

# Algoritmusok és adatszerkezetek II.

## Geometriai algoritmusok

Szegedi Tudományegyetem



## Definíció

A  $P_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$  pontot  $P_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$  és  $P_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$  pontok **konvex kombinációjának** nevezzük, amennyiben  $x_3 = (1 - \alpha)x_1 + \alpha x_2$ , valamint  $y_3 = (1 - \alpha)y_1 + \alpha y_2$  teljesül valamely  $0 \leq \alpha \leq 1$ -ra

## Definíció

$\overline{P_1 P_2}$  szakasz a  $P_1$  és  $P_2$  pontokból konvex kombinációinak halmaza

## Megjegyzés

Ha a pontok sorrendje is számít, irányított szakaszcól beszélünk, és  $\overrightarrow{P_1 P_2}$  módon jelöljük  
 $\vec{p}$ -vel  $\overrightarrow{OP}$ -t, vagyis az  $O$  origóból a  $P$ -be menő irányított szakaszt (vektort) jelöljük

## $P_1 \times P_2$ keresztszorzata

$$\det \left( \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \right) = x_1 y_2 - x_2 y_1 = P_1 \times P_2 = -P_2 \times P_1$$

## Megjegyzés

A keresztszorzat valójában háromdimenziós fogalom: egy  $\vec{p}_1$ -re és  $\vec{p}_2$ -re merőleges, velük **jobbsodrású rendszert alkotó vektor**, melynek hossza  $|x_1 y_2 - x_2 y_1|$ .



## $P_1 \times P_2$ keresztszorzata

$$\det \left( \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \right) = x_1 y_2 - x_2 y_1 = P_1 \times P_2 = -P_2 \times P_1$$

## Megjegyzés

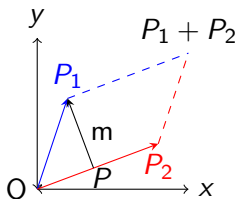
A keresztszorzat valójában háromdimenziós fogalom: egy  $\vec{p}_1$ -re és  $\vec{p}_2$ -re merőleges, velük **jobbsodrású rendszert alkotó vektor**, melynek hossza  $|x_1 y_2 - x_2 y_1|$ .

Más megfogalmazásban  $\vec{p}_1 \times \vec{p}_2 = \|\vec{p}_1\| \|\vec{p}_2\| \sin(\theta) \hat{n}$ , ahol  $\theta$  a  $\vec{p}_1$  és  $\vec{p}_2$  által bezárt szög, valamint  $\hat{n} \perp \vec{p}_1$  és  $\hat{n} \perp \vec{p}_2$



## Keresztszorzat mint előjeles terület

$P_1 \times P_2$  megadja az  $O$ ,  $P_1$ ,  $P_2$ ,  $P_1 + P_2$  koordinátákkal rendelkező paralelogramma előjeles területét



- $P_1 \times P_2 < 0 \Rightarrow P_1$ -ből jobbra fordulva érjük el  $P_2$ -t
- $P_1 \times P_2 > 0 \Rightarrow P_1$ -ből balra fordulva érjük el  $P_2$ -t
- $P_1 \times P_2 = 0 \Rightarrow P_1$  és  $P_2$  kollineáris



# Merre fordul a következő szakasz?

- $\overline{P_0P_1}$  és  $\overline{P_1P_2}$  szakaszokat folyamatosan bejárva merre kell fordulni  $P_1$  pontban?
- Az előzőekben lényegében az origó viselkedett  $P_0$ -ként



# Merre fordul a következő szakasz?

- $\overline{P_0P_1}$  és  $\overline{P_1P_2}$  szakaszokat folyamatosan bejárva merre kell fordulni  $P_1$  pontban?
- Az előzőekben lényegében az origó viselkedett  $P_0$ -ként

Ötlet: tegyük úgy, mintha  $P_0$  lenne az origó

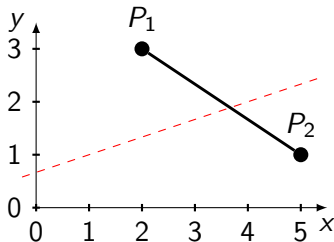
$$(P_1 - P_0) \times (P_2 - P_0) = \det \left( \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix} \right)$$

- Szemléletesen:  $P_1$ -ből és  $P_2$ -ből  $P_0$ -t kivonva  $P_0$  központúvá tesszük a koordinátarendszerünket



## Átfogó szakasz

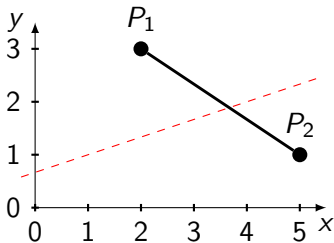
Egy  $\overline{P_1P_2}$  szakasz átfog egy egyenest, ha a  $P_1$  pont az egyenes egyik oldalára,  $P_2$  pont pedig a másik oldalára esik





## Átfogó szakasz

Egy  $\overline{P_1P_2}$  szakasz átfog egy egyenest, ha a  $P_1$  pont az egyenes egyik oldalára,  $P_2$  pont pedig a másik oldalára esik



## Átfedés meglétének eldöntése

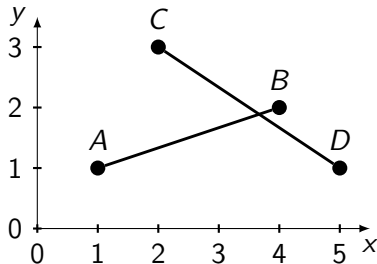
Egy (kevésbé hatékony) lehetőség, ha az egyenes egyenletét kiszámolva döntünk  $P_1$  és  $P_2$  relatív helyzetéről

**Támaszkodjunk helyette a forgásirányokra!**



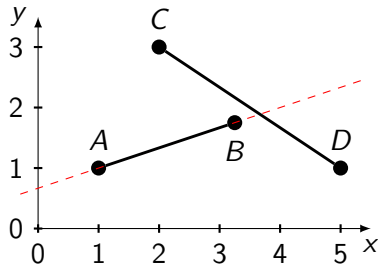
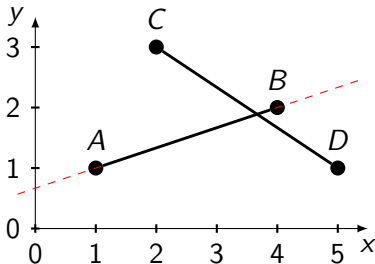
## Szükségesség

$\overline{CD}$  úgy metszheti  $\overline{AB}$  szakaszt, ha  $\overline{CD}$  átfogja az  $\overline{AB}$  szakaszra illeszkedő egyenest.



## Szükségesség

$\overline{CD}$  úgy metszheti  $\overline{AB}$  szakaszt, ha  $\overline{CD}$  átfogja az  $\overline{AB}$  szakaszra illeszkedő egyenest.



```
FORGÁSIRÁNY(P1, P2, P3) {  
    return (P2.x-P1.x)*(P3.y-P1.y) - (P3.x-P1.x)*(P2.y-P1.y)  
}
```

```
METSZŐSZAKASZOK(A, B, C, D) {  
    d1 = FORGÁSIRÁNY(A, B, C)  
    d2 = FORGÁSIRÁNY(A, B, D)  
    d3 = FORGÁSIRÁNY(C, D, A)  
    d4 = FORGÁSIRÁNY(C, D, B)  
    return d1 * d2 < 0 és d3*d4 < 0  
}
```



```
FORGÁSIRÁNY(P1, P2, P3) {  
    return (P2.x-P1.x)*(P3.y-P1.y) - (P3.x-P1.x)*(P2.y-P1.y)  
}
```

```
METSZŐSZAKASZOK(A, B, C, D) {  
    d1 = FORGÁSIRÁNY(A, B, C)  
    d2 = FORGÁSIRÁNY(A, B, D)  
    d3 = FORGÁSIRÁNY(C, D, A)  
    d4 = FORGÁSIRÁNY(C, D, B)  
    return d1 * d2 < 0 és d3*d4 < 0  
}
```

Ezzel csak „valódi” metszéseket találunk meg, a szakaszra illeszkedő végpontú szakaszt nem kezeltük így



- Adott szakaszok  $n$  elemű halmaza, és tudni szeretnénk, hogy van-e köztük egymást metsző szakaszpár
- Nyers erővel  $\binom{n}{2} = O(n^2)$
- Bizonyos egyszerűsítő feltételezések mellett  $O(n \log(n))$  is megoldható
  - $y$  tengellyel párhuzamos szakaszokat nem kezelünk
  - Adott szakaszok között nincs 3 egy pontban metsző



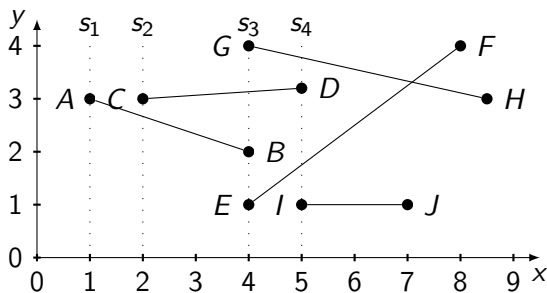
# Metsző szakaszpár keresése – söprés

## Söprés

Söprés során egy képzeletbeli függőleges *söprő egyenes* halad át geometriai objektumok halmazán (általában balról jobbra)

## Két szakasz összehasonlítása adott $x$ koordináta mentén

$s_1$  szakasz fölötté van  $s_2$ -nek  $x$ -nél ( $s_1 \succ_x s_2$ ), ha  $s_1$   $y$ -koordinátája nagyobb  $s_2$   $y$ -koordinátájánál adott  $x$ -koordináta mentén.



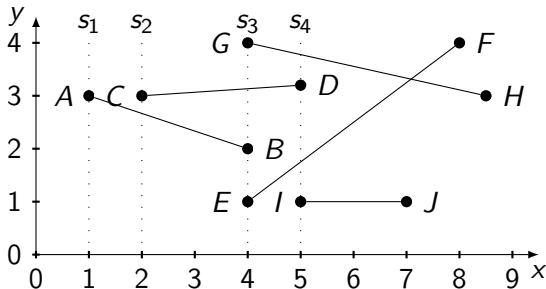
# Metsző szakaszpár keresése – söprés

## Söprés

Söprés során egy képzeletbeli függőleges *söprő* egyenes halad át geometriai objektumok halmazán (általában balról jobbra)

## Két szakasz összehasonlítása adott $x$ koordináta mentén

$s_1$  szakasz fölötté van  $s_2$ -nek  $x$ -nél ( $s_1 \succ_x s_2$ ), ha  $s_1$   $y$ -koordinátája nagyobb  $s_2$   $y$ -koordinátájánál adott  $x$ -koordináta mentén.



## Példák

$\overline{GH} \succ_4 \overline{EF}$ , de  $\overline{EF} \succ_8 \overline{GH}$





# Szakaspár metszése a söprő egyenes szemszögéből

- Bármely adott  $x$  értékre a  $\succ_x$  reláció az  $x$ -nél lévő söprő egyenest metsző szakaszok teljes rendezése

## Kulcsészrevétel

Ha  $\exists$  egymást metsző szakaspár  $\Rightarrow \exists$  söprő egyenes, mely mentén való rendezés esetén azok egymás után következnek

- A söprő egyenes mozgatása
  - Elég a szakaszvégpontokban a be,-és kilépő szakaszok alapján összehasonlításokat végezni
- Kétféle adathalmazt kell kezelni a keresés során
  - Söprő egyenes állapotleírása
  - Esetpontok rendezett listája



# Szakaszpár metszése – állapotleírás és esetpontok

- A söprő egyenes állapotleírása a szakaszok adott egyenes menti  
     $\succ$  teljes rendezési reláció szerinti rendezését tartalmazza
- A söprő egyenes állapotleírásában változás csak esetpontokban  
    (=szakaszvégpontokban) történik
- Esetpontok rendezése
  - Kovertikális (azonos  $x$ -koordinátájú) szakaszvégpontok esetén  
        a bal/belépő végpontokat a jobb/kilépő végpontok elé soroljuk
- Elegendő azt vizsgálni csupán, hogy a
  - belépő szakaszok metszik-e megelőzőjüket/rákövetkezőjüket
  - kilépő szakaszok megelőzője és rákövetkezője metszi-e egymást



# Szakaszpár metszése – állapotleírás és esetpontok

- A söprő egyenes állapotleírása a szakaszok adott egyenes menti  
➤ teljes rendezési reláció szerinti rendezését tartalmazza
- A söprő egyenes állapotleírásában változás csak esetpontokban  
(=szakaszvégpontokban) történik
- Esetpontok rendezése
  - Kovertikális (azonos  $x$ -koordinátájú) szakaszvégpontok esetén  
a bal/belépő végpontokat a jobb/kilépő végpontok elé soroljuk
- Elegendő azt vizsgálni csupán, hogy a
  - belépő szakaszok metszik-e megelőzőjüket/rákövetkezőjüket
  - kilépő szakaszok megelőzője és rákövetkezője metszi-e egymást

## Fontos

A mindenkori állapotleírást kiegyensúlyozott keresőfában tároljuk



# Metsző szakaspár keresése

```
VAN-E-METSZŐ-SZAKASZPÁR(S) { // T az állapotleírás fája
    L = S-beli szakaszvégpontok rendezett listája
    for p in L
    do
        if p egy s szakasz bal végpontja {
            BESZÚR(T,s)
            if MEGELŐZ(T,s) vagy RÁKÖVET(T,s) metszi s-et {
                return IGAZ
            }
        }
        if p egy s szakasz jobb végpontja {
            if MEGELŐZ(T,s) metszi RÁKÖVET(T,s)-t {
                return IGAZ
            }
            TÖRÖL(T,s)
        }
    return HAMIS
}
```



- Állapotleírás  $T$  kiegyensúlyozott fájának létrehozása  $O(1)$
- Szakaszvégpontok rendezése  $O(n \log n)$
- for ciklus "hossza" legfeljebb  $2n = O(n)$
- Megelőz, illetve Rákövet metódusok végrehajtása  $O(\log n)$



# Metsző szakaszpár keresésének futási ideje

- Állapotleírás  $T$  kiegyensúlyozott fájának létrehozása  $O(1)$
- Szakaszvégpontok rendezése  $O(n \log n)$
- for ciklus "hossza" legfeljebb  $2n = O(n)$
- Megelőz, illetve Rákövet metódusok végrehajtása  $O(\log n)$

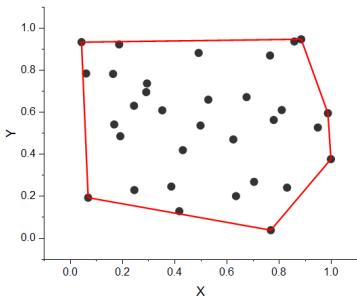
⇒ Összességében  $O(n \log n)$



# Konvex burok definíciója

## Konvex burok

$Q$  pontthalmaz konvex burka az a legkisebb  $P$  konvex poligon, amelyre  $Q$  minden pontja vagy  $P$  határán van, vagy a belsejében.

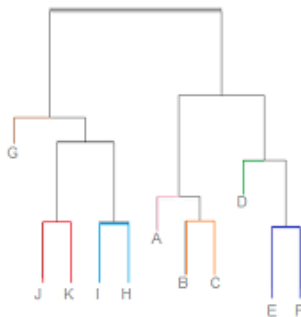
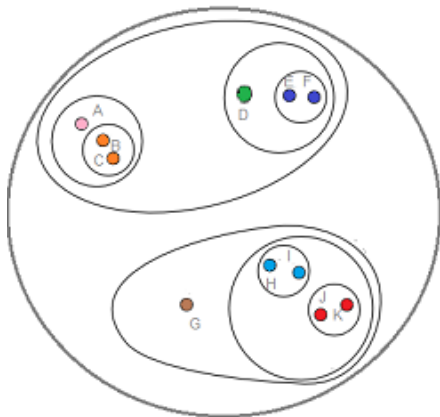


$Q$  konvex burkát  $CH(Q)$ -val jelöljük.  
 $CH(Q) \subseteq Q$  pontjait  $Q$  **extrém pontjainak**.  
Jelöljük a továbbiakban  $|Q|$ -t  $n$ -nel.



# CH gyakorlati felhasználása

- Objektumok ütközésének elkerülése: vegyük akadályok koordinátáit, ezek CH-a képezze az elkerülendő régiót
- Agglomeratív klaszterezés: gépi tanuló eljárás, melynek célja vektorokkal leírt megfigyelések homogén részsokaságainak kialakítása





## Tétel

*P pont csak abban az esetben extrém pontja Q-nak, ha az a Q-ból kialakítható háromszögek mindegyikén kívül esik, vagy annak egy csúcsa.*

- Egy csúcs extrém voltának eldöntése  $\binom{n-1}{3} = O(n^3)$
- Mivel  $n$  csúcs van, így az elvi algoritmus  $O(n^4)$  futási idejű



- Különféle megközelítések léteznek
  - 1 Növekményes módszer: "balról jobbra" számítjuk ki a CH-t
  - 2 Oszd-meg-és-uralkodj: CH számítása a pontok részhalmazára, majd ezek egyesítése
  - 3 Eltávolító és kereső módszer
- $n$  pont esetében többnyire  $O(n \log n)$  futási idejű algoritmusok, de vannak  $O(nh)$ , illetve  $O(n \log h)$  algoritmusok is<sup>1</sup>

---

<sup>1</sup> $h$  a CH-ban lévő csúcsok száma

# Konvex burok meghatározása hatékonyan

- Különféle megközelítések léteznek
  - 1 Növekményes módszer: "balról jobbra" számítjuk ki a CH-t
  - 2 Oszd-meg-és-uralkodj: CH számítása a pontok részhalmazára, majd ezek egyesítése
  - 3 Eltávolító és kereső módszer
- $n$  pont esetében többnyire  $O(n \log n)$  futási idejű algoritmusok, de vannak  $O(nh)$ , illetve  $O(n \log h)$  algoritmusok is<sup>1</sup>

## Megjegyzés

Egy  $O(nh)$  algoritmusnak nyilván csak  $h < \log n$  esetén van haszna

---

<sup>1</sup> $h$  a CH-ban lévő csúcsok száma

# Konvex burok – Graham-féle pásztázás

```
GRAHAM-PÁSZTÁZÁS(S) {  
    P0 = minimális x-koordinátájú Q-beli pont (több ilyen  
    esetén válasszuk az y-koordináta szerint is minimálisat)  
    P = POLÁRSZÖGSZERINTRENDEZ(Q)  
    S = VERMETLÉTESÍT()  
    VEREMBE(P0, S)  
    VEREMBE(P1, S)  
    VEREMBE(P2, S)  
    for i=3 to m {  
        while LEGFELSŐ-ALATTI(S), LEGFELSŐ(S) és Pi nem  
            fordul balra {  
                VEREMBŰL(S)  
            }  
        VEREMBE(Pi, S)  
    }  
    return S  
}
```

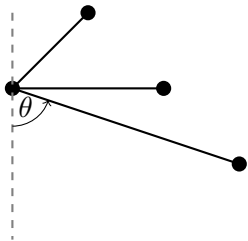


# Polárszög szerinti rendezés

## Pontok helyzetének polárkoordinátákkal történő megadása

$P$  pontot  $(x, y)$  koordinátapár helyett egy referenciaponttól vett távolság és egy referenciaírányral bezárt szög párosaként adjuk meg

- A referenciaponttól számított  $\frac{\Delta y}{\Delta x}$  eltérések szerinti sorrend adja a pontok polárszög szerinti rendezését
  - Azonos hányadossal rendelkező pontok közül azt soroljuk előre, amelyek a referenciaponthoz közelebb találhatók<sup>2</sup>



## Megjegyzés

A valóságban a vektor hosszának és forgásszögének kiszámítása költséges és numerikusan sem jó ötlet (sqrt és szögfüggvény miatt)

<sup>2</sup>Kivéve, ha  $\Delta x = 0$ , mert akkor a másodlagos rangsorolást fordítva végezzük



- Numerikus hibák mérséklése és a 0-val való osztás elkerülésére
  - a forgásszöget a pontszorzattal számoljuk
  - a vektor hossza helyett a négyzetét számoljuk
    - Emlékeztetőül:  $\|\vec{p}\|_2 = \|\vec{OP}\|_2 = \sqrt{\sum_{i=1}^d P[i]^2}$
  - Bármilyen összehasonlító rendezést (pl. gyorsrendezés) tudunk használni a forgásirányokra támaszkodva

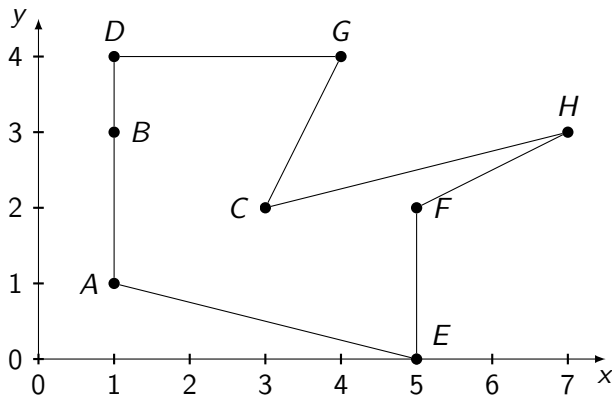
## pontok lista rendezése R referenciapont mentén

```
Collections.sort(pontok, (A, B) ->  
    return (int) Math.signum(FORGÁSIRÁNY(R, A, B));  
);
```



# Zárt nem metsző poligon

- A pontokat a polárszöges rendezés sorrendjében összekötve megkapjuk a pontok által alkotott zárt, nem metsző poligont



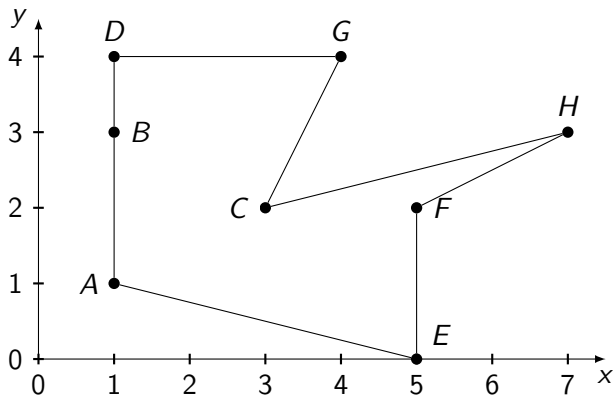
## A rendezés

- 1 A
- 2 E
- 3 F
- 4 H
- 5 C
- 6 G
- 7 D
- 8 B



# Zárt nem metsző poligon

- A pontokat a polárszöges rendezés sorrendjében összekötve megkapjuk a pontok által alkotott zárt, nem metsző poligont



## A rendezés

- 1 A
- 2 E
- 3 F
- 4 H
- 5 C
- 6 G
- 7 D
- 8 B

A pontok polárszög szerinti rendezésével nyert sorrendben történő összekötése nem feltétlen eredményez konvex poligont





- Kezdetben:  $S = [A, E, F]$ 
  - 1 Forgásirány(E,F,H), Veremből(S), Forgásirány(A,E,H), Verembe(S, H)
  - 2 Forgásirány(E,H,C), Verembe(S, C)
  - 3 Forgásirány(H,C,G), Veremből(S), Forgásirány(E,H,G), Verembe(S, G)
  - 4 Forgásirány(H,G,D), Verembe(S,D)
  - 5 Forgásirány(G,D,B), Verembe(S,B)
- Végezetül:  $S = [A, E, H, G, D, B]$

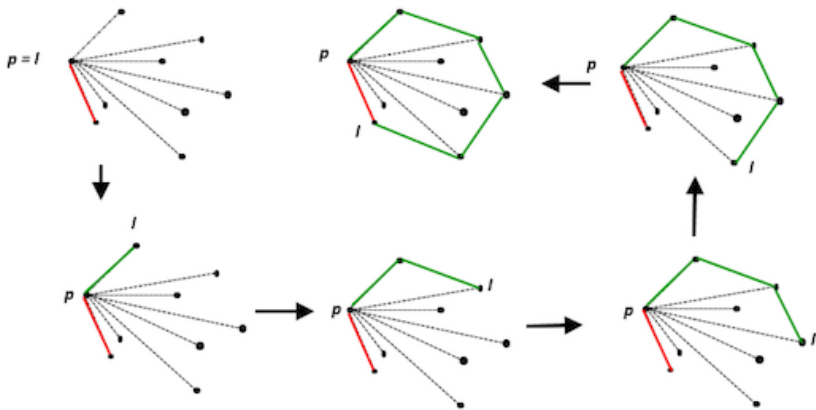


# CH meghatározása Jarvis meneteléssel

- Ajándékcsomagolás elvén működik  $O(nh)$  időben
- Kezdsnek válasszuk ki a legbaloldalibb pontot
- Amíg vissza nem érünk a kezdőpontba, válasszuk ki a legutolsónak választott ponttól leginkább balra eső pontot (vagyis azt a pontot, amelytől minden további ponthoz jobbra fordulásra van szükség)
- A Jarvis-féle menetelés folyamatosan bővíti  $CH(Q)$ -t (egy iterációban  $O(n)$  próbát tesz), "vakvágányoktól mentes"
  - A Graham-féle pásztázás minden csúcsot potenciálisan  $CH(Q)$ -beliként kezel



# Jarvis menetelése illusztrálva

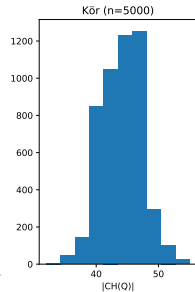
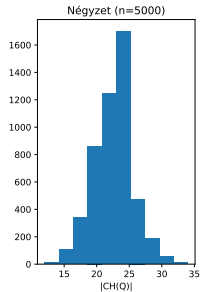
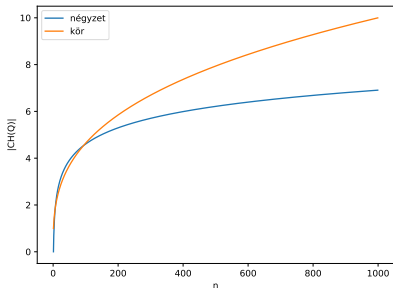


The execution of jarvis's March



# CH(Q) várható mérete

- Négyzeten/körleapon 2d-ben **véletlenszerűen** elhelyezkedő ponthalmaznak átlagosan  $O(\log n)/O(n^{1/3})$  elemű a CH-ja



Ne feledjük!

Legrosszabb esetben  $CH(Q) = n$  is teljesülhet



- $n$  elemű ponthalmazban találjuk meg azon  $(P_i, P_j)$  pontpárt, melyek a legtávolabb fekszenek egymástól
  - $P_i$  és  $P_j$  pontok távolságát euklideszi értelemben véve
$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
- Nyers erővel ez is  $\binom{n}{2} = O(n^2)$  összehasonlítás lenne

## Észrevétel

A ponthalmaz legtávolabbi pontpárja a CH-on található csúcspárok valamelyike kell legyen



- Az észrevétel indirekt módon bizonyítható
  - Tegyük föl, hogy  $P_i$  és  $P_j$  pontok közötti távolság maximális, és  $P_i \in CH(Q)$ ,  $P_j \notin CH(Q)$
  - Vegyünk egy  $P_i$  középpontú  $\|\overline{P_i P_j}\|$  sugarú gömböt  
 $\Rightarrow$  egyetlen  $P_k \in Q$  pont sem eshet a gömbön kívül



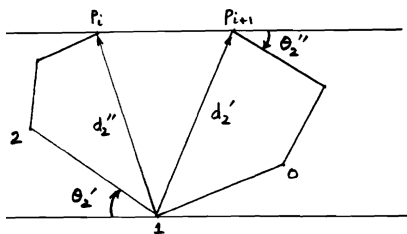
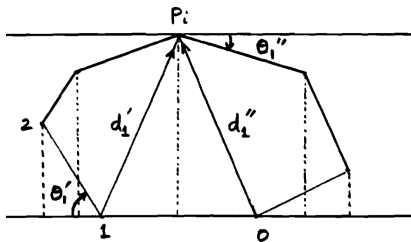
# A legtávolabbi pontpár a CH-on lesz

- Az észrevétel indirekt módon bizonyítható
  - Tegyük föl, hogy  $P_i$  és  $P_j$  pontok közötti távolság maximális, és  $P_i \in CH(Q)$ ,  $P_j \notin CH(Q)$
  - Vegyünk egy  $P_i$  középpontú  $\|\overline{P_i P_j}\|$  sugarú gömböt
    - $\Rightarrow$  egyetlen  $P_k \in Q$  pont sem eshet a gömbön kívül, máskülönben nem  $(P_i, P_j)$  alkotná a legtávolabbi pontpárt
    - $\Rightarrow P_j \in CH(Q)$ , ami viszont ellentmondás



# Legtávolabbi pontpár megtalálása – forgatásos söprés

- Átellenes (párhuzamos egyenesekre illeszkedő) csúcsokat keresünk, és ezeket görgetjük végig:  $O(h)$





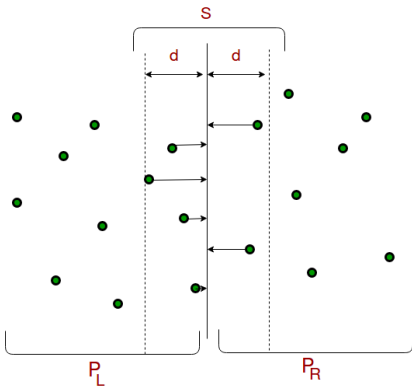
- Adott  $n$  elemű  $P$  ponthalmazra mi az a  $(P_i, P_j) \in P \times P$  pontpár, ami a legközelebb helyezkedik el egymáshoz?
- Nyers erővel szintén  $\binom{n}{2} = O(n^2)$
- Oszd meg és uralkodj eljárással  $O(n \log(n))$  is megoldható



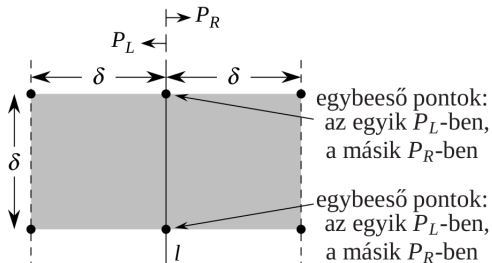
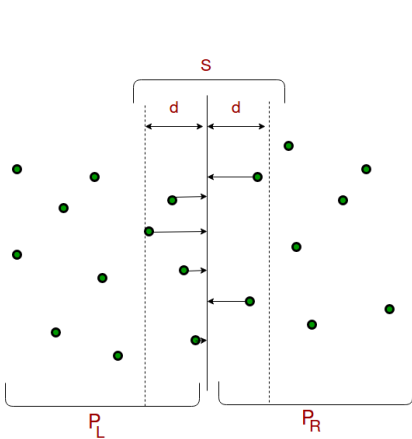
- Ha legfeljebb 3 pont maradt, akkor használjuk a nyers erő módszerét
- Egyébként hajtsuk végre a következőket
  - Vegyük azt az  $l$  egyenest, ami két egyenlő részre vágja a pontokat ( $P_L$  és  $P_R$ )
  - A  $P_L$  és  $P_R$ -beli pontok közül rekurzívan határozzuk meg a legközelebbi pontpár közötti távolságot  $d = \min(d_L, d_R)$
  - Döntsük el, hogy találni-e olyan  $(P_i, P_j)$  pontpárt, melyre  $P_i \in P_L$  és  $P_j \in P_R$ , továbbá távolságuk  $d' < d$



# Legközelebbi pontpár megtalálása – illusztráció



# Legközelebbi pontpár megtalálása – illusztráció



## Jó hír

Elegendő az  $y$  koordináta alapján vett rendezés szerinti 7 rákövetkező ponttal összevetni az  $S$ -beli pontokat



# Legközelebbi pontpár megtalálásának hatékonysága

- A pontok  $x$  és  $y$  koordináta szerinti előrendezésével kezdünk  $O(n \log n)$
- A rekurzív lépés során  $T(n) = 2 * T(\frac{n}{2}) + O(n)$  művelet
  - Előrendezés nélkül a rekurzív lépés  $T(n) = 2 * T(\frac{n}{2}) + O(n \log n)$  műveletigényű lenne, a teljes eljárás mester módszerrel kapott műveletigénye pedig  $O(n \log^2 n)$ -re nőne
- Az eljárás több szempontból is hasonlít az összefésülő rendezés működéséhez



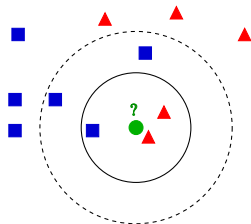
# Legközelebbi szomszéd(ok) meghatározása

## Probléma

Adott ponthoz találjuk meg a hozzá legközelebb eső ponto(ka)t

## Felhasználása

Gépi tanulás: k-legközelebbi osztályozó, klaszterező eljárások



## k-d fák

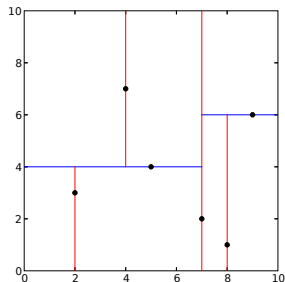
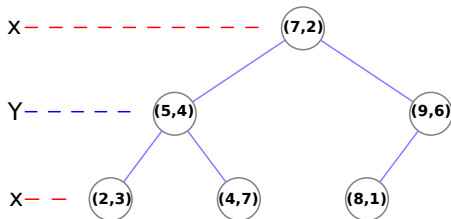
$n$  pont esetén a k-d fák fő műveletei átlagos esetben  $O(n \log n)$  műveletigényűek

- Hatékonysága **statikus ponthalmaz** esetén garantált
- Egy k-d fa módosítása (beszúrás/törlés) a kiegyensúlyozottságának megtartása mellett nem triviális



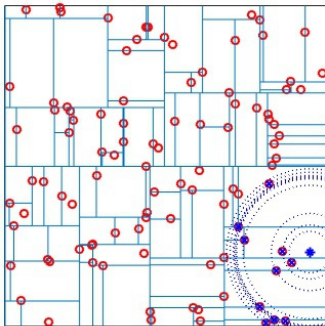
# k-d fák illusztrációja

- A fa minden szintje a pontok váltakozó koordináták mentén történő megfelelését szolgálják (egyéb stratégiák is léteznek)
  - A fa egy csúcsa ossza ketté a részfában található pontokat az adott szintre vonatkozó dimenzió mentén
  - A fa kiegyensúlyozott lesz, mivel a részfák olyan feltételeket definiálnak, amelyeket a mediánok mentén hozunk létre

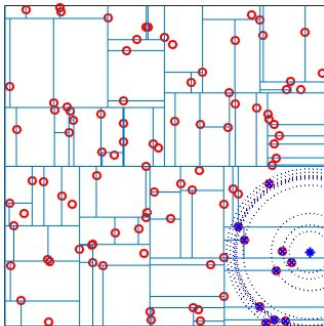




- Keressük meg azt a régiót, ahova a lekérdezett pont esik
- Nem garantált, hogy a legközelebbi pont ebben a régióban lesz



- Keressük meg azt a régiót, ahova a lekérdezett pont esik
- Nem garantált, hogy a legközelebbi pont ebben a régióban lesz
  - Viszont az azon belüli legkisebb távolság alapján sok régió esélytelenné válik a legközelebbi pont tartalmazására nézve



- A geometriai algoritmusok számos gyakorlati probléma (pl. gépi tanulás) megoldása során felmerülnek
- A numerikus hibák (egy része) kiküszöbölhető ForgásIrány használatával
- Nagy méretű inputok esetén fontos, hogy a négyzetes (vagy annál is rosszabb) futási idejű algoritmusoknál hatékonyabbakat használjunk

