Dynamische Programmierung

Überlappende Teilprobleme Algorithmus kehrt immer wieder zum gleichen Teilproblem zurück. Anzahl der TP polynomiell.
Beispiel: Längste gemeinsame Teilsequenz

$$S_1 = ATTGCGAATGA$$
$$S_2 = ATGGCGTAAGA \qquad \Longrightarrow ATGCGAAGA \text{ ist lgT.}$$

Definition

Sei $X = (x_1, x_2, \ldots, x_m)$. $Z = (z_1, z_2, \ldots, z_k)$ heißt Teilsequenz von X, wenn es $i_1 < i_2 < \cdots < i_k$ gibt, sodass $x_{i_j} = z_j$.

Definition

Eine Teilsequenz von $X = (x_1, x_2, \ldots, x_m)$, die auch Teilsequenz von $Y = (y_1, y_2, \ldots, y_n)$ ist, heißt längste gemeinsame Teilsequenz von X und Y, falls die Anzahl ihrer Einträge maximal ist.

Bemerkung: Es gibt 2^m Teilsequenzen von $X = (x_1, x_2, \ldots, x_m)$.

Definition

Das i-te Präfix von $X = (x_1, x_2, \ldots, x_m)$ *ist definiert durch* $X_i = (x_1, x_2, \ldots, x_i)$

Satz

Seien $X = (x_1, x_2, \ldots, x_m)$ *und* $Y = (y_1, y_2, \ldots, y_n)$ *zwei Sequenzen, sei weiters* $Z = (z_1, z_2, \ldots, z_k)$ *lgT von* X *und* Y*. Dann gilt:*

1. *Ist* $x_m = y_n$*, so gilt* $z_k = x_m = y_n$ *und* $Z_{k-1} = (z_1, z_2, \ldots, z_{k-1})$ *ist lgT von* X_{m-1}, Y_{n-1}.
2. *Ist* $x_m \neq y_n$*, dann folgt aus* $z_k \neq x_m$*, dass* Z *eine lgT von* X_{m-1} *und* Y *ist.*
3. *Ist* $x_m \neq y_n$*, dann folgt aus* $z_k \neq y_n$*, dass* Z *eine lgT von* Y_{n-1} *und* X *ist.*

Beweis:

1. Sei $x_m = y_n$. Wenn $z_k \neq x_m$, dann könnte man $x_m = y_n$ an Z anhängen und erhielte eine gemeinsame Teilsequenz von X und Y der Länge $k + 1$ ↯

   Noch zu zeigen: Z_{k-1} ist lgT von X_{m-1} und Y_{n-1}.
   Sei W eine lgT von X_{m-1} und Y_{n-1} der Länge k. Dann kann wiederum $x_m = y_n$ an W angehängt werden $\rightsquigarrow$ Teilsequenz von X und Y der Länge $k + 1$ ↯
2. Sei $x_m \neq y_n$ und $z_k \neq x_m$.
   Das ist Z eine gemeinsame Teilsequenz von X_{m-1} und Y.
   Gäbe es eine gemeinsame Teilsequenz W von X_{m-1} und Y, die länger als Z ist, dann wäre W auch eine gemeinsame Teilsequenz von X und Y ↯
3. Sei $x_m \neq y_n$ und $z_k \neq y_n$.
   Dieser Fall ist symmetrisch zu Fall 2. $\square$

Dynamische Programmierung

Buchhaltung mittels Matrizen B und C:
$C[i,j]$ sei die Länge der lgT von X_i und Y_j

$$C[i,j] = \begin{cases} 0, & \text{wenn } i = 0 \text{ oder } j = 0 \\ C[i-1,j-1]+1, & \text{wenn } x_i = y_j \\ \max(C[i-1,j], C[i,j-1]), & \text{wenn } x_i \neq y_j. \end{cases}$$

$B[i,j]$ ist ein Pfeil, der uns hilft in C zu navigieren, um die optimale Lösung zu konstruieren.

Dynamische Programmierung

Algorithm lgT(X, Y)

```
m := |X|
n := |Y|
Seien B[1..m, 1..n] und C[0..m, 0..n] Datenfelder
for i = 1 to m do
    C[i, 0] := 0
end for
for j = 0 to n do
    C[0, j] := 0
end for
for i = 1 to m do
    for j = 1 to n do
        if x_i = y_j then
            C[i, j] := C[i − 1, j − 1] + 1
            B[i, j] := „↖“
        else if C[i − 1, j] ≥ C[i, j − 1] then
            C[i, j] := C[i − 1, j]
            B[i, j] := „↑“
        else
            C[i, j] := C[i, j − 1]
            B[i, j] := „←“
        end if
    end for
end for
```

Dynamische Programmierung

Beispiel: $X = [\text{A,B,C,B,D,A,B}]$, $Y = [\text{B,D,C,A,B,A}]$.

i \ j	y_j \ x_i	0	1 B	2 D	3 C	4 A	5 B	6 A
0		0	0	0	0	0	0	0
1	A	0	0↑	0↑	0↑	1↖	1←	1↖
2	B	0	1↖	1←	1←	1↑	2↖	2←
3	C	0	1↑	1↑	2↖	2←	2↑	2↑
4	B	0	1↖	1←	2↑	2↑	3↖	3←
5	D	0	1↑	2↖	2↑	2↑	3↑	3↑
6	A	0	1↑	2↑	2↑	3↖	3↑	4↖
7	B	0	1↖	2↑	2↑	3↑	4↖	4↑

Somit ist [B,C,B,A] eine lgT.

Aufwand: $\mathcal{O}(mn)$.

Dynamische Programmierung

Algorithm Print-lgT(B, X, Y, i, j)

```
if i = 0 or j = 0 then
    return
end if
if B[i, j] =„↖“ then
    Print-lgT(B, X, Y, i − 1, j − 1)
    print x_i
else if B[i, j] =„↑“ then
    Print-lgT(B, X, Y, i − 1, j)
else
    Print-lgT(B, X, Y, i, j − 1)
end if
```

21) Geometrische Algorithmen

Algorithmische Geometrie in der Ebene ($\mathbb{R}^2$),

Eingabe: Punktmenge Q, Punkte $p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $x_i, y_i \in \mathbb{R}$.

Eigenschaften von Strecken

$p_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$, $p_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$, $p_3 = \alpha p_1 + (1-\alpha)p_2,\ 0 \le \alpha \le 1,$
konvexe Linearkombination.
Strecke: $\overline{p_1 p_2} = \{\alpha p_1 + (1-\alpha)p_2 \mid 0 \le \alpha \le 1\}$, $\overrightarrow{p_1 p_2}$ (gerichtet)

$$p_1 \otimes p_2 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \otimes p_1 = \pm \| p_1 \times p_2 \|:$$

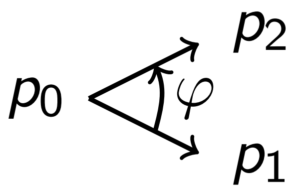
vorzeichenbehafteter Flächeninhalt des von den Strecken $\overline{\begin{pmatrix} 0 \\ 0 \end{pmatrix} p_1}$

und $\overline{\begin{pmatrix} 0 \\ 0 \end{pmatrix} p_2}$ aufgespannten Parallelogramms

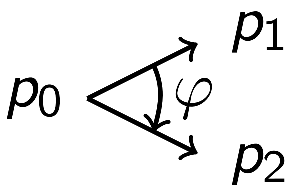
Lagebestimmung von $\overrightarrow{p_0 p_1}$ zu $\overrightarrow{p_0 p_2}$

Mögliche Lagen:

(I)



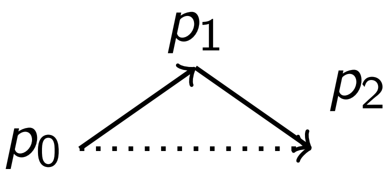
(II)



Wähle p_0 als Ursprung, dann gilt

$$(p_1 - p_0) \otimes (p_2 - p_0) \begin{cases} > 0 & \text{im Fall } (I), \\ < 0 & \text{im Fall } (II), \\ = 0 & \text{falls } \varphi = 0 \text{ oder } \varphi = \pi. \end{cases}$$

Aufeinanderfolgende Strecken: Links- oder Rechtskurve?

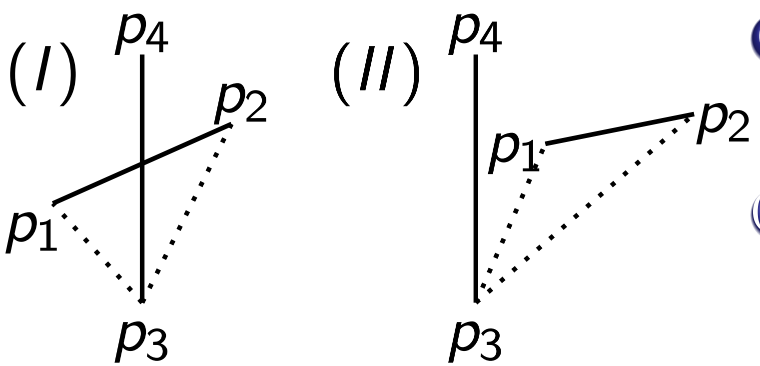


$\measuredangle p_1 p_0 p_2$

$$(p_1 - p_0) \otimes (p_2 - p_0) \begin{cases} > 0 & \text{im Fall einer Linkskurve,} \\ < 0 & \text{im Fall einer Rechtskurve,} \\ = 0, & \text{falls beide Strecken auf einer Linie.} \end{cases}$$

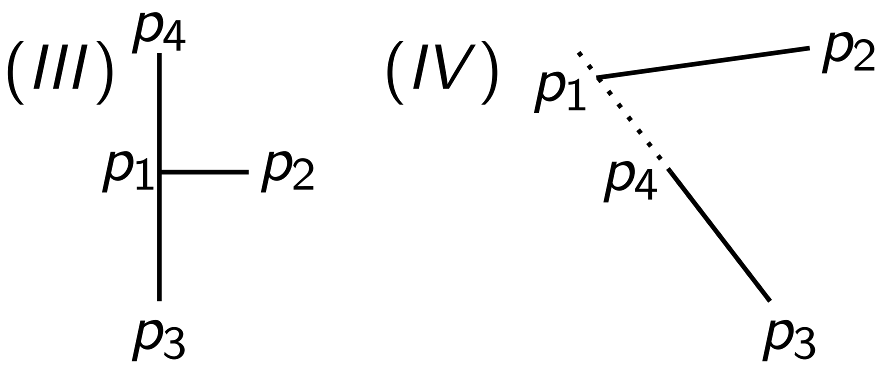
Geometrische Algorithmen

Wann schneiden sich die Strecken $\overline{p_1 p_2}$ und $\overline{p_3 p_4}$?



- **(I)** $0 \neq \operatorname{sgn}((p_4 - p_3) \otimes (p_2 - p_3))$
  $= -\operatorname{sgn}((p_4 - p_3) \otimes (p_1 - p_3))$
- **(II)** $0 \neq \operatorname{sgn}((p_4 - p_3) \otimes (p_2 - p_3))$
  $= \operatorname{sgn}((p_4 - p_3) \otimes (p_1 - p_3))$

  (negativ: $\overline{p_1 p_2}$ rechts von $\overrightarrow{p_3 p_4}$)



(III), (IV)

$\operatorname{sgn}((p_4 - p_3) \otimes (p_1 - p_3)) = 0$

(III) $\Longleftrightarrow$

$\min(x_3, x_4) \leq x_1 \leq \max(x_3, x_4)$

und

$\min(y_3, y_4) \leq y_1 \leq \max(y_3, y_4)$

Algorithm DIRECTION(p, q, r)

return $(r - p) \otimes (q - p)$

Algorithm ON-SEGMENT(p, q, r)

if $\min(x_p, x_q) \le x_r \le \max(x_p, x_q) \wedge \min(y_p, y_q) \le y_r \le \max(y_p, y_q)$ **then**
    return TRUE
else return FALSE
end if

Algorithm SEGMENT-INTERSECT(p_1, p_2, p_3, p_4)

$d_1 :=$DIRECTION(p_3, p_4, p_1)
$d_2 :=$DIRECTION(p_3, p_4, p_2)
$d_3 :=$DIRECTION(p_1, p_2, p_3)
$d_4 :=$DIRECTION(p_1, p_2, p_4)
if $d_1 d_2 < 0 \wedge d_3 d_4 < 0$ **then return** TRUE
else if $d_1 = 0 \wedge$ON-SEGMENT(p_3, p_4, p_1) **then return** TRUE
else if $d_2 = 0 \wedge$ON-SEGMENT(p_3, p_4, p_2) **then return** TRUE
else if $d_3 = 0 \wedge$ON-SEGMENT(p_1, p_2, p_3) **then return** TRUE
else if $d_4 = 0 \wedge$ON-SEGMENT(p_1, p_2, p_4) **then return** TRUE
else return FALSE
end if

Bestimmen der konvexen Hülle

Gegeben: Punktmenge Q

Gesucht: Konvexe Hülle $[Q]$

Es genügt, konvexes Polygon $P = \{p_0, p_1, \ldots, p_n\}$ zu bestimmen, sodass für alle $q \in Q$ Skalare $\lambda_1, \ldots, \lambda_n$ existieren mit

$$q = \sum_{i=1}^{n} \lambda_i p_i, \text{ und } \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1.$$

Voraussetzungen:

- *Es gibt 3 Punkte $q_1, q_2, q_3 \in Q$, die nicht kollinear sind.*
- *Die Menge P sei minimal, d.h. P besteht genau aus den Extremalpunkten von $[Q]$.*

  Daraus folgt insbesondere $P \subseteq Q$.

Grahamsches Scannen

verwaltet Stack S aus Kandidaten x für P

- Alle $x \in Q$ kommen im Laufe des Algorithmus auf den Stack.
- Alle $x \in Q \setminus P$ werden schließlich entfernt und kommen nie wieder auf den Stack
- Wenn der Algorithmus terminiert, dann ist S „=“ P, wenn von unten nach oben gelesen, geordnet in mathematisch positiver Richtung; i.Z. $S \sim P$.

Hilfsfunktionen:

- TOP(S): Ausgabe des obersten Elements von S;
- NEXT-TO-TOP(S): Ausgabe des zweitobersten Elements von S;
- POP(S): Entfernen des obersten Elements;
- PUSH(S, x): x auf den Stack legen.

Algorithm GRAHAM-SCAN(Q)

$p_0 := q \in Q$; Punkt in Q mit kleinster y-Koordinate, von all diesen jener mit kleinster x-Koordinate.

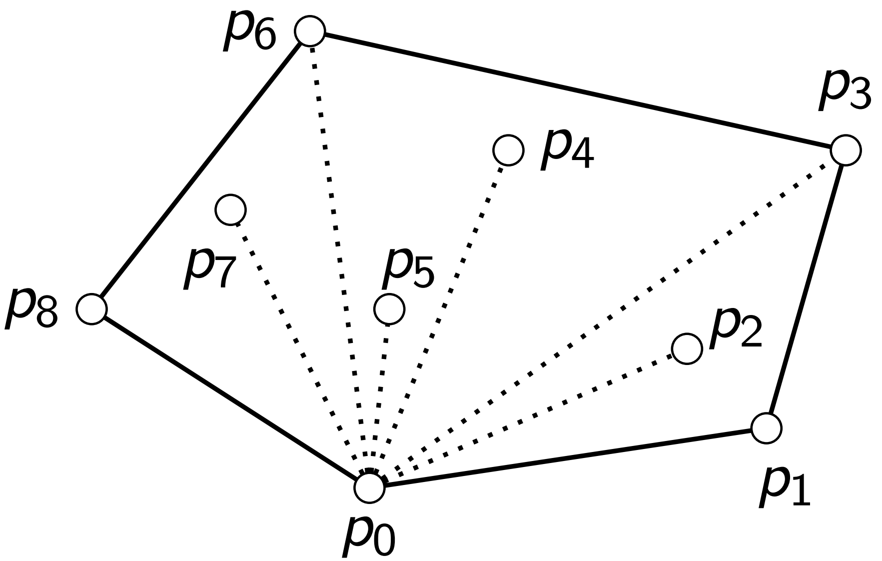
$(p_1, p_2, \ldots, p_m) :=$ Punkte von $Q \setminus \{p_0\}$, geordnet nach Polarwinkel $\varphi(p_i)$, wobei bei mehreren Punkten mit gleichem Polarwinkel nur der mit maximalem Abstand zu p_0 genommen wird.

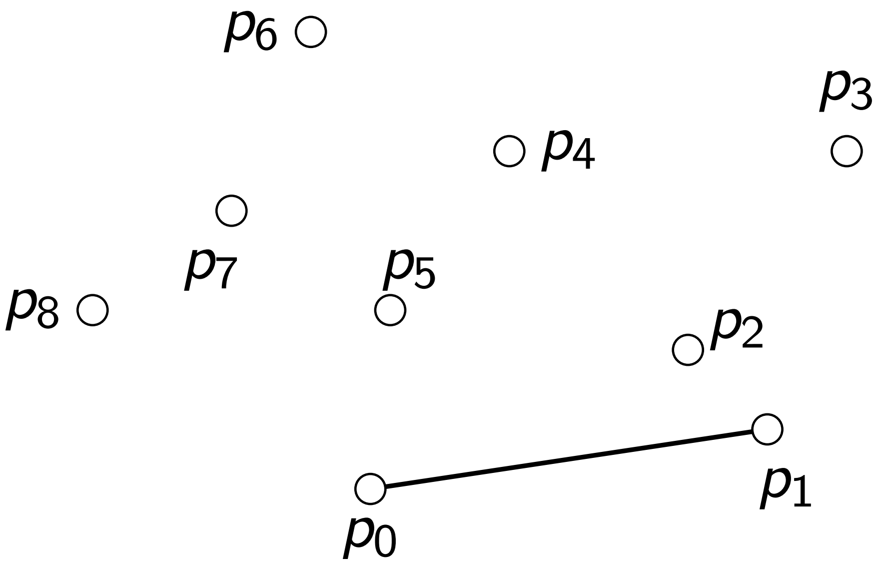
if $m \leq 2$ **then**
    return Q
else
    $S := [\,]$ % leerer Stack
    PUSH(S, p_0); PUSH(S, p_1); PUSH(S, p_2)
    for $i = 3$ **to** m **do**
        while NEXT-TO-TOP(S) $\rightarrow$TOP(S) $\rightarrow p_i$ keine Linkskurve **do**
            POP(S)
        end while
        PUSH(S, p_i)
    end for
    return S
end if

Bemerkung: Der Polarwinkel von p_i bezüglich p_0 ist der Winkel, den die Trägergerade von $\overrightarrow{p_0 p_i}$ mit der x-Achse einschließt.

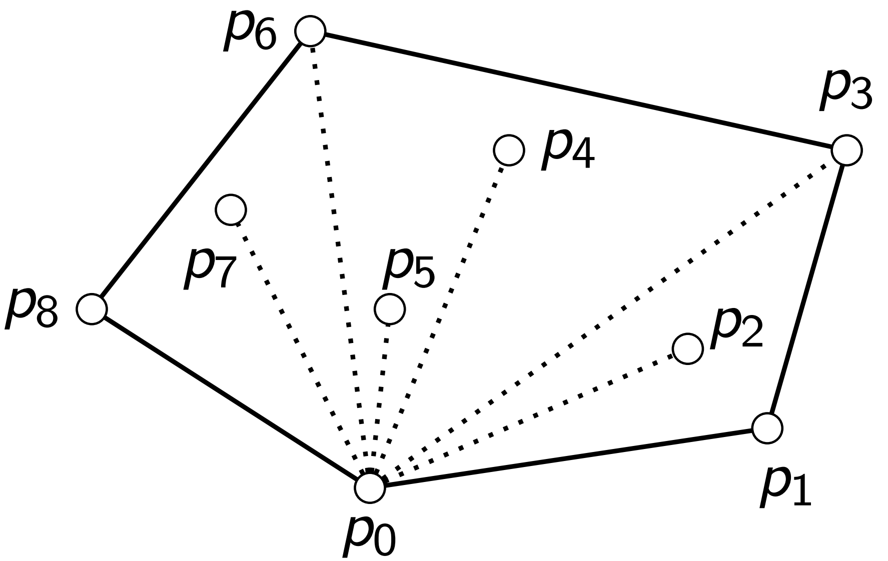
Laufzeit: Sei $n = |Q|$.
Sortieren nach $\varphi(p_i)$: $\mathcal{O}(n \log n)$
PUSH,POP, etc.: $\mathcal{O}(1)$
for-Schleife: $m - 2$ mal PUSH, höchstens $m - 2$ mal POP,
also $\mathcal{O}(n)$.
Insgesamt also $\mathcal{O}(n \log n)$.

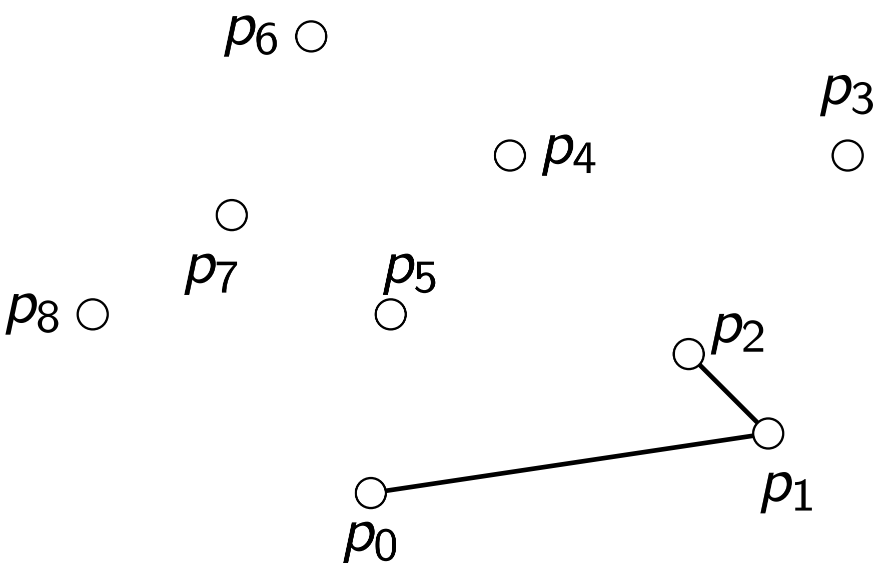
Beispiel





Beispiel

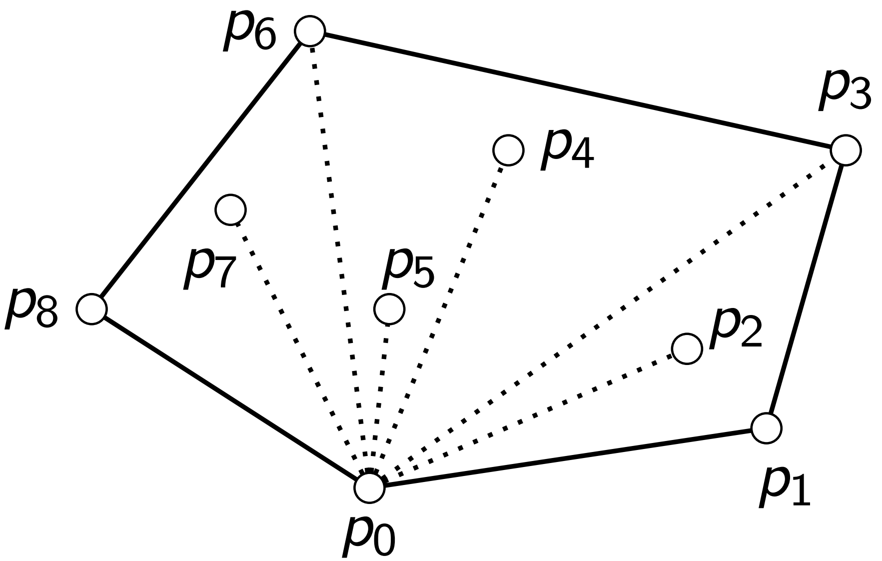


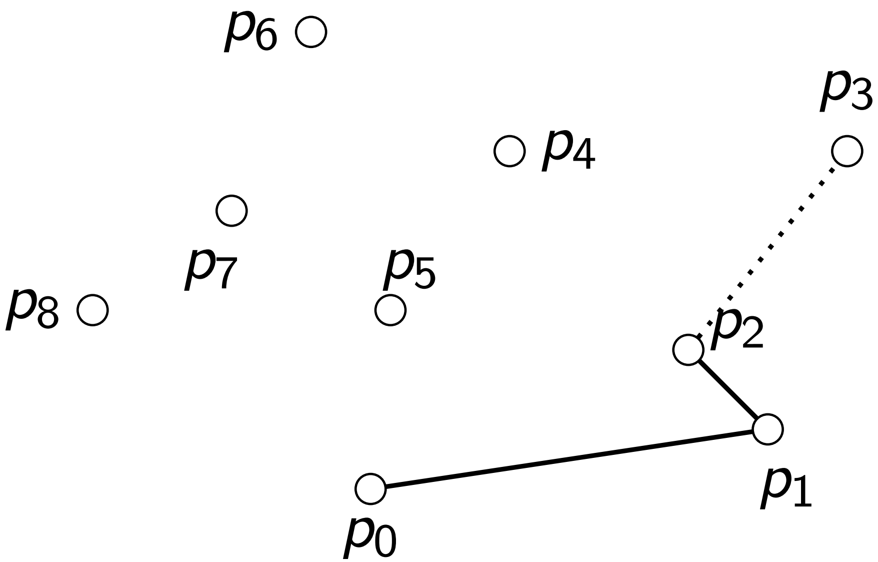


$$S = \begin{bmatrix} p_2 \\ p_1 \\ p_0 \end{bmatrix}$$

Geometrische Algorithmen

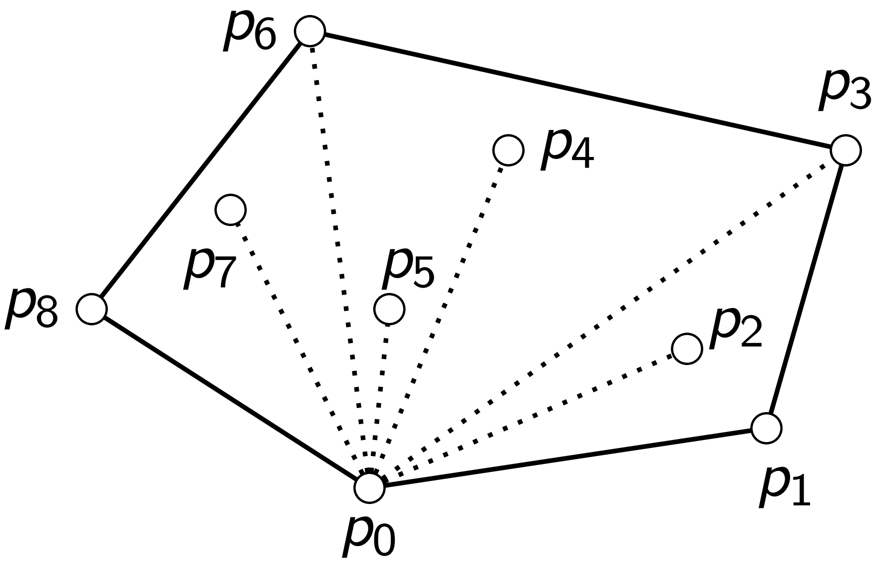
Beispiel

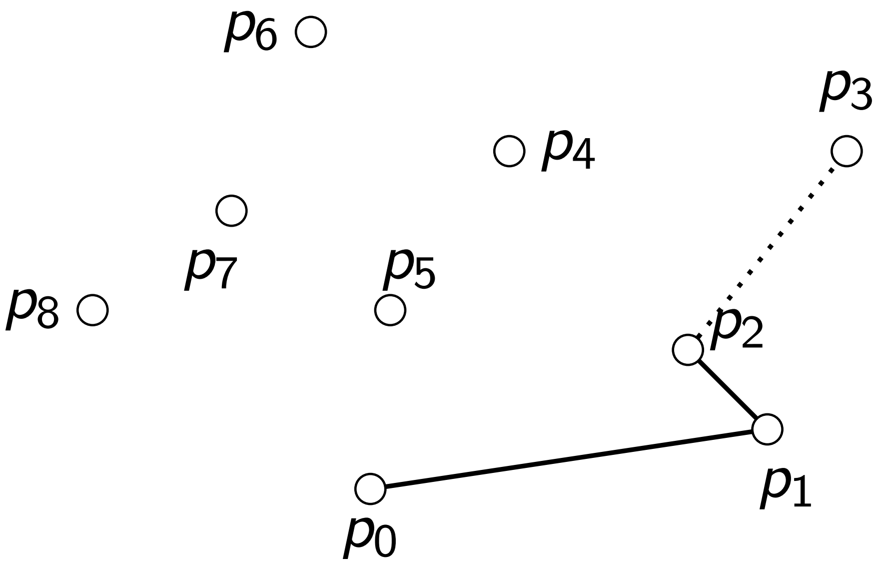




$$S = \begin{bmatrix} p_2 \\ p_1 \\ p_0 \end{bmatrix}$$

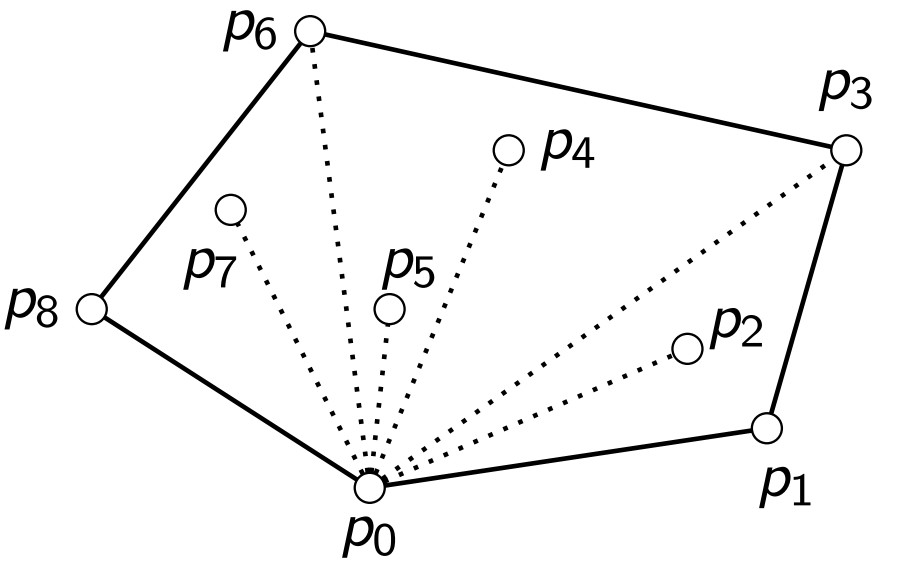
Beispiel

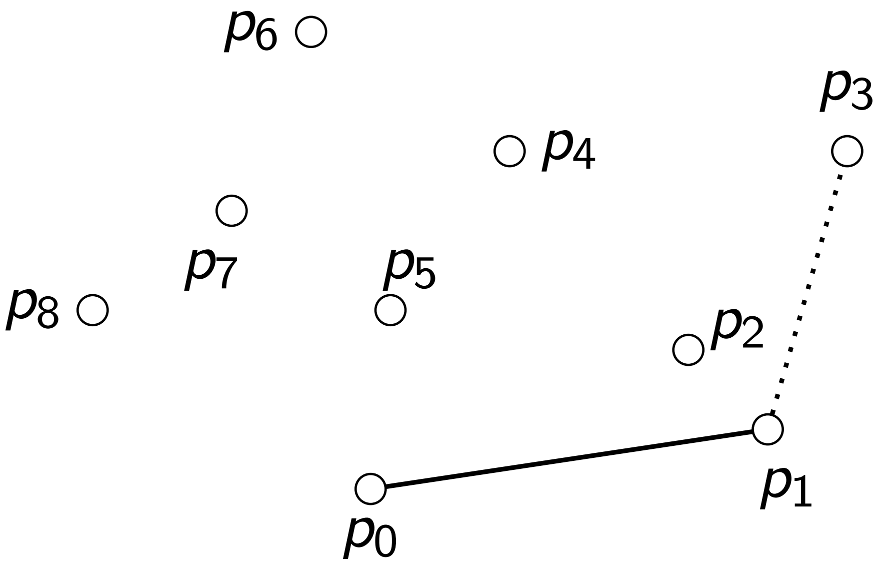




$$S = \begin{bmatrix} p_1 \\ p_0 \end{bmatrix}$$

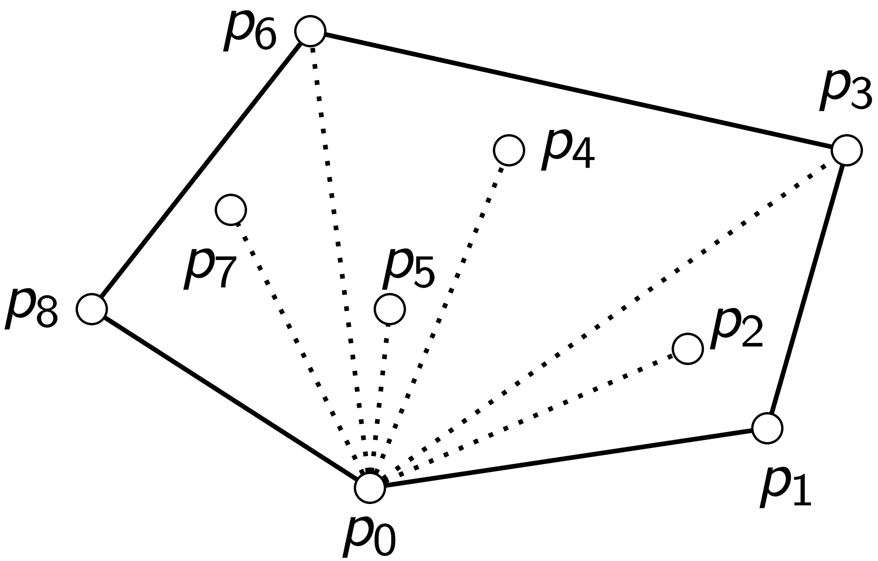
Beispiel

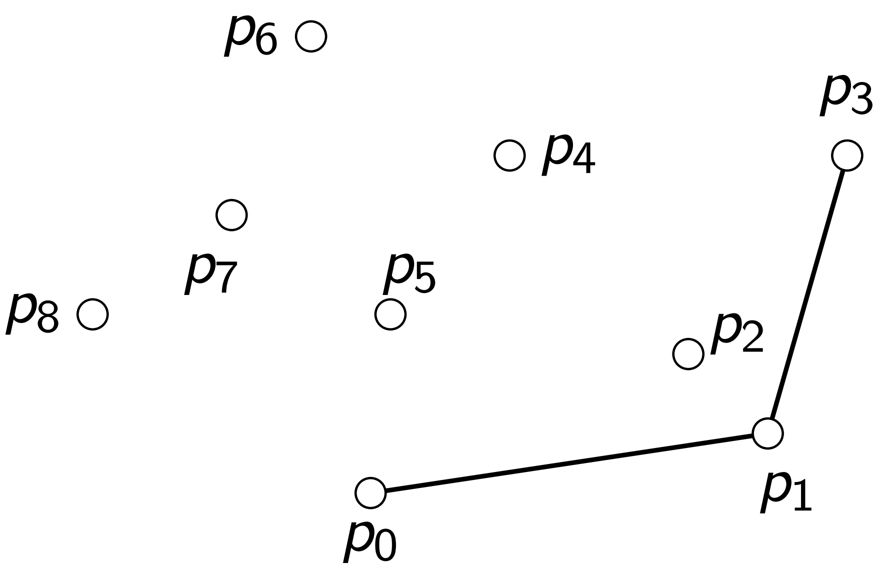




$$S = \begin{bmatrix} p_1 \\ p_0 \end{bmatrix}$$

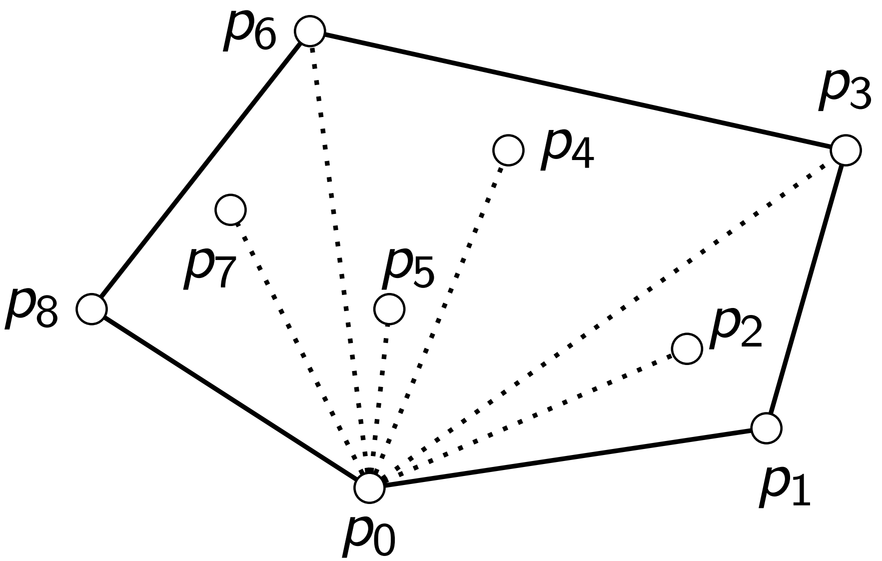
Beispiel

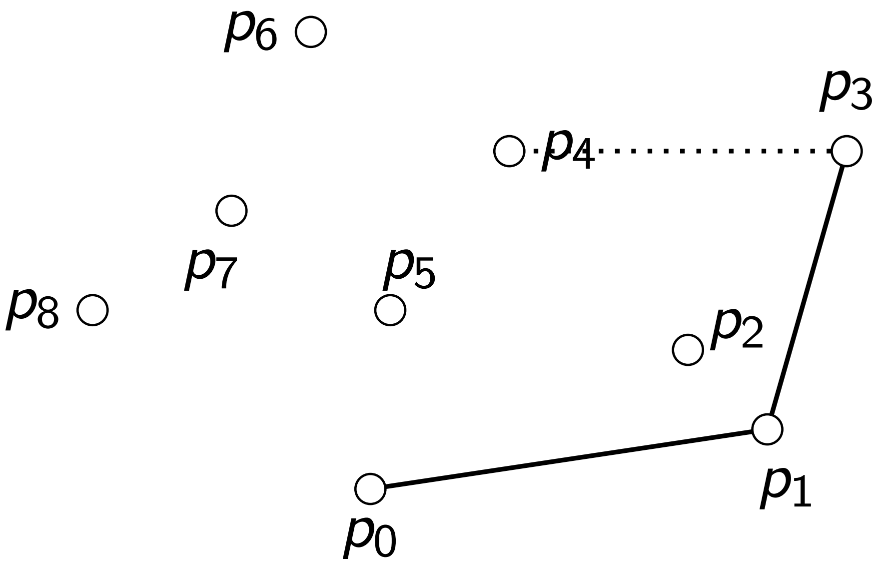




$$S = \begin{bmatrix} p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

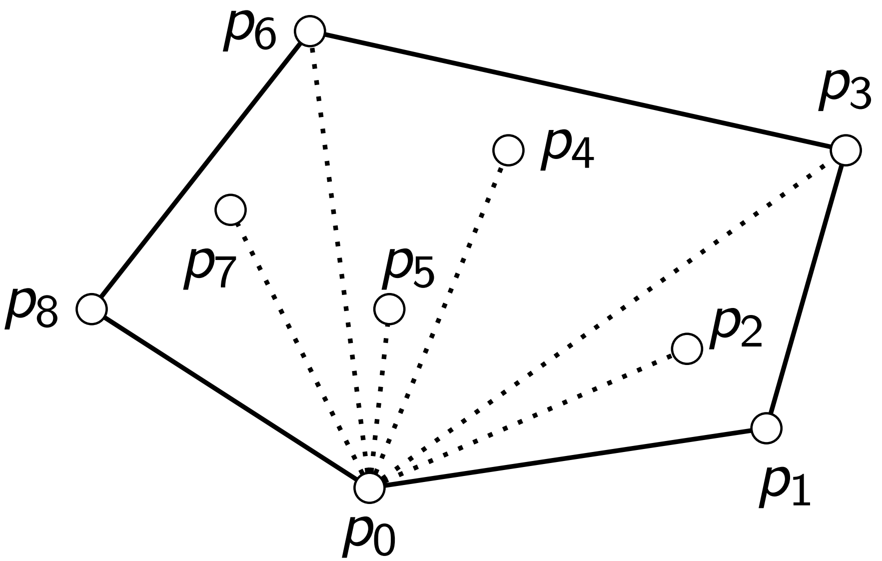
Beispiel

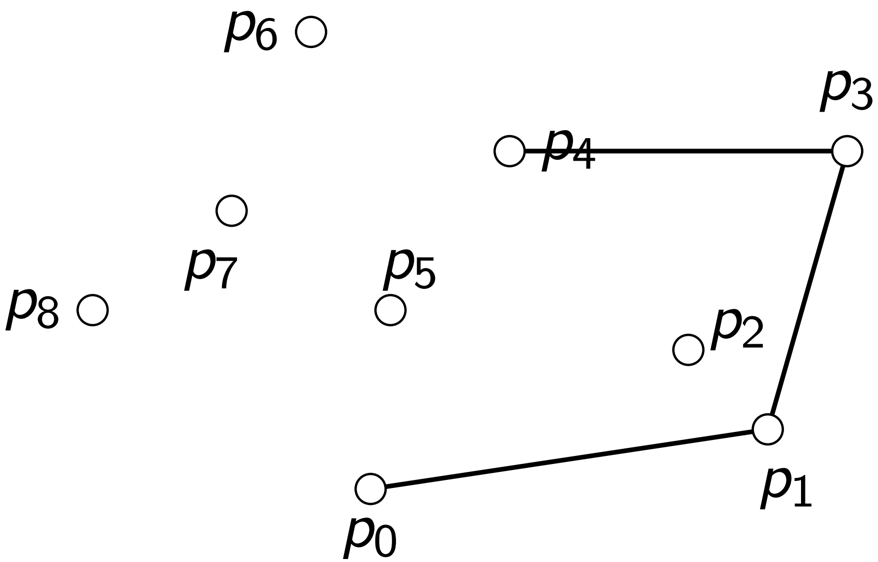




$$S = \begin{bmatrix} p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

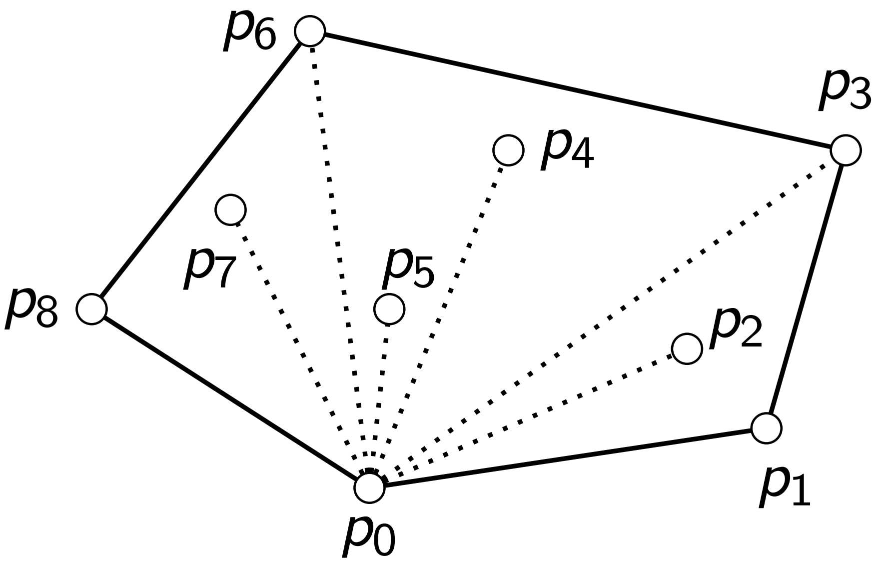
Beispiel

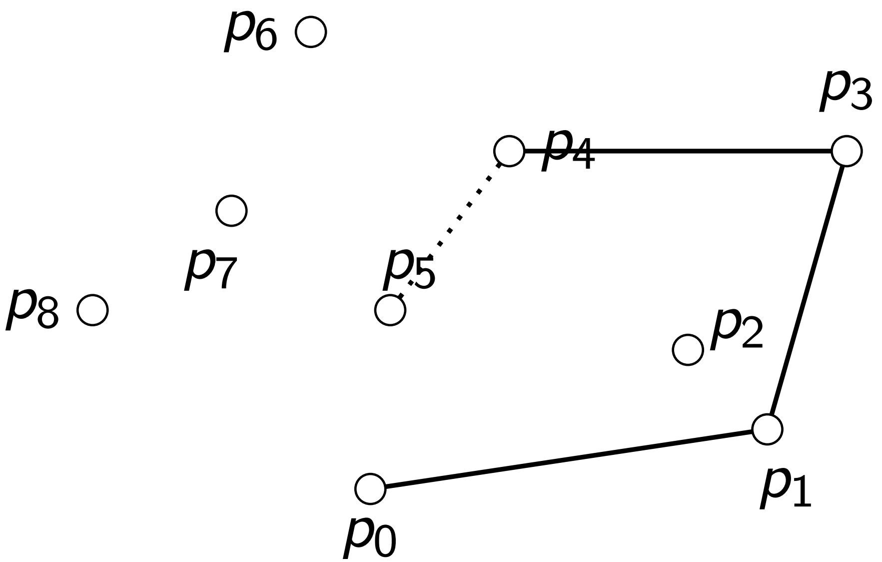




$$S = \begin{bmatrix} p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

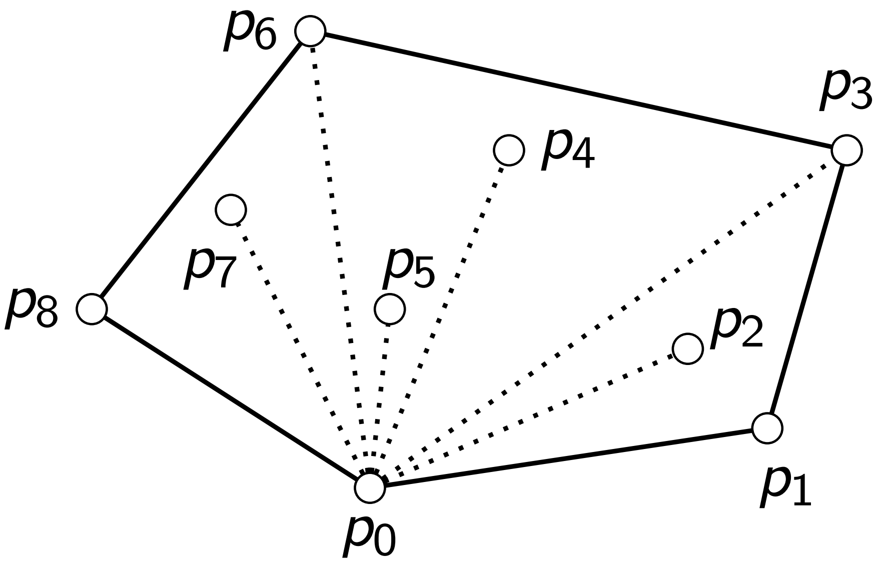
Beispiel

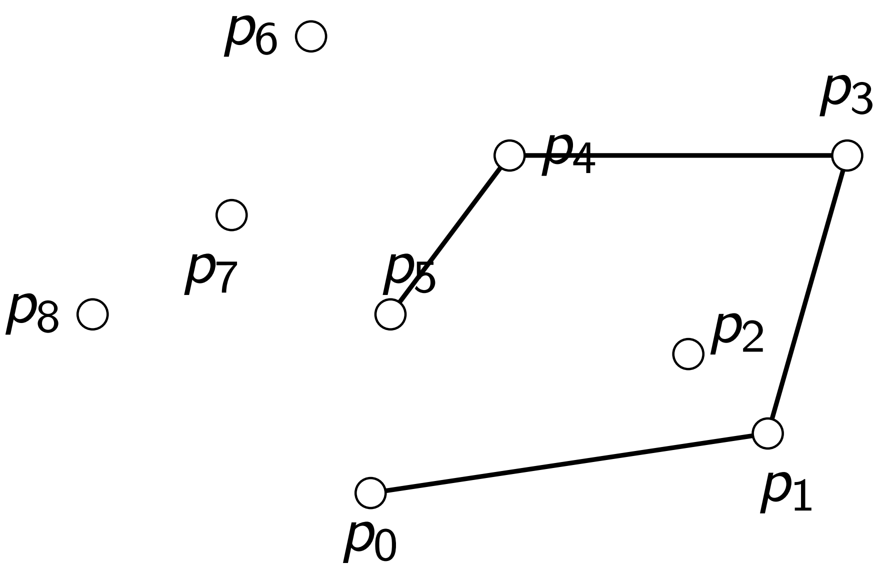




$$S = \begin{bmatrix} p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

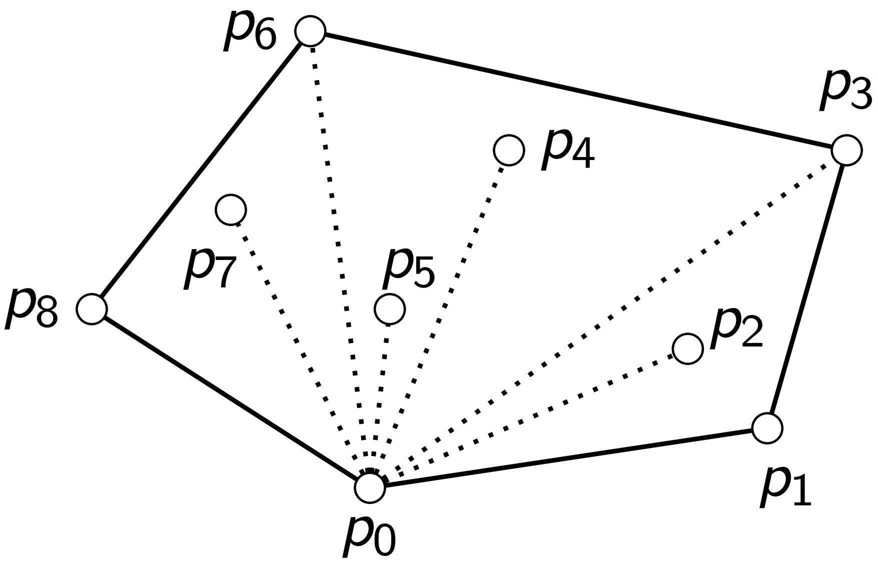
Beispiel

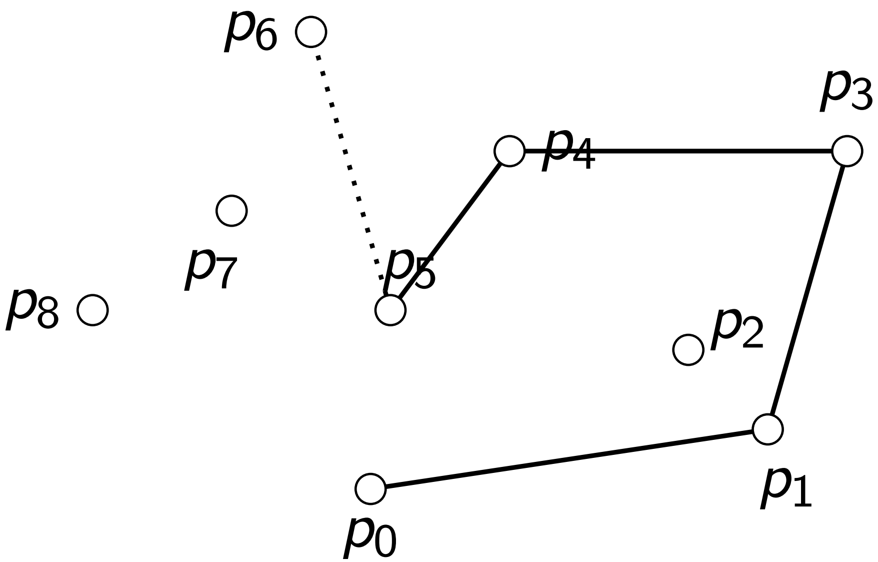




$$S = \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

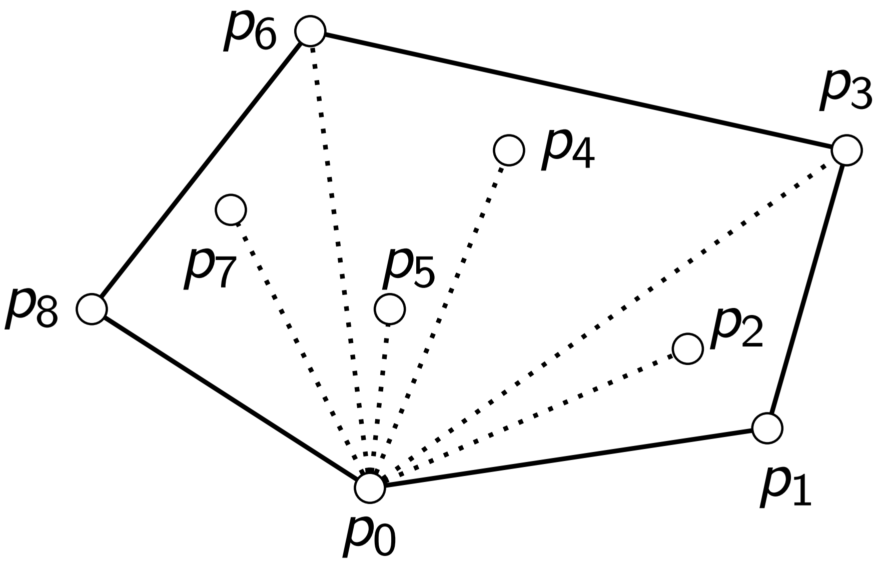
Beispiel

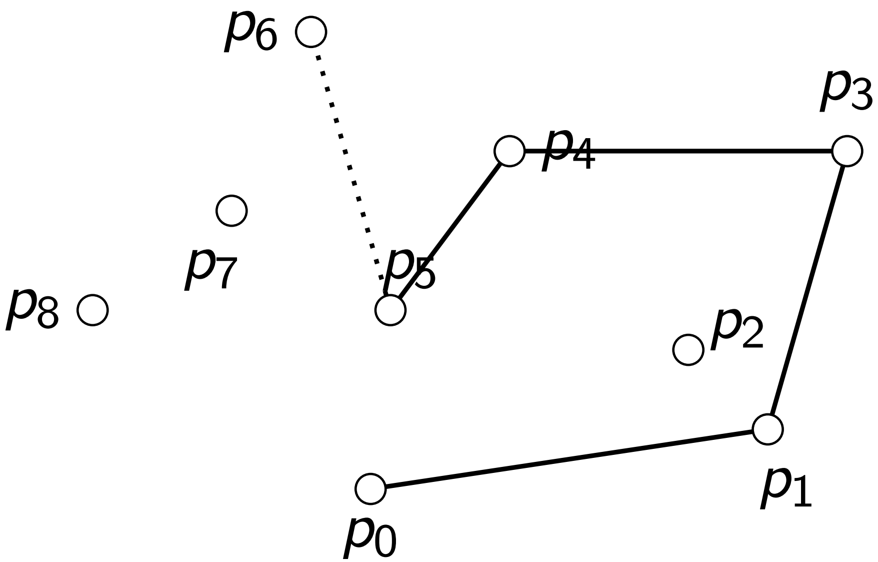




$$S = \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

Beispiel

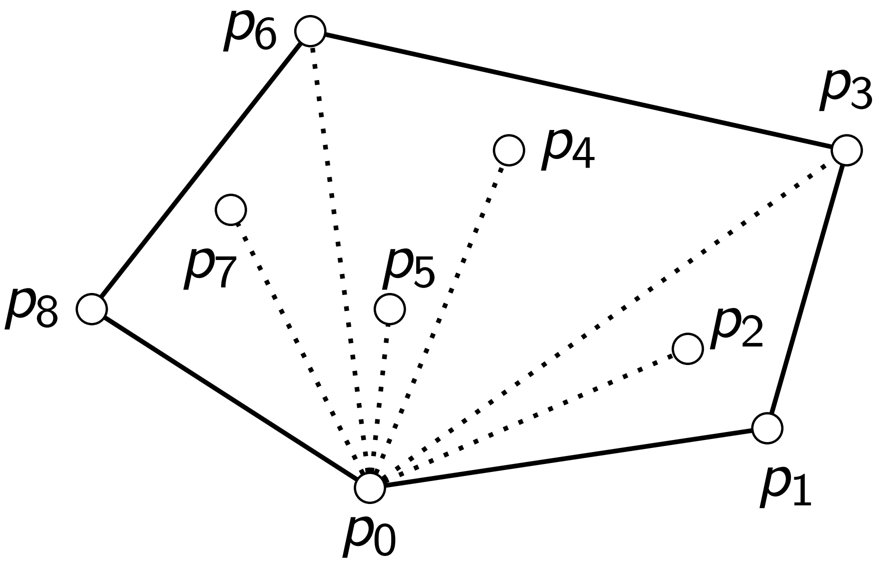


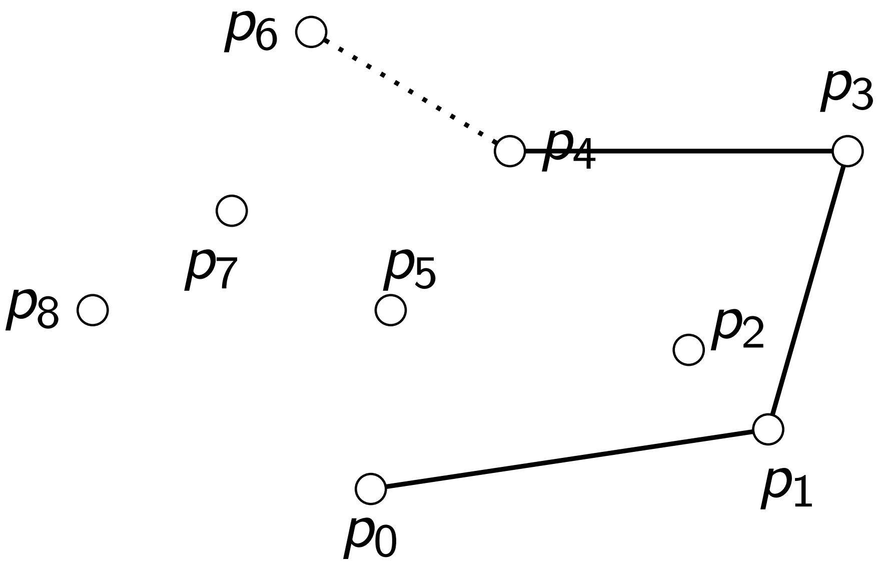


$$S = \begin{bmatrix} p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

Geometrische Algorithmen

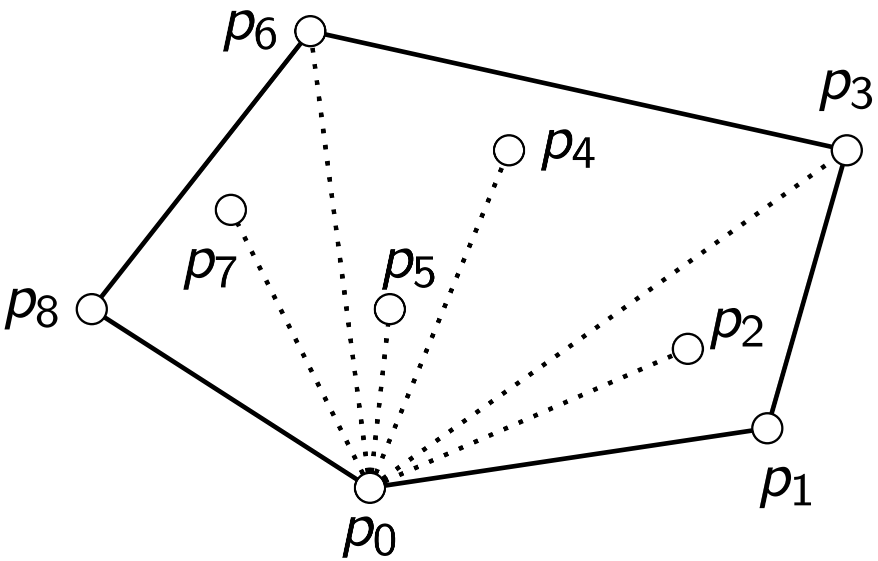
Beispiel

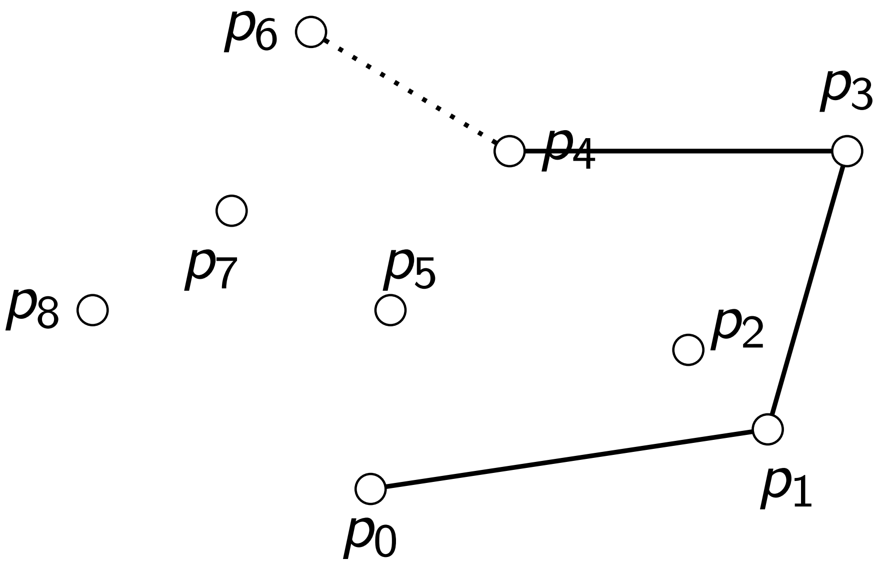




$$S = \begin{bmatrix} p_4 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

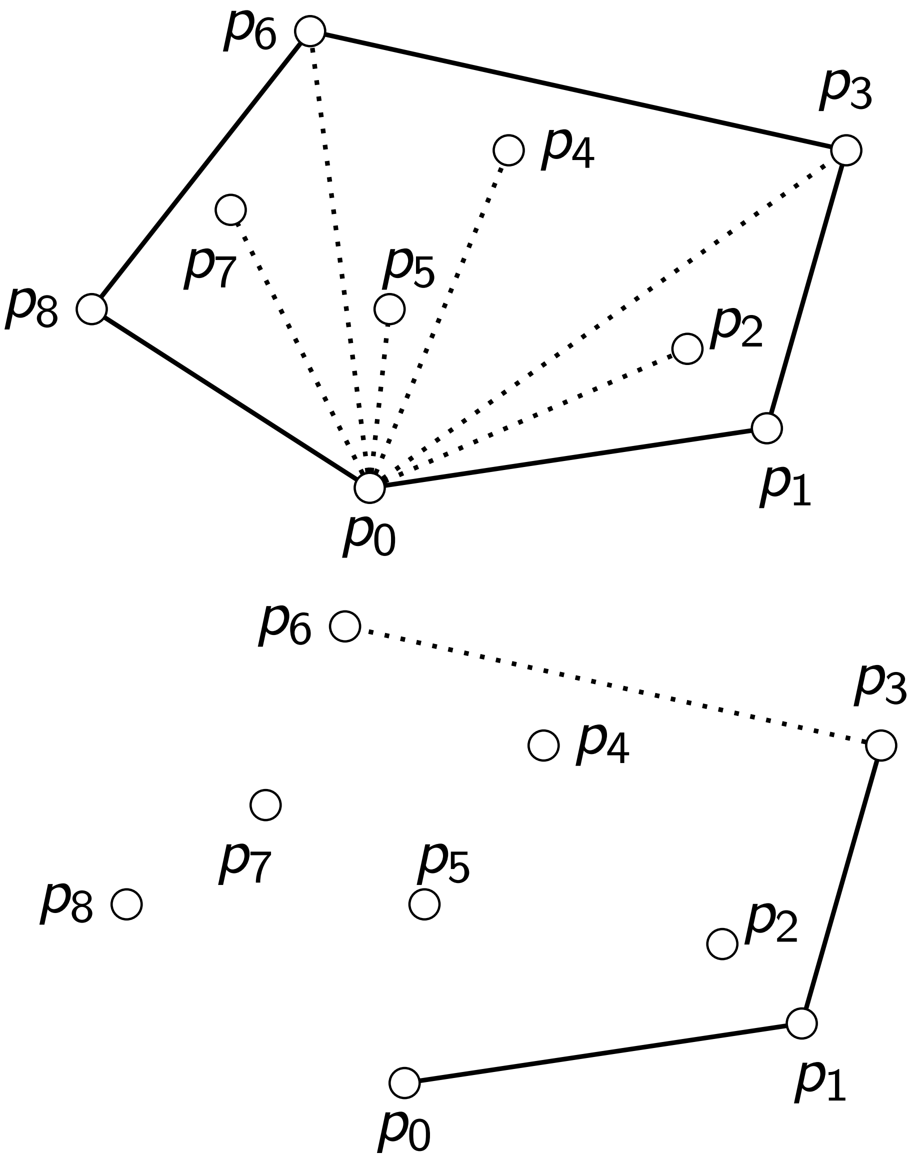
Beispiel





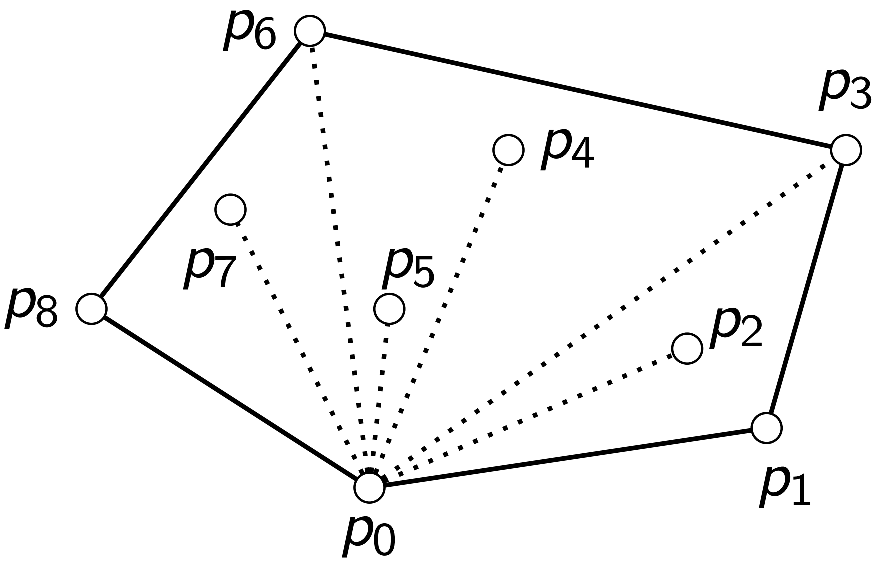
$$S = \begin{bmatrix} p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

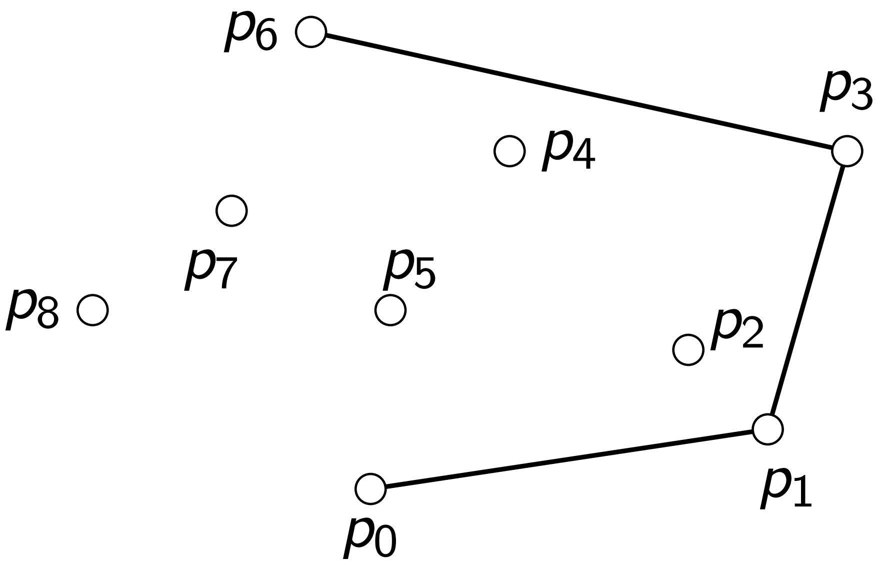
Beispiel



$$S = \begin{bmatrix} p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

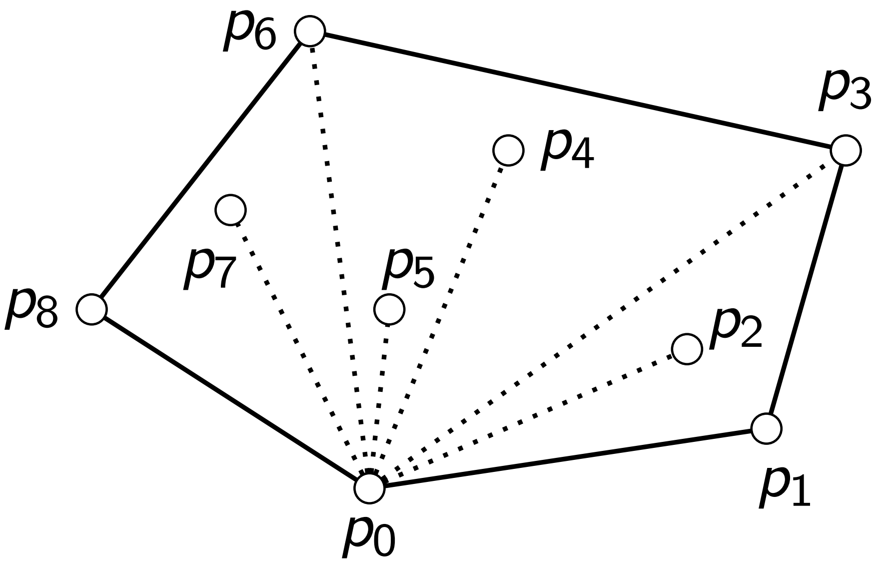
Beispiel

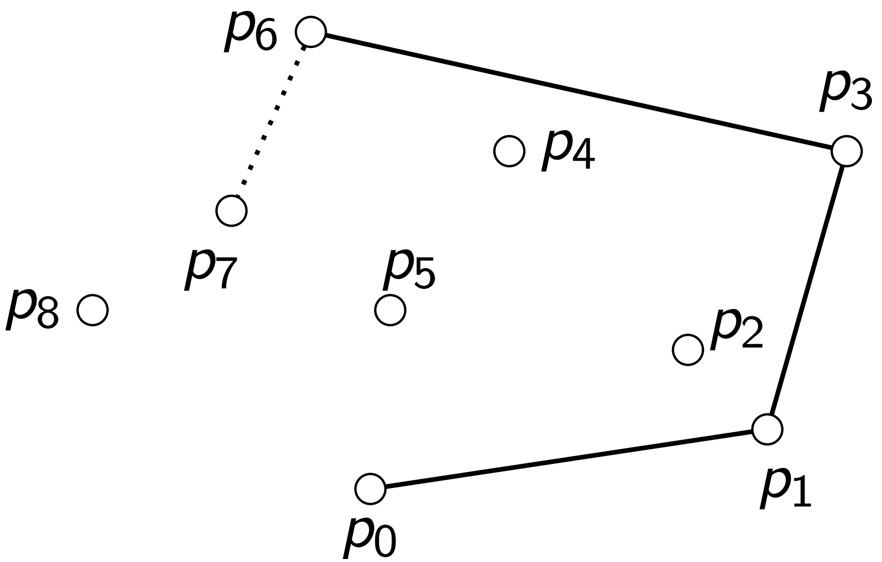




$$S = \begin{bmatrix} p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

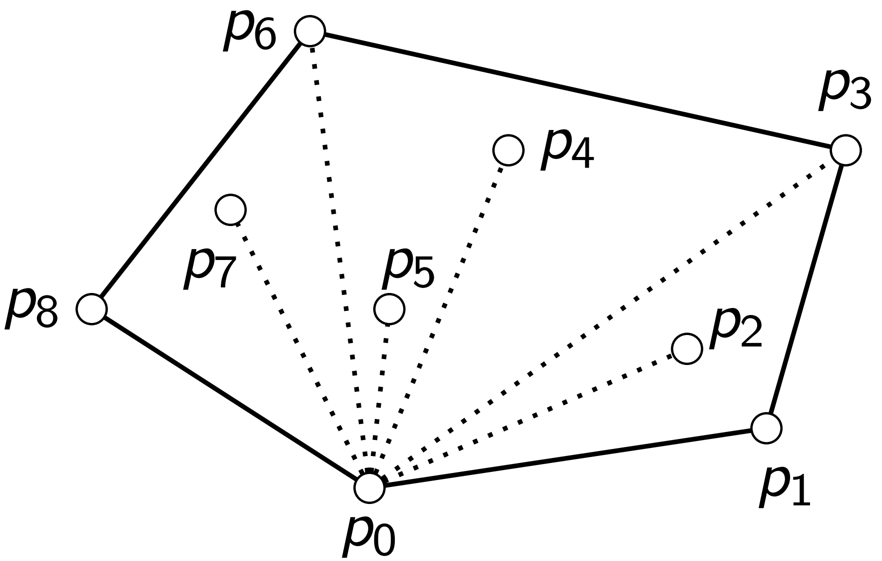
Beispiel

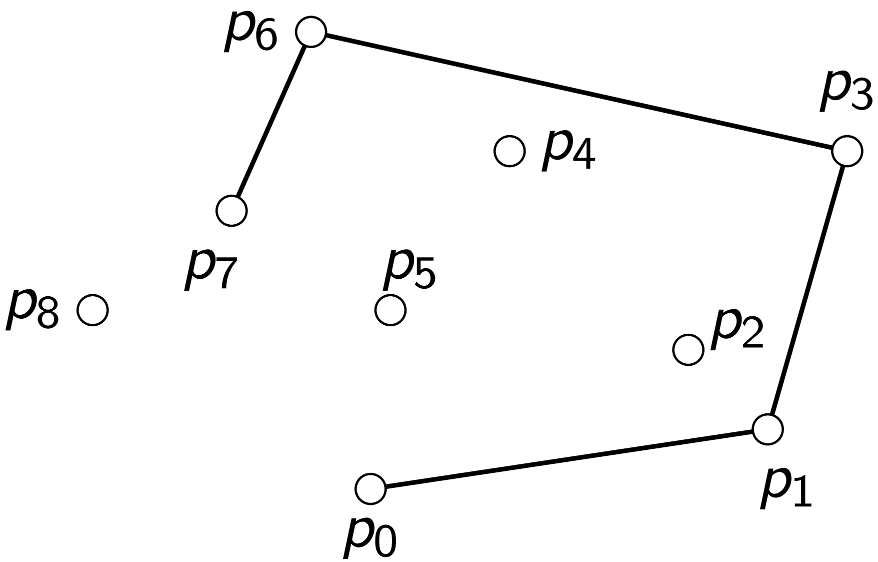




$$S = \begin{bmatrix} p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

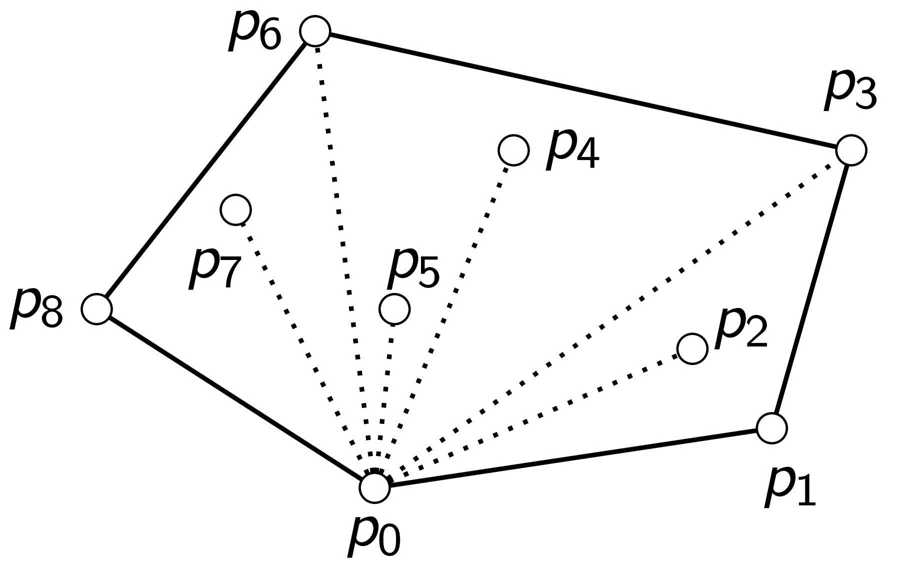
Beispiel

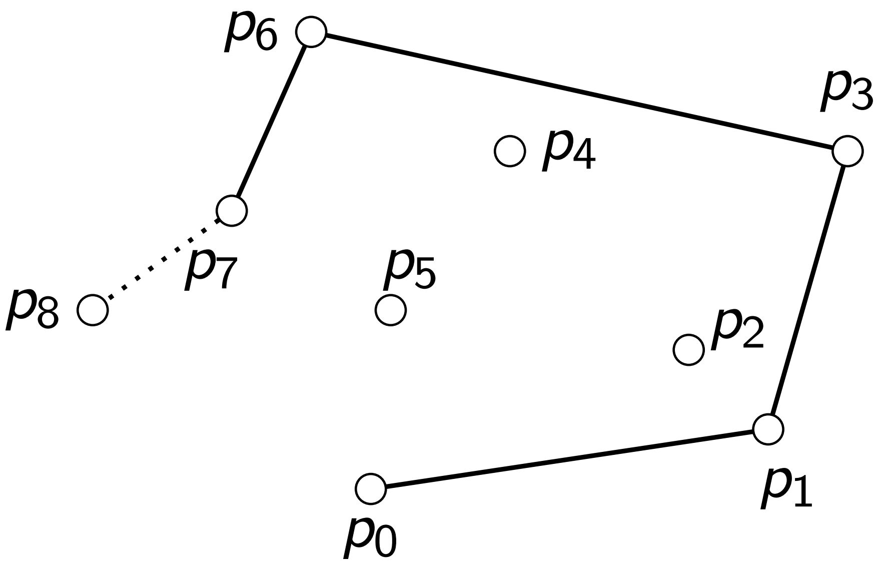




$$S = \begin{bmatrix} p_7 \\ p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

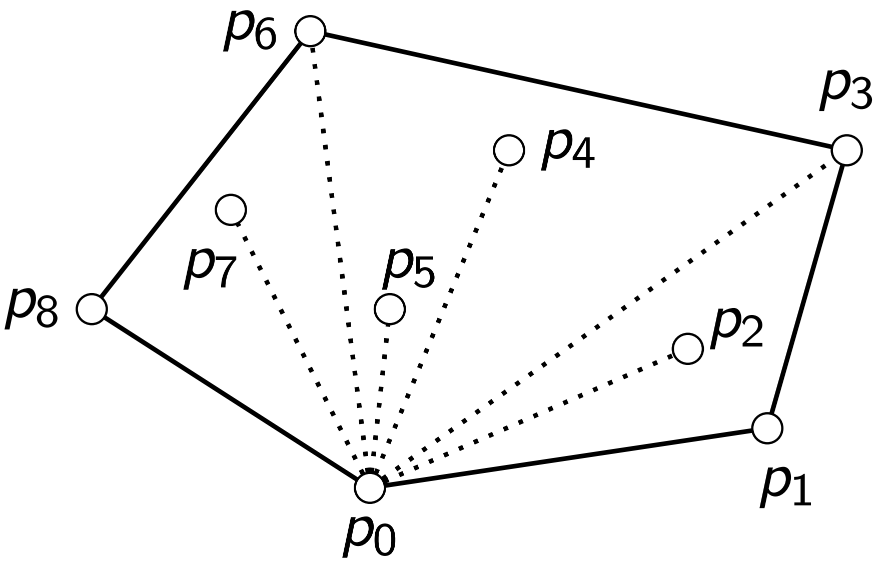
Beispiel

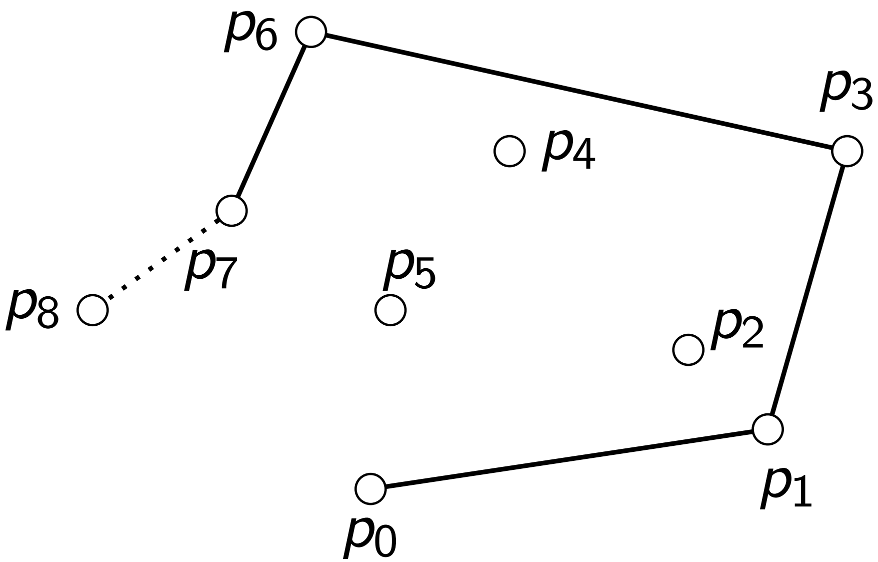




$$S = \begin{bmatrix} p_7 \\ p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

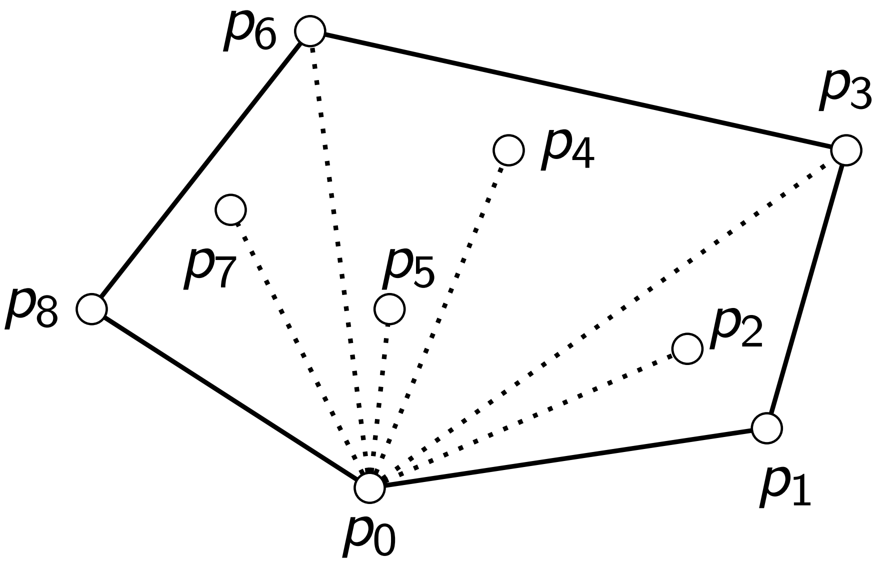
Beispiel

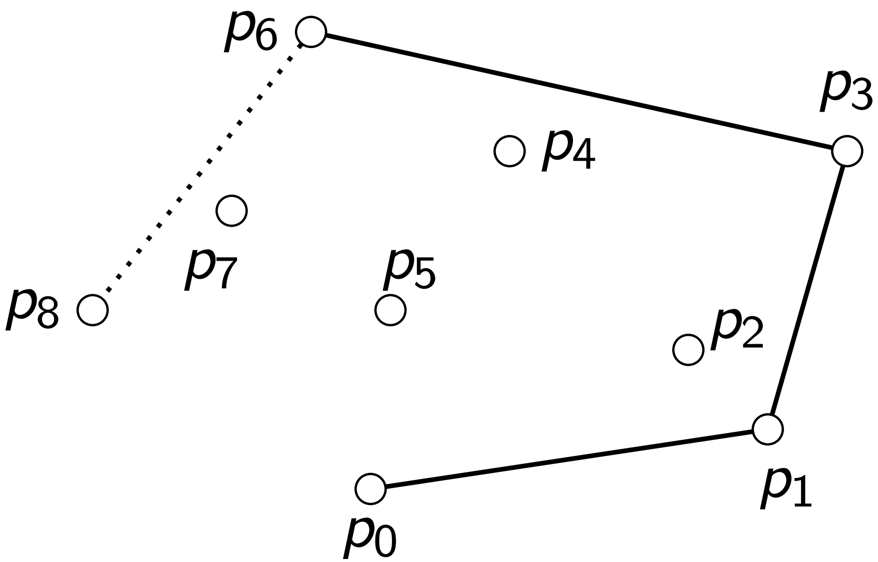




$$S = \begin{bmatrix} p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

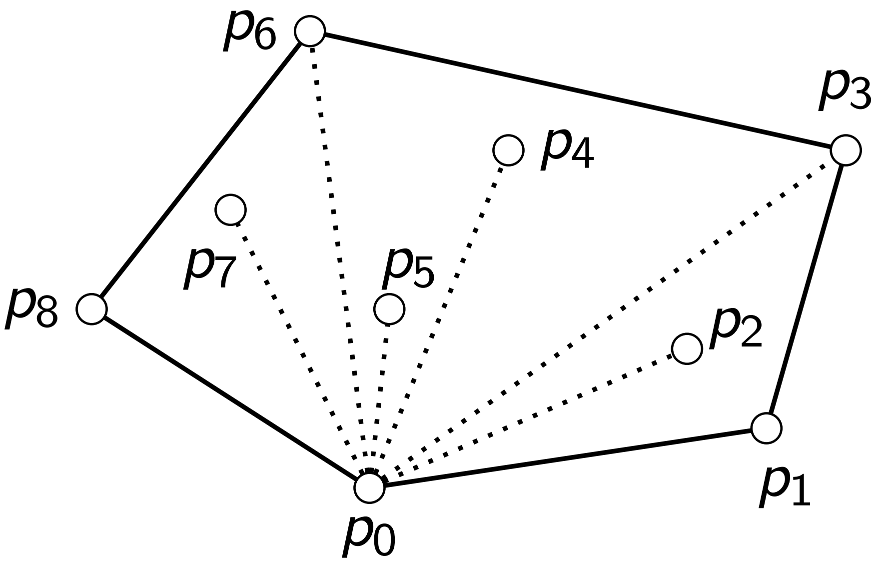
Beispiel

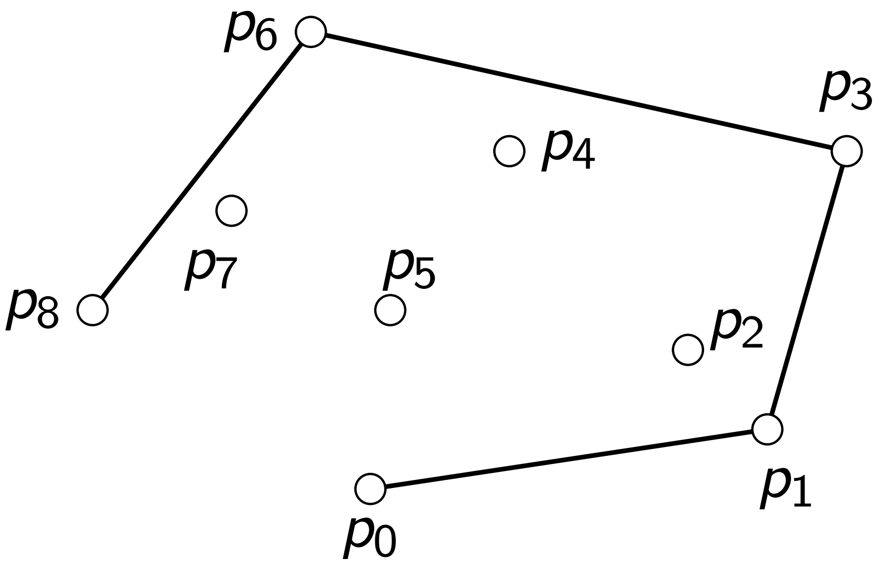




$$S = \begin{bmatrix} p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

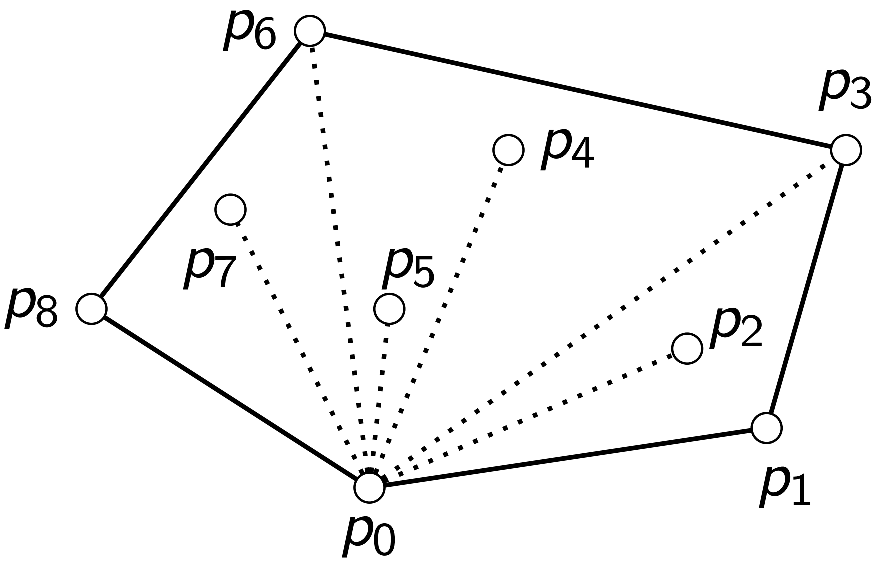
Beispiel

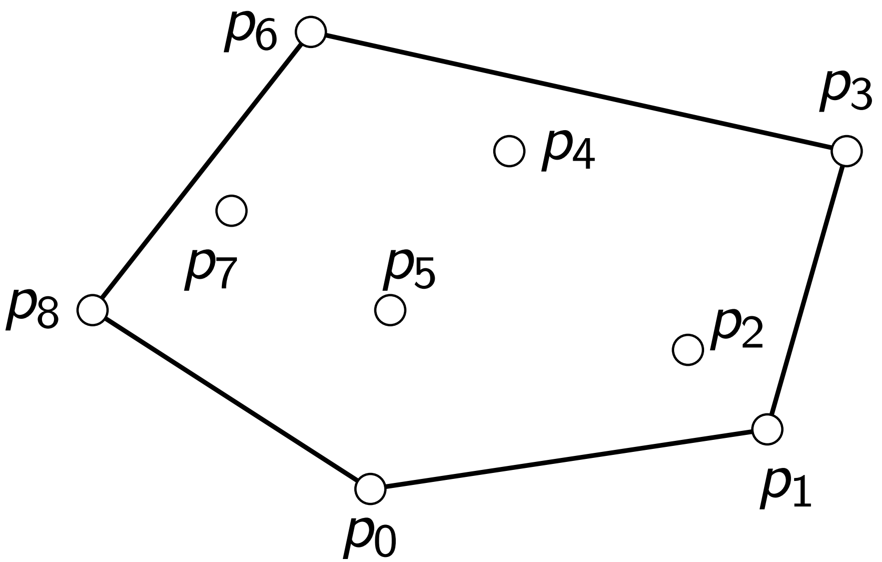




$$S = \begin{bmatrix} p_8 \\ p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

Beispiel





$$S = \begin{bmatrix} p_8 \\ p_6 \\ p_3 \\ p_1 \\ p_0 \end{bmatrix}$$

Satz

Das Grahamsche Scannen, angewendet auf eine Punktmenge Q, liefert schließlich einen Stack S mit $S = (e_0, \ldots, e_\ell)$, sodass $e_0, \ldots, e_\ell$ die Extremalpunkte von $[Q]$ in mathematisch positiver Richtung sind. Wir notieren das als $S \sim E$ mit $E = \{e_0, \ldots, e_\ell\}$.

Beweis:
Sei $\{p_0, p_1, \ldots, p_m\}$ die in Schritt 1 u 2 konstruierte Menge.
Weiters sei $Q_i := \{p_0, p_1, \ldots, p_i\}$. Dann ist $Q \setminus Q_m$ die Menge der anfangs entfernten Punkte und $[Q_m] = [Q]$.
Sei E_i die Menge der Extremalpunkte von $[Q_i]$.

Es gilt $E_i \subseteq Q_i$ und $p_0, p_1, p_i \in E_i$.

Zu zeigen: Am Ende gilt $S \sim E_m$.

Schleifeninvariante: „Am Beginn jeder Iteration der **for**-Schleife gilt $S \sim E_{i-1}$."

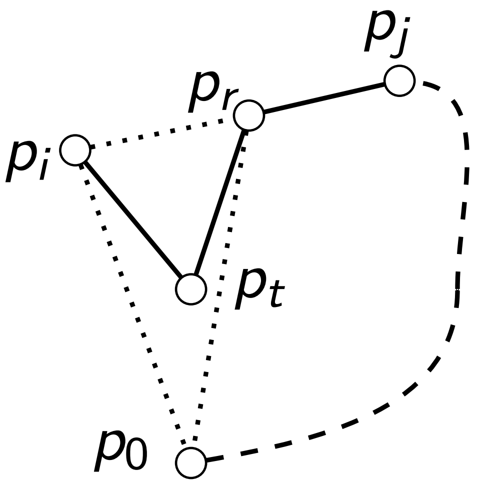
Initialisierung:

$$i = 3\colon\ S = \begin{bmatrix} p_2 \\ p_1 \\ p_0 \end{bmatrix},\ Q_2 = \{p_0, p_1, p_2\} = E_2;\ \text{daher } S \sim E_2.$$

Aufrechterhaltung:
Am Beginn der Iteration zum Wert i haben wir $\mathrm{TOP}(S) = p_{i-1}$.
Nach Ausführen der **while**-Schleife gelte $p_j := \mathrm{TOP}(S)$ und $p_k := \text{NEXT-TO-TOP}(S)$.
D.h. S sieht aus wie unmittelbar nach Ausführen der Iteration zum Wert j der **for**-Schleife ($j < i$).
Nach der Induktionsvoraussetzung haben wir $S \sim E_j$.
$\varphi(p_i) > \varphi(p_j)$ und $p_k \to p_j \to p_i$ ist Linkskurve, denn andernfalls wäre p_j entfernt worden.

Nun wird PUSH(p_i, S) ausgeführt.
Behauptung: $[Q_j \cup \{p_i\}] = [Q_i]$

Bew. d. Beh.: Sei p_t ein Punkt, der durch die **while**-Schleife aus S entfernt wurde, und sei p_r der Punkt unter p_t in S ($r \geq j$).
Dann ist $p_r \to p_t \to p_i$ keine Linkskurve und außerdem $\varphi(p_t) > \varphi(p_r)$.



p_t liegt im Dreieck $p_0 p_r p_i$ (innen oder auf $\overline{p_r p_i}$).
Daraus folgt $p_t \notin E_i$ und daher $[Q_i \setminus \{p_t\}] = [Q_i]$.

Mit $P_i := \{x \in \{p_0, \ldots, p_m\} \mid$ POP(x) in **while**-Schleife der Iteration $i\}$ folgt schließlich $[Q_i \setminus P_i] = [Q_i]$. $\square_{\text{Beh.}}$

Wegen $[Q_i \setminus P_i] = [Q_j \cup \{p_i\}] = [E_j \cup \{p_i\}]$ folgt schließlich $S \sim E_i$.

Terminierung: $i = m + 1$ impliziert $S \sim E_m$. $\square$

Algorithm JARVIS-MARCH(Q)

$x :=$ Punkt in $Q = \{q_0, \ldots, q_n\}$ mit kleinster y-Koordinate, von all diesen jener mit kleinster x-Koordinate
$y :=$ dummy point
$p_0 := x$
$i := 0$
while $y \neq p_0$ **do**
    $p_i := x$
    $y := q_0$
    for $j = 1$ to $|Q| - 1$ **do**
        if $((y = x) \vee (p_i \rightarrow y \rightarrow q_j$ keine Linkskurve$)) \wedge p_i \neq q_j$ **then**
            $y := q_j$
        end if
    end for
    $i := i + 1$
    $x := y$
end while

Laufzeit: $\mathcal{O}(n|E_m|)$
Technik: Wrapping

Das Nächste-Punktepaar-Problem

Gegeben: Punktmenge Q, $|Q| = n \geq 2$

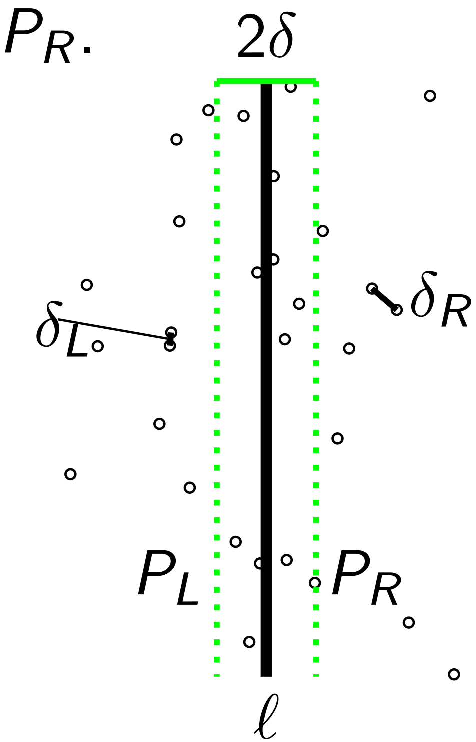
Gesucht: $p_1, p_2 \in Q : \|p_1 - p_2\|_2 = \min_{x,y \in Q} \|x - y\|_2$.

- Brute force: $\binom{n}{2}$ Punktepaare vergleichen: Aufwand $\Theta(n^2)$.
- Divide & Conquer: $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$, also $T(n) = \Theta(n \log n)$.
  - Eingabe:
    - $P \subseteq Q$,
    - Feld X: P sortiert nach der x-Koordinate,
    - Feld Y: P sortiert nach der y-Koordinate.
  - Eingabe bereits sortiert, sonst hätten wir $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n \log n)$ und somit $T(n) = \Theta\left(n(\log n)^2\right)$.
  - Prüfe, ob $|P| \leq 3$.
    - Ja: Brute force
    - Nein: Divide & Conquer.

<u>Teilen</u>: bestimme vertikale Gerade ℓ, sodass $P = P_L \cup P_R$ mit $|P_L| = \lceil |P|/2 \rceil$ und $|P_R| = \lfloor |P|/2 \rfloor$.
Analog: $X = X_L \cup X_R$, $Y = Y_L \cup Y_R$

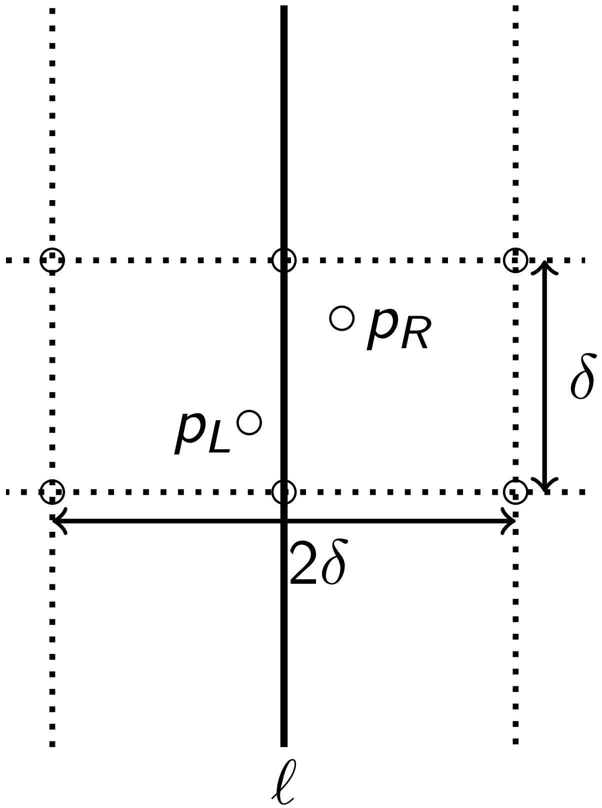
<u>Conquer</u>: rekursive Anwendung auf P_L und P_R liefert minimale Distanzen δ_L, δ_R. Setze $\delta := \min(\delta_L, \delta_R)$.

<u>Kombinieren</u>: gesuchtes Paar (a, b) ist entweder jenes (p, q) mit $\|p - q\|_2 = \delta$ oder eines mit $a \in P_L$, $b \in P_R$.



- $Y' = Y \setminus \{x \in Y \mid d(x, \ell) > \delta\}$
- $p \in Y'$, suche $x \in Y'$ mit $\|x - p\|_2 < \delta$ durch Vergleich von p mit den 7 nachfolgenden Punkten aus Y'.
- $\delta' = \min_{x,x' \in Y'} \|x - x'\|_2$.
- **if** $\delta' < \delta$ **return** (x, x', δ') **else return** (p, q, δ)

Korrektheit: klar, außer: Warum 7 Punkte?
Annahme: $\delta' < \delta$ und $\delta' = \|p_L - p_R\|_2$.



Wegen $\|p_L - p_R\|_2 < \delta$ müssen p_L, p_R in einem $\delta \times 2\delta$ Rechteck R um die Gerade ℓ liegen.

Innerhalb jeder $\delta \times \delta$-Hälfte von R: Alle Punkte haben voneinander Mindestabstand δ.

Maximal vier Punkte pro Rechteckshälfte.

Implementierung und Laufzeit

Hauptproblem: X_L, X_R, Y_L, Y_R, Y' sortiert

Lösung: X, Y vorsortieren; Kosten $\mathcal{O}(n \log n)$.

Y in zwei sortierte Listen zerlegen (umgekehrtes Mergesort):

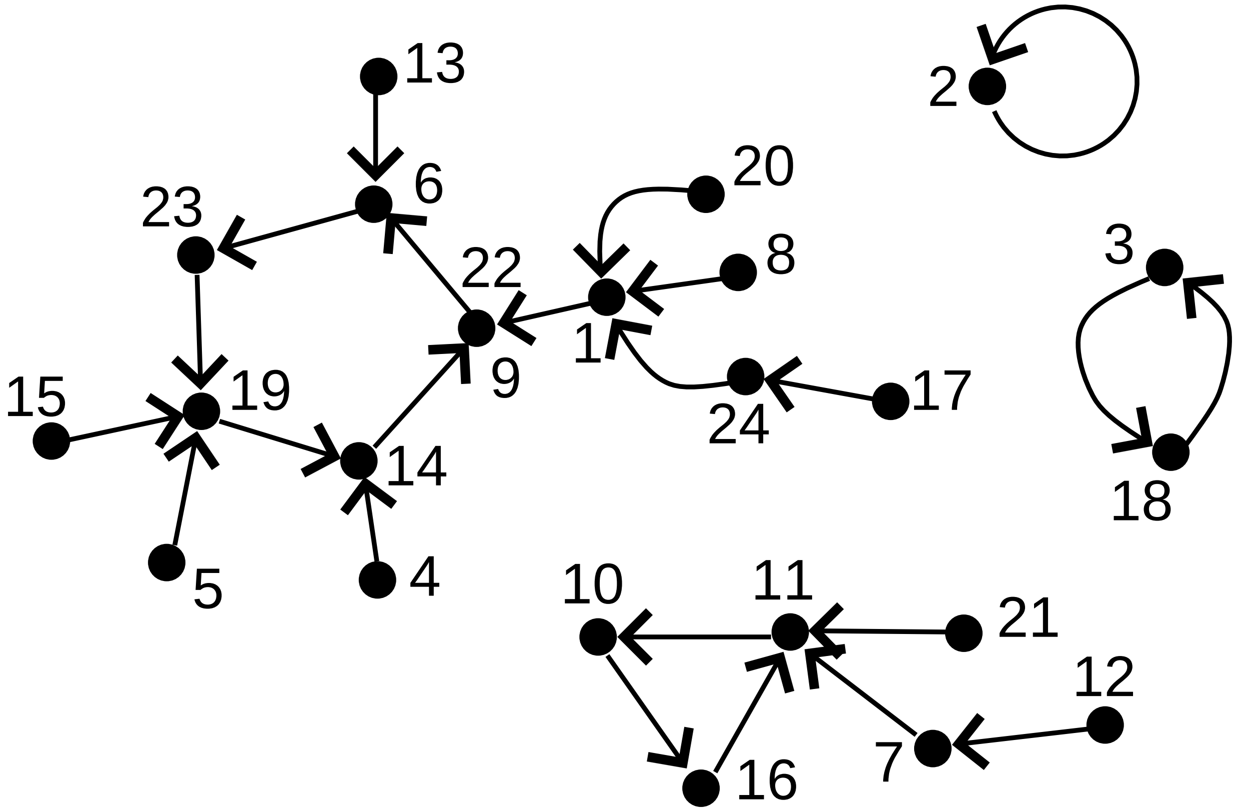
```
Seien Y_L und Y_R neue Felder
|Y_L| := 0; |Y_R| := 0
for i = 1 to |Y| do
    if Y[i] ∈ P_L then
        |Y_L| := |Y_L| + 1
        Y_L[|Y_L|] := Y[i]
    else
        |Y_R| := |Y_R| + 1
        Y_R[|Y_R|] := Y[i]
    end if
end for
```

Aufwand: $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$, also $T(n) = \Theta(n \log n)$.

Faktorisierung – Pollards ρ-Methode

Faktorisierung – Pollards ρ-Methode

Sei $\sigma \in \{1, 2, \ldots, n\}^{\{1,2,\ldots,n\}}$ zufällig gewählt.
Beispiel für $n = 24$:



Für einen zufällig ausgewählten Punkt $x \in \{1, 2, \ldots, n\}$ gilt:

- Die mittlere Distanz zum Zyklus ist $\sim \sqrt{\pi n/2}$.
- Der Zyklus von x hat mittlere Länge $\sim \sqrt{\pi n/2}$.

Faktorisierung – Pollards ρ-Methode

Heuristik: $f(x) = x^2 - 1 \mod n$ verhält sich wie σ.
(allgemeiner: $g(x) = ax^2 + b$)

Erzeuge Folge $(x_i)_{i \geq 0}$ mit $x_{i+1} = x_i^2 - 1 \mod n$
und betrachte die Folge $(x'_i)_{i \geq 0}$ definiert durch

$$x'_i = x_i \mod p$$

für einen nichttrivialen Teiler p von n.

Offensichtlich gilt $x'_{i+1} = (x'_i)^2 - 1 \mod p$. Daher erreicht x'_i nach $t = \Theta(\sqrt{p})$ Schritten einen Zyklus der Länge $\ell = \Theta(\sqrt{p})$, also gilt für alle $i \geq 0$: $x'_{t+\ell+i} = x'_{t+i}$.
Daraus folgt $p \mid x_{t+\ell+i} - x_{t+i}$ und daher $\mathrm{ggT}(x_{t+\ell+i} - x_{t+i}, n) > 1$.

Setzt man $y_i := x_k$ mit $k = 2^{\lfloor \ln_2(i) \rfloor}$ und $y'_i = y_i \mod p$, dann ist für i mit $2^{\lfloor \ln_2(i) \rfloor} \geq t$ der Wert von y'_i im Zyklus von x'_i.

Sobald $k > \ell$ ist y_i für $i = k, k+1, \ldots, 2k-1$ konstant und $2k - 1 > 2\ell$, während x'_i den Zyklus mindestens einmal durchläuft.
$\Longrightarrow$ Für $i = k + \ell$ ist dann $\mathrm{ggT}(x_{k+\ell} - y_{k+\ell}, n) > 1$.

Faktorisierung – Pollards ρ-Methode

Algorithm POLLARD(n)

```
i := 1
x := RANDOM(0, n − 1)
y := x
k := 2
while TRUE do
    i := i + 1
    x := x^2 − 1  mod n
    d := ggT(y − x, n)
    if d ≠ 1 and d ≠ n then print d
    end if
    if i = k then
        y := x
        k := 2k
    end if
end while
```