Tema 2 – Arbori de sufixe

Mihai Nan, Andrei Voicu, Lucian Grigore

Data postării: 15.04.2024
Deadline: 30.04.2024 ora 23:59

1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să implementeze și să utilizeze arbori în rezolvarea unor probleme;
- să înțeleagă conceptul de arbore de sufixe;
- să implementeze operații standard pentru arbori multicăi;
- să transpună o problemă din viața reală într-o problemă care uzitează arbori multicăi.

2 Arborele de sufixe

Considerăm un arbore de sufixe drept un arbore multicăi care conține toate sufixele unui text T. Pentru un arbore de sufixe avem îndeplinite următoarele proprietăți:

- Nodul rădăcină are un număr de fii egal cu dimensiunea alfabetului din care este alcătuit textul T (numărul de caractere distincte din T). Spre exemplu, pentru un text ce utilizează alfabetul englez cu litere mici, rădăcina va avea 26 de fii.
- Muchia dintre un nod părinte și un copil este etichetată cu o literă.
- Fiecărui sufix al textului T o să îi corespundă o cale de la rădăcină la un nod frunză. Sufixul poate fi poate fi recompus prin concatenarea tuturor literelor de pe această cale.

Observație

Convenim să terminăm fiecare sufix prin caracterul special \$ (un simbol care nu face parte din alfabet), asigurându-ne astfel că fiecărui sufix îi va corespunde o frunză în arbore.

2.1 Construcția arborelui

Vom porni de la șirul `banana` și vom explica modalitatea prin care putem construi arborele de sufixe.

1. Concatenăm caracterul \$ la șirul `banana`. Vom obține `banana$`.
2. Generăm toate sufixele acestui șir: `$`, `a$`, `na$`, `ana$`, `nana$`, `anana$`, `banana$`.

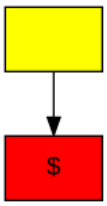
3. Vom insera fiecare astfel de șir în arborele nostru, pornind de la un arbore ce conține un singur nod.

Observație

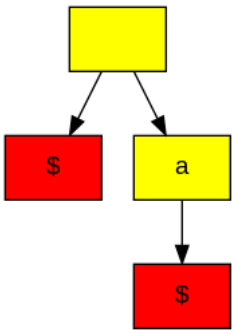
Vom vrea ca nodurile copil să fie sortate alfabetic în funcție de litera asociată muchiei dintre părinte și copil. Vom considera că $ < a.

Pornesc de la arborele care conține un singur nod.

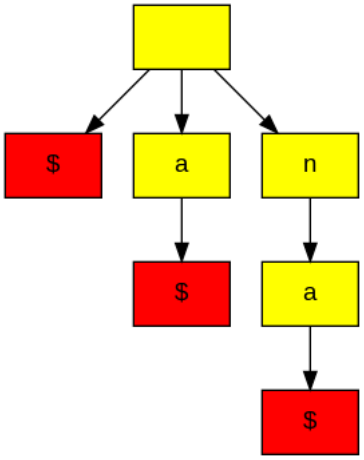
Introduc sufixul $ și voi obține arborele:



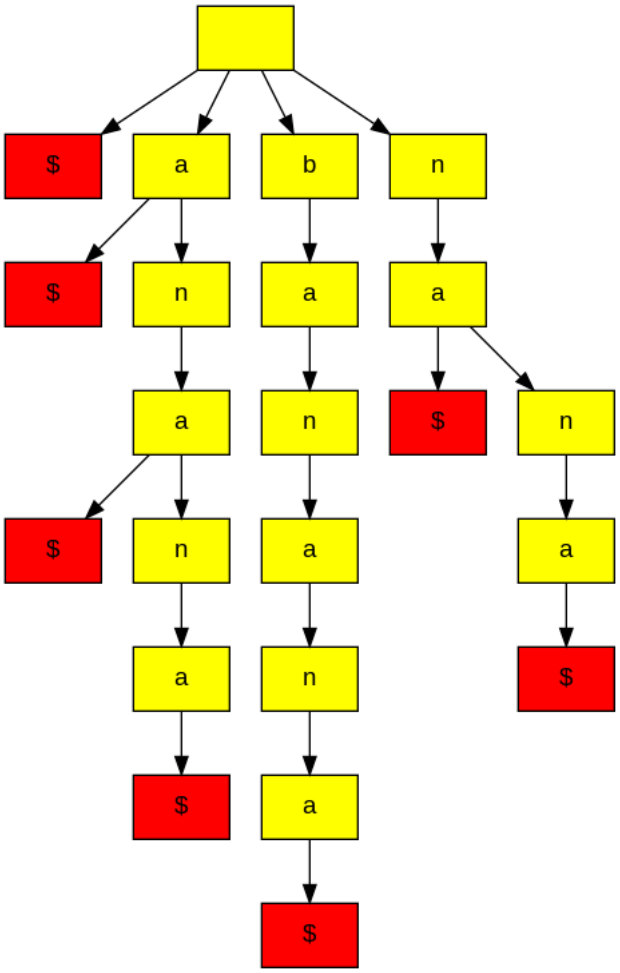
Introduc sufixul a$ și voi obține arborele:



Introduc sufixul na$ și voi obține arborele:



Introduc apoi și celelalte sufixe până ajung la arborele din figura de mai jos.



2.2 Reprezentarea structurii

Pentru reținerea arborelui în memorie, veți utiliza o reprezentare în care fiecare nod conține un vector de pointeri la nodurile fii (succesori direcți). Puteți să alegeți să alocați și realocați dinamic acest vector sau puteți folosi un vector de dimensiune fixă. Numărul maxim de copii pe care i-ar putea avea un nod din arbore este 26 (numărul de caractere din alfabetul englez) + 1 (simbolul folosit pentru terminator `$`).

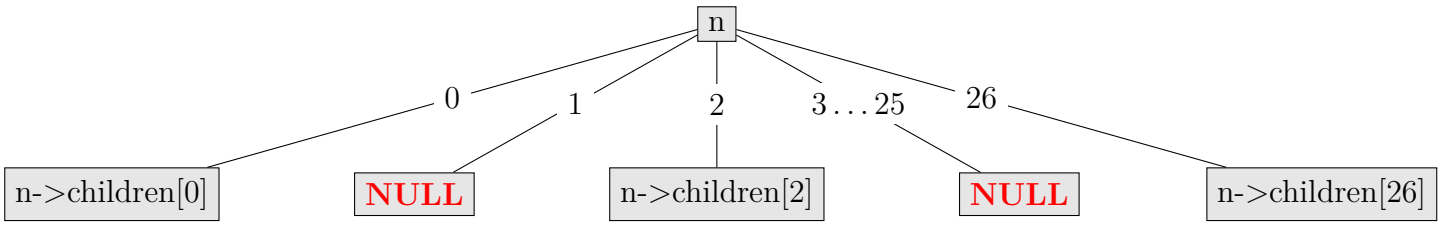
Exemplu de reprezentare în cod:

```
typedef struct node {
    // Orice informație considerată de voi utilă
    struct node *children[27];
} Node, *Tree;
```

Dacă presupunem că avem un nod ce are 3 copii:

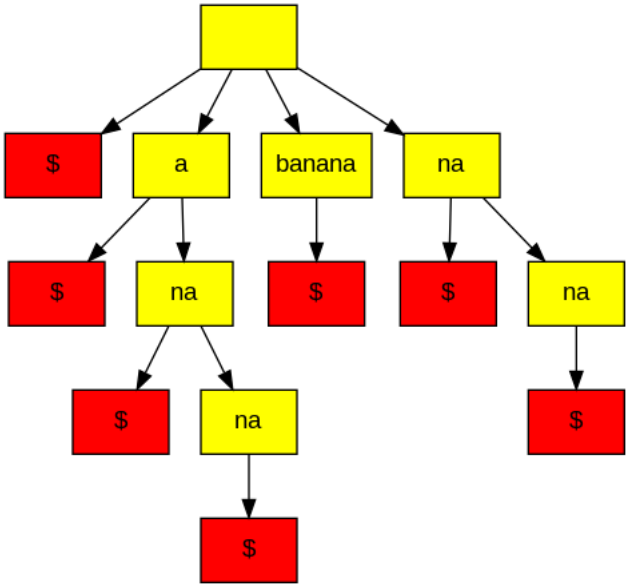
- un copil pentru `$`
- un copil pentru `b`
- un copil pentru `z`

Acesta ar arăta, din punct de vedere al unei posibile reprezentări grafice, în felul următor:

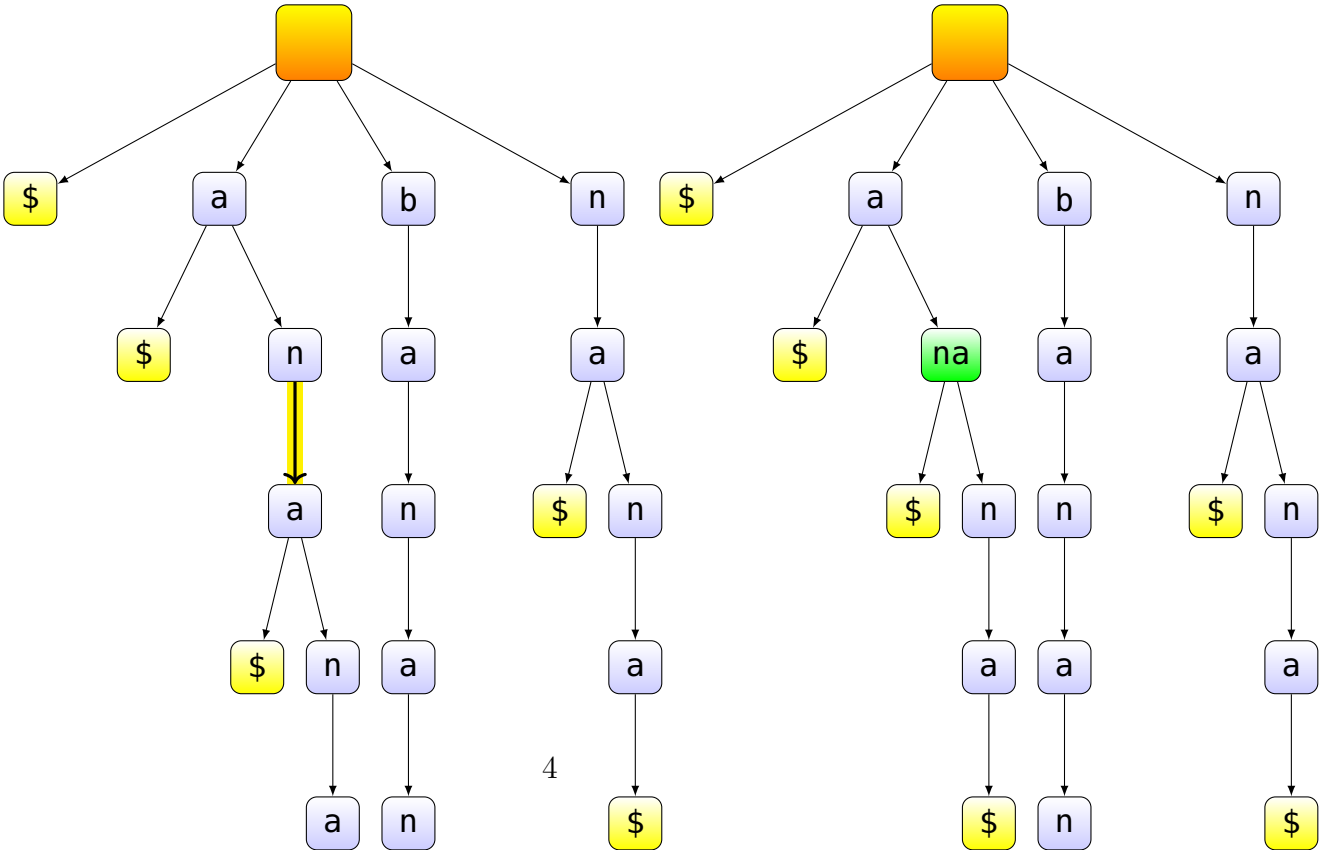


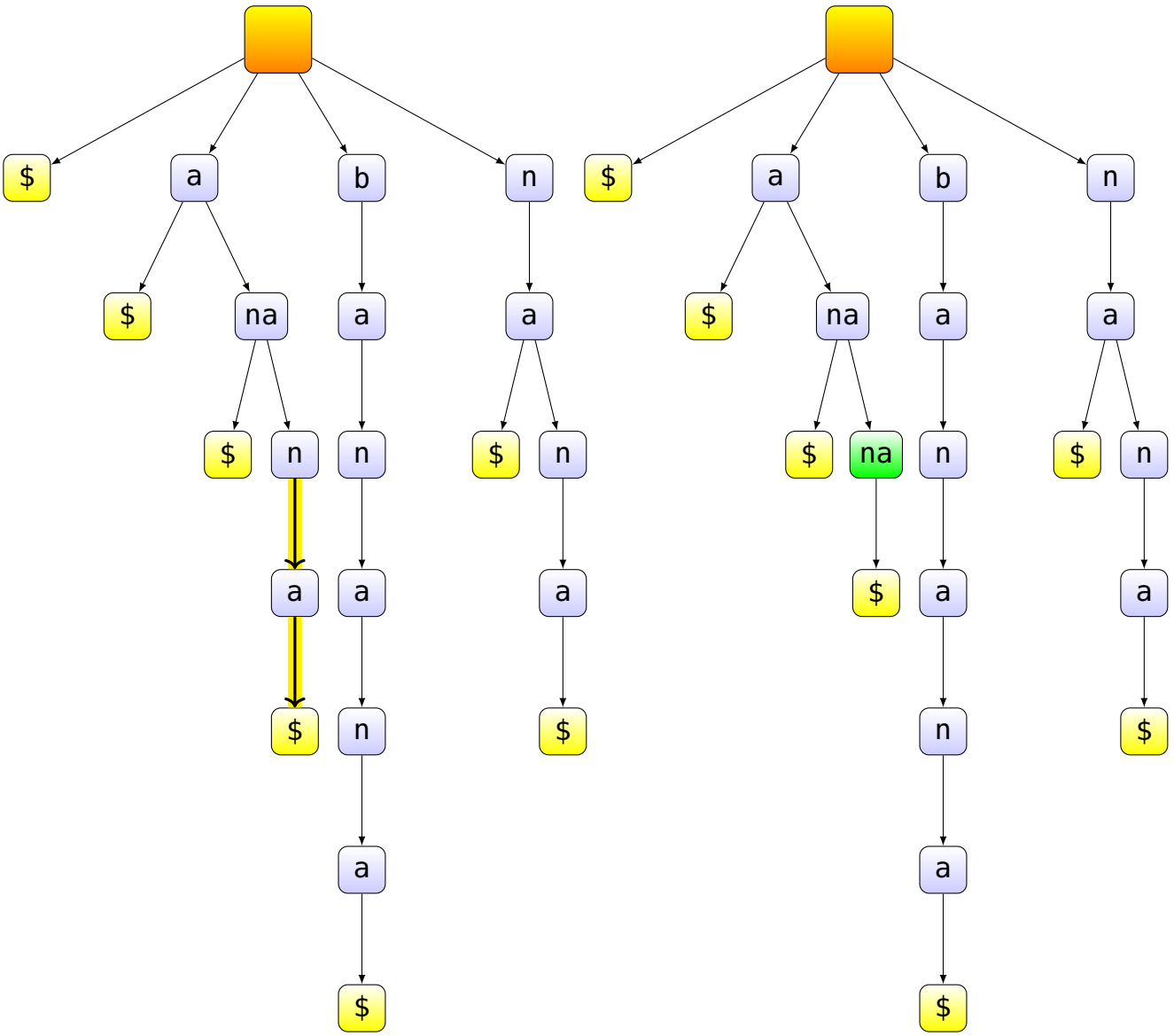
2.3 Compresia căilor

Când muchiile sunt etichetate cu cel mai lung prefix comun al sufixelor din subarborele respectiv, arborele de sufixe se numește compact. Spre exemplu, daca vom aplica compresia căilor pentru obținerea unui arbore de sufixe compact pentru exemplul anterior vom obține arborele următor.

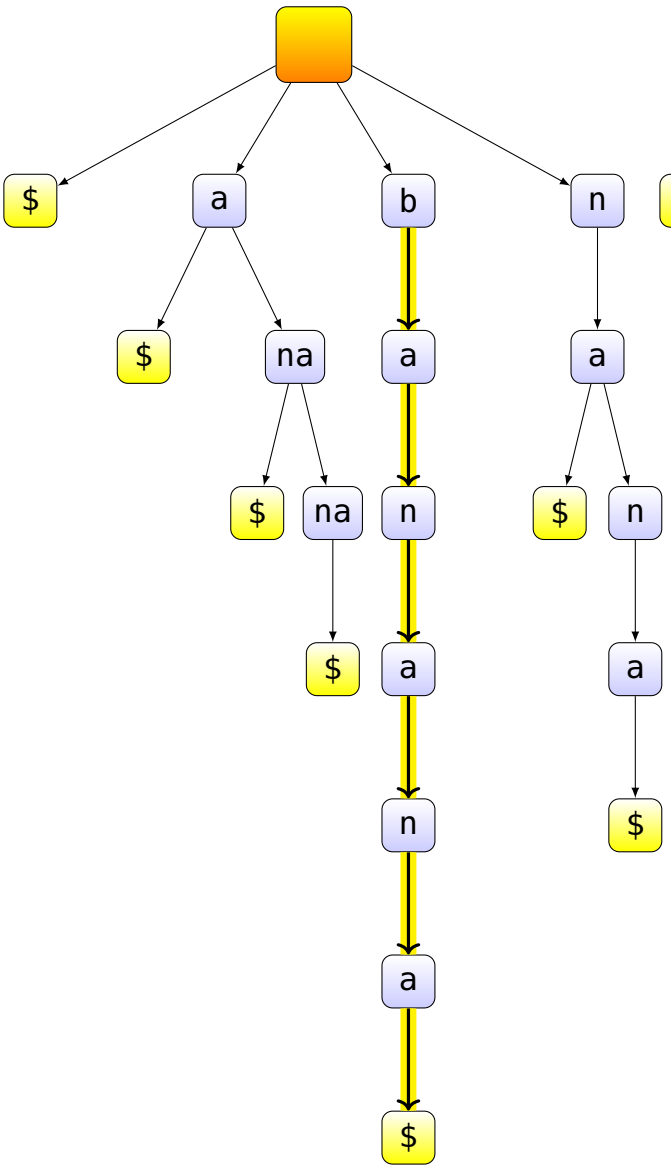


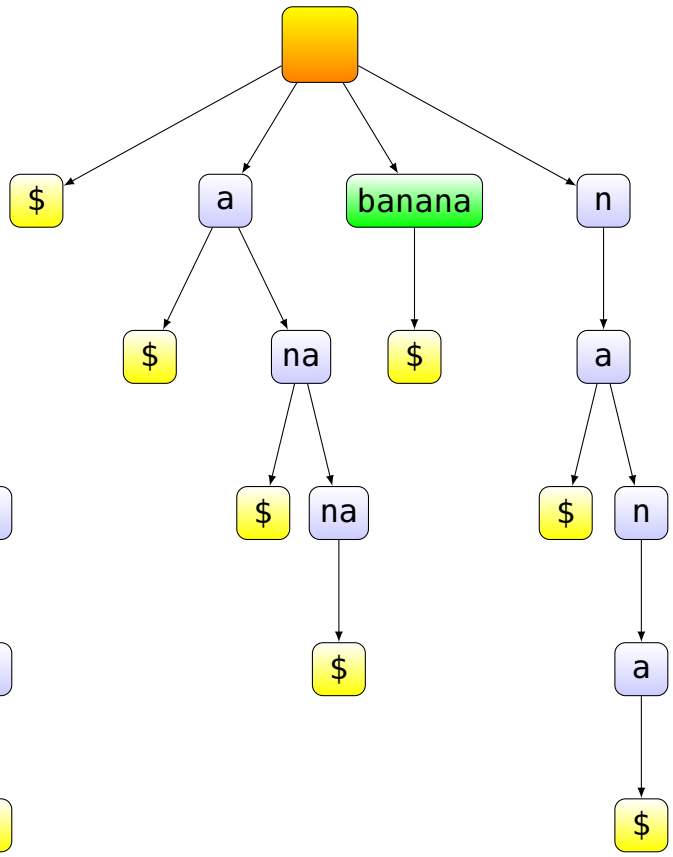
Pentru a înțelege mai bine cum putem determina această variantă compactă, vom evidenția o posibilă transformare pas cu pas:

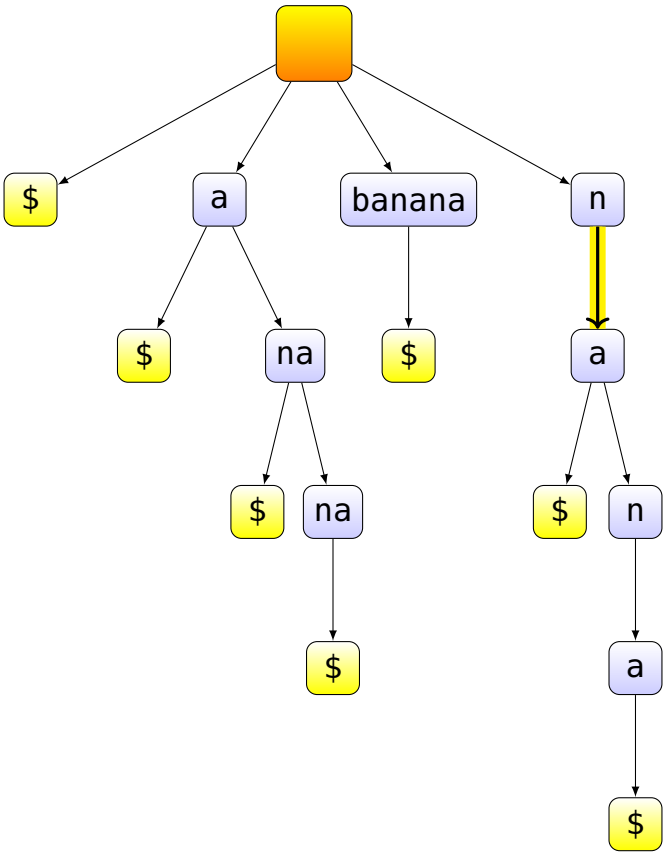


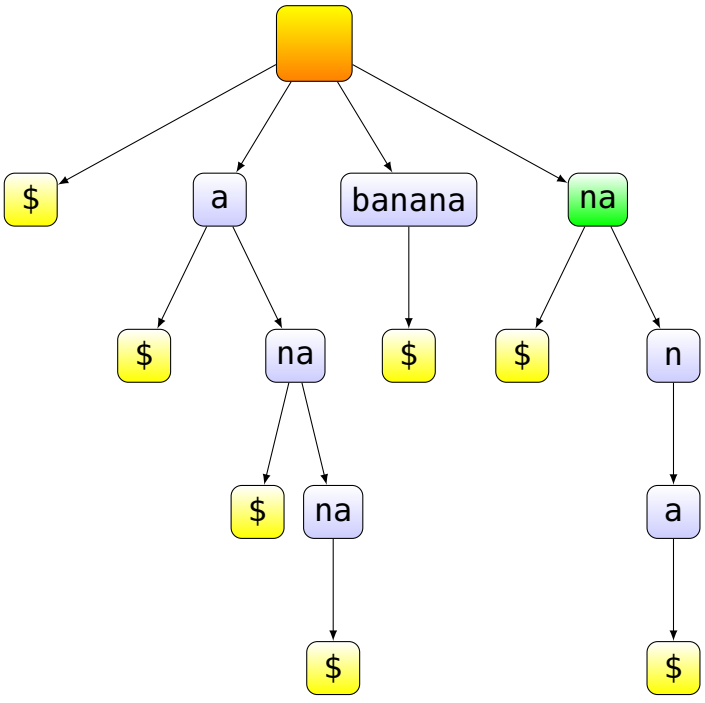

$
a
b
n
$
na
a
a
$
n
n
$
n
a
a
a
$
n
n
a
$
$
a
$

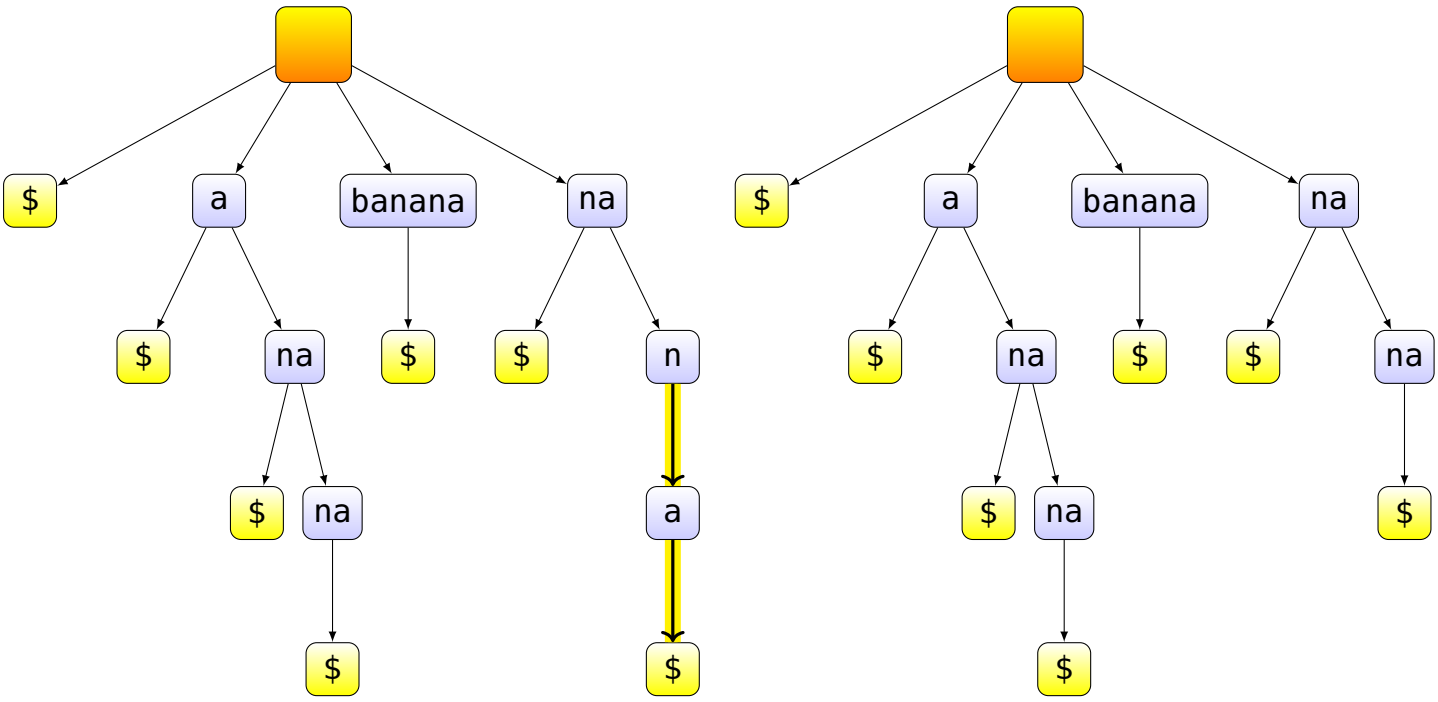
$
a
b
n
$
na
a
a
$
na
n
$
n
a
$
a
$
n
a
$


$
a
b
n
$
na
a
a
$
na
n
n
$
n
$
a
a
n
a
$


$
a
banana
n
$
na
$
a
$
na
$
n
$
a
$


$
a
banana
n
$
na
$
a
$
na
$
n
$
a
$


$
a
banana
na
$
na
$
$
n
$
na
$
a
$



3 Cerințe

3.1 Cerința 1

Pentru prima cerință va trebui să construiți arborele de sufixe pentru textele din fișierul de intrare furnizat ca argument în linia de comandă. În fișierul de intrare va exista pe prima linie un număr natural N, iar pe următoarele N linii vor exista șirurile de caractere (câte un șir pe rând) pentru care va trebui să inserați toate sufixele în arborele de sufixe.

În fișierul de ieșire veți afișa parcurgerea în lățime pentru arborele rezultat.

3.2 Cerința 2

Pentru a doua cerință va trebui să construiți arborele de sufixe pentru un fișier similar ca format cu cel de la Cerința 1 (3.1) și să realizați o serie de statistici pentru arborele rezultat. Pentru a primi punctajul aferent acestei cerințe va trebui să determinați aceste statistici prin aplicarea unor operații pe structura de date de tip arbore și **NU** pe șirurile de caractere din fișierul de intrare.

Statisticile pe care le aveți de determinat sunt următoarele:

- numărul de noduri frunză din arbore;
- numărul de sufixe de dimensiune K (numărul K o să fie specificat ca argument în linia de comandă);
- numărul maxim de descendenți direcți pe care îi are un nod din arbore.

Toate aceste numere vor fi afișate în ordinea specificată, câte unul pe rând.

3.3 Cerința 3

În cadrul cerinței 3 va trebui să verificați dacă anumite sufixe există sau nu în arborele

construit pe baza textelor. Fișierul de intrare va conține pe prima linie două numere naturale N și M. Pe următoarele N linii vor exista șirurile de caractere pe baza cărora vom construi arborele (câte unul pe linie) și pe ultimele M linii din fișier vor exista sufixele (câte unul pe linie) pentru care trebuie să realizăm verificarea pe baza arborelui de sufixe.

Pentru fiecare din cele M sufixe vom scrie în fișierul de ieșire 0 dacă nu există sufixul în arbore sau 1 dacă sufixul există.

3.4 Cerința 4

În cadrul cerinței 4 va trebui să determinați un arbore de sufixe compact pentru textele dintr-un fișier de intrare similar cu cel de la Cerința 1 (3.1). Pentru rezolvarea acestei cerințe puteți construi inițial arborele atomic și apoi să-l prelucrați pentru a ajunge la arborele compact sau puteți construi direct varianta compactă. Acest lucru este la alegerea voastră. După construcția acestui arbore, veți afișa parcurgerea în lățime a arborelui în fișierului de ieșire.

4 Exemple

4.1 Cerința 1

Intrare	Ieșire

1
banana
	$ a b n
	$ n a a
	a n $ n
	$ n a a
	a n $
	$ a
	$

4.2 Cerința 2

Intrare (considerăm k = 3)	Ieșire
1 banana	7 1 4

4.3 Cerința 3

Intrare	Ieșire
1 3 banana na ana ban	1 1 0

4.4 Cerința 4

Intrare	Ieșire
1 banana	$ a banana na $ na $ $ na $ na $ $

5 Restricții și precizări

Temele trebuie să fie încărcate pe vmchecker. **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.

O rezolvare constă într-o arhivă de tip **zip** care conține toate fișierele sursă alaturi de un **Makefile**, ce va fi folosit pentru compilare, și un fisier README, în care se vor preciza detaliile implementării.

Makefile-ul trebuie să aibă obligatoriu regulile pentru **build** și **clean**. Regula **build** trebuie să aibă ca efect compilarea surselor și crearea binarului **tema2**.

Programul vostru va primi, ca argumente în linia de comandă, numele fișierului de intrare și a celui de ieșire, dar și o opțiune în felul următor:

./tema2 [-c1 | -c2 <K> | -c3 | -c4] [fișier_intrare] [fișier_ieșire] unde:

- **-c1** indică faptul că programul va rezolva cerința 1 (**factor** = pragul impus pentru arborele de compresie);
- **-c2 <K>** indică faptul că programul va rezolva cerința 2 (**<K>** = lungimea pentru care dorim să determinăm numărul de sufixe);
- **-c3** indică faptul că programul va rezolva cerința 3;
- **-c4** indică faptul că programul va rezolva cerința 4;
- **fișier_intrare** reprezintă numele primului fișier de intrare;
- **fișier_ieșire** reprezintă numele fișierului de ieșire, în care se va scrie, în funcție de comanda primită, rezultatul execuției programului.

6 Punctaj

Cerinta	Punctaj
Cerința 1	30 puncte
Cerința 2	20 puncte
Cerința 3	20 puncte
Cerința 4	25 puncte
README	5 puncte
BONUS (testat cu valgrind)	20 puncte

Atenție!

Orice rezolvare care nu conține structurile de date specificate **NU** este punctată.
Temele vor fi punctate doar pentru testele care sunt trecute pe vmchecker.
Nu lăsați warning-urile nerezolvate, deoarece veți fi depunctați.
Tema este individuală! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.