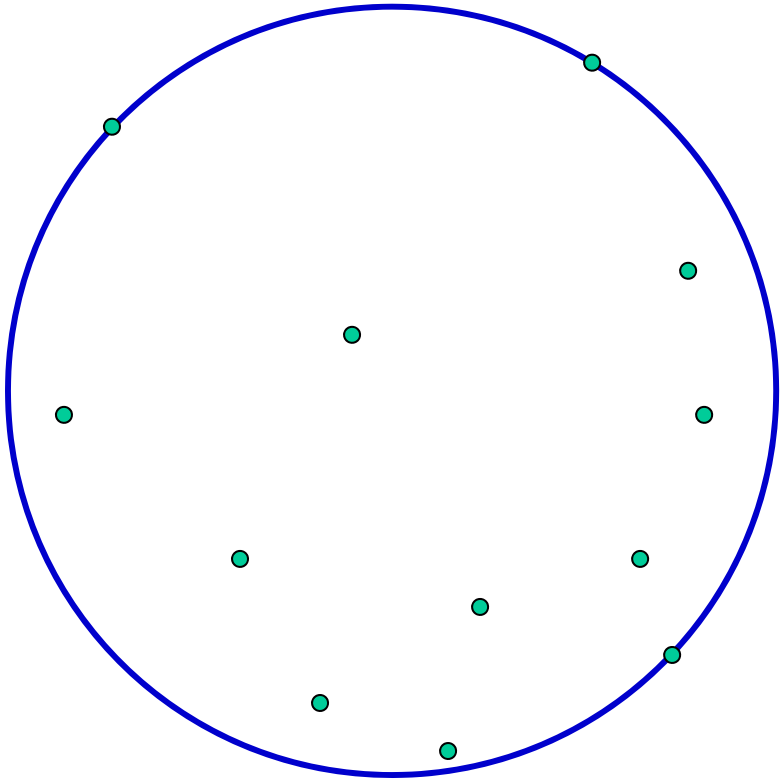
テーマ7：最小包含円

点集合を包含する半径最小の円

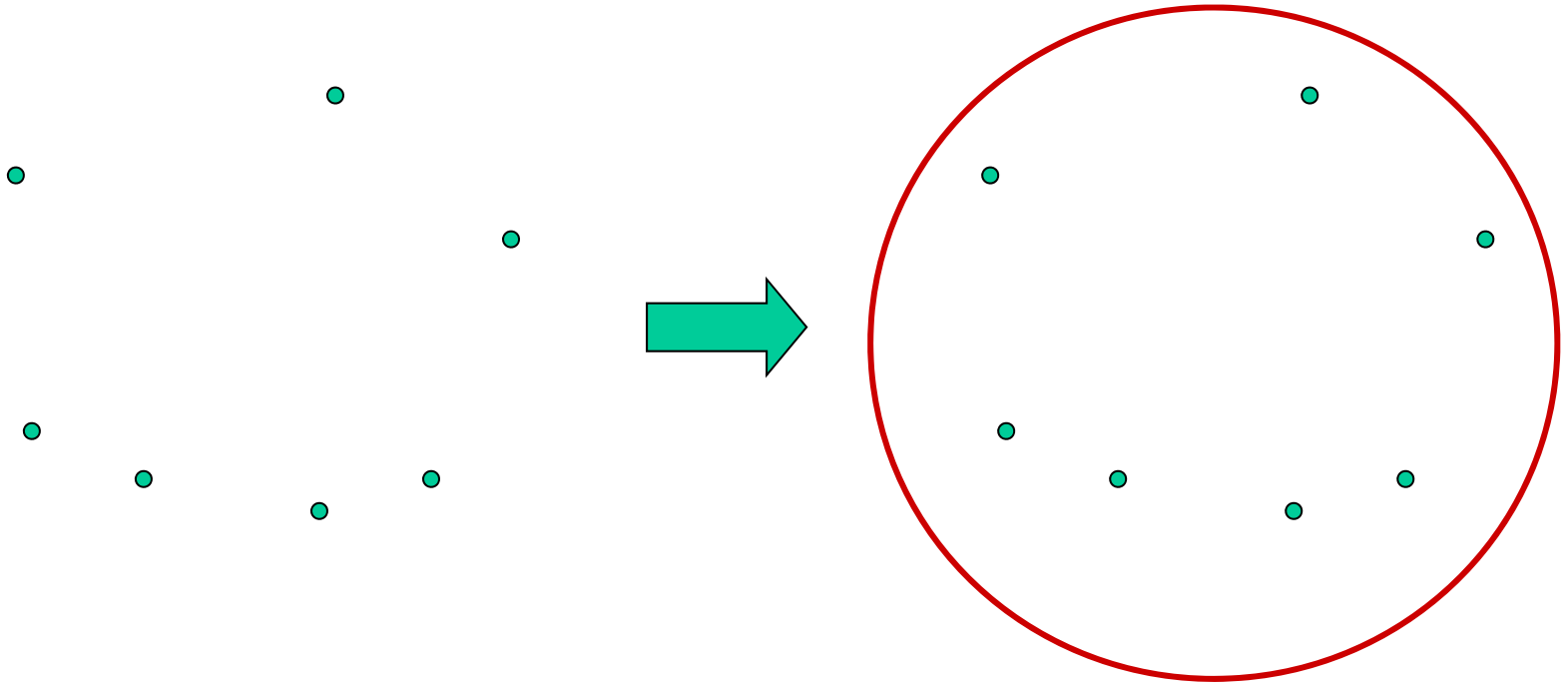
最小包含円問題

問題: 平面上にn点の集合が与えられたとき，これらの点をすべて内部に含む半径最小の円 を効率よく求める方法を示せ.



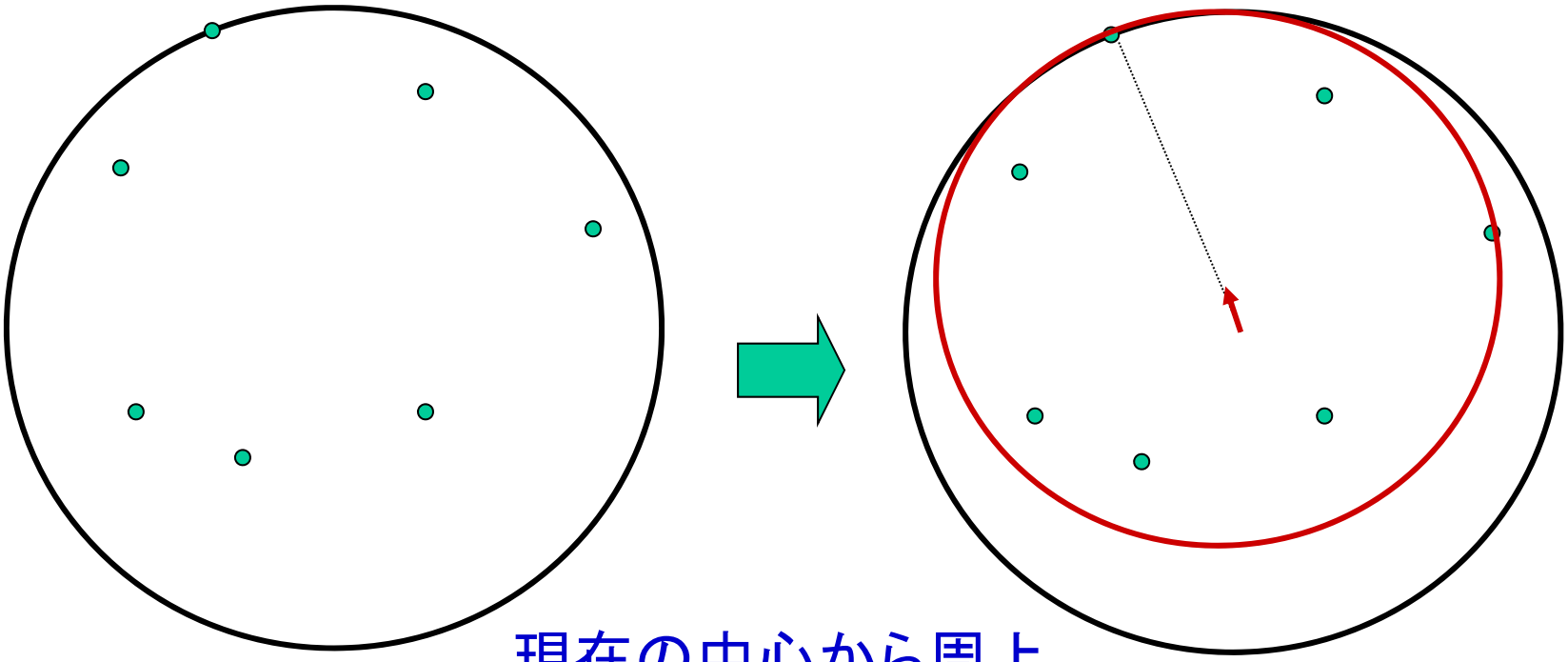
どの点にも接触しない包含円

すべての点を内部に含む包含円を求める



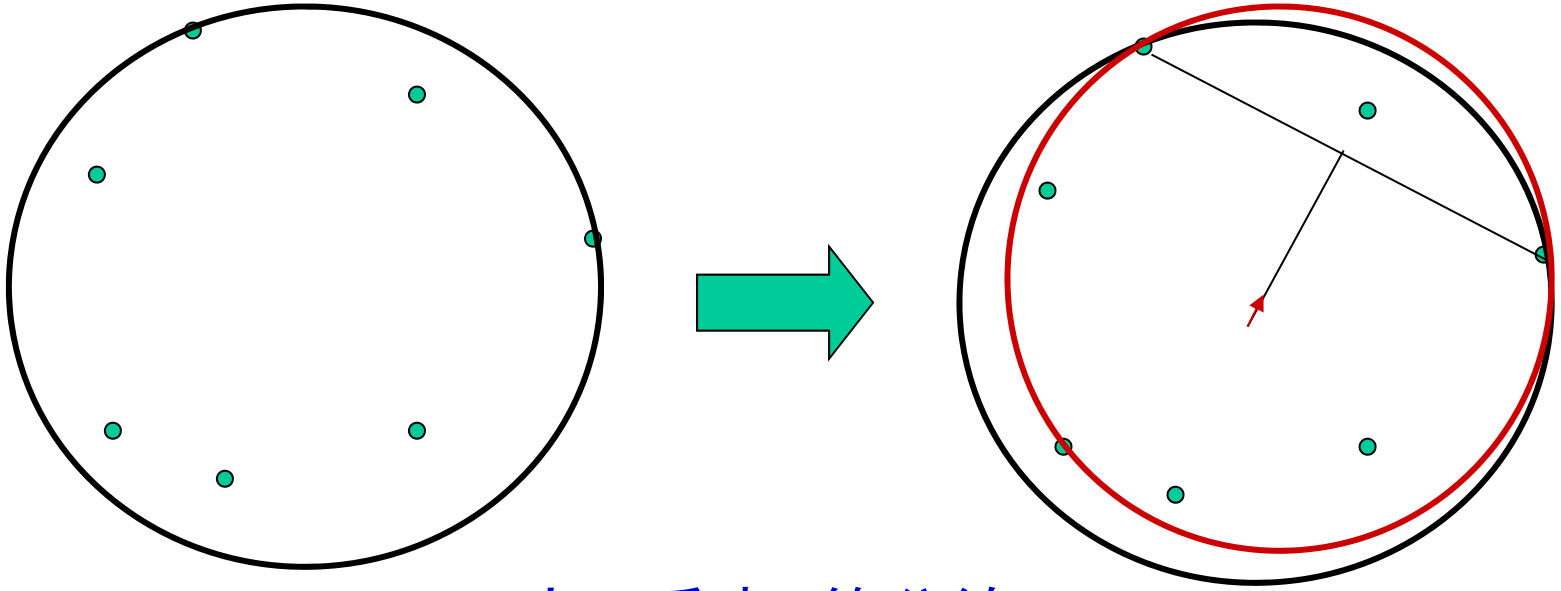
十分に大きな包含円から
始め, 点にぶつかるまで
徐々に半径を小さくする

１点にしか接触しない包含円



現在の中心から周上の点に向けて中心を移動する

２点を周上に含む包含円の場合

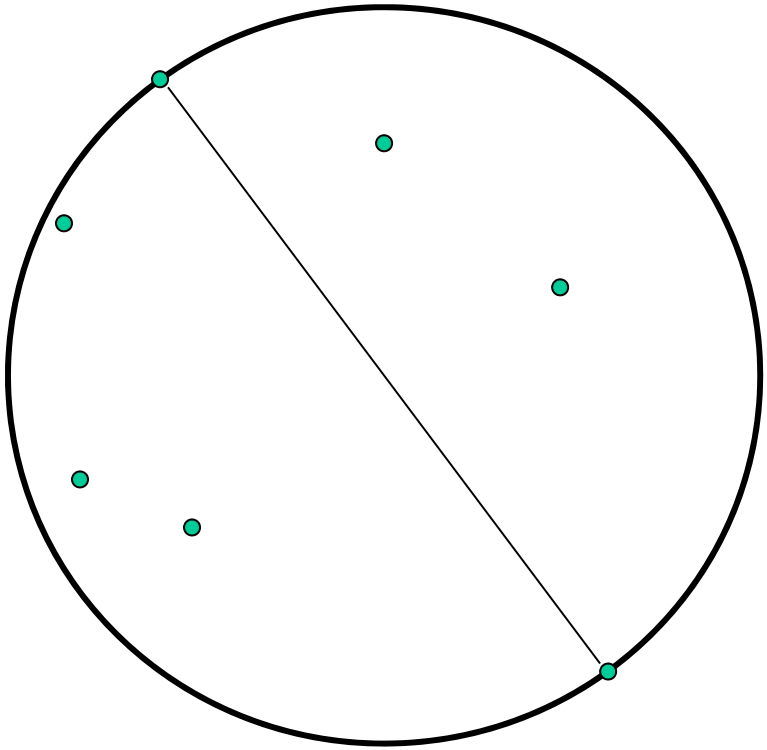


２点の垂直２等分線
上で中心を移動

これは最小包含円か？

特別な場合

周上の２点を直径の両端点とする円が
包含円である場合



この場合は，この包含円が最小包含円
である

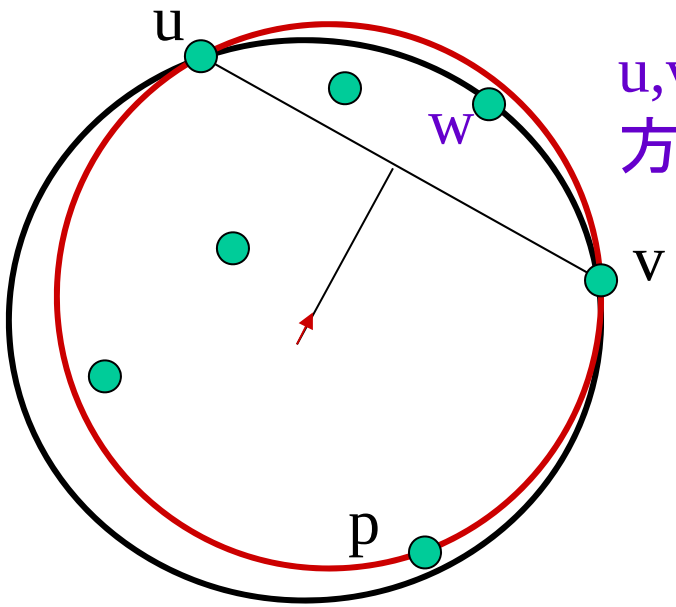
最小包含円を求める素朴な方法

Step0: 点集合Pを与える
Step1 : 与えられた点をすべて含む十分大きな包含円D1を求める.
Step2: 円D1の中心は変えずに, Pの点 u が周上に来るまで,
半径だけを徐々に小さくする. 得られた円をD2とする.
Step3: Pの別の点 v が周上に来るまで, 円D2の中心を点 u に
向けて徐々に移動する. ただし, このとき点 u が周上に
保つこと. 得られた円をD3とする.
Step4: 2点u, v を直径の両端点とする円が他のすべての点を含
むなら, この円が答. そうでない場合は, Pの別の点wが
円周上に来るまで, 円の中心を2点u, vの垂直2等分線
上で直線uvに向けて徐々に移動.

この方法で最小包含円が求まっているか？
具体的にプログラムとして実現せよ.
計算時間を解析せよ.

この方法で最小包含円が求まっているか？

(答) 求まらないことがある.



u,v,wを通る円よりp,v,wで決まる円の方が小さい

問題は，このような状況が何回生じるか？

Combinatorial question

→（組み合わせ問題）

包含円は最大何通りあるか？

円は3点で決まるから，包含円は$O(n^3)$通り

本当にそれだけあるか？

もし，本当に$O(n^3)$通りの包含円があるなら，
3重ループのアルゴリズムで最小包含円を求めることができる.

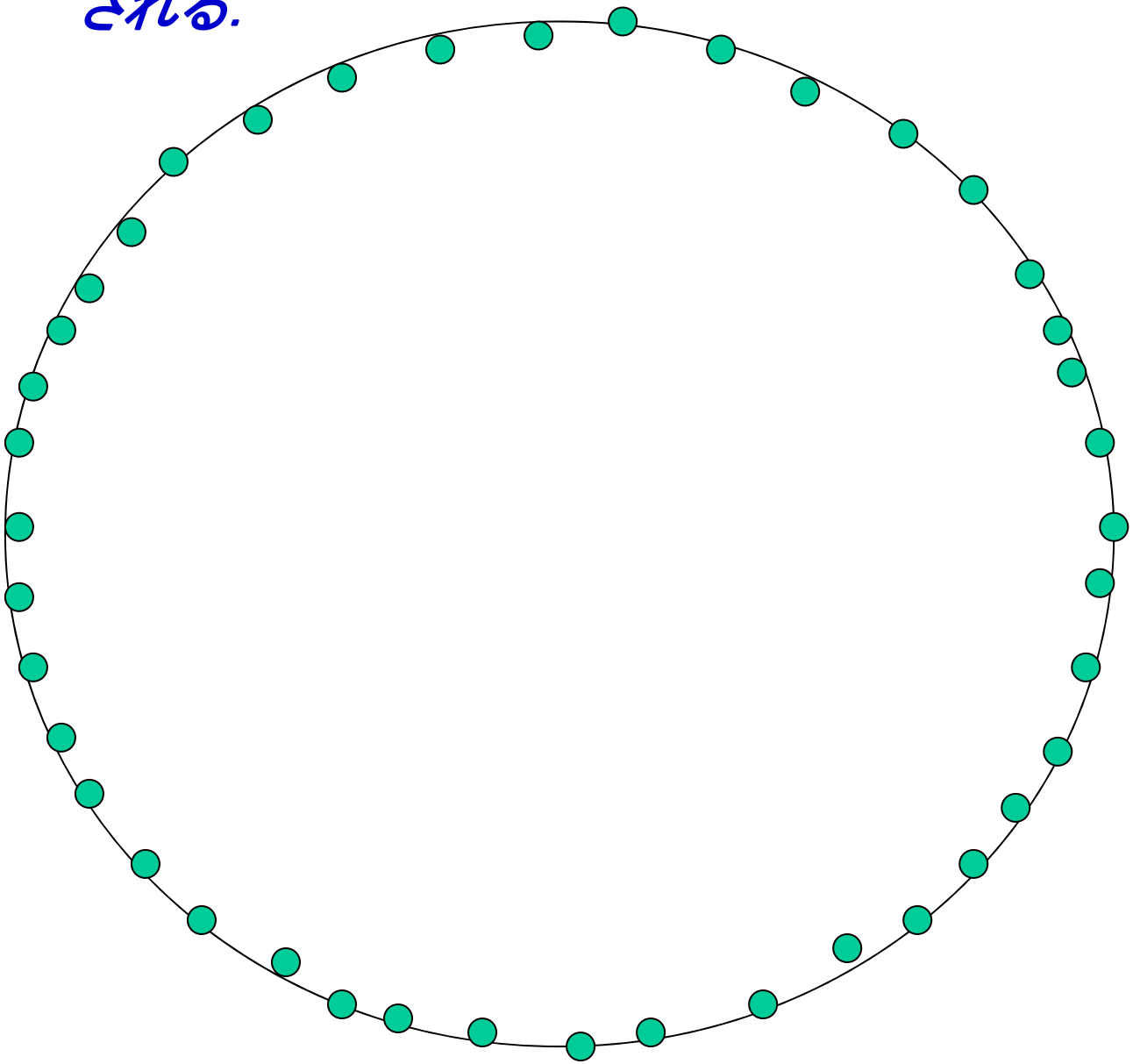
素朴なアルゴリズム

Algorithm Exhaustive-search

P = {**p**1, **p**2, ... , **p**n}
D* = 半径無限大の円
for i=1 to n-2 do
  for j=i+1 to n-1 do
    for k=j+1 to n do
      pi, **p**j, **p**kを通る円を求め，**D**とする
      if radius(**D**) < radius(**D***) then
        check = false
        for h=1 to n s.t. h != i,j,k do
          if 点**p**hは **p**i, **p**j, **p**kを通る円の外にある
            then check = true;
        if check = false
          then **D*** = **D**（最適解の更新）
return **D***

例題：

どの3点をとっても，包含円に近い円が定義される.

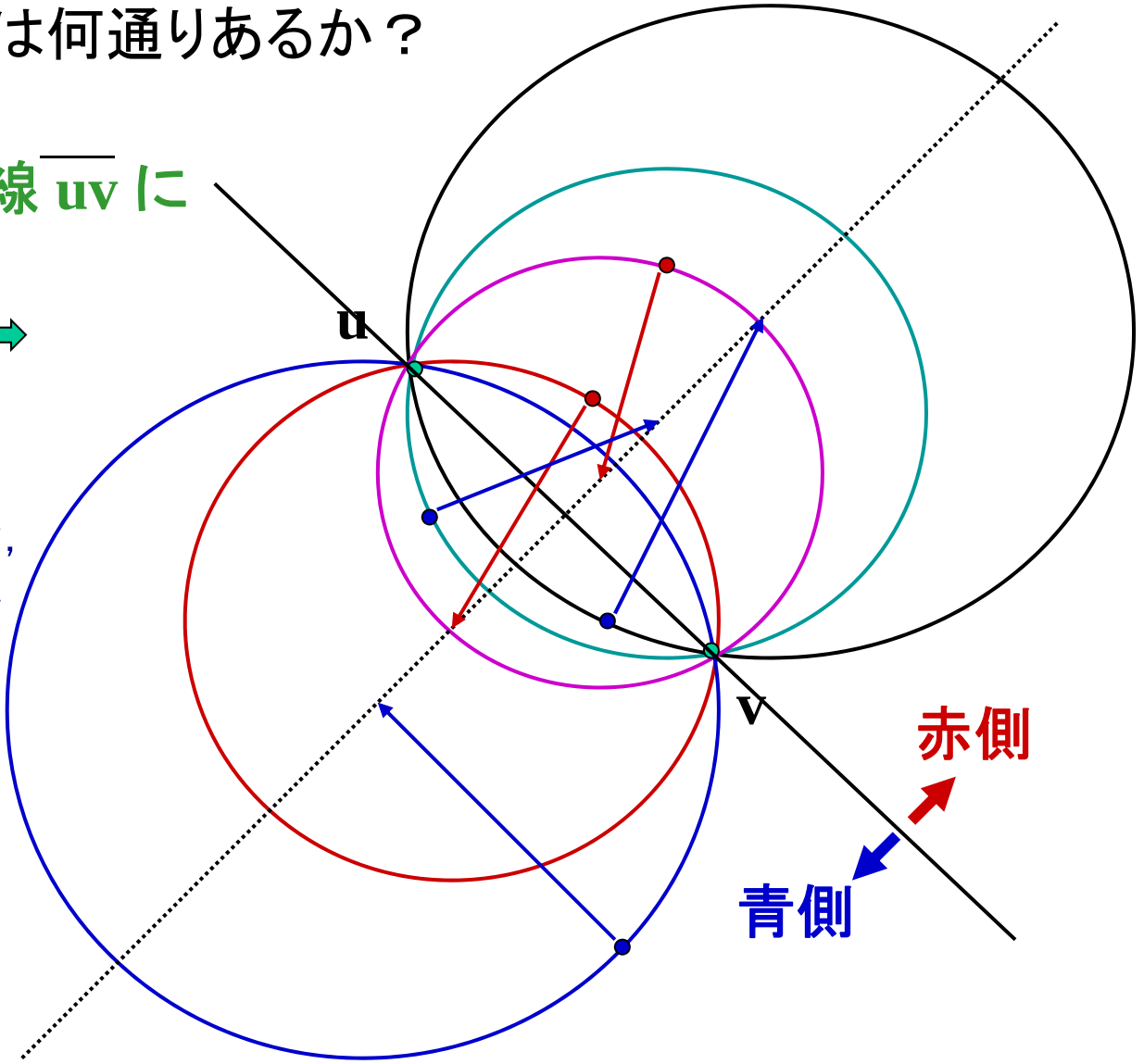


包含円は本当にO(n^3)通りあるか？

２点u, vを固定したとき, この２点を通る円の中で
包含円となりうるものは何通りあるか？

u, v以外の点を直線 $\overline{uv}$ に関して2分割
赤側と青側 ➡

各点について，その点と
u, vを通る円を求めたとき，
青側の点は，青側に中心
をもつ半径最大の円の
内部にある



u, v以外の各点**pi** に対し,
u, v, pi を通る円の中心**Ci**を求める.

B={**pi** | **pi**も**Ci**も青側にある},

R = {**pj** | **pj**も**Cj**も赤側にある}

(1) **B**が空でないとき,

Bの要素**pi**は**R**の点に対応する円には含まれない

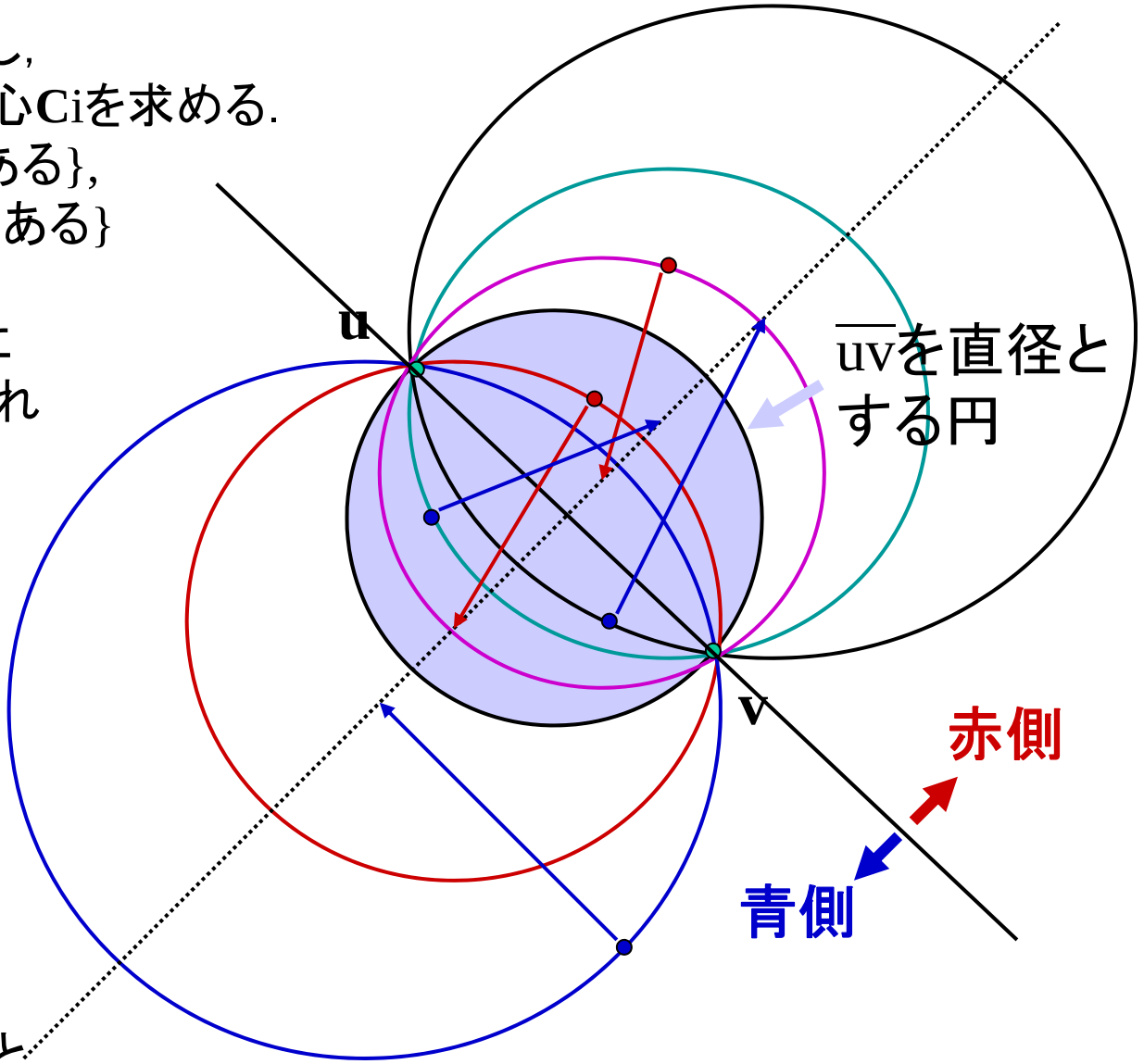
Bの要素の中で半径最大の円に対応する点は最も外.

よって, 半径最大の円だけが包含円の候補

(2)**R**が空でない場合についても同じ

(3)**B**も**R**も空のとき,

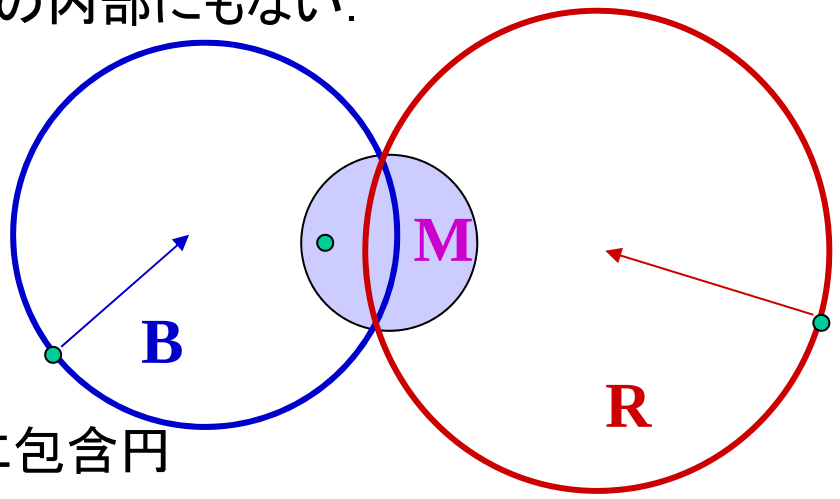
u, v を直径の両端点とする円がすべての点を内に含む



観察：２点u, vを固定したとき，
u, vと別の１点を通る円の中で包含円となりうるものは高々１個しかない．

証明：

B={**p**i | **p**iも**C**iも青側にある}, **R** = {**p**j | **p**jも**C**jも赤側にある}
uvを直径とする円の内部にある点の集合を**M**とする．
Mの点は，**B**の任意の点**p**iに対応する円の内部に含まれる
Bの点は，**M**または**R**のどの点に対応する円の内部にもない．
　　（**R**の点についても同じ）
(1)**B, R**が共に空でないとき，
　　u, vを通る包含円は存在しない
(2)**B**は空でないが，**R**は空のとき，
　　Bの点に対応する半径最大の円だけ
　　が包含円の候補．
　　この円が**M**の点もすべて含めば，確かに包含円
(3)**R**は空でないが，**B**は空のときも同じ
(4)**B, R**が共に空のとき，
　　uvを直径とする円だけが，u, vを通る包含円
したがって，u, vを通る包含円は高々１個だけ．



結論： 各点対に対して高々1つの包含円しか定義されないから，全部で高々$O(n^2)$個の包含円しか存在しない．

この観察より，$O(n^3)$時間の方法が得られる

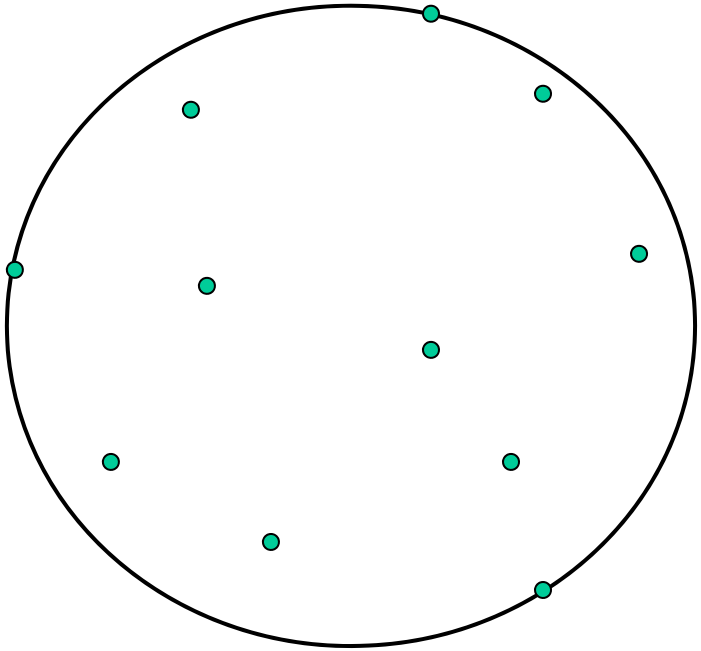
Algorithm Naive-2

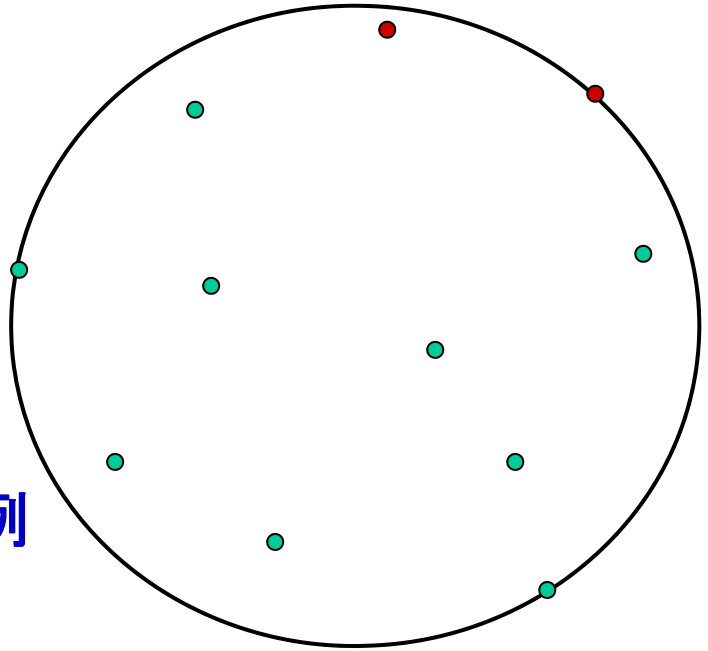
P = {**p**1, **p**2, ... , **p**n}
D* = 半径無限大の円
for i=1 to n-1 do
  for j=i+1 to n do
    pi**p**jを直径とする円**C**ijが包含円なら，その円を**D***として終了．
    D = 半径0の円
    for k=1 to n do s.t. **p**kは**C**ijの外部
      pi, **p**j, **p**kを通る円を求め，**D'**とする
      if radius(**D**) < radius(**D'**) then **D**=**D'**
    if 円**D**が包含円 AND
      radius(**D**) < radius(**D***) then **D*** = **D**
return **D***

では，$O(n^2)$個の包含円が存在するような例はあるか？

この問に答えるために，別の観察が必要

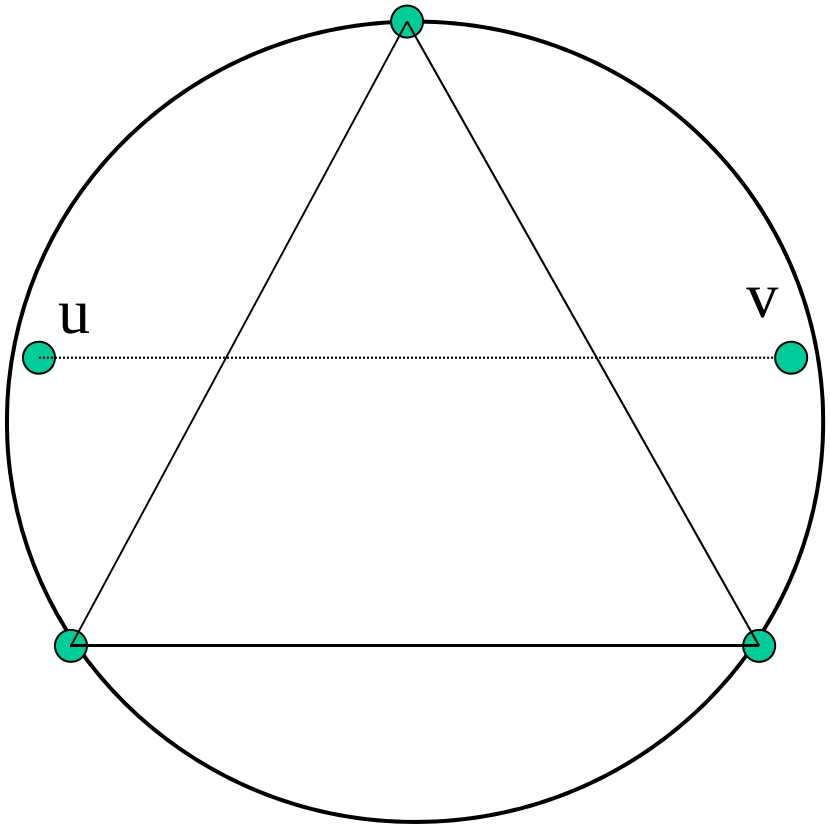
Q1: y座標最大の点を最小包含円は通るか？



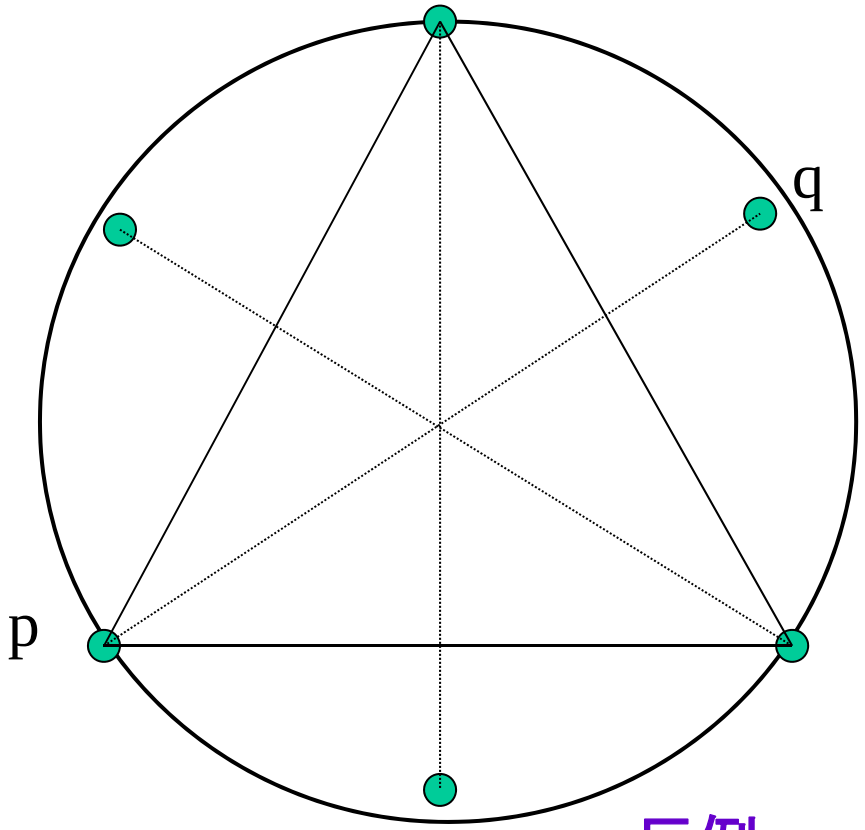


反例

Q2: 最小包含円は最も遠く離れた2点を通るか？



Q3: 最小包含円が点pを通るとき，pから最も遠く離れた点qも通るか？



反例

Q1: y座標最大の点を最小包含円は通るか？

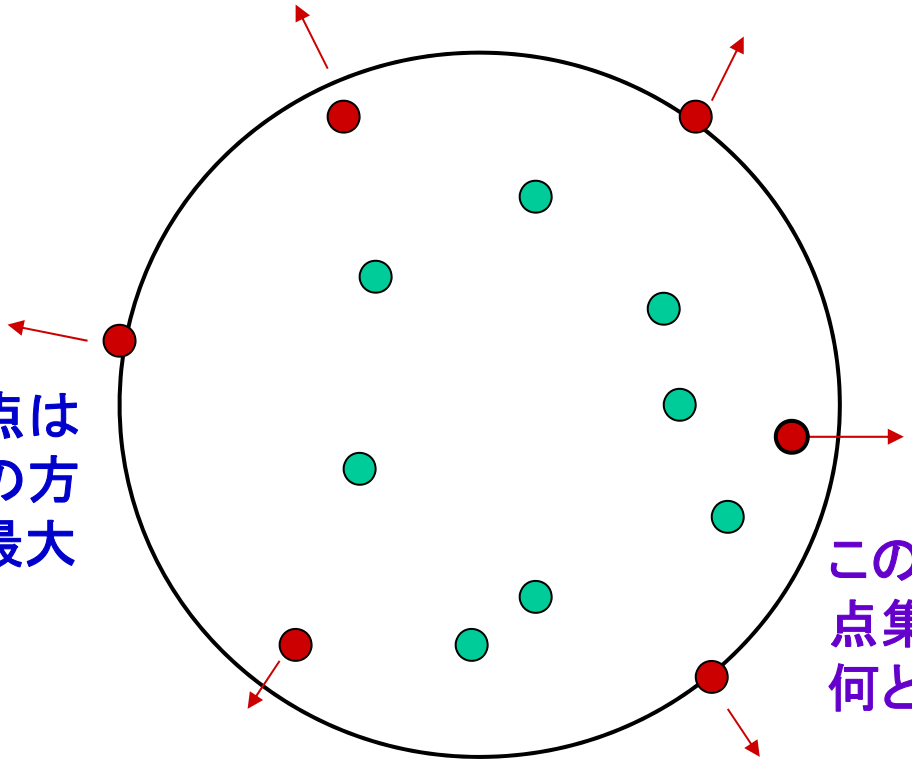
Q2: 最小包含円は最も遠く離れた2点を通るか？

Q3: 最小包含円が点pを通るとき，pから最も
遠く離れた点qも通るか？

何が悪いのか?

Q1: y座標最大の点を最小包含円は通るか？

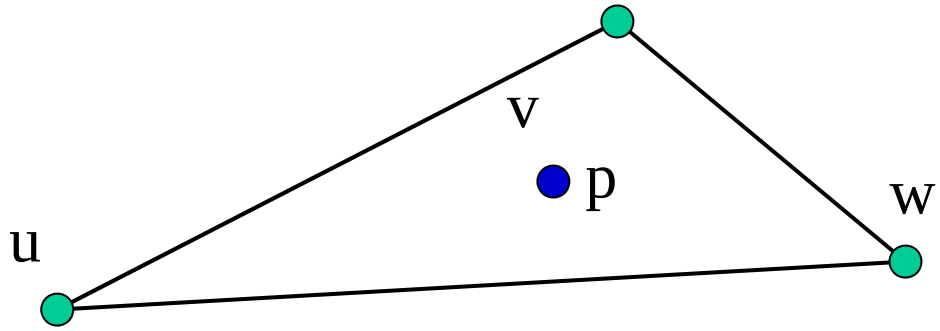
どの方向でも最大（最小）でない点は最小包含円に寄与しない.



赤の点は
矢印の方
向に最大

このような
点集合を
何と言うか？

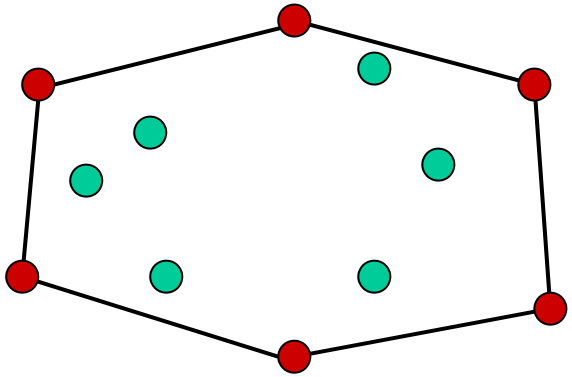
３点で決まる三角形の内部にある点は最小包含円に寄与しない.



点pを通る円は，３点u, v, wをすべて同時に内部に含むことはない.
よって，点pを通る包含円は存在しない.

点**p**は，点集合**P**の**極大点**である
⟺ 点集合**P**からどのように3点を選んでも，その3点で決まる
三角形の真の内部に点**p**が含まれることはない.

極大点だけをつなぐと，凸包ができる.



凸包はO(n log n)時間で構成可能

y座標最大の点は必ず極大点である.

証明:

y座標最大の点をuとする. uが極大点でないとすると, 三角形pqrの内部にuを含むように3点p, q, rをとることができる.
点u は三角形pqrの内部にあるから, p,q,rのどれかは必ずuよりy座標が大きくなければならないが, これはuがy座標最大の点であるという仮定に反する.

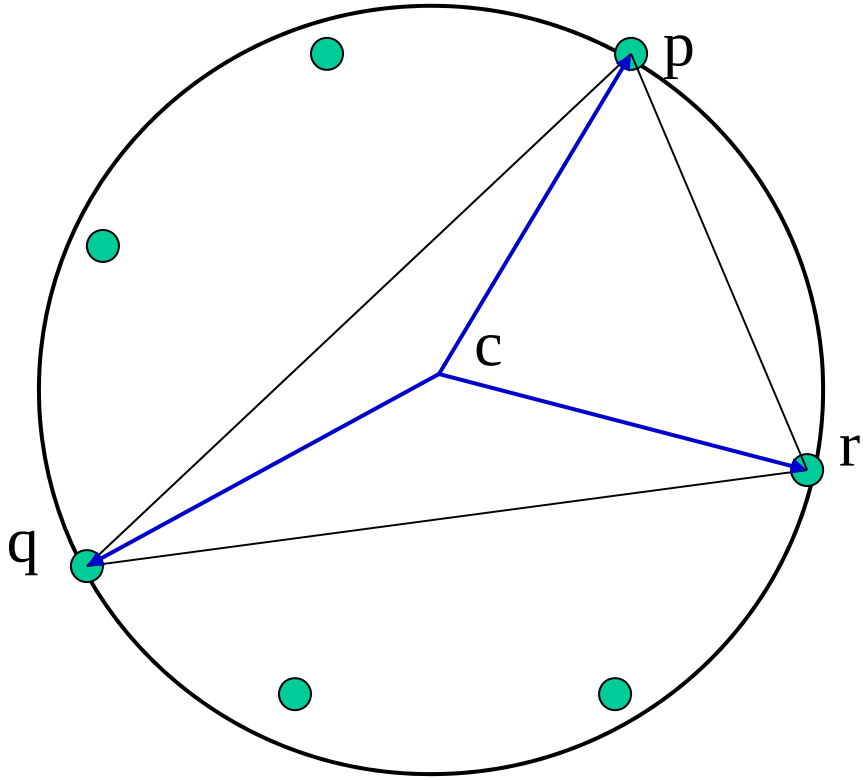
点を偏角順にソートして, 各角度で極大な点を順に求めれば凸包が構成できる.

偏角順のソート・・・・・O(n log n)時間
極大でない点の除去・・・O(n)時間

凸包上にない点は包含円に寄与しないから, 取り除いてよい.

では, 凸包上のどのような点対に対して包含円が構成されるか？

凸包上のどのような点対に対して包含円が構成されるか？



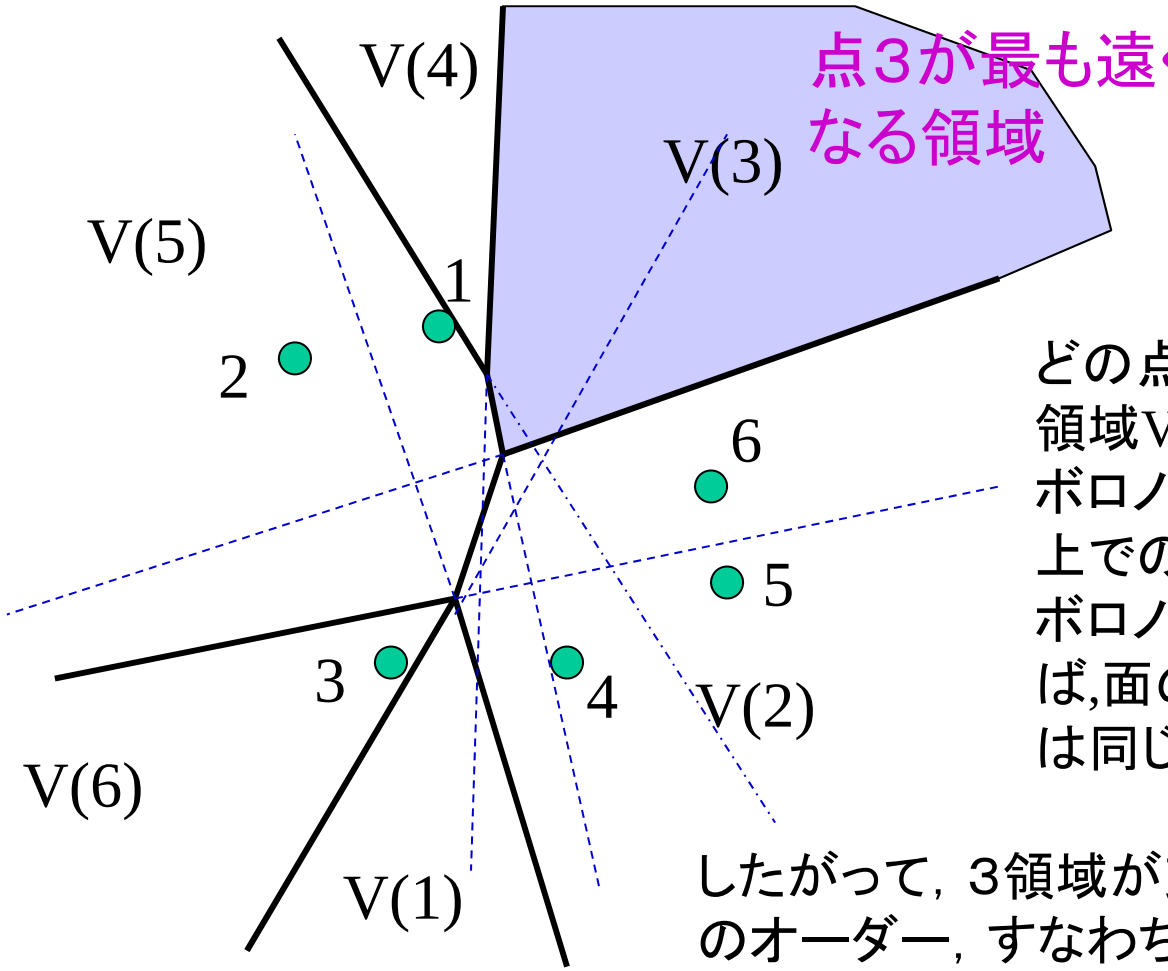
３点 **p, q, r**で決まる円の中心を**c** とするとき，
p, q, rは**c** から**最も遠い点**である．

このような3点から等距離にあって，しかもそれら3点がその点からの最遠点になっているような点は何通りあるか？

これが包含円の個数と同じ．

最遠点ボロノイ図

Pの点p→ V(p): 点pがPの中で最も遠くなる領域



どの点pについてもボロノイ領域V(p)は連結で開領域.
ボロノイ領域の並び方は凸包上での点の並び方と同じ.
ボロノイ図を平面グラフと見れば,面の数と頂点数のオーダーは同じ.

したがって，3領域が交わる頂点の個数も点数のオーダー，すなわちO(n)である.
つまり，**包含円の個数もO(n)である**.

最遠点ボロノイ図に基づく方法

結局，包含円の中心の候補として最遠点ボロノイ図の頂点だけに限定可能.

包含円かどうかを判定する必要はない.
半径（最遠点までの距離）が最小であるものを求めればよい.

ボロノイ図の頂点はO(n)個しかないから，計算時間は
　最遠点ボロノイ図を構成する時間 O(n log n)
　+
　各頂点について最遠点までの距離を求める時間 O(n)

全体の計算時間はO(n log n)時間

結局，包含円は高々O(n)個しか存在しない．

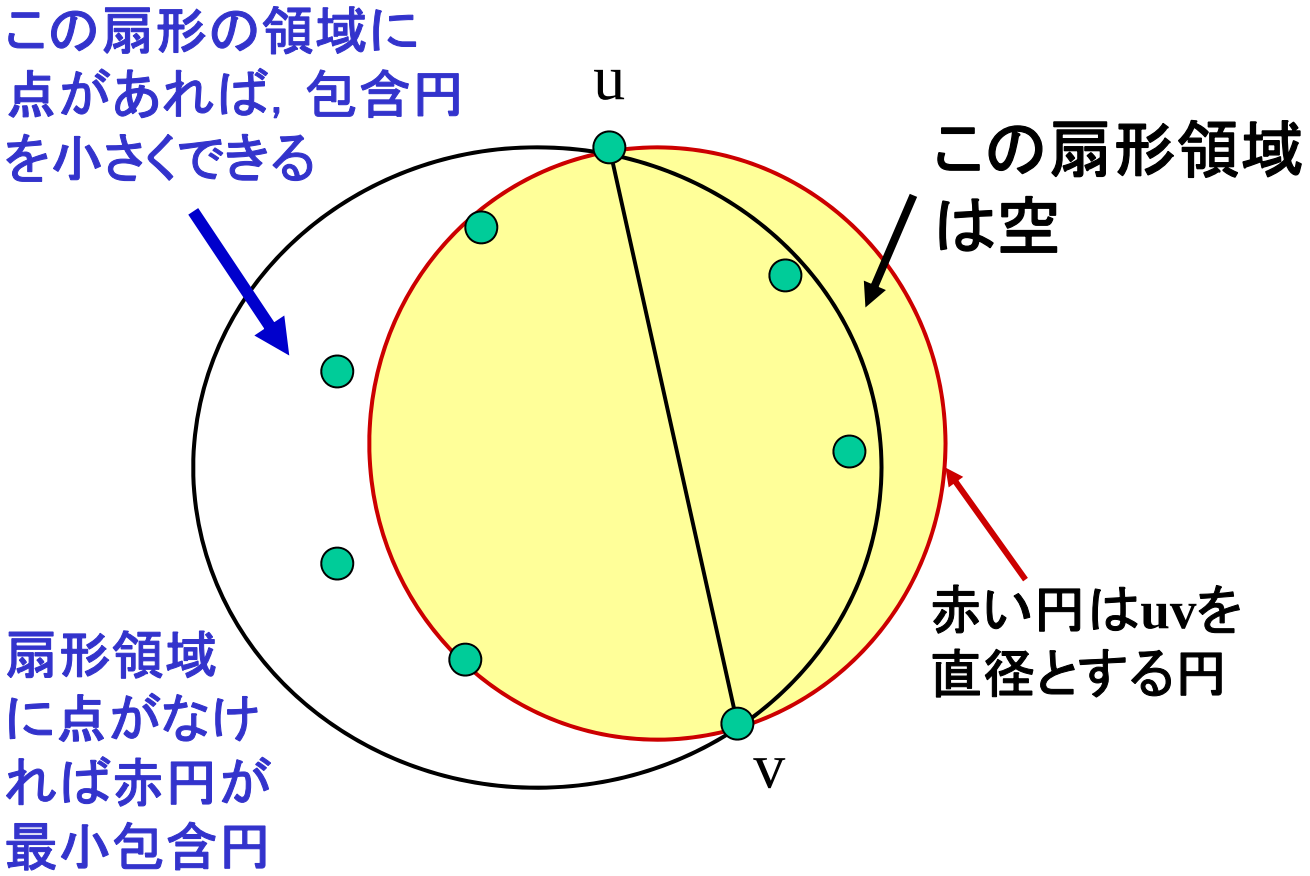
Algorithm Naive-2ではすべての点対に対して包含円を考えているが，包含円は全部でO(n)個しかないから，これは無駄．

最初に考えた素朴な方法が効果的

最初，十分大きな包含円から始めて，徐々に半径を小さくして，
３点を通る包含円を求める．
その後，包含円の半径をさらに改善する．
この改善はO(n)回で終り，毎回O(n)時間しかかからないから，
全体では$O(n^2)$時間．

最初に考えた素朴な方法が効果的

最初，十分大きな包含円から始めて，徐々に半径を小さくして，
３点を通る包含円を求める．
その後，包含円の半径をさらに改善する．この改善はO(n)回で
終り，毎回O(n)時間しかかからないから，全体ではO(n^2)時間．

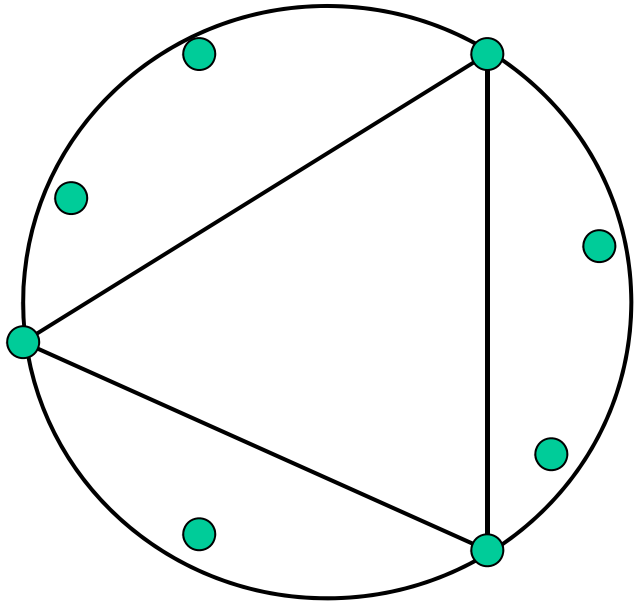


上記の方法では，毎回包含円の半径が小さくなり，かつ包含円が全体でもO(n)個しかないから，O(n)回の繰り返しの後，終了．

観察：包含円を決める3点が鋭角三角形を構成するなら，対応する包含円は最小である．

証明：

この三角形の外接円はこの３点に対する最小包含円である．よって，もしこれがすべての点を内に含む包含円なら，それ以上半径を小さくすることはできない．



別の観察:

P*i* = {**p***1*, **p***2*, ... , **p***i*}（最初の *i* 個の点）

D*i* : **P***i* に対する最小包含円

とすると，**p***i* が **D***i-1*に含まれているとき，**D***i* = **D***i-1*

そうでないとき，**p***i* はD*i* の境界上にある

アルゴリズム MiniDisc(P)

D*2* = {**p***1*, **p***2*}に対する最小包含円

for *i*=3 to n{

　if **p***i* が **D***i-1*に含まれる

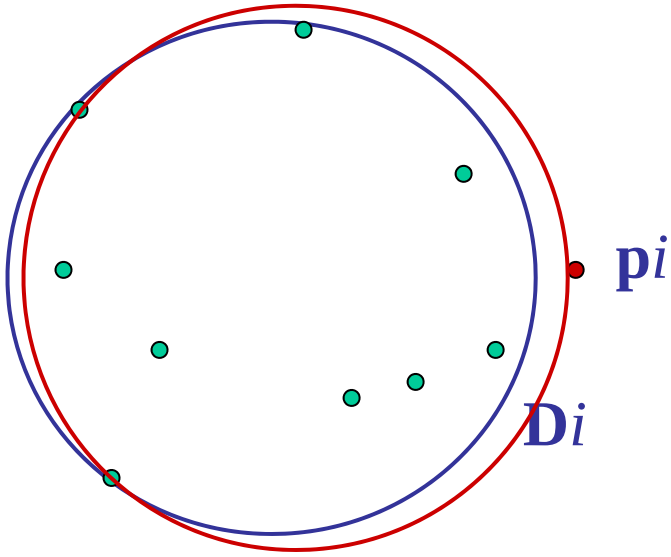
　　then **D***i* = **D***i*-1;

　　else　**D***i* = MiniDiscWithPoint(**P***i-1*, **p***i*);

}

return **D***n*;

点 **p***i* を円周上にもち，
点集合 **P***i-1*を包含する
半径最小の円



MiniDiscWithPoint(**P**, **q**)

D*1* = {**q**, **p***1*}に対する最小包含円

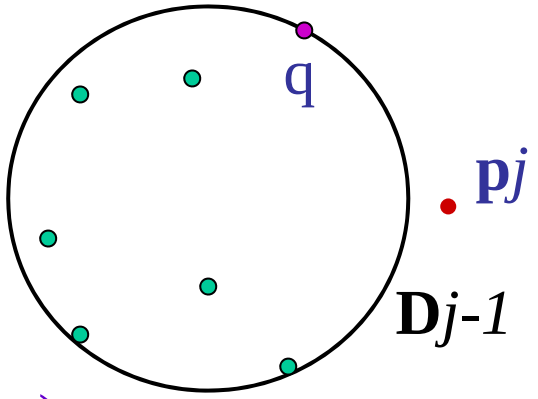
for *j*=*2* to |**P**|

　if **p***j* が **D***j-1* の内部に含まれる

　　then **D***j* = **D***j-1*

　　else **D***j* = MiniDiscWith2Points(**P***j-1*, **p***j*, **q**);

return D*n*;



MiniDiscWith2Points(**P**, **q**1, **q***2*)

D*0* = {q*1*, **q***2*}に対する最小包含円

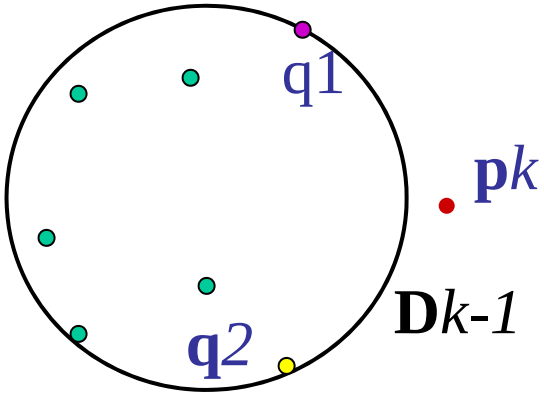
for *k*=*2* to |**P**|

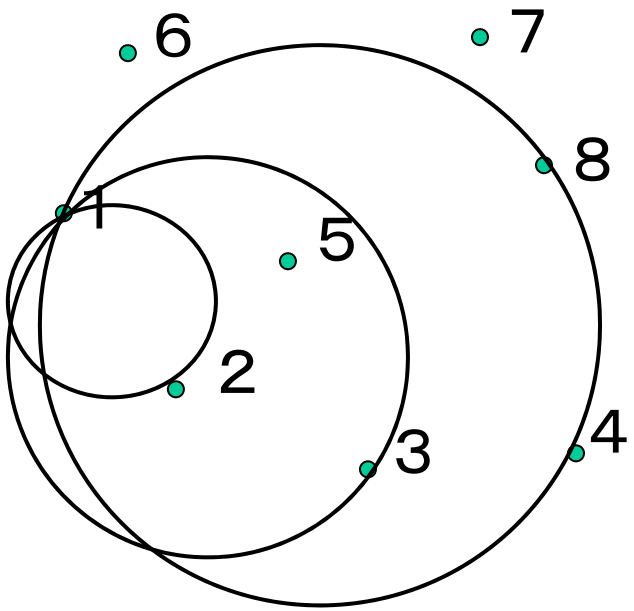
　if **p**k が **D***k-1* の内部に含まれる

　　then **D***k* = **D***k-1*;

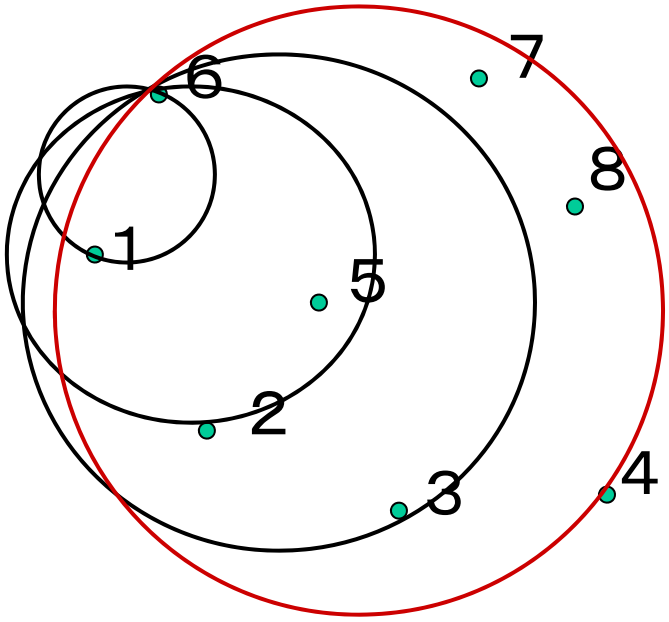
　　else **D***k* = disc(**q***1*, **q***2*, **p***k*);

return **D***n*;





D2=disc(1,2)
D3=MDP(3,{1,2})
  D1’=disc(3,1)
  D2’=disc(3,1) + 2
D3=disc(3,1)
D4=MDP(4,{1,2,3})
  D1’=disc(4,1)
  D2’=disc(4,1) + 2
  D3’=disc(4,1) + 2,3
D4=disc(4,1)
D5=disc(4,1) + 5
D6=MDP(6,{1,2,3,4,5})



D1’=disc(6,1)
  D2’=MDP2(6,2,{1})
    D0”=disc(6,2)
    D1”=disc(6,2) + 1
  D2’=disc(6,2)
  D3’=MDP2(6,3,{1,2})
    D0”=disc(6,3)
    D1”=disc(6,3)+1
    D2”=disc(6,3)+1,2
  D3’=disc(6,3)
  D4’=MDP2(6,4,{1,2,3})
    D0”=disc(6,4)
    D1”=disc(6,4) + 1
    D2”=disc(6,4) + 1,2
    D3”=disc(6,4) + 1,2,3
  D4’=disc(6,4)

最悪の場合の時間解析

アルゴリズム MiniDisc(P)　・・・・・時間 T(n)

D*2* = {**p***1*, **p***2*}に対する最小包含円
for *i*=3 to n{
　if **p***i* が **D***i-1*に含まれる then **D***i* = **D***i*-1;
　　else **D***i* = MiniDiscWithPoint(**P***i-1*, **p***i*);
} return **D***n*;

MiniDiscWithPoint(P, q)・・・・・・・・・・・・・・・時間 $T_1(n)$

D*1* = {**q**, **p***1*}に対する最小包含円
for *j*=*2* to |**P**|
　if **p***j* が **D***j-1* の内部に含まれる then **D***j* = **D***j-1*
　　else **D***j* = MiniDiscWith2Points(**P***j-1*, **p***j*, **q**);
return D*n*;

最悪の場合の時間解析

MiniDiscWith2Points(P, q1, q2)…… 時間 $T_2(n)$

D*0* = {q*1*, **q***2*}に対する最小包含円
for *k*=*2* to |**P**|
　if **p**k が **D***k-1* の内部に含まれる then　**D***k* = **D***k-1*;
　　else　**D***k* = **disc(q***1*, **q***2*, **p***k*);
return **D***n*;

$T(n) = 1 + \sum_{i=3}^{n} max(1, T_1(i))$　毎回更新されるのが最悪
$T_1(i) = 1 + \sum_{j=2}^{i} max(1, T_2(j))$　毎回更新されるのが最悪
$T_2(j) = 1 + \sum_{k=2}^{j} max(1,1) = j$

$T_2(j)=j$, $T_1(i) = 1+i(i+1)/2-1=i(i+1)/2$
$T(n)=O(n^3)$

最悪の場合の時間解析

MiniDisc(P)の計算時間：**$O(n^3)$**

・毎回D*i*が更新されるのが最悪
（MiniDiscWithPoint(P, q)を呼び出す）

本当に$O(n^3)$時間かかるような例題は作れるか？

ちょっと，鼻薬を嗅がせてみよう！

毎回，点につけた番号をランダムに入れ替えてみよう

アルゴリズム MiniDisc(P)

Pの点をランダムに並び替える

D*2* = {**p***1*, **p***2*}に対する最小包含円

for *i*=3 to n{

　if **p***i* が **D***i-1*に含まれる

　　then **D***i* = **D***i*-1;

　　else　**D***i* = MiniDiscWithPoint(**P***i-1*, **p***i*);

}

return **D***n*;

MiniDiscWithPoint(P, q)

Pの点をランダムに並び替える

D*1* = {**q**, **p***1*}に対する最小包含円

for *j=2* to **|P|**

　if **p***j* が **D***j-1* の内部に含まれる

　　then **D***j* = **D***j-1*

　　else　**D***j* = MiniDiscWith2Points(**P***j-1*, **p***j*, **q**);

return D*n*;

MiniDiscWith2Points(P, q1, q2)
Pの点をランダムに並び替える
D*0* = {q*1*, **q***2*}に対する最小包含円
for *k*=*2* to |**P**|
　if **p**k が **D***k-1* の内部に含まれる
　　then **D***k* = **D***k-1*;
　　else **D***k* = disc(**q***1*, **q***2*, **p***k*);
return **D***n*;

アルゴリズム MiniDisc(P)
Pの点をランダムに並び替える
D*2* = {**p***1*, **p***2*}に対する最小包含円
for *i*=3 to n{
　if **p***i* が **D***i-1*に含まれる
　　then **D***i* = **D***i*-1;
　　else **D***i* = MiniDiscWithPoint(**P***i-1*, **p***i*);
}
return **D***n*;

MiniDiscの実行時間の解析

一度もelse 部が実行されなければ，O(n)

では，MiniDiscWithPoint()が呼び出される確率は？

Backward analysisを用いる:

部分集合{p*1*, ... , p*i*}を固定してみよう.

{p*1*, ... , p*i*}を包含しqを通る円をDiとする.

{p*1*, ... , p*i*}からどの１点を除いたとき，円が変化するか？

円D*i*を決定する３点の内の１つを除いたときその確率は

2/*i*（q以外の２点の内の１つ）．よって，実行時間の期待値は，

$$O(n)+\sum_{i=2}^{n} O(i)\frac{2}{i}=O(n)$$

つまり，実行時間の期待値はO(n)である！

もう少し詳細な解析

最悪時の計算時間解析

$T(n) = 1 + \sum_{i=3}^{n} max(1, T_1(i))$

$T_1(i) = 1 + \sum_{j=2}^{i} max(1, T_2(j))$

$T_2(j) = 1 + \sum_{k=2}^{j} max(1,1) = j$

$T_2(j)=j$, $T_1(i) = 1+i(i+1)/2-1=i(i+1)/2$

$T(n)=O(n^3)$

確率を考慮した計算時間解析

$T(n) = 1 + \sum_{i=3}^{n} 1*(1-2/i) + T_1(i)*2/i$

$T_1(i) = 1 + \sum_{j=2}^{i} 1*(1-2/j) + T_2(j)*2/j$

$T_2(j) = j$

$T_1(i) = i - 2\ log\ i + 2i = 3i - 2\ log\ i$

$T(n) < n - 2\ log\ n + 3n - 2\ log^2\ n = O(n)$

したがって，$T(n) = O(n)$