به نام خدا

Problem 1:

۱- ابتدا نقش هر کلمه در جملات را مانند زیر مشخص میکنیم.

Mark can watch

N M V

Will can mark watch

N M V N

Can Tom watch?

M N V

Tom will mark watch

N M V N

a)

سپس جدول زیر را تشکیل میدهیم. سطر های این جدول کلمات موجود در مجموعه جملات داده شده هستند.

ستون های آن تگ ها هستند. برای مثال کلمه Mark یک بار به عنوان اسم ظاهر شده است و دو بار به عنوان فعل. سپس هر ستون بر تعداد باری که ظاهر شده است تقسیم میشود. این احتمالات، emission probabilities هستند.

کلمات	noun	modal	verb
Mark	1/6	0	2/4
Can	0	3/4	0
Watch	2/6	0	2/4
Will	1/6	1/4	0
Tom	2/6	0	0

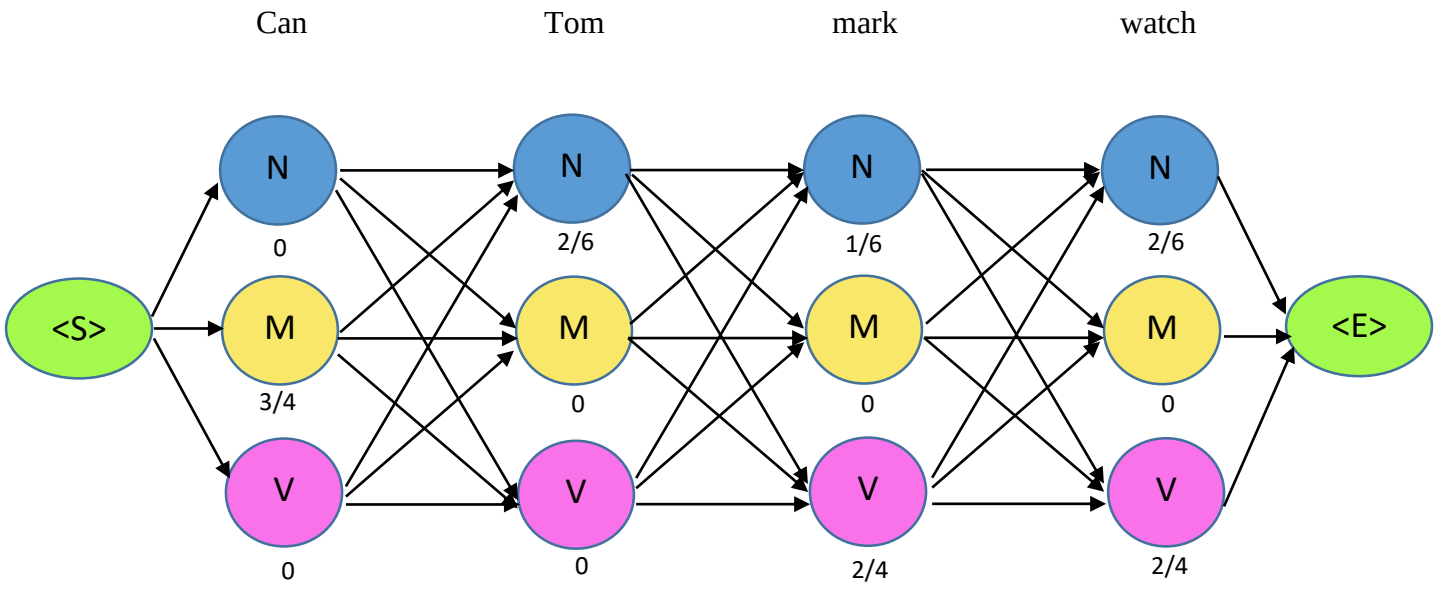
b)

مقادیر جدول زیر transmission probabilities هستند.

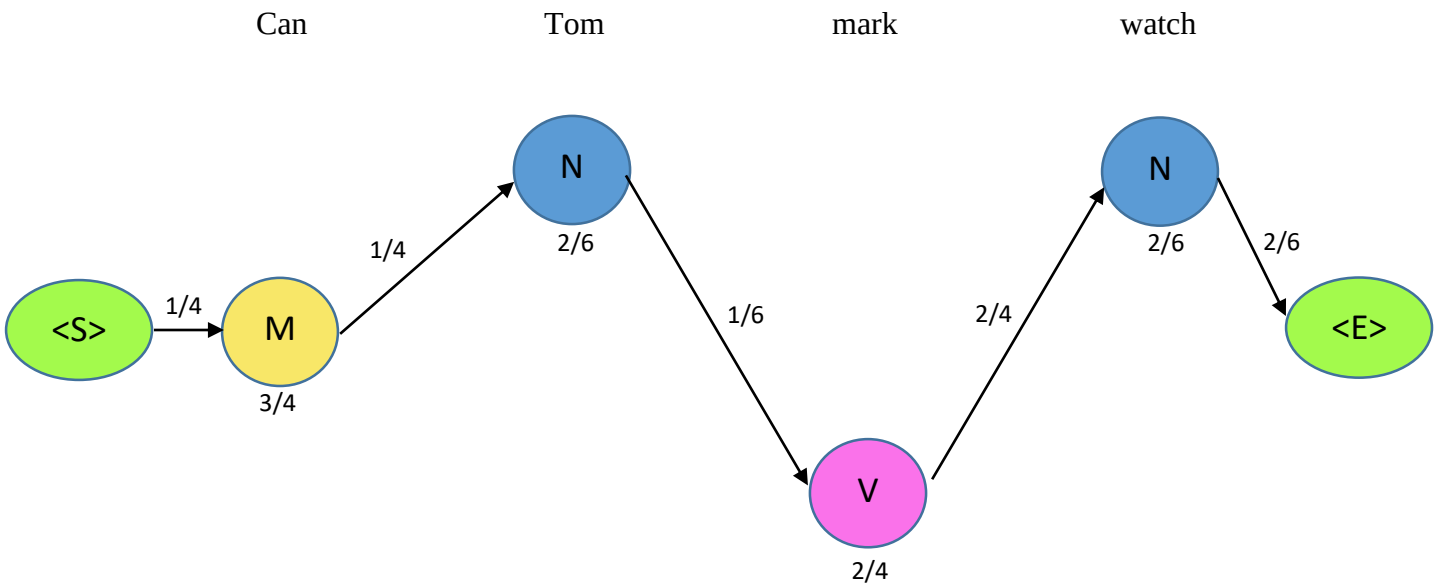
	N	M	V	<E>
<S>	3/4	1/4	0	0
N	0	3/6	1/6	2/6
M	1/4	0	3/4	0
V	2/4	0	0	2/4

c)

ابتدا ۸۱ ترکیب ممکن برای تگ ها را رسم میکنیم.



سپس یال ها و نود هایی که احتمال آن صفر است را حذف میکنیم.



<S>⟶M⟶N ⟶V ⟶N ⟶<E> = 1/4 * 3/4 * 1/4 * 2/6 * 1/6 * 2/4 * 2/4 * 2/6 * 2/6 = 0.0002893

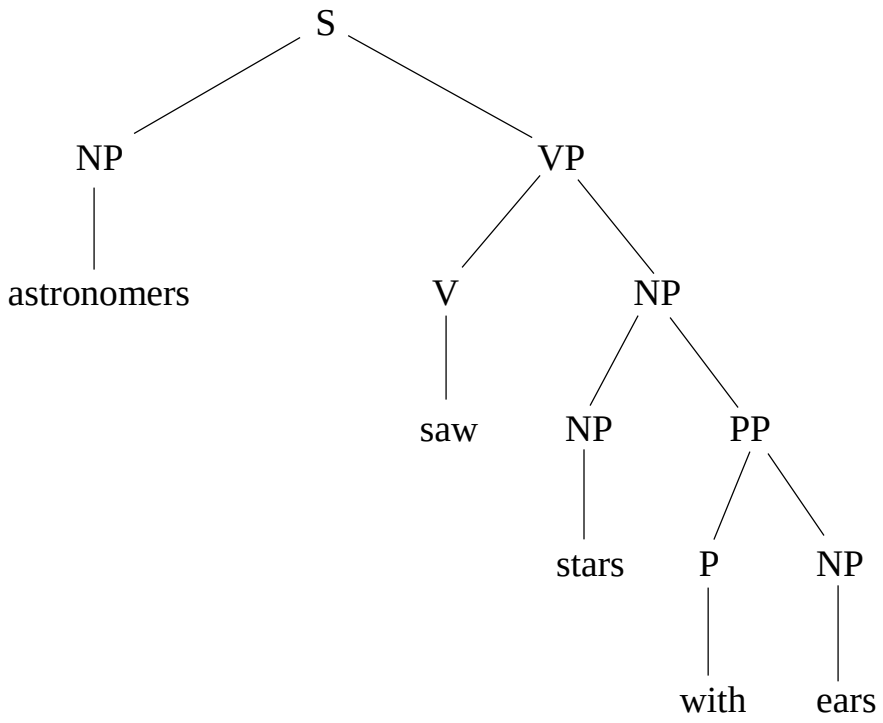
در نهایت ضرب احتمالات این مسیر را محاسبه میکنیم. از آنجایی که یک مسیر بیشتر در این گراف باقی نمانده است، در نتیجه HMM هر کلمه در جمله را مطابق با دنباله به دست آمده در گراف بالا تگ میزند یعنی:

Can	Tom	mark	watch
Modal	Noun	Verb	Noun

:Problem 2

Astronomers	saw	stars	with	ears
NP → astronomers 0.1		S → NP VP 0.0126		S → NP VP 0.0009072
	V → saw 1.0 NP → saw 0.04	VP → V NP 0.126		VP → V NP 0.009072 ~~VP → VP PP 0.006804~~
		NP → stars 0.18		NP → NP PP 0.01296
			P → with 1.0	PP → P NP 0.18
				NP → ears 0.18

این جدول در ۵ مرحله پر میشود. در هر مرحله قطر ها پر میشوند که رنگ آن ها یکسان است. ترتیب پر کردن از بزرگترین قطر به کوچکترین است. برای پر کردن هر خانه تمام ترکیب ها ممکن از constituent ها را در نظر میگیریم و آن هایی که در قوانین وجود دارد را برداشته و constituent های بزرگتر را میسازیم. درخت تجزیه با استفاده از backward trace به صورت زیر به دست می آید.



Problem 3:

(a

ساده ترین راه حل برای پیدا کردن این کتاب ها سرچ داخل متن تمام کتاب های موجود است به طوری که کتاب ها کلمات "satisfaction"، "goals"و "rich" را شامل شوند و کلمات "harassmen" و "hate" را شامل نشود. این کار در unix به عنوان grepping شناخته میشود. یعنی تمام فایل ها را برای رخداد رشته ای از کاراکتر ها بگردیم به طوری که با یک پترن مشخص تطبیق پیدا کند.

(b

خیر این روش به دلایل زیر مناسب نیست:

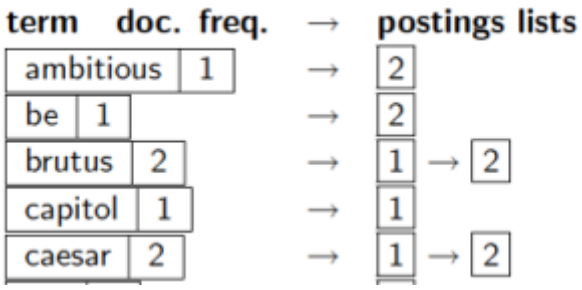
۱- روش grepping برای کارهایی که سایز آن اندازه کتاب های فروید هستند مناسب و سریع است اما برای corpus های بزرگ روشی کند و آرام به حساب می آید و اسکن خطی تمام داکیومنت ها امکان پذیر نیست.

۲- قسمت هایی از کوئری مانند not harassment بدیهی نیستند. این مشکل برای دیگر عملگر ها در کوئری های پیچیده صادق است. برای مثال پیدا کردن کلمه romans در کنار countryman با دستور grep امکان پذیر نیست.

۳- بهترین داکیومنت ها به ترتیب به ما نشان داده نمیشوند. در بازیابی اطلاعات ما انتظار داریم که بهترین نتایج رتبه بندی شوند درحالیکه در اسکن خطی چنین اتفاقی نمی افتد.

(c

برای اینکه داده ها ساختار یافته شوند میتوان از inverted index استفاده نمود که در بازیابی اطلاعات مدرن از آن استفاده میشود. در این ساختار ما دو قسمت اصلی به نام dictionary و postings داریم. شکل زیر نمایش این ساختار است:



با توجه به شکل یک دیکشنری برای کلمات متمایز موجود در تمام داکیومنت ها ساخته میشود. سپس هر کلمه با استفاده از یک اشاره گر به یک posting list اشاره میکند که شامل آی دی داکیومنت ها است و براساس همین آی دی ها مرتب شده است. این posting list میتواند یه linked list باشد که آرایه با طول متغیر است.

(d

در ابتدا یک مرحله اولیه به اسم پردازش متن داریم. در این مرحله ۴ پردازش به ترتیب انجام میشود که به شرح زیر است:

۱- Tokenization: دنباله کاراکترها به توکن های کلمه تبدیل میشود.
۲- Normalization: کلمات مربوط به متن داخل داکیومنت ها و کوئری به یک شکل تبدیل میشوند. برای مثال میخواهیم U.S.A و USA مطابقت داشته باشند.

۳- Stemming: ریشه کلمات را پیدا میکنیم. برای مثال کلمه goals در سوال ما تبدیل به goal میشود.

۴- Stop words elimination: کلمات رایجی مانند a ،the و ... که به آن ها stop word گفته میشود حذف میشوند.

در مرحله بعد یک indexer میسازیم. مراحل ساخت indexer به صورت زیر است:

۱- کلمات پیش پردازش شده از مرحله قبل را به همراه آی دی داکیومنت هایشان در یک جدول مانند جدول زیر قرار میدهیم.

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

۲- جدول را بر اساس term ها و سپس docID ها مرتب میکنیم. که جدول آن به صورت زیر قابل مشاهده است.

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

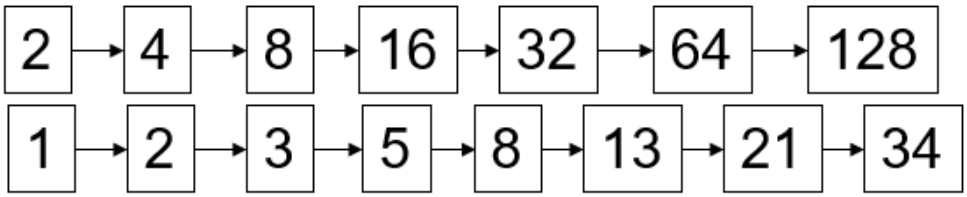
در این مرحله dictionary و posting ها ساخته میشوند. برای اینکار:

- کلمات یکسانی که چندین بار در یک داکیومنت وجود دارند ادغام میشوند.
- جدول به دو قسمت dictionary و postings تبدیل میشود که در سوال قبلی ساختار آن شرح داده شد.
- تعداد باری که این کلمه در داکیومنت ها مختلف آورده شده است در .doc.freq ذخیره میشود.

حال نوبت پردازش کوئری است.

برای اینکه الگوریتم را توضیح دهیم کوئری سوال را کمی ساده تر در نظر میگیریم. برای مثال فرض میکنیم که کوئری ما satisfaction AND rich است.

ابتدا مکان کلمه satisfy (پردازش شده) را در دیکشنری پیدا میکنیم و posting آن را برمیگردانیم. سپس مکان کلمه rich (پردازش شده) را در دیکشنری پیدا میکنیم و posting آن را برمیگردانیم. سپس بین دو لیست برگردانده شده از posting ها اشتراک میگیریم که برای مثال میتواند لیست های ما به صورت شکل زیر باشند.



برای انجام اشتراک دو لیست از الگوریتم زیر استفاده میکنیم (اگر طول لیست ها به ترتیب برابر با x و y باشد، زمان انجام این اشتراک O(x,y) است که خطی است. دلیل خطی بودن آن به ترتیب مرتب بودن لیست ها براساس آی دی ها میباشد) :

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

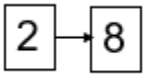
در این الگوریتم دو اشاره گر ابتدای هر دو لیست قرار میگیرند.

اگر آی دی آن ها یکسان بود به عنوان جواب در یک لیست به اسم answer ذخیره میشوند. سپس هر دو اشاره گر یکی به جلو میروند.

اگر آی دی آن ها یکسان نبود، اشاره گری که به آی دی کوچکتر اشاره میکند یکی به جلو میرود.

این روند تا جایی ادامه دارد که یکی از اشاره گر ها به انتهای لیست خود برسد سپس لیست جواب برگردانده میشود.

در مسئله ما نتیجه به صورت زیر است:



Problem 4:

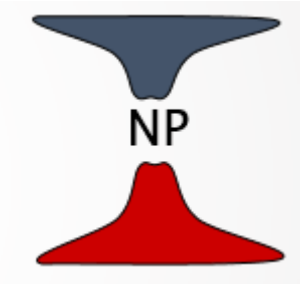
(a

محدودیت های PCFG:

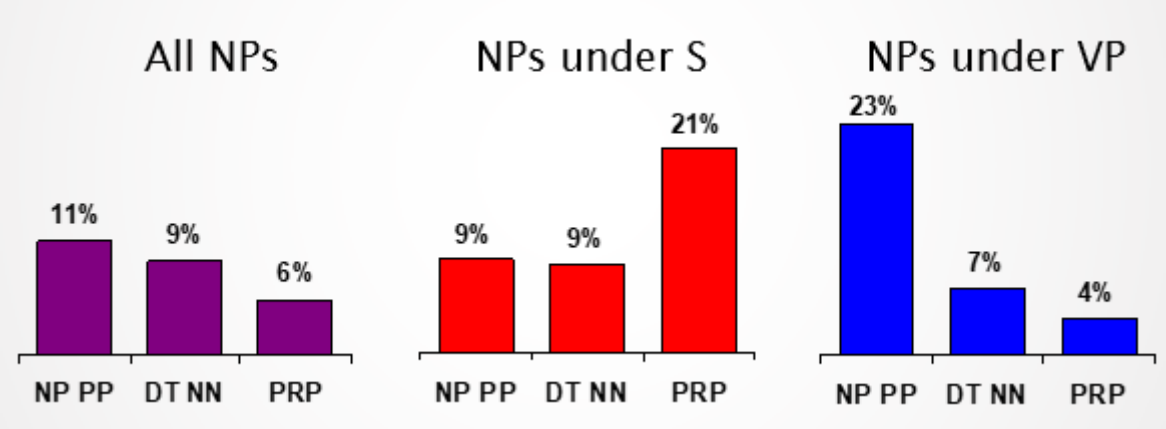
۱- فرض استقلال:

- در هر نودی، فرزندان داخل آن نود مستقل از بقیه درخت است (والد و نود های مجاور و ...).
- هر اطلاعاتی که در محاسبه احتمال آن قانون دخیل است نظیر برچسب آن نود، برچسب پدر و یا کلمه ای که فرزند هد دارد و ... باید در برچسب نود آورده شود.

شکل زیر توضیحات بالا را واضح تر نشان میدهد:



این فرض استقلال، فرض بسیار قوی ای میباشد.



برای مثال احتمالات قوانین شکل بالا با در نظر گرفتن نود والد (نمودار قرمز و آبی) کاملا متفاوت از حالتی است که نود والد در نظر گرفته نشود (نمودار بنفش).

مثال:

٪۹۱ از فاعل ها ضمیر هستند و ٪۹ درصد آن چیزی غیر از ضمیر.

- 91% of the subjects are pronouns.
  - She's able to take her baby to work with her. (91%)
  - Uh, my wife worked until we had a family. (9%)

اما برای مفعول ٪۳۴ درصد از آن ضمیر و ٪۶۶ درصد آن ها چیزی غیر از ضمیر هستند.

- But only 34% of the objects are pronouns.
  - Some laws absolutely prohibit it. (34%)
  - All the people signed confessions. (66%)

۲- عدم توجه به کلمات:

اطلاعات لغوی در PCFG فقط با احتمالا نود های pre-terminal بازنمایی میشوند (مثل verb، noun و ...). درحالیکه اطلاعات لغوی و وابستگی برای مدل کردن احتمالات نحوی مهم است.

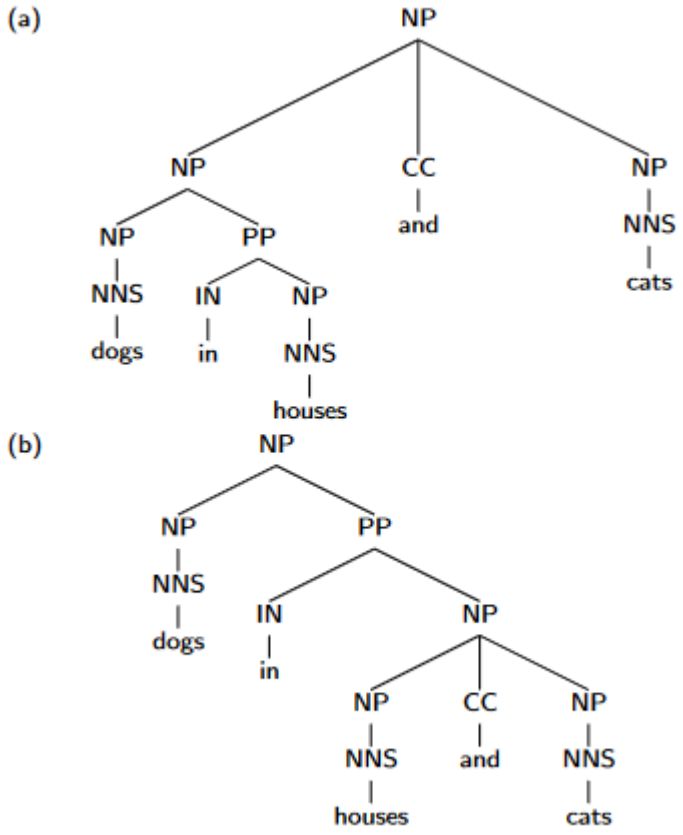
برای مثال:

Moscow sent more than 100,000 soldiers into Afghanistan.

در PCFG، into Afghanistan ممکن است به NP (more than 100,000) یا VP (sent) متصل شود. آمار نشان میدهد اتصال NP ٪۶۷ یا ٪۵۲ است بنابراین PCFG یک نتیجه ناصحیح تولید میکند. چرا؟ چون کلمه send میتواند با حرف اضافه into بیاید. در واقع وقتی فعل send است، into همیشه به آن وصل میشود. به همین دلیل وارد کردن اطلاعات لغوی مهم است. جدول زیر اهمیت لحاظ کردن این کلمات را نشان میدهد:

	come	take	think	want
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%

۳- Coordination ambiguity:



این دو درخت تجزیه به لحاظ ساختار متفاوت هستند و علیرغم اینکه ساختار (a)، ساختار صحیح است ولی PCFG به هر دو درخت احتمالات یکسان را تخصیص میدهد زیرا هر دو درخت از قوانین یکسان استفاده میکنند. قوانین آن ها در جداول زیر قابل مشاهده است:

(a)

Rules
NP → NP CC NP
NP → NP PP
NP → NNS
PP → IN NP
NP → NNS
NP → NNS
NNS → dogs
IN → in
NNS → houses
CC → and
NNS → cats

(b)

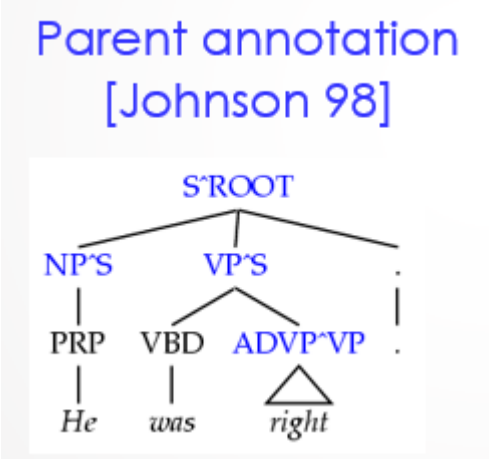
Rules
NP → NP CC NP
NP → NP PP
NP → NNS
PP → IN NP
NP → NNS
NP → NNS
NNS → dogs
IN → in
NNS → houses
CC → and
NNS → cats

(b

با استفاده از lexalized grammer مشکلات نام برده در بخش قبل به شرح زیر حل میشوند:

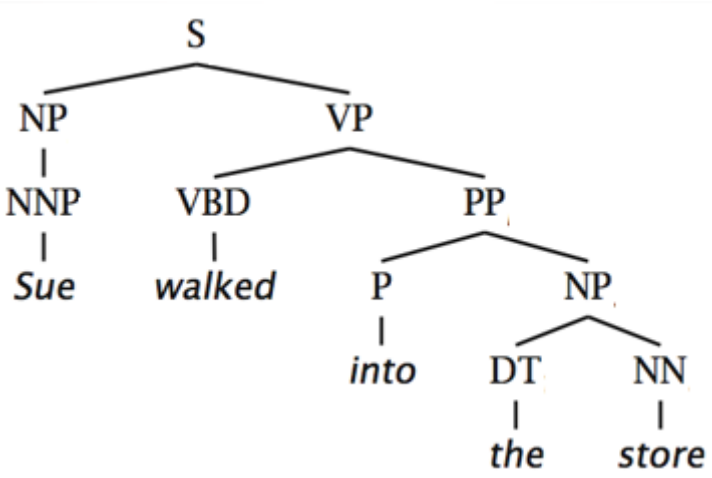
۱- رفع مشکل فرض استقلال:

برای رفع این مشکل میتوانیم از parent annotation استفاده کنیم. به این ترتیب هر قانونی در کنار خود نود پدر خود را هم لحاظ میکند و به آن وابسته است. شکل زیر این وابستگی را نشان میدهد.

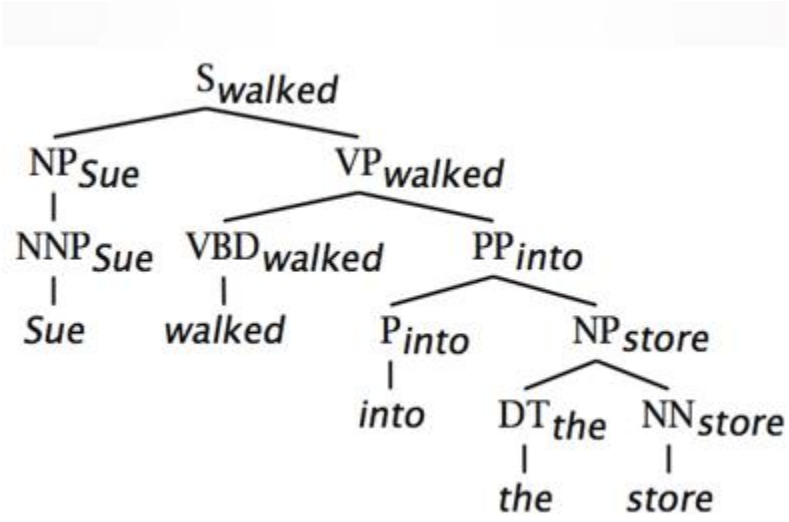


۲- رفع اشکال عدم توجه به کلمات:

برای رفع این مشکل از پایین درخت که برگ ها یا همان non-terminal های ما هستند شروع میکنیم. هر نود والد non-terminal یکی از فرزندان terminal خود را به عنوان head انتخاب میکند و در کنار خود نگه میدارد. این فرایند تا ریشه ادامه خواهد داشت. درخت زیر قبل از دخیل کردن کلمات است:



بعد از استفاده از کلمات درخت آن به صورت زیر خواهد بود:



۳- رفع مشکل coordination ambiguity:

در بخش قبل دیدیم که PCFG به دو درخت تجزیه با ساختار متفاوت احتمالات یکسانی تخصیص میدهد زیرا قوانین آن یکسان بودند. با استفاده از lexicalized grammer و دخیل کردن لغات در قوانین، مجموعه قوانین از یک دیگر تمایز داده شده و احتمال تشکیل درخت تجزیه صحیح بالا میرود.

منابع:

spark-public.s3.amazonaws.com/nlp/slides/Parsing-Lexicalization.pptx

https://www.slideserve.com/teagan/chapter-12-lexicalized-and-probabilistic-parsing

http://www.cs.columbia.edu/~mcollins/cs4705-spring2020/slides/parsing2.2.pdf

Problem 5:

(a

entity یا موجودیت: چیزهای فیزیکی گسته هستند.

Named entity: چیزهای فیزیکی گسسته ای که نام دارند. برای مثال شخصی به اسم Chris Manning یا دانشگاه standford.

در متن های پزشکی میتوان اسامی دارو ها، بیماری ها، ویروس ها و میکروب ها، اعضای بدن، افرادی که روی یک حوزه تحقیقاتی پژوهش انجام داده اند، واکسن ها، شرکت های دارویی و تاریخ ها و... را استخراج کرد.