3.1. Úvod do výpočtovej geometrie (Computational Geometry)

Disciplína výpočtovej geometrie vzniká koncom 70-tich rokov. Pokrýva geometrické algoritmy určené na spracovanie geometrických objektov ako body, úsečky, priamky, kružnice a mnohé iné. Stáva sa pravidlom, že väčšinu úloh, ktorými sa zaoberá, vyrieši človek, s papierom a rysovacími potrebami, omnoho jednoduchšie ako počítač. Keď sa ale vyžaduje spracovávanie veľkého množstva dát, počítače tvoria neodmysliteľnú pomôcku.

Už zdanlivo triviálne geometrické úlohy vyžadujú netriviálne dlhý kód. Nespočetné singulárne prípady ako napr. nenápadné kolineárnosti bodov, degenerované uhly či splývajúce úsečky, spôsobujú vo väčšine prípadov fatálne ťažkosti. Pri návrhu geometrických algoritmov sa najskôr tieto úskalia zanedbajú, čo napomáha lepšie pochopiť samotný algoritmus, avšak v druhej fáze návrhu treba zohľadniť *všetky* degenerované prípady(degenerate cases), teda akési rozloženia objektov, ktoré nie sú celkom všeobecné, napr. už spomínané kolineárnosti spracúvaných objektov. Geometrické úlohy sú natoľko rozmanité, že zanedbanie už len jedného špeciálneho prípadu vedie, napriek zdaniu, k úplne nefunkčnému algoritmu na skoro každom vstupnom rozložení.

Nakoniec treba zobrať na zreteľ, že počítače „nie sú dokonalé“ a predsa len pracujeme v obmedzenom svete, ktorý *nevie* počítať s reálnymi číslami s absolútnou presnosťou! V tretej fáze návrhu sa teda zameriavame na robustnosť a stabilitu našich implementácií. Na to, ako celý náš výpočet ovplyvnia nepredpokladané chyby v zaokrúhľovaní medzivýsledkov. Napr. takmer vždy, keď sčítame nejaké desatinné čísla, ktorých súčet by pri absolútnej presnosti mal byť 0, dostaneme len niečo blízke číslu 0, niekedy dokonca so záporným znamienkom!

3.1.1. Dôležitá analytická geometria

Aby sme mohli na realizáciu nejakých geometrických algoritmov vôbec pomýšľať, treba mať určitú zručnosť v analytickej geometrii, čiže byť schopný jednoduché geometrické vlastnosti objektov počítať naslepo – nie znázornením v rovine, ale len výpočtom z ich súradníc (analytického vyjadrenia)!

Zameriame sa na dvojrozmernú euklidovskú rovinu E_2. Bod v rovine je určený svojimi súradnicami – usporiadanou dvojicou $[x, y]$. Uvažujme dva body $A = [a_x, a_y]$ a $B = [b_x, b_y]$. Body $[x, y] \in E_2$ nachádzajúce sa na priamke $\overleftrightarrow{AB}$ spĺňajú rovnosť

$$(a_y - b_y)x + (b_x - a_x)y + (a_x b_y - a_y b_x) = 0$$

Teda ak sa bod roviny nachádza na priamke $\overleftrightarrow{AB}$, tak jeho súradnice $[x, y]$ spĺňajú uvedenú rovnosť, inak uvedenú rovnosť nespĺňajú. Opačne, ak nejaký bod so súradnicami $[x, y]$ uvedenú rovnosť spĺňa, tak sa nachádza na priamke $\overleftrightarrow{AB}$, inak sa na nej nenachádza.

Uvažujme teraz dve priamky *p* a *q*, dané nasledujúcimi rovnicami:

$$p : a_1 x + b_1 y + c_1 = 0$$
$$q : a_2 x + b_2 y + c_2 = 0$$

Určíme, či sú priamky rovnobežné(alebo totožné) alebo (v prípade, ak sú rôznobežné) zistíme ich priesečník. Zaveďme si najskôr spôsob výpočtu determinantu matice tvaru *2x2*:

$$\mathrm{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad |\mathrm{A}| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Na zistenie uvedených vlastností priamok si vypočítame nasledujúce determinanty:

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \qquad D_1 = \begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix} \qquad D_2 = \begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}$$

Potom, ak $D = 0$, tak sú priamky rovnobežné (ešte nevieme, či sú aj totožné), inak ($D \neq 0$), tak $X = [\frac{D_1}{D}, \frac{D_2}{D}]$ je priesečník priamok *p* a *q*. Ak dve rovnobežné priamky majú spoločný bod, tak sú totožné.

Vektor ***u*** je usporiadaná dvojica $\boldsymbol{u} = (u_1, u_2)$, ktorá predstavuje akýsi posun, smer. Nech máme bod $A = [a_x, a_y]$, bod *A* posunutý o vektor ***u*** je bod so súradnicami $[a_x + u_1, a_y + u_2]$. Všimnime si, že aj bod $[x, y]$ je vlastne vektor. Majme teraz dva body $A = [a_x, a_y]$ a $B = [b_x, b_y]$, chceme nájsť vektor ***u*** taký, že posunutím bodu *A* o vektor ***u*** dostaneme bod *B*. Jednoduchým výpočtom dostaneme $\boldsymbol{u} = (b_x - a_x, b_y - a_y)$.

Uvažujme teraz priamku *p* danú rovnicou $p : a_1 x + b_1 y + c_1 = 0$, potom ***s*** (smerový vektor – nejaký vektor rovnobežný s jej smerom) a ***t*** (kolmý vektor – nejaký vektor kolmý na smer ***s***) vypočítame nasledovne:

$$\boldsymbol{s} = (-b_1, a_1), \quad \boldsymbol{t} = (a_1, b_1)$$

Definujme normu $\|\boldsymbol{u}\|$, vektora $\boldsymbol{u} = (u_1, u_2)$, ako $\|\boldsymbol{u}\| = \sqrt{u_1^2 + u_2^2}$. Uvažujme vektory $\boldsymbol{u} = (u_1, u_2), \boldsymbol{v} = (v_1, v_2)$, uhol α, ktorý zvierajú tieto vektory vypočítame z rovnice $u_1 v_1 + u_2 v_2 = \|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\| \cos\alpha$. Všimnime si, ak sú vektory ***u*** a ***v*** kolmé, platí $u_1 v_1 + u_2 v_2 = 0$, a opačne. Uhol dvoch priamok je rovnaký uhlu ich smerových vektorov. Obsah rovnobežníka, ktorý určujú vektory ***u*** a ***v*** bude $|S|$:

$$S = \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix}$$

Tento poznatok nám postačuje aj na výpočet obsahu trojuholníka daného troma bodmi *A*, *B* a *C*, tak že si zostrojíme vektory $\boldsymbol{u} = B - A, \boldsymbol{v} = C - B$ a obsah rovnobežníka, ktorý určujú je práve dvojnásobok hľadaného obsahu trojuholníka. Všimnime si, že správny obsah trojuholníka bude $\frac{|S|}{2}$, pričom znamienko *S* je dané tým, či pri „cestovaní“ po bodoch $A \to B \to C$ sa točíme doľava ($S > 0$), doprava ($S < 0$), alebo sa netočíme ($S = 0$). Toto nám umožňuje zisťovať dôležitú vlastnosť (ľavo/pravo točivosť, kolineárnosť) trojice bodov!

Vzdialenosť bodu $M = [m, n]$ od priamky $p : ax + by + c_1 = 0$ a vzdialenosť dvoch rovnobežných priamok *p* a $q : ax + by + c_2 = 0$ vypočítame nasledovne:

$$|M, p| = \frac{|am + bn + c_1|}{\sqrt{a^2 + b^2}} \qquad |p, q| = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}}$$

Tieto základné poznatky nám vystačia na zostrojenie niekoľkých jednoduchých geometrických algoritmov.

3.2. Konvexné obaly (Convex hulls)

Pri spracovaní väčšieho množstva bodov sa zaujímame o ich rozsah či hranice. Človek, pozerajúci na množinu bodov vyznačených v rovine, bez vážnych ťažkostí rozozná body ležiace vo vnútri od bodov ležiacich na okraji danej množiny bodov. Tento rozdiel zohráva vo výpočtovej geometrii dôležitú úlohu.

Matematicky túto prirodzenú hranicu nazývame *konvexný obal*$_{(convex\ hull)}$. Konvexný obal množiny bodov v rovine definujeme ako najmenší konvexný mnohouholník (útvar, v ktorom spojnica každých dvoch jeho vnútorných bodov leží celá v ňom), ktorý ich všetky obsahuje. Ekvivalentne, je to mnohouholník s najmenším obvodom spomedzi všetkých takých konvexných mnohouholníkov, ktoré obsahujú danú množinu bodov.

Zrejme vrcholy obalu budú niektoré z bodov pôvodnej množiny. Hľadanie konvexného obalu potom spočíva v nájdení týchto bodov. Tento pojem sa prirodzene rozširuje aj do vyšších rozmerov, avšak úloha je potom zväčša prekomplikovaná a jej využitie diskutabilné. Pri našich ďalších úvahách sa preto zameriame len na konvexné obaly v dvojrozmernej euklidovskej rovine.

3.2.1. Hľadanie konvexného obalu

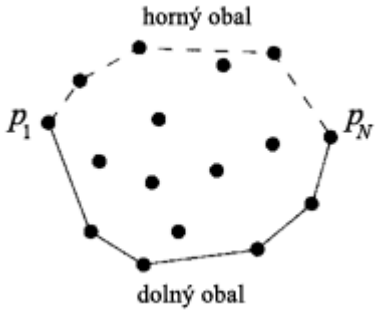
Dôležitou vlastnosťou konvexného obalu je, že každá priamka nepretínajúca obal sa ho, pri pohybe z pomyselného nekonečna smerom k nemu, dotkne v nejakom jeho vrchole. Týmto spôsobom vieme poľahky nájsť niektoré body, ktoré sa istotne nachádzajú na obale. Stačí keď si zoberieme vodorovnú a zvislú priamku – body s najmenšou a najväčšou x-ovou a y-ovou súradnicou sú všetky na obale.

Spôsobov hľadania konvexného obalu je veľa, medzi najznámejšie patria *Grahamov prechod*$_{(Graham\ scan)}$, *Jarvisov pochod*$_{(Jarvis's\ march)}$ – známe aj ako metóda *Balenia balíčka* (*Package-Wrapping*) a *QuickHull* (funguje podobne ako *QuickSort*). My sa budeme zaoberať jednou z inkrementálnych metód, ktorá využíva len jednoduché výpočty (sčítanie a násobenie), ktoré sa napr. v prípade bodov, ktoré majú celočíselné súradnice, potom všetky odohrávajú len v celých číslach. Na reprezentáciu bodov využijeme nasledovnú štruktúru:

```
type point = record
  x,y:integer;
end;
```

```
typedef struct
{ int x,y;
} point;
```

Daná je teda vstupná množina bodov $\boldsymbol{p_1}$, $\boldsymbol{p_2}$, **…**, $\boldsymbol{p_N}$, chceme nájsť jej konvexný obal (vymenovaním jeho vrcholov po obvode v protismere hodinových ručičiek) $\boldsymbol{o_1}$, $\boldsymbol{o_2}$, ..., $\boldsymbol{o_M}$. Vstupné body najskôr lexikograficky utriedime – najskôr podľa *x*-ovej súradnice, body s rovnakou *x*-ovou podľa *y*-ovej – do vzostupného poradia, čím dostaneme postupnosť $\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_N}$.



Náš algoritmus potom spočíta výsledný konvexný obal zlúčením dvoch menších: *dolného obalu* – časť obalu z $\boldsymbol{p_1}$ do $\boldsymbol{p_N}$ pri usporiadaní vrcholov proti smeru hodinových ručičiek, *horného obalu* – časť obalu v opačnom smere z $\boldsymbol{p_N}$ do $\boldsymbol{p_1}$. Zameriame sa na výpočet dolného obalu, horný obal sa vypočíta analogicky.

Algoritmus postupne pridáva body vstupnej množiny, pričom po každom pridaní si udržiava čiastočný konvexný obal doposiaľ pridanej časti bodov. Kľúčový krok je preto pozmenenie obalu po pridaní bodu $\boldsymbol{p_i}$.

Inými slovami, daný je konvexný obal bodov p_1, …, p_{i-1}, musíme vypočítať konvexný obal množiny p_1, …, p_i. Predpokladajme, že pridanie bodu p_i do konvexného obalu máme vyriešené (procedúra `insert`), potom implementácia nášho inkrementálneho algoritmu (p_1, p_2, …, p_N *sú už zotriedené!*) vyzerá nasledovne:

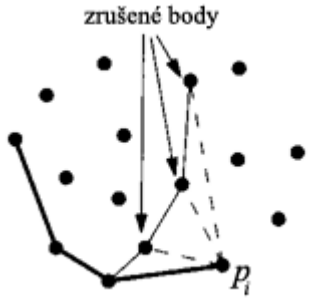
```
var p:array[1..maxn] of point;
    o:array[1..2*maxn] of point;
    np,no:integer;

procedure hull;
var i:integer;
begin
  { začíname }
  o[1]:=p[1]; o[2]:=p[2];
  no:=2;
  { dolný obal }
  for i:=3 to np do
    insert(i);
  { horný obal }
  for i:=np-1 downto 1 do
    insert(i);
  { obal sú body o[1..no] }
end;
```

```
point p[maxn];
point o[2*maxn];  /* obal */
int np, no;

void hull(void)
{
  int i;
  /* začíname */
  o[0]=p[0]; o[1]=p[1];
  no=2;
  /* dolný obal */
  for(i=2;i<np;i++)
    insert(i);
  /* horný obal */
  for(i=np-2;i>=0;i--)
    insert(i);
  /* obal sú body o[0..no-1] */
}
```

Teraz doriešime pridanie ďalšieho bodu vstupnej množiny. Uvažujme, pri obchádzaní obvodu mnohouholníka sa v každom vrchole točíme buď doľava alebo doprava (alebo ideme rovno). Pre ľubovoľný mnohouholník sa môžeme točiť rôzne, ale pre konvexný mnohouholník by sme sa mali točiť vždy jedným smerom (pre dolný obal: doľava).



Bod p_i pridáme, pretože na čiastočnom obale určite bude. Opakovane sa teraz pozrieme na poslednú trojicu bodov na konvexnom obale. Pokiaľ pri prechode z o_{j-2} cez o_{j-1} do o_j sa točíme doprava, znamená to, že bod o_{j-1} je vnútorný bod iného konvexného mnohouholníku a preto z definície nemôže byť na výslednom konvexnom obale bodov p_1, …, p_i. Preto ho zrušíme a postup opakujeme až pokým sme z obalu nezrušili všetky body, ktoré tam nepatria – čím dostaneme platný konvexný obal. Nasledujúca implementácia je priamočiari prepis týchto myšlienok:

```
function ccw(a,b,c:point):integer;
begin
  { <0 doprava, =0 rovno, >0 doľava }
  ccw:=(c.x-a.x)*(c.y-b.y)-
       (c.x-b.x)*(c.y-a.y);
end; { vekotorový súčin }

procedure insert(i:integer);
begin
  { pokým sa točíme doprava }
  while (no>=2) and
   (ccw(o[no-1],o[no],p[i]) < 0)
      do dec(no);
  { nový bod }
  inc(no); o[no]:=p[i];
end;
```

```
int ccw(point a, point b, point c)
{
  /* <0 doprava, =0 rovno, >0 doľava */
  return(c.x-a.x)*(c.y-b.y) -
        (c.x-b.x)*(c.y-a.y);
} /* vekotorový súčin */

void insert(int i)
{
  /* pokým sa točíme doprava */
  while (no>=2 &&
    ccw(o[no-2],o[no-1],p[i]) < 0)
      no--;
  /* nový bod */
  o[no++]=p[i];
}
```

3.2.2. Zložitosť hľadania konvexného obalu

Časová náročnosť nášho postupu je zjavne *O(N* log *N)*, pretože každý bod vstupnej množiny do nášho pracovného konvexného obalu najviac raz pridáme a najviac raz ho z neho zrušíme. Od algoritmu na hľadanie konvexného obalu vyžadujeme, aby zistil vrcholy obalu v poradí v akom na ňom ležia.

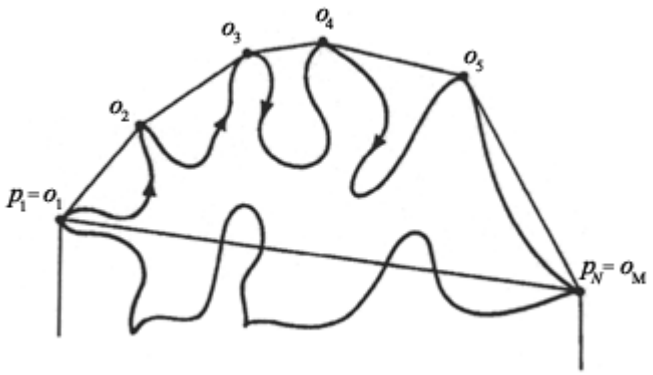
Uvažujme, že máme utriediť čísla $x_1, x_2, \ldots, x_N$, transformujme ich do roviny ako body $q_1, q_2, \ldots, q_N$, kde $q_i = (x_i, x_i^2)$ čo zodpovedá parabole. Zrejme sa všetky tieto body nachádzajú na konvexnom obale (parabola je konvexná), a ich poradie na obvode určuje ich usporiadanie.

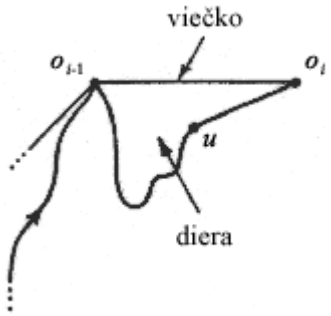
Každý algoritmus na výpočet konvexného obalu dokáže preto aj triediť. Z tohto vyplýva, že ak by sme vedeli vytvoriť konvexný obal rýchlejšie ako v čase *O(N* log *N)*, vedeli by sme aj rýchlejšie triediť, čo sa ale nedá a preto uvedená zložitosť pre konvexný obal je najlepšia možná.

Otázkou je, či by to, v nejakých konkrétnych prípadoch, nešlo aj lepšie. Vezmime si napríklad daný mnohouholník (nekonvexný), ako rýchlo dokážeme určiť jeho obal? Jedna možnosť je zabudnúť na to, že dané body sú jeho vrcholy zadané po obvode, a vypočítať konvexný obal týchto bodov ako neusporiadanej množiny, čo by trvalo čas *O(N* log *N).* Ak však využijeme vedomosť, že máme dané poradie bodov, ktoré tvoria nepretínajúci sa mnohouholník, dá sa to aj lepšie.

3.2.3. Konvexný obal mnohouholníka

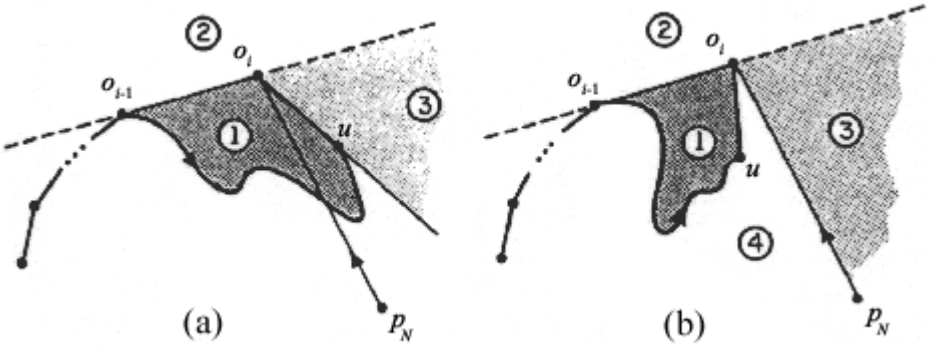
Najskôr si musíme uvedomiť, že prístup, ktorý sme využívali pri hľadaní obalu neusporiadanej množiny bodov sa, bez triedenia, nedá použiť. Treba preto vymyslieť niečo dôvtipnejšie. Rovnako ako predtým, konštrukciu rozdelíme do dvoch etáp – *horný* a *dolný* obal. Zamerajme sa na horný obal. Nekonvexné oblasti mnohouholníka budeme považovať za diery, ktoré treba hranami konvexného obalu pozakrývať.





Algoritmus bude prechádzať bodmi v poradí $p_1, p_2, \ldots, p_N$, pričom bude konštruovať postupnosť viečok, ktoré zakrývajú nekonvexnosti (diery) mnohouholníka. Postup je, rovnako ako predtým, inkrementálny – máme konvexný obal (postupnosť viečok) vrcholov $p_1, \ldots, p_i$, chceme určiť obal vrcholov $p_1, \ldots, p_{i+1}$. Dôležitý je preto krok rozšírenia existujúceho riešenia o ďalší vrchol.

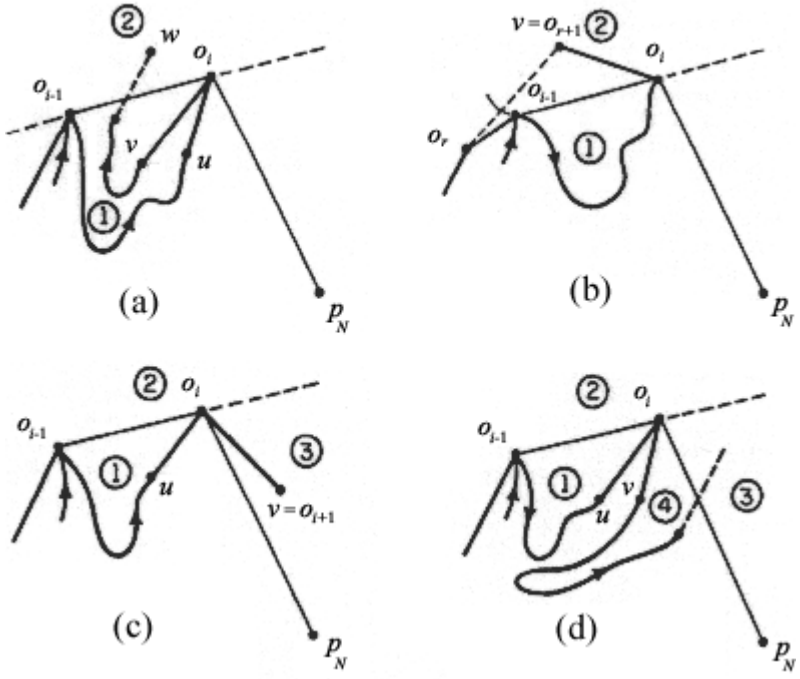
Keď pridávame vrchol, situácia vyzerá (obr. vpravo) tak, že sme uzavreli dieru ďalším viečkom. Prechod obvodu mnohouholníka pokračuje teda za vrcholom o_i. Označme u vrchol, ktorý predchádza vrcholu o_i na obvode mnohouholníka, potom rozlíšime dva prípady, závislé na relatívnej polohe bodu u vzhľadom na úsečku $\overrightarrow{p_N o_i}$.



a) $\boldsymbol{u}$ sa nachádza *na* alebo *napravo* od $\overrightarrow{\boldsymbol{p_N o_i}}$. V tomto prípade rozlišujeme tri oblasti označené **1**, **2**, **3** definované intuitívne podľa obrázku – rozdelené priamkou $\overleftrightarrow{\boldsymbol{o_{i-1} o_i}}$, polpriamkou $\overrightarrow{\boldsymbol{o_i u}}$ a okrajom zakrytej diery.
b) $\boldsymbol{u}$ sa nachádza *naľavo* od $\overrightarrow{\boldsymbol{p_N o_i}}$. V tomto prípade nám vzniká ešte jedna oblasť **4**.

Označme $\boldsymbol{v}$ vrchol, ktorý nasleduje za vrcholom $\boldsymbol{o_i}$ na obvode mnohouholníka. Podľa toho, v ktorej oblasti sa nachádza bod $\boldsymbol{v}$, algoritmus bude vykonávať nasledovné úkony:

a) $\boldsymbol{v} \in$ (oblasť č.1.): V tomto prípade, hranica mnohouholníka pokračuje cez dieru, budeme ju nasledovať, pokým nevstúpi do oblasti č.2. (vrchol $\boldsymbol{w}$). Vrchol $\boldsymbol{w}$ budeme považovať za vrchol $\boldsymbol{v}$, ktorý je v oblasti č.2. (nasledujúca diskusia).
b) $\boldsymbol{v} \in$ (oblasť č.2.): Zjavne $\boldsymbol{v}$ sa nachádza v obale našich bodov, takže vrchol $\boldsymbol{v}$ pridáme do obalu a body, nachádzajúce sa vnútri (na obrázku: $\boldsymbol{o_{r+1}}$, **…**, $\boldsymbol{o_i}$) z obalu vylúčime.
c) $\boldsymbol{v} \in$ (oblasť č.3.): $\boldsymbol{v}$ bude ďalší vrchol obalu, pretože sa nachádza smerom doprava od $\overrightarrow{\boldsymbol{o_{i-1} o_i}}$.
d) $\boldsymbol{v} \in$ (oblasť č.4.): Hranica vstupuje do vnútra konvexného obalu, preto ju stačí nasledovať, až pokým nepríde do vrcholu $\boldsymbol{p_N}$ (algoritmus končí) alebo nevkročí do oblasti č.3.



Takto vygenerujeme hornú časť obalu. Podobným postupom v smere $\boldsymbol{p_N}$, $\boldsymbol{p_{N-1}}$, **…**, $\boldsymbol{p_1}$ vytvoríme aj dolný obal. Ich spojením vznikne konvexný obal, ktorý je už výsledkom nášho algoritmu. Keďže algoritmus pracuje na každom vrchole mnohouholníku len konštantný čas, nájdenie konvexného obalu sa dá realizovať v optimálnom čase *O(N)* a pamäti *O(N)*.

Implementácia popísaných myšlienok je jednoduchá a je odporúčaná ako ľahké cvičenie v programovaní.