Algoritmer og datastrukturer
Kapittel 2 - Delkapittel 2.1

2.1 Punkter, linjesegmenter og polygoner

2.1.1 Polygoner og internett

HTML-sider kan ha tegninger, grafikk o.l. med såkalte klikkbare områder (engelsk: hot spot). Det tillates tre typer klikkbare områder - rektangel, sirkel og polygon.

Figur 2.1.1 : Rektangel, sirkel og polygon er tillatt som klikkbart område

I web-leseren ligger det et program (en algoritme) som hele tiden registrerer posisjonen til markøren, og avgjør om den er innenfor eller utenfor et klikkbart område. Hvis området er et rektangel eller en sirkel, så er algoritmen svært enkel.

I html oppgis et rektangel ved hjelp av 4 heltall. Det er de to koordinatene til rektangelets øverste venstre hjørne, og de to koordinatene til det nederste høyre hjørne. La disse hete (x_1 , y_1) og (x_2 , y_2). Husk også at i Javas grafikksystem ligger origo i det øverste venstre hjørnet på skjermen, x-aksen går mot høyre og y-aksen **nedover**. Dette er det omvendte av hvordan vi normalt lager et xy-koordinatsystem. Da har vi også x-aksen mot høyre, men vi har normalt y-aksen oppover. Metoden som avgjør om et punkt (x_0 , y_0) ligger innenfor (eller på) rektangelet, blir slik:

```
public static boolean
innenforRektangel(int x0, int y0, int x1, int y1, int x2, int y2)
{
  if (x0 < x1 || x0 > x2 || y0 < y1 || y0 > y2) return false;
  else return true;
}
```

Programkode 2.1.1 a)

En sirkel oppgis i HTML ved hjelp av tre heltall. Det er de to koordinatene til sirkelens senter og dens radius. Flg. metode avgjør om punktet (x_0 , y_0) ligger innenfor (eller på) sirkelen med senter i (x_1 , y_1) og med radius r:

```
public static boolean
innenSirkel(int x0, int y0, int x1, int y1, int r)
{
  return (x0 - x1)*(x0 - x1) + (y0 - y1)*(y0 - y1) <= r*r;
```

```
}
```

Programkode 2.1.1 b)

Hvis vi har et polygon blir situasjonen langt vanskeligere. Det er ingen enkel oppgave å avgjøre om et punkt er innenfor et generelt polygon eller ikke. Det er imidlertid noe enklere hvis polygonet er konvekst. Måten vi gjør det på er å velge et punkt langt unna, d.v.s. et punkt vi helt sikkert vet ligger utenfor polygonet. Hvis vårt punkt heter (x_0 ,y_0) og punktet som ligger langt unna heter (u_0 ,v_0), så teller vi opp det antallet ganger linjesegmentet fra (x_0 ,y_0) til (u_0 ,v_0) skjærer polygonet. Hvis det skjer et odde antall ganger så er (x_0 ,y_0) innenfor polygonet, og ellers utenfor.

Siden polygonet består av en serie linjesegmenter eller kanter, må vi ha en måte å avgjøre om to linjesegmenter skjærer hverandre eller ikke. I tilegg må vi ta hensyn til at hvis skjæringen er gjennom et polygonhjørne, så skjæres polygonet egentlig bare en gang. Vi skal utvikle teori for dette i avsnittene som følger.

2.1.2 Punkter og rette linjer

Et punkt $p = (x,y)$ i et xy-plan er gitt ved to koordinater - x-koordinaten og y-koordinaten. Disse to koordinatene er normalt desimaltall, men her skal vi kun se på punkter med **heltallige koordinater**. Grunnen er at vi skal lage plangeometriske algoritmer til bruk på en dataskjerm, og der har alle punktene (pikslene) heltallige koordinater.

Vi vil normalt bruke p og q som navn på punkter. Hvis vi trenger flere punkter vil vi indeksere både punktene og punktenes koordinater - f.eks. $p_1 = (x_1 ,y_1)$ og $p_2 = (x_2 ,y_2)$.

Gitt to **forskjellige punkter** $p_1 = (x_1 ,y_1)$ og $p_2 = (x_2 ,y_2)$ i xy-planet. La tallene a, b og c være gitt ved

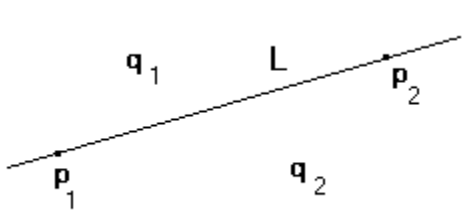
$$a = y_1 - y_2$$
$$b = x_2 - x_1$$
$$c = by_1 + ax_1$$

Likningen for den rette linjen ***L*** gjennom de to punktene er da gitt ved

2.1.2 a) $ax + by - c = 0$

La $q = (x_0 ,y_0)$ være et vilkårlig punkt. Ved hjelp av likningen for ***L*** kan vi bestemme på hvilken side av ***L*** punktet q ligger. La

2.1.2 b) $d = ax_0 + by_0 - c = (x_2 - x_1)(y_0 - y_1) - (y_2 - y_1)(x_0 - x_1)$



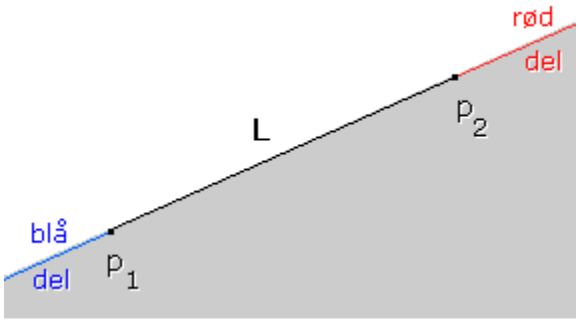
Figur 2.1.2 : Linje gjennom to punkter

Vi vet at hvis $d = 0$ ligger q på linjen ***L***. Vi orienterer linjen ***L*** slik at **positiv retning er fra p_1 til p_2**. Dermed er det veldefinert hva som er den høyre siden av ***L*** og hva som er den venstre siden. På tegningen til høyre blir dermed undersiden av ***L*** det samme som den høyre siden. Men dette er avhengig av hvordan koordinatsystemet vårt er orientert. I tegningen til høyre, og alle andre tegninger lenger nedover, er det underforstått at x-aksen er orientert mot høyre og y-aksen oppover. Det betyr at hvis ***d < 0*** ligger q på **høyre side** og hvis ***d > 0*** ligger q på **venstre side**.

I tegningen ligger $q = q_1$ til venstre for linjen ***L*** og punktet $q = q_2$ til høyre. Det betyr at $d > 0$ for $q = q_1$ og $d < 0$ for $q = q_2$.

Husk at hvis vi arbeider i grafikksystemet til Java så defineres origo til å være det øverste venstre hjørnet på skjermen, den positive x-aksen går mot høyre og den positive y-aksen går nedover. Dermed blir forholdet mellom fortegnet til d og høyre/venstre side av linjen snudd på hodet.

2.1.3 Punkter og linjesegmenter

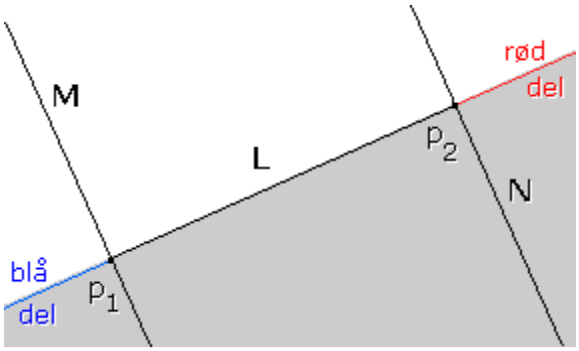


Figur 2.1.3 a) : Et linjesegment

Vi bruker symbolet $[p_1,p_2]$ for linjesegmentet fra p_1 til p_2. Linjesegmentet består av alle punktene på linjen fra og med p_1 til og med p_2.

Fortegnet til d definert i (2.1.2 b) avgjør om et punkt q ligger til venstre for den rette linjen ***L*** gjennom p_1 og p_2, til høyre for linjen eller på linjen. Hvis q ligger på ***L*** er det av interesse å avgjøre om q ligger på selve linjesegmentet $[p_1, p_2]$ eller om q ligger utenfor linjesegmentet. Hvis q ligger utenfor linjesegmentet så er det to muligheter. Enten ligger q på p_1-siden av ***L*** (den blå delen på tegningen) eller på p_2-siden (den røde delen på tegningen).

Spørsmålet er nå om det finnes en enkel formel som avgjør om q ligger på den «blå» delen eller på den «røde» delen.



Figur 2.1.3 b) : Vinkelrett på linjesegment

Vi kan lage en rett linje ***M*** gjennom p_1 vinkelrett på ***L*** og bruke likningen for denne rette linjen til å avgjøre problemet. Vi orienterer ***M*** oppover. Se tegningen til høyre. Vektoren $p_1 - p_2$ er en normalvektor på ***M***. Likningen for ***M*** er da på formen $ax + by = c$ der $a = x_1 - x_2$ og $b = y_1 - y_2$. Verdien til c kan vi bestemme siden ***M*** går gjennom p_1 og dermed blir $c = ax_1 + by_1$. La q som før være punktet $q = (x_0 , y_0)$ og la e være definert ved

2.1.3 a) $$e = ax_0 + by_0 - c = (x_1 - x_2)(x_0 - x_1) + (y_1 - y_2)(y_0 - y_1)$$

Dermed får vi at $q = (x_0 , y_0)$ ligger til venstre for ***M*** (obs ***M*** er orientert oppover) hvis $e > 0$, til høyre for ***M*** hvis $e < 0$ og på ***M*** hvis $e = 0$. Med andre ord ligger q på den blå delen av ***L*** hvis $d = 0$ (se (2.1.2 b)) og $e > 0$.

Kravet $e > 0$ kan forenkles til

2.1.3 b) $$(x_1 - x_2)(x_0 - x_1) > 0 \quad \text{eller} \quad (y_1 - y_2)(y_0 - y_1) > 0$$

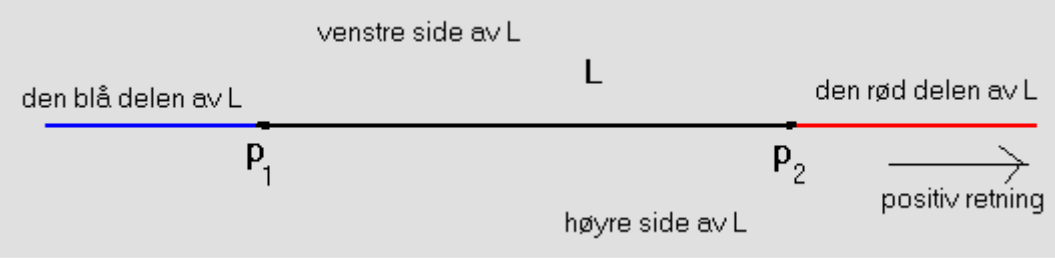
I utrykket for e (siste delen av (2.1.3 a)) er det en sum av to ledd. Hvis $e > 0$ må nødvendigvis minst ett av leddene være større enn 0. Dermed får vi at hvis $e > 0$ så må (2.1.3 b) være sann.

Anta omvendt at (2.1.3 b) er sann. Isolert sett må ikke dermed $e > 0$ være sann. Men hvis vi i tillegg har at $d = 0$ vil vi få at $e > 0$. For å se dette antar vi først at første del av (2.1.3 b) er sann - d.v.s. $(x_1 - x_2)(x_0 - x_1) > 0$. Dermed er også $(x_1 - x_2)/(x_0 - x_1) > 0$. Med andre ord hvis produktet av to heltall er positivt så er også kvotienten positiv. Hvis $(y_1 - y_2)(y_0 - y_1) = 0$ er $e > 0$ oppfylt. Vi kan derfor anta at $(y_1 - y_2)(y_0 - y_1)$ er ulik 0 og dermed at kvotienten $(y_1 - y_2)/(y_0 - y_1)$ er ulik 0. Nå er $d = 0$ det samme som at $(y_1 - y_2)/(y_0 - y_1) = (x_1 - x_2)/(x_0 - x_1)$. Dermed blir $(y_1 - y_2)/(y_0 - y_1) > 0$. Altså hvis $(x_1 - x_2)(x_0 - x_1) > 0$ så må $(y_1 - y_2)(y_0 - y_1) >= 0$, og dermed blir $e > 0$. Hvis vi isteden starter med å anta at $(y_1 - y_2)(y_0 - y_1) > 0$ så finner vi på samme måte at $(x_1 - x_2)(x_0 - x_1) >= 0$ og dermed at $e > 0$.

Til slutt skal vi finne en formel for når $q = (x_0, y_0)$ ligger på den "røde" delen av ***L***. Vi kan f.eks. finne likningen for den rette linjen ***N*** som går gjennom p_2 og står vinkelrett på ***L***. Hvis vi orienterer ***N*** oppover finner vi fort at q ligger til høyre for ***N*** hvis

2.1.3 c) $(x_2 - x_1)(x_0 - x_2) + (y_2 - y_1)(y_0 - y_2) > 0$

Setning 2.1.3: La ***L*** være den rette linjen gjennom $p_1 = (x_1, y_1)$ og $p_2 = (x_2, y_2)$, orientert fra p_1 til p_2 . La $q = (x_0, y_0)$ være et vilkårlig punkt i xy-planet. Da gjelder følgende regel for hvor q ligger i forhold til ***L***:



Figur 2.1.3 : Rett linje med en blå, svart og rød del

1. Hvis $(x_2 - x_1)(y_0 - y_1) < (y_2 - y_1)(x_0 - x_1)$ så ligger q på høyre side av ***L***.
2. Hvis $(x_2 - x_1)(y_0 - y_1) > (y_2 - y_1)(x_0 - x_1)$ så ligger q på venstre side ***L***.
3. Hvis $(x_2 - x_1)(y_0 - y_1) = (y_2 - y_1)(x_0 - x_1)$ og $(x_1 - x_2)(x_0 - x_1) > 0$ eller $(y_1 - y_2)(y_0 - y_1) > 0$ så ligger q på den blå delen av ***L***.
4. Hvis $(x_2 - x_1)(y_0 - y_1) = (y_2 - y_1)(x_0 - x_1)$ og $(x_2 - x_1)(x_0 - x_2) + (y_2 - y_1)(y_0 - y_2) > 0$ så ligger q på den røde delen av ***L***.
5. Hvis verken 1), 2), 3) eller 4) er oppfylt ligger q på linjesegmentet [p_1 , p_2].

Vi lager en Java-metode som forteller oss om posisjonen til $q = (x_0, y_0)$ i forhold til den rette linjen ***L*** gjennom $p_1 = (x_1, y_1)$ og $p_2 = (x_2, y_2)$. Den skal returnere -1 hvis q ligger på den høyre siden eller på den blå delen av ***L***, returnere 1 hvis q ligger på den venstre siden eller på den rød delen og returnere 0 hvis q ligger på linjesegmentet [p_1 , p_2].

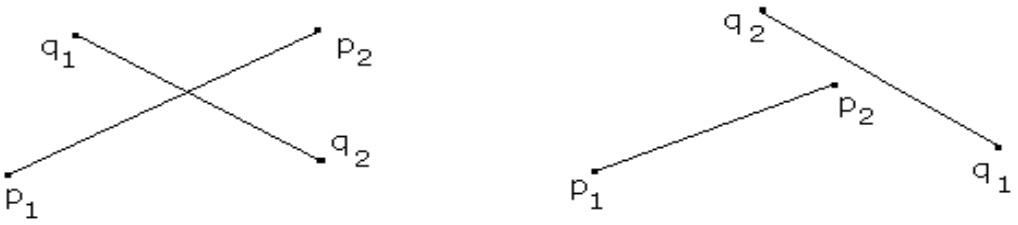
```
public static int posisjon(int x0, int y0, int x1, int y1, int x2, int y2)
{
  // L er den rette linjen gjennom (x1,y1) og (x2,y2)
  // orientert fra (x1,y1) til (x2,y2)
```

```
    int d = (x2 - x1)*(y0  - y1) - (y2 - y1)*(x0  - x1);

    if (d < 0)   return -1;   // (x0,y0) på høyre side av L

    else if (d > 0)   return 1;  // (x0,y0) på venstre side av L

    else if ((x1 - x2)*(x0  - x1) > 0   // (x0,y0) på den blå delen av L
    || (y1 - y2)*(y0  - y1) > 0) return -1;

    else if ((x2 - x1 )*(x0 - x2 )      // (x0,y0) på den rød delen av L
    + (y2 - y1 )*(y0 - y2 ) > 0) return 1;

    else return 0;  // (x0,y0) på linjesegmentet
}
```

Programkode 2.1.3 a)

I hele utledningen over er det forutsatt at p_1 og p_2 er forskjellige punkter. Det er imidlertid ikke gjort slike forutsetninger i metoden *posisjon*. Hva skjer hvis $p_1 = p_2$, d.v.s. hvis $x_1 = x_2$ og $y_1 = y_2$? Vi ser at returverdien blir 0 for dette tilfellet.

2.1.4 Skjæring mellom linjesegmenter



Figur 2.1.4 a) : Skjæring og ikke skjæring mellom to linjesegmenter

I *Figur* 2.1.4 *a)* har vi to tilfeller med to linjesegmenter. I det første tilfellet (til venstre) har vi skjæring og i det andre er det ingen skjæring. Spørsmålet er nå om det kan settes opp en enkel formel for når to linjesegmenter skjærer hverandre?

Hvis vi tenker oss litt om finner vi at linjesegmentene $[p_1,p_2]$ og $[\ q_1\ ,\ q_2\ ]$ skjærer hverandre hvis q_1 og q_2 ligger på hver sin side av den rette linjen gjennom p_1 og p_2 samtidig med at p_1 og p_2 ligger på hver sin side av den rette linjen gjennom q_1 og q_2 .

Vi får også skjæring hvis q_1 eller q_2 ligger på linjesegmentet $[\ p_1\ ,\ p_2\ ]$, og vi får skjæring hvis p_1 eller p_2 ligger på linjesegmentet $[\ q_1\ ,\ q_2\ ]$. Andre muligheter for skjæring er det ikke.

Dette kan vi sammenfatte i metoden *skjæring*:

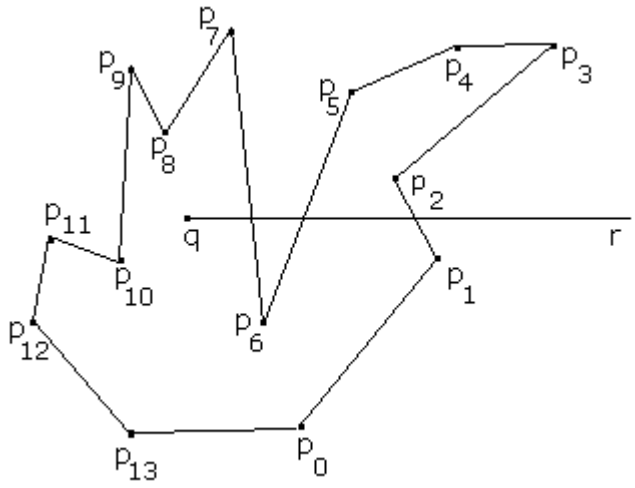
```
public static boolean
skjæring(int x1, int y1, int x2, int y2,
int u1, int v1, int u2, int v2)
{
  // p1 = (x1,y1), p2 = (x2,y2), q1 = (u1,v1), q2 = (u2,v2)
```

```
    return
     posisjon(u1,v1,x1,y1,x2,y2) * posisjon(u2,v2,x1,y1,x2,y2) <= 0
     && posisjon(x1,y1,u1,v1,u2,v2) * posisjon(x2,y2,u1,v1,u2,v2) <= 0;
}
```

Programkode 2.1.4 a)

2.1.5 Punkt i polygon

Gitt n punkter, d.v.s. p_0 , p_1 , p_2 . . . . p_{n-1} som utgjør hjørnene i et polygon, og et punkt q som vi skal undersøke. Algoritmen for å avgjøre om q ligger inne i eller utenfor polygonet, gitt at polygonet ikke skjærer seg selv, blir nå:



Figur 2.1.5 a) : Et ikke-konvekst polygon

1. Anta at y-koordinaten til q er forskjellig fra y-koordinatene til alle p-ene. Det betyr at en horisontal rett linje gjennom q ikke går gjennom noen av p-ene. La r være et punkt som har samme y-koordinat som q og som ligger så langt til høyre at det helt sikkert er utenfor polygonet. Finn det antallet ganger linjesegmentet fra q til r skjærer polygonet. Hvis antallet er odde er q inne i eller på polygonet og hvis antallet er jevnt er q utenfor polygonet.

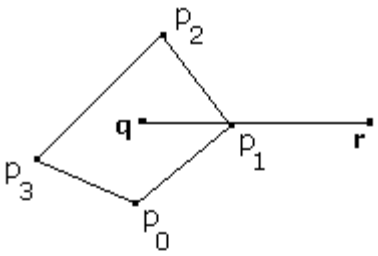
På tegningen til høyre, der vi har 14 punkter, ser vi at linjesegmentet fra q til r skjærer polygonet 3 ganger - d.v.s. et odde antall ganger, og vi ser at det er i overenstemmelse med at q ligger inne i polygonet.

2. Hvis minst en av p-ene har samme y-koordinat som q, så er situsjonen mer kompleks. Tegningene nedenfor til høyre viser eksempler der det horisontale linjesegmentet fra q til r går gjennom en av p-ene. Poenget er at vår algoritme alltid skal returnere et odde tall når q ligger inne i eller på polygonet og et jevnt tall ellers - d.v.s. når q ligger utenfor.

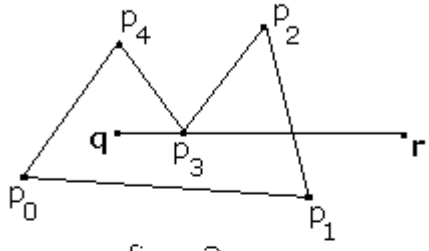
Når vi skal implementere en algoritme som finner antallet skjæringer mellom polygonet og linjesegmentet fra q til r vil det være naturlig å bruke metoden (*programkode 2.1.4*) som avgjør om to linjesegmenter skjærer hverandre eller ikke. Denne metoden anvender vi da fortløpende på linjesegmentene [p_0 , p_1], [p_1 , p_2], [p_2 , p_3] o.s.v. til og med segmentet [p_{n-1} , p_0] der n er antallet hjørnepunkter i polygonet. Dette blir imidlertid problematisk når linjesegmentet [q , r] skjærer polygonet i et hjørnepunkt.

Figur 1 til venstre viser et eksempel der linjesegmentet [q , r] skjærer polygonet i et hjørnepunkt. Linjesegmentet skjærer både [p_0 , p_1] og [p_1 , p_2]. Vi får derfor registrert to skjæringer - noe som gir galt antall. Dette kunne vi håndtere ved å si at i et slikt tilfelle skal det bare telles opp som en skjæring. Dermed får vi i dette eksemplet at antallet skjæringer er en og det stemmer med at q ligger inne i polygonet.

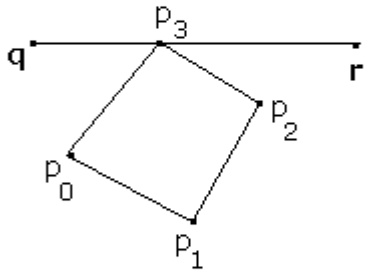
I *Figur 2* til høyre har vi en annen situasjon. Der skjærer (eller tangerer) [q , r] polygonet i hjørnet p_3 - d.v.s. segmentet skjærer [p_2 , p_3] og [p_3 , p_4]. Men nå kan ikke dette telles opp som kun en skjæring. Hvis vi gjør det vil antallet skjæringer mellom [q , r] og polygonet bli registrert som 2 siden vi også har en skjæring mellom [q , r] og [p_1 , p_2]. Men det er et jevnt tall og dermed i strid med at q ligger inne i polygonet.



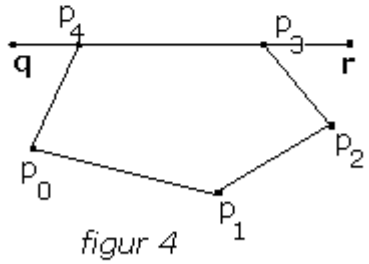
figur 1



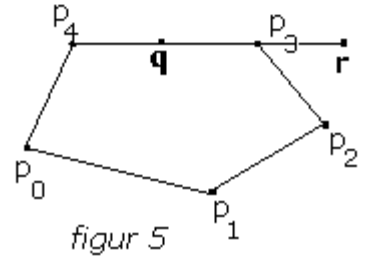
figur 2



figur 3



figur 4



figur 5

Figur 2.1.5 b) : Forskjellige polygoner

I *Figur 3* har vi et tilsvarende tilfelle som i *Figur 2*. Linjesegmentet [*q* , *r*] skjærer (eller tangerer) både [p_2 , p_3] og [p_3 , p_0]. Registerer vi dette som bare en skjæring mellom [*q* , *r*] og polygonet, blir det galt siden *q* ligger utenfor polygonet.

Enda et problem er at [*q* , *r*] kan falle sammen med en av polygonets kanter. I *Figur 4* har vi en slik situasjon. Vår algoritme vil her registrere at [*q* , *r*] skjærer både [p_2 , p_3], [p_3 , p_4] og [p_4 , p_0]. Skal algoritmen vå fungere riktig må dette registreres som et jevnt antall skjæringer - f.eks. 0, 2 eller 4 - siden *q* nå ligger utenfor polygonet.

En måte vi kan takle dette på er å dele tilfellene der linjesegmentet fra *q* til *r* skjærer polygonet i ett eller flere hjørnepunkter, i to grupper. Den ene gruppen omfatter det vi kan kalle en slags tangeringer. *Figur 2*, *Figur 3* og *Figur 4* hører til i den gruppen. Den andre gruppen omfatter ekte skjæringer slik som i *Figur 1*. Har vi en tangering registrerer vi det som 0 skjæringer og har vi en ekte skjæring blir det registrert som 1. Men det vil være tilfeller som ikke uten videre kan plasseres i noen av gruppene. Hvis vi som i *Figur 5* har *q* inne på kanten [p_3 , p_4] blir det da skjæring eller "tangering"? I dette tilfellet ligger *q* på polygonet og vi må derfor registere det som 1 skjæring.

Konklusjon Hvis vi skal få til en algoritme som virker korrekt også for alle spesialtilfellene må vi være meget nøye og omhyggelige under implementasjonen. Det som er enkelt å få til er å telle opp antallet skjæringer hvis linjesegmentet fra *q* til *r* ikke skjærer noen av polygonets hjørner. Men også de spesialtilfellene som er diskutert over må behandles på en korrekt måte, og det må sjekkes nøye om det finnes ytterligere spesialtilfeller utover de som er diskutert her.

Oppgave 1

Metoden *skjæring* i *programkode 2.1.4* returnerer *true/false* avhengig av om linjesegmentene [p_1 , p_2] og [q_1 , q_2] skjærer hverandre eller ikke. Sjekk at metoden vil komme til å returnere korrekt svar for de 7 tilfellene i figur 1 - 7 ovenfor.