ADS II

Michal Vaner

2. ledna 2013

Obsah

1 Paralelní algoritmy

1.1 Hradlové sítě

Hradlová síť je matematický model elektroniky. Dělí se na boolovské obvody (jedničky a nuly) a na kombinační obvody (nad libovolnou konečnou abecedou).

Skládá se to z hradel, sestavuje se z nich graf (orientovaný). Obecně lze považovat za funkce nebo algebraické operace. Dále zde máme vstupní a výstupní porty (lze považovat za hradla bez vstupů nebo výstupů).

- Musí to být acyklické.
- Na žádný vstup nevede více příchozích hran.
- Vše musí být zapojené.

Boolovský obvod je acyklický orientovaný graf s vrcholy $V = I \cup O \cup H$ a hranami E, takových:

- $$\forall i \in I : deg^{+}(i) = 0$$
- $$\forall o \in O : deg^{-}(o) = 0; deg^{+}(o) = 1$$
- $$\forall h \in H \exists f(h) : \{0,1\}^{a(h)} \to \{0,1\}$$
  $$deg(h)^{+} = a(h)$$
  vstupní hrany jsou očíslované $1 \dots a(h)$

Nechť $a(h) \leq 2$.

Výpočet sítě probíhá po taktech, v 0-tém taktu jsou definované právě všechny vstupy. V i-tém taktu vydají výstup hradla, která měla definované vstupy v $i-1$-tém taktu. Když jsou definovány hodnoty všech hradel i portů, síť se zastaví a vydá výsledek.

i-tá vrstva jsou takové vrcholy $\{v | v \in H \cup I \cup O, max(\{|v,i| \, | \forall i \in I\}) = i\}$

Časová složitost sítě je počet vrstev (tedy nejdelší cesta ze vstupu do výstupu).

Prostorová složitost sítě je počet hradel (dalo by se použít počet hradel v největší vrstvě).

Síť budeme vybírat dle velikosti vstupu (tedy, máme posloupnost sítí $S_1, S_2, \dots, S_\infty$. Zavedeme podmínku, že daná síť musí být vygenerovatelná polynominálním algoritmem z velikosti vstupu (kvůli omezení příliš silného modelu—umělo by to jinak řešit problémy neřešitelné obvyklými algoritmy).

1.1.1 Multi-or

Zjištění, jestli je mezi nimi alespoň jedna jednička.

Možno spárovat sousedy (dvojice), totéž s výsledky, nakonec zbyde jeden výstup.

Lineární prostorová složitost, logaritmická časová.

1.1.2 Sčítání n-bitových čísel

Pomocí „sčítání pod sebe“ lze udělat v lineárním čase.

Blok může buď generovat přenos vždy, nebo nikdy, nebo ho zachovávat. Tyto bloky se dají skládat. Lze poskládat výhybkové stanice.

2 Toky v sítích

Síť je čtveřice Orientovaný graf G, ***Zdroj*** $Z \in V_G$, ***Stok*** $S \in V_G$ a funkce $c\,(E_G) \to \mathbb{R}^+$ udávající kapacitu hran.

- Hledáme maximální tok.
- Ve zdroji se „objevuje“, v stoku „mizí“.
- Tok na každé hraně je omezen její kapacitou.
- Nikde se to neztrácí a neobjevuje, s výjimkou zdroje a stoku.

O maximu:
Pro každou síť existuje maximální tok.

Řez v grafu je množina hran, taková, že neexistuje cesta ze zdroje do stoku, která nepoužije alespoň jednu z těch hran.

Hlavní věta o tocích:
Když S je síť, pak maximální tok je roven kapacitě minimálnímu řezu.

Lemma:
$A \subset V_G, Z \in A, S \notin A$. f je tok. $f = f(A, V_G - A) - f(V_G - A, A)$. **Důkaz**:
Z definice velikosti toku a z toho, že „potrubí nikde neteče“.

Důsledek:
Když f je tok a R řez, pak velikost toku je vždy shora omezená kapacitou libovolného řezu.

Redefinice:
Cesta od teď může využívat hrany i v protisměru, součet všech po směru mínus rozdíl proti směru musí být mezi nulou a kapacitou.

Nasycená cesta je cesta, kde existuje plně využitá hrana.

Nasycený tok je takový tok, kde je každá jeho cesta nasycená.

Každý maximální tok je nasycený. Pro maximální tok existuje řez, který má stejnou kapacitu.

Důkaz sporem:
Tok je maximální, ale není nasycený. Nějaká cesta se dá zvětšit, čímž se zvětší tok.

2.1 Algoritmus

Dokud to jde, najde nenasycenou cestu ze zdroje do cíle a nasytit ji, když doběhne, tak je to dobře. Problém—může se zacyklit, ale pokud je síť trochu rozumná, tak se zastaví. Mějme síť, ve které již něco teče. ***Síť rezerv*** je síť popisující zbylé kapacity hran (tedy, ty samé vrcholy, hrany tam a zpět a kapacity kolik zbývá a kolik už teče). Může to být multigraf ale násobnosti max. 2 (tedy, mohou být až 2 paralelní hrany).

Lze předefinovat Ford-Falkosnův algoritmus tak, že vždy najde nenulovou cestu v síti rezerv a podle té to zvýší.

Lze však zlepšit i podle toky v síti rezerv.

2.2 Dinicův algoritmus

- Začne s nulovým tokem.
- Sestrojí síť rezerv, bez hran o kapacitě 0.
- Změří nejkratší vzdálenost ze zdroje do spotřebiče (pokud nekonečno, tak je již nejlepší).
- Sestrojí ***pročištěnou síť***—ponechá jen vrcholy a hrany ležící na nejkratších cestách.
- Sestrojíme ***blokující tok*** G.
- Zlepšíme F podle G.
- Opakuje („otrhané“ hrany a vrcholy zase vrátí).

2.2.1 Pročištěná síť

Obsahuje vrstvy, zdroj je ve vrstvě 0, další vrstva je ve vzdálenosti 1, spotřebič je ve vrstvě l. Hrany vždy jdou z vrstvy k do $k + 1$.

Hrany lze podle nějakých pravidel vyházet (vracející se, nevedoucí do spotřebiče, v rámci jedné vrstvy, ty které mají vstupní nebo výstupní hranu nula).

2.2.2 Blokující tok

Je tok, kde po každé orientované cestě ze zdroje do spotřebiče existuje alespoň jedna plná hrana.

Vyrazím libovolnou cestou k spotřebiči a takovou cestu naplním, kolik se vejde.

Takové „plné“ hrany rovnou vyhazuji (a updatnu mazání slepých uliček).

Jedna fáze lze zhora odhadnout na $O(M * N)$.

V každé fázi se l zvětší alespoň o 1, $l_{max} = N$ – zastaví se, celková složitost je $O(M * N^2)$. Nelze dokázat tím, že cesty délky l zablokuju – přibývají jiné (opačné) hrany—ale vznikají tím jen hrany z k do $k-1$ vrstvy.

Lze upravit tak, že celou dobu jen snižujeme rezervy, až to bude celé hotové, tak se ty rozdíly odečtou a máme správný tok. Obě čištění mohou být průběžné. Lze také prohledáváním do hloubky i v ne zcela pročištěné sítě a mazat při vracení ze slepých uliček a z neslepých updatovat rezervy hran.

Toky v sítích lze použít např. pro hledání maximálního párování.

2.3 Goldbergův algoritmus

Funkce $f : E \to \mathbb{R}^+$ je ***vlna*** v síti, pokud nejsou překročeny kapacity a ve vrcholech se může něco ztrácet, ale nesmí se objevovat.

Vrcholům přiřadíme výštky $h : V \to \mathbb{N}$. Umíme 2 operace:

- Převedení toku. Když mám vrchol, který má kladný přebytek a sousední vrchol, který má menší výšku a rezerva hrany do toho vrcholu je nenulová, tak část přebytku převedeme (minimum z přebytku a rezervy).
- Když už žádná hrana z kopce nevede a stále je přebytek, tak vrchol o 1 zvýšíme.

Nastavíme všechny výšky na 0, až na zdroj, který nastavíme na N. Počáteční vlna je všude 0, jen ze zdroje má vlna hodnotu rovnou kapacitě hran. Pak opakujeme výše zmíněné operace, dokud to jde, resp. dokud někde něco přebývá.

2.3.1 Invariant A

f je vlna a h nikdy neklesá. f je na začátku vlna a žádným krokem to nezničíme - nevyrobíme záporný přebytek. To že výšky neklesají, je zřejmé. Zdroj má celou dobu výšku N a spotřebič 0.

2.3.2 Invariant S (o spádu)

Každá nenasycená hrana má spád (rozdíl výšek) maximálně 1. Na začátku zřejmě platí (jediný spád je N, ale ty hrany jsou nasycené). A rozbít to

nemůžeme—nikdy neklesáme a zvednout to, co už má spád 1, nesmíme, napřed musíme nasytit, nebo vyprázdnit vrchol, ale prázdný vrchol nezvedáme. Taktéž neodnasytíme hranu, která má vyšší spád, protože bychom museli strkat vodu do kopce.

2.3.3 Lemma K (korektnost)

Když se algoritmus zastaví, f je maximální tok. Určitě je to vlna a zastaví se ve chvíli, kdy není žádný přebytek $\Rightarrow$ je to tok. Kdyby nebyl maximální, existuje nenasycená cesta P ze zdroje do spotřebiče. Zdroj má výšku N a spotřebič výšku 0. Ta má maximálně $N-1$ hran. Na ní je určitě hrana, která má spád alespoň 2, tedy nemůže být nenasycená.

2.3.4 Invariant o cestě do zdroje

Je-li v vrchol na cestě ze zdroje do spotřebiče, pak existuje nenasycená cesta z v do zdroje.

$A := \{u | \exists$ nenasycená cesta $v \to u\}$. Pokud máme nějakou hranu z $b \notin A$ do $a \in A$, tak po ní nic neteče, jinak by rezerva opačně nebyla nulová, a tedy by tam musela být.

To co teče do A je 0, to co teče z A může být nezáporné. Tedy, ten přebytek musí pocházet zevnitř a může pocházet jen od zdroje.

2.3.5 Invariant H (o výšce)

$\forall v \in V; h(v) \leq 2N$. Předpokládejme opak. Museli jsme někdy zvednout vrchol, který má výšku alespoň $2N$. Zvedali jsme ho, proto měl kladný přebytek. Ale podle invariantu o cestě do zdroje existuje cesta do zdroje, která není nasycená a ta cesta má spád alespoň N, tedy každý úsek je z kopce. Tedy ji nemáme důvod zvedat, protože z tohoto vrcholu musí téct.

2.3.6 Lemma o zvedání

Každý vrchol tedy zvedneme maximálně $2N$-krát. Tedy počet zvednutí jsou maximálně $O(N^2)$.

Převedení se nazývá ***nasycené***, pokud se nasytí hrana. Převedení, které není ***nenasycené***, „vyprázdní“ vrchol.

2.3.7 Lemma S – sytých převedení

Sytých převedení je maximálně MN. Převedení může nastat jen, když je to z kopce. Pokaždé se musí hrana překlopit na druhou stranu, překlopení vyžaduje minimálně jedno zvednutí.

2.3.8 Lemma N—nenasycených převedení

Součet výšek vrcholů s přebytkem (kromě zdroje a spotřebičem) nazveme ***potenciál***. Určitě neklesne pod nulu, na počátku je potenciál roven 0. Zvednutí vrcholu zvýší potenciál o 1. Nenasycené převedení ho sníží buď o 1 nebo o $h(u)$. Nasycené se zvýší maximálně o $2N$.

Potenciál lze zvýšit (dohromady) maximálně o $2N^2+2N^2M$, pokaždé ho snížíme alespoň o 1, tedy nenasycených převedení je maximálně $O(N^2M)$.

Tedy, celý algoritmus běží v $O(N^2M)$.

Můžu vždy vybrat z těch, které mají přebytek, ten nejvýš.

2.3.9 Lemma N'

Tentokrát lze dosáhnout $O(N^2\sqrt{M})$. **Lemma** N':

V algoritmu G' (Goldbergův algoritmus se zvedáním nejvyššího vrcholu s přebytkem) je počet převedení $O(N^3)$. Označme H maximální výšku vrcholu s přebytkem.

Rozdělíme si běh algoritmu na fáze, každá fáze končí změnou H. Počet nenasycených převedení je maximálně tolik, s kolika vrcholy na nižší hladině ta fáze začala (každé převedení je jen o jednu hladinu dolů).

Fáze může skončit buď zvýšením nebo snížením H. Zvýšení může být nejvíce $O(N^2)$, snižovat se dá jen to, co se předtím narostlo, takže také.

Tedy všech fází dohromady je $O(N^2)$, tedy všech nenasycených převedení je maximálně $O(N^3)$.

Lemma N'':
Stejný algoritmus, lepší odhad. Rozdělme si fáze na laciné a drahé, tedy ty, které udělají nejvíce k převedení a více než k. Převedení v laciných fázích je $O(k \cdot N^2)$.

Pro odhad drahých fází definujme potenciál $\Psi := \frac{\sum_{V-\{z,s\}} p(v)}{k}$, kde $p(v)$ je počet vrcholů s přebytkem na stejné a nižších hladinách.

Na začátku je $\Psi \leq \frac{N^2}{k}$.

Zvednutí může způsobit, že se až N vrcholů dostane pod něj, tedy Ψ může vzrůst až o $\frac{N}{k}$, na ostatních se sníží o něco, tedy celkem se může zvýšit až o $\frac{N}{k}$.

Nasycené převedení může způsobit nový vrchol s přebytkem, tedy se to mohlo zvýšit až o $\frac{N}{k}$.

Nenasycené převedení způsobí vynulování jednoho vrcholu, tedy z něj vypadne $\frac{p(u)}{k}$ a může přibýt $\frac{p(v)}{k}$, ale v je v nižší hladině. Tedy, klesne to o $\frac{p}{k}$, kde p je počet vrcholů na nejvyšší hladině. Ale je to drahá fáze, takže $p \geq k$, tedy to klesne alespoň o 1. V levných fázích to určitě nezvýší.

Dohromady potenciál naroste na $\frac{N^2}{k}+\frac{N\cdot 2N^2}{k}+\frac{N\cdot NM}{k} = \frac{N}{K}\left(N+2N^2+NM\right) = O\left(\frac{N^2M}{k}\right)$. Tedy, počet nenasycených převedení v drahých fází je maximálně tolik.

Nyní již stačí vybrat $k = \sqrt{M}$.

3 Paralelní třídění

Definujme ***komparátorovou síť*** jako kombinační obvod (pracující ne s dvojkovými, ale s obecnými hodnotami) a hradla budou pouze komparátory.

Komparátor je něco, co dostane dva vstupy A a B a na výstupu vydá min a max.

Nechť se výstupy komparátorů nikde nevětví.

Při přepsání bublinek to jde udělat na $O(N)$ hladin a $O(N^2)$ komparátorů.

Čistě bitonická posloupnost je taková, která napřed roste a potom klesá. ***Bitonická posloupnost*** je taková, která lze zrotovat o nějaké k, abychom dostali čistě bitonickou.

Separátor budeme říkat komparátorové síti, která vždy porovnává x_{0+k} s $x_{\frac{n}{2}+k}$, kde $k := 0, \ldots, \frac{n}{2} - 1$.

Lemma:
Pokud vstupem komparátoru S_n je bitonická posloupnost, tak na výstupu dostaneme 2 bitonické posloupnosti, kde všechny prvky v levé jsou menší než ty v té pravé.

Na čistě bitonickém vstupu je to vidět jednoduše. Stačí najít dělící bod. Pro ostatní lze nahlédnout, že separátory lze rotovat.

Pomocí separátorů lze setřídit libovolná bitonická posloupnost, používáme menší a menší separátory, až dojdeme k posloupnostem délky 1. Na to stačí $O(\log N)$ hladin a $O(N \cdot \log N)$ komparátorů.

Pomocí bitonické třídičky lze slévat setříděné posloupnosti. Pomocí tohoto lze sestavit něco jako mergesort. Celková hloubka je tedy $1 + 2 + 3 + \ldots + \log N = O(\log^2 N)$, celkově má $O(N \cdot \log^2 N)$.

4 Hledání podřetězců

Mějme Σ nějakou abecedu (např. a…z). Označme Σ^* množinu všech slov nad touto abecedou. Slovo bude nějaká konečná posloupnost písmen z abecedy. Prázdné slovo označíme λ. Délka slova α je počet písmen tohoto slova, značené $|\alpha|$. Zřetězení slov α a β budeme značit $\alpha\beta$.

Pozorování:

- $\lambda\alpha = \alpha\lambda = \alpha$
- $|\alpha\beta| = |\alpha| + |\beta|$
- Prázdné slovo je podslovem libovolného slovo.

- Každé slovo je podslovo sebe sama, pokud to chceme vyloučit, říkáme ***vlastní podslovo***.

Budeme indexovat jako v pythonu.

4.1 Definice problému

Mějme dvě slova, ι je hledané slovo ($|\iota| = J$) a σ slovo prohledávané ($|\sigma| = S$). Chceme $\{i; \sigma[i : i + J] = \iota\}$.

4.2 Vyhledávací automat

(Knut, Moris, Pratt, neboli KMP).

Máme graf, každý vrchol je nějaký prefix hledaného slova a přechody mezi nimi jsou hrany. Počáteční stav odpovídá λ. Hrany, které zvětšují prefix (o 1 písmeno) nazýváme ***dopředné hrany***. Takové hrany jsou $d(\alpha, x)$ a to je buď $d(\alpha, x) := \alpha x$, pokud αx je stav, nebo není definovaná. Dále jsou zde ***zpětné hrany***, odpovídají návratům v textu, když se to nenalezne. Zpětná hrana je funkce $z(\alpha)$ a je definovaná pro všechny stavy kromě λ.

Vždy chceme, aby aktuální stav byl nejdelší možný prefix končící na aktuální pozici v textu. Pokud se vracíme, tak se snažíme vrátit co nejméně tak, aby se opět dostal do nějakého stavu i po přidání aktuálního písmene.

4.2.1 Algoritmus

- Vždy vezmu písmeno.
- Dokud nelze jít dopředu nebo nejsem na začátku, vracím se po zpětných hranách.
- Pokud jde dopředu, jdu dopředu.
- Pokud jsem ve stavu ι, započítám to.

4.2.2 Implementace

Není potřeba si pamatovat celé stavy. Místo stavu si mohu pamatovat jen číslo, jak dlouhý prefix už mám. Dopředná hrana je reprezentovaná jen písmenkem, které ji podmiňuje, přičítá jedničku. Zpětné hrany jsou čísla, kam se vracíme.

4.2.3 Důkaz správnosti

Stav po přečtení po vstupu označme $\alpha(\beta)$. Tento stav je nejdelší sufix β takový, že je prefixem α. Pokud se podaří dokázat tento invariant, je správnost jasná.

Invariant dokážeme indukcí. Na začátku zřejmě platí, tedy čtení znaku c. Mějme stav $\alpha(\beta)$, o kterém platí, a chceme z něj dokázat, že platí i pro $\alpha(\beta c)$. Potřebujeme najít takový stav, ke kterému lze přidat c. Když to nejde, tak ho co nejméně zkrátíme a zkusíme to znovu. A tak dále, dokud se to nepovede.

4.2.4 Časová složitost

Vyhledávání doběhne v čase $O(S)$. Každé písmeno přečte právě jednou. Sice se můžeme vícekrát vracet, ale to musí vždy alespoň o jedna a kolikrát jdeme dozadu, tolikrát jsme museli předtím museli jít dopředu a k tomu jsme vždy potřebovali písmeno. Všech kroků zpět je tedy maximálně tolik, kolik je kroků dopředu a tedy je jich maximálně S.

Paměť je také lineární.

4.2.5 Konstrukce zpětných hran

Zpětná hrana ze stavu $\alpha(\beta)$ musí vést do stavu, do kterého se musí automat dostat bez prvního znaku, tedy do $\alpha(\beta[1:])$. (Potřebujeme alespoň o znak kratší, tak ho jednoduše usekneme) Protože $\alpha(\beta)$ je prefix ι, tedy $\beta = \iota[:i]$ a tedy $\alpha(\beta[1:]) = \alpha(\iota[1:|\beta|])$.

Toho lze využít při tvorbě automatu, jednoduše proto, že když děláme k-tou zpětnou hranu, tak ho krmíme vstupem $\iota[1:k]$, jehož délka je $k-1$, tedy tuto zpětnou hranu ještě nebude určitě potřebovat a ty předtím už máme z minule.

Z toho je zřejmé, že i časová složitost je také lineární.

4.3 Rabin & Kasp

Když si vezmeme nějakou hashovací funkci, tak můžeme začít porovnávat, až když souhlasí hash. Kdyby šel přepočítat (posunout o 1 doprava) v konstantním čase, pak by to šlo přibližně lineárně s délkou vstupu. Mohla by to být například lineární kombinace s váhami mocninami nějakého prvočísla.

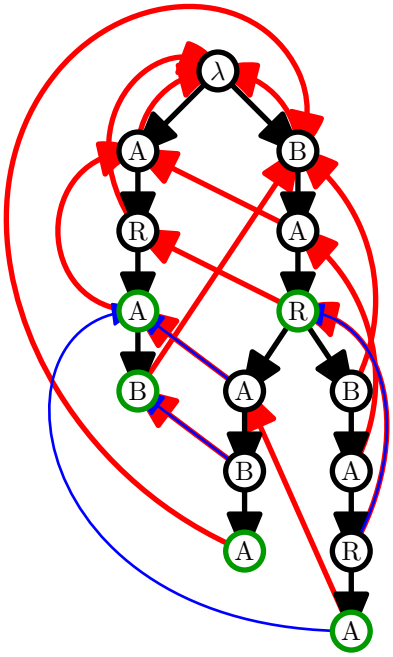
4.4 Vyhledávání množiny řetězců

Máme σ seno a $\iota_1, \ldots, \iota_k$ a chceme najít všechny dvojice takové, že $(i,j) : \iota_i = \sigma[j : j + |\iota_i|]$. Opět provedeme pomocí vyhledávacího automatu.

Automat nebude cestička, bude to složitější. Např. pro tato slova bude vypadat dle obrázku (se všemi informacemi, viz. níže).

- ARA
- BAR
- ARAB

- BARABA
- BARBARA



Obrázek 1: Příklad vyhledávacího automatu

Prohledávání probíhá stejně. Problém je ale s výpisy (výpis jen každý vrchol, kde něco končí, nestačí). Přidáme ještě zkratkové hrany – zpětná hrana vedoucí do nejbližšího konce slova. V každém kroku prohledám a vypíšu sebe, pokud jsem označen a všechny, kam se dá postupně dostat po zkratkách.

4.4.1 Časová složitost

Vlastní procházení funguje stejně rychle, jako s jedním slovem.

Druhá část je lineární s počtem výskytů (každý vypíše právě jednou a při každém kroku po zkratkové hraně právě jeden vypíše).

4.4.2 Konstrukce automatu

Aho & Corasicková

1. Postavíme strom – viz trie.
2. Označíme, kde končí slova – lze společně se stavbou stromu.

3. Spočítáme zpětné hrany. Stejným způsobem jako minule, jen musíme dělat všechny slova zároveň – jednoduše po hladinách. (Lze udělat průchodem do šířky, čímž hledáme více slov najednou.)

4. Dopočítají se zkratky.

Lze jednoduše dokázat, že toto funguje v $O(\sum_i \iota_i)$.

Poznámka:
Předpokládáme, že operace se stromy jsou konstantní. Toho lze dosáhnout buď malou abecedou, nebo vhodným použitím hashovacích tabulek.

5 Práce s polynomy

Polynom je nějaká $P(x) = \sum_{j=0}^{n-1} B \cdot x^j$.

Mějme polynomy $P(x)$ a $Q(x)$ (stejného stupně). Sčítání je bez problémů. Součin je $R(x) = P(x) \cdot Q(x)$, tedy $r_l = \sum_{j=0}^{l} p_j \cdot q_{l-j}$. Toto se dá stihnout v $O(N^2)$.

5.1 Násobení

Můžeme si všimnout, že $\forall x R(x) = P(x) \cdot Q(x)$ a polynom je určen jeho hodnotou v $n+1$ bodech.

Vezmu body $x_0, \ldots, x_{2n-1}$ (n je stupeň původních polynomů). Spočítám a vynásobím hodnoty zdrojových polynomů. Tím získám body, kterými R prochází, z čehož spočítám R.

5.1.1 Vyhodnocení polynomu v n bodech

BÚNO $n = 2^m$.

Rozdělíme na sudé a liché koeficienty, z těch lichých vytkneme k, čímž získáme sudé mocniny x v obou polovinách. Čímž se to zredukuje na vyhodnocení 2 polynomy v bodě x^2. Lze využít toho, že můžeme použít x kladné a záporné (tedy získáme tím 2 výsledky na 1 vyhodnocování).

Pokud použijeme komplexní čísla, tak dokážeme, aby i x^2 mohlo být záporné, atd.

Komplexní čísla lze zapsat také jako $x(\cos\varphi + i\sin\varphi)$. Dále lze použít formuli, že $e^{i\varphi} = \cos\varphi + i\sin\varphi$, což lze hezky použít při umocňování a odmocňování. Odmocnin může být více (např. n-tých odmocnin z 1 je právě n).

Komplexní číslo x je ***primitivní n-tou odmocninou z*** $1 \Leftrightarrow x^n = 1, x^1, x^2, \ldots, x^{n-1} \neq 1$.

Pozorování:
Dá se zjistit, že existuje vždy alespoň jedna pro každé n a to ta s nejmenší φ, její číslo komplexně združené je také takové. Budeme mu říkat ω a $\overline{\omega}$

Pozorování:
Pro různá k, j, $\omega^k \neq \omega^j$

Toto lze použít v algoritmu, stačí používat jako x vhodnou primitivní odmocninu z 1. Celková časová složitost je tedy $O(N \cdot \log N)$.

`TODO: Fourierova transformace`

Pomocí FFT lze z bodů dostat opět polynom.

Lze použít také k zpracování signálů či kompresi dat.

6 Geometrie v rovině

6.1 Hledání konvexního obalu

$M \subseteq \mathbb{R}^2$ je ***konvexní*** $\Leftrightarrow \forall a, b \in M; \overline{ab} \subseteq M$ ($\overline{ab}$ je úsečka z a do b).

6.1.1 Zametání roviny

Jede přímka v nějakém směru, sbírá body a to, co už je zametené je konvexní obal a vždy když se potká nový se to jen rozšíří.

Vezmeme první 3 body, ty tvoří trojúhelník. Když potkáme další bod, tak ho buď připojíme rovnou, nebo napřed některé starší odebereme. To lze zjednodušit nalezením horní a dolní obálky, ta horní stále zatáčí doprava, dolní doleva. Při přidávání odebíráme staré body tak dlouho, dokud by přidání nového porušilo tuto podmínku a pak jej tam přidáme.

Každý bod je přidán maximálně N-krát, odebíráme také tak, proto po setřídění to stihneme v $O(N)$. Setřídění ale trvá (v obecném případu) $O(N \cdot \log N)$.

6.2 Rozdělení roviny na oblasti, které mají někam nejblíže

Konečnou množinu $M \subseteq \mathbb{R}^2, M = \{M_1, \ldots, M_n\}$ míst v rovině. Systém množin $M_1, \ldots, M_n$ takový, že $\forall i \forall j \forall x \in M_i; d(x, m_i) \leq d(x, mj) \wedge \bigcup_i M_i = \mathbb{R}^2$ nazveme ***voroneho diagram***.

Pozorování:
$\forall i; M_i$ je ohraničena lomenou čarou a je konvexní.

Důkaz:
Každá dvojice vrcholů si rozdělí rovinu na poloroviny, takováto oblast je jen průnikem takových polorovin, vždy směrem k tomu bodu.

Každý segment lomené čáry nazveme ***hranou***.

Pozorování:
Všechny body na nějaké hraně jsou stejně daleko od nějakých dvou míst.

Křižovatky těchto hran nazveme ***vrcholy***.

Pozorování:
Každý vrchol je stejně vzdálen alespoň od 3 různých míst.

Pozorování:
Pokud se žádné 4 body neleží na kružnici, tak každý vrchol má stupeň 3.

Pozorování:
Počet vrcholů + počet hran = $O(N)$.

Důkaz:
Přes konstrukci rovinného grafu.

6.2.1 Zametání

Nejde vzít přímku a jednoduše zamést, je třeba pamatovat si bezpečné paraboly okolo oscanovaných vrcholů.

Potřebujeme tyto datové struktury:

- Máme haldu událostí
- Pobřežní linii (z těch parabol, stromečkem)
- Výsledek

6.2.2 Algorinmus

1. Připravíme haldu událostí, vložíme všechny místní události.
2. Připravíme pobřežní linii P, zatím prázdná (žádné paraboly).
3. Připravíme reprezentaci diagramu.
4. Pro každou událost z haldy:
   (a) Je-li na řadě místní událost, najdeme průsečík s pobřežím P, vložíme další parabolu a přidáme vznik hran a přepočítáme, kdy zanikají paraboly.
   (b) Je-li událost zániku paraboly, smažeme parabolu, zapíšeme hrany a přepočítáme zániky parabol.

TODO: nějak to pochopit a doplnit, tady jsem usnul :-(

6.2.3 Složitost

Místních událostí je N. Parabola vznikne jen při místní události, proto počet zániků je také lineární.

Velikost pobřeží i haldy je maximálně lineární, výsledek také.

Celé to tedy běží v $O(N \cdot \log N)$. P je třída (rozhodovacích) problémů, které jsou řešitelné v polynomiálním čase. NP je třída (rozhodovacích) problémů, které jsou řešitelné s lineárně velkou nápovědou (tedy, umíme to zkontrolovat v polynomiálním čase).

NP***-těžký problém*** je takový, na který lze převést všechny NP problémy. NP***-úplný problém*** je takový, který je v NP a je NP-těžký.

6.3 NP-úplné úlohy

6.3.1 Logické

- SAT
- 3-SAT
- $3,3$-SAT
- Obvodový SAT

6.3.2 Grafové

- Klika
- Nezávislá množina
- $3D$-párování
- 3-obarvení
- Hamiltnovská cesta/kružnice.
- Nejdelší cesta

6.3.3 Číselné

- Problém batohu
- Problém 2 loupežníků
- $Ax = b$, $x = \{0,1\}^n$, na celých číslech
- Celočíselné lineární programování

7 Řešení NP-úplných problémů

7.1 Stačí speciální případ

Například nezávislá množina na stromech – stačí trhat od listů po vrstvách.

Problém batohu pro malá přirozená čísla – začneme 0 předměty, budeme zlepšovat o jeden další předmět a pamatujeme si všechny součty, které se tím dají získat.

7.2 Přibližné řešení

Problém obchodního cestujícího – pokud nám platí trojúhelníková nerovnost a graf je úplný – najdeme minimální kostru a objede ji, když bysme někudy procházeli, tak to obejdeme.

Když není trojúhelníková nerovnost. Pokud by existoval nějaký aproximační algoritmus, pak $P = NP$. Doplníme na úplný graf tak, že staré hrany mají 1, nové mají ∞. Z toho lze potom poznat, jestli má hamiltnovskou kružnici, což je NP-úplný problém.

Problém batohu – zaokrouhlíme.

8 Celá čísla

8.1 Značení

Libovolná čísla jsou vždy celá.

- $a \backslash b \Leftrightarrow \exists c : b = ac$
- $\gcd(a, b)$ – největší společný dělitel a a b
- $a \equiv_n b \Leftrightarrow n \backslash a - b$
- $a \perp b \Leftrightarrow \gcd(a, b) = 1$

8.2 Časové složitosti

• $+, -$	$O(N)$
• $*, /, \mod$	$O(N^2)$
• $\gcd(N^3)$	$O(N^3)$

8.3 Algebraické struktury

Viz algebra

- Grupa $G(\cdot, 1, ^{-1})$
- Podgrupa
- Generátor grupy

La-Grangerova věta o grupách:
Když $H \subseteq G$ pro G konečnou grupy, pak $|H| \backslash |G|$.

Multiplikativní grupa: $\mathbb{Z}_n^* = \{x; 1 \leq x < n; \exists y : xy = 1\}$ $(* \bmod n, 1, ^{-1})$.

8.4 Řešení rovnic

Věta:
Rovnice $ax \equiv_n b$ pro $a, b, n \in \mathbb{Z}$ má řešení $\Leftrightarrow \gcd(a, n)b$ a existuje program s časovou složitostí $O(N^3)$, které to řešení najde.

Důkaz:
Rovnice je ekvivalentní $ax - ny = b$.

Označme si $g := \gcd(a, n)$. $g \backslash ax, g \backslash ny$. Pokud $g \perp b$, pak levá strana je a pravá není dělitelná g, takže to nemá řešení.

Vyřeší se to Euklidovým algoritmem. Každý mezivýsledek je lineární kombinace typu $\alpha x + \beta y$. I výsledek musí být zapsatelný tímto způsobem. Zapamatujeme si α a β a to použijeme pro obnovení výsledku, tyto α a β vynásobíme g a vypadne výsledek.

Eulerova funkce je $\varphi(n) = |\{x; 1 \leq x < n \wedge x \perp n\}|$.

Pro všechna n je to $n - 1$. Mocnina prvočísla $n = p^k$. Pak to vyjde $(p-1) \cdot p^{k-1}$. Pro $a, b \in \mathbb{Z}, a \perp b; \varphi(ab) = \varphi(a)\varphi(b)$.

Eulerova věta:
$\forall n, a \in \mathbb{Z}, a \perp n; a^{\varphi(n)} \equiv_n 1$.

Důkaz:
Buď $m > 0$ nejmenší takové, že $a^m \equiv_n 1$. Vezmeme posloupnost $A := \left\{a^0, a^1, a^2, \ldots, a^{m-1}\right\}$. Určitě se ta posloupnost začne někdy opakovat (řekněme, že pro i, j jsou stejné výsledky). Pak $a^{j-i} \equiv_n 1$. $A \subseteq \mathbb{Z}_N^*$. Použijeme Lagrangerovu větu, proto $m \backslash \varphi(n)$. Mohu rozepsat jako $(a^m)^{\frac{\varphi(n)}{m}} \equiv_n 1^{\frac{\varphi(n)}{m}}$.

8.5 Fermatův test

Vezme se vzorec $a^{n-1} \pmod n = 1$. Vezmu náhodné číslo, zkusím, jestli je to nesoudělné (pokud není nesoudělné, tak to není prvočíslo), pokud to nevyjde jedna, tak je také složené a jinak se neví.

Bohužel existují Carmichaelova čísla, která takové číslo (fermatova svědka) nemají.

Věta:
Pokud to číslo není carmichaelovo ani prvočíslo, pak existuje alespoň $\frac{n-1}{2}$ fermatových svědků.

Důkaz:
Mějme $H := \left\{a \in \{1, \ldots, n-1\}; a^{n-1} \pmod n = 1@, a \perp n\right\}$. H je podgrupa $\mathbb{Z}_n^*$. Každé z nich je invertibilní. Je tam jednička, je tam součin dvou takových čísel.

Protože to číslo není carmichaelovo, musí existovat jeden svědek, velikost H dělí velikost $\mathbb{Z}_n^*$, tedy je nejvýše polovina tohoto.

8.5.1 Vylepšení

Můžeme přeskočit kontrolu soudělnosti – soudělné číslo stejně hodí 0, takže neprojde potom.

8.6 Rabin-Mullerův test

Vylepšený Fermatův, potom ještě pro všechna $i; 2^i \backslash n - 1$. Pokud najdu takové, že nevychází jednička po použití exponentu $\frac{n-1}{2^i}$, našel jsem carmichaelovo číslo.