Porteføljeopgave 1

Mathias Balling Christiansen

November 20, 2024

Contents

1 Opgave

Koden:

```cpp
std::string highest_word_count(std::string input) {
  if (input.empty()) {
    return "";
  }
  std::unordered_map<std::string, int> word_count;
  while (!input.empty()) {
    size_t dilimiter_index = input.find(' ');
    std::string word = "";
    if (dilimiter_index != std::string::npos) {
      word = input.substr(0, dilimiter_index);
      // Make word lowercase
      for (auto &c : word) {
        c = std::tolower(c);
      }
      input = input.substr(dilimiter_index + 1);
    } else {
      // Last word here
      word = input;
      // Make word lowercase
      for (auto &c : word) {
        c = std::tolower(c);
      }
      input.erase();
    }
    if (word.find(',') != std::string::npos ||
        word.find('.') != std::string::npos) {
      // Dilimiter will always be in the end.
      word.erase(word.end() - 1);
    }
    if (word_count.contains(word)) {
      word_count[word]++;
    } else {
      word_count[word] = 1;
    }
  }

  std::pair<std::string, int> highest_word_count = {"", 0};
  for (const auto &word_pair : word_count) {
    if (word_pair.second > highest_word_count.second) {
      highest_word_count.first = word_pair.first;
      highest_word_count.second = word_pair.second;
    }
  }

  return highest_word_count.first;
}
```

Koden kører igennem input string en gang hvilket har tidskompleksitet $O(n)$.

Jeg bruger derefter **find** til at finde næste mellemrum hvilke også har tidskompleksitet $O(n)$.

Her ligger jeg de ord jeg finder ind i et hashmap og til allersidst finder jeg ordet med højeste værdi.

2 Opgave

```
BinaryNode *BinarySearchTree::getOnlyChild(BinaryNode *t) const {
  if (t->left != nullptr && t->right == nullptr)
    return t->left;
  else if (t->right != nullptr && t->left == nullptr) {
    return t->right;
  } else {
    return nullptr;
  }
}

int BinarySearchTree::countBranches(BinaryNode *t) const {
  // Early return
  if (t == nullptr)
    return 0;
  // Check for only child
  BinaryNode *onlyChild = getOnlyChild(t);
  if (onlyChild == nullptr) {
    // Not only child (0 or 2 children)
    // Skip nodes with siblings
    if (t->left != nullptr && t->right != nullptr) {
      return 0 + countBranches(t->left->left) + countBranches(t->left->right) +
             countBranches(t->right->left) + countBranches(t->right->right);
    }
    // Skip nodes with no children
    return 0;
  }
  // Check for child with only child
  BinaryNode *onlyChildOnlyChild = getOnlyChild(onlyChild);
  if (onlyChildOnlyChild == nullptr) {
    // Not only child (0 or 2 children)
    // Skip nodes with siblings
    if (onlyChild->left != nullptr && onlyChild->right != nullptr) {
      return 0 + countBranches(onlyChild->left->left) +
             countBranches(onlyChild->left->right) +
             countBranches(onlyChild->right->left) +
             countBranches(onlyChild->right->right);
    }
    // Skip nodes with no children
    return 0;
  }
  // Check for leaf
  if (onlyChildOnlyChild->left == nullptr &&
      onlyChildOnlyChild->right == nullptr) {
    return 1;
  }
  // Not leaf check from onlychild and down
  return 0 + countBranches(onlyChild->left) + countBranches(onlyChild->right);
}
```

Supplerende opgave 1:

Hvad er træet:
Det er et binært søgetræ. Det er ikke komplet eller balanceret da alle noder ikke er fyldt fra venstre

til højre og forskellen i dybden er mere end 1.

Sædvanlige oplysninger:
- Højde: 6 ($7 \to 28 \to 55 \to 51 \to 48 \to 40 \to 35$)
- Internal path length: $0 + 2*1 + 2*2 + 3*3 + 4*4 + 5*2 + 6 = 47$

Optimale højde + matematisk udtryk:
Den optimale højde er $h = \log(n+1)$
Da vi har $n = 15$ elementer så er optimale højde $h = \log(16) = 4$

Supplerende opgave 2:

Fra binære søgetræ til prioritetskø (trin + tidskompleksitet):

3 Opgave

List rækkefølgen I hvilken noderne vil blive besøgt I en en in order og i en level order traversering.

In order: 1 2 3 9 11 13 17 25 57 90

Level order: 11 2 13 1 9 57 3 25 90 17

Hvad er træets internal path length?

$$0 + 2*1 + 3*2 + 3*3 + 4 = 21$$

balancen i træet er ved node 13, hvor højden af det højre subtræ er 3, og det venstre subtræ er 0. Hvordan kan man omarrangere noderne i det højre subtræ, så hele træet bliver et AVL-træ?

Man kan rykke indsætte 17 på 13's plads og derefter rykke 13 til at være i venstre subtræ af 17.

Kunne træet have været et AVL-træ før den seneste operation (insert eller delete, men ikke rotation)? Eksempler på seneste operation kunne være indsættelse af node 3 eller sletning af node 12 (venstre barn af node 13). Begrund dit svar.

Nej, Der er ikke noget man kunne have indsat eller fjernet for at det var et AVL træ. Lige meget hvad ville mere end 1.

4 Opgave

List rækkefølgen i hvilken noderne besøges i en post order og i en pre order traversering.

Post order: 1 8 5 15 12 10 22 20 28 30 38 45 50 48 40 36 25

Pre order: 25 20 10 5 1 8 12 15 22 36 30 28 40 38 48 45 50

Hvad er træets internal path length?

$$0 + 2*1 + 4*2 + 5*3 + 5*4 = 45$$

Er det et AVL-træ? Hvorfor eller hvorfor ikke?

Nej, der er ubalance i 20, da det venstre sub-træ går 3 ned og det højre kun går 1 ned.

5 Opgave

Jeg bruger Kruskal's

Weight	Node-pair
1	(0,1)
1	(0,4)
1	(3,6)
1	(3,7)
2	(0,5)
2	(9,10)
3	(2,5)
3	(5,8)
3	(7,11)
4	(8,9)
5	(10,11)

Totale vægt: 26

6 Opgave

v	Known	d_v	p_v
A	true	0	0
B	true	5	A
C	true	3	A
D	true	9	E
E	true	7	G
F	true	8	E
G	true	6	B