

1. Paralelní algoritmy

Pomocí počítačů řešíme stále složitější a rozsáhlejší úlohy a potřebujeme k tomu čím dál víc výpočetního výkonu. Rychlost a kapacita počítačů zatím rostla exponenciálně, takže se zdá, že stačí chvíli počkat. Jenže podobně rostou i velikosti problémů, které chceme řešit. Navíc exponenciální růst výkonu se určitě někdy zastaví – nečekáme třeba, že by bylo možné vyrábět transistory menší než jeden atom.

Jak si poradíme? Jedna z lákavých možností je zapřáhnout do jednoho výpočtu více procesorů najednou. Ostatně, vícejádrové procesory, které dneska najdeme ve svých stolních počítačích, nejsou nic jiného než miniaturní víceprocesorové systémy na jednom čipu.

Nabízí se tedy obtížnou úlohu rozdělit na několik částí, nechat každý procesor (či jádro) spočítat jednu z částí a nakonec jejich výsledky spojit dohromady. To se snadno řekne, ale s výjimkou triviálních úloh už obtížněji provede.

Pojďme se podívat na několik zajímavých paralelních algoritmů. Abychom se nemuseli zabývat detaily hardwaru konkrétního víceprocesorového počítače, zavedeme poměrně abstraktní výpočetní model, totiž hradlové sítě. Tento model je daleko paralelnější než skutečný počítač, ale přesto se techniky používané pro hradlové sítě hodí i prakticky. Konec konců sama vnitřní struktura procesorů se našemu modelu velmi podobá.

1.1. Hradlové sítě

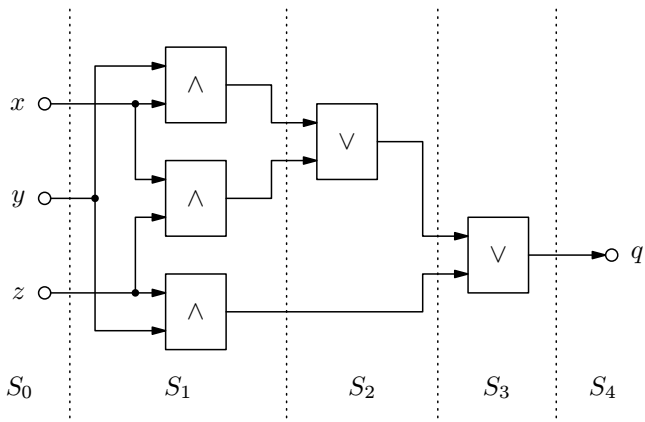
Hradlové sítě jsou tvořeny navzájem propojenými *hradly*. Každé hradlo přitom počítá nějakou (obecně libovolnou) funkci $\Sigma^k \rightarrow \Sigma$. Množina Σ je konečná abeceda, stejná pro celou síť. Přirozené číslo k udává počet vstupů hradla, jinak též jeho *aritu*.

Příklad: Často studujeme hradla *booleovská* pracující nad abecedou $\Sigma = \{0, 1\}$. Ta počítají jednotlivé logické funkce, například:

- *nulární funkce*: to jsou konstanty (FALSE = 0, TRUE = 1),
- *unární funkce*: identita a negace (NOT, \neg),
- *binární funkce*: logický součin (AND, & či \wedge), součet (OR, \vee), aritmetický součet modulo 2 (XOR, \oplus), ...

Propojením hradel pak vznikne *hradlová síť*. Než vyřkneme formální definici, pojďme se podívat na příklad jedné takové sítě na obr. 1.1. Síť má tři vstupy, pět booleovských hradel a jeden výstup. Na výstupu je přitom jednička právě tehdy, jsou-li jedničky přítomny na alespoň dvou vstupech. Vrací tedy hodnotu, která na vstupech převažuje, neboli *majoritu*.

Obecně každá hradlová síť má nějaké vstupy, hradla a výstupy. Hradla dostávají data ze vstupů sítě a výstupů ostatních hradel. Výstupy hradel mohou být připojeny na libovolně mnoho vstupů dalších hradel, případně na výstupy sítě. Jediné omezení je, že v propojení nesmíme vytvářet cykly.



Obr. 1.1: Hradlová síť pro majoritu ze tří vstupů

Nyní totéž formálněji:

Definice: *Hradlová síť* je určena:

- *Abecedou* Σ , což je nějaká konečná množina symbolů.
- Po dvou disjunktními konečnými množinami I (*vstupy*), O (*výstupy*) a H (*hradla*).
- Acyklickým orientovaným multigrafem (V, E) s množinou vrcholů $V = I \cup O \cup H$ (*multigraf* potřebujeme proto, abychom uměli výstup jednoho hradla připojit současně na více různých vstupů jiného hradla).
- Zobrazením F , které každému hradlu $h \in H$ přiřadí nějakou funkci $F(h) : \Sigma^{a(h)} \rightarrow \Sigma$, což je funkce, kterou toto hradlo vykonává. Číslo $a(h)$ říkáme *arita* hradla h .
- Zobrazením $z : E \rightarrow \mathbb{N}$, jež o hranách vedoucích do hradel říká, kolikátému argumentu funkce odpovídají. (Na hranách vedoucích do výstupů necháváme hodnotu této funkce nevyužitou.)

Přitom jsou splněny následující podmínky:

- Do vstupů nevedou žádné hrany.
- Z výstupů nevedou žádné hrany. Do každého výstupu vede právě jedna hrana.
- Do každého hradla vede tolik hran, kolik je jeho arita. Z každého hradla vede alespoň jedna hrana.
- Všechny vstupy hradel jsou zapojeny. Tedy pro každé hradlo h a každý jeho vstup $j \in \{1, \dots, a(h)\}$ existuje právě jedna hrana e , která vede do hradla h a $z(e) = j$.

Na obrázcích většinou sítě kreslíme podobně jako elektrotechnická schémata: místo více hran z jednoho hradla raději nakreslíme jednu, která se cestou rozvětví. V místech křížení hran tečkou rozlišujeme, zda jsou hrany propojeny či nikoliv.

Poznámka: Někdy se hradlovým sítím také říká *kombinační obvody* a pokud pracují nad abecedou $\Sigma = \{0, 1\}$, tak *booleovské obvody*.

Definice: *Výpočet sítě* postupně přiřazuje hodnoty z abecedy Σ vrcholům grafu. Výpočet probíhá po *taktech*. V nultém taktu jsou definovány pouze hodnoty na vstupech sítě a v hradlech arity 0 (konstantách). V každém dalším taktu pak ohodnotíme vrcholy, jejichž všechny vstupní hrany vedou z vrcholů s již definovanou hodnotou.

Hodnotu hradla h přitom spočteme funkcí $F(h)$ z hodnot na jeho vstupech uspořádaných podle funkce z . Výstup sítě pouze zkopíruje hodnotu, která do něj po hraně přišla.

Jakmile budou po nějakém počtu taktů definované hodnoty všech vrcholů, výpočet se zastaví a síť vydá výsledek – ohodnocení výstupů.

Podle průběhu výpočtu můžeme vrcholy sítě rozdělit do vrstev (na obrázku 1.1 jsou naznačeny tečkovaně).

Definice: *i -tá vrstva* S_i obsahuje ty vrcholy, které vydají výsledek poprvé v i -tém taktu výpočtu.

Lemma (o průběhu výpočtu): Každý vrchol vydá v konečném čase výsledek (tedy patří do nějaké vrstvy) a tento výsledek se už nikdy nezmění.

Důkaz: Jelikož síť je acyklická, můžeme postupovat indukcí podle topologického pořadí vrcholů.

Pokud do vrcholu v nevede žádná hrana, vydá výsledek v 0. taktu. V opačném případě do v vedou hrany z nějakých vrcholů u_1, \dots, u_k , kteří leží v topologickém pořadí před v , takže už víme, že vydaly výsledek v taktech t_1, \dots, t_k . Vrchol v tedy musí vydat výsledek v taktu $\max_i t_i + 1$. A jelikož výsledky vrcholů u_1, \dots, u_k se nikdy nezmění, výsledek vrcholu v také ne. \square

Každý výpočet se tedy zastaví, takže můžeme definovat časovou a prostorovou složitost očekávaným způsobem.

Definice: *Časovou složitost* definujeme jako *hloubku sítě*, tedy počet vrstev obsahujících aspoň jeden vrchol. *Prostorovou složitost* bude rovna počtu hradel v síti. Všimněte si, že čas ani prostor nezávisí na konkrétním vstupu, pouze na jeho délce.

Poznámka (o aritě hradel): Kdybychom připustili hradla s libovolně vysokým počtem vstupů, mohli bychom jakýkoliv problém se vstupem délky n a výstupem délky ℓ vyřešit v jedné vrstvě pomocí ℓ kusů n -vstupových hradel. Každému hradlu bychom prostě přiřadili funkci, která počítá příslušný bit výsledku ze všech bitů vstupu.

To není ani realistické, ani pěkné. Jak z toho ven? Omezíme arity všech hradel nějakou pevnou konstantou, třeba dvojkou. Budeme tedy používat výhradně nulární, unární a binární hradla. (Kdybychom zvolili jinou konstantu, dopadlo by to podobně, viz cvičení 7.)

Poznámka (o uniformitě): Od běžných výpočetních modelů, jako je třeba RAM, se hradlové sítě liší jednou podstatnou vlastností – každá síť zpracovává výhradně vstupy jedné konkrétní velikosti. Řešením úlohy tedy typicky není jedna síť, ale posloupnost sítí pro jednotlivé velikosti vstupu. Všechny sítě přitom používají stejné typy hradel a stejnou abecedu. Takovým výpočetním modelům se říká *neuniformní*.

Obvykle budeme chtít, aby existoval algoritmus (klasický, neparalelní), který pro danou velikost vstupu sestrojí příslušnou síť. Tento algoritmus by měl běžet v polynomiálním čase – kdybychom dovolili i pomalejší algoritmy, mohli bychom během konstrukce provádět nějaký náročný předvýpočet a jeho výsledek zabudovat do struktury sítě. To je málokdy žádoucí.

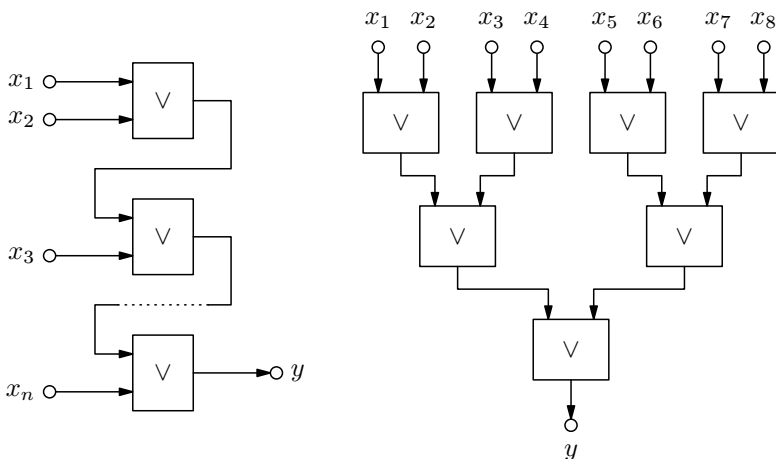
Hledá se jednička

Abychom si nový výpočetní model osahali, zkusme nejprve sestrojít booleovský obvod, který zjistí, zda se mezi jeho n vstupy vyskytuje alespoň jedna jednička. To znamená, že počítá n -vstupovou funkci OR.

První řešení (obrázek 1.2 vlevo): Spočítáme OR prvních dvou vstupů, pak OR výsledku s třetím vstupem, pak se čtvrtým, a tak dále. Každé hradlo závisí na výsledcích všech předchozích, takže výpočet běží striktně sekvenčně. Časová i prostorová složitost činí $\Theta(n)$.

Druhé řešení (obrázek 1.2 vpravo): Hradla budeme spojovat do dvojic, výsledky těchto dvojic opět do dvojic, a tak dále. Síť se tentokrát skládá z $\Theta(\log n)$ vrstev, které celkem obsahují $n/2 + n/4 + \dots + 1 = \Theta(n)$ hradel.

Logaritmičká časová složitost je pro paralelní algoritmy typická a budeme se jí snažit dosáhnout i u dalších problémů.



Obr. 1.2: Dvě hradlové sítě pro n -bitový OR

Cvičení

1. Jak vypadá všech 16 booleovských funkcí dvou proměnných?
2. Dokažte, že každou booleovskou funkci dvou proměnných lze vyjádřit pomocí hradel AND, OR a NOT. Proto lze každý booleovský obvod s nejvýše dvouvstupovými hradly upravit tak, aby používal pouze tyto tři typy hradel. Jeho hloubka přitom vzroste pouze konstanta-krát.
3. Pokračujeme v předchozím cvičení: dokažte, že stačí jediný typ hradla, a to NAND (negovaný AND). Podobně by stačil NOR (negovaný OR). Existuje nějaká další funkce s touto vlastností?
4. Sestavte hradlovou síť ze čtyř hradel NAND (negovaný AND), která počítá XOR dvou bitů.
5. Dokažte, že n -bitový OR nelze spočítat v menší než logaritmické hloubce.
6. Sestrojte hradlovou síť pro majoritu ze 4 vstupů.
7. Ukažte, že libovolnou booleovskou funkci s k vstupy lze spočítat booleovským obvodem hloubky $\mathcal{O}(k)$ s $\mathcal{O}(2^k)$ hradly. To speciálně znamená, že pro pevné k lze booleovské obvody s nejvýše k -vstupovými hradly překládat na obvody s 2-vstupovými hradly. Hloubka přitom vzroste pouze konstanta-krát.
- 8.* Exponenciální velikost obvodu z minulého cvičení je nepříjemná, ale bohužel nevyhnutelná: Dokažte, že pro žádné k neplatí, že všechny n -vstupové booleovské funkce lze spočítat obvody s $\mathcal{O}(n^k)$ hradly.
9. Ukažte, jak hradlovou síť s libovolnou abecedou přeložit na ekvivalentní booleovský obvod s nejvýše konstantním zpomalením. Abecedu zakódujte binárně, hradla simulujte booleovskými obvody.
10. Definujeme *výhybku* – to je analogie operátoru $?$: v jazyce C, tedy ternární booleovské hradlo se vstupy x_0 , x_1 a p , jehož výsledkem je x_p . Ukažte, že libovolnou k -vstupovou booleovskou funkci lze spočítat obvodem složeným pouze z výhybek a konstant. Srovnejte s cvičením 7. Jak by se naopak skládala výhybka z binárních hradel?
11. Dokažte, že každou booleovskou formuli lze přeložit na booleovský obvod. Velikost obvodu i jeho hloubka přitom budou lineární v délce formule.
- 12.**Ukažte, že obvody z minulého cvičení si vystačí s logaritmickou hloubkou v délce formule.
13. Místo omezení arity hradel bychom mohli omezit typy funkcí, řekněme na AND, OR a NOT, a požadovat polynomiální počet hradel. Tím by také vznikl realistický model, byť s trochu jinými vlastnostmi. Dokažte, že síť tohoto druhu s n vstupy lze přeložit na síť s omezenou aritou hradel, která bude pouze $\mathcal{O}(\log n)$ -krát hlubší. K čemu bylo nutné omezení počtu hradel?

1.2. Sčítání a násobení binárních čísel

Nalezli jsme rychlý paralelní algoritmus pro n -bitový OR. Zajímavější úlohou, jejíž paralelizace už nebude tak triviální, bude sčítání dvojkových čísel. Mějme

dvě čísla x a y zapsané ve dvojkové soustavě. Jejich číslice označme $x_{n-1} \dots x_0$ a $y_{n-1} \dots y_0$, přičemž i -tý řád má váhu 2^i . Chceme spočítat dvojkový zápis $z_n \dots z_0$ čísla $z = x + y$.

Školní algoritmus

Ihned se nabízí použít starý dobrý „školní algoritmus sčítání pod sebou“. Ten funguje ve dvojkové soustavě stejně dobře jako v desítkové. Sčítáme čísla zprava doleva, vždy sečteme x_i s y_i a přičteme přenos z nižšího řádu. Tím dostaneme jednu číslici výsledku a přenos do vyššího řádu. Formálně bychom to mohli zapsat třeba takto:

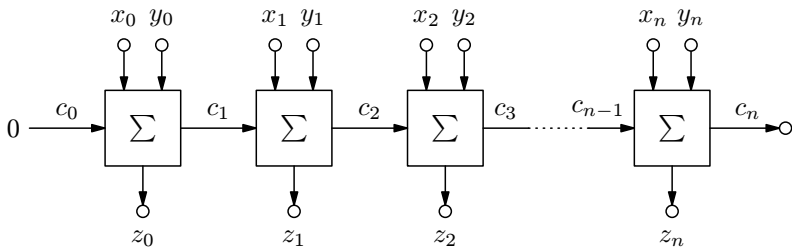
$$z_i = x_i \oplus y_i \oplus c_i,$$

kde z_i je i -tá číslice součtu, \oplus značí operaci XOR (součet modulo 2) a c_i je *přenos* z $(i - 1)$ -ního řádu do i -tého. Přenos do vyššího řádu nastane tehdy, pokud se nám potkají dvě jedničky pod sebou, nebo když se vyskytne alespoň jedna jednička a k tomu přenos z nižšího řádu. Čili tehdy, jsou-li mezi třemi xorovanými číslicemi alespoň dvě jedničky – k tomu se nám hodí již známý obvod pro majoritu:

$$c_0 = 0,$$

$$c_{i+1} = (x_i \wedge y_i) \vee (x_i \wedge c_i) \vee (y_i \wedge c_i).$$

O tomto předpisu snadno dokážeme, že funguje (zkuste to), nicméně pokud podle něj postavíme hradlovou síť, bude poměrně pomalá. Můžeme si ji představit tak, že je složena z nějakých podsítí („krabiček“), které budou mít na vstupu x_i , y_i a c_i a jejich výstupem bude z_i a c_{i+1} . To je hezky vidět na obrázku 1.3.



Obr. 1.3: Sčítání školním algoritmem

Každá krabička má sama o sobě konstantní hloubku, ovšem k výpočtu potřebuje přenos vypočítaný předcházející krabičkou. Jednotlivé krabičky proto musí ležet v různých vrstvách sítě. Časová i prostorová složitost sítě jsou tedy lineární, stejně jako sčítáme-li po bitech na RAMu.

Bloky a jejich chování

To, co nás při sčítání brzdí, je evidentně čekání na přenosy z nižších řádů. Jakmile je zjistíme, máme vyhráno – součet už získáme jednoduchým xorováním,

které zvládneme paralelně v čase $\Theta(1)$. Uvažujme tedy nad způsobem, jak přenosy spočítat paralelně.

Podívejme se na libovolný *blok* výpočtu školního algoritmu. Tak budeme říkat části sítě, která počítá součet bitů $x_j \dots x_i$ a $y_j \dots y_i$ v nějakém intervalu indexů $[i, j]$. Přenos c_{j+1} vystupující z tohoto bloku závisí kromě hodnot sčítanců už pouze na přenosu c_i , který do bloku vstupuje.

Pro konkrétní sčítance se tedy můžeme na blok dívat jako na nějakou funkci, která dostane jednobitový vstup (přenos zespona) a vydá jednobitový výstup (přenos nahoru). To je milé, neboť takové funkce existují pouze čtyři:

$f(x) = 0$	konstantní 0 , blok <i>pohlcuje</i> přenos
$f(x) = 1$	konstantní 1 , blok <i>vytváří</i> přenos
$f(x) = x$	identita (značíme $<$), blok <i>kopíruje</i> přenos
$f(x) = \neg x$	negace; ukážeme, že u žádného bloku nenastane

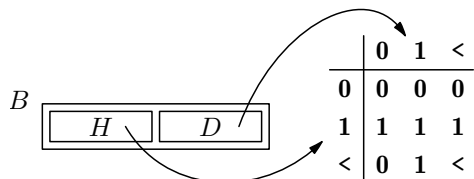
Této funkci budeme říkat *chování bloku*.

Jednobitové bloky se chovají velice jednoduše:

0	0	1	1
0	1	0	1
0	$<$	$<$	1

Blok prvního druhu vždy předává nulový přenos, ať už do něj vstoupí jakýkoliv – přenos tedy pohlcuje. Poslední blok naopak sám o sobě přenos vytváří, ať dostane cokoliv. Prostřední dva bloky se chovají tak, že samy o sobě žádný přenos nevytvorí, ale pokud do nich nějaký přijde, tak také odejde.

Větší bloky můžeme rozdělit na části a podle chování částí určit, jak se chová celek. Mějme blok B složený ze dvou menších podbloků H (horní část) a D (dolní). Chování celku závisí na chování částí takto:



Pokud vyšší blok přenos pohlcuje, pak ať se nižší blok chová jakkoli, složení obou bloků musí vždy pohlcovat. V prvním řádku tabulky jsou tudíž nuly. Analogicky pokud vyšší blok generuje přenos, tak ten nižší na tom nic nezmění. V druhém řádku tabulky jsou tedy samé jedničky. Zajímavější případ nastává, pokud vyšší blok kopíruje – tehdy záleží čistě na chování nižšího bloku.

Všimněme si, že skládání chování bloků je vlastně úplně obyčejné skládání funkcí. Nyní bychom mohli prohlásit, že budeme počítat nad tříprvkovou abecedou

a že celou tabulku dokážeme spočítat jedním jediným hradlem. Pojdme si přeci jen rozmyslet, jak bychom takovou operaci popsali čistě binárně.

Tři stavy můžeme zakódovat pomocí dvou bitů, řekněme jim třeba p a q . Dvojice (p, q) přitom může nabývat hned čtyř možných hodnot, my dvěma z nich přiřadíme stejný význam:

$$(1, *) = < \quad (0, 0) = \mathbf{0} \quad (0, 1) = \mathbf{1}.$$

Kdykoliv $p = 1$, blok kopíruje přenos. Naopak $p = 0$ odpovídá tomu, že blok posílá do vyššího řádu konstantní přenos, a q pak určuje, jaký. Kombinování bloků (skládání funkcí) pak můžeme popsat následovně:

$$\begin{aligned} p_B &= p_H \wedge p_D, \\ q_B &= (\neg p_H \wedge q_H) \vee (p_H \wedge q_D). \end{aligned}$$

Průchod přenosu blokem (dosazení do funkce) bude vypadat takto:

$$c_{j+1} = (p \wedge c_i) \vee (\neg p \wedge q).$$

Rozmyslete si, že tyto formule odpovídají výše uvedené tabulce. (Mimoходом, totéž by se mnohem přímočařeji formulovalo pomocí výhybek z cvičení 1.1.10.)

Paralelní sčítání

Od popisu chování bloků je už jenom krůček k paralelnímu předpovídání přenosů, a tím i k paralelní sčítačce. Bez újmy na obecnosti budeme předpokládat, že počet bitů vstupních čísel n je mocnina dvojky; jinak vstup doplníme zleva nulami.

Algoritmus bude rozdělen na dvě části:

První část spočítá chování všech *kanonických bloků* – tak budeme říkat blokům, jejichž velikost je mocnina dvojky a pozice je dělitelná velikostí (bloky téže velikosti se tedy nepřekrývají). Nejprve v konstantním čase stanovíme chování bloků velikosti 1, ty pak spojíme do dvojic, dvojice zase do dvojic atd., obecně v i -tém kroku spočteme chování všech kanonických bloků velikosti 2^i .

Druhá část pak dopočítá přenosy, a to tak, aby v i -tém kroku byly známy přenosy do řádů dělitelných $2^{\log n - i}$. V nultém kroku známe pouze $c_0 = 0$ a c_n , které spočítáme z c_0 pomocí chování bloku $[0, n]$. V prvním kroku pomocí bloku $[0, n/2]$ dopočítáme $c_{n/2}$, v druhém pomocí $[0, n/4]$ spočítáme $c_{n/4}$ a pomocí $[n/2, 3/4 \cdot n]$ dostaneme $c_{3/4 \cdot n}$, atd. Obecně v i -tém kroku používáme chování bloků velikosti $2^{\log n - i}$. Každý krok přitom zabere konstantní čas.

Celkově bude sčítací síť vypadat takto (viz obr. 1.4):

- $\Theta(1)$ hladin výpočtu chování bloků velikosti 1,
- $\Theta(\log n)$ hladin počítajících chování všech kanonických bloků,
- $\Theta(\log n)$ hladin dopočítávajících přenosy „zahušťováním“,
- $\Theta(1)$ hladin na samotné sečtení: $z_i = x_i \oplus y_i \oplus c_i$ pro všechna i .

Jde to ale rychleji, použijeme-li jednoduchý, téměř kouzelnický trik. Sestrojíme *kompresor* – to bude obvod konstantní hloubky, jenž na vstupu dostane tři čísla a vypočte z nich dvě čísla mající stejný součet jako zadaná trojice.

K čemu je to dobré? Máme-li sečíst n čísel, v konstantním čase dokážeme tento úkol převést na sečtení $2/3 \cdot n$ čísel (vhodně zaokrouhleno), to pak opět v konstantním čase na sečtení $(2/3)^2 \cdot n$ čísel atd., až nám po $\log_{3/2} n = \Theta(\log n)$ krocích zbudou dvě čísla a ta sečteme klasickou sčítačkou. Zbývá vymyslet kompresor.

Konstrukce kompresoru: Označme vstupy kompresoru x, y a z a výstupy p a q . Pro každý řád i spočteme součet $x_i + y_i + z_i$. To je nějaké dvoubitové číslo, takže můžeme jeho nižší bit prohlásit za p_i a vyšší za q_{i+1} .

Jinými slovy všechna tři čísla jsme normálně sečetli, ale místo abychom přenosy posílali do vyššího řádu, vytvořili jsme z nich další číslo, které má být k výsledku časem přičteno. To je vidět na obrázku 1.5.

Naše síť pro paralelní násobení nyní pracuje v čase $\Theta(\log n)$ – nejdříve v konstantním čase vytvoříme mezivýsledky, pak použijeme $\Theta(\log n)$ hladin kompresorů konstantní hloubky a nakonec jednu sčítačku hloubky $\Theta(\log n)$.

Jistou vadou na kráse ovšem je, že spotřebujeme $\Theta(n^2)$ hradel. Proto se v praxi používají spíš násobící sítě odvozené od rychlé Fourierovy transformace, s níž se potkáme v kapitole ??.

Cvičení

1. Modifikujte sčítací síť, aby odčítala.
2. Sestrojte hradlovou síť hloubky $\mathcal{O}(\log n)$, která porovná dvě n -bitová čísla x a y a vrátí jedničku, pokud $x < y$.
3. Ukažte, jak v logaritmicke hloubce otestovat, zda je dvojkové číslo dělitelné jedenácti.
- 4.** Pro ctitele teorie automatů: Dokažte, že každý regulární jazyk lze rozpoznávat hradlovou sítí logaritmicke hloubky. Ukažte, jak pomocí toho vyřešit všechna předchozí cvičení.
- 5.* Sestrojte hradlovou síť logaritmicke hloubky, která dostane matici sousednosti neorientovaného grafu a rozhodne, zda je graf souvislý.
6. Sestrojte hradlovou síť, která pro zadané dvojkové číslo $x_{n-1} \dots x_0$ spočítá dolní celou část z jeho dvojkového logaritmu, čili nejvyšší i takové, že $x_i = 1$.

1.3. Třídící síť

Ještě zkusíme paralelizovat jeden klasický problém, totiž třídění. Budeme k tomu používat *komparátorovou síť* – to je hradlová síť složená z *komparátorů*.

Jeden komparátor umí porovnat dvě hodnoty a rozhodnout, která z nich je větší a která menší. Nevrací však booleovský výsledek jako běžné hradlo, ale má dva výstupy: na jednom z nich vrací menší ze vstupních hodnot a na druhém tu větší.

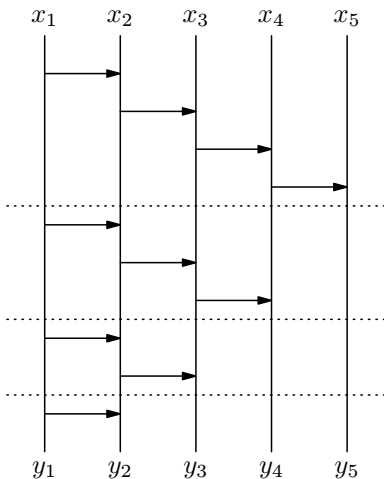
V našem formalismu hradlových sítí bychom mohli komparátor reprezentovat dvojicí hradel: jedno z nich by počítalo minimum, druhé maximum. Hodnoty, které třídíme, bychom považovali za prvky abecedy. Komparátorovou síť můžeme také snadno přeložit na booleovský obvod, viz cvičení 4.

Ještě se dohodneme, že výstupy komparátorů se nikdy nebudou větvit. Každý výstup přivedeme na vstup dalšího komparátoru, nebo na výstup sítě. Větvení by nám ostatně k ničemu nebylo, protože na výstupu potřebujeme vydat stejný počet hodnot, jako byl na vstupu. Nemáme přitom žádné hradlo, kterým bychom mohli hodnoty slučovat, a definice hradlové sítě nám nedovoluje výstup hradla „zahodit“.

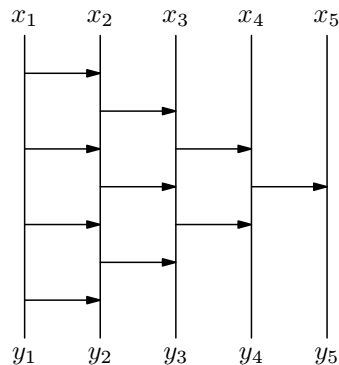
Důsledkem je, že výstup každé vrstvy, a tedy i celé sítě, je nějaká permutace prvků ze vstupu.

Jako rozcvičku zkusíme do řeči komparátorových sítí přeložit *bublínkové třídění*. Z něj získáme obvod na obrázku 1.6. Komparátory kreslíme jako šipky: shora do šipky vedou vstupy, zdola z ní vycházejí výstupy, větší výstup leží ve směru šipky.

Toto nakreslení ovšem poněkud klame – pokud síť necháme počítat, mnohá porovnání budou probíhat paralelně. Skutečný průběh výpočtu znázorňuje obrázek 1.7, na němž jsme všechny operace prováděné současně znázornili vedle sebe. Ihned vidíme, že paralelní bublínkové třídění pracuje v čase $\Theta(n)$ a potřebuje kvadratický počet komparátorů.



Obr. 1.6: Bublínkové třídění



Obr. 1.7: Skutečný průběh výpočtu

Bitonické třídění

Nyní vybudujeme rychlejší třídící algoritmus. Půjdeme na něj menší oklikou. Nejdříve vymyslíme síť, která bude umět třdit jenom něco – totiž bitonické posloupnosti. Z ní pak odvodíme obecné třídění. Bez újmy na obecnosti přitom bude-

me předpokládat, že každé dva prvky na vstupu jsou navzájem různé a že velikost vstupu je mocnina dvojky.

Definice: Posloupnost x_0, \dots, x_{n-1} je *čistě bitonická*, pokud ji můžeme rozdělit na nějaké pozici k na rostoucí posloupnost x_0, \dots, x_k a klesající posloupnost x_k, \dots, x_{n-1} . Jak rostoucí, tak klesající část mohou být prázdné či jednoprvkové.

Definice: Posloupnost x_0, \dots, x_{n-1} je *bitonická*, jestliže ji lze získat rotací (cyklickým posunutím) nějaké čistě bitonické posloupnosti. Tedy pokud existuje číslo j takové, že posloupnost $x_j, x_{(j+1) \bmod n}, \dots, x_{(j+n-1) \bmod n}$ je čistě bitonická.

Definice: *Separátor řádu n* je komparátorová síť S_n se vstupy x_0, \dots, x_{n-1} a výstupy y_0, \dots, y_{n-1} . Dostane-li na vstupu bitonickou posloupnost, vydá na výstup její permutaci s následujícími vlastnostmi:

- $y_0, \dots, y_{n/2-1}$ a $y_{n/2}, \dots, y_{n-1}$ jsou bitonické posloupnosti;
- $y_i < y_j$, kdykoliv $0 \leq i < n/2 \leq j < n$.

Jinak řečeno, separátor rozdělí bitonickou posloupnost na dvě poloviční a navíc jsou všechny prvky v první polovině menší než všechny v té druhé.

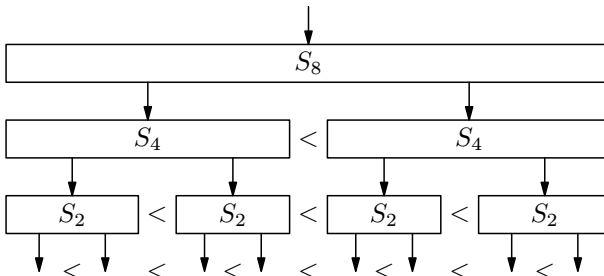
Lemma: Pro každé sudé n existuje separátor S_n konstantní hloubky, složený z $\Theta(n)$ komparátorů.

Důkaz tohoto lemmatu si necháme na konec. Nejprve předvedeme, k čemu jsou separátory dobré.

Definice: *Bitonická třídačka řádu n* je komparátorová síť B_n s n vstupy a n výstupy. Dostane-li na vstupu bitonickou posloupnost, vydá ji setříděnou.

Lemma: Pro libovolné $n = 2^k$ existuje bitonická třídačka B_n hloubky $\Theta(\log n)$ s $\Theta(n \log n)$ komparátory.

Důkaz: Konstrukce bitonické třídačky je snadná: nejprve separátorem S_n zadanou bitonickou posloupnost rozdělíme na dvě bitonické posloupnosti délky $n/2$, každou z nich pak separátorem $S_{n/2}$ na dvě části délky $n/4$, atd., až získáme jednoprvkové posloupnosti ve správném pořadí. Celkem použijeme $\log n$ hladin, každá hladina má konstantní hloubku a leží na ní $n/2$ komparátorů. \square



Obr. 1.8: Bitonická třídačka B_8

Bitonické třídičky nám nyní pomohou ke konstrukci třídičky pro obecné posloupnosti. Ta bude založena na třídění sléváním – nejprve se tedy musíme naučit slít dvě rostoucí posloupnosti do jedné.

Definice: *Slévačka řádu n* je komparátorová síť M_n s $2 \times n$ vstupy a $2n$ výstupy. Dostane-li dvě setříděné posloupnosti délky n , vydá setříděnou posloupnost vzniklou jejich slítním.

Lemma: Pro $n = 2^k$ existuje slévačka M_n hloubky $\Theta(\log n)$ s $\Theta(n \log n)$ komparátorů.

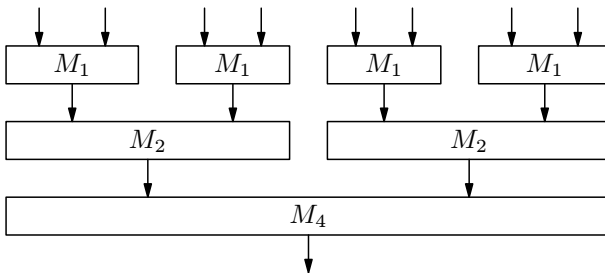
Důkaz: Stačí jednu vstupní posloupnost obrátit a „přilepit“ za tu druhou. Tím vznikne bitonická posloupnost, již setřídíme bitonickou třídičkou B_{2n} . \square

Definice: *Třídící síť řádu n* je komparátorová síť T_n s n vstupy a n výstupy, která pro každý vstup vydá jeho setříděnou permutaci.

Věta: Pro $n = 2^k$ existuje třídící síť T_n hloubky $\Theta(\log^2 n)$ složená z $\Theta(n \log^2 n)$ komparátorů.

Důkaz: Síť bude třidit sléváním, podobně jako algoritmus Mergesort z oddílu ?? . Vstup rozdělíme na n jednoprvkových posloupností. Ty jsou jistě setříděné, takže je slévačkami M_1 můžeme slít do dvouprvkových setříděných posloupností. Na ty pak aplikujeme slévačky $M_2, M_4, \dots, M_{n/2}$, až všechny části slijeme do jedné, setříděné.

Celkem provedeme $\log n$ kroků slévání, i -tý z nich obsahuje slévačky $M_{2^{i-1}}$ a ty, jak už víme, mají hloubku $\Theta(i)$. Celkový počet vrstev tedy činí $\Theta(1 + 2 + 3 + \dots + \log n) = \Theta(\log^2 n)$. Každý krok přitom potřebuje $\Theta(n \log n)$ komparátorů, což dává celkem $\Theta(n \log^2 n)$ komparátorů. \square

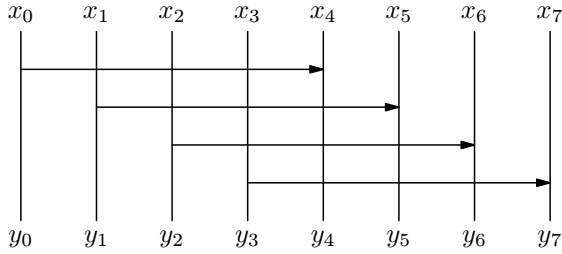


Obr. 1.9: Třídící síť T_8

Konstrukce separátoru

Zbývá dokázat, že existují slíbené separátory konstantní hloubky. Vypadají překvapivě jednoduše: pro $i = 0, \dots, n/2 - 1$ zapojíme komparátor se vstupy $x_i, x_{i+n/2}$, jehož minimum přivedeme na y_i a maximum na $y_{i+n/2}$.

Proč separátor separuje? Nejprve předpokládejme, že vstupem je čistě bitonická posloupnost. Označme m polohu maxima této posloupnosti; maximum bez újmy na obecnosti leží v první polovině (jinak celý důkaz provedeme „zrcadlově“). Označme



Obr. 1.10: Separátor S_8

dále k nejmenší index, pro který komparátor zapojený mezi x_k a $x_{n/2+k}$ hodnoty prohodí, tedy $k = \min\{i \mid x_i > x_{n/2+i}\}$.

Jelikož maximum je jedinečné, musí platit $x_m > x_{n/2+m}$, takže k existuje a navíc platí $0 \leq k \leq m < n/2$. Situace tedy odpovídá obrázku 1.11.

Nyní nahlédneme, že pro $i = k, \dots, n/2 - 1$ už komparátory vždy prohazují: Platí $x_i > x_{n/2+k}$ (pro $i \geq m$ je to vidět přímo, pro $i < m$ je $x_i \geq x_k > x_{n/2+k}$). Ovšem $x_{n/2+k} \geq x_{n/2+i}$, protože zbytek posloupnosti je klesající.

Separátor se tedy chová velice přímočaře: levá polovina výstupu vznikne slepením rostoucího úseku x_0, \dots, x_{k-1} s klesajícím úsekem $x_{n/2+k}, \dots, x_{n-1}$; pravou polovinu tvoří spojení klesajícího úseku $x_{n/2}, \dots, x_{n/2+k-1}$, rostoucího úseku x_k, \dots, x_{m-1} a klesajícího úseku $x_m, \dots, x_{n/2-1}$.

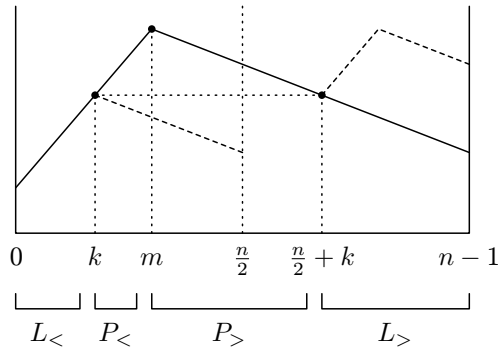
Snadno ověříme, že obě poloviny jsou bitonické: ta první je dokonce čistě bitonická, druhou lze na čistě bitonickou zrotovat díky tomu, že $x_{n/2-1} > x_{n/2}$.

Zbývá dokázat, že levá polovina je menší než pravá. Zdá se to být zřejmé z obrázku: křivku rozkrojíme vodorovnou tečkovanou linkou a části přeskládáme. Jenže nesmíme zapomínat, že x_k a $x_{n/2+k}$ jsou různé prvky, takže tečkovaná linka není ve skutečnosti vodorovná.

Proveďme podobnou úvahu precizně: Levou polovinu rozdělíme na rostoucí část $L_< = x_0, \dots, x_{k-1}$ a klesající část $L_> = x_{n/2+k}, \dots, x_{n-1}$; podobně pravou na $P_< = x_k, \dots, x_{m-1}$ a $P_> = x_m, \dots, x_{n/2+k-1}$ (ve výstupu prvky leží v jiném pořadí, ale to teď nevádí). Tyto části nyní porovnáme:

- $L_< < P_<$: obě části původně tvořily jeden společný rostoucí úsek;
- $L_< < P_>$: $\max L_< = x_{k-1} < x_{n/2+k-1} = \min P_>$ (kdyby neplatila prostřední nerovnost, mohli bychom snížit k);
- $L_> < P_<$: $\max L_> = x_{n/2+k} < x_k = \min P_<$;
- $L_> < P_>$: obě části původně tvořily jeden společný klesající úsek.

Doplňme, co se stane, pokud vstup není čistě bitonický. Zde využijeme toho, že separátor je symetrický, tudíž zrotujeme-li jeho vstup o p pozic, dostaneme o p pozic zrotované i obě poloviny výstupu. Podle definice ovšem pro každou bitonickou posloupnost existuje její rotace, která je čistě bitonická, a pro níž, jak už víme,



Obr. 1.11: Ilustrace činnosti separátoru

separátor funguje. Takže pro nečistou bitonickou posloupnost musí vydat výsledek pouze zrotovaný, což na jeho správnosti nic nemění. \square

Shrnutí

Nalezli jsme paralelní třídící algoritmus o časové složitosti $\Theta(\log^2 n)$, který využívá $\Theta(n \log^2 n)$ komparátorů. Dodejme, že jsou známé i třídící sítě hloubky $\Theta(\log n)$, ale jejich konstrukce je mnohem komplikovanější a dává obrovské multiplikační konstanty, jež brání praktickému použití.

Z dolního odhadu složitosti třídění v oddílu ?? navíc plyne, že logaritmický počet hladin je nejmenší možný. Máme-li totiž libovolnou třídící síť hloubky h , můžeme ji simulovat po hladinách a získat tak sekvenční třídící algoritmus. Jelikož na každé hladině může ležet nejvýše $n/2$ komparátorů, náš algoritmus provede maximálně $hn/2$ porovnání. Už jsme nicméně dokázali, že pro každý třídící algoritmus existují vstupy, na kterých porovná $\Omega(n \log n)$ -krát. Proto $h = \Omega(\log n)$.

Cvičení

1. Jak by vypadala komparátorová síť pro třídění vkládáním? Jak se bude její průběh výpočtu lišit od paralelního bublinkového třídění?
2. Navrhněte komparátorovou síť pro hledání maxima: dostane-li n prvků, vydá takovou permutaci, v níž bude poslední hodnota největší.
3. Navrhněte komparátorovou síť pro zatřídění prvku do setříděné posloupnosti: dostane $(n - 1)$ -prvkovou setříděnou posloupnost a jeden prvek navíc, vydá setříděnou permutaci.
4. Ukažte, jak komparátorovou síť přeložit na booleovský obvod. Každý prvek abecedy Σ reprezentujte číslem o $b = \lceil \log_2 |\Sigma| \rceil$ bitech a pomocí cvičení 1.2.2 sestrojte komparátory o $\mathcal{O}(\log b)$ hladinách.
5. Upravte algoritmus bitonického třídění, aby fungoval i pro vstupy, jejichž délka není mocninou dvojky.

6. Dokažte *nula-jedničkový princip*: pro ověření, že komparátorová síť třídí všechny vstupy, ji postačí otestovat na všech posloupnostech nul a jedniček.
- 7.* *Batcherovo třídění*: Stejně složitosti paralelního třídění lze také dosáhnout následujícím rekurzivním algoritmem pro slévání setříděných posloupností:

Procedura BMERGE

Vstup: Setříděné posloupnosti (x_0, \dots, x_{n-1}) a (y_0, \dots, y_{n-1})

1. Je-li $n \leq 2$, vyřešíme triviálně.
2. $(a_0, \dots, a_{n-1}) \leftarrow \text{BMERGE}((x_0, x_2, \dots, x_{n-2}), (y_0, y_2, \dots, y_{n-2}))$
3. $(b_0, \dots, b_{n-1}) \leftarrow \text{BMERGE}((x_1, x_3, \dots, x_{n-1}), (y_1, y_3, \dots, y_{n-1}))$

Výstup: $(a_0, \min(a_1, b_0), \max(a_1, b_0), \min(a_2, b_1), \max(a_2, b_1), \dots, b_{n-1})$

Pomocí předchozího cvičení dokažte, že tato procedura funguje. Zapište tento algoritmus ve formě třídící sítě.