



**OPPA European Social Fund  
Prague & EU: We invest in your future.**

---

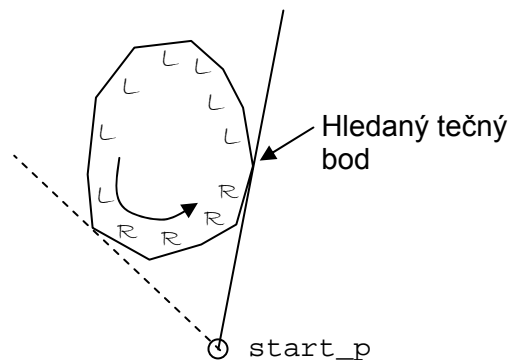
## Tipy k úloze číslo 3 – Chanově algoritmu

### ***Měkolik poznámek k vlastnímu binárnímu vyhledávání***

Pro každou z malých obálek, které jsme vypočítali Grahamovým algoritmem, potřebujeme najít tečnu z naposledy přidaného bodu do výsledné konvexní obálky. Z těchto tečen k malým obálkám určíme tu, která proti směru hodinových ručiček zatočí nejméně – z ní se stane další bod výsledné konvexní obálky (Viz slajd 29 ve 4. přednášce).

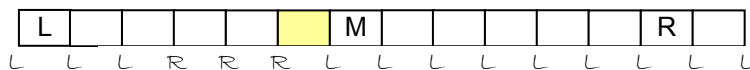
### **1. Co platí pro tečný bod a proč tento test nestačí pro určení, zda v dalším kroku jít doleva či doprava?**

Označíme naposledy přidaný bod jako  $start\_p$ . V našem případě jsou body malé KO uloženy v poli. Při sekvenčním hledání tečného bodu by stačilo najít vrchol, v němž je  $start\_p$  od příchozí úsečky napravo a od odchozí nalevo. Pro všechny úsečky malé KO před tečným bodem je  $start\_p$  napravo, za tečným bodem je nalevo (viz Obrázek 1, kde jsou úsečky malé konvexní obálky označeny psacími písmeny  $L$  a  $R$ ). Pokud tedy začneme od nějakého vrcholu a jeho příchozí i odchozí úsečka je označena písmenem  $L$  ( $start\_p$  je od ní nalevo) stačí postupovat CW a hledaný tečný bod nemůžeme minout. Obdobně CCW pro segmenty označené  $R$ .



**Obrázek 1** Tečný bod a poloha bodu  $start$  nalevo ( $L$ ) a napravo ( $R$ ) od jednotlivých úseček

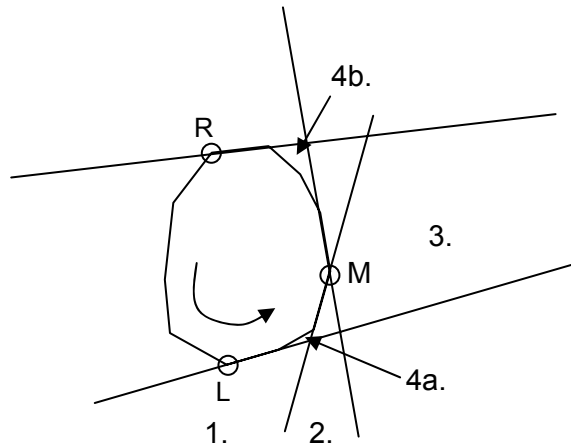
Při půlení se díváme pouze na levý ( $L$ ), pravý ( $R$ ) a vypočítaný prostřední bod ( $M$ ) a snažíme se rozhodnout, zda pokračovat nalevo či napravo od  $M$ . Obrázek 2 znázorňuje situaci, kdy je hledaný tečný bod v levém podintervalu, ale pouze podle směrů přicházejících a odcházejících úsečky v  $M$  bychom „šli doprava“ – po směru hodinových ručiček - CW, tj. přesunuli bychom levý index  $L$  do  $M+1$  a úsek  $R$  minuli. Proto pouze informace u  $M$  nestačí a my musíme brát v úvahu i aktuální polohu bodu  $L$  a  $R$ , resp. Směry odchozí úsečky v  $L$  a příchozí v  $R$  – viz dále”.



**Obrázek 2** Situace, kdy ve vypočítaném prostředním bodě ( $M$ ) navrhne algoritmus pravou polovinu (CW) a přitom by měl jít do levé, neboť v ní je hledaný tečný bod (označen žlutě)

## 2. Určení, ve kterém ze dvou oblouků pokračovat při půlení oblouku mezi L a R.

Představme si situaci podle obrázku. Je na ní jedna malá konvexní obálka – pro dvojici bodů L a R jsme z předchozího postupu algoritmu určili, že máme hledat napravo – v oblouku LR. Vypočetli jsme bod M v polovině oblouku LR a potřebujeme se rozhodnout, jestli budeme při hledání tečného bodu pokračovat v oblouku LM, nebo MR, tj. v levé či pravé polovině.



Obrázek 3 Oblasti ohraničené směry úseček incidujících s body L, M a R

Odpověď bude záviset na tom, ve které oblasti leží bod  $start_p$ .

1. Kdyby  $start_p$  ležel napravo od úsečky nebo na úsečce vycházející z L a zároveň nalevo od úsečky přicházející do bodu M, budeme v hledání tečného bodu pokračovat v intervalu LM, ( $R = M - 1$ )
2. Kdyby  $start_p$  ležel napravo od úsečky nebo na úsečce přicházející do bodu M a zároveň nalevo od úsečky vycházející z bodu M, jsme hotovi, neboť bod M je hledaným tečným bodem
3. Kdyby  $start_p$  ležel na úsečce vycházející z M nebo napravo od této úsečky a zároveň nalevo od úsečky přicházející do bodu R, budeme v hledání tečného bodu pokračovat v intervalu MR, ( $L = M + 1$ )
4. Zbývá rozhodnout situaci, kdyby  $start_p$  ležel zároveň nalevo od všech čtyř zmiňovaných úseček. Protože nemůže ležet uvnitř malé konvexní obálky (na začátku je extrémním bodem a pak vždy tečným), zbývají malé „trojúhelníkové“ oblasti mezi malou konvexní obálkou a dvojicemi úseček z L a do M (oblast 4a) a z M a do R (oblast 4b). Pro bod  $start_p$  v oblasti mezi L a M pokračujeme obloukem LM, pro bod  $start_p$  v oblasti mezi M a R pokračujeme obloukem MR. A jak je od sebe rozlišíme? Stačí zjistit polohu bodu  $start_p$  např. vůči spojnici bodů M a R. Oblast blíže L a M leží od MR nalevo, oblast blíže M a R leží napravo.

Protože pro zjištění směrů z L, do M, z M a do R potřebujeme znát předcházející a následující body, je vhodné se binárním dělením zabývat pouze tehdy, když je bodů mezi L a R včetně alespoň 5. Pokud je bodů méně, splývají se sousedy a trojice bodů z nichž dva jsou totožné jsou vždy kolineární. Vzhledem k dalším singularitám je vhodnější, pokud je bodů alespoň 7.

Pro určení úsečky vycházející z bodu L potřebujeme kromě bodu L ještě bod s indexem  $L+1$ , značíme ho  $L+$ . Obdobně pro určení příchozí úsečky do bodu M potřebujeme znát body M a jeho předchůdce s indexem  $M-1$  - značíme  $M-$ . Viz Obrázek 4.



**Obrázek 4** Indexy bodů, které potřebujeme pro rozhodnutí v binárním vyhledávání (zde pro 14 a 7 bodů)

Příklad testu (poloha bodu  $start\_p$  vůči úsečce M-1,M:

```
if( CGAL::orientation( start_p, *(first+mid-1), *(first+mid)) != CGAL::LEFT_TURN )
```

### 3. Situace, kdy je zkoumaný bod roven bodu $start\_p$

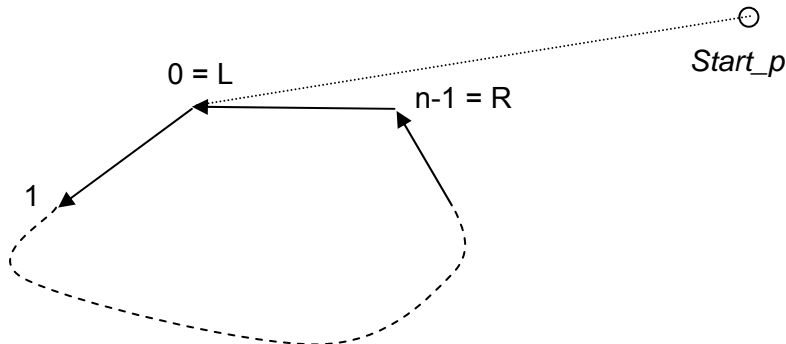
Jde o singularitu – pokud narazíme při prohledávání malé konvexní obálky na bod  $start\_p$ , znamená to, že je tento bod zrovna součástí zkoumané malé obálky. V takovém případě končíme s vyhledáváním a vracíme následující bod. Např. pro L+1 vracíme L+2 a pro bod R vracíme bod L, neboť jde o cyklickou strukturu.

Příklad testu (bod M je totožný s bodem  $start\_p$ )

```
if( equal_points ( start_p, *(first+mid) ) )
    return *(first+mid+1);
```

### 4. Stav na začátku algoritmu

Na začátku algoritmu binárního vyhledávání je jako bod L dosazen první bod pole vrcholů a jako pravý bod R poslední bod pole vrcholů malé KO. Jde o to rozhodnout, zda je hledaný tečný bod oblouku RL, který je tvořen jedinou úsečkou, nebo v oblouku LR, tedy ve zbylé malé KO. Jde o další singularitu – je třeba spočítat, polohu bodu  $start\_p$  vůči trojici úseček s indexy  $(n-2,n-1)$ ,  $(n-1, 0)$  a  $(0,1)$ .



**Obrázek 5** První krok algoritmu - mezi prvním a posledním bodem v poli

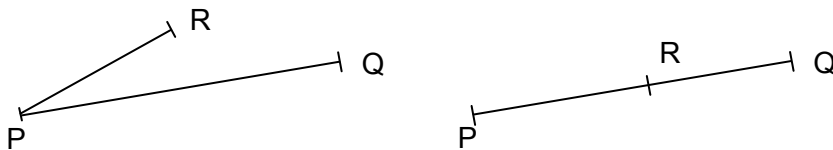
Díky tomu, že vyřešíme konec a začátek pole samostatně, nemusíme již dále uvažovat cykličnost KO a stačí použít klasický algoritmus plnění intervalu – nestane se totiž, že bychom museli přejít z konce na začátek pole a naopak.

### Co se hodí z knihovny CGAL?

Následující kód ukazuje hledání tečného bodu sekvenčně (viz implementace v knihovně CGAL v souboru `ch_jarvis_impl.h`) – jako `rotation_predicate` využíváme `ch_traits.less_rotate_ccw_2_object()`, protože je `true` nejen, pokud pro body P,Q,R je bod Q

nalevo od úsečky PR, ale zároveň zahrnuje i případ, kdy jsou tyto tři body kolineární a vrací true, pokud je bod Q dále od P než bod R (viz Obrázek 6).

```
include <boost/bind.hpp>
#include <algorithm>
// find the point with minimal angle in relation to start_p in O(n)
template <class ForwardIterator, class Point, class Traits>
Point linFindMin( ForwardIterator first, ForwardIterator last, const Point& start_p, Traits
ch_traits )
{
    typedef typename Traits::Less_rotate_ccw_2 Less_rotate_ccw;
    Less_rotate_ccw
        rotation_predicate = ch_traits.less_rotate_ccw_2_object();
    ForwardIterator it = std::min_element( first, last, bind( rotation_predicate,
boost::cref(start_p), _1, _2 ));
    return *it;
};
```



**Obrázek 6** Situace, kdy predikát `less_rotate_ccw_2` vrací true

My ale potřebujeme logaritmickou složitost hledání tečného bodu – proto jsou `first` a `last` deklarovány jako `RandomAccessIterator` a použijeme binární vyhledávání. Kromě `less_rotate_ccw_2_object()` se hodí i `ch_traits.equal_2_object()` – využíváme ho na test, jestli už jsme dosáhli bodu `start_p`, tj. obálka se uzavřela a na test, zda jedním ze zkoumaných bodů není `start_p`. Dále využijeme již známý test na určení polohy bodu vůči úsečce `CGAL::orientation...`

```
// find the point with minimal angle in relation to start_p in O(log n)
template <class ForwardIterator, class RandomAccessIterator, class Point, class Traits>
Point logFindMin( ForwardIterator first, RandomAccessIterator last, const Point& start_p,
Traits ch_traits )
{
    typedef typename Traits::Less_rotate_ccw_2 Less_rotate_ccw;
    typedef typename Traits::Point_2 Point_2;
    typedef typename Traits::Equal_2 Equal_2;

    Less_rotate_ccw less_rotate = ch_traits.less_rotate_ccw_2_object();
    Equal_2 equal_points = ch_traits.equal_2_object();

    if( CGAL::orientation( start_p, *(first+mid_1), *(first+mid)) != CGAL::LEFT_TURN )
    ...

    if( !equal_points( min_p, start_p) ) // end condition - back in the starting point
    {
        *res++ = min_p;
        prev_p = min_p;
    }
    else
        return true; // convex hull closed - we are done
```



**OPPA European Social Fund  
Prague & EU: We invest in your future.**

---