In-place BFS a DFS

v lineárním čase na modelu RAM

Tomáš Sláma

7. 4. 2020

Úvod

Předpoklady

- pracujeme s modelem RAM
  - paměť je pole slov velikosti w
  - operace se slovy (čtení, psaní, přístupy) jsou konstantní
  - vstup je na prvních $n \in \mathbb{N}$ slovech
  - velikost $w = \Omega(\log n)$
  - restore model: vstup může být měněn v průběhu výpočtu, ale na jeho konci musí být v původním stavu
- graf je zadán v setříděném seznamu sousedů

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	5	7	9	12	14	17	12	2	5	1	3	4	2	4	2	3	5	1	4
	n			T			$2m$												

Obrázek: setřízená reprezentace

DFS

Přehled

1. graf převedeme do speciální reprezentace
2. budeme v ní kódovat stav DFS (a trochu ji tím rozbijeme)
3. obnovíme původní stav
4. převedeme zpět do setřízené reprezentace

Disclaimer

operace	*význam*	*podpora*
`pre-process`	po vstoupení do vrcholu	✓
`pre-explore`	po iteraci přes souseda	
`post-process`	po zpracovaní všech sousedů	✓
`post-explore`	po vrácení z vrcholu	

Tabulka: podporované funkce DFS

Reprezentace

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	5	7	9	12	14	17	12	2	5	1	3	4	2	4	2	3	5	1	4
	n	T					$2m$												

	0	1	2	3	4	5	6	**7**	8	**9**	10	11	**12**	13	**14**	15	16	**17**	18
A	5	7	9	12	14	17	12	9	17	7	12	14	9	14	9	12	17	7	14
	n	T					$2m$												

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	5	9	7	9	9	7	12	**1**	17	**2**	12	14	**3**	14	**4**	12	17	**5**	14
	n	T					$2m$												

Obrázek: setřízená, pointerová a prohozená reprezentace

- pozor na stupně 0 (zachovávají si jména vrcholů)!

Lemma 1

Setříděná reprezentace jde do pointerové reprezentace převést in-place v čase $\mathcal{O}(n)$.

Důkaz

Nastavíme $A[i] = T[A[i]]\,\forall i \in \{n+2, \ldots, n+m+2\}$
a $T[v] = v$ (pro vrcholy v stupně 0). $\square$

Lemma 2

Pointerová jde do prohozené reprezentace (a zpět) převést in-place v čase $\mathcal{O}(n)$.

Důkaz

Převod $\rightarrow$:

$$T[i] = A[T[i]] \text{ a } A[T[i]] = i \quad \forall i \in \{1, \ldots, n\}, i \neq T[i]$$

Převod $\leftarrow$:

$$T[A[i]] = i \text{ a } T[A[i]] = i \quad \forall i \in \{n+2, \ldots, n+m+2\}, A[i] < n$$

$\square$

PROHOZENÁ → SETŘÍZENÁ

Lemma 3

Prohozená jde do setřízené reprezentace převést in-place v čase $\mathcal{O}(n)$.

Důkaz

Nejprve nastavíme

$$A[i] = A[A[i]]\ \forall i \in \{n+2, \dots, n+m+2\}, A[i] > n$$

a

$$T[i] = A[T[i]]\ \forall i \in \{1, \dots, n\}$$

Pak postupně hledáme jména vrcholů v na pozicích p v každém poli sousednosti a nastavujeme:

$$A[p] = T[v] \text{ a } T[v] = p$$

Nakonec opravíme vrcholy stupně 0 procházením T a nastavením

$$T[i] = T[i-1]\ \forall i \in \{1, \dots, n\}, T[i] = i$$

$\square$

- důkaz z článku je problematický[1], jelikož nepoznáme, když procházíme nové pole sousednosti

9	10	11	12	13	14	15	16	17
3	**4**	5	6	7	12	13	**5**	16

Obrázek: problematický případ

Důkaz (fix)

Při nalezení chybného prohození iterujeme o rozdíl správného v s aktuálním a prohazujeme zpět. $\square$

[1] Pokud jsem ho tedy pochopil správně :).

DFS

Vrcholy se stupněm ≥ 2

Invariant

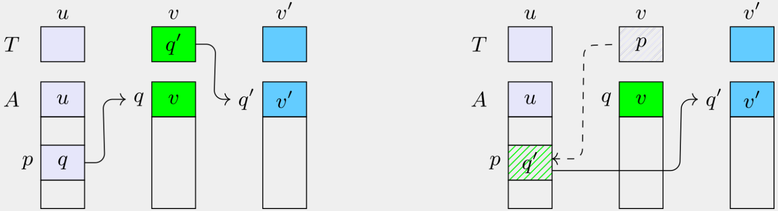
Vrchol v je bílý $\iff A[T[v]] \le n$.

- stav vrcholů budeme udržovat přes invariant
- na začátku platí pro všechny vrcholy

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	5	9	7	9	9	7	12	**1**	17	**2**	12	14	**3**	14	**4**	12	17	**5**	14
	n	$\underbrace{}_{T}$					$2m$												

Obrázek: prohozená reprezentace

- jsme ve vrcholu u a iterujeme přes jeho sousedy
- na pozici p jsme narazili na pointer q do bílého vrcholu v, jehož nejmenší soused je na pozici q'
- nastavíme $T[v] = p$ a $A[p] = q'$

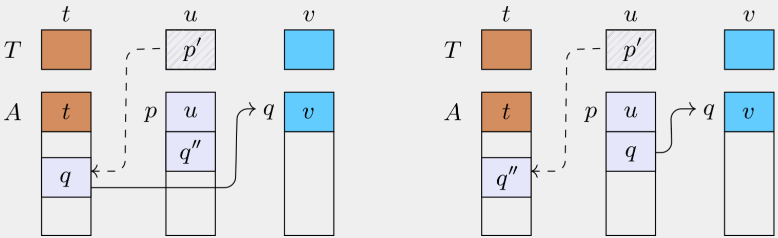


Obrázek: diagram konstrukce obráceného pointeru

- pozorování: vrchol v už není bílý

Problémy s obrácenými pointery

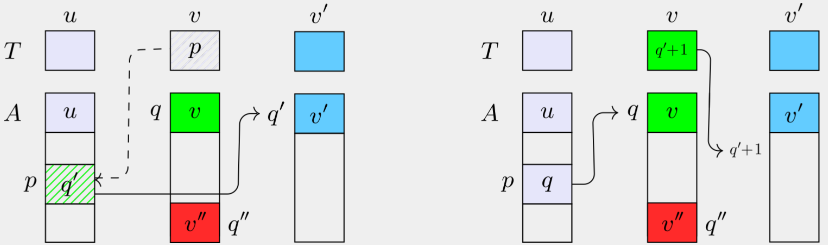
- na pozici p uvažujeme první hranu vrcholu u (uložená na $A[T[u]]$) směřující do v: kam nasměrovat obrácený pointer?
  1. uložit do $A[p]$ – ztratíme pojem o tom, kde u začíná
  2. uložit do $T[u]$ – tam už je obrácený pointer
- problém vyřešíme tím, že ho nebudeme řešit:
  - pokud by měl nastat, tak prohodíme $T[u]$ s $A[p+1]$
  - jelikož jsou pointery setřízené, tak prohození poznáme



Obrázek: problémy a řešení konstrukce obráceného pointeru

Backtrackování

- po prozkoumání sousedů v narazíme na pozici q'' na vrchol v''
  - poznáme tak, že $A[q''] \leq n$
- iterujeme zpět do té doby, než $A[p] \leq n$



Obrázek: backtrackování přes obrácený pointer

- vrchol v už je zase bílý – vyřešíme nastavením $q' = q' + 1$
  - stupně všech vrcholů jsou ≥ 2, q' ukazuje na pointer

DFS

Průběh algoritmu

Invarianty před a po follow/backtrack

1. Vrchol v je bílý $\iff$ není startovní a $1 \leq A[T[v]] \leq n$.
2. Každý šedočerný vrchol (kromě startovního) na aktuální DFS cestě si ukládá obrácený pointer na pozici $T[v]$, která ukazuje, kde byl pointer na v původně uložen v poli sousednosti svého rodiče.
3. První pointer aktuálně neprocházeného šedočerného vrcholu v (uložený v $T[v]$), ukazuje na druhou pozici pole sousedů nějakého vrcholu.

~~Pseudokód~~ Python

dfs.py

DFS

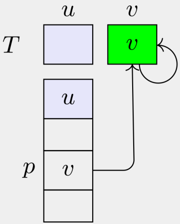
Vrcholy se stupněm 0

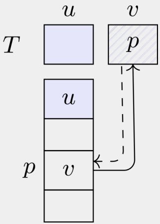
Rozšíření invariantu

Invariant

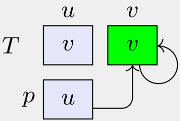
Vrchol v je bílý $\iff A[T[v]] \leq n \vee T[v] = v$.

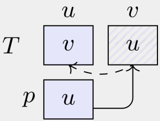
- chováme se normálně





- vytvoříme pointer do $T[u]$





- na rozdíl od vrcholů stupně 2 po sobě neuklízíme
  - vlastně vytváříme smyčky (bude se hodit při obnovení)

DFS

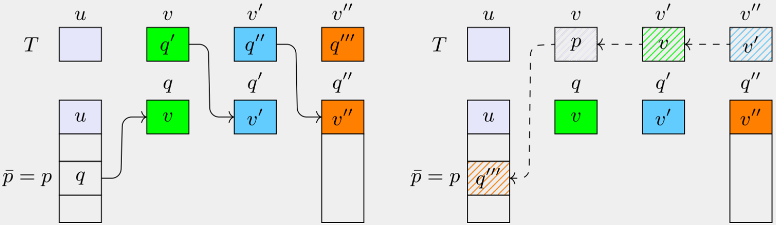
Vrcholy se stupněm 1

Dokončení invariantu

Invariant

Vrchol v je bílý $\iff$ $(\underbrace{T[v] = v}_{\text{stupeň 0}} \vee \underbrace{T[v] > n}_{\text{stupeň 1 a 2}}) \wedge \underbrace{A[T[v]] \leq n}_{\text{stupeň} \geq 2}$.

- z vrcholu u stupně ≥ 2 navštívíme bílý vrchol v stupně 1
- podle stupně jediného souseda v' vrcholu v se chováme různě:
  - 0) jako jsme popisovali výše
  - 1) nastavíme $T[v'] = v$ a pokračujeme
  - ≥ 2) nastavíme $A[\overline{p}] = T[v'']$ a $T[v''] = v'$



DFS

Obnovení

Obnovení

$\geq 2)$ stačí nastavit $T[v] = T[v] - 1$

0) tvoří smyčku, nebo ukazuje na druhou pozici vrcholu:
   - iterujeme přes pole sousednosti a hledáme $v = A[p] : v \leq n$:
     - $T[v] = p$ (smyčka, stupeň 2)
     - $T[v] \leq n \wedge T[T[v]] = p + 1$ (prohozený 1 a 2 vrchol, stupeň 2)
     - $T[v] \leq n \wedge T[T[v]] = v$ (stupeň 1)

1) postupně hledáme vrcholy v' stupně 1
   - pokud ještě nebyly opraveny, obracíme prohozené pointery

BFS

Přehled

1. graf získáme v setřízené reprezentaci
2. T zkomprimujeme na $\mathcal{T}$ s tím, že:
    - zachováme konstantní přístup
    - uvolníme lineární počet bitů
3. uvolněné místo použijeme na datovou strukturu pro ukládání stavů vrcholů
4. provedeme BFS
5. $\mathcal{T}$ dekomprimujeme na T, čímž obnovíme původní stav

Color choice dictionary

- zobecnění struktury choice dictionary
- udržuje $S_0, \ldots, S_{c-1}$ podmnožin $\{1, \ldots, n\}$, kde:

$$\bigcap_{i=0}^{c-1} S_i = \emptyset \qquad \bigcup_{i=0}^{c-1} S_i = \{1, \ldots, n\}$$

- vyžaduje řádově nc bitů paměti

operace	význam	složitost
`setColor(v, c)`	nastaví barvu v na c	$\mathcal{O}(1)$
`getColor(v)`	získá barvu v	$\mathcal{O}(1)$
`choice(c)`	získá libovolný v s barvu c	$\mathcal{O}(1)$

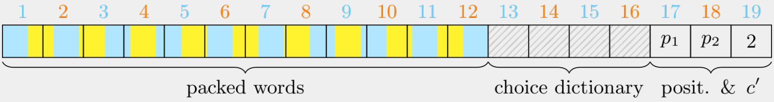
Tabulka: podporované operace CCD

Komprimace T

- potřebujeme uvolnit nc bitů

- najdeme pozice, kde se mění $c+1$ MSbitů ve slovech z T
  - díky setříděnosti jich bude právě 2^{c-1}
  - každé slovo o tolik bitů zkrátíme
  - zapamatujeme si pozice změn v T

- dohromady $nw - n(w-(c+1)) = n(c+1) = nc + n$
  - omezení: $n \geq 2^{c-1} w$, abychom mohli uložit pozice



Obrázek: komprimovaná reprezentace $\mathcal{T}$

Získávání hodnot z $\mathcal{T}$

1. bitovými operacemi získáme $w - (c + 1)$ LSbitů slova
   - i-té slovo v $\mathcal{T}$ začíná na pozici $(w - (c + 1))(i - 1)$
2. procházíme postupně 2^{c-1} pozic, podle kterých zrekonstruujeme $c + 1$ MSbitů

Průběh BFS

```
D = ColorChoiceDictionary(WHITE, LIGHT, DARK, BLACK)
D.setColor(start, LIGHT)

while choice(LIGHT) is not None:
    while choice(LIGHT) is not None:
        v = D.choice(LIGHT)  # pop node

        # open all white neighbours
        for i in range(T[v], T[v+1]):
            if D.getColor(A[i]) is WHITE:
                D.setColor(A[i], DARK)

        D.setColor(v, BLACK)  # close the node

    LIGHT, DARK = DARK, LIGHT  # next round
```

Díky za pozornost!

Zdroje a dodatečné materiály

- Torben Hagerup.
  Small uncolored and colored choice dictionaries.
  CoRR, abs/1809.07661, 2018.
- Frank Kammer and Andrej Sajenko.
  Linear-time in-place DFS and BFS in the restore model.
  CoRR, abs/1803.04282, 2018.
- Frank Kammer and Andrej Sajenko.
  Space efficient (graph) algorithms.
  `https://github.com/thm-mni-ii/sea`, 2018.
- Tomáš Sláma.
  Zdrojový kód k prezentaci.
  `https://github.com/xiaoxiae/inline-bfs-dfs-presentation`.