Dinamikus programozás

Hajnal Péter

Bolyai Intézet, TTIK, SZTE, Szeged

2021. ősz

Az alapötlet

• A dinamikus programozás alapötlete egyszerű. Az alapfeladatot egy nagyobb „feladat-sokaságba” ágyazzuk.

• Az alapfeladat a sokaság egyik feladata. A sokaságnak azonban lesznek nagyon egyszerű esetei, amelyek megválaszolása senkit sem állít kihívás elé.

• A sokaság feladatainak lesz egy természetes sorrendje (ez lehet rendezés, de lehet részben rendezés is).

• A feladatok sorrendben való elvégése egyszerű lesz. A korábban megoldott feladatokra alapozva az elsőre nehéz kérdések is könnyen megválaszolhatók lesznek.

Az ötlet „matematikai tartalma”

• A kihívás a jó feladat-sokaság megválasztása. Ha ezt jól választottuk meg, akkor feladataink „felgönygyölítése” gyakran rutinból megoldható.

• Egy algoritmikus probléma-sereg dinamikus programozással történő megoldása nagy hasonlóságot mutat egy állítás-sereg teljes indukciós bizonyításával.

Dinamikus programozás vs rekurzió

- A dinamikus programozást a teljes indukcióhoz „hasonlítottuk”.
- Igazából egy rekurzióról van szó.
- Lényeges különbség van a dinamikus programozás és emögött lévő rekurziós gondolat programozásilag leírt rekurzív kódolása között.
- Formális leírás helyett nézzünk példákat.

A Fibonacci-számok definíciója

Definíció: Fibonacci-számok

$$F_1 = F_2 = 1,$$

ha $n > 2$, akkor

$$F_n = F_{n-1} + F_{n-2}.$$

Példa az első Fibonacci-számokra

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, \ldots$

A Fibonacci-számok és a dinamikus programozás

• Az F_{18} Fibonacci-számot a következő „programmal" számolhatjuk ki:

```
var F(1..18):  natural;
for i=1 to 18 do
     if n=1 or 2 then F(i)=1
     if i>2 then F(i)=F(i-1)+ F(i-2);
print F(18)
```

• Ez a dinamikus programozás logikáját követi. Egy 10 hosszú tömbbel dolgozik. A futás végén mind a 10 szám ott lesz a memóriában, pedig számunkra csak F_{10} fontos.

• A dinamikus programozási algoritmusok sebessége attól függ hány részfeladatot tartalmaz feladat-seregünk.

• A dinamikus programozás formailag gyakran úgy jelentkezik, hogy egy számsorozatot, vagy egy számtáblázatot, vagy bonyolultabb szám-sereget töltünk ki meghatározott sorrendben.

A Fibonacci-számok és a matematikai rekurzió

• A Fibonacci-számokat a következő program is kiszámolja:

```
Fibonacci(n):  if n>2 return Fibonacci(n-1)+ F(n-2)
else return 1;
print Fibonacci(18).
```

• Mit csinál ezzel a gép? Egy bonyolult gépi szintű feloldás történik.

• Emiatt a rekurzív program futása lassabb lesz (sok újra számolás történik).

Szünet

Az alapkérdés

A legnagyobb független ponthalmaz megkeresése egy fában

Adott egy T fa. Határozzuk meg a legnagyobb független halmazának méretét.

Fa

Egy gráf fa egy körmentes, összefüggő gráf.

Független csúcshalmaz gráfban

Egy F csúcshalmaz független, ha nincs olyan él amely mindkét végpontja F-beli.

Gyökeres fák

• Először jelöljünk ki egy speciális $r \in V(T)$ csúcsot, amit gyökérnek nevezünk.

• A (T, r) gyökeres fa új elnevezéseket indukál.

• A csúcshalmaz diszjunkt osztályokba sorolható a gyökértől vett távolság szerint.

• Legyen G_i a gyökértől i távolságra lévő csúcsok halmaza. $G_0 = \{r\}$, G_1 a gyökér szomszédait tartalmazó csúcshalmaz, G_2 a gyökér másodszomszédainak halmaza.

Minden $e \in E(T)$ él esetén van olyan $i \in \mathbb{N}$, hogy e az $x \in G_i$ és $y \in G_{i+1}$ csúcsokat kösse össze. Ebben az esetben x az e élnek megfelelő szülő csúcs, míg y a gyerek csúcs. Azt mondjuk, hogy x gyereke y, y-nak x a szülője.

Gyökeres fák (folytatás)

• Ha x nem a gyökér, akkor x-ből tehetünk egy egyértelmű lépést a gyökér felé. Ezzel x egyetlen szülőjébe (gyakori az *apa* elnevezés, ekkor a gyerekre a *fiú* egy alternatív terminológia) jutunk.

• G_1 tartalmazza a gyökér gyerekeit/fiait, G_2 tartalmazza a gyökér unokáit.

• A gyökér a fa *ősapja*, ennek nincs szülője.

• Ha egy csúcsnak nincs gyereke, akkor *levélnek* nevezzük.
• Ha y-ból a gyökér felé haladva elérjük az x csúcsot, akkor x az y csúcs felmenője, y az x leszármazottja.

• Az r gyökér minden más csúcsnak a felmenője, r-nek minden nem-gyökér csúcs leszármazottja. Egy ℓ levélcsúcs esetén ℓ-nek nincs leszármazottja.

Gyökeres fák (folytatás)

• Egy x csúcs meghatároz/„generál” egy gyökeres részfát, amely csúcsai x és leszármazottjai, élei a T fa ezen csúcsai között haladó élei, gyökere x.

• Ezt a gyökeres részfát T_x-szel jelöljük.

• Egy n csúcsú fának n darab csúcs által generált rész gyökeres fája van (a gyökér az eredeti gyökeres fát generálja, az ℓ levél egy egyetlen csúcsból álló fát generál).

• Egy gyökeres fa mélysége a leghosszabb gyökér-levél út hossza. A 0 mélységű gyökeres fa egyetlen csúcsból áll, a gyökérből, ami az ősapa és egyben levélcsúcs is.

• A fenti szleng/terminológia onnan ered, hogy egy gyökeres fára egy családfa a legjobb példa. Az Árpád-ház családfájában a fenti elnevezések mindennapi értelmet kapnak.

A dinamika leírása

- Megjegyzésünk után az alapfeladatra úgy gondolunk, hogy inputja egy (T, r) gyökeres fa. A független halmazok fogalmát a gyökér nem befolyásolja, csupán kibővíti „nyelvünket”.

A feladatsereg

$$\{\mathcal{F}_x\}_{x \in V(T)},$$

ahol $\mathcal{F}_x$ a (T_x, x) gyökeres fára vonatkozó feladat, azaz a legnagyobb független csúcshalmaz méretének megválaszolása.

- Tehát $|V(T)|$ darab részfeladatot definiáltunk.

A feladatok felgöngyölítési sorrendje

A mélység szerinti növekvő sorrend.

A kezdeti „algoritmus”

Algoritmus-vázlet a legnagyobb független csúcshalmaz megkeresésére

(0) Azaz azokkal a feladatokkal kezdjük, ahol (T_x, x) 0 mélységű, azaz x az eredeti fa egy levele. Ekkor a kérdésre a válasz: 1.

// Ekkor (T_x, x) egy csúcsú, 0 élű gráf, amelyben $\{x\}$ egyben független halmaz is (az $\emptyset$ mellett), azaz a legnagyobb független halmaz mérete 1.

(Göngyölítés) Csúcsainkat mélység szerinti sorban vizsgáljuk. Tegyük fel, hogy feladataink között (T_x, x) a soros.

// A kisebb mélységű gyökeres fákra vonatkozó problémákat már megoldottuk.

A korábbi eredményekből számoljuk ki a soros kérdésre adandó választ.

(Output) (T_r, r) kérdésre adott választ jelentsük be outputként.

A korábbi eredményekből történő számolás alapötlete

• A soros (T_x, x) gyökeres fa független halmazait osszuk két osztályba:

(a) Az x-et nem tartalmazók.

(b) Az x-et tartalmazók.

A két fajta független halmazokra küéöm-külön határozzuk meg a legnagyobb méretét.

(a) eset

- Legyen $f_1, \ldots, f_d$ az x csúcs fiai.

Észrevétel

Ekkor $T_{f_1}, T_{f_2} \ldots, T_{f_d}$ részfákban tetszőlegesen/függetlenül vehetünk egy független halmazt és ezek uniója egy (a) eset alá tartozó független halmazt ad.

Másrészt minden (a) eset alá tartozó független halmazt megkapunk így.

- A független választások miatt egyszerű az elemszám maximalizálása: Mindegyik T_{f_i}-ből egy legnagyobb méretű független halmazt kell választanunk. Ezek az M_i méretek rendekezésünkre állnak, mert (T_x, x) mélysége nagyobb mint (T_{f_i}, f_i) mélysége.
- Az aktuális feladatra adott válasz $\sum_i M_i$.

(b) eset

- Legyen $u_1, \ldots, u_D$ az x csúcs unokái.

Észrevétel

Ekkor $T_{u_1}, T_{u_2} \ldots, T_{u_D}$ részfákban tetszőlegesen/függetlenül vehetünk egy független halmazt és ezek uniója, x-szel kibővítve egy (b) eset alá tartozó független halmazt ad.

Másrészt minden (b) eset alá tartozó független halmaz előáll így.

- Mindegyik T_{u_i}-ből egy legnagyobb méretű független halmazt kell választanunk. Ezek a μ_i méretek rendekezésünkre állnak, mert (T_x, x) mélysége nagyobb mint (T_{u_i}, u_i) mélysége.
- Az aktuális feladatra adott válasz $1 + \sum_i \mu_i$.

Az algoritmus

Algoritmus a legnagyobb független csúcshalmaz megkeresésére

(0) Azaz azokkal a feladatokkal kezdjük, ahol (T_x, x) 0 mélységű, azaz x az eredeti fa egy levele. Ekkor a kérdésre a válasz: 1.

// Ekkor (T_x, x) egy csúcsú, 0 élű gráf, amelyben $\{x\}$ egyben független halmaz is (az $\emptyset$ mellett), azaz a legnagyobb független halmaz mérete 1.

(Göngyölítés) Csúcsainkat mélység szerinti sorban vizsgáljuk. Tegyük fel, hogy feladataink között (T_x, x) a soros.

// A kisebb mélységű gyökeres fákra vonatkozó problémákat már megoldottuk.

A korábbi eredményekből számoljuk ki $\max\{\sum_i M_i, 1 + \sum_i \mu_i\}$-t. Ez lesz a soros kérdésre adandó választ.

(Output) (T_r, r) kérdésre adott választ jelentsük be outputként.

Végső megjegyzés

Könnyű az algoritmust átlakítani úgy, hogy ne csak egy maximális méretet adjon meg, de egy legnagyobb független csúcshalmaz legyen az output.

Szünet

Az alapkérdés

Legnagyobb konvex helyzetű részhalmaz megkeresése egy véges síkbeli ponthalmazban.

Adott n darab pont a síkon: $\mathcal{P} = \{P_i(x_i, y_i)\}_{i=1}^{n}$.

// Feltesszük, hogy pontjaink x-koordinátái különbözőek.

Határozzuk meg azt a legnagyobb $\mathcal{R}$ részhalmazát, amelyek konvex helyzetű.

Definíció: konvex helyzet

$\mathcal{R} \subset \mathbb{R}^2$ véges ponthalmaz konvex helyzetű, ha egy konvex $|\mathcal{R}|$-szög csúcshalmaza.

Példa

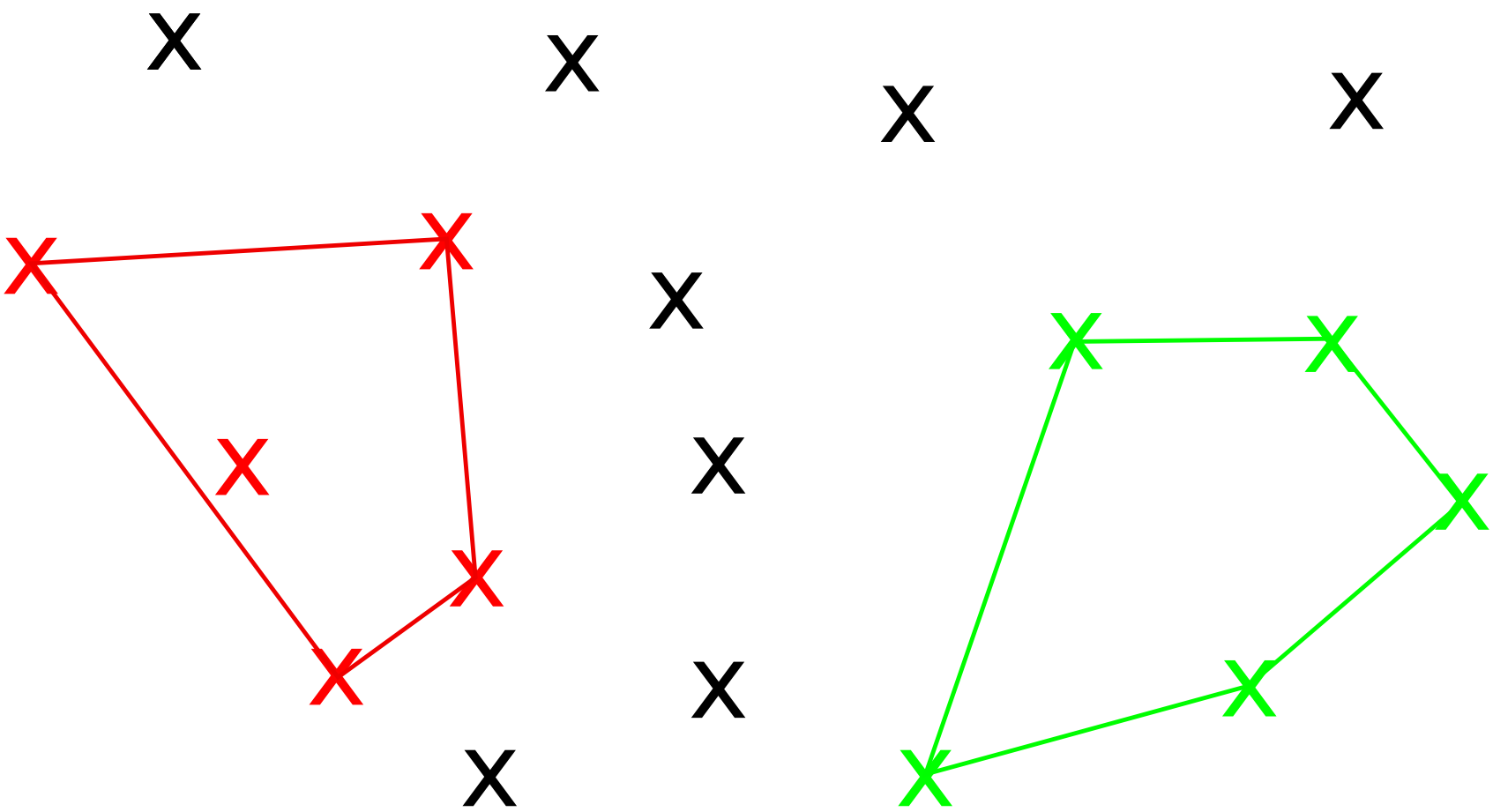

Figure: Egy véges ponthalmaz és egy öt elemű része (piros), amely nem konvex helyzetű, továbbá egy másik öt elemű része (zöld), amely konvex helyzetű.

Kezdeti lépések: seprés

• Először is ponthalmazunkat balról jobbra végigpásztázzuk/seperjük. Pontosabban az x koordinátákat sorbarendezzük.

• // Az egyszerűség kedvéért feltettük, hogy $i < j$ esetén $x_i < x_j$.

• $\mathcal{O}(n \log n)$ összehasonlítással megoldható ez a része algoritmusunknak.

Kezdeti lépések: sapka/csésze problémára való szétszedés

• Másodszor is k konvex helyzetű csúcs konvex burka egy konvex k-szög.

• Ennek kerületének két pontját kiemeljük: a minimális és a maximális x koordinátájú csúcsot (m és M).

• Ez a kerületet, mint körszerűen rendezett véges csúcshalmazt két „ívre” osztja (a két kiemelt csúcsot mindkettőhöz hozzáértjük): egy felsőre és egy alsóra.

• A felső ív csúcsai is konvex helyzetűek, azaz a konvex burkukhoz minden pont hozzájárul mint csúcs. Sőt, mM egy oldal lesz és az egész konvex burok ezen oldal felett fekszik. Azt mondjuk a felső ív egy *sapka* (angolul cap).

• Hasonlóan kezelhető az alsó ív. Erre azt mondjuk, hogy *csésze* (angolul cup).

Példa

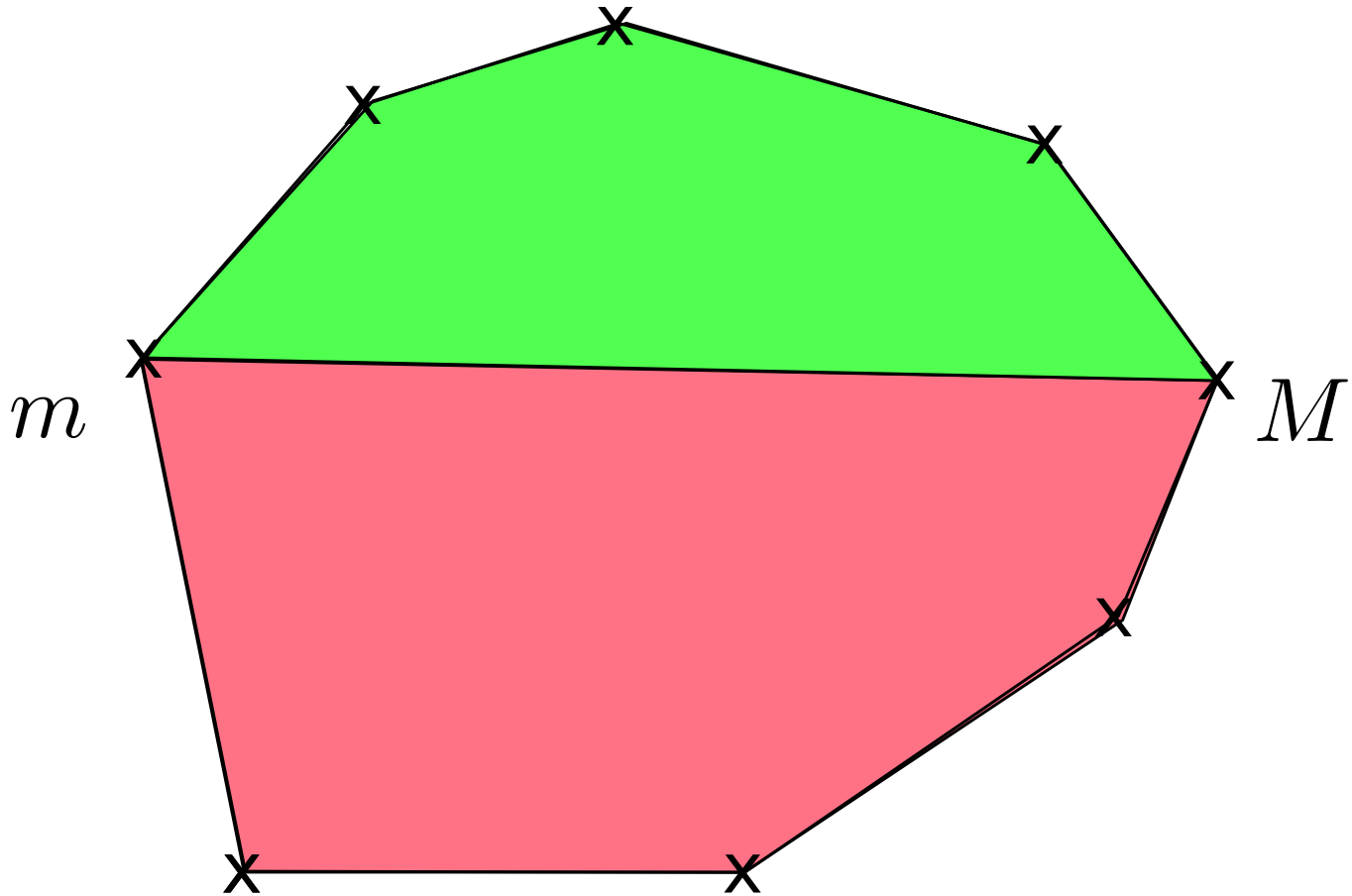

Figure: Egy konvex helyzetű ponthalmaz, a kiemelt két csúccsal. A zöld rész a sapka összetevő konvex burka, a piros a csésze összetevő konvex burka.

Kezdei lépések: elég a sapkákat nézni

• Ha minden $i < j$ esetén tudjuk az P_i-vel kezdődő és P_j-vel végződő leghosszabb sapka, illetve csésze méretét, akkor az alapkérdést is megválaszoljuk.

• Ez az ötlet már egy „dinamikus programozásos” ötlet.

• A tervezés eéső lépése mellett arra használjuk, hogy az alapfeladatot „elfelezzük”, csak a legnagyobb sapkaméret meghatározását tárgyaljuk adott P_i és P_j pontok között.

• Természetesen az alábbi ötletek a legnagyobb csészeméretek meghatározásához is használhatók.

• Így az eredeti feladatot megoldó algoritmus kiolvasható az alábbiakból.

P_i-től P_j-be vezető leghosszabb sapka meghatározásának problémája dinamikus programozással

- A jelölés leegyszerűsítése miatt feltesszük, hogy $i = 1$ és $j = n$.

A feladatsereg

Legyen $\mathcal{F}_{k,\ell,s}$ $(k \leq \ell < s)$ az a feladat, amely megadja a leghosszabb P_k-val kezdődő, P_s-ben végződő, oda közvetlenül P_ℓ-ből jutó sapka problémáját.

- Így $\mathcal{O}(n^3)$ részfeladatot definiáltunk.

A felgöngyölítési sorrend

$\ell - k$ szerint növekvő sorrendben kezeljük a problémákat.

- A legegyszerűbb/kezdő problémák az $\mathcal{F}_{k,k,s}$ problémák. Ekkor a P_k, P_s két pont egy sapkát alkot, ami az optimális sapka.

A göngyölítési lépés

• Tegyük fel, hogy $k < \ell$ és $\mathcal{F}_{k,\ell,s}$ problémával foglalkozunk, de az összes $\mathcal{F}_{k_0,\ell_0,s_0}$ probléma megoldása ismert, ha $\ell_0 - k_0 < k - \ell$.

• Csak azok a feladatok érdekesek, ahol a P_k, P_ℓ, P_s hármas sapkát alkot (ez nem volt probléma a $K = \ell < s$ esetben). Ezt a továbbiakban feltesszük.

• Tippeljük meg milyen csúcs lehet a sapka hátulról harmadik (azaz a P_ℓ előtti) pontja.

• Ez a csúcs geometriailag egy háromszögből kerül ki: A P_k-n átmenő függőleges egyenestől balra kell lennie a sapka hiányzó részének. A P_kP_ℓ egyenes felett és a $P_\ell P_s$ egyenes alatt. A kijelölt háromszög határán nem fekhet a csúcsunk, de kivételt alkot P_k, ami előfordulhat, mint közvetlenül P_ℓ előtt álló csúcs.

title

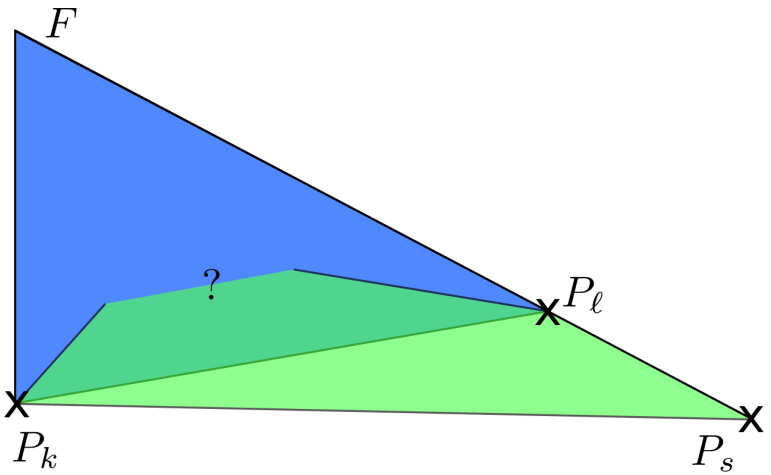

Figure: F a P_i-n áthaladó függőleges egyenes és a $P_\ell P_s$ egyenes metszéspontja. A kék $P_i P_\ell F$ háromszögben kell megkeresni az optimális sapkát.

Vesszük $P_k P_\ell F$ háromszögbe eső pontjainkat (ezek P_k és még esetleg további belső pontok a háromszögből). Mindegyik P_{ℓ^-} pontra a korábbi munkánkból kiolvassuk $\mathcal{F}_{k,\ell^-,\ell}$ megoldását. Ezekből a számokból a legnagyobbhoz 1-et hozzáadva kapjuk $\mathcal{F}_{k,\ell,s}$ megoldását.

Végső megjegyzések

• Fontos, hogy az $\mathcal{O}(n^3)$ feladat közül a már megoldottak megoldását tároljuk és ebből olvassuk ki ami szükséges számunkra.

• Könnyű az algoritmust átlakítani úgy, hogy ne csak egy maximális méretet adjon meg, de egy legnagyobb konvex helyzetű ponthalmaz legyen az output.

Vége van!

Köszönöm a figyelmet!