Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Метрология стандартизация и сертификация в информационных технологиях (МСиСвИнфТ)

ОТЧЕТ

по лабораторной работе № 1

Тема работы: Расчет метрик Холстеда

Выполнили

студенты: гр. 851006 Иваненко Е. Е.

Рязанов В. В.

Проверил: Болтак С.В.

Минск 2019

Содержание

[1 Программный Код 3](#_Toc23786492)

[2 Расчет базовых метрик Холстеда](#_Toc23786493) 6

# Программный Код

package main

import (

"fmt"

)

type Dependency struct {

nodeId int

children []int

}

// converts dependencies to edges

func dependenciesToEdges(dependencies []Dependency) []Edge {

edges := []Edge{}

for \_, node := range dependencies {

for \_, child := range node.children {

edges = append(edges, Edge{start: node.nodeId, end: child})

}

}

return edges

}

type Edge struct {

start int // start and end are node ids

end int

}

type Node struct {

id int

children []int

selected bool

incoming int

}

// takes a list of edges as input with start and end node\_ids

// useful for cases like task ordering

func topologicalSort(input []Edge) ([]int, [][]int) {

// generate a mapping for node\_id and number of incoming edges

incoming := make(map[int]int)

for \_, edge := range input {

if \_, ok := incoming[edge.start]; !ok {

incoming[edge.start] = 0

}

incoming[edge.end] += 1

}

nodes := make([]\*Node, 0)

for k, v := range incoming {

chdn := []int{}

for \_, edge := range input {

if edge.start == k {

chdn = append(chdn, edge.end)

}

}

nodes = append(nodes, &Node{id: k, children: chdn, selected: false, incoming: v})

}

res := []int{}

var levels [][]int

for i := 0; i < len(queue); i++ {

node := nodes[idx]

level = append(level, node.id)

// decrement incoming values for all children

for k := 0; k < len(node.children); k++ {

childId := node.children[k]

for j := 0; j < len(nodes); j++ {

child := nodes[j]

if !child.selected {

child.incoming--

switch child.id {

case 0:

level = 0

case 1:

level = child.id \* 10

case 2:

if node == nil {

level = child.id \* 20

} else {

level = child.id \* 30

}

default:

level = child.id

}

}

}

}

}

res = append(res, level...)

levels = append(levels, level)

return res, levels

}

func BuildMaxHeap(slice []int) MaxHeap {

h := MaxHeap{slice: slice, heapSize: len(slice)}

for i := len(slice) / 2; i >= 0; i-- {

h.MaxHeapify(i)

}

return h

}

func (h MaxHeap) MaxHeapify(i int) {

l, r := 2\*i+1, 2\*i+2

max := i

if l < h.size() && h.slice[l] > h.slice[max] {

max = l

}

if r < h.size() && h.slice[r] > h.slice[max] {

max = r

}

//log.Printf("MaxHeapify(%v): l,r=%v,%v; max=%v\t%v\n", i, l, r, max, h.slice)

if max != i {

h.slice[i], h.slice[max] = h.slice[max], h.slice[i]

h.MaxHeapify(max)

}

}

func (h MaxHeap) size() int { return h.heapSize } // ???

func heapSort(slice []int) []int {

h := BuildMaxHeap(slice)

//log.Println(slice)

for i := len(h.slice) - 1; i >= 1; i-- {

h.slice[0], h.slice[i] = h.slice[i], h.slice[0]

h.heapSize--

h.MaxHeapify(0)

if i == len(h.slice)-1 || i == len(h.slice)-3 || i == len(h.slice)-5 {

element := (i - len(h.slice)) \* -1

fmt.Println("Heap after removing ", element, " elements")

fmt.Println(h.slice)

}

}

return h.slice

}

func main() {

var edges []Edge

edges = append(edges, Edge{1, 2})

edges = append(edges, Edge{2, 3})

edges = append(edges, Edge{4, 3})

edges = append(edges, Edge{4, 5})

edges = append(edges, Edge{3, 6})

edges = append(edges, Edge{5, 6})

order, levels := topologicalSort(edges)

fmt.Println("result: ", order)

fmt.Println("levels: ", levels)

// or you can specify dependencies

var deps []Dependency

deps = append(deps, Dependency{nodeId: 1, children: []int{2}})

deps = append(deps, Dependency{nodeId: 2, children: []int{3}})

deps = append(deps, Dependency{nodeId: 4, children: []int{3, 5}})

deps = append(deps, Dependency{nodeId: 5, children: []int{6}})

deps = append(deps, Dependency{nodeId: 3, children: []int{6}})

edges = dependenciesToEdges(deps)

order, levels = topologicalSort(edges)

fmt.Println("result: ", order)

fmt.Println("levels: ", levels)

}

**2. РАСЧЕТ БАЗОВЫХ МЕТРИК ХОЛСТЕДА**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j | Оператор | F1 | i | Операнд | F2 |
| 1. | for | 10 | 1. | levels | 7 |
| 2. | switch | 1 | 2. | fmt | 6 |
| 3. | if | 8 | 3. | order | 4 |
| 4. | return | 5 | 4. | edges | 19 |
| 5. | . | 50 | 5. | deps | 11 |
| 6. | () | 1 | 6. | 6 | 4 |
| 7. | {} | 49 | 7. | Int | 9 |
| 8. | = | 29 | 8. | children | 9 |
| 9. | ; | 11 | 9. | 3 | 7 |
| 10. | & | 1 | 10. | nodeId | 6 |
| 11. | ! | 2 | 11. | Dependency | 5 |
| 12. | / | 1 | 12. | 5 | 5 |
| 13. | \* | 7 | 13. | 4 | 3 |
| 14. | - | 6 | 14. | 2 | 9 |
| 15. | + | 2 | 15. | 1 | 9 |
| 16. | > | 2 | 16. | Edge | 8 |
| 17. | < | 5 | 17. | slice | 24 |
| 18. | := | 25 | 18. | h | 29 |
| 19. | -- | 4 | 19. | element | 2 |
| 20. | ++ | 3 | 20. | i | 22 |
| 21. | += | 1 | 21. | 0 | 11 |
| 22. | >= | 2 | 22. | heapSize | 3 |
| 23. | != | 1 | 23. | max | 9 |
| 24. | == | 5 | 24. | r | 4 |
| 25. | && | 2 | 25. | l | 4 |
| 26. | || | 2 | 26. | MaxHeap | 1 |
| 27. | Println() | 6 | 27. | res | 4 |
| 28. | topologicalSort() | 2 | 28. | level | 9 |
| 29. | dependenciesToEdges() | 1 | 29. | .. | 1 |
| 30. | append() | 17 | 30. | id | 7 |
| 31. | len() | 10 | 31. | child | 10 |
| 32. | MaxHeapify() | 3 | 32. | default | 1 |
| 33. | BuildMaxHeap() | 1 | 33. | 30 | 1 |
| 34. | size() | 2 | 34. | 20 | 1 |
| 35. | make() | 2 | 35. | nil | 1 |
|  |  |  | 36. | node | 8 |
|  |  |  | 37. | 10 | 1 |
|  |  |  | 38. | incoming | 7 |
|  |  |  | 39. | selected | 2 |
|  |  |  | 40. | j | 4 |
|  |  |  | 41. | nodes | 6 |
|  |  |  | 42. | k | 7 |
|  |  |  | 43. | childId | 1 |
|  |  |  | 44. | idx | 1 |
|  |  |  | 45. | queue | 1 |
|  |  |  | 46. | v | 2 |
|  |  |  | 47. | false | 1 |
|  |  |  | 48. | chdn | 4 |
|  |  |  | 49. | Node | 2 |
|  |  |  | 50. | end | 3 |
|  |  |  | 51. | edge | 7 |
|  |  |  | 52. | start | 4 |
|  |  |  | 53. | input | 2 |
|  |  |  | 54. | \_ | 5 |
|  |  |  | 55. | ok | 2 |
|  |  |  | 56. | map | 1 |
|  |  |  | 57. | dependencies | 1 |
|  |  |  | 58. | “levels: ” | 2 |
|  |  |  | 59. | “result: ” | 2 |
|  |  |  | 60. | “ elements” | 1 |
|  |  |  | 61. | “Heap after removing” | 1 |
| µ1 = 35 |  | N1 = 279 | µ2 = 61 |  | N2 = 343 |

Словарь программы µ = 35 + 61 = 96.

Длина программы N = 279 + 343 = 622.

Объем программы V = 622log2 96 = 4069