

# Rozdział 1

## Zaimplementowane algorytmy

W poniższych podrozdziałach zaprezentowano zasadę działania zaimplementowanych w oprogramowaniu algorytmów.

### 1.1 Działanie przycisków

W aplikacji każdy przycisk jest obiektem klasy `ButtonOperator`, która dziedziczy po klasie `QPushButton`. Dzięki czemu fizyczne obiekty przycisków mogą korzystać zarówno z metod klasy nadrzędnej np. `isChecked()`, jak i klasy dziedziczącej. Właśnie dzięki nadpisaniu metod domyślnych klasy `QPushButton` - `hoverEnter()` i `hoverLeave()` stworzono możliwość detekcji, nad którym przyciskiem aktualnie znajduje się punkt fiksacji wzroku. Podczas wywołania obu metod zmieniana jest wartość logiczna zmiennej `isHovered` aktualnie obserwowanego przycisku. Prócz informacji o tym, czy dany przycisk jest aktualnie używany przechowuje się również dane o tym, czy przycisk zalicza się do tkzw. "specjalnych", czy też nie, jak i listę dostępnych dla niego tekstów do wyświetlania – wyglądu przycisku dla 6 stanów klawiatury (małe litery, duże litery, znaki specjalne karta pierwsza, znaki specjalne karta druga, polskie litery, menu kontekstowe). Wszystkie te dane wprowadza się podczas uruchamiania klawiatury i wtedy także zapisuje się przyciski kolejno do specjalnej listy. Kolejność jest znacząca w przypadku przycisków "specjalnych", gdyż ich obsługa zależna jest od wartości ich ID zapisanego w pliku ze stałymi. Określenie przycisku działającego odbywa się poprzez sprawdzenie ID przycisku, którego zmienna `isHovered` jest prawdziwa w klasie `TimeManager`. Do zarządzania stanem klawiatury, ostatnio wybranym przyciskiem specjalnym oraz ostatnio obserwowanym przyciskiem powstała klasa `HoverManager`. Zbiera ona na bieżąco informacje o ID przycisku ostatnio najechanego, o tym przez jaki czas dany przycisk jest już pod punktem fiksacji, o aktualnym stanie klawiatury, o ostatnio wybranym specjalnym przycisku (np. `CapsLock`) oraz przez jaki czas działanie tego przycisku się utrzymuje. Większość z tych danych odświeżana jest co 200ms (stała zdefiniowana w pliku ze stałymi) podczas każdego wykonywania się metody

TimerStep(). Jej działanie przedstawiono za pomocą pseudokodu 1.

---

**Algorytm 1** Działanie funkcji TimerStep()
 

---

```

pobierz ID aktualnie fikowanego przycisku
if pobrane ID różne jest od -1 then
  if zwykle przyciski są w trybie wstrzymania then
    if jeśli aktualnie fikowany przycisk jest specjalny then
      hoverState (obiekt klasy HoverManager) ustaw aktualnie aktywny
      przycisk na pobrane ID
      Wykonaj krok timera (funkcja executeTimerStep())
      Pokaż czas przez jaki przycisk jest fikowany na pasku postępu dla
      informacji użytkownika.
    end if
  else
    Wykonaj powyższe funkcje dla wszystkich przycisków - niezależnie od
    tego, czy są specjalne, czy nie.
  end if
end if
Wywołaj funkcję verifyTimerTickCount().
  
```

---

Zadaniem wyżej wspomnianej funkcji executeTimerStep() jest sprawdzenie, czy dany przycisk był fikowany przez odpowiednią ilość czasu (zmienna, której wartość zależy od ilości pomyłek popełnionych przez użytkownika i dynamicznie zmieniana podczas korzystania z klawiatury – analizowane to jest w funkcji verifyTimerTickCount()). Jeżeli warunek ten został spełniony to dalszy przebieg działań zależy od tego, czy przycisk był specjalny, czy też nie oraz czy klawiatura nie znajduje się w trybie wstrzymania.

### 1.1.1 Działanie przycisków specjalnych

W tabeli 1.1 wymieniono wszystkie specjalne przyciski z widoku klawiatury oraz opisano po krótku algorytm ich działania. W tabeli 1.2 przedstawiono przyciski widoku menu oraz ich zachowania.

Jako wciśnięcie rozumie się tu czas fikacji nad przyciskiem przekraczający progową wartość. Stany klawiatury to odpowiednio 0-małe litery, 1-wielkie litery, 2-znaki specjalne strona pierwsza, 3-znaki specjalne strona druga, 4-polskie znaki, 5-menu kontekstowe dla polskich znaków.

### 1.1.2 Działanie przycisków normalnych

Za wpisywanie znaków z klawiatury do edytora odpowiedzialna jest funkcja update() klasy Dictionary. Jest ona głównym zarządcą jeśli chodzi o wprowadzanie znaków w odpowiedniej pozycji. W pierwszym rzędzie sprawdzane jest, czy stan klawiatury różny jest od piątego (menu kontekstowe). Gdy spełniony jest dany warunek to porównywana jest aktualnie wprowadzana wartość z uprzednio wprowadzoną, o ile nie jest to pierwsze wprowadzenie znaku do edytora. W przypadku dwóch identycznych liter czyszczona jest zawartość piątego stanu klawiatury i przechodzi się do wywołania funkcji switchBetweenKeyboards(), która

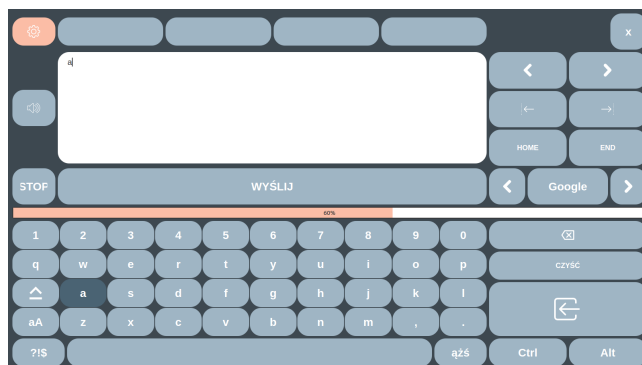
Nazwa przycisku	Działanie
Backspace	Wywołuję metodą <code>backspace()</code> klasy <code>Dictionary</code> , której działanie opisano w podrozdziale 1.2.1.
CapsLock	W przypadku pierwszego wciśnięcia zmienia stan klawiatury z 0 na 1 i utrzymuje ją dopóki nie zostanie wybrany po raz wtóry lub wykonana zostanie czynność innego przycisku zmieniającego stan klawiatury.
Czyść	Korzysta z metody <code>resetAll()</code> klasy <code>Dictionary</code> i powoduje czyszczenie edytora tekstowego oraz z nim związanych zmiennych jak <code>currentPosition</code> , <code>currentWord</code> , <code>currentWordStart</code> , <code>wholeText</code> (więcej na ten temat w rozdziale 1.2).
Menu	Otwiera okno klasy <code>Personalize</code> .
Przesuń się o jedno słowo w prawo lub w lewo	Wywołuje funkcję <code>jumpWord()</code> klasy <code>Dictionary</code> , która powoduje przesunięcie się kursora na koniec poprzedniego słowa, lub na początek kolejnego. Każdorazowo zmieniana jest wartość zmiennej <code>currentWord</code> i <code>wholeText</code> . 1.2
Przesuń się w kierunku początku tekstu (home) lub na jego koniec (end)	Wybranie przycisku powoduje przesunięcie się kursora w jednym z dwóch kierunków oraz zmianę wartości zmiennej <code>wholeText</code> i <code>currentWord</code> (1.2).
Przyciski podpowiedzi	Wyświetlane na nich są podpowiedzi, których tworzenie jest opisane w rozdziale 1.3. Ich użycie powoduje zastąpienie aktualnie wpisywanej frazy auto uzupełnionym słowem pobranym ze słownika.
Shift	W przypadku wciśnięcia zmieniony zostaje stan klawiatury z 0 na 1, a po użyciu przycisku ze znakiem stan klawiatury wraca do stanu 0.
Stop i start	Wprowadza klawiaturę w stan wstrzymania lub w stan pracy. W zależności od wartości zmiennej <code>isStop</code> możliwe, lub nie, jest korzystanie z przycisków ze znakami.
Strzałka w lewo i prawo	Korzystając z metody <code>moveCursor</code> klasy <code>Dictionary</code> przemieszcza kursor o jeden znak w danym kierunku. Może zmienić wartość <code>currentWord</code> (1.2) korzystając z metod <code>getCurrentWord()</code> oraz <code>getCurrentWordStart()</code> (2) opisanych w późniejszych rozdziałach.
Wyjdź	Powoduje opuszczenie aplikacji.
Wyszukaj na podanej platformie	Przyciśnięcie przycisku powoduje sprawdzenie połączenia internetowego, a następnie wysłanie wpisanej treści pola tekstowego do funkcji <code>search()</code> klasy <code>GoogleSearcher</code> . TUTAJ ODSYŁNIK DO KLASY
Wyślij	Wysyła dotychczas wpisany tekst na broadcast na port 45454 za pomocą metody <code>writeDatagram()</code> klasy <code>QUdpSocket</code> .
Zmiana trybu wyszukiwania	Strzałki powodują na zmianę trybu wyszukiwania. Do wyboru "Google", "YouTube", "Filmweb". Zmieniają wartość zmiennej <code>sendingState</code> niezbędnej do poprawnego działania funkcji <code>search()</code> klasy <code>GoogleSearcher</code> . TUTAJ ODSYŁNIK DO KLASY
Znaki specjalne	Zmiana stan klawiatury na odpowiednio 2 i 3 przy kolejnych kliknięciach.

Tab. 1.1 – Lista specjalnych przycisków w raz z ich działaniem.

Nazwa przycisku	. Działanie
Strzałki zmieniające aktualną wartość progową czasu fiksacji	Strzałki powodują zmniejszenie lub zwiększenie wartości zmiennej tickCounter przechowującej ilość 200ms interwałów, po upływie których (w przypadku braku zmiany fikowanego punktu) przycisk można uznać za wciśnięty. Początkowo czas ten wynosi 3s, jednak w wyniku dynamicznej korekty następującej, gdy użytkownik wykona mniej, lub więcej niż 5 błędów w ciągu minuty (średnio 25% błędnie wpisanych znaków), następuje zmniejszenie lub zwiększenie liczby tych interwałów.
Strzałki zmieniające wielkość czcionki w edytorze tekstu	Strzałki powodują zmianę właściwości edytora tekstu.
Strzałki zmieniające wygląd aplikacji	Użytkownik może wybrać między jednym z 5 wersji kolorystycznych poprzez dynamiczną zmianę wyglądu na podstawie danych z listy i aktualnego indeksu.
Wyjście	Powoduje zamknięcie okna menu.

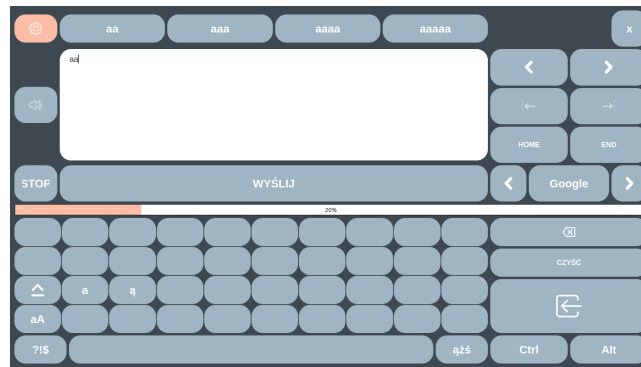
Tab. 1.2 – Lista specjalnych przycisków z menu w raz z ich działaniem.

zwraca informację o tym, stan klawiatury się zmienił. Tam sprawdzane jest, czy wprowadzony znak jest jedną z liter posiadających polskie odpowiedniki tj. *a, c, e, l, n, o, s, z*. Gdy znajdzie się wśród wymienionej listy, to dla odpowiadającego przycisku ustawiany jest test dla klawiatury stanu piątego i następuje jej wyświetlenie. Działanie takiego zachowania widać na rysunkach 1.1, 1.2.



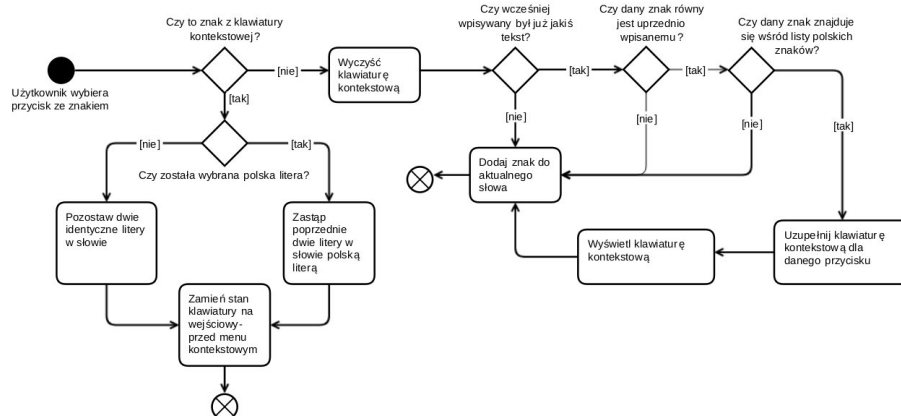
Rys. 1.1 – Widok aplikacji z klawiaturą w stanie zerowym po wpisaniu litery 'a'.

W innym wypadku aktualna litera dopisywana jest do aktualnie tworzonego słowa (*currentWord*), kursor przesuwany jest o jedną pozycję w prawo. Gdy dodana jest spacja aktualnie przetwarzane słowo jest uznawane za skończone i zapamiętywany jest nowy początek kolejnego słowa. Odświeżony zostaje również widok podpowiedzi, których powstawanie omówiono w późniejszym rozdziale 1.3. W sytuacji, gdy wybrana zostaje litera z menu kontekstowego (klawiatury w stanie piątym), to albo wpisany tekst zostaje podmieniony na polską literę, lub nie ulega zmianie. Przykładowo po wpisaniu "aa" użytkownik po raz kolejny



Rys. 1.2 – Widok aplikacji z klawiaturą w stanie piątym po wpisaniu drugiej litery 'a'.

wybiera literę "a" - tzn. planował wpisanie frazy "aa". Jeśli jednak decyduje się na literę "ą", to w miejsce widniejącego napisu "aa" pojawia się "ą". Poglądowy diagram przepływu przedstawiono na rysunku 1.3.

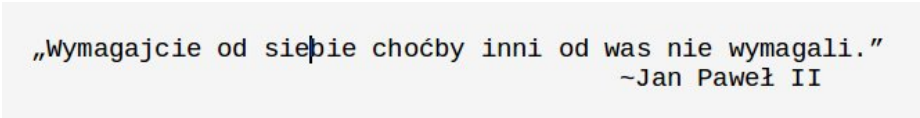


Rys. 1.3 – Widok aplikacji z klawiaturą w stanie piątym po wpisaniu drugiej litery 'a'.

## 1.2 Praca z tekstem

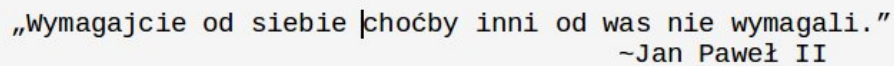
Wpisywanie tekstu w aplikacji odbywa się poprzez specjalny algorytm kontrolujący zawartość aktualnie pisanego słowa oraz całego tekstu. Podczas każdej chwili działania programu w pamięci przechowywany jest `currentWord`, czyli aktualne słowo tj. ciąg znaków, liter lub cyfr, które zaczynają się na początku wpisywanego tekstu lub po spacji, a kończą się w pozycji kursora. Dodanie spacji po ciągu znaków kończy słowo i usuwa je ze zmiennej `currentWord`, a

dodaje do zmiennej zwanej `wholeText`, która przechowuje dotychczas wpisany tekst. Przykładowo jeśli mamy tekst jak na rysunku 1.4 to w zmiennej `wholeText` przechowujemy `„Wymagajcie od bie choćby inni od was nie wymagali.” Jan Paweł II`, a w `currentWord` `”sie”`. W ten sposób odpowiedzi generowane będą jedynie dla części `”sie”`, a tekst wpisywany będzie w pozycji kursora, która również monitorowana jest przez zmienną `currentPosition`. Scalenie `wholeText` i `currentWord` następuje, gdy zmieniamy pozycję kursora strzałkami, lub jeśli do tekstu dodana zostanie spacja, a za kursorem nie znajduje się żaden znak przykładowo w takiej sytuacji znajdziemy się na rysunku 1.5.



```
„Wymagajcie od siebie choćby inni od was nie wymagali.”
~Jan Paweł II
```

Rys. 1.4 – Przykład zapisywanie tekstu do zmiennych w zależności od pozycji kursora.



```
„Wymagajcie od siebie choćby inni od was nie wymagali.”
~Jan Paweł II
```

Rys. 1.5 – Przykład zapisywanie tekstu do zmiennych w zależności od pozycji kursora.

Zmienna `currentWord` została zespolona z `wholeText` poprzez wklejenie jej na pozycji zapisanej jako `currentWordStart`. W celu dynamicznego ustalania pozycji `currentWordStart` oraz zawartości `currentWord` powstały funkcje: `getCurrentWordStart()` oraz `getCurrentWord()`. Działanie pierwszej zademonstrowano za pomocą pseudokodu 2. Działanie drugiej sprowadza się do wycięcia

---

#### Algorytm 2 Działanie funkcji `getCurrentWordStart()`

---

```
if currentWord nie jest pusty && currentPosition nie jest na początku tekstu
then
  for każda pozycja aż do początku tekstu do
    pobierz literę z wholeText w danej pozycji
    if pobrana wartość nie jest liczbą ani literą then
      currentWordStart = currentPosition + 1
    end if
  end for
end if
```

---

fragmentu tekstu między `currentWordStart`, zaimplementowanym w wyniku działania poprzedniej funkcji, a `currentPosition`.

### 1.2.1 Usuwanie znaków

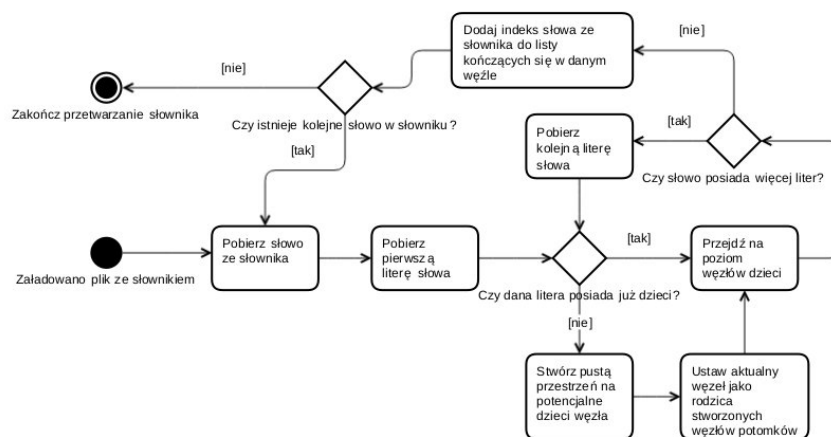
Poprzez ułatwiające kontrolę nad tekstem `currentWord` i `wholeText` proces usuwania znaków okazał się trudniejszy niż można było się tego spodziewać. Pier-

wszym napotkanym problemem była sytuacja, w której `currentWord` jest puste. W takim wypadku należy usunąć znak znajdujący się w `currentPosition` ze zmiennej `wholeText`, a następnie za pomocą wcześniej opisanych funkcji `getCurrentWordStart()` oraz `getCurrentWord()` otrzymać informacje o nowej wartości `currentWord`. Kolejnym krokiem jest wycięcie wartości zmiennej `currentWord` z `wholeText`, przesunięcie kursora dla informacji użytkownika w odpowiednią pozycję oraz odświeżenie wyglądu odpowiedzi. Gdy `currentWord` obdarzony jest wartością, sytuacja upraszcza się do usunięcia ostatniego znaku z `currentWord`. W celu reprezentacji tekstu dla użytkownika należy scalić `wholeText` z `currentWord`, ale by nie utracić wartości `wholeText` stworzono tymczasową zmienną `currentText`, który zawiera całą treść wpisanego tekstu.

### 1.3 Trie tree

Jak wyżej ?? wspomniano, w pracy wykorzystano drzewo typu Trie w celu pracy z rozległym słownikiem. Słowa z pliku w formacie TXT wczytywane są do programu podczas uruchamiania klawiatury- każda z linii dostarczanego pliku powinna stanowić pojedynczy wyraz. Słowa te, za pomocą sztucznie wygenerowanych list kodujących oraz dekodujących polski alfabet, wprowadzane są do drzewa typu Trie. Drzewo takie powstaje poprzez stworzenie pustego węzła typu `node` - struktury zadeklarowanej w pliku nagłówkowym klasy obsługującej współpracę ze słownikiem – `Dictionary`. Struktura `node` przechowuje informacje o rodzicu bieżącego węzła, o jego potomkach – czyli węzłach następujących oraz wektor zawierający informację o przynależności danego węzła literowego do słowa. Po zainicjowaniu węzła zerowego, po kolei, analizowane są słowa ze słownika w funkcji `insertWord()`. Każde jest rozpatrywane jako tablica liter (`char`) i iteracyjnie następuje najpierw kodowanie litery na odpowiadający jej indeks (za pomocą mapy alfabet), potem, sprawdzane jest, czy drzewie nie istnieje już węzeł odpowiadający danej wartości. Gdy nie znaleziono gałęzi drzewa pasujących do poszukiwanej wartości tworzy się za pomocą funkcji `calloc` przestrzeń w pamięci na przyszłe dzieci tego węzła, a jako ich rodzica podaje się aktualnie przeglądany węzeł z literą. Kolejno, niezależnie od wyniku uprzednio rozpatrywanego warunku, o ile słowo wciąż posiada litery do przeglądu, przenosi się o poziom niżej w drzewie ( na poziom dziecka uprzednio rozpatrywanej litery) i proces zachodzi od początku. Gdy przetworzono wszystkie litery słowa do listy wystąpień ostatniego odwiedzonego węzła dopisany zostaje indeks słowa w słowniku – tym samym oznaczając jego koniec. W sposób graficzny przedstawiono działanie danej funkcji na rysunku 1.6

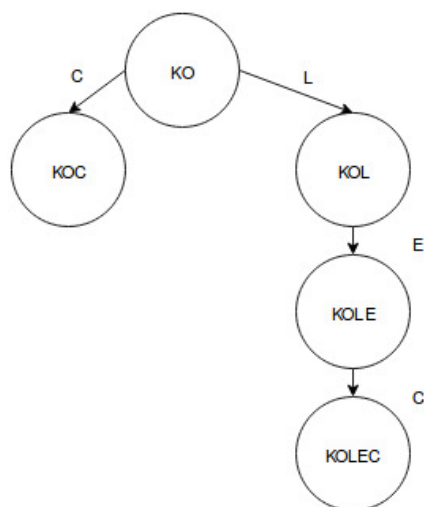
Po uzupełnieniu słownika można przejść do jego wykorzystywania - auto uzupełniania wpisywanego tekstu. Każde wpisanie litery powoduje wywołanie metody `updateHints()`, która jest odpowiedzialna za tworzenie oraz wyświetlanie odpowiedzi, zgodnie z wprowadzonym do tej pory słowem. Jako poszukiwaną frazę traktujemy ciąg znaków, które użytkownik wpisał do pozycji kursora od ostatniej spacji, bądź początku tekstu. W wypadku, gdy ten ciąg znaków jest dłuższy niż dwa, wykorzystywana jest funkcja komunikująca się ze stworzonym słownikiem – `searchWord()`. Przekazywane jest drzewo Trie słownika oraz aktualnie poszukiwana fraza (bez formatowania). Wpisywana fraza, również traktowana jest jako zbiór liter, które przeglądane są jedna po drugiej.



Rys. 1.6 – Diagram przepływu dla funkcji `insertWord()` wprowadzającej słowa do drzewa typu Trie.

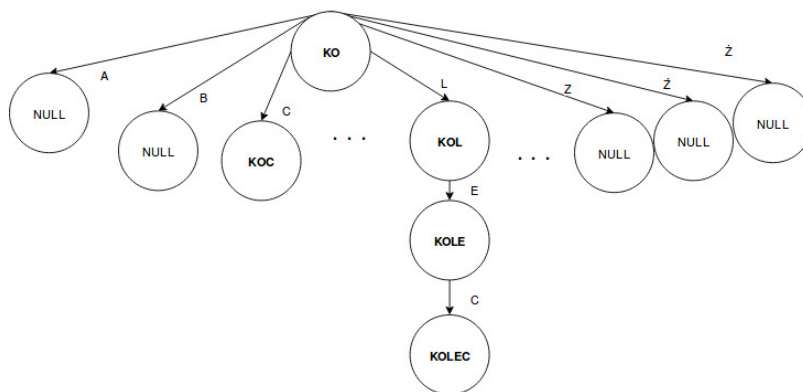
Dla każdej następuje zmiana dzięki mapie kodującej alfabet na odpowiadający indeks, który umożliwia przeszukiwanie słownika. Sprawdzane jest, czy istnieje potomek drzewa Trie o danym indeksie – jeśli tak, to następuje zmiana węzła na węzeł dziecka, tak, że przy przeszukiwaniu drzewa brane pod uwagę będą jedynie węzły z rodzicem będącym pierwszą literą poszukiwanej frazy. Gdy przeszuka się wszystkie litery, bądź w trakcie tego procesu zabraknie węzłów potomków dla danej kombinacji liter to zwracany jest odpowiednio ostatni węzeł wspólny dla danej frazy lub też węzeł pusty. Przykładowo - założmy, że mamy drzewo takie jak na rysunku ???. Jeśli wyszukamy frazę "ko" to funkcja wróci nam węzeł i jego dzieci - czyli możliwe auto uzupełnienia wyglądają tak jak na rysunku 1.7. Złożenie końcówek wyrazów zachodzi w funkcji `getSimilarEndings()`, której przekazywany jest znaleziony węzeł, pusty wektor, do którego mają zostać zapisane wynikowe wyrazy oraz węzeł znaków niezbędny do dekodowania. W pierwszym kroku sprawdzane jest, czy w danym węźle kończą się jakieś słowa, jeśli tak (wektor wystąpień węzła jest różny od zera), to każdą literę zapisaną w wyżej wymienionym wektorze znaków zapisuje się do jednego słowa (tworząc końcówkę do auto uzupełniania). Gotowy ciąg znaków zapisywany jest do wektora z końcówkami. W wypadku pierwszego wykonania się funkcji mamy do czynienia z pustym wektorem znaków- toteż nie zostanie stworzona żadna końcówka. Nawet jeśli "ko" było pełną formą słowa nie powinna ona się wyświetlać w proponowanych opcjach użytkownika. Aby uzupełnić wektor znaków należy przejrzeć przesłany węzeł (ten z rysunku 1.7) – w tym celu sprawdza się, czy dzieci węzła odpowiadającej każdej literze alfabetu nie są puste, gdyż programowe drzewo ma, prócz wcześniej przedstawionych gałęzi, jeszcze 33 (zakładając, że zaimplementowany alfabet posiada 35 liter) nieobdarzone wartością gałęzie przedstawione na rysunku 1.8. Gdy znaleziono element o niezerowej wartości pobierana jest za pomocą mapy symetrycznej (`reverseAlfabet`) wartość literowa węzła i wprowadzana jest do wektora znaków. Węzeł z "ko" zmieniany jest węzeł pierwszego dziecka – w tym wypadku "koc". Następnie przez rekurencję ponownie rozpoczyna się sprawdzanie, czy





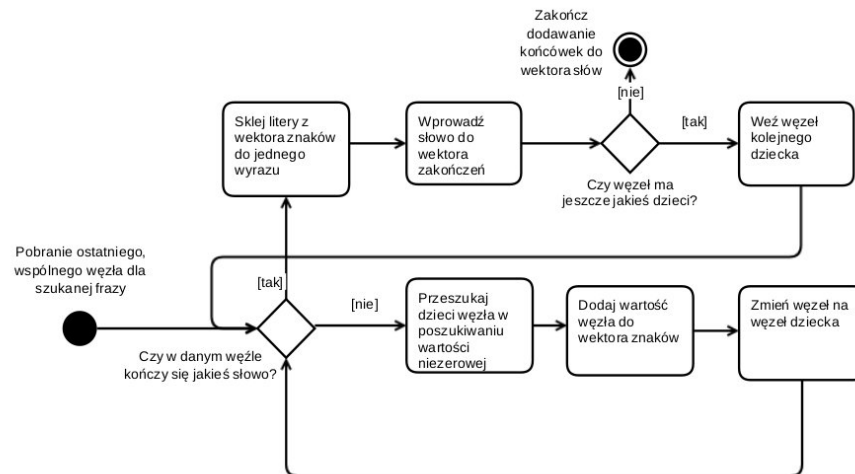
Rys. 1.7 – Przykładowy węzeł do autouzupełniania słów po wpisaniu frazy "ko".

dane słowo kończy się w tym węźle, jeśli tak to proces zachodzi według powyżej opisanych kroków, jeśli nie to znów poszukiwany jest węzeł potomny z kolejną literą końcówki słowa do auto uzupełniania. Algorytm przedstawiono w sposób graficzny na rysunku 1.9 Ostatnim krokiem w stworzeniu podpowiedzi jest



Rys. 1.8 – Reprezentacja przykładowego węzła widzianego w programie.

skrócenie listy końcówek do liczby przycisków przeznaczonych na podpowiedzi oraz zespolenie ich z dotychczas wpisanym słowem. Taki ciąg znaków można przedstawić użytkownikowi jako napis na przycisku, który po wybraniu wpisuje reprezentowany tekst do pola tekstowego zamieniając dotychczasową frazę na wybraną oraz dodając znak spacji na końcu nowowybranego słowa.



Rys. 1.9 – Diagram przepływu dla funkcji getSmlarEndings().

## 1.4 Korzystanie z wyszukiwarki internetowej

W celu pracy z przeglądarką niezbędne jest podłączenie drugiego ekranu, na którym otwierać może się okno przeglądarki. W innym wypadku okno klawiatury ulegnie minimalizacji i nie ma możliwości do jej powrotu.

W przypadku skorzystania z jednego z trybów wy wysyłania ("Google", "YouTube", "Filmweb") podczas przyciśnięcia przycisku wywołwana jest metoda, GoogleSearcher, search().

W zależności od przesłanej wartości sendingState, informującego o tym, gdzie wysłane ma być zapytanie, do wyszukiwanego tekstu (pobranego z pola tekstowego) dołączana jest informacja, w którym serwisie szukać wyników. Taka informacja załączana jest jako argument uzupełniający URL powstały poprzez stworzenie spersonalizowanej wyszukiwarki poprzez skorzystanie z specjalnego Google API. Następnie obiekt klasy QNetworkAccessManager wywołuje metodę RESTową GET na obiekcie klasy QNetworkRequest, który za argument konstruktora przyjmuje powstałe URL z parametrem.

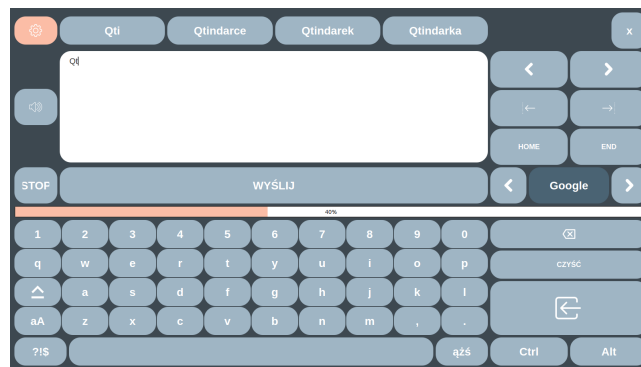
### 1.4.1 Google Api

Jak już wcześniej wspomniano ?? do projektu wykorzystano specjalne API Google - Google Custom Search Engine. Dzięki temu powstał specjalny link URL umożliwiający przyjmowanie różnych parametrów jako zapytanie do wyszukiwarki. Co więcej specjalne API na rządanie GET zwraca iformację w postaci QNetworkReply, gdzie funkcja handleNetworkData() umożliwia jego przetworzenie w ustrukturyzowaną postać JSON. Obiekt JSON zawiera 10 pierwszych wyników wyszukiwania w specjalnej przeglądarce. Każdy z obiektów typu JSON posiada informacje o wyniku wyszukiwania. W skład takiego obiektu wchodzi ?:

- "kind": "customsearch#result"

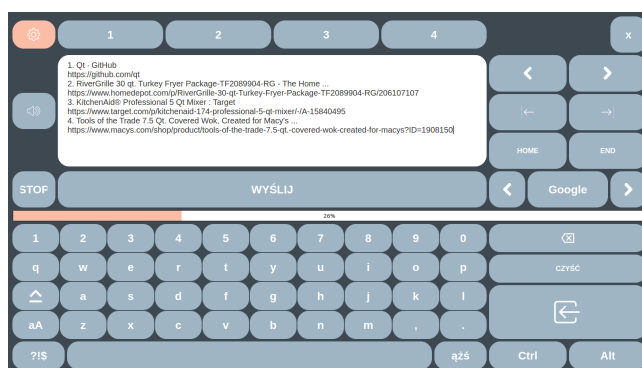
- "title": string
- "htmlTitle": string
- "link": string
- "displayLink": string
- "snippet": string
- "htmlSnippet": string
- "cacheId": string
- "mime": string,
- "fileFormat": string
- "formattedUrl": string
- "htmlFormattedUrl": string
- "pagemap"

Wykorzystując dane z "title" oraz "link" albo przedstawia się użytkownikowi cztery pierwsze wyniki wyszukiwania w polu tekstowym, a ich wywołanie (otworzenie strony) odbywa się przez wybranie przystosowanych przycisków odpowiedzi z numerem wyniku wyszukiwania. Przykład działania przedstawiono na rysunkach 1.10 oraz 1.11.



Rys. 1.10 – Przykładowy tekst wpisany do pola tekstowego, wysyłany jako argument wyszukiwania Google.

W wypadku wyszukiwań w Youtube oraz Filmweb użytkownik nie ma możliwości wyboru wyniku wyszukiwania - otwierany jest pierwszy wynik, który zawiera odpowiednie słowa kluczowe.



Rys. 1.11 – Wyniki wyszukiwania tekstu wpisanego na rysunku 1.10.