Planfejning

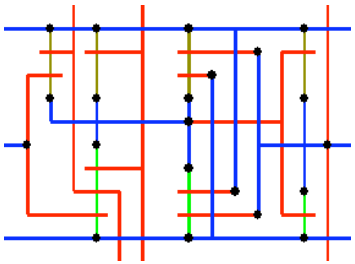
Skæring

Geometrisk skæring

Afgørelse af om der findes skæringer blandt geometriske objekter
Bestemmelse af alle skæringspunkter

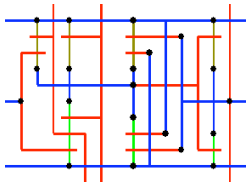
Løsningsmetoder:

- Rå kraft
- **Planfejning** (eng. plane sweep)

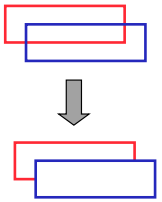


Anvendelser

Design af integrerede kredsløb:



Computergrafik (fjernelse af skjulte linjer)

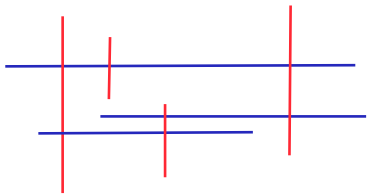


Skæring af vandrette og lodrette linjestykker

Givet:

- H = horisontale linjestykker
- V = vertikale linjestykker
- $S = H \cup V$
- n = antallet af linjestykker, $|S|$

Find alle par af skærende linjestykker (under antagelse af, at der ikke er sammenfaldende linjestykker)



Rå kraft algoritme


```
for each h in H
  for each v in V
    if h intersects v
      report(h, v)
```

Algoritmen kører i $O(n_H \cdot n_V) = O(n^2)$ tid

Men antallet af skæringer kan være meget mindre end $O(n^2)$

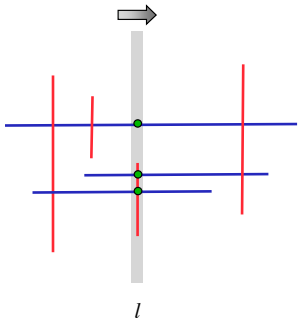
Vi ønsker en **uddatasensitiv** algoritme med køretid $f(n, s)$, hvor s er antallet af skæringer

Planfejning

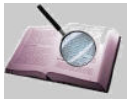
En lodret **feje-linje** l afsøger området fra venstre mod højre, startende til venstre for alle inddatapunkter

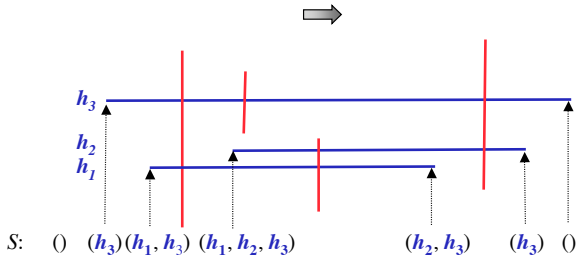
Under afsøgningen benyttes en ordbog, S, der indeholder alle de vandrette linjestykker, der skæres af l, sorteret i stigende rækkefølge med hensyn til deres y-koordinat

Et vandret linjestykke indsættes i S, når l møder dets venstre endepunkt. Et vandret linjestykke fjernes fra S, når l møder dets højre endepunkt

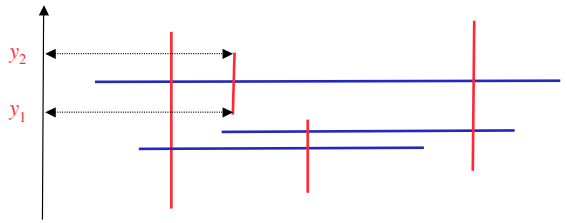


Ændringer i ordbogen





1-dimensional områdesøgning benyttes til at bestemme skæringer



Når et lodret linjestykke mødes, foretages områdesøgning i S med et interval, der er givet ved y-koordinaterne for linjestykkets to endepunkter

Hændelser og aktioner

Hændelse	Aktion
Et venstre endepunkt for et vandret linjestykke h mødes	Indsæt h i ordbogen S
Et højre endepunkt for et vandret linjestykke h mødes	Fjern h fra ordbogen S
Et lodret linjestykke v mødes	Foretag områdesøgning med interval-grænser givet ved y-koordinaterne for v's endepunkter

Datastrukturer

Ordbogen
- skal indeholde vandrette linjestykker
- skal muliggøre indsættelse, fjernelse og områdesøgning

Løsning: et AVL-træ eller et rød-sort-træ (nøglerne er y-koordinaterne)

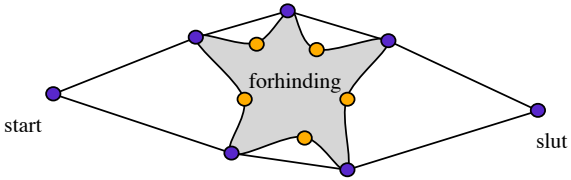
Handlingsplanen
- skal indeholde x-koordinaterne for hændelserne i den rækkefølge, de indtræffer
- skal muliggøre et sekventiel gennemløb

Løsning: et array eller en liste, der er sorteret med hensyn til hændelsernes x-koordinat

Tidskompleksitet

Sortering af hændelser	$O(n \log n)$
Hændelser	
Venstre endepunkt for vandret linje	Antal $\leq n$ Tid for hver indsættelse i S: $O(\log n)$
Højre endepunkt for vandret linje	Antal $\leq n$ Tid for hver fjernelse fra S: $O(\log n)$
Lodret linje	Antal $\leq n$ Tid for områdesøgning: $O(\log n + s_h(v))$
Samlet tidskompleksitet:	$O(n \log n + \Sigma_{v \in V} s_h(v))) = O(n \log n + s)$

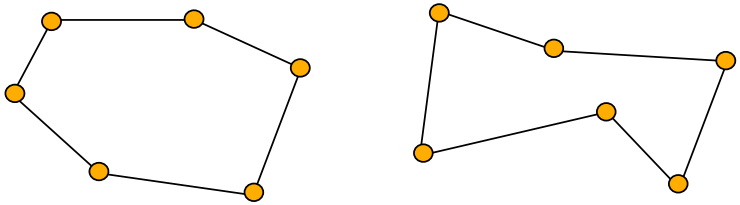
Konvekst hylster



Konveks polygon

En **konveks polygon** er en polygon uden skæringer, hvis indre vinkler alle er konvekse (d.v.s. mindre end $\pi = 180°$)

I en konveks polygon ligger ethvert linjestykke, der forbinder to knuder, helt inden for polygonen

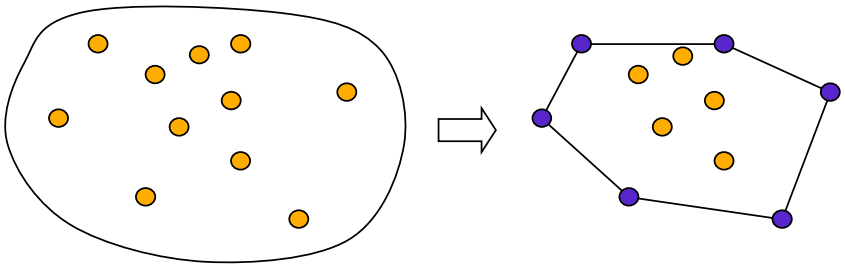


konveks ikke-konveks

Konvekst hylster

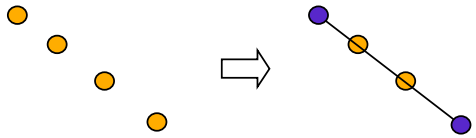
Det **konvekse hylster** for en mængde af punkter er den mindste konvekse polygon, der indeholder punkterne

Tænk på en elastik, der lægges stramt rundt om punkterne



Specialtilfælde

Det konvekse hylster er et **linjestykke**
- To punkter
- Alle punkter ligger på linje



Det konvekse hylster er et **punkt**
- Der er kun ét punkt
- Alle punkter er sammenfaldende

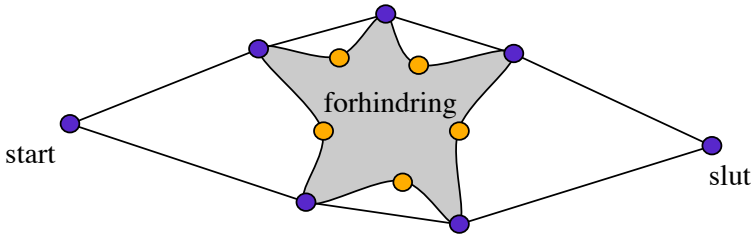
Anvendelser

• Planlægning af bevægelse

Find en optimal rute for en robot, der undgår forhindringer

• Geometriske algoritmer

Bestemmelse af det konvekse hylster er en form for to-dimensional sortering



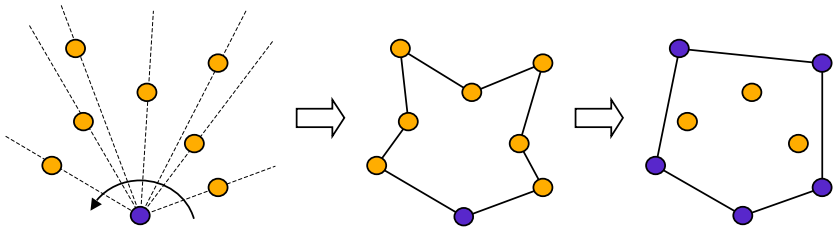
Bestemmelse af det konvekse hylster

Nedenstående metode bestemmer det konvekse hylster for en mængde af punkter:

Fase 1: Find det laveste punkt (**ankerpunktet**)

Fase 2: Dan en ikke-skærende polygon ved at sortere punkterne imod uret rundt om ankerpunktet

Fase 3: Så længe polygonen har et ikke-konvekst hjørne, så fjern det



Orientering

Orienteringen af tre punkter i planet er *"med uret"*, *"mod uret"* og *"på linje"*

orientation(***a***, ***b***, ***c***)

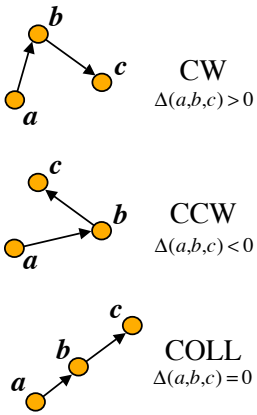
med uret (CW, drej højre om)

mod uret (CCW, drej venstre om)

på line (COLL, ingen drejning)

Orienteringen af tre punkter er karakteriseret ved fortegnet af determinanten, hvis absolutte værdi er det dobbelte areal af trekanten med hjørner ***a***, ***b*** og ***c***

$$\Delta(a,b,c) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} = x_a y_b - x_b y_a + x_c y_a - x_a y_c + x_b y_c - x_c y_b$$



Sortering efter vinkel

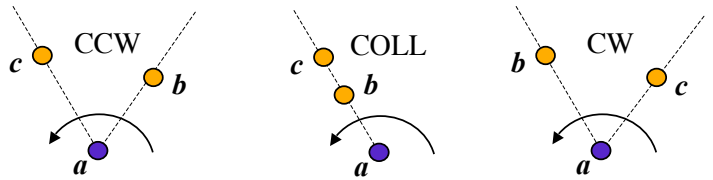
Bestemmelse af vinkler ud fra koordinater er besværlig og fører til numerisk unøjagtighed

Vi kan sortere punkterne efter deres vinkel med hensyn til ankerpunktet ***a*** ved hjælp af en comparator-baseret sammenligningsfunktion:

b < ***c*** ⇔ *orientation*(***a***, ***b***, ***c***) = CCW

b = ***c*** ⇔ *orientation*(***a***, ***b***, ***c***) = COLL

b > ***c*** ⇔ *orientation*(***a***, ***b***, ***c***) = CW



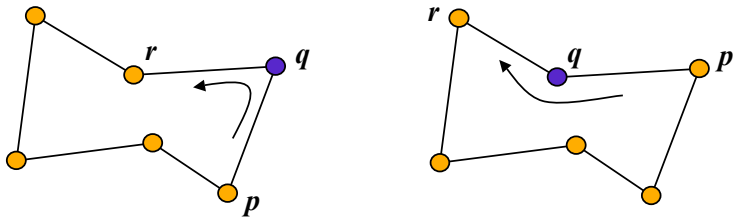
Fjernelse af ikke-konvekse hjørner

Afgørelse af, om hjørnet er konvekst, kan foretages ved brug af orienteringsfunktionen:

Lad p, q og r være tre konsekutive hjørner for polygonen i rækkefølge mod uret

q konveks $\Leftrightarrow$ *orientation*$(p, q, r) =$CCW

q ikke-konveks $\Leftrightarrow$ *orientation*$(p, q, r) =$CW eller COLL

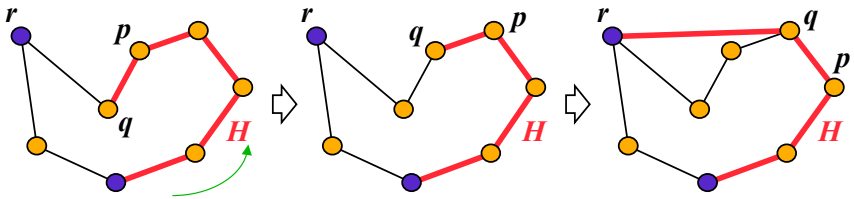


Graham-afsøgning

Graham-afsøgning er en systematisk procedure til at fjerne ikke-konvekse hjørner fra en polygon

Polygonen traverseres imod uret, og en sekvens H af hjørner vedligeholdes

for each vertex r of the polygon
  Let q and p be the last and second last vertex of H
  while *orientation*$(p, q, r) =$CW or COLL
    remove q from H
    $q \leftarrow p$
    $p \leftarrow$ vertex preceding p in H
  Add r to the end of H



Analyse

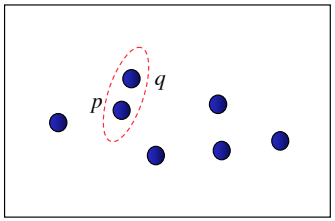
Bestemmelse af det konvekse hylster for en mængde af punkter tager $O(n \log n)$ tid:

- Bestemmelse af ankerpunktet tager $O(n)$ tid
- Sortering af punkterne imod uret omkring ankerpunktet tager $O(n \log n)$ tid
  Brug orienterings-comparatoren og en hvilken som helst sorteringsalgoritme, der kører i $O(n \log n)$ tid (f.eks., heap-sort eller merge-sort)
- Graham-afsøgningen tager $O(n)$ tid
  Ethvert punkt indsættes netop en gang i sekvensen H
  Ethvert hjørne fjernes højest én gang fra sekvensen H

Nærmeste par af punkter

Nærmeste punkter

Givet en mængde, P, af n punkter i planet (f.eks. byer i Danmark, computere i et netværk, transistorer på en printplade)
Find to punkter, p og q, hvis indbyrdes afstand, $dist(p, q)$, er minimal

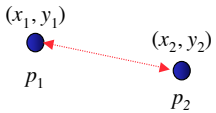


Algoritmer:

- rå kraft $O(n^2)$
- planfejning $O(n \log n)$
- del-og-hersk $O(n \log n)$

Rå kraft

Beregn alle afstande og vælg den mindste



$$dist(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

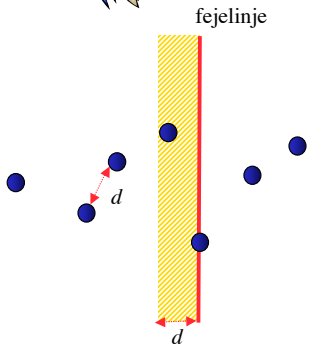
Tidskompleksitet: $O(n^2)$

Planfejning

Vi behøver ikke at beregne alle afstande

Planfejning kan benyttes. Vi udnytter følgende observation:

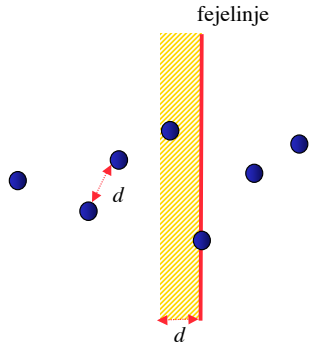
Hvis det nærmeste par af punkter til venstre for fejelinjen har en afstand på d, kan det næste punkt, der mødes af linjen ikke være nærmeste par med et punkt, der ligger mere end d enheder til venstre for linjen



Lagret information

Vedligehold følgende information:

- nærmeste par (p, q) af punkter, der er fundet indtil nu, samt deres afstand, d
- ordnet ordbog, S, af alle de punkter, der ligger i bæltet med bredde d enheder til venstre for fejelinjen, idet deres y-koordinat bruges som nøgle



Opdatering

Når fejelinjen møder et punkt, p:

(1) Fjern alle punkter, r, hvor $x(p) - x(r) \geq d$

(2) Find det nærmeste punkt, q, til p ved søgning i S

(3) Hvis $dist(p, q) < d$, så opdater det aktuelt nærmeste par og d

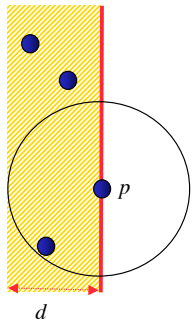
(4) Indsæt p i S

Søgning i ordbogen

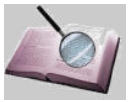
Hvor hurtigt kan vi søge i ordbogen?

Bemærk: der kan være $O(n)$ punkter i ordbogen!

I stedet for at søge i hele bæltet, kan vi nøjes med at søge i en halvcirkel med centrum i p og radius d



Søgning i ordbogen
(fortsat)



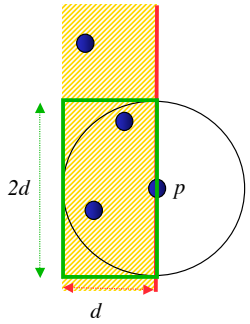
Hvorledes kan vi søge i en halvcirkel?

Et rektangel er "næsten" en halvcirkel
Foretag en områdesøgning i intervallet
$[y(p) - d, y(p) + d]$

Brug rå kraft til søgning blandt de punkter, der blev resultatet af områdesøgningen

Kan der ikke være mange punkter at søge i?

Nej, områdesøgningen kan aldrig resultere i mere end 6 punkter
(bevis herfor baseres på, at ethvert par af punkter i S har en afstand på mindst d)



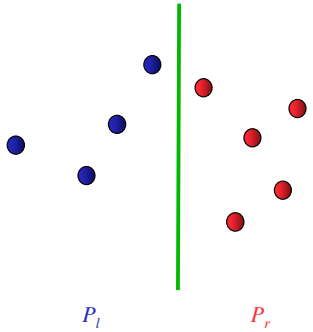
Tidskompleksitet

Sortering af hændelser	$O(n \log n)$
Hvert punkt indsættes og fjernes fra S præcis en gang	Samlet tid for indsættelse og fjernelse: $O(n \log n)$
Der søges i ordbogen, hver gang et punkt indsættes i S	Hver forespørgsel tager $O(\log n + 6)$ tid Samlet tid for forespørgsel: $O(n \log n)$
Afstandsberegninger	$O(6n)$
Samlet tidskompleksitet:	$O(n \log n)$

Del og hersk

Sorter punkterne efter deres x-koordinat og opdel punktmængden i to halvdele

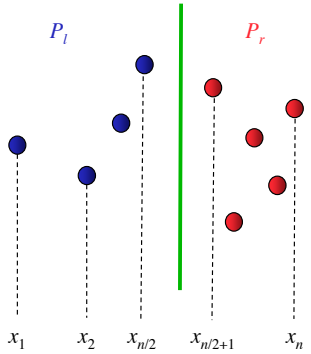
Det nærmeste par er i en af de to halvdele eller har et punkt i hver af de to halvdele



Del og hersk (2)

Fase 1:
Sorter punkterne efter deres x-koordinat

Opdel punktmængden i to halvdele:
$p_1, p_2, \cdots p_{n/2} \cdots p_{n/2+1}, \cdots, p_n$

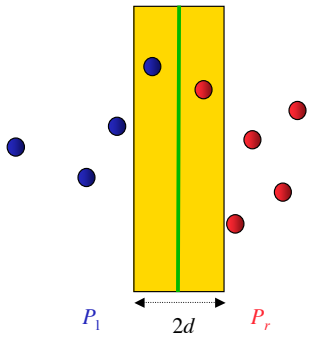


Del og hersk (3)

Fase 2:
Bestem rekursivt de to nærmeste par og deres indbyrdes afstande, d_l og d_r, i hver af de to halvdele

Find det nærmeste par og deres indbyrdes afstand, d_m, i det centrale bælte af bredde $2d$, hvor $d = \min\{d_l, d_r\}$

Returner $\min\{d_m, d_l, d_r\}$



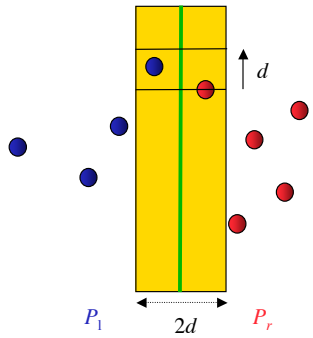
Del og hersk (4)

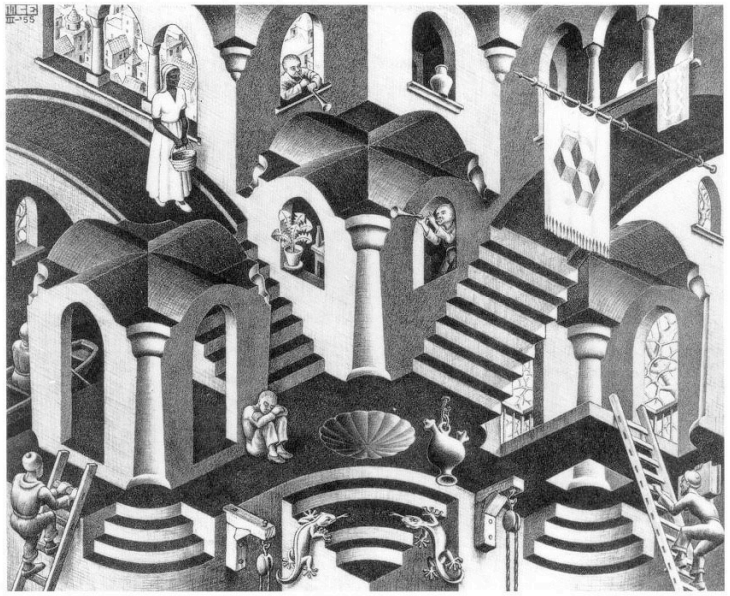
For ethvert punkt p i bæltet undersøges afstanden til de punkter, der ligger i intervallet $[y(p) - d, y(p)]$

Der er højst 4 sådanne punkter

Benyt en liste af punkterne sorteret efter deres x-koordinat og en liste af punkterne sorteret efter deres y-koordinat

Tidskompleksitet:
Fase 1: Sortering: $O(n \log n)$
Fase 2: Rekursionsligning $T(n) = 2T(n/2) + n$.
Vi får $T(n)$ er $O(n \log n)$
Samlet tid: $O(n \log n)$





M. Escher: Concave and Convex