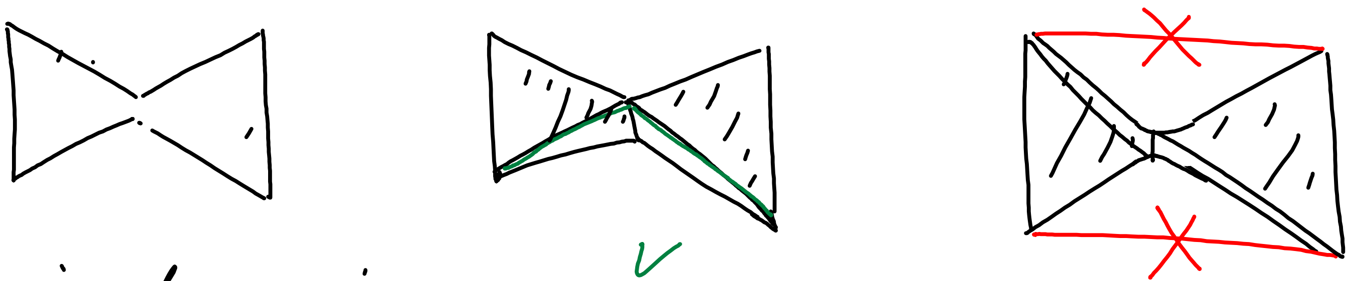
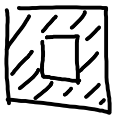
TRIANGULACE MNOHOÚHELNÍKŮ

Máme obecně nekonvexní mnohoúhelník P. Úkolem je rozdělit jej na trojúhelníky, jejichž vrcholy jsou původními vrcholy mnohoúhelníka a jejichž strany procházejí mnohoúhelníkem



P je obecně nekonvexní, ale je jednoduše souvislý
Lze ho stáhnout do bodu

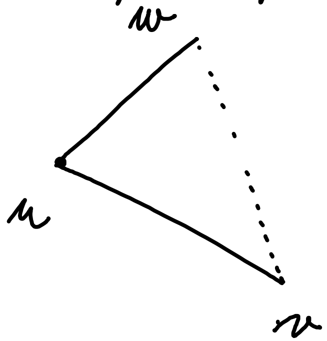


-2-

Věta: Každý jednoduchý rovinný n-úhelník lze triangulovat. Každá triangulace se skládá z $(n-2)$ trojúhelníků

Důkaz provedeme indukcí $n=3$ zřejmé

Nechť to platí pro $n-1 \geq 3$. Vezme vrchol u nejvíce vlevo



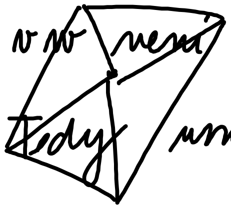
z u vedou 2 hrany mnohoúhelníka do v a do w

1) Úsečka wv leží celá v P

V tomto případě Δ uvw odřízneme a získáme $(n-1)$-úhelník, který lze rozdělit na $(n-1)-2$ trojúhelníků. Tedy P lze rozdělit na $(n-1)-2+1 = n-2$ Δ.

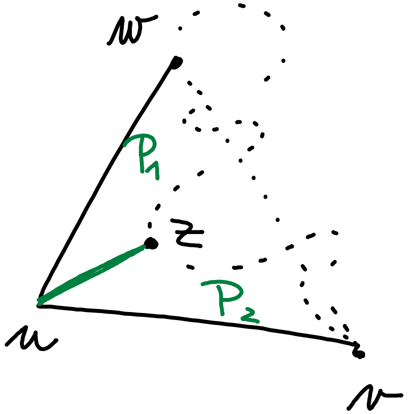
-3-

(2) vw není celá v P



Tedy uvnitř $\Delta\, vwu$ leží nějaký vrchol mnohoúhelníka P

Vezmeme z vrch. těch, který leží nejblíže k u – označme jej z.



Úsečka uz leží celá v P a rozdělují P na n_1-úhelník P_1 a n_2-úhelník P_2

Pro počet Δ v triangulaci P_1 a P_2 platí:

počet Δ v P_1 $\quad n_1-2$

počet Δ v P_2 $\quad n_2-2$

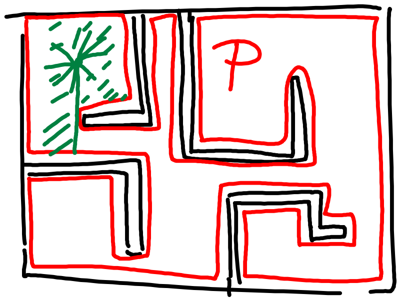
$$n_1+n_2=n+2$$

Počet Δ v P_1 a P_2, tj. v P je roven

$$n_1-2+n_2-2 = n+2-4 = n-2.$$

-4-

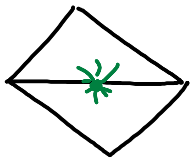
Hlídání umělecké galerie — motivace k triangulaci



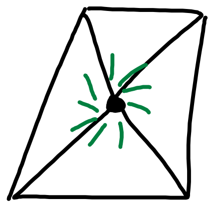
Úkol - rozmístit kamery tak aby jich bylo co nejméně.

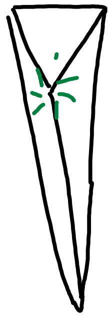
Umíme-li P rozdělit na △, lze toho využít k rozmístění kamer

$(n-2)$ △ ... stačí $(n-2)$ kamer



$\approx \frac{n-2}{2}$ kamer také stačí





Lemma $\left[\frac{n-2}{3}\right]$ kamer stačí

Důkaz: Vrcholy triangulace lze obarvit 3 barvami tak, že vrcholy spojené úsečkou v triangulaci mají různou barvu

lépe: vrcholy △ mají různou barvu.

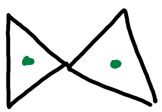
Máme-li takové obarvení, vezmeme vrcholy jedné barvy

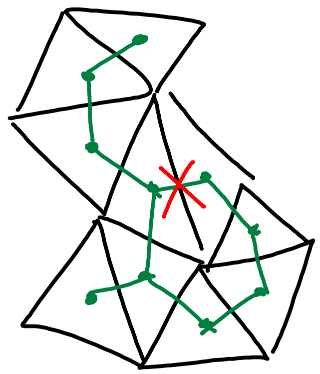
Do těchto vrcholů dáme kamery

-6-

Obarvení vrcholů

Triangulace tvoří rovinný graf. K rovinnému grafu určuji graf duální .. jeho vrcholy jsou △ triangulace
jeho hrany jsou hrany mezi sousedními △





Tento duální graf je souvislý
(Lze z něj vybrat podgraf, který je stromem a obsahuje všechny vrcholy.

Algoritmus, který prochází těmito vrcholy stromu nám umožní postupné obarvení vrcholů △.

-7-

Triangulace n-úhelníka

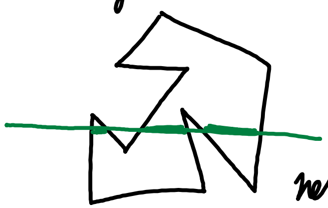
Náš algoritmus bude mít 2 části:

(1) rozdělení mnohoúhelníka na tzv. monotónní části

(2) triangulace monotónního mnohoúhelníka

Rozdělení n-úhelníka na monotónní mnohoúhelníky v čase $O(n \log n)$

Monotónní mnohoúhelník je mnohoúhelník, jehož průnik s vodorovnou přímkou je konvexní množina ($\emptyset$, bod, úsečka)

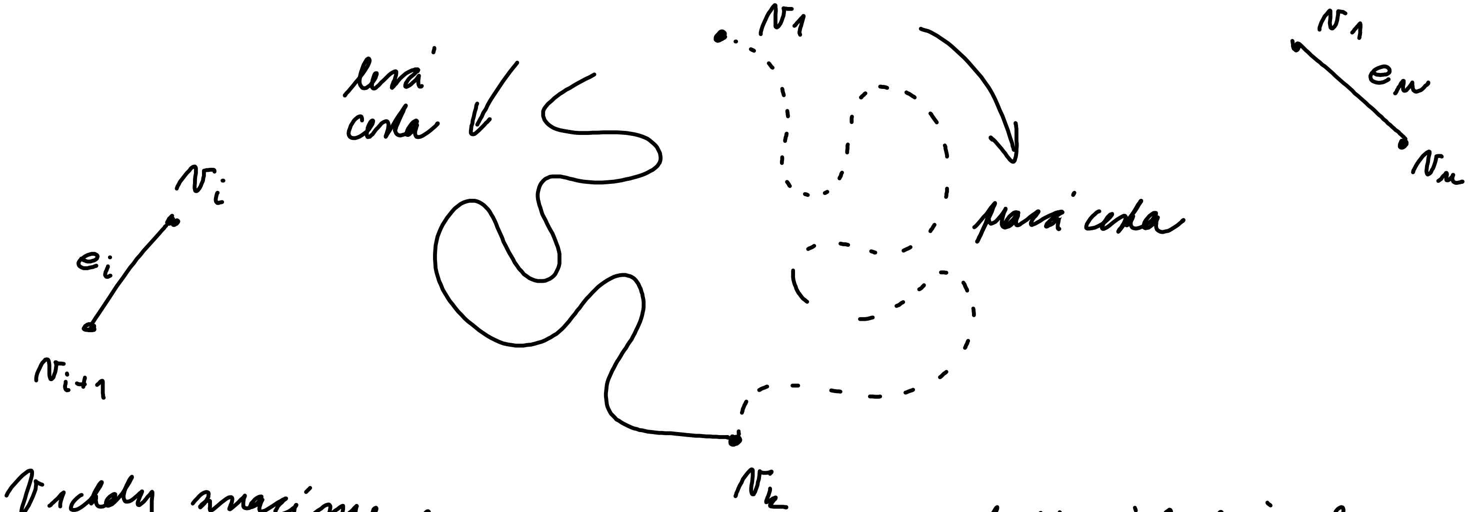


není monotónní

je monotónní

$y = konst$

Mějme nerozhodnutelnost. Nechť nejvyšší uzel je v_1 a nejnižší uzel v_k



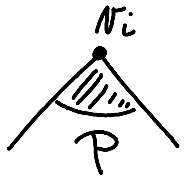
Vrcholy značíme $v_1, v_2, \ldots v_n$ proti směru hodinových ručiček

Hrany $e_1, e_2, \ldots e_n$ po směru hod ručiček

Uvažujme lexikografické uspořádání $p > q$ $\quad p_y > q_y$ nebo $p_y = q_y$ a $p_x < q_x$

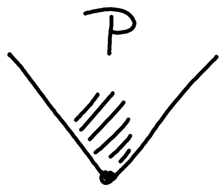
Typy uzlů v mnohoúhelníku

① start



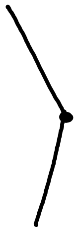
při průchodu levou nebo pravou cestou jdeme nahoru - dolů a P je pod

② end

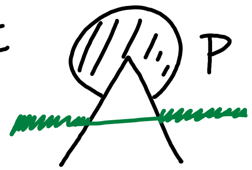


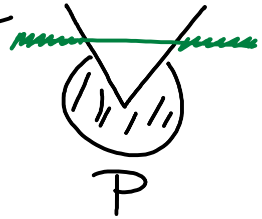
při průchodu levou nebo pravou cestou jdeme dolů - nahoru a P je nad

③ regular



průchod dolů - dolů
(P je vlevo nebo vpravo)

④ split  jdeme nahoru - dolů a P je nad

⑤ merge  jdeme dolů - nahoru a P je pod

Tvrzení: Mnohoúhelník je monotónní právě když neobsahuje žádný vrchol typu split a merge

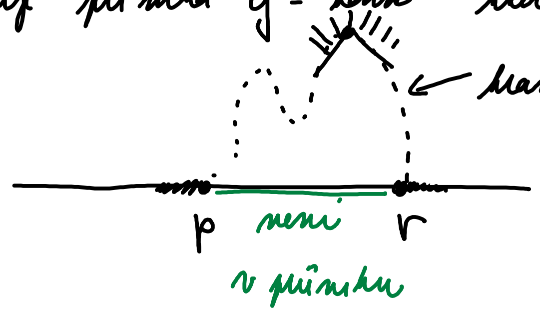
Důkaz. Obsahuje split nebo merge => není monotónní

Nechť P není monotónní.

-11-

Existuje přímka y = konst která protíná P v nekonvexní množině

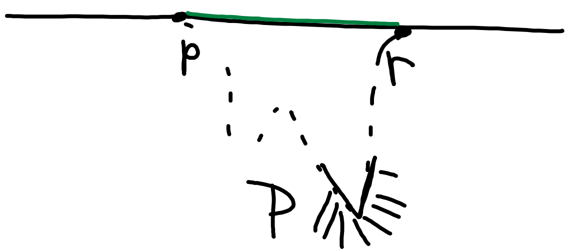
(1)



hranice P mezi p a r je nad přímkou

Potom nejvyšší bod této cesty je typu split

(2)



cesta mezi p a r jde pod přímkou

nejnižší vrchol je typu merge

-12-

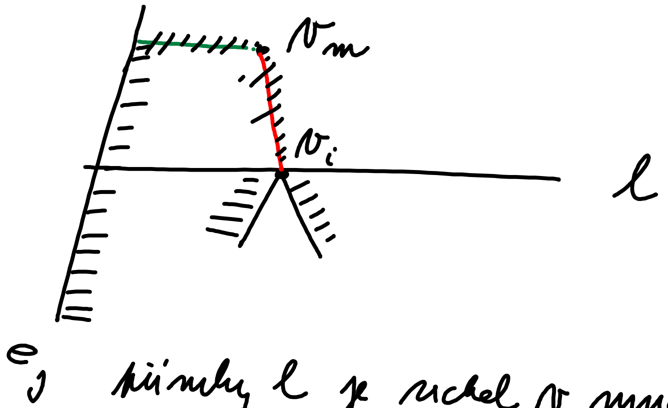
Základní myšlenka algoritmu - přidáváním úhlopříček odstranit vrcholy typu split a merge.

Provádíme metodou zametací přímky, která postupuje shora dolů. Fronta událostí bude dána pouze vrcholy mnohoúhelníka P. Vrcholy typu split odstraňujeme při průchodu zametací přímky, vrcholy typu merge odstraníme až z jednoho následujícího vrcholu

13-

Odstranění split vrcholu

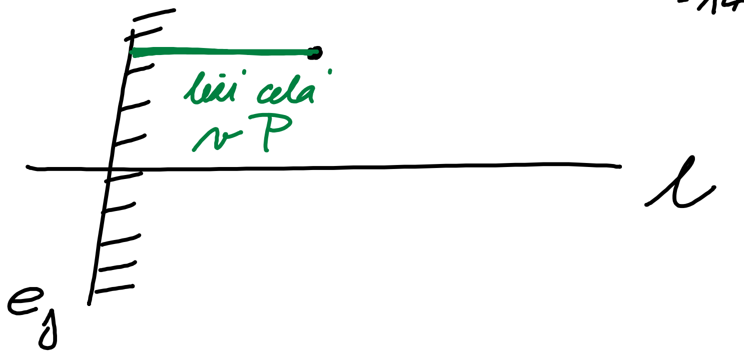
- při průchodu zametací přímky ho spojíme s vrcholem nad zametací přímkou, který „sousedí" s levou hranou od tohoto vrcholu



Definice: Nechť e_j je hrana mnohoúhelníka, která má mnohoúhelník P vpravo. Pomocník (helper) hrany e_j vzhledem k poloze zametací přímky l je vrchol v mnohoúhelníka s těmito vlastnostmi

(1) spojnice (vodorovná) v a e_j leží v P

-14-



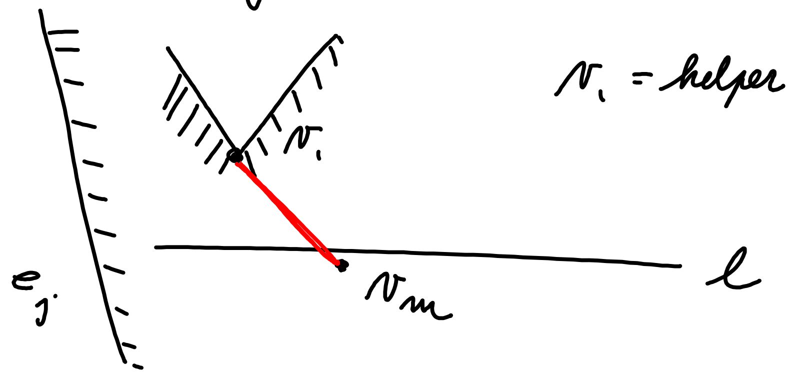
$helper(e_j) = v$

(2) v je nejblíže nad l z těch, co mají vlastnost (1)

Odstranění split uzlu ... znamená spojit split uzel v_i v okamžiku kdy jím ~~…~~ zametací přímka s helperem(e_j), kde e_j je nejbližší hrana mnohoúhelníku, která má mnohoúhelník vpravo.

-15-

<u>Odstranění merge vrcholu</u> v_i se provede při průchodu zametací přímky nejvyšším vrcholem v_m pod v_i, po nějž je v_i helperem okolní hrany e_j



$v_i = helper(e_j)$

-16-

Co je v tomto algoritmu stromem?

Strom T je tvořen hranami mnohoúhelníka, které

(1) protínají zametací přímku

(2) mají P vpravo

Strom udržuje pořadí těchto hran tak, jak jsou protínány zametací přímkou zleva doprava

Ke každé hraně držíme ve stromu její helper. Ten se samozřejmě při přechodu zametací přímky nějakou událostí může měnit.

-17-

Struktura algoritmu – viz pseudo pdf (str 7?)

Popis algoritmu v jednotlivých typech uzlů

– uzel spojujeme s jiným uzlem

– odstraňujeme e_j ze stromu

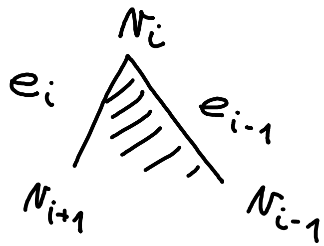
– přidáváme hranu do stromu

– měníme helper některých hran

① Start

– přidáme e_i do stromu

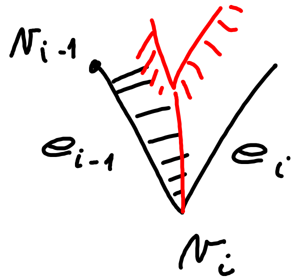
– $v_i \to$ helper(e_i)



- 18 -

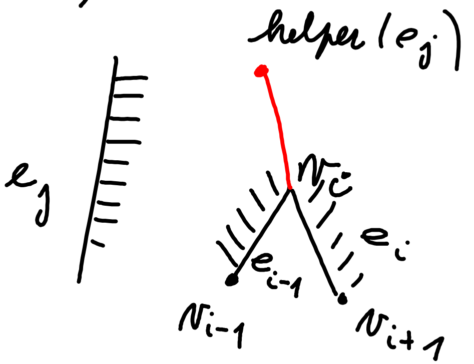
② End

- je-li helper (e_{i-1}) merge, uděláme hranu v_i helper(e_{i-1})
- odebereme e_{i-1} z T



③ Split

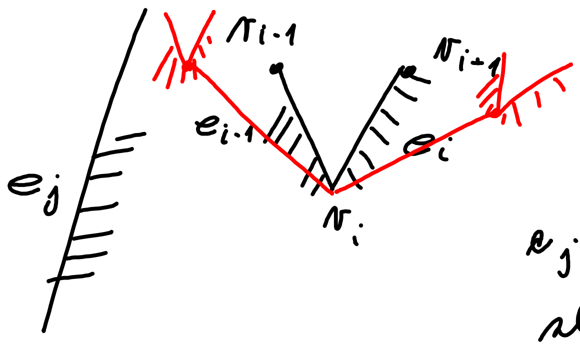
- spojíme v_i s helperem(e_j)
- $v_i \longrightarrow$ helper(e_j)
- e_i přidáme do T
- $v_i \longrightarrow$ helper(e_i)



e_j nejbližší k v_i zleva

④ Merge

- je-li helper e_j merge, vytvoříme hranu
  v_i helper(e_j)
- je-li helper e_i merge, vytvoříme hranu
  v_i helper(e_i)
- $v_i \longrightarrow$ helper(e_j)
- e_i odstraníme z T



e_j je nejbližší k v_i zleva

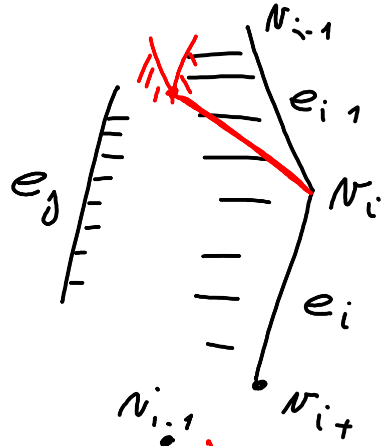
- 20 -

(5a) Regular, P vlevo

- je-li helper(e_j) merge, vytvoříme hranu

  v_i helper(e_j)

- $v_i \rightarrow$ helper(e_j)



e_j je nejbližší vlevo od v_i

(5b) Regular, P vpravo

- je-li helper(e_{i-1}) merge, vytvoříme hranu

  v_i helper(e_{i-1})

- e_{i-1} odstraníme z $\mathcal{T}$
- e_i přidáme do $\mathcal{T}$ $\Rightarrow$ $v_i \rightarrow$ helper e_i