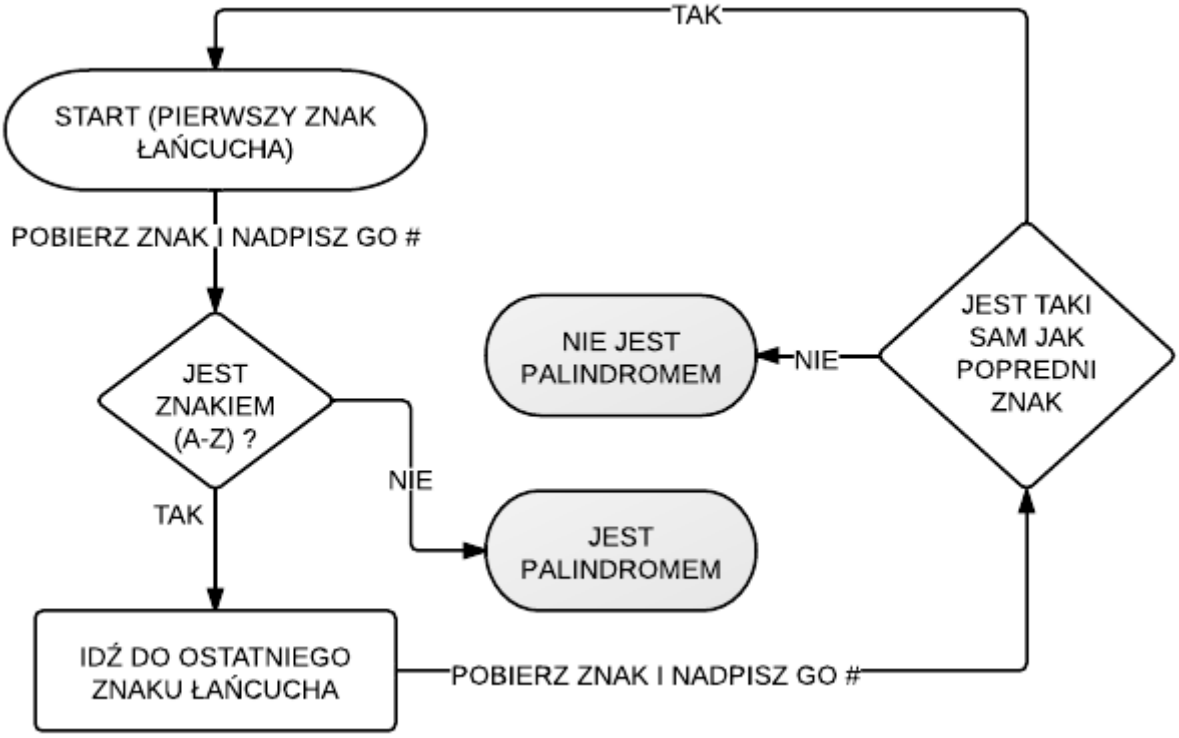
2 Program sprawdzający, czy dany łańcuch jest palindromem. Łańcuch wejściowy składa się z pojedynczego słowa dowolnej długości. Alfabet: {a,…,z,#}

1. Algorytm

Wejściem algorytmu jest ciąg znaków (a-z), stanem końcowym są „NIE JEST PALINDOMEM" oraz „JEST PALINDROMEM".

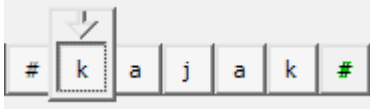
Zgodnie z definicją palindromem jest „*wyrażenie brzmiące tak samo czytane od lewej do prawej i od prawej do lewej*" (źródło 1). Zgodnie z tą definicją wyrazy o nieparzystej liczbie znaków takie jak „kajak" również są palindromami.

Algorytm jest trywialny i przedstawia go poniższy schemat (Rys.1).



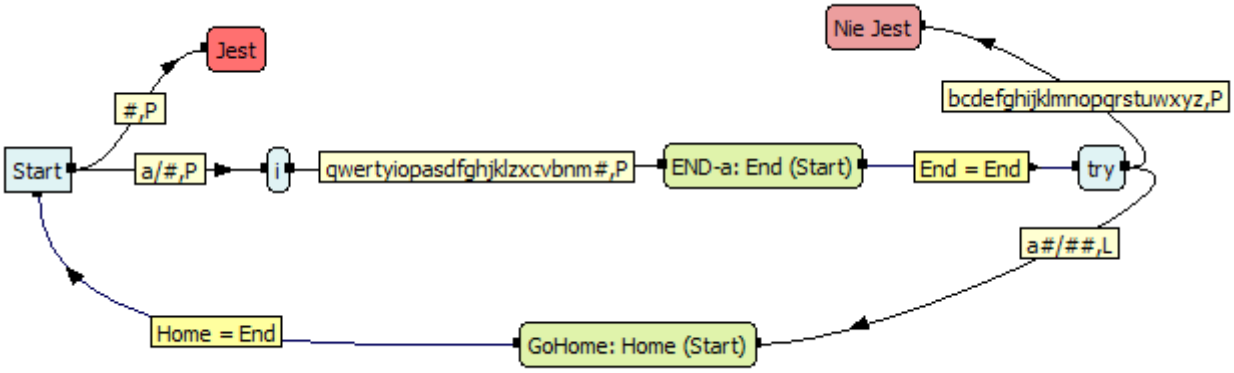
Rys. 1 – Schemat algorytmu

2. Implementacja

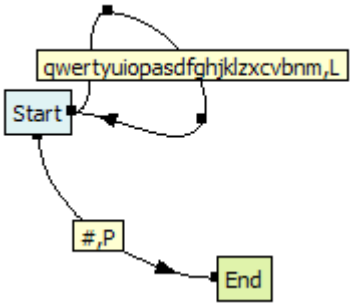


Rys. 2 – Przykładowe wejście (kursor ustawiony na pierwszy znak)

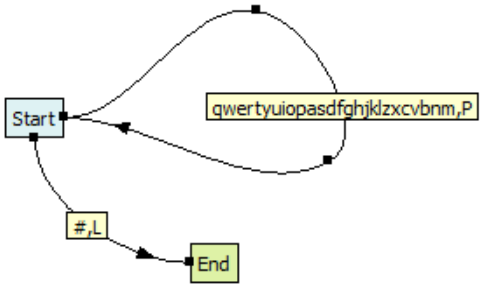
Algorytm sprowadza się do dwóch funkcji oraz porównywania. Jego złożoność związana jest z ilością znaków oraz rozgałęzieniem stanów stanowiącą metodę „pamięci". Algorytm działa do momentu, w którym nie znajdzie znaków (a-b) – sytuacja, w której podany ciąg znaków był palindromem lub do pierwszej różnicy (pierwszy i ostatni znak jest różny).



Rys. 3 – implementacja algorytmu dla jednego znaku (wersja dla znaków a-z w pliku zad2.smt)



Rys. 4 – funkcja Home – idzie do pierwszego znaku



Rys. 5 – funkcja End – idzie do ostatniego znaku

Mechanizm działa poprawnie dla parzystej i nieparzystej liczby znaków.

3. Wnioski

- Część algorytmiczna projektu nie przysporzyła problemów
- Żmudnym procesem było powielanie przypadków dla znaków a-z
- Dzięki zastosowaniu funkcji udało się zredukować objętość algorytmu

4. Źródła

1. http://pl.wikipedia.org/wiki/Palindrom

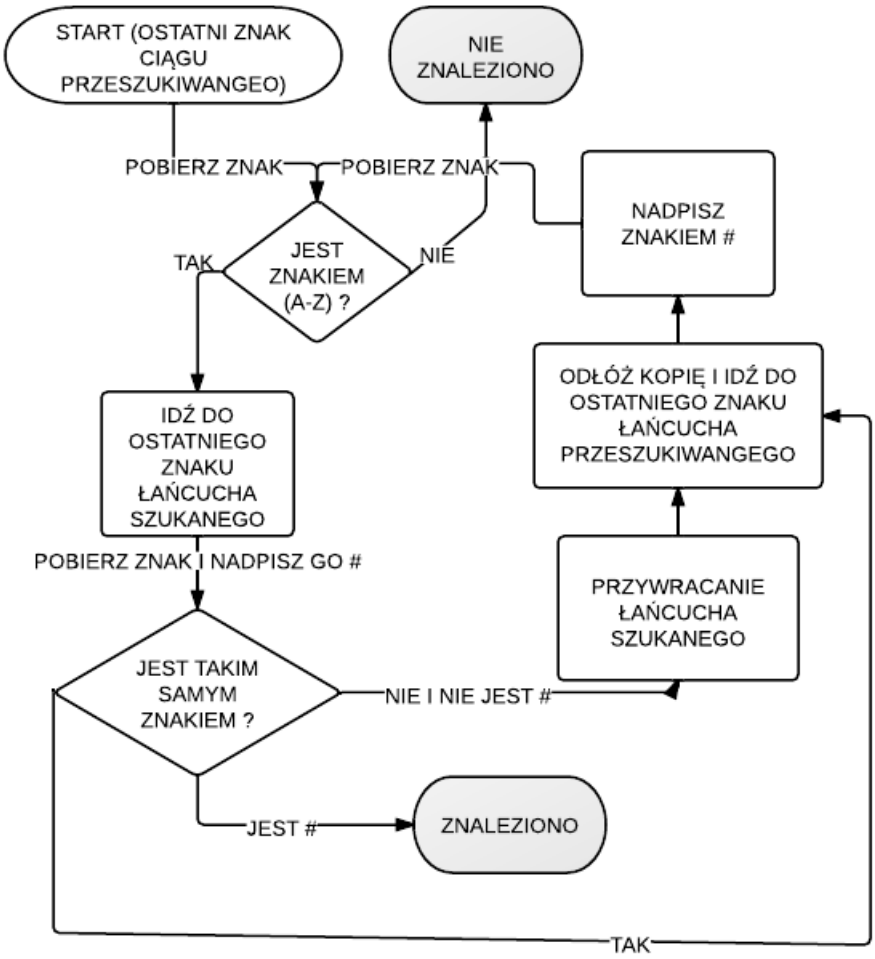
3 Program sprawdzający, czy dany łańcuch symboli zawiera w sobie inny łańcuch. Przy starcie na taśmie zapisany jest łańcuch szukany S i łańcuch przeszukiwany T (szukamy S w T) oddzielone pojedynczym separatorem (…#S#T#...). Łańcuchy mogą zawierać symbole alfabetu {a,…,z} i być dowolnej długości. Dopuszczalny alfabet: {a,…,z,#}.

1. Algorytm

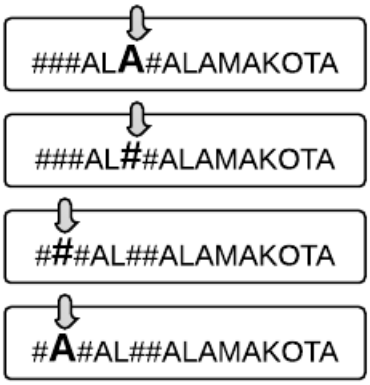
Wejściem algorytmu jest ciąg składający się z łańcucha szukanego oraz łańcucha przeszukiwanego oddzielony znakiem #. Wyjściem natomiast jest stan „ZNALEZIONO” oraz stan „NIE ZNALEZIONO”. Początkowo wskaźnik (kursor) ustawiony musi być na ostatni znak łańcucha przeszukiwanego. Aby poradzić sobie z lokalizowaniem ostatniego znaku nadpisuje użyte znaki. Znaki z frazy szukanej są kopiowane i przechowywane do czasu ich przywrócenia. Algorytm operuje na alfabecie {A,B,C, … , Z, #}.

Schemat przechowywania danych jest następujący

...#*SKOPIOWANE ZNAKI*#*FRAZA SZUKANA*#*FRAZA PRZESZUKIWANA*#...



Rys. 1 – Schemat algorytmu



Rys. 2 – Odkładanie znaku frazy szukanej

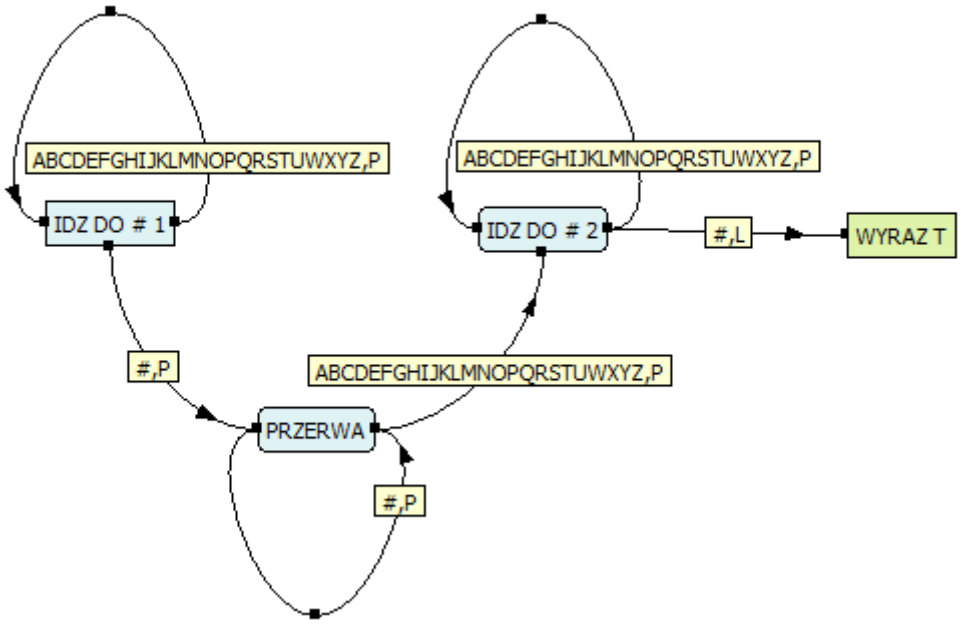
Znaki frazy przeszukiwanej są zamazywane. Jedynym odstępstwem jest przypadek przywrócenia kopii, ostatni znak nie może zostać napisany, ponieważ musi zostać ponowie sprawdzony, tym razem z ostatnim znakiem frazy szukanej.

2. Implementacja

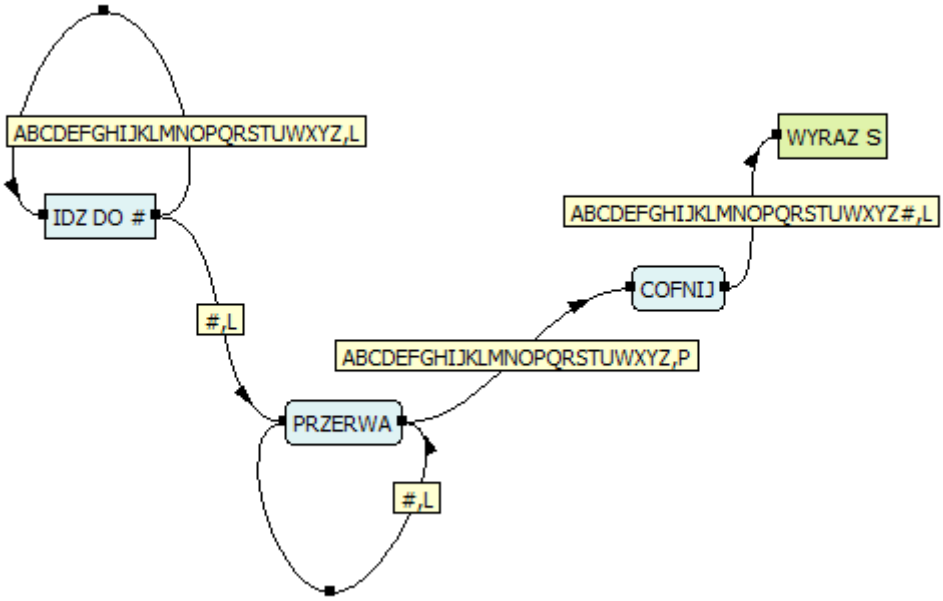
Rys. 3 – Przykładowe wejście – kursor ustawiony na ostatni znak

Algorytm porusza się między łańcuchami za sprawą schematu znakowego. Pomiędzy frazą szukaną a frazą przeszukiwaną może występować dowolna ilość „#”.

...#*SKOPIOWANE ZNAKI****#****FRAZA SZUKANA (S)****#...#****FRAZA PRZESZUKIWANA (T)****#...***



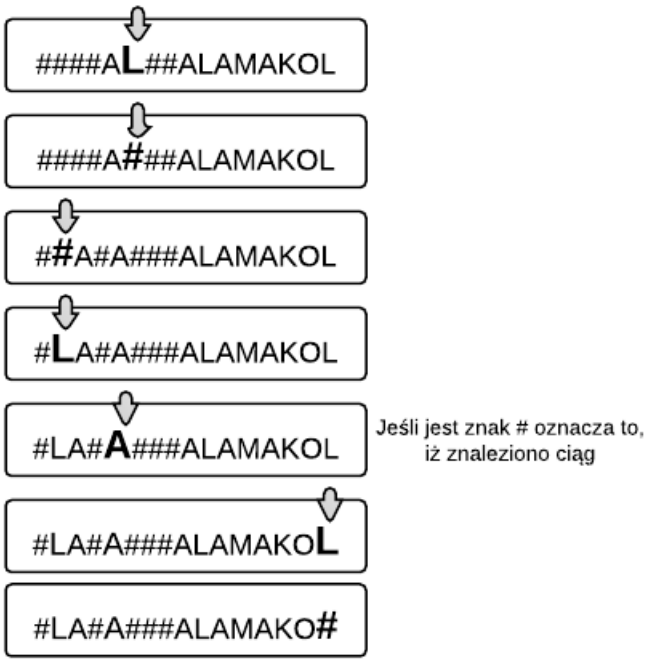
Rys. 4 – Poruszanie się w prawo (do frazy przeszukiwanej)



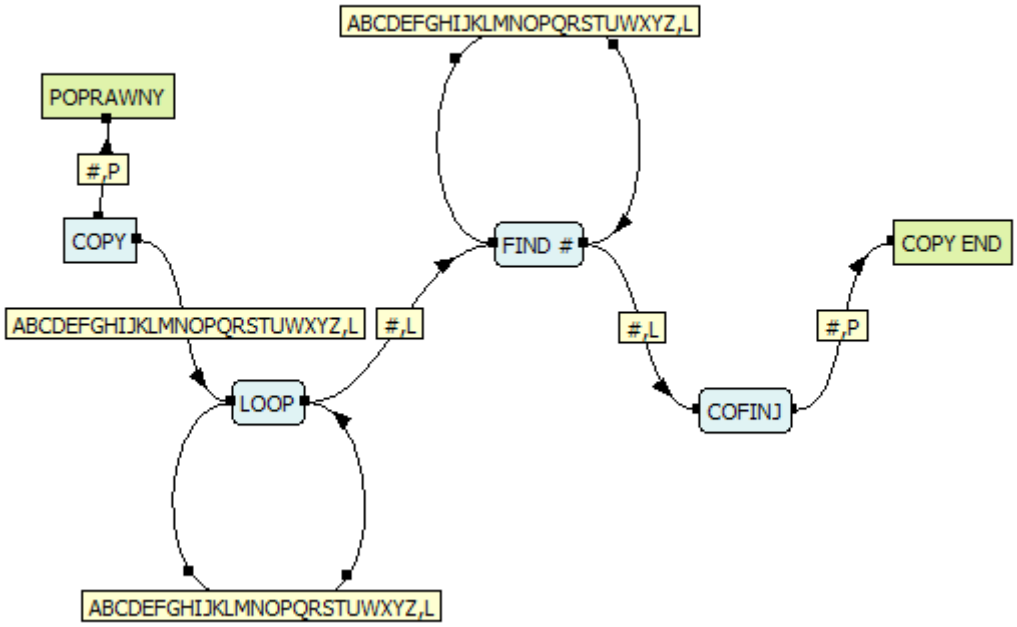
Rys. 5 – Poruszanie się w lewo (do frazy szukanej)

Kolejnym istotnym elementem związanym z utrzymaniem schematu jest odkładanie znaków frazy szukanej. Jest to niezbędna operacja, ponieważ fraza szukana może znajdować się w dowolnym fragmencie frazy przeszukiwanej.

W związku z ograniczonym alfabetem znaki sprawdzone muszą zostać „przykryte” znakiem „#”, który również jest znakiem oddzielającym ciągi. Operacja odkładania znaku ma na celu możliwość zachowania spójności ciągu oraz możliwość jego przywrócenia. Po wykonaniu odłożenia kopii znaku wykonuje się algorytm przywracający kursor do początku frazy szukanej.

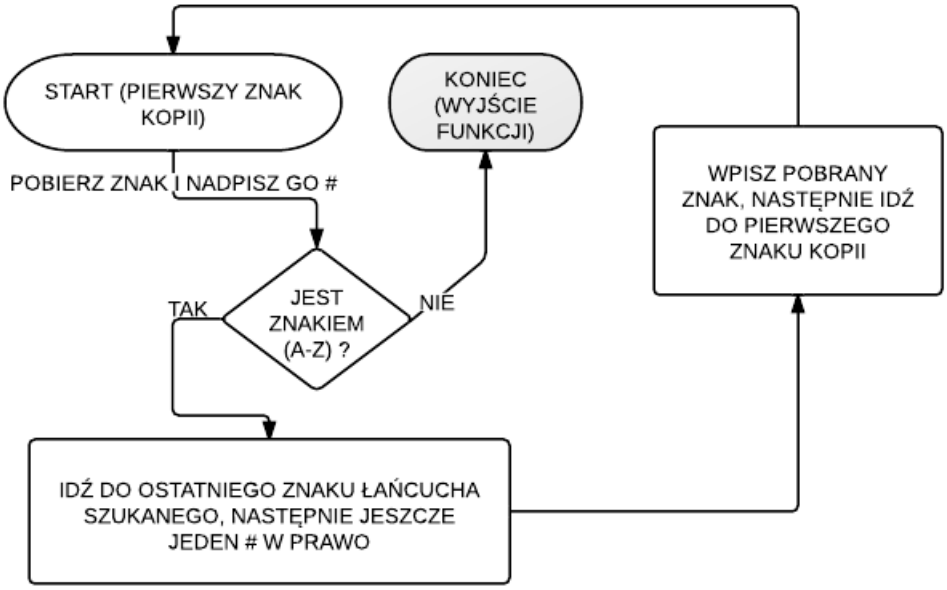


Rys. 6 – Odkładanie kolejnego znaku frazy szukanej



Rys. 7 – Algorytm odpowiedzialny za dojście do ostatniego znaku kopii

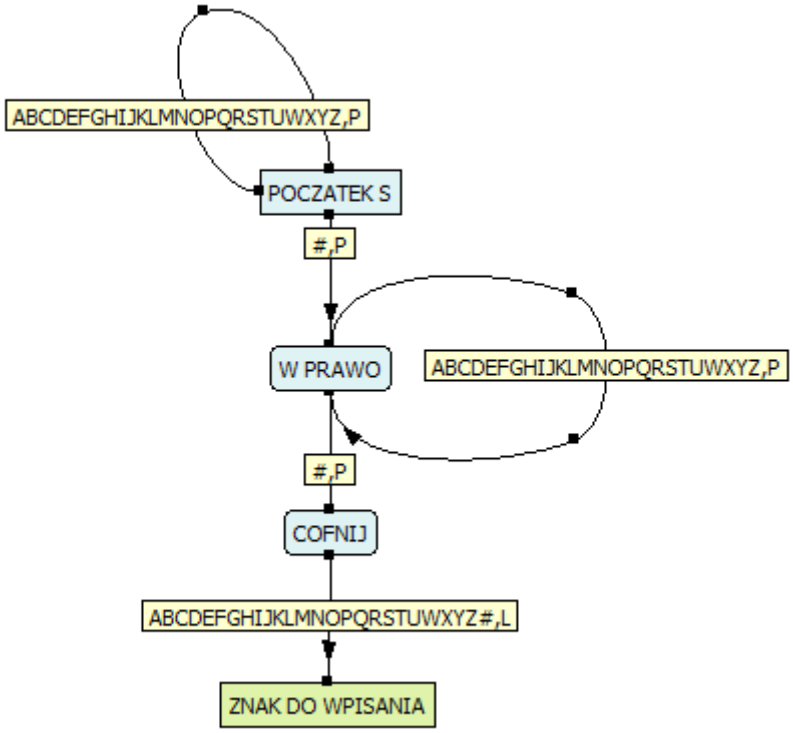
Odwrotnym działaniem jest przywrócenie kopii. Jego implementacja jest analogiczna do działania jakim jest odkładanie.



Rys. 8 – Schemat przywracania ciągu znaków z kopii

Algorytm sterujący pętlą przywracania znaków oraz przypadkami (wpisywanie konkretnych znaków) ze względu na dużą liczbę znaków (A-Z) zajmuje dużo miejsca. Dlatego też dla tego algorytmu pozostanę tylko na schemacie działania.

W załączonym pliku (*zad3.smt*) algorytm znajduje się w zakładce „PRZYWRACANIE”.

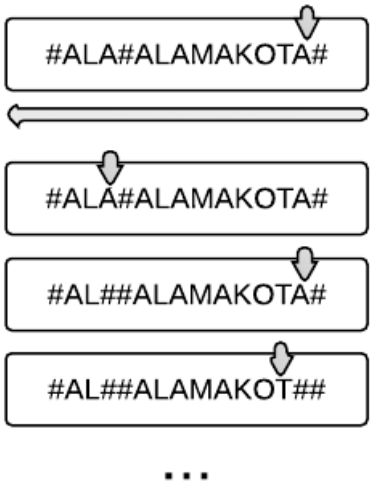


Rys. 9 – Wyszukiwanie miejsca odłożenia (przywrócenia) znaku z kopii

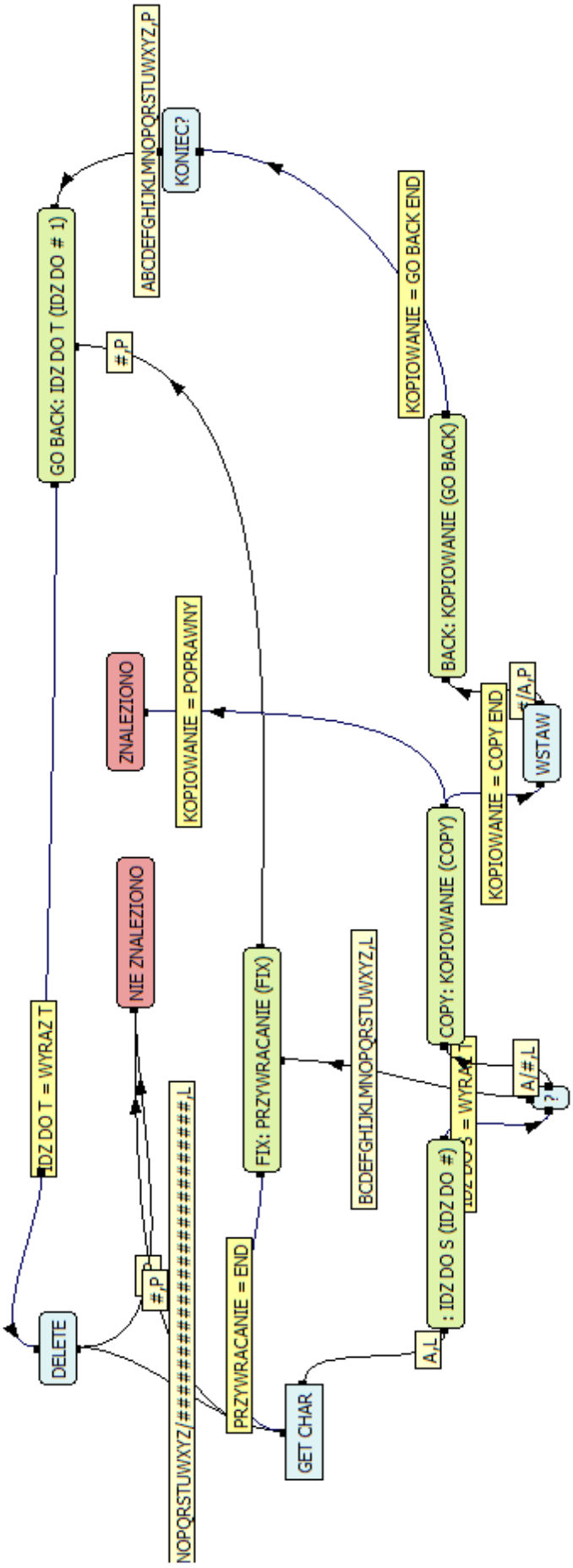
Ostatnią i najważniejszą częścią algorytmu jest mechanizm sterujący. Jego schemat został już przedstawiony w sekcji „Algorytm” (Rys. 1). Algorytm zrealizowany dla wszystkich znaków alfabetu znajduje się w pliku (*zad3.smt*) zakładka „Program główny”.

Algorytm rozpoczyna się od stanu „GET CHAR” - stąd wynika fakt, iż wskaźnik musi wskazywać na ostatni element ciągu przeszukiwanego.

Końcowe stany to „ZNALEZIONO” i „NIE ZNALEZIONO”.



Rys. 10 – Krok działania funkcji głównej (powielany dla kolejnych znaków)



Rys. 11 – Główny algorytm sterujący (wersja czytelna – dla jednego znaku)

3. Wnioski

- Implementacja oraz planowanie algorytmu wymaga zrozumienia praw jakimi rządzi się maszyna Turinga. Prostota maszyny jest zarówno jej wadą i zaletą. Mechanizm jej działania jest trywialny i jego zrozumienie nie przysporzyło mi problemów. Inaczej ma się sprawa implementacji, często musiałem dodawać stan, tylko po to, aby cofnąć kursor.
- W przypadku algorytmu wyszukiwania ciągu znaków niezbędna była implementacja algorytmu „pamięci" - kopii danych. Jego idea jest prosta, lecz implementacja ze względu na ilość możliwości (A-Z) jest czasochłonna.
- Program dostarczony wraz z zadaniem pozwala na proste i intuicyjne tworzenie schematu działania. Jednak to rozwiązanie ma również wady. Pierwszą i najbardziej dokuczającą mi wadą był format pliku (.exe). Fork wine na Mac OS nie był w stanie poprawnie uruchomić programu, więc musiałem używać maszyny wirtualnej. Kolejną wadą jest binarny format outputu – nie pozwala to na modyfikację i łatwe powielenie powtarzających się elementów. Niemniej jednak dzięki pomysłowemu środowisku graficznemu proces debugowania programu to sama przyjemność. Program ma bardzo duże możliwości związane z układaniem elementów, co ułatwia jego czytanie.