ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

Bakalárska práca

Študijný odbor: **Informatika**

**Pavol Šurin**

**Experimentálne porovnanie prioritných frontov**

Vedúci práce: **Ing. Peter Jankovič PhD.**

Reg.č. X/2021 Máj 2021

**Čestné vyhlásenie**

Prehlasujem, že som bakalársku prácu *Experimentálne porovnanie prioritných frontov* vypracoval samostatne pod vedením Ing. Peter Jankovič PhD, a uviedol v nej všetky použité literárne a iné odborné zdroje v súlade s právnymi predpismi, vnútornými predpismi Žilinskej univerzity a vnútornými aktmi riadenia Žilinskej univerzity a Fakulty riadenia a informatiky.

V Žiline, dňa .............. ..........................................

Meno Priezvisko

**Poďakovanie**

Na tomto mieste by som chcela poďakovať vedúcemu diplomovej práce.... za cenné pripomienky a odborné rady, ktorými prispel k vypracovaniu tejto XXXXX práce. Taktiež ďakujem môjmu ... . Zároveň ďakujem mojej rodine a priateľom za ich nekonečnú podporu a trpezlivosť.

**ABSTRAKT V ŠTÁTNOM JAZYKU**

PRIEZVISKO, Meno: Názov témy záverečnej práce. [Bakalárska práca]. – Žilinská Univerzita v Žiline. Fakulta riadenia a informatiky; Katedra informatiky. – Školiteľ/Vedúci: Ing. Oľga Chovancová – Stupeň odbornej kvalifikácie: magister. – Mesto: FRI UNIZA, 20xx. Počet strán

(napr. 35 s.)

Cieľom bakalárskej práce je ...

**Kľúčové slová**: kľúčové slovo 1, kľúčové slovo 2, kľúčové slovo 3, ... kľúčové slovo x.

**ABSTRAKT V CUDZOM JAZYKU**

The aim of the thesis is to .......

**Key words:** key word 1, key word 2, key word 3, ... , key word x.

**Obsah**

Obsah

[Zoznam obrázkov iv](#_Toc65508972)

[Zoznam tabuliek v](#_Toc65508973)

[Zoznam skratiek vi](#_Toc65508974)

[Úvod 1](#_Toc65508975)

[1 Teoretická implementácia prioritného frontu 2](#_Toc65508976)

[1.1 Abstraktná údajová štruktúra prioritný front 2](#_Toc65508977)

[1.2 Spôsoby implementácie prioritného frontu 3](#_Toc65508978)

[Binárna halda 3](#_Toc65508979)

[Binomiálna halda 4](#_Toc65508980)

[Fibonacciho halda 6](#_Toc65508981)

[Párovacia halda 6](#_Toc65508982)

[1.3 Zhrnutie teoretických časových náročnosti operácií 7](#_Toc65508983)

[2 Praktická implementácia prioritného frontu 8](#_Toc65508984)

[3 Návrh testov pre overenie výkonnosti implementácií prioritných frontov 9](#_Toc65508985)

[4 Vyhodnotenie výsledkov testov implementácií prioritných frontov 10](#_Toc65508986)

[Zoznam použitej literatúry 11](#_Toc65508987)

[Zoznam príloh 12](#_Toc65508988)

# Zoznam obrázkov

# Zoznam tabuliek

# Zoznam skratiek

# Úvod

Cieľom bakalárskej/diplomovej práce je ...... Náplňou prvej časti práce je oboznámenie sa s problematikou ............... Ďalšia časť sa zaoberá rozborom existujúcich ............... Ďalšia kapitola popisuje spôsob implementácie ................. Posledná časť práce ............................

Napísať o tom čo to zlepší, alebo aký je zmysel toho čo robíte alebo aký to ma dopad.

Postup práce:

1. Oboznámenie sa s problematikou .........
2. Rozbor existujúcich .......
3. Implementácia v....
4. Experimentálne porovnanie .....

# Teoretická implementácia prioritného frontu

V tejto časti sa budeme venovať prioritnému frontu ako abstraktnej údajovej štruktúry a jeho jednotlivým implementáciám. Cieľom tejto časti je popísanie rôznych spôsobov implementovania prioritného frontu, rovnako ako aj časové náročnosť ich jednotlivých operácií.

## Abstraktná údajová štruktúra prioritný front

Prioritný front predstavuje abstraktnú údajovú štruktúru, ktorá uchováva dáta zoradené na základe priority, ktorá bola dátam priradená. Jednotlivé dáta sú z prioritného frontu vyberané na základe priority, kde prvé sa vyberú dáta s najvyššou prioritou. Podporovanými operáciami a funkčnosťou sa prioritný front podobá údajovým štruktúra front a zásobník.

Podľa označenia priorít rozdeľujeme prioritný front do dvoch skupín:

Min-heap – s rastúcim označením priority, klesá priorita prvku

Max-heap – s rastúcim označením priority, rastie priorita prvku

Kvôli jednoduchosti budú všetky implementácie prioritných frontov typu min-heap.

Operácie prioritného frontu

Prioritný front musí podporovať prinajmenšom nasledovné operácie:

Vlož(Q, K, X) -> void

Vloží do prioritného frontu Q dáta X, ku ktorým je priradená priorita K

Vyber(Q) -> X

Vyberie a odstráni z prioritného frontu Q dáta X, ktoré mali priradenú najvyššiu prioritu

Nad prioritným frontom môžeme ďalej zadefinovať nasledovné operácie:

Vráť-minimum(Q) -> X

Vráti z prioritného frontu Q dáta X, ktoré majú priradenú najvyššiu prioritu

Spoj(Q1, Q2) -> Q

Spojí prioritné fronty Q1 a Q2 do jedného frontu Q, ktorý následne vráti

## Spôsoby implementácie prioritného frontu

### Binárna halda

Jedná sa o implementáciu prioritného frontu pomocou úplného binárneho stromu, kde predok prvku nazývame rodičom a potomkov synmi. Tento strom musí dodržiavať vlastnosť haldy[[1]](#footnote-1). Spomenutý strom je možne efektívne implementovať pomocou poľa.

Operácie

Vlož(Q, K, X) -> void

Prvok X je vložený spolu s priradenou prioritou K na prvé voľne miesto v binárnom strome, následne je vymieňaný s otcom, pokiaľ nie je splnená vlastnosť haldy. Keďže maximálny počet predkov prvku, s ktorými môžeme prvok vymeniť je , časová náročnosť samotnej operácie je . Ak je strom implementovaný pomocou poľa, môže počas vkladania prvku dôjsť k rozšíreniu poľa, čo by zvýšilo časovú náročnosť operácie v najhoršom prípade na .

Vyber(Q) -> X

Prvok X s najväčšou prioritou K, ktorý sa nachádza na začiatku zoznamu je vymenený s posledným prvkom a odoberieme ho zo zoznamu. Následne je prvok na začiatku zoznamu vymieňaný s jeho potomkami, pokiaľ nebude splnená vlastnosť haldy. Časová náročnosť tejto implementácie je , keďže maximálny počet postupných výmen s potomkami prvku je .

Vráť-minimum(Q) -> X

Vráti hodnotu prvku X s najväčšou prioritou K, ktorý sa nachádza na začiatku zoznamu. Prístup k tomuto prvku je konštantný, preto časová náročnosť danej operácie je .

Spoj(Q1, Q2) -> Q

Jednotlivé prvky z oboch háld sú vložené do novej haldy a obe haldy sú zničené. Nová halda je následne utriedená, aby dodržiavala vlastnosť haldy. Časová náročnosť utriedenia prvku, aby bola vlastnosť haldy je , čo pre n prvkov znamená, že časová náročnosť operácie Spoj je . Dá sa však dokázať, že amortizovaná časová náročnosť je (viz. Cormen a kol., 2009, s. 156).

Využitie

Triedenie prvkov pomocou heapsort.

### Binomiálna halda

Ide o prioritný front implementovaný pomocou zreťazeného zoznamu binomiálnych stromov[[2]](#footnote-2) zoradených vzostupne podľa ich indexu tak, že v zozname je vždy najviac jeden strom s daným indexom. Ide o front ktorý umožňuje efektívne spájanie frontov, preto ho môžeme považovať aj za spojiteľný prioritný front[[3]](#footnote-3).

Operácie

Vlož(Q, K, X) -> void

Vytvoríme novú binomiálnu haldu Q1, ktorý obsahuje len jeden prvok X s prioritou K. Tento front následne spojíme s frontom Q. Vidíme, že časová náročnosť tejto operácie je závislá len od časovej náročnosti spájania. Ide teda o časovú náročnosť .

Vyber(Q) -> X

V zozname binomiálnych stromov nájdeme ten binomiálny strom, ktorého koreň ma najväčšiu prioritu. Minimálny strom je izolovaný a potomkovia koreňa tohto stromu sú následne spojené so zoznamom binomiálnych stromov. Minimálny prvok je následne odstránený. Hľadanie minimálneho prvku má časovú náročnosť , to však môžeme urýchliť vedením smerníka k minimálnemu prvku, čo nám zefektívni operáciu na , budeme však musieť pri každej operácií Spoj tento smerník aktualizovať. Samotné spájanie má časovú náročnosť . Celková časová náročnosť operácie je teda .

Vráť-minimum(Q) -> X

V zozname binomiálnych stromov nájdeme a vrátime koreň binomiálneho stromu, ktorý ma najväčšiu prioritu. Táto operácia má časovú náročnosť , čo však môže byť urýchlené vedením smerníka k minimálnemu prvku, čo nám zefektívni operáciu na časovú náročnosť .

Spoj(Q1, Q2) -> Q

Vytvoríme novú binomiálnu haldu Q, ktorá v sebe obsahuje jednotlivé binomiálne stromy z binomiálnych háld Q1 a Q2 zoradených vzostupne podľa indexu. Následne sú všetky binomiálne stromy s rovnakým indexom prepojené[[4]](#footnote-4). Táto operácia je ekvivalentná k binárnemu sčítaniu. Maximálny počet prepojení je m. Časová náročnosť tejto operácie je teda (viz. Brown., 1978, s. 301).

### Fibonacciho halda

Fibonacciho halda je implementácia prioritného frontu, ktorá vznikla rozšírením Binomiálnej haldy. Rovnako ako pri Binomiálnej halde, aj tu použitý zreťazený zoznam binomiálnych stromov, neplatí však obmedzenie na počet stromov s rovnakým indexom a nutnosť zoradenia. Taktiež ide o spojiteľný prioritný front.

Operácie

Vlož(Q, K, X) -> void

Prvok X s prioritou K vložíme na ľubovoľné miesto v zozname. Ak sa jedná o prvok s najvyššou prioritou, aktualizujeme smerník k prvku s najvyššou prioritou. Časová náročnosť tejto operácie je teda .

Vyber(Q) -> X

Vyberieme prvok s najväčšou prioritou.

Vráť-minimum(Q) -> X

Spoj(Q1, Q2) -> Q

Vytvoríme novú Fibonacciho haldu Q, ktorá v sebe obsahuje prepojený zoznám stromov z oboch háld, pričom haldy Q1 a Q2 zaniknú. Časová náročnosť operácie je závislá výlučne od rýchlosti prepojenia zoznamov oboch háld, čo je pri zreťazenom zozname .

### Párovacia halda

Párovacia halda predstavuje implementáciu prioritného frontu, ktorý je reprezentovaný viacuzlovým stromom, ktorý dodržuje vlastnosť haldy. Ide o samo-vyrovnávaciu alternatívu Binomiálnej haldy.

Operácie

Vlož(Q, K, X) -> void

Vytvoríme novú párovaciu haldu Q1, ktorej koreňom bude prvok X s prioritou K. Túto haldu prepojíme s haldou Q. 4asová náročnosť tejto operácie je .

Vyber(Q) -> X

Implementácia two-pass metódou

Po odstránení koreňa nám vznikne zoznam stromov. V zozname sú jednotlivé prvky navzájom v pároch prepojené. Následne sú jednotlivé páry postupne prepájané, čím vznikne nový viacuzlový strom. Tento strom predstavuje nový stav haldy. Takáto implementácia haldy nám však dáva najhoršiu možnú časovú zložitosť .

Vráť-minimum(Q) -> X

Vráti prvok, ktorý sa nachádza v koreni stromu. Časová náročnosť tejto operácie je .

Spoj(Q1, Q2) -> Q

Korene párovacích háld Q1 a Q2 prepojíme4 a vzniknutý koreň stanovíme ako koreň novej haldy Q. Keďže časová náročnosť operácie je závislá len od rýchlosti prepojenia, ktoré je vykonané v čase, tak aj celková náročnosť prepojenia je .

## Zhrnutie teoretických časových náročnosti operácií

# Praktická implementácia prioritného frontu

Presny popis

# Návrh testov pre overenie výkonnosti implementácií prioritných frontov

V tejto časti navrhneme sady testov, pomocou ktorých bude možné overiť efektívnosť implementácie prioritných frontov. Tieto testy budú navrhnuté tak, aby simulovali praktické použitie prioritných frontov.

Test Vytvor, Spoj

Budeme vytvárať dva prioritné fronty náhodnej veľkosti, ktoré budeme nasledovne spájať. Výkon implementácie prioritného frontu sa bude skúmať v dvoch prípadoch, a to:

Veľkosť oboch frontov bude podobná.

Veľkosť oboch frontov bude značne rozdielna.

Testy budú vykonané n krát pre každú veľkosť prioritného frontu. Veľkosť prioritného frontu sa bude pohybovať od 100 po 100000 prvkov.

Sledujeme

Čas potrebný na vytvorenie jednotlivých frontov, čas potrebný na spojenie oboch frontov.

Test náhodné operácie Vlož, Vráť-minimum, Vyber

Test fázové operácie Vlož, Vráť-minimum, Vyber

# Vyhodnotenie výsledkov testov implementácií prioritných frontov

# Zoznam použitej literatúry

1. Cormen, T. H., LEISERSON, C. E., & STEIN C. (2009). *Introduction to algorithm.* London: The MIT Press
2. BROWN, M. R., (1978). *Implementation and Analysis of Binomial Queue Algorithms. SIAM Journal on Computing, 7(3), 298–319.* doi:10.1137/0207026
3. Michel Goossens, Frank Mittelbach a Alexander Samarin. *The LATEX Companion*. Reading,

Massachusetts: Addison-Wesley, 1993.

1. Jiexun Li, Rong Zheng a Hsinchun Chen. “From fingerprint to writeprint”. In: *Communications of the ACM* 49.4 (2006), s. 76–82. issn: 00010782. doi: [10.1145/1121949.1121951.](http://dx.doi.org/10.1145/1121949.1121951)

url: [http://portal.acm.org/citation.cfm?doid=1121949.1121951.](http://portal.acm.org/citation.cfm?doid=1121949.1121951)

# Zoznam príloh

**Príloha A :** Použivateľská príručka

**Príloha B :** Programatorská príručka

**Príloha C :** Obsah CD disku

**Prílohy**

1. Priorita každého prvku je menšia alebo rovná priorite otca [↑](#footnote-ref-1)
2. Každý strom je definovaný svojim indexom. Strom s indexom 0 predstavuje prvok s prioritou. Strom úrovne k, je tvorený prepojením dvoch stromov s indexom k-1. [↑](#footnote-ref-2)
3. Mergeable priority queue [↑](#footnote-ref-3)
4. Stromy s rovnakým indexom sú prepojené tak, že jeden strom sa stane ľavým synom druhého, pričom sa musí dodržať vlastnosť haldy. [↑](#footnote-ref-4)