Materiały dla finalistów

Malachoviacus Informaticus 2014

7 kwietnia 2014

Wprowadzenie

Poniższy dokument zawiera opisy zagadnień, które będą niezbędne do rozwiązania zadań w drugim etapie konkursu. Polecamy dokładnie zapoznać się z dostarczonymi wyjaśnieniami. Przydatne może być również wyszukanie dodatkowych informacji na poniższe tematy na własną rękę.

Zapoznanie się i zrozumienie informacji zawartych poniżej jest częścią konkursu. Nie jest wymagane odpowiadanie na pytania oznaczone przez **Zastanów się** oraz rozwiązanie zadań na końcu każdej z sekcji, jednak może pomóc w lepszym zrozumieniu materiału.

1 Automaty skończone

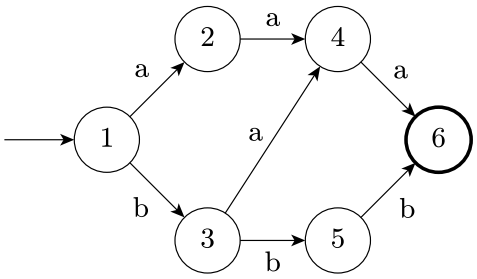
Teoria automatów jest nieodzowną częścią teorii języków – działu informatyki niezbędnego, między innymi, przy budowie kompilatorów, czyli programów tłumaczących kod z języka programowania zrozumiałego dla człowieka na kod maszynowy, zrozumiały dla komputera.

Automat skończony możemy przedstawić jako graf (czym jest graf powinieneś pamiętać z pierwszego etapu), w którym wierzchołki reprezentują *stany*, natomiast krawędzie wskazują nam możliwe *przejścia*.

Tym razem jednak każda krawędź posiada przypisany *symbol*, a także *kierunek*, co oznacza, że po danej krawędzi możemy poruszać się tylko w jednym kierunku. Na diagramie kierunek krawędzi oznaczać będziemy strzałką.

Pośród stanów każdego automatu wyróżniamy jeden stan *początkowy* (na rysunku 1 oznaczony strzałką bez początku) oraz jeden lub więcej stanów *akceptujących* (na rysunku oznaczony pogrubionym obramowaniem).

Taki automat skończony może przyjmować *łańcuchy znaków*, czyli po prostu pewne zbitki liter (lub innych symboli). Każdy taki łańcuch może być *zaakceptowany* bądź *odrzucony* przez automat.



Rysunek 1: Przykładowy automat skończony.

Sposób w jaki ta decyzja jest podejmowana jest dość prosty. Zaczynając od stanu początkowego wędrujemy po automacie wybierając za każdym razem to przejście, które odpowiada następnemu znakowi w łańcuchu. Jeżeli na końcu znajdujemy się w stanie akceptującym, łańcuch został zaakceptowany. W każdym innym przypadku łańcuch zostaje odrzucony.

Weźmy na przykład nasz automat z rysunku 1 oraz łańcuch znaków *baa*. Zaczynamy od stanu 1 i rozpatrujemy pierwszy symbol łańcucha, czyli *b*. Symbol ten prowadzi nas do stanu 3. Tutaj rozpatrujemy drugi symbol łańcucha, czyli *a*, które prowadzi nas do stanu 4. W końcu ze stanu 4 przechodzimy do stanu 6 wybierając przejście *a*. Łańcuch znaków *baa* doprowadził nas do stanu akceptującego, a więc zostaje zaakceptowany.

Zastanów się Jakie inne łańcuchy znaków akceptuje ten automat?

Gdybyśmy jednak spróbowali zrobić to samo z łańcuchem *aa*, zatrzymamy się w stanie 4, który nie jest stanem akceptującym, zatem łańcuch ten zostaje odrzucony.

Zastanów się Jakie inne łańcuchy znaków doprowadzą nas do stanów, które nie są akceptujące?

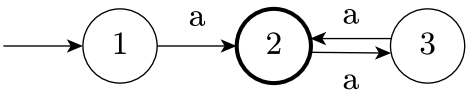
Łańcuch *aba* również zostanie odrzucony, jednak z innego powodu. Po rozpatrzeniu pierwszego symbolu znajdziemy się w stanie 2, skąd nie ma przejścia odpowiadającego symbolowi *b*. Wykraczamy w ten sposób poza automat, nie mamy już możliwości dotarcia do stanu akceptującego, a więc łańcuch zostaje automatycznie odrzucony.

Zastanów się Ile jest różnych łańcuchów znaków, które wyprowadzą nas poza automat? Podaj kilka przykładów.

Pokazany powyżej automat skończony jest dość prosty. Akceptuje tylko kilka różnych łańcuchów znaków i bardzo łatwo można odnaleźć je wszystkie

po spojrzeniu na diagram.

Można jednak zbudować taki automat skończony, który akceptować będzie nieskończenie wiele różnych łańcuchów. Spójrzmy na automat poniżej.



Rysunek 2: Przykładowy automat skończony, który akceptuje nieskończenie wiele różnych łańcuchów znaków.

Możemy łatwo sprawdzić, że akceptuje on łańcuchy *a*, *aaa*, *aaaaa* itp. Nie jesteśmy w stanie, tak jak zrobiliśmy to poprzednio, wypisać wszystkich akceptowanych łańcuchów. Musimy zastosować jakiś inny sposób, żeby je wszystkie opisać.

Jedną z możliwości byłoby zastosowanie *wyrażeń regularnych*, jednak teraz ograniczymy się do opisu słownego. Możemy powiedzieć, że automat z rysunku 2 akceptuje tylko łańcuchy znaków składające się z nieparzystej liczby liter *a*.

Zastanów się Jak będzie wyglądał automat skończony, który będzie akceptować tylko łańcuchy znaków składające się z liter *a*, które są odrzucane przez automat z rysunku?

Zadania

1. Narysuj automat skończony, który będzie akceptował wyłącznie łańcuchy *a*, *b*, *abab* i *bbab*.

2. Narysuj automat skończony, który będzie akceptował wyłącznie łańcuchy znaków składające się z parzystej liczby powtórzeń ciągu *ab*, tj. *abab*, *abababab* itd.

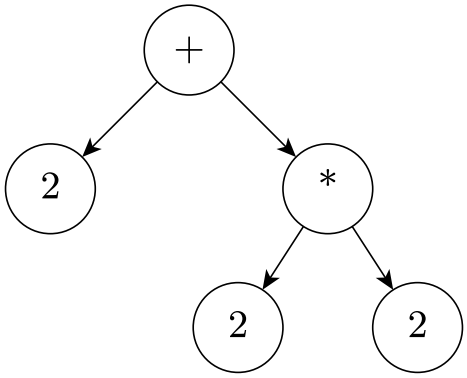
3. Narysuj automat skończony, który będzie akceptował wyłącznie łańcuchy znaków składające się z liter *a* i *b*, o długości podzielnej przez 3, np. *aaa*, *abb*, *abaabb*, *abbabbaba* itp.

2 Drzewa składniowe

Najbardziej rozpowszechnionym sposobem zapisu działań matematycznych jest tzw. *zapis wrostkowy (infiksowy)*. W tym zapisie znak mówiący nam o tym jakie działanie chcemy wykonać (*operator*) znajduje się pomiędzy

liczbami, na których to działanie ma zostać wykonane (*argumenty*). Przykładem działania w zapisie wrostkowym będzie $2 + 2$.

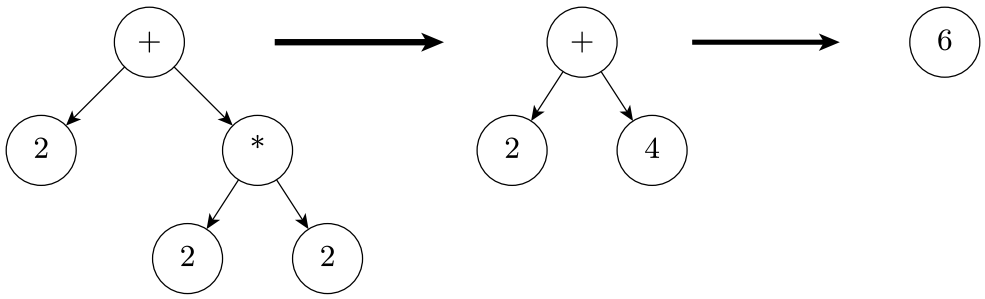
Istnieją jednak inne sposoby zapisu działań matematycznych, niektóre z nich nie są nawet zapisami w jednej linii. Jednen z takich sposobów zapisu stanowią *drzewa składniowe*.



Rysunek 3: Drzewo składniowe przedstawiające działanie $2 + 2 \cdot 2$

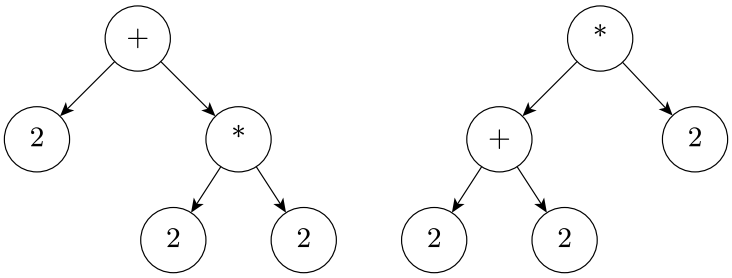
W drzewie składniowym operator znajduje się ponad argumentami i posiada odwołania (oznaczone przez strzałki) do argumentów. Aby poznać wartość działania zapisanego przy użyciu takiego drzewa nie musimy zmieniać jego formy na wrostkową. Możemy wykonywać obliczenia bezpośrednio na drzewie.

Aby to zrobić, musimy znaleźć taki operator, który posiada dwie liczby jako argumenty. Zastępujemy ten operator wynikiem pojedynczego działania, które reprezentuje. Powtarzamy te kroki aż do momentu, kiedy drzewo składać będzie się tylko z jednej liczby.



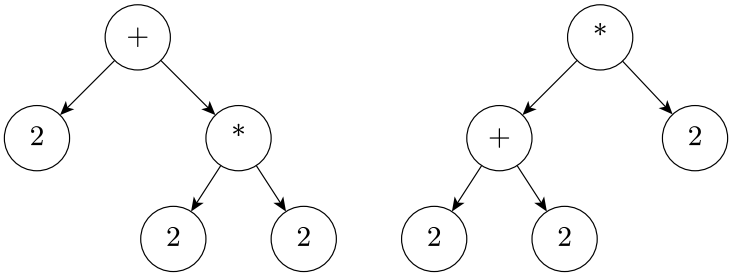
Rysunek 4: Przykład wykonywania działań na drzewie składniowym.

Zwróć uwagę, że taki zapis nie wymaga stosowania nawiasów. Drzewo reprezentujące działanie $(2+2) \cdot 2$ będzie posiadać po prostu inną strukturę niż to dla działania $2 + 2 \cdot 2$.



Rysunek 5: Dwa drzewa posiadające te same liczby i operatory, ale inne struktury. Drzewo po lewej reprezentuje $2 + 2 \cdot 2$, natomiast to po prawej $(2 + 2) \cdot 2$.

Nie oznacza to jednak, że kolejność argumentów nie ma znaczenia. W przypadku działań, które nie są przemienne (jak odejmowanie czy potęgowanie), pierwszy argument zapisujemy po lewej, a drugi po prawej.



Rysunek 6: Ułożenie argumentów ma znaczenie. Po lewej działanie $2^2 - 2$, po prawej natomiast $2 - 2^2$.

Zadania

1. Narysuj drzewo składniowe dla działania $5 \cdot 8 + 2$. Wykonując obliczenia na drzewie znajdź wynik tego działania. Sprawdź czy wynik się zgadza obliczając wartość tego działania bezpośrednio z zapisu wrostkowego.

2. Narysuj drzewo składniowe dla działania 2+2+2+2. Czy istnieje tylko jedno takie drzewo? Uwaga! Upewnij się, że drzewo które narysowałeś reprezentuje dokładnie działanie powyżej, a nie, na przykład, $4 \cdot 2$.

3. Narysuj drzewo składniowe dla działania $(6/3) - (2 \cdot 2)$. Znajdź wynik tego działania wykonując obliczenia na drzewie.

3 Stos

Informacje w pamięci komputera zapisywane są w postaci różnych *struktur danych*. Każda taka struktura posiada jakąś wyróżniającą ją cechę, która czyni ją przydatną w pewnych zastosowaniach.

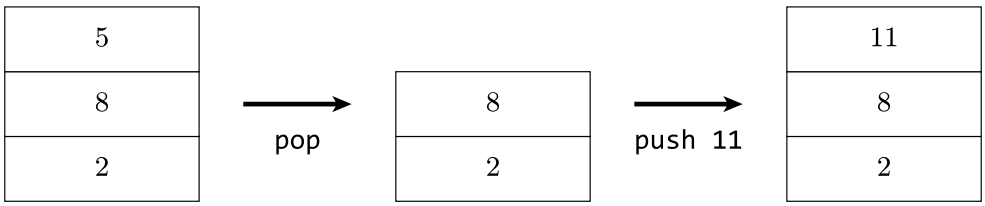
Przykładem takiej struktury danych jest *stos*. Stos przechowuje pewne elementy, umożliwia dodawanie nowych oraz usuwanie tych już zapisanych. Działa przy tym według zasady „ostatni wchodzi, pierwszy wychodzi" (ang. Last In First Out).

Oznacza to, że elementy są usuwane ze stosu w kolejności odwrotnej do tej, w której zostały dodawane. Stąd właśnie pochodzi nazwa „stos".

Dla lepszego zrozumienia wyobraźmy sobie stos książek. Nie możemy wyciągnąć książki ze środka, ponieważ całość się przewróci. Nie możemy też włożyć żadnej książki w środek stosu, ponieważ nie ma tam miejsca. Możemy jedynie kłaść nowe pozycje na górze lub zdejmować je z góry.

Podobnie działa stos w pamięci komputera – jedynym elementem dostępnym bezpośrednio jest ten znajdujący się na samym szczycie. Nowe elementy też możemy dokładać tylko na górę.

Mając dany stos możemy wykonywać na nim dwa rodzaje operacji: `push x` umieszcza wartość x na górze stosu, natomiast `pop` zdejmuje element znajdujący się na szczycie i w odpowiedzi podaje nam jego wartość.



Rysunek 7: Przykład wykonywania operacji na stosie

Operacje te możemy łączyć w ciągi. Będą one wtedy wykonywane w kolejności, w jakiej są podane. Zwróć uwagę, że stos sam w sobie nie umożliwia wykonywania obliczeń. Aby przeprowadzać obliczenia przy użyciu stosu potrzebujemy jakiejś zewnętrznej maszyny, która będzie operować na stosie i wykonywać działania na liczbach na nim się znajdujących.

Zadania

1. Mamy dany stos, który jest początkowo pusty. Wykonujemy na nim poniższy ciąg operacji: `push 5`, `push 2`, `pop`, `push 3`, `push 1`, `pop`. Jak będzie wyglądać stos po wykonaniu każdego z tych działań? Jak będzie wyglądać na końcu?

2. Do stosu podłączona została maszyna, która wspiera następujące operacje:

   - `push x` umieszcza x na szczycie stosu
   - `pop` zdejmuje wartość ze szczytu stosu i wypisuje ją na ekran
   - `add` zdejmuje dwie wartości ze szczytu stosu, dodaje je do siebie i umieszcza wynik na szczycie stosu

   Stos jest początkowo pusty. Do maszyny przekazano następujący program:

```
push 6
push 2
push 3
add
push 5
add
pop
```

   Co wypisze maszyna? Jak będzie wyglądał stos po wykonaniu wszystkich operacji?

3. Napisz program dla maszyny z poprzedniego punktu, który policzy i wypisze sumę liczb od 1 do 5.