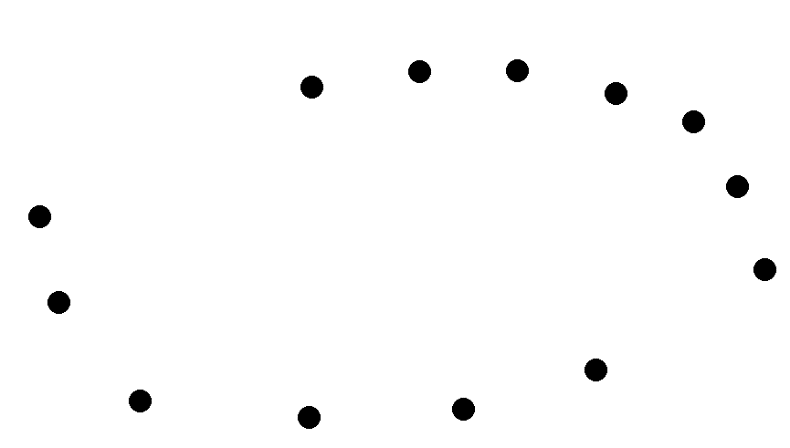
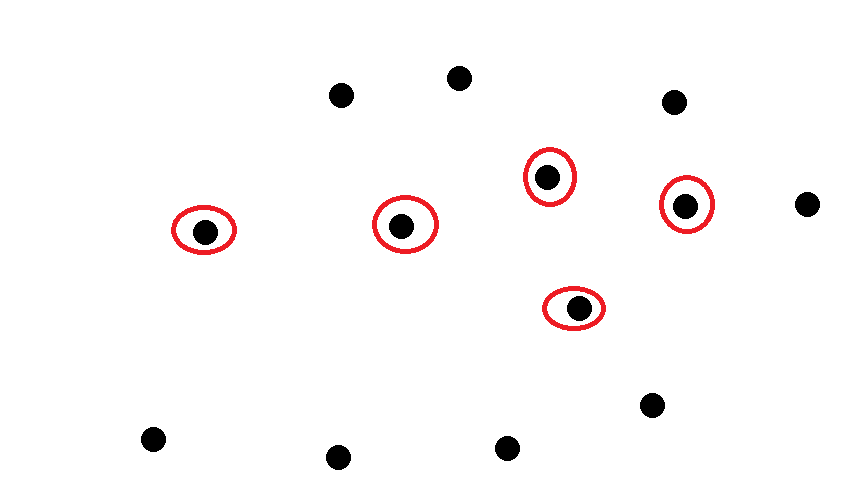
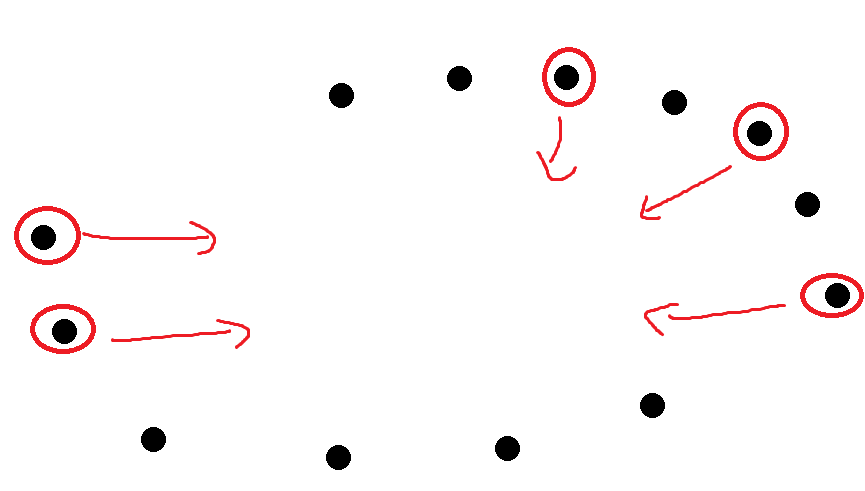
Pozdravljeni. Tu sem, da vam povem kaj je Grahamov pregled. Ampak najprej si poglejmo primer. H nam pride pastirček s problemom. Moti ga, da mora na [[krave]] stalno paziti, da mu ne uidejo in ne more v miru igrati igric na telefonu. Pove nam še, da si [[krave]] ko pridejo na pašnik najprej spočijejo in se nekaj časa ne premikajo. Ko si spočijejo, se prosto sprehajajo po pašniku. Kako bi pastirčku lahko pomagali?

Hja, pastirček bi lahko postavil ograjo. A kako bi jo postavil? Kako bi se pastirček odločil kje bi potekala ograja, da bi lahko igral igrice, popolnoma brez skrbi, da mu [[krave]] pobegnile?

*PRIMER: VSE NA KONVEKSNI*

**

Kot sem že rekel… Ko se [[krave]] spočijejo, se začnejo premikati. Tako se recimo premakne *{ta},{ta}…* Naš pastirček pride preveriti, če je z prej postavljeno ograjo še vedno vse v redu in opazi, da so se krave premaknile. Je njegova ograje še vedno optimalno postavljena? Kako se ta postavitev [[krav]] razlikuje od prejšnje?



Vidimo, da pastir ne rabi krajišča ograje postaviti h vsaki kravi. Če ograjo postavimo *TAKO*, bodo vanjo še vedno ujete vse krave. Temu pravimo konveksna ovojnica. In Grahamov pregled je eden izmed številnih algoritmov, ki iščejo konveksno ovojnico.

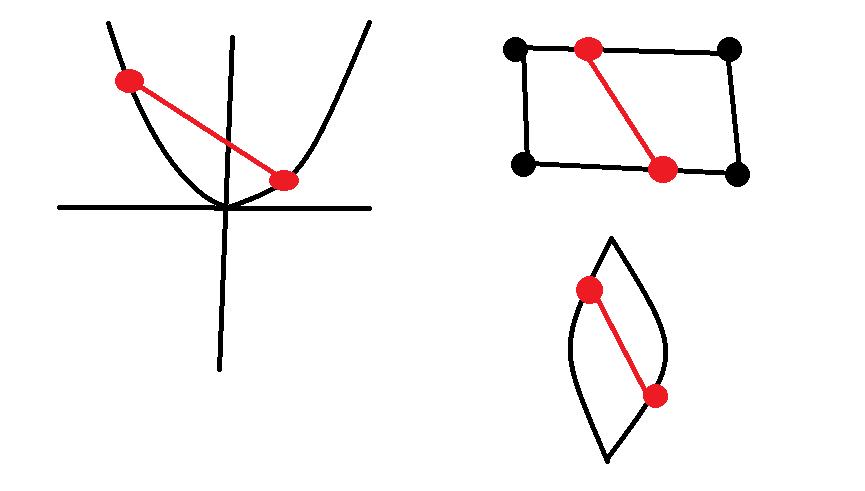
**Konveksnost**

S pojmom konveksnosti smo se že večkrat srečali. Poznamo konveksno funkcijo (npr. kvadratna funkcija), konveksno množico (npr. pravokotnik), konveksno lečo…

Definicija konveksnosti za funkcije pravi, da je funkcija konveksna, če za poljubni točki A,B iz naše funkcije, velja, da je graf funkcije med točkama A in B vedno pod daljico med A in B.

Konveksnost množice v geometriji, podobno kot za konveksne funkcije, pomeni, da za poljubni točki iz te množice velja, da je daljica med tema točkama prav tako znotraj te množice.

*Nariši x^2, lik, lečo, pokaži konveksnost*

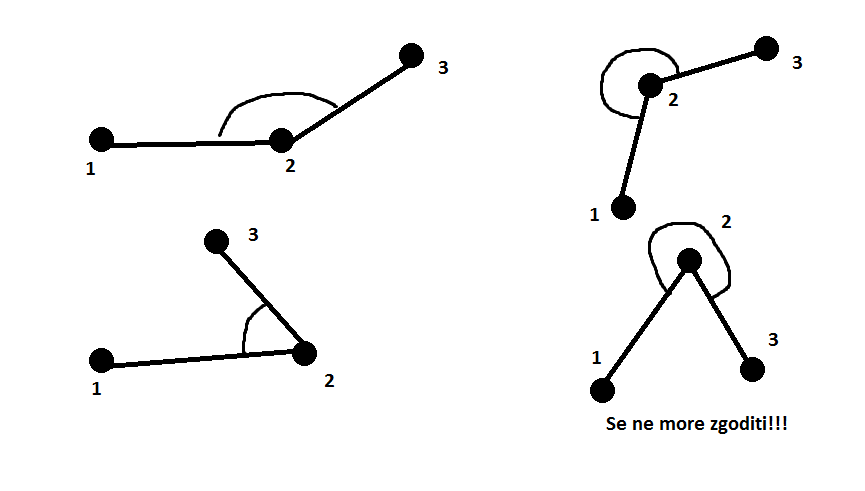
**

**Konveksna ovojnica**

Konveksna ovojnica je najmanjša možna podmnožica naše množice, ki vsebuje vse elemente množice. *POKAŽI NA [[KRAVE]] OD PREJ*

**Levi in desni obrat**

Med iskanjem konveksne ovojnice, bomo uporabljali levi (oz. v nekaterih različicah desni) obrat. Levi obrat je, ko tri točke zavzamejo kot na intervalu [0,π], desni pa, ko kot zavzame vrednosti med (π,2\*π)



**GP**

Grahamov pregled je leta 1972 objavil Ronald Lewis Graham. Zgodovinsko gledano je bil to prvi algoritem za iskanje konveksne ovojnice s časovno zahtevnostjo O(n\*log(n)). Za boljše shranjevanje podatkov si pomagamo s skladom.

**Potek:**

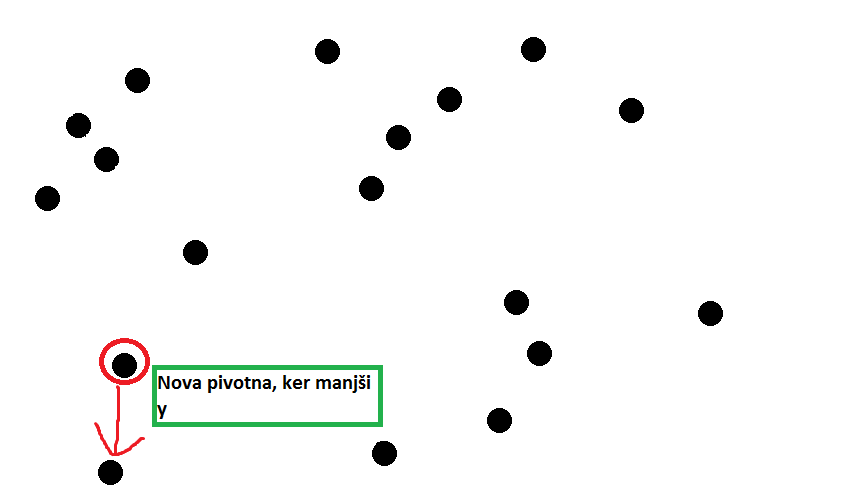
Najprej izberemo pivotno točko. Pivotna točka je točka, ki je zagotovo v konveksni ovojnici. Za tako točko obstaja več možnosti: *POKAŽI NA PREJ NARISANEM PRIMERU*

* Najmanjša y in najmanjša x koordinata
* Najmanjša y in največja x koordinata
* Največja y in najmanjša x koordinata
* Največja y in največja x koordinata

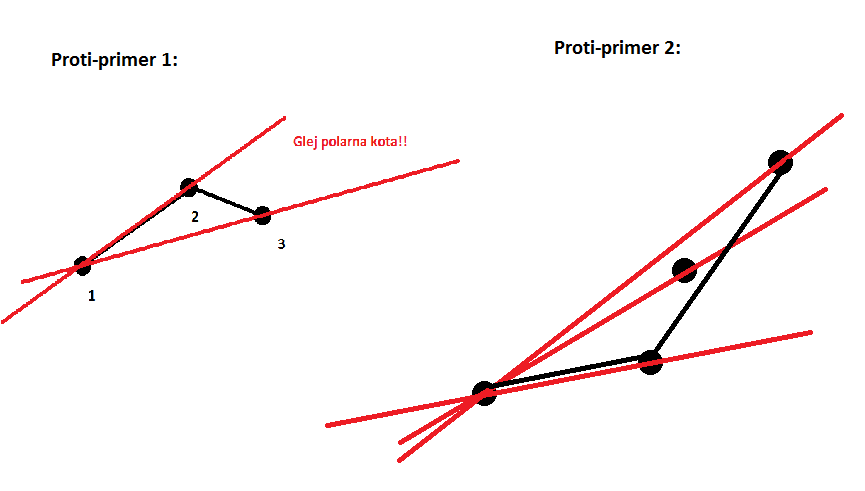
Običajno za pivotno točko izberemo tisto, ki ima najmanjši y. Če ima več točk enak y, vzamemo tisto z najmanjšim x-om. *ZAČNI NOV PRIMER*

**

Čez pivotno točko in vse ostale točke, narišemo premice. Točke lahko zložimo po vrsti glede na to, kakšen polarni kot zajema premice čez točko z x-osjo. Če ima več točk enak polarni kot, lahko odstranimo vse, razen tiste, ki ima največjo abs(x) koordinato, saj je ta najboljši kandidat za to, da je na konveksni ovojnici **(Pazi na tri točke z enakim polarnim kotom)**. Vemo še, da mora biti polarni kot na intervalu [0, π), saj bi drugače izbrali drugo pivotno točko. *POKAŽI PRIMER ZA TO*



Sedaj ustvarimo sklad. Sklad nam bo pomagal zgolj kot pomožna podatkovna struktura, kot nekakšen kup, kamor bomo odlagali za konveksno ovojnico primerne točke. Na sklad postavimo prve tri točke, torej pivotno točko in prvo in drugo točko iz urejenega seznama. Te točke zagotovo delajo levi obrat, saj bi bile drugače razporejene na seznamu (proti-primer 1).

**

*OPOMBA:* Če so točke na prvem koraku v skladu, še ne pomeni da so tudi v drugem(proti-primer 2). V skladu so zagotovo točke p0, p1 in pm (zadnja točka iz urejenega seznama).

Sedaj lahko naredimo zanko čez vse ostale točke v urejenem seznamu. Tega se lotimo tako, da za vsako točko iz preostanka seznama vzamemo vrhnji točki iz sklada. Pogledamo kakšen obrat tvorita točki iz sklada s trenutno točko. Če je obrat levi, točko damo na sklad, drugače se premaknemo naprej. Ko pridemo čez cel seznam točk, lahko vrnemo sklad. **Bodi pozoren na primer ko padejo tri točke iz ovojnice.** Na koncu so v skladu vsi elementi konveksne ovojnice.

OPOMBA: Celoten postopek bi lahko delali tudi v drugo smer. Na začetku bi kote razporedili od največjega proti najmanjšemu. Med preverjanjem, pa bi iskali desni obrat (levi ne bi bil v redu). Od tu izhaja, da je pm v konveksni ( pm je p1 v drugi smeri).

**Psevdokoda**

Podan imamo seznam točk-Q.

GrahamScan(Q):

1. Naj bo p0 točka z najmanjšo y koordinato. V primeru več točk z isto y koordinato, vzemi tisto z najmanjšo x koordinato.

2. Naj bodo {p1, … , pm} ostale točke iz Q razporejene po polarnem kotu v obratni smeri urinega kazalca okoli točke p0. Če ima več točk enak polarni kot, odstrani vse, razen , ki ima največjo abs(x) koordinato glede na p0.

3. Ustvarimo prazen sklad S.

4. Na sklad damo p0.

5. Na sklad damo p1.

6. Na sklad damo p2.

7. Za i med 3 in m:

8. Pogledamo kakšen obrat je med pi, vrh(S) in »zraven« vrha(S):

Če obrat ni levi:

9. pi na sklad ne vstavimo

10. Drugače: pi na sklad vstavimo

11.vrnemo sklad

**Časovna zahtevnost**

Pokažimo, da je časovna zahtevnost Grahamovega algoritma enaka O(n\*log(n)), kjer je n = |Q|. Korak 1 porabi O(n) časa. Korak 2 porabi O ( n\*log(n)) časa, ker uporabljamo »mergesort« ali »heapsort«, da razporedimo in primerjamo polarne kote (odstranjevanje vseh, razen najbolj oddaljene točke z istimi polarnimi koti, je lahko storjeno v O(n) časa). Koraki 3-6 nam vzamejo O(1) časa. Ker je m <= n-1, je for zanka za korake 7-10 opravljena največ (n-3)-krat. Ker preverjanje v vrstici 8 in dajanje elementa na sklad vzame O(1), nam vsaka ponovitev v for zanki vzame O(1) časa for zanka vzame O(n) časa.

Torej je pretečeni čas Grahamovega pregleda enak O (n\*log(n)), ker urejanje elementov dominira dejanski algoritem iskanja konveksne ovojnice.

**Algoritmi za IKO**

Za iskanje konveksne ovojnice poleg Grahamovega pregleda poznamo še nekaj drugih algoritmov. Tu so napisani vsi, skupaj z njihovo časovno zahtevnostjo.

Algoritem zavijanja darila (Jarvisov obhod) (O(nh)), Quick hull (O(n\*log(n))), Divide and Conquer(O(n\*log(n))), Andrew's Monotone Chain Algorithm(O(n\*log(n))), Incremental CH Algorithm(O(n\*log(n))), The Ultimate Planar CH Algorithm(O(n\*log(h))), Chan's Algorithm(O(n\*log(h))).

Pri tem bi posebej omenil Andrew's Monotone Chain Algoritm in Chanov Algoritem. Prvi deluje na podobnem principu kot Grahamov pregled. Točke razporedimo glede na koordinato x (če isti x, še na y), potem pregledujemo če dobimo levi obrat itd.

Chanov algoritem je trenutno najboljši (in tudi precej preprost) algoritem za iskanje konveksne ovojnice (tudi v prostoru). Ta algoritem za reševanje iskanja konveksne ovojnice v ravnini uporablja kombinacijo Grahamovega pregleda in Jarvisovega obhoda, tako da dobi O(n\*log(h)). Množico točk razdeli na m enako velikih delov. V vsakem delu z Grahamovim pregledom najde konveksno ovojnico tistega dela, nato pa dele združi med seboj z Jarvisovim obhodom.