Oblig 5 i IN3030 våren 2019. Den konvekse innhyllinga til en punktmengde - et rekursivt geometrisk problem. Innleveringsfrist onsdag 10. mai 2019 kl. 23.59

En punktmengde P i planet består av n forskjellige punkter p_i (i =0,1,2,…,n-1) med heltalls-koordinater og vi kan anta at *ikke* alle punktene ligger på én og samme linje. Den konvekse innhyllinga til denne punktmengden er en mangekant (et polygon) med linjer mellom noen av disse punktene slik at alle de andre punktene er på innsiden av denne mangekanten og også slik at alle indre vinklene i denne mangekanten er ≤ 180 – sagt litt enklere : det er ingen innoverbulker på dette sett av linjer rundt punktmengden (fig1).

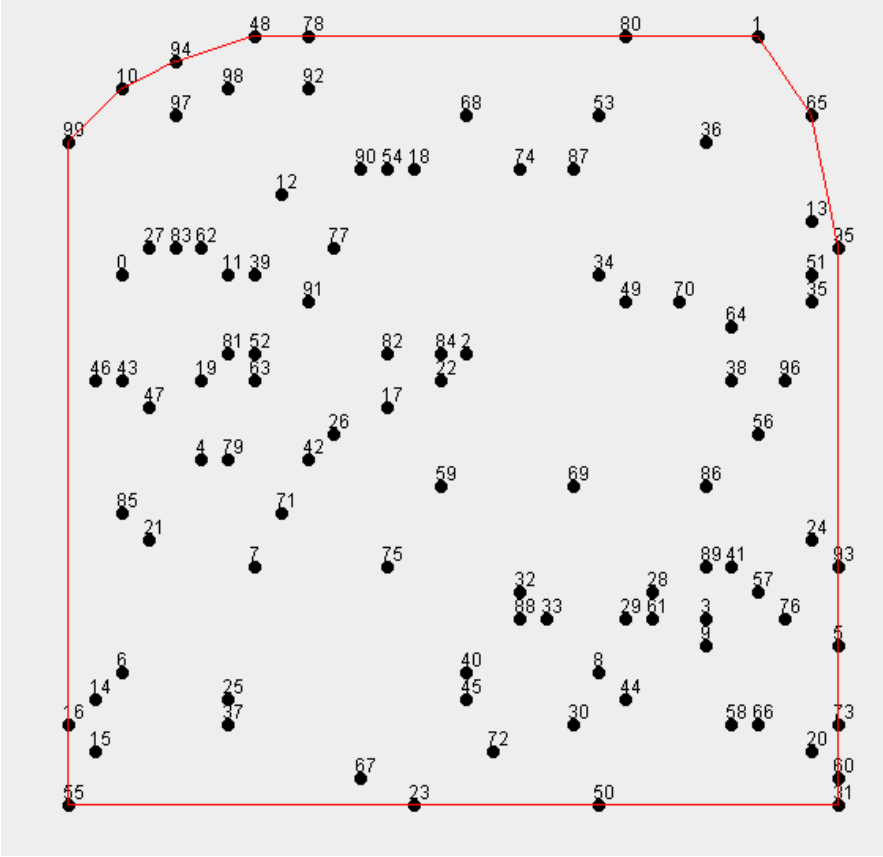


Fig1. Den konvekse innhyllinga av 100 tilfeldige punkter i planet (de samme som nyttes i oppgaven.)

Når vi noterer en slik konveks innhylling, starter vi med et vilkårlig av punktene i innhyllinga, og nevner punktene i rekkefølge mot klokka. I figur 1 kan det være: 16, 55, 23, 50, 31, 60, 73, 5, 93, 95, 65, 1, 80, 78, 48, 94, 10, 99. Legg merke til at hvis det er flere punkter på en linje (her f.eks. 31, 60, 73, 5, 93 og 95), så skal alle disse punktene med på innhyllinga. Det er altså ikke lov å gå rett fra 31 til 95, vi må spesifisere alle linjene 31-60, 60-73, 73-5, 5-93 og 93-95; ikke hoppe over noen av tallene mellom. Hvis det brukes i løsningen din kan du anta at det er alltid mulig å velge de punktene som har størst og minst x- og y-verdier slik at det er 4 forskjellige punkter (se pkt. 55 og 31 i Fig.1).

Hvordan finner vi innhyllinga av n punkter?

Vi ser at følgende to punkter opplagt er på innhyllinga – det med minst x-verdi og det med størst x-verdi. Er det flere med denne samme minste eller største x-verdi, velger man bare en av de største og en av de med minst x-verdi (i eksempelet f.eks. 16 og 5). Resten av algoritmen baserer seg på en enkel geometrisk sats.

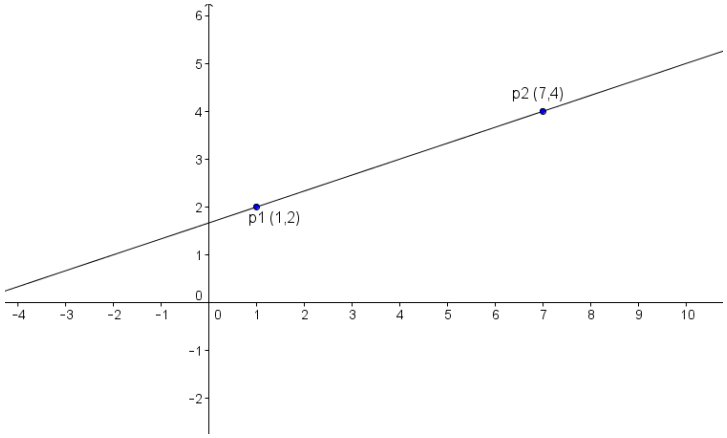
Ligningen for en linje

Enhver linje fra et punkt p1 (x1,y1) til p2(x2,y2) kan skrives på formen:

$$ax + by + c = 0$$

Hvor: $a = y1 - y2, b = x2 - x1 \; og \; c = y2 * x1 - y1 * x2.$

Merk at dette er **en rettet linje *fra* p1 *til* p2**.



Figur2. En linje fra p1 (1,2) til p2(7,4) har da linjeligningen:

$$(2-4)x \; + \; (7-1)y + (4*1 - 2*7) = 0; \; dvs: \; -2x + 6y - 10 = 0$$

Linjeligningen betyr at alle punkter på denne linja passer inn i formelen; setter vi x og y til ethvert punkt på denne linja får vi 0 som svar. Denne linja kan vi også si deler planet i to – de punkter som er på venstre side sett i retningen på linja: fra p1 til p2, og de som på høyre side i retningen fra p1 til p2.

Avstanden til linja fra andre punkter.

Setter vi inn punkter som ikke er på linja p1-p2, ser vi at alle punkter, eks. p3(5,1) på høyre side av linja gir et tall < 0 innsatt i linjeligninga: $ax + by + c$, og alle punkter til venstre for linja, som p4(3,6), gir et tall > 0.

Jo lenger fra linja punktene er, desto større negative og positive tall blir det. Generelt er det slik at avstanden vinkelrett ned fra et punkt p(x,y) til en linje: $ax + by + c$ er:

$$d = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

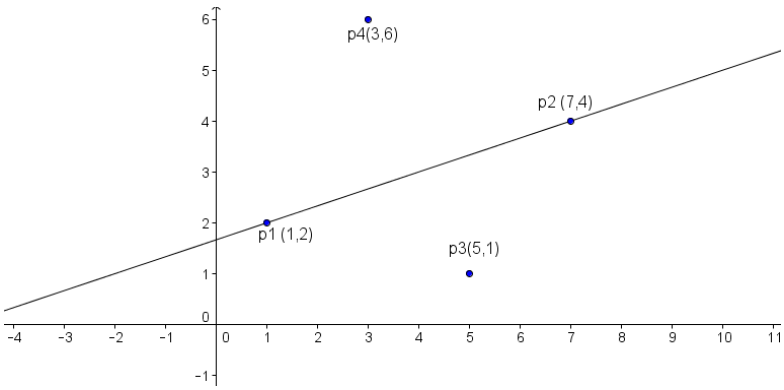


Fig3. Avstanden fra et punkt til en linje. Linja deler planet i to: de med negativ avstand (til høyre) og de med positiv avstand (til venstre) fra linja.

Vi ser på to punkter i fig 3: p3(5,1) og p4(3,6) , finner at avstanden fra p4 til linja p1-p2 er: $\frac{-2*3+6*6-10}{\sqrt{(-2)^2+6^2}} = \frac{20}{\sqrt{40}} = 3{,}16..$, mens avstanden fra p3(5,1) til p1-p2 er: $\frac{-2*5+6*1-10}{\sqrt{40}} = \frac{-14}{6{,}32} = -2{,}21..$

Algoritmen for den konvekse innhyllinga

Vi kan nå formulere algoritmen for den konvekse innhylling etter å ha gjort én observasjon til:

- Det punktet som har lengst negativ avstand fra en linje pi-pj , er selv et punkt på den konvekse innhyllinga (se fig 4 punktet P og linja I - A)).

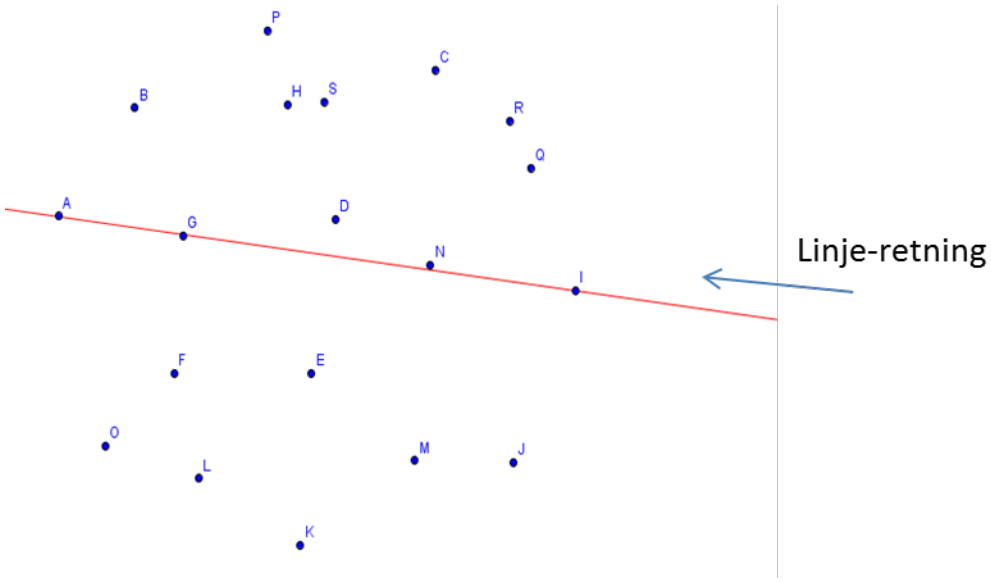


Fig.4. Starten på å finne innhyllinga fra maxx (I) til minx (A).

Algoritme:

1. Trekk linja mellom de to punktene vi vet er på innhyllinga fra maxx -minx (I - A i fig4).
2. Finn det punktet med størst negativ (eller 0) avstand fra linja (i fig 4 er det P). Har flere punkter samme avstand, velger vi bare ett av dem.
3. Trekk linja fra de to punktene på linja til dette nye punktet på innhyllinga (i fig5: I-P og P-A).

4. Fortsett rekursivt ut fra de to nye linjene og finn da for hver av disse linjene et nytt punkt på innhyllinga i størst ikke-positiv avstand (≤ 0).
5. Gjenta pkt. 3 og 4 inntil det ikke er flere punkter på utsida av disse linjene.
6. Gjenta steg 2-4 for linja minx-maxx, og finn alle punkter på innhyllinga under denne.

I figur 5 (neste side) er denne prosessen illustrert. Her er illustrasjonen på å finne punktene på innhyllinga over linja I-A. Du kan forvente å finne om lag 1.4* $\sqrt{n}$ elementer i den konvekse innhyllinga av n tilfeldig valgte punkter i planet, men alle verdier fra 3 til n er mulig (vi får n hvis alle punktene er på en sirkel).

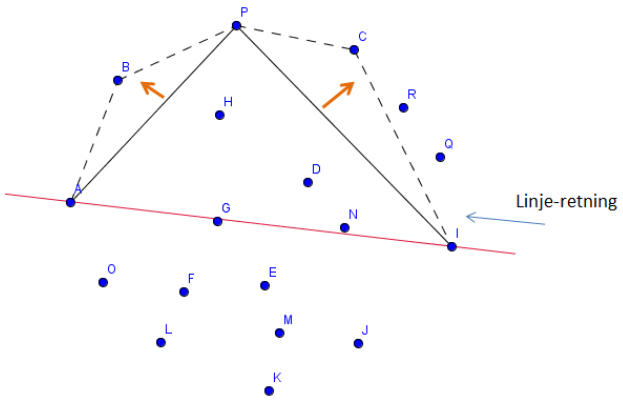


Fig 5. Vi finner rekursivt punktet P på innhyllinga fra linja I-A (det med størst negativ avstand fra maxx-minx), så C fra P-I, R fra I-C og til sist Q fra I-R.

Denne rekursive algoritmen skal du altså programmere først sekvensielt og så lage en parallell versjon av og finne speedup for når n=100, 1000, 10 000, …, 10 mill.

Om å generere n ulike punkter.

a) Datastrukturen din for n punkter skal være to int-arrayer x og y, hver n lange:

```
int []x = new int[n], y = new int[n];
```

For å fylle disse arrayene med x- og y-verdier som ikke gir to like punkter, skal du bruke den ferdigkonstruerte klassen NPunkter som du finner i mappa til Oblig5.

b) For hver verdi av n du skal ha i programmet ditt, skal du først lage et objekt av klassen NPunkter :

```
NPunkter p = new NPunkter(n);
```

og så fylle x[] og y[] arrayene dine ved et kall på metoden i objektet p:

```
p.fyllArrayer(int [] x, int [] y);
```

Grunnen til at skal bruke denne klassen , som du heller ikke skal endre på, er at alle besvarelsene for en gitt n skal ha de samme punktene. Da vil det bli mye lettere for gruppelærerne å rette obligene (det er heller ikke helt lett å relativt raskt generere opp til 10 mill. tilfeldige punkter som man er sikker på alle er ulike, men denne oppgaven er altså ikke en del av Oblig 5).

c) Denne generering av n punkter skal ikke inkluderes i tidtakingen.

Effektivitet:

a) Du er egentlig ikke er interessert i den virkelige avstanden for et punkt til en linje, men bare et mål for denne avstanden hvor den som har størst tall 'vinner'. Formelen for avstanden d kan da forenkles.

b) Det vil ta lenger tid hvis du hver gang må gå gjennom hele punktmengden P for å finne de få punktene som er på utsida av en linje. Du skal derfor la programmet teste på stadig mindre punktmengder.

c) Husk at du ikke ukritisk bare bruker tråder istedenfor rekursjon (tenk om lag som for Quicksort). Tråder skal bare brukes i toppen av rekursjonstreet.

d) Er **`ArrayList <Integer>`** rask nok til å holde til dels store punktmengder, eller bør du lage en egen klasse **`IntList`** med de 'samme' metodene du trenger og hvor de heltallene du skal lagre ligger i en enkel heltallsarray?

Oppgaven.

I Oblig 5 skal du:

- Programmere sekvensiell utgave av algoritmen som angivet i denne oppgave.
- Parallellisere den sekvensielle utgave etter metode 1 eller 2, som beskrevet i uke 14 powerpoint slides.
- Bruke precode `npunkter17.java & TegnUt.java` (publiceret på web)

Krav til innleveringa.

Oblig 5 leveres i Devilry innen fredag 10. mai kl. 23.59. I leveransen skal du ha:

a) Koden både for den sekvensielle og parallelle løsningen pakket i en zip fil.

b) En utskrift på den konvekse innhyllinga for n=1000.

c) En rapport med både en tabell og en kurve som viser speedup som funksjon av n (= 100, 1000 ..., 10 mill.) og din vurdering på hvorfor kjøretidene blir så lave som de blir, og om hvorfor du får den speedup du får. NB: kjøretid for n=100 mill. bør være under 14s og n=10 mill. bør være under 2s. Ta med data om den maskinen du har kjørt testene på (modell-betegnelse, klokkefrekvens, antall kjerner) og hovedhukommelsens størrelse, og om mulig: cachenes størrelse og forsinkelse (kjør programmet latency.exe hvis du er på en Windows-maskin) og gjerne med en tegning fra TegnUt (publiseres på kurs web) av tilfellet n=100.

Bruk Oblig-5-recommendations – publiseres i Oblig mappa på kurs web senere.

Tips.

Dette er noe du ikke behøver å følge, men gir tips på ulike punkter hvor du kan få problemer:

I) Den sekvensielle løsningen

a) Hvordan representere et punkt.

Bruk indeksen i x[] og y[] – innholdet av disse endrer seg ikke under kjøring; de bare leses.

b) Debugging

Svært få, om noen, greier å få riktig kode første gang. Vi trenger å feilrette koden, og på et grafisk problem som dette, vil være greit å få tegnet ut punktene for et lite eksempel og den innhyllinga man etterhvert finner. Hvis dere ikke vil bruke en egen tegnepakke, kan dere bruke klassen TegnUt slik fra mainklassen (for eksempler hvor n <200) slik:

```
TegnUt tu = new TegnUt (this, koHyll);
```

Første linje er for å få tegnet ut punktene og den konvekse innhyllinga som forventes å være lagt inn i en IntList koHyll (en triviell forenkling av ArrayList <Integer>).

c) Finne punktene på den konvekse innhyllinga i riktig rekkefølge.

Tips: Du bruker to metoder for det sekvensielle tilfellet (antar her at du bruker ArrayList, men denne kan trivielt byttes ut med en raskere IntList hvis du lager en slik):

`sekvMetode()` som finner minx, maxx og starter rekursjonen. Starter du på maxx-minx, legges maxx inn i lista med innhyllinga (men ikke minx enda), og starter rekursjonen med to kall på den rekursive metoden:

`sekvRek (int p2, int p2, int p3, ArrayList <Integer> m, ArrayList <Integer> koHyll )` som får en mengde punkter m, som inneholder alle punktene som ligger på eller under linja p1-p2, og p3 er allerede funnet som det det med størst negativ avstand fra p1-p2. koHyll er mengden hvor du skal notere punktene på den konvekse innhyllinga du finner i riktig rekkefølge.

Merk at du først tar rekursjon først på høyre del (linja I-P, fig. 5) og så på venstre del(linja P-A) fordi vi vil ha punktene i rekkefølge notert mot klokka.

Du kan la hvert kall på sekvRek legge inn ett punkt: p3 i innhyllinga-lista, men hvor i koden er det ?

Når legges minx inn i innhyllingslista?

d) Få med alle punktene på innhyllinga hvor flere/mange ligger på samme linje (avstand = 0) som i høyre kant i fig. 6, og i riktig rekkefølge.

Husk at når du finner at største negative avstand =0, må du ikke inkluderer p1 eller p2 som mulig nytt punkt (de er allerede funnet)

Si at du har funnet p1=31 og p2=5 interessert i å finne de punktene som ligger mellom p1 og p2 på linja (60 og 73), og da må du teste om nytt punkt p3 har både y og x-koordinater mellom tilsvarende koordinater for p1 og p2. Da finner du ett av punktene med kall på sekRek over linja p1-p2 (31-5). Gjenta rekursivt til det ikke lenger er noen punkter mellom nye p1 og p2.

Punktene videre oppetter linja (f.eks. 5-95) finnes av rekursjon tilsvarende som for 60 og 73.

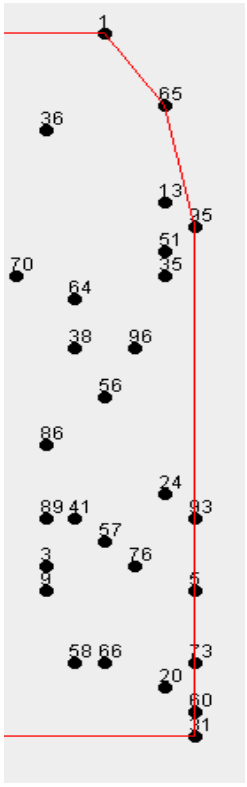


Fig 6. Mange punkter med samme avstand =0

II) Den parallelle løsningen

a) Du kan/bør ikke bruke Model2-koden direkte, men dele det parallelle tilfellet også opp i to metoder. Den første parallell-kalles fra main-tråden (mellom de to setningene der du tar tiden på det parallelle tilfellet). Den finner først ut hvor langt ned i kalltreet vi skal bruke tråder. Har vi k kjerner i maskinen, kan det være fornuftig å gå ned til et lag i kalltreet hvor det er om lag k noder på tvers av treet. Eksempel: har vi 8 kjerner, og kaller toppen i treet for nivå1, vil nivå4 ha 8 noder. Inntil dette nivået vi da nye tråder erstatte rekursive kall, på etterfølgende nivåer brukes så vanlig rekursive kall fra hver av de trådene vi har startet.

b) Den første parallelle metoden som kalles fra og kjører på main-tråden, lager altså to tråder. De etterfølgende trådene startes fra disse to trådene og deres 'etterkommer'-tråder . Disse startes av den andre parallelle metoden, som kalles fra run()-metodene.

c) Den andre parallelle metoden har de samme parametere som tilsvarende sekvensielle:p1,p2,p3, og søkemengden, og i tillegg to til: nivået i treet og mengden (IntArray) av punkter på innhyllingen som denne tråden hittil har funnet.

d) I den andre parallelle, rekursive metoden avgjøres det så om vi skal fortsette å lage tråder eller om det er rekursjon som nå gjelder. Før det, har vi funnet punktet som ligger i mest

negativ avstand og søkemengden for neste kall (=de punktene som er negative satt inn i linja fra p1-p2) og vi har også deklarert og generert to mengder, en til hver av de to rekursjonene (IntList eller ArrayList<Integer>) som skal holde den konkvekse innhyllinga dette kaller finner.

e) Denne metoden minner sterkt om den tilsvarende sekvensielle metoden, og etter at vi har kommet ned på det nivået hvor vi ikke lager nye tråder lenger, kan vi faktisk bruke den sekvensielle rekursive metoden her.

f) Husk at når vi starter tråder så må vi først skape begge, og så etterpå legge oss å vente (f.eks. med join()) på dem. Ellers blir det ikke parallellitet.

Finne punktene på den konvekse innhyllinga i den parallelle løsningen i riktig rekkefølge.

Vi får imidlertid et ekstra problem, mens man i den sekvensielle løsningen kan få punktene på innhyllinga i riktig rekkefølge ved å legge dem inn riktig rekkefølge (ett punkt per rekursivt kall), kommer punktene i en parallell løsning fra ulike tråder og i uforutsigbar rekkefølge. En måte å løse dette på er å bruke:

1. Trådene startes i toppen av rekursjonstreet bare ned til et visst nivå for alle nodene i treet (nivået er som sagt avhengig av hvor mange kjerner vi har)..
2. Etter at det ikke startes flere nye tråder, vil hver tråd gå over til den sekvensielle løsningen (på sin del av data).
3. Fra hver av disse sekvensielle utførelsene får vi punktene i riktig rekkefølge (mot klokka).
4. Problemet er da å skjøte sammen disse riktige 'små'-delene av innhyllinga (+ de få punktene som finnes av trådene i toppen av rekursjonstreet) om lag slik: Først det høyre-tråden har produsert, så det punktet noden selv finner, så venstre-trådens punkter.
5. Punkt 4 gjøres så (på tilbaketrekking etter å ha funnet sine punkter) oppover til vi kommer til toppen og er ferdig.