Valentine Algorithm cUp

Algorithm Languages Documentation

Prepared by

AmirMohammad Dehghan

amirmd76@gmail.com

Attention: **Any character in your code should be an English letter, a digit or a punct from the set ~`!@#$%^&*()-_=+{}[]\|;:”,’<>./?**

In each line there should be at most one command.

Autolan

This language is processing on a string using an automaton.

An automaton is a directed graph that there is a character on each edge. We call its nodes states instead of vertices. Initially, you are in a state. Characters of the string are given to you one by one and for each character like *c*, if you are in state *s* and there is an edge to state *t* that character on this edge equals to *c*, you move to state *t* (if there is no such edge, do nothing). You name some states, acceptable states. After that characters are all given to you, if the current state is acceptable, then the given string is a valid or Accepted string (with properties you want it to fulfill) otherwise it's Rejected.

You should build this automaton.

So, here some elements are important:

The initial state, acceptable state and edges.

Valid commands:

N n : It should be used in the first line of code, it says that your automaton has *n* states ($1 \leq n \leq 10^6$). Order : *w(n)*

I x : It should be used in the second line of code, it says that initial state is state *x* ($1 \leq x \leq n$). IOrder : *w(1)*

$A\ k\ v_1\ v_2 \ldots v_k$: It should be used in the second line of code, it says that acceptable states are states $v_1, v_2, \ldots, v_k$ ($1 \leq v_i \leq n, 1 \leq k \leq n$). Order : *w(k)*

Your code must contain the commands above exactly once.

E s c t : it says that there is an edge from state *s* to *t* written *c* on it ($1 \leq s, t \leq n$ and there shouldn't be any other command like *E s c x* in your code). Order : *w(1)*

Cursle

This language is processing on a string using a pointer (like curser).

Initially your pointer is one the first character of this string. If at any time string becomes empty your program ends.

Valid commands:

C x : If the character that pointer's on it equals to *x* then, go to the next line, otherwise go to the line after that (after next line). Order : *w(1)*

T d : Go to the line number *Current_line + d* (*d* could be negative). Order : *w(1)*

X : Delete the current character from the string and pointer goes on the next character or the previous one if there is no character after that . Order : *w(1)*

A x : Add character *x* before the pointer.

N : If pointer is not on the last character, set pointer on the next character. Order : *w(1)*

P : If pointer is not on the first character, set pointer on the previous character. Order : *w(1)*

DIT

This language is processing on four storages number from 1 to 4, each containing a non-negative integer.

Valid commands:

D x : If the number in the *x-th* storage equals to 0, then go to the next line, otherwise, decrease it and go to the line after next line.
Order : *w(1)*

I x : Increase the number in the *x-th* storage by 1 ($1 \leq x \leq 4$).
Order : *w(1)*

T d : go to the line *Current_line + d* (*d* could be negative). Order : *w(1)*

IDXT

This language is processing on four storages number from 1 to 4, each containing a non-negative integer.

Valid commands:

I x : Increase the number in the *x-th* storage by 1 ($1 \leq x \leq 4$).
Order : *w(1)*

D x : If the number in the *x-th* storage equals to 0, then go to the next line, otherwise, decrease it and go to the line after next line.
Order : *w(1)*

X a b : Construct numbers *c, d* : if $a \geq 0$ then *c = a*, otherwise *c* = the number in the storage number *–a* . Similarly, if *b > 0* then *d =b* otherwise *d* = the number in the storage number *–b* . Then check if $d \mid c$ then go to the next line, otherwise, go to the line after next line ($-4 \leq a, b \leq 10^9, b \neq 0$). Order : *w(1)*

T d : go to the line *Current_line + d* (*d* could be negative). Order : *w(1)*

Prolan

This language is processing on a string *s*.

There is only one valid command:

(x, y) : Check if *x* is a substring of *s*, then replace the first occurrences of *x* in *s* by *y* and go to the first line, otherwise go to the next line. Order : *w(|s| + |x| + |y|)* (the result of the check doesn't matter, order will be applied anyway).

x and *y* shouldn't contain *(or) or ,* or whitespaces. *y* could be empty but *x* couldn't .