In [34]:
```
from IPython.display import HTML

# some global setup
HTML("""
<style>
/* JupyterLab: center notebook and leave margins */
.jp-NotebookPanel-notebook {
  width: 85% !important;
  margin-left: auto !important;
  margin-right: auto !important;
  max-width: 1100px !important;       /* optional cap */
}

/* Make output area match nicely */
.jp-OutputArea {
  max-width: 100% !important;
}
</style>
""")

# make high-resolution of images.

%config InlineBackend.figure_format = 'svg'
```

0. Preparing Your Environment

0.1 Install Anaconda and setup environment

> If you are using macOS or Linux, you are ready to go. ⚠️ **If you are using Windows, I strongly recommend installing Ubuntu 22.04 via WSL. In the rest of the instructions, I assume you are using Ubuntu/Linux or macOS.**

- **Step 1: Install Anaconda.** Download and install Anaconda from: https://www.anaconda.com/download.

- **Step 2: Setup your git.** Make sure you have git and ssh in your system (macOS usually already has both git and ssh). After the above step, you are ready to download our course github project. In your Ubuntu/Linux system, install git and sssh via

  `sudo apt update`

`sudo apt install -y git openssh-client`

Then, generate an SSH key (ed25519)

`ssh-keygen -t ed25519 -C "your_email@example.com"`

Press Enter for the default location, and optionally set a passphrase. This creates:

- ~/.ssh/id_ed25519 (private key)
- ~/.ssh/id_ed25519.pub (public key)

Then, you can start ssh-agent and add the key:

`eval "$(ssh-agent -s)"`

`ssh-add ~/.ssh/id_ed25519`

Add the public key to GitHub, that is, copy the generated key:

`cat ~/.ssh/id_ed25519.pub`

It will be something like, `ssh-ed25519 XXXXXXX...XXXXXXX your-email@xxx.com`. **Copy this line**, and then in GitHub: **Settings → SSH and GPG keys → New SSH key → paste.** To test the SSH connection, you can use `ssh -T git@github.com`, if it says something like "Hi USERNAME! ...", you're good. After these steps, you are ready to clone our course github via

`git clone git@github.com:baojian/llm-26.git`

After the above steps, you are ready to create our course env.

- **Step 3: Create and activate the course environment.** Install your conda environment
  - **Option 1:** If you have GPU in your machine (Linux + NVIDIA GPU), please create your env via

    `cd llm-26 # make sure your are in our course folder.`

    `conda env create -f environment-gpu.yml`

    Activate it with:

    `conda activate llm-26-gpu`

    Note: environment-gpu.yml uses `pytorch-cuda=12.1`. If your GPU driver/CUDA setup does not support CUDA 12.1, change this version accordingly (e.g., 11.8).

- **Option 2:** If you use macOS / Windows (WSL) / Linux CPU-only, please create your env via

  `conda env create -f environment.yml`

  Activate it with:

  `conda activate llm-26`

  Note, if you got errors when installing `en_core_web_sm` or `zh_core_web_sm`, please remove these two from yml file and install separately in your env via

  ```
  conda activate llm-26 # or conda activate llm-26-gpu
  python -m spacy download en_core_web_sm
  python -m spacy download zh_core_web_sm
  ```
  After the above steps, you are ready to download our course github project.

- **Step 4: Open your jupyter notebook.**

  All your code will run on Jupyter notebook, you can activate jupyter notebook, via

  `cd llm-26 # make sure you are in our course folder`

  `conda activate llm-26 # or conda activate llm-26-gpu`

  `jupyter lab`

  For students using Windows WSL (Ubuntu 22.04), even though Jupyter runs inside WSL (Ubuntu), you can open it directly in the Windows browser via port forwarding (usually automatic). In WSL Ubuntu, start Jupyter like this:

  `jupyter lab --no-browser --ip=127.0.0.1 --port=8888`

  It will print a URL like: `http://127.0.0.1:8888/lab?token=...`

  Now on Windows, open your browser and go to: `http://localhost:8888/lab`
  Paste the token if it asks.

  ✅ On WSL2, Windows automatically forwards localhost:8888 to the WSL instance in most setups. If localhost:8888 doesn't work, in WSL run: `hostname -I`, suppose it prints something like 172.27.123.45 ..., then open in Windows: `http://172.27.123.45:8888/lab`

0.2 Checking your device

- After install conda and your env, you can check whether these packages are installed in the right way.

```
python -c "import torch; print('torch', torch.__version__);
print('cuda?', torch.cuda.is_available()); print('mps?',
getattr(torch.backends,'mps',None) is not None and
torch.backends.mps.is_available()); print('cuda device:',
torch.cuda.get_device_name(0) if torch.cuda.is_available() else 'no
gpu'); print('using:', 'cuda' if torch.cuda.is_available() else
('mps' if (getattr(torch.backends,'mps',None) is not None and
torch.backends.mps.is_available()) else 'cpu'))"
```

In [36]:
```
!python -c "import torch; print('torch', torch.__version__); print('cuda?',
```

```
torch 2.9.1
cuda? False
mps? True
cuda device: no gpu
using: mps
```

- The following code provides a more symmetric way to perform the same checks.

In [37]:
```
import torch

def detect_torch_device(verbose: bool = True) -> str:
    """
    Returns one of: 'cuda', 'mps', 'cpu'
    Priority: CUDA GPU > Apple MPS > CPU
    """
    has_cuda = torch.cuda.is_available()
    has_mps = getattr(torch.backends, "mps", None) is not None and torch.bac

    if has_cuda:
        device = "cuda"
    elif has_mps:
        device = "mps"
    else:
        device = "cpu"

    if verbose:
        print(f"torch: {torch.__version__}")
        print(f"device: {device}")

        if has_cuda:
            print(f"cuda devices: {torch.cuda.device_count()}")
            for i in range(torch.cuda.device_count()):
                print(f"  [{i}] {torch.cuda.get_device_name(i)}")
        elif has_mps:
            print("mps available: True (Apple Metal)")
        else:
            print("cpu only")

    return device
```

```
device = detect_torch_device()
# generate 2x3 random matrix to check torch
x = torch.rand(2, 3)
x = x.to(device)
print("device:", x.device)
```

```
torch: 2.9.1
device: mps
mps available: True (Apple Metal)
device: mps:0
```

- During our lectures, some datasets are very large. To make sure you have enough disk space, you can check your disk, memory and cpu information via the following code.

In [38]:
```
import os
import platform
import shutil

def bytes_to_gb(x: int) -> float:
    return x / (1024 ** 3)

def system_report(path: str = "."):
    print("=== System Report ===")
    print("OS:", platform.system(), platform.release())
    print("Platform:", platform.platform())
    print("Python:", platform.python_version())

    # CPU
    print("\n--- CPU ---")
    print("CPU cores (logical):", os.cpu_count())

    # Memory (best-effort, cross-platform)
    print("\n--- Memory (RAM) ---")
    try:
        import psutil  # you already have this in env
        vm = psutil.virtual_memory()
        print(f"Total: {bytes_to_gb(vm.total):.2f} GB")
        print(f"Available: {bytes_to_gb(vm.available):.2f} GB")
        print(f"Used: {bytes_to_gb(vm.used):.2f} GB ({vm.percent}%)")
    except Exception as e:
        print("psutil not available or failed:", e)

    # Disk
    print("\n--- Disk ---")
    total, used, free = shutil.disk_usage(path)
    print("Path checked:", os.path.abspath(path))
    print(f"Total: {bytes_to_gb(total):.2f} GB")
    print(f"Free:  {bytes_to_gb(free):.2f} GB")
    print(f"Used:  {bytes_to_gb(used):.2f} GB")

    # PyTorch device
    print("\n--- PyTorch Device ---")
    try:
```

```
        import torch
        cuda = torch.cuda.is_available()
        mps = hasattr(torch.backends, "mps") and torch.backends.mps.is_avail
        print("torch:", torch.__version__)
        print("CUDA available:", cuda)
        print("MPS available:", mps)
        if cuda:
            print("GPU:", torch.cuda.get_device_name(0))
        device = "cuda" if cuda else ("mps" if mps else "cpu")
        print("Suggested device:", device)
    except Exception as e:
        print("torch not available or failed:", e)

system_report(".")
```

```
=== System Report ===
OS: Darwin 24.3.0
Platform: macOS-15.3.1-arm64-arm-64bit
Python: 3.12.9

--- CPU ---
CPU cores (logical): 12

--- Memory (RAM) ---
Total: 18.00 GB
Available: 3.35 GB
Used: 7.48 GB (81.4%)

--- Disk ---
Path checked: /Users/baojianzhou/git/llm-26/lecture-01
Total: 926.35 GB
Free:  320.90 GB
Used:  605.45 GB

--- PyTorch Device ---
torch: 2.9.1
CUDA available: False
MPS available: True
Suggested device: mps
```

1. Explore LLMs via ollama

In this section, we'll have some fun exploring LLMs with ollama.

1.1 Install ollama and download LLMs

We can download some popular LLMs such as Qwen series.

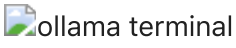
- **Step 0:** Please download and install ollama via download.
- **Step 1:** After the installation of ollama, you can download qwen3:0.6b and qwen3:1.7b via:

`ollama run qwen3:0.6b`

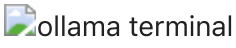
It will automatically download the model into your local disk. It takes about 498MB disk space.

`ollama run qwen3:1.7b`

Similarly, you can run, `shell ollama run qwen3:1.7b` . It takes about 1.3GB disk space. The terminal looks like this:



- **Step 2:** Ask a question, like: `please tell me what is NLP ?` , it gives us:



1.2 Get response for a given prompt

- Next, we go a little deeper, we call the ollama api via python to generate some outputs given prompts.

In [2]:
```python
import os
import time
import math
import requests

# If you have proxy, make sure bypass proxies for local services (e.g., Olla
# You can check whether Ollama is running or not by tpying: http://127.0.0.1
# export OLLAMA_HOST=http://127.0.0.1:11434
# export NO_PROXY=localhost,127.0.0.1
# export no_proxy=localhost,127.0.0.1
os.environ["OLLAMA_HOST"] = "http://127.0.0.1:11434"
os.environ["NO_PROXY"] = "localhost,127.0.0.1"
os.environ["no_proxy"] = "localhost,127.0.0.1"

s = requests.Session()
s.trust_env = False  # <- ignore ALL proxy env vars

print(s.get("http://127.0.0.1:11434/api/version", timeout=3).json())

# !!!! make sure to import after the s.trust_env !!!!

import ollama
from ollama import chat
from ollama import ChatResponse

s = requests.Session()
s.trust_env = False  # <- ignore ALL proxy env vars

response: ChatResponse = chat(model='qwen3:0.6b', messages=[
  {
    'role': 'user',
```

```
    'content': 'Fudan University is located in which city? Answer with one w
  },
])
print(response['message']['content'])

models = ["qwen3:0.6b", "qwen3:1.7b"]
prompt = "Fudan University is located in which city? Answer with one word."

for model in models:
    print('-' * 50)
    start_time = time.time()
    for _ in range(10):
        resp = ollama.generate(
            model = model,
            prompt = prompt
        )
        print(f"{model} with resp {_ + 1}: {resp["response"]}")
    print(f'total runtime of 10 responses of {model} is: {time.time() - star
```

```
{'version': '0.13.5'}
Hangzhou
--------------------------------------------------
qwen3:0.6b with resp 1: Shanghai
qwen3:0.6b with resp 2: Hangzhou
qwen3:0.6b with resp 3: Hangzhou
qwen3:0.6b with resp 4: Shanghai
qwen3:0.6b with resp 5: Shanghai
qwen3:0.6b with resp 6: Shanghai
qwen3:0.6b with resp 7: Shanghai
qwen3:0.6b with resp 8: Shanghai
qwen3:0.6b with resp 9: Shanghai
qwen3:0.6b with resp 10: Beijing
total runtime of 10 responses of qwen3:0.6b is: 10.3 seconds
--------------------------------------------------
qwen3:1.7b with resp 1: Shanghai
qwen3:1.7b with resp 2: Shanghai
qwen3:1.7b with resp 3: Shanghai
qwen3:1.7b with resp 4: Shanghai
qwen3:1.7b with resp 5: Fudan
qwen3:1.7b with resp 6: Shanghai
qwen3:1.7b with resp 7: Shanghai
qwen3:1.7b with resp 8: Shanghai
qwen3:1.7b with resp 9: Shanghai
qwen3:1.7b with resp 10: Shanghai
total runtime of 10 responses of qwen3:1.7b is: 26.4 seconds
```

- **Some key observations:**
  - The smaller model Qwen3:0.6b produces lower quality answers while the relative bigger model Qwen3:1.7b produces high quality answers.
  - The response is a kind of random as each time the response may not be the same.

- Certainly, there are ways to make sure the above to generate a fixed answer. For example, you can use the following code where each time it always produce the maximal probability as the answer:

```
resp = ollama.generate(
model=model,
prompt=prompt,
options={
    "temperature": 0.0,
    "top_p": 1.0,
    "top_k": 0,
    # optional:
    "num_predict": 32,
},
)
print(resp["response"])
```

Let us generate some response that are not one word but a sequence of words.

In [4]:
```
model = "qwen3:1.7b"
prompt = "I am an undergraduate student, please explain LLMs in three senter
resp = ollama.generate(
            model=model,
            prompt=prompt
        )
print(f"Prompt: {prompt} \nResp: {resp["response"]}")
```

```
Prompt: I am an undergraduate student, please explain LLMs in three sentence
s.
Resp: Large Language Models (LLMs) are AI systems trained on massive text da
tasets to understand and generate human-like text. They use complex algorith
ms and neural networks to learn patterns and context in language, enabling t
asks like translation, writing, and answering questions. LLMs are widely use
d in various applications, from customer service to creative writing, demons
trating their versatility and power in natural language processing.
```

1.3 Get response probability from LLMs

From above outputs, you see that each time, the response of these LLMs could be different. Actually, all LLMs are probabilitic model where each time, they response (i.e., answers) prompts (i.e., questions) differently. We can use math language to describe this inference process.

Let $p_\theta(\cdot)$ be the trained probability model. Here, you can think $p_\theta(\cdot)$ as Qwen3:0.6b, Qwen3:1.7b or any other models. Given the prompt *Fudan University is located in which city? Answer with one word.* (a sequence of tokens), the above response will give you an answer, which is also a sequence of tokens. So, the model will use an algorithm to predict the response given the prompt. Use the math language, we can think it tries to calculate the following probability:

$$p_\theta\left(w_{n+1}|w_1, w_2, \ldots, w_n\right),$$

where

- Prompt=[*Fudan University is located in which city? Answer with one word.*] $= [w_1, w_2, \ldots, w_n]$,
- Response=[w_{n+1}].

The model will try to output the most likeliy word w_{n+1} using its own *inference algorithm*. We will detail this part in later lectures. By the definition of conditional probability, we may calculate it as

$$p_\theta\left(w_{n+1}|w_1, w_2, \ldots, w_n\right) = \frac{p_\theta\left(w_1, w_2, \ldots, w_n, w_{n+1}\right)}{p_\theta\left(w_1, w_2, \ldots, w_n\right)}.$$

The above is essential to say that, if we can learn a model that can represent any length sequence of distribution $p_\theta\left(w_1, w_2, \ldots, w_k\right)$, where k could be any positive integer, then we can compute the conditional distribution easily.

In [5]:
```
model = "qwen3:0.6b" # "qwen3:1.7b"
prompt = "Fudan University is located in which city? Answer with one word."
num_top_tokens = 20 # number of top alternatives per generated token
resp = ollama.generate(
    model = model,
    prompt = prompt,
    stream = False,
    logprobs = True,
    think = False,
    top_logprobs = num_top_tokens
)
print("response:", repr(resp["response"]))

# Each element usually corresponds to one generated token
for i, lp in enumerate(resp.get("logprobs", [])):
    tok = lp.get("token")
    logp = lp.get("logprob")
    p = math.exp(logp) if logp is not None else None
    print(f"{i:02d} token={tok!r:>16} logp={logp: .4f} p={p:.4f}")
```

```
response: 'Beijing'
00 token=            'Be' logp=-0.7056 p=0.4938
01 token=         'ijing' logp=-0.2268 p=0.7971
```

In [6]:
```
import os, math, ollama

model = "qwen3:0.6b"
prompt = "Fudan University is located in which city? Answer with one word."

res = ollama.generate(
    model=model,
    prompt=prompt,
    logprobs=True,
    think = False,
```

```
    top_logprobs=10,
    options={"temperature": 0.0, # greedy decoding, it pick the maximal toke
             "num_predict": 20,
            "think": False # do not use thinking model.
            },
)

answer = ''
lp = res["logprobs"]
tokens = [d.get("token", "") for d in lp]
print(f'We use model: {model}')
for i in range(len(lp)):
    tok = lp[i].get("token", "")
    logp = lp[i].get("logprob", None)
    alts = lp[i].get("top_logprobs", [])
    p = math.exp(logp) if logp is not None else float("nan")
    if tok == "\n" or tok == "\n\n": # stop when answer ends (often newline)
        break
    answer += tok
    print(f"--- top probabilities of token-{i:02d} ---")
    for a in alts[:20]:
        prob_a = math.exp(a['logprob'])
        print(f"{a['token']!r:>12}:{prob_a:.5f}")
    print(f"Partial Response: {answer}\n")
print(f"Final Response: {answer}")
```

```
We use model: qwen3:0.6b
--- top probabilities of token-00 ---
        'Be':0.49206
        'Sh':0.19568
         'F':0.15662
        'Ch':0.05117
        'Gu':0.03007
         'H':0.01783
         'S':0.00864
         'J':0.00582
         'Y':0.00489
         'B':0.00326
Partial Response: Be

--- top probabilities of token-01 ---
     'ijing':0.79820
       'ij':0.04685
       'ih':0.03993
     'iping':0.02492
       'iz':0.01518
       'id':0.01420
       'il':0.01024
       'iy':0.00869
      'jing':0.00839
       'ib':0.00628
Partial Response: Beijing

Final Response: Beijing
```

In [7]:
```
model = "qwen3:1.7b"
prompt = "Fudan University is located in which city? Answer with one word."

res = ollama.generate(
    model=model,
    prompt=prompt,
    logprobs=True,
    think = False,
    top_logprobs=10,
    options={"temperature": 0.0, # greedy decoding, it pick the maximal toke
             "num_predict": 20,
            "think": False # do not use thinking model.
            },
)
print(res["response"])
```

```
Fudan University is located in **Shanghai**.
```

- If we do not use thinking mode, it will gives the above response. However, we can still see how Shanghai is chosen during the inference stage.

In [8]:
```
import os, math, ollama

model = "qwen3:1.7b"
prompt = "Fudan University is located in which city? Answer with one word."

res = ollama.generate(
    model=model,
    prompt=prompt,
    logprobs=True,
    think = False, # Do not use the thinking model.
    top_logprobs=10,
    options={"temperature": 0.0, # greedy decoding, it pick the maximal toke
             "num_predict": 20,
            "think": False # do not use thinking model.
            },
)

answer = ''
lp = res["logprobs"]
tokens = [d.get("token", "") for d in lp]
print(f'We use model: {model}')
for i in range(len(lp)):
    tok = lp[i].get("token", "")
    logp = lp[i].get("logprob", None)
    alts = lp[i].get("top_logprobs", [])
    p = math.exp(logp) if logp is not None else float("nan")
    if tok == "\n" or tok == "\n\n": # stop when answer ends (often newline)
        break
    answer += tok
    print(f"--- top probabilities of token-{i:02d} ---")
    for a in alts[:20]:
        prob_a = math.exp(a['logprob'])
        print(f"{a['token']!r:>12}:{prob_a:.5f}")
```

```
    print(f"Partial Response: {answer}\n")
print(f"Final Response: {answer}")
```

```
We use model: qwen3:1.7b
--- top probabilities of token-00 ---
         'F':0.99949
        'An':0.00017
        'Sh':0.00016
        'Fu':0.00014
         'W':0.00001
         'Z':0.00001
         'P':0.00000
      'Hang':0.00000
         'J':0.00000
        'Be':0.00000
Partial Response: F

--- top probabilities of token-01 ---
        'ud':0.99999
        'uj':0.00001
        'uz':0.00000
     'uding':0.00000
        'uy':0.00000
       'udu':0.00000
        'UD':0.00000
       'uda':0.00000
       'udd':0.00000
      'udge':0.00000
Partial Response: Fud

--- top probabilities of token-02 ---
        'an':1.00000
       'ans':0.00000
        'am':0.00000
        'ан':0.00000
     'anian':0.00000
       'anh':0.00000
        'un':0.00000
        'AN':0.00000
        'on':0.00000
       'ian':0.00000
Partial Response: Fudan

--- top probabilities of token-03 ---
' University':1.00000
'University':0.00000
  ' Univers':0.00000
     ' Univ':0.00000
 ' Universe':0.00000
' Universities':0.00000
' Üniversitesi':0.00000
' Universität':0.00000
      ' Uni':0.00000
       ' Un':0.00000
Partial Response: Fudan University

--- top probabilities of token-04 ---
       ' is':1.00000
       '是中国':0.00000
```

```
         '是':0.00000
   ' adalah':0.00000
       ' là':0.00000
        ' 是':0.00000
       '.is':0.00000
        'is':0.00000
       '是我国':0.00000
      ' are':0.00000
Partial Response: Fudan University is

--- top probabilities of token-05 ---
  ' located':1.00000
  ' Located':0.00000
        '位于':0.00000
 ' situated':0.00000
   'located':0.00000
        '位於':0.00000
   'Located':0.00000
       ' in':0.00000
   ' locate':0.00000
 ' location':0.00000
Partial Response: Fudan University is located

--- top probabilities of token-06 ---
       ' in':1.00000
         '在':0.00000
       ' 0.00000:'في
        ' в':0.00000
       ' on':0.00000
        'ใน':0.00000
        'in':0.00000
       ' at':0.00000
   ' within':0.00000
       ' In':0.00000
Partial Response: Fudan University is located in

--- top probabilities of token-07 ---
       ' **':1.00000
 ' Shanghai':0.00000
        ' __':0.00000
      ' the':0.00000
   ' ______':0.00000
        ' *':0.00000
       ' Sh':0.00000
      ' ***':0.00000
     ' ****':0.00000
     ' ____':0.00000
Partial Response: Fudan University is located in **

--- top probabilities of token-08 ---
        'Sh':0.73949
         'W':0.15348
         'F':0.06007
         'P':0.02521
         'N':0.00849
         'S':0.00208
```

```
      'Ping':0.00169
        'Gu':0.00096
      'Hang':0.00095
 ' Shanghai':0.00095
Partial Response: Fudan University is located in **Sh

--- top probabilities of token-09 ---
    'anghai':0.99938
       'ang':0.00058
    'enzhen':0.00001
        'en':0.00001
       'eng':0.00000
        'an':0.00000
       'eny':0.00000
       'enz':0.00000
       'ANG':0.00000
       'aan':0.00000
Partial Response: Fudan University is located in **Shanghai

--- top probabilities of token-10 ---
        '**':1.00000
       '**,':0.00000
       '***':0.00000
       '**:':0.00000
     ' City':0.00000
       ' **':0.00000
       ')**':0.00000
      '****':0.00000
     ' city':0.00000
    '**\n\n':0.00000
Partial Response: Fudan University is located in **Shanghai**

--- top probabilities of token-11 ---
         '.':0.99997
     '.\n\n':0.00003
       '.\n':0.00000
         '。':0.00000
   '.\n\n\n':0.00000
 '.\n\n\n\n':0.00000
        '.<':0.00000
 '.\r\n\r\n':0.00000
         '。':0.00000
       '.</':0.00000
Partial Response: Fudan University is located in **Shanghai**.

Final Response: Fudan University is located in **Shanghai**.
```

2. Basics for Python and spaCy

- **Python**: We will use Python-3.12 in our course.
- **spaCy**: As introduced in https://github.com/explosion/spaCy, spaCy is a library for advanced Natural Language Processing in Python and Cython. It's built on the very

latest research, and was designed from day one to be used in real products. We will use it to demonostrate how to do text tokenization.

- **nltk tool**: In our previous courses, we will introduce nltk tool for tokenization stuff. But we will not cover this part in our new course as these tools are largely iirelvant and outdated. One can find details of nltk at https://github.com/nltk/nltk and website at https://www.nltk.org/.

2.1 Python basics: regular expression

In [19]:
```
# in Python, there is a built in lib re, we can import them
import re
```

- **Disjunction []**

In [24]:
```
# Task: Find woodchuck or Woodchuck : Disjunction
test_str = "This string contains Woodchuck and woodchuck."
result=re.search(pattern="[wW]oodchuck", string=test_str)
print("Matched" if result is not None else "Not found")
result=re.search(pattern=r"[wW]ooodchuck", string=test_str)
print("Matched" if result is not None else "Not found")
```

```
Matched
Not found
```

In [26]:
```
# Find the word "woodchuck" in the following test string
test_str = "interesting links to woodchucks ! and lemurs!"
result = re.search(pattern="woodchuck", string=test_str)
print("Matched" if result is not None else "Not found")
```

```
Matched
```

In [31]:
```
# Find !, it follows the same way:
test_str = "interesting links to woodchucks ! and lemurs!"
result = re.search(pattern="!", string=test_str)
print("Matched" if result is not None else "Not found")
result = re.search(pattern="!!", string=test_str)
print("Matched" if result is not None else "Not found")
assert re.search(pattern="!!", string=test_str) == None # match nothing
```

```
Matched
Not found
```

- **Disjunction [0-9]**

In [36]:
```
# Find any single digit in a string.
result=re.search(pattern=r"[0123456789]", string="plenty of 7 to 5")
print("Matched" if result is not None else "Not found",
     result)
result=re.search(pattern=r"[0-9]", string="plenty of 7 to 5")
print("Matched" if result is not None else "Not found",
     result)
```

```
Matched <re.Match object; span=(10, 11), match='7'>
Matched <re.Match object; span=(10, 11), match='7'>
```

- **Negation:[^**

In [39]:
```
# Negation: If the caret ^ is the first symbol after [,
# the resulting pattern is negated. For example, the pattern
# [^a] matches any single character (including special characters) except a.

# -- not an upper case letter
print(re.search(pattern=r"[^A-Z]", string="Oyfn pripetchik"))

# -- neither 'S' nor 's'
print(re.search(pattern=r"[^Ss]", string="I have no exquisite reason for't")

# -- not a period
print(re.search(pattern=r"[^.]", string="our resident Djinn"))

# -- either 'e' or '^'
print(re.search(pattern=r"[e^]", string="look up ^ now"))

# -- the pattern ‘a^b’
print(re.search(pattern=r'a\^b', string=r'look up a^b now'))
```

```
<re.Match object; span=(1, 2), match='y'>
<re.Match object; span=(0, 1), match='I'>
<re.Match object; span=(0, 1), match='o'>
<re.Match object; span=(8, 9), match='^'>
<re.Match object; span=(8, 11), match='a^b'>
```

- **Operation: |**

In [40]:
```
# More disjuncations
str1 = "Woodchucks is another name for groundhog!"
result = re.search(pattern="groundhog|woodchuck",string=str1)
print(result)
```

```
<re.Match object; span=(31, 40), match='groundhog'>
```

In [41]:
```
str1 = "Find all woodchuckk Woodchuck Groundhog groundhogxxx!"
result = re.findall(pattern="[gG]roundhog|[Ww]oodchuck",string=str1)
print(result)
```

```
['woodchuck', 'Woodchuck', 'Groundhog', 'groundhog']
```

- **Operation:?,*,+,.,$**

In [44]:
```
# Some special chars

# ?: Optional previous char
str1 = "Find all color colour colouur colouuur colouyr"
result = re.findall(pattern="colou?r",string=str1)
print(result)
```

```
# *: 0 or more of previous char
str1 = "Find all color colour colouur colouuur colouyr"
result = re.findall(pattern="colou*r",string=str1)
print(result)

# +: 1 or more of previous char
str1 = "baa baaa baaaa baaaaa"
result = re.findall(pattern="baa+",string=str1)
print(result)
# .: any char
str1 = "begin begun begun beg3n"
result = re.findall(pattern="beg.n",string=str1)
print(result)
str1 = "The end."
result = re.findall(pattern=r"\.$",string=str1)
print(result)
str1 = "The end? The end. #t"
result = re.findall(pattern=r".$",string=str1)
print(result)
```

```
['color', 'colour']
['color', 'colour', 'colouur', 'colouuur']
['baa', 'baaa', 'baaaa', 'baaaaa']
['begin', 'begun', 'begun', 'beg3n']
['.']
['t']
```

In [45]:
```
# find all "the" in a raw text.
text = "If two sequences in an alignment share a common ancestor, \
mismatches can be interpreted as point mutations and gaps as indels (that \
is, insertion or deletion mutations) introduced in one or both lineages in \
the time since they diverged from one another. In sequence alignments of \
proteins, the degree of similarity between amino acids occupying a \
particular position in the sequence can be interpreted as a rough \
measure of how conserved a particular region or sequence motif is \
among lineages. The absence of substitutions, or the presence of \
only very conservative substitutions (that is, the substitution of \
amino acids whose side chains have similar biochemical properties) in \
a particular region of the sequence, suggest [3] that this region has \
structural or functional importance. Although DNA and RNA nucleotide bases \
are more similar to each other than are amino acids, the conservation of \
base pairs can indicate a similar functional or structural role."
matches = re.findall("[^a-zA-Z][tT]he[^a-zA-Z]", text)
print(matches)
```

```
[' the ', ' the ', ' the ', ' The ', ' the ', ' the ', ' the ', ' the ']
```

In [46]:
```
# A nicer way is to do the following
matches = re.findall(r"\b[tT]he\b", text)
print(matches)
```

```
['the', 'the', 'the', 'The', 'the', 'the', 'the', 'the']
```

2.2 (Old way) Tokenization tool: spaCy

- We assume you already installed spaCy tool.

- If you haven't installed yet, open your terminal and activate your llm-26 env, and then run the following to download English LMs.

  ```
  python -m spacy download en_core_web_sm
  ```

  ```
  python -m spacy download zh_core_web_sm
  ```

- **Tokenization via spaCy**

There are two type of tokenizations

- **Top-down tokenization**: We define a standard and implement rules to implement that kind of tokenization.
  - word tokenization (spaCy, nltk)
  - charater tokenization
- **Bottom-up tokenization**: We use simple statistics of letter sequences to break up words into subword tokens.
  - subword tokenization (modern LLMs use this type!)

- **Simple tokenization via white space**

In [49]:
```
# Use split method via the whitespace " "
text = """While the Unix command sequence just removed all the numbers and p
print(text.split(" "))
```

```
['While', 'the', 'Unix', 'command', 'sequence', 'just', 'removed', 'all', 't
he', 'numbers', 'and', 'punctuation']
```

In [50]:
```
# But, we have punctuations, icons, and many other small issues.
text = """Don't you love 🤗 Transformers? We sure do."""
print(text.split(" "))
```

```
["Don't", 'you', 'love', '🤗', 'Transformers?', 'We', 'sure', 'do.']
```

- **White space cannot tokenize Chinese,Japanese,...**

In [52]:
```
text = '姚明进入总决赛'
print(text.split(" "))
```

```
['姚明进入总决赛']
```

- **spaCy works much better**

In [58]:
```
import spacy

nlp = spacy.load("zh_core_web_sm")
text = '姚明进入总决赛'
doc = nlp(text)
print([token for token in doc])
```

[姚明, 进入, 总决赛]

- **Character level tokenization**

In [63]:
```
from spacy.lang.zh import Chinese
nlp_ch = Chinese()
text = '姚明进入总决赛'
print([*nlp_ch(text)])
```

[姚, 明, 进, 入, 总, 决, 赛]

- **Sentence-level tokenization**

In [73]:
```
text = """自然语言处理（英语：Natural Language Processing，缩写作 NLP）是人工智能和
研究计算机处理、理解与生成人类语言的技术。此领域探讨如何处理及运用自然语言；自然语言处理包括
基本有认知、理解、生成等部分。自然语言认知和理解是让电脑把输入的语言变成结构化符号与语义关系
然后根据目的再处理。自然语言生成系统则是把计算机数据转化为自然语言。\
自然语言处理要研制表示语言能力和语言应用的模型，建立计算框架来实现并完善语言模型，\
并根据语言模型设计各种实用系统及探讨这些系统的评测技术。"""
nlp = spacy.load("zh_core_web_sm")
doc = nlp(text)
print([sent.text for sent in doc.sents])
```

['自然语言处理（英语：Natural Language Processing，缩写作 NLP）是人工智能和语言学领域的交叉学科，研究计算机处理、理解与生成人类语言的技术。', '此领域探讨如何处理及运用自然语言；自然语言处理包括多方面和步骤，基本有认知、理解、生成等部分。', '自然语言认知和理解是让电脑把输入的语言变成结构化符号与语义关系，然后根据目的再处理。', '自然语言生成系统则是把计算机数据转化为自然语言。', '自然语言处理要研制表示语言能力和语言应用的模型，建立计算框架来实现并完善语言模型，并根据语言模型设计各种实用系统及探讨这些系统的评测技术。', '\u200c\u200c']

In [85]:
```
 # You need to install it via: python -m spacy download zh_core_web_sm
from spacy.lang.zh.examples import sentences
nlp = spacy.load("zh_core_web_sm")
doc = nlp(sentences[0])
text = """\
字节对编码是一种简单的数据压缩形式，这种方法用数据中不存的一个字节表示最常出现的连续字节数据
Z <- aa 数据转变为 ZabdZabac。在这个数据中，字节对“Za”出现的次数最多，我们用另外一个字
Z <- aa, Y <- Za, YbdYbac。我们再次替换最常出现的字节对得到：Z <- aa, Y <- Za, X <
"""
doc = nlp(text)
sentences = [sent.text for sent in doc.sents]
for ind, sent in enumerate(sentences):
    print(f"sentence-{ind}: {sent}\n")
```

sentence-0: 字节对编码是一种简单的数据压缩形式，这种方法用数据中不存的一个字节表示最常出现的连续字节数据。

sentence-1: 这样的替换需要重建全部原始数据。

sentence-2: 字节对编码实例: 假设我们要编码数据 aaabdaaabac, 字节对“aa”出现次数最多，所以我们用数据中没有出现的字节“Z”替换“aa”得到替换表
Z <- aa 数据转变为 ZabdZabac.

sentence-3: 在这个数据中，字节对“Za”出现的次数最多，我们用另外一个字节“Y”来替换它（这种情况下由于所有的“Z”都将被替换，所以也可以用“Z”来替换“Za”），得到替换表以及数据
Z <- aa, Y <- Za, YbdYbac.

sentence-4: 我们再次替换最常出现的字节对得到: Z <- aa, Y <- Za, X <-

sentence-5: Yb.

sentence-6: XdXac 由于不再有重复出现的字节对，所以这个数据不能再被进一步压缩。

sentence-7: 解压的时候，就是按照相反的顺序执行替换过程。

- **spaCy handles special tokens**

In [74]:
```
# spacy works much better

nlp = spacy.load('en_core_web_sm')
text = """Special characters and numbers will need to be kept in prices ($45
we don't want to segment that price into separate tokens of "45" and "55". A
Twitter hashtags (#nlproc), or email addresses (someone@cs.colorado.edu)."""
doc = nlp(text)
print([token for token in doc])
```

```
[Special, characters, and, numbers, will, need, to, be, kept, in, prices, (,
$, 45.55, ), and, dates, (, 01/02/06, ), ;,
, we, do, n't, want, to, segment, that, price, into, separate, tokens, of,
", 45, ", and, ", 55, ", ., And, there, are, URLs, (, https://www.stanford.e
du, ), ,,
, Twitter, hashtags, (, #, nlproc, ), ,, or, email, addresses, (, someone@c
s.colorado.edu, ), .]
```

In [75]:
```
prompt = "Fudan University is located in Shanghai"
nlp = spacy.load("en_core_web_sm")
doc = nlp(prompt) # i want to do text preprocessing for our prompt.

# Text: The original word text.
# Lemma: The base form of the word.
# POS: The simple UPOS part-of-speech tag.
# Tag: The detailed part-of-speech tag.
# Dep: Syntactic dependency, i.e. the relation between tokens.
# Shape: The word shape — capitalization, punctuation, digits.
# is alpha: Is the token an alpha character? (whether it consists only of le
# is stop: Is the token part of a stop list, i.e. the most common words of t
#         (A stop list (or stopwords list) is a list of commonly used words
```

```python
#         are usually ignored during natural language processing (NLP) tasks

for token in doc:
    print(f"--- token: {token.text} ---")
    print(f"lemma: {token.lemma_}\npos: {token.pos_}\ntag: {token.tag_}\ndep
```

```
--- token: Fudan ---
lemma: Fudan
pos: PROPN
tag: NNP
dep: compound
shape: Xxxxx
is_alpha:True
is_stop: False
--- token: University ---
lemma: University
pos: PROPN
tag: NNP
dep: nsubjpass
shape: Xxxxx
is_alpha:True
is_stop: False
--- token: is ---
lemma: be
pos: AUX
tag: VBZ
dep: auxpass
shape: xx
is_alpha:True
is_stop: True
--- token: located ---
lemma: locate
pos: VERB
tag: VBN
dep: ROOT
shape: xxxx
is_alpha:True
is_stop: False
--- token: in ---
lemma: in
pos: ADP
tag: IN
dep: prep
shape: xx
is_alpha:True
is_stop: True
--- token: Shanghai ---
lemma: Shanghai
pos: PROPN
tag: NNP
dep: pobj
shape: Xxxxx
is_alpha:True
is_stop: False
```

- **Lemmatization (词形还原)** Lemmatization is the task of determining that two words have the same root, despite their surface differences. For some NLP situations, we also want two morphologically different forms of a word to behave similarly. For example in web search, someone may type the string woodchucks but a useful system might want to also return pages that mention woodchuck with no s.
  - **Example 1**: The words **am**, are, and is have the shared lemma **be**.
  - **Example 2**: The words **dinner** and **dinners** both have the lemma **dinner** .

In [81]:
```
text = """
The Brown Corpus, a text corpus of American English that was compiled in the
is widely used in the field of linguistics and natural language processing.
words (or "tokens") across a diverse range of texts from 500 sources, categc
as news, editorial, and fiction, to provide a comprehensive resource for stu
This corpus has been instrumental in the development and evaluation of varic
algorithms and tools.
"""
text = text.replace("\n", " ").strip()
nlp = spacy.load('en_core_web_sm')
doc = nlp(text)
lemmas = [token.lemma_ for token in doc]
for ori,lemma in zip(doc[:30], lemmas[:30]):
    print(ori, lemma)
```

```
The the
Brown Brown
Corpus Corpus
, ,
a a
text text
corpus corpus
of of
American American
English English
that that
was be
compiled compile
in in
the the
1960s 1960
at at
Brown Brown
University University
, ,
is be
widely widely
used use
in in
the the
field field
of of
linguistics linguistic
and and
natural natural
```

- **Stemming (词干提取).** The Porter-Stemmer method

  Lemmatization algorithms can be complex. For this reason we sometimes make use of a simpler but cruder method, which mainly consists of chopping off words final affixes. This naive version of morphological analysis is called stemming.

In [82]:
```python
import spacy
nlp = spacy.load("en_core_web_sm")

text = """\
This was not the map we found in Billy Bones's chest, but \
an accurate copy, complete in all things-names and heights \
and soundings-with the single exception of the red crosses \
and the written notes.\
"""
doc = nlp(text)

for tok in doc:
    if tok.is_alpha:
        print(tok.text, tok.lemma_)
```

```
This this
was be
not not
the the
map map
we we
found find
in in
Billy Billy
Bones Bones
chest chest
but but
an an
accurate accurate
copy copy
complete complete
in in
all all
things thing
names name
and and
heights height
and and
soundings sounding
with with
the the
single single
exception exception
of of
the the
red red
crosses cross
and and
the the
written write
notes note
```

- **Sentence tokenization**

In [86]:
```python
# A modern and fast NLP library that includes support for sentence segmentat
# spaCy uses a statistical model to predict sentence boundaries, which can b
# than rule-based approaches for complex texts.
# Install via conda: conda install conda-forge::spacy
# Install via pip:   pip install -U spacy
# Download data: python -m spacy download en_core_web_sm
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Here is a sentence. Here is another one! And the last one.")
sentences = [sent.text for sent in doc.sents]
for ind, sent in enumerate(sentences):
    print(f"sentence-{ind}: {sent}\n")
```

```
sentence-0: Here is a sentence.

sentence-1: Here is another one!

sentence-2: And the last one.
```

3. Datasets Exploration

Please check this part in our slides.

3.1 Explore wikitext-2 dataset

- **Please note that huggingface cannot be directly used in China. An alternative way is to use https://hf-mirror.com/.**

In [9]:
```python
import os
# If this does not work, please add export HF_ENDPOINT=https://hf-mirror.com
os.environ["HF_ENDPOINT"] = "https://hf-mirror.com"
os.environ["HF_HUB_ETAG_TIMEOUT"] = "60"
os.environ["HF_HUB_DOWNLOAD_TIMEOUT"] = "60"

from datasets import load_dataset

# loads train/validation/test
ds = load_dataset("wikitext", "wikitext-2-raw-v1")
```

In [10]:
```python
print(ds)
train = ds["train"]
for i in range(4):
    print(train[i])
```

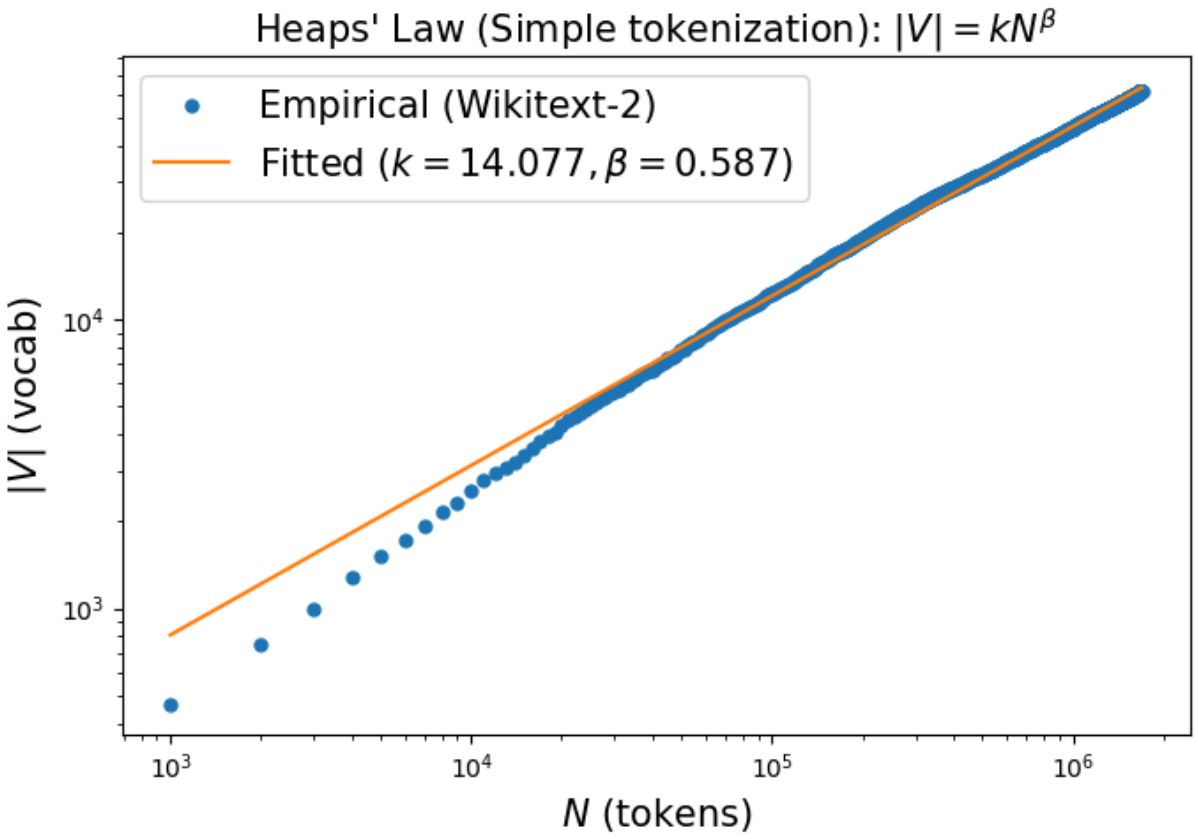
```
DatasetDict({
    test: Dataset({
        features: ['text'],
        num_rows: 4358
    })
    train: Dataset({
        features: ['text'],
        num_rows: 36718
    })
    validation: Dataset({
        features: ['text'],
        num_rows: 3760
    })
})
{'text': ''}
{'text': ' = Valkyria Chronicles III = \n'}
{'text': ''}
{'text': ' Senjō no Valkyria 3 : Unrecorded Chronicles ( Japanese : 戦場のヴァルキュリア3 , lit . Valkyria of the Battlefield 3 ) , commonly referred to as Valkyria Chronicles III outside Japan , is a tactical role @-@ playing video game developed by Sega and Media.Vision for the PlayStation Portable . Released in January 2011 in Japan , it is the third game in the Valkyria series . Employing the same fusion of tactical and real @-@ time gameplay as its predecessors , the story runs parallel to the first game and follows the " Nameless " , a penal military unit serving the nation of Gallia during the Second Europan War who perform secret black operations and are pitted against the Imperial unit " Calamaty Raven " . \n'}
```

- We can use a simple tokenization and testing the Heap's law.

In [11]:
```python
import re
import numpy as np
import matplotlib.pyplot as plt

train = ds["train"]

# simple word tokenizer (lowercased)
word_re = re.compile(r"[A-Za-z]+(?:'[A-Za-z]+)?")

def heaps_curve(dataset, step=1000):
    V = set()
    N = 0
    Ns, Vs = [], []

    for ex in dataset:
        text = ex["text"]
        if not text or text.strip() == "": # empty word -> continue
            continue
        # make all words to low cases
        words = word_re.findall(text.lower())
        for w in words:
            N += 1
            V.add(w)

            if N % step == 0:
```

```
                Ns.append(N)
                Vs.append(len(V))
    return np.array(Ns), np.array(Vs)

Ns, Vs = heaps_curve(train, step=1000)

# Fit log |V| = log k + beta log N  -> linear regression on logs
logN = np.log(Ns)
logV = np.log(Vs)
beta, logk = np.polyfit(logN, logV, 1)
k = np.exp(logk)

print(f"Fitted Heaps' law: |V| ≈ {k:.2f} * N^{beta:.3f}")

# Plot (log-log)
plt.figure(figsize=(7, 5))
plt.loglog(Ns, Vs, marker='o', linestyle='none', markersize=5, label="Empiri

# fitted line
Ns_line = np.linspace(Ns[0], Ns[-1], 200)
Vs_line = k * (Ns_line ** beta)
plt.loglog(Ns_line, Vs_line, label=fr"Fitted ($k={k:.3f},\beta=${beta:.3f})"

plt.xlabel("$N$ (tokens)", fontsize = 15)
plt.ylabel("$|V|$ (vocab)", fontsize = 15)
plt.title(r"Heaps' Law (Simple tokenization): $|V|=kN^{\beta}$", fontsize =
plt.legend(fontsize = 15)
plt.tight_layout()
plt.show()
```

```
Fitted Heaps' law: |V| ≈ 14.08 * N^0.587
```



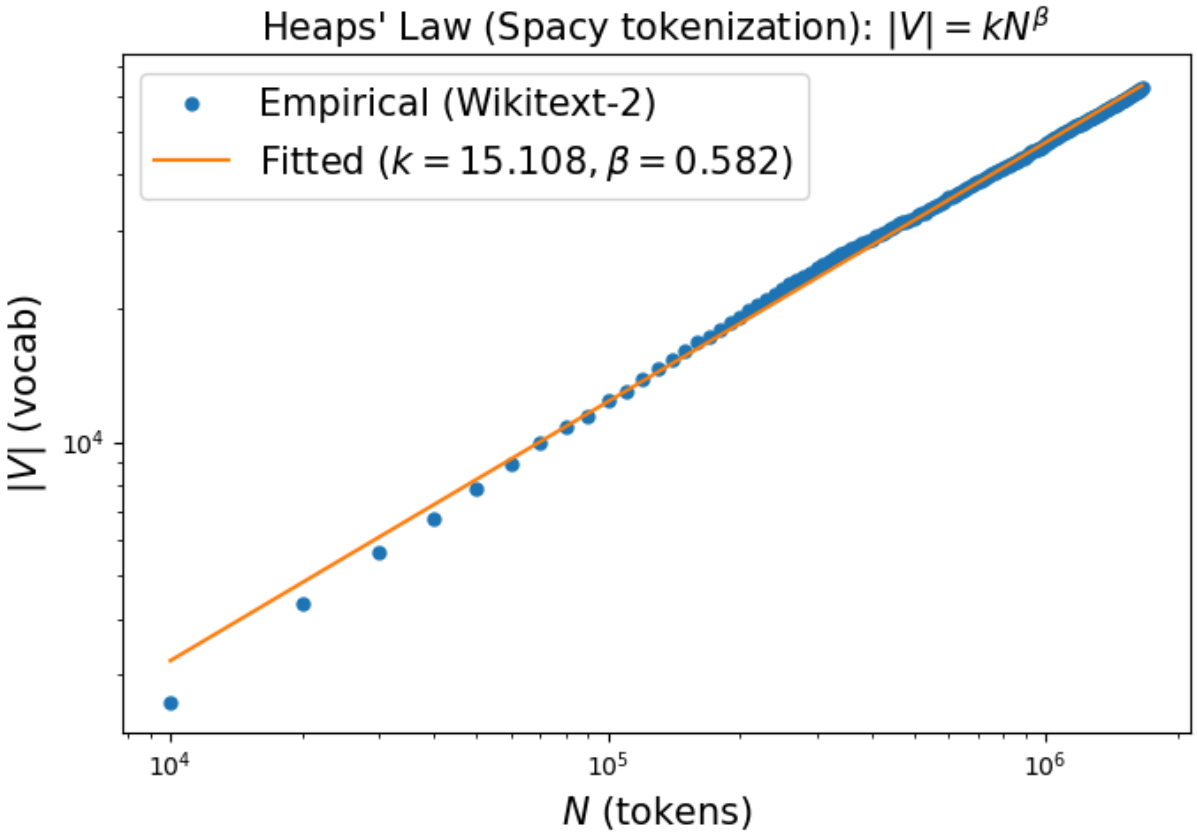
- Next, we make the tokenization a little bit different, we can use spacy to do the tokenization.

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
import spacy

nlp = spacy.load("en_core_web_sm", disable=["tagger","parser","ner","lemmati
# tokenizer still works even with pipeline disabled

train = ds["train"]

def heaps_curve_spacy(dataset, step=10_000, batch_size=256):
    V = set()
    N = 0
    Ns, Vs = [], []

    texts = (ex["text"] for ex in dataset if ex["text"] and ex["text"].strip
    for doc in nlp.pipe(texts, batch_size=batch_size):
        for tok in doc:
            # choose your definition of "word"
            if tok.is_alpha:
                w = tok.text.lower()
                N += 1
                V.add(w)
                if N % step == 0:
                    Ns.append(N)
                    Vs.append(len(V))
```

```
    return np.array(Ns), np.array(Vs)

Ns, Vs = heaps_curve_spacy(train, step=10_000)

# Fit log |V| = log k + beta log N
logN = np.log(Ns)
logV = np.log(Vs)
beta, logk = np.polyfit(logN, logV, 1)
k = np.exp(logk)

print(f"Fitted Heaps' law (spaCy): |V| ≈ {k:.2f} * N^{beta:.3f}")
# Plot (log-log)
plt.figure(figsize=(7, 5))
plt.loglog(Ns, Vs, marker='o', linestyle='none', markersize=5, label="Empiri

# fitted line
Ns_line = np.linspace(Ns[0], Ns[-1], 200)
Vs_line = k * (Ns_line ** beta)
plt.loglog(Ns_line, Vs_line, label=fr"Fitted ($k={k:.3f},\beta=${beta:.3f})"

plt.xlabel("$N$ (tokens)", fontsize = 15)
plt.ylabel("$|V|$ (vocab)", fontsize = 15)
plt.title(r"Heaps' Law (Spacy tokenization): $|V|=kN^{\beta}$", fontsize = 1
plt.legend(fontsize = 15)
plt.tight_layout()
plt.show()
```

```
Fitted Heaps' law (spaCy): |V| ≈ 15.11 * N^0.582
```



3.2 Explore wikitext-103 dataset

We can download a larger dataset from the following:

```
export HF_HOME=$HOME/.cache/huggingface

huggingface-cli download Salesforce/wikitext --repo-type dataset --
resume-download --include "wikitext-103-raw-v1/*.parquet"
```

In [16]:
```
from datasets import load_dataset
ds = load_dataset("wikitext", "wikitext-103-raw-v1")  # will hit cache if pr
```

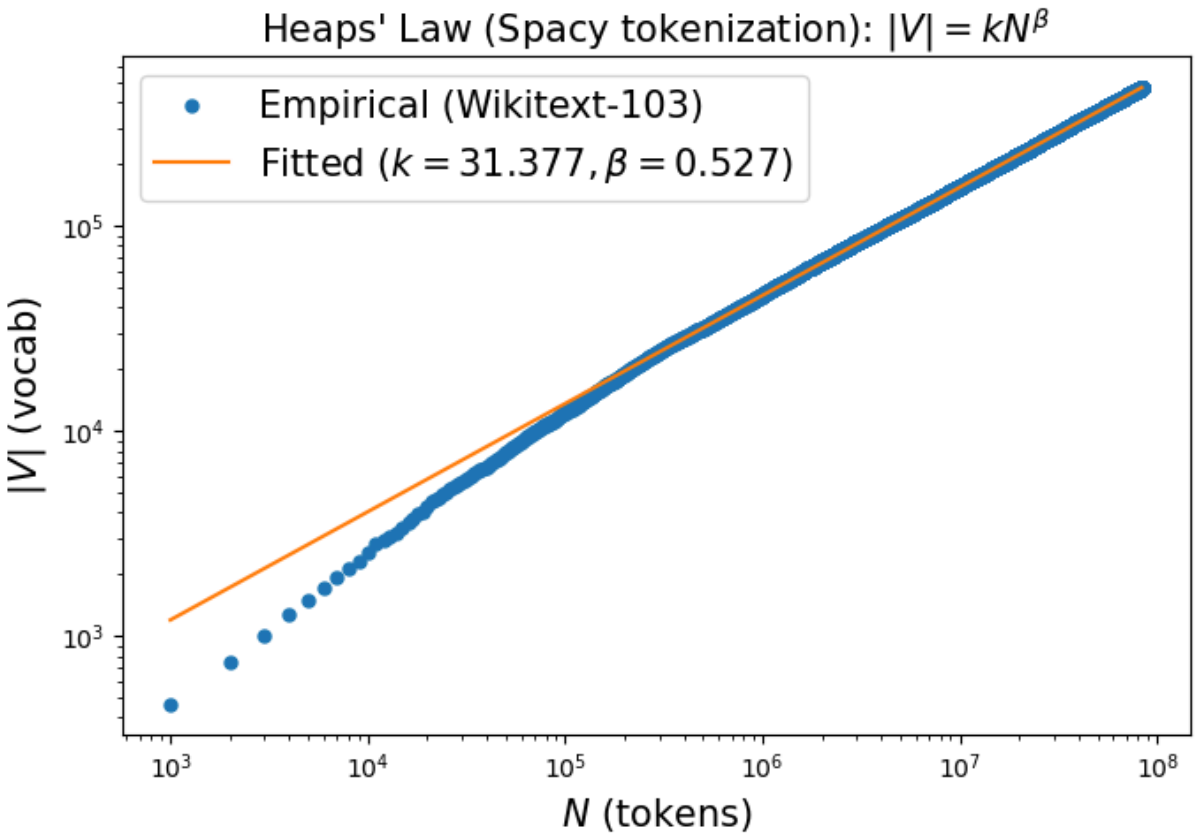
In [17]:
```
print(ds)
train = ds["train"]
for i in range(4):
    print(train[i])
```

```
DatasetDict({
    test: Dataset({
        features: ['text'],
        num_rows: 4358
    })
    train: Dataset({
        features: ['text'],
        num_rows: 1801350
    })
    validation: Dataset({
        features: ['text'],
        num_rows: 3760
    })
})
{'text': ''}
{'text': ' = Valkyria Chronicles III = \n'}
{'text': ''}
{'text': ' Senjō no Valkyria 3 : Unrecorded Chronicles ( Japanese : 戦場のヴァルキュリア3 , lit . Valkyria of the Battlefield 3 ) , commonly referred to as Valkyria Chronicles III outside Japan , is a tactical role @-@ playing video game developed by Sega and Media.Vision for the PlayStation Portable . Released in January 2011 in Japan , it is the third game in the Valkyria series . Employing the same fusion of tactical and real @-@ time gameplay as its predecessors , the story runs parallel to the first game and follows the " Nameless " , a penal military unit serving the nation of Gallia during the Second Europan War who perform secret black operations and are pitted against the Imperial unit " Calamaty Raven " . \n'}
```

In [18]:
```
train = ds["train"]
# simple word tokenizer (lowercased)
word_re = re.compile(r"[A-Za-z]+(?:'[A-Za-z]+)?")

def heaps_curve(dataset, step=1000):
    V = set()
    N = 0
    Ns, Vs = [], []
```

```
    for ex in dataset:
        text = ex["text"]
        if not text or text.strip() == "": # empty word -> continue
            continue
        # make all words to low cases
        words = word_re.findall(text.lower())
        for w in words:
            N += 1
            V.add(w)

            if N % step == 0:
                Ns.append(N)
                Vs.append(len(V))
    return np.array(Ns), np.array(Vs)

Ns, Vs = heaps_curve(train, step=1000)

# Fit log |V| = log k + beta log N  -> linear regression on logs
logN = np.log(Ns)
logV = np.log(Vs)
beta, logk = np.polyfit(logN, logV, 1)
k = np.exp(logk)

print(f"Fitted Heaps' law (spaCy): |V| ≈ {k:.2f} * N^{beta:.3f}")
# Plot (log-log)
plt.figure(figsize=(7, 5))
plt.loglog(Ns, Vs, marker='o', linestyle='none', markersize=5, label="Empiri

# fitted line
Ns_line = np.linspace(Ns[0], Ns[-1], 200)
Vs_line = k * (Ns_line ** beta)
plt.loglog(Ns_line, Vs_line, label=fr"Fitted ($k={k:.3f},\beta=${beta:.3f})"

plt.xlabel("$N$ (tokens)", fontsize = 15)
plt.ylabel("$|V|$ (vocab)", fontsize = 15)
plt.title(r"Heaps' Law (Spacy tokenization): $|V|=kN^{\beta}$", fontsize = 1
plt.legend(fontsize = 15)
plt.tight_layout()
plt.show()
```

```
Fitted Heaps' law (spaCy): |V| ≈ 31.38 * N^0.527
```



3.3 BookCorpus datasets

BookCorpus dataset has been used in training GPT-1. See GPT-1 paper (PDF)

In [ ]:
```
import time
import tarfile

path = "../.datasets/books1.tar.gz"
start_time = time.time()
with tarfile.open(path, "r:gz") as tar:
    names = tar.getnames()
    print("num members:", len(names))
    print("\n".join(names[:50]))
print(f"total time to scan BookCorpus: {time.time() - start_time:.2f} seconc
```

You can check `shell tokenization_bookcorpus.py` to generate heaps statistics in jsonl file.

In [ ]:
```
import json
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

def load_heaps_jsonl(log_path: str):
    Ns, Vs = [], []
    with open(log_path, "r", encoding="utf-8") as f:
        for line in f:
```

```
            r = json.loads(line)
            Ns.append(r["total_tokens"])
            Vs.append(r["vocab_size"])
    return np.array(Ns), np.array(Vs)

Ns, Vs = load_heaps_jsonl("books1_wordcounts.heaps_log.jsonl")

# fit log |V| = log k + beta log N
beta, logk = np.polyfit(np.log(Ns), np.log(Vs), 1)
k = np.exp(logk)
print(f"Fit: |V| ≈ {k:.2f} * N^{beta:.3f}")


plt.figure(figsize=(7,5))

plt.loglog(Ns, Vs, marker="o", linestyle="none",
           markersize=5, label="Empirical (BookCorpus)")
Ns_line = np.linspace(Ns[0], Ns[-1], 200)
plt.loglog(Ns_line, k * (Ns_line ** beta), label=fr"Fitted Curve $(k={k:.3f}
plt.xlabel("$N$ (tokens)", fontsize = 14)
plt.ylabel("$|V|$ (vocab)", fontsize = 14)
plt.title(r"Heaps' Law (Simple tokenization): $|V|=k N^{\beta}$", fontsize =
plt.legend(fontsize = 14)
plt.tight_layout()
plt.show()
```

4. LLMs tokenization

There are three standard way to do modern LLM tokenization:

- **Subword tokenization:**
- **Sentence tokenization:**
- **Wordpiece tokenization:**

4.1 (Modern way) Subword tokenization: BPE

4.4 Huggingface tokenizer

PreTrainedTokenizer, PreTrainedTokenizerBase, AutoTokenizer

- https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_u
- https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_u
- https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_u
- Check tokenizers at https://github.com/huggingface/transformers/blob/main/setup.py
- If you want to train a tokenizer by yourself, then go to: https://github.com/huggingface/tokenizers
- A fast BPE tokenizer is also at: https://github.com/openai/tiktoken
- An implementation of sentencepiece is at: https://github.com/google/sentencepiece

- There are 3 most common methods for tokenization: https://github.com/huggingface/tokenizers/tree/main/tokenizers/src/models
- - BPE: https://aclanthology.org/P16-1162.pdf
  - Unigram: https://arxiv.org/pdf/1804.10959
  - WordPiece https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/

In [ ]:
```
from transformers import Qwen2Tokenizer, Qwen2TokenizerFast
```

5. Minimum Edit Distance

5.1 Algorithm analysis

Assessing the similarity between two strings is essential in many NLP tasks. For spell correction, one must choose the most similar word to a misspelled input from a set of candidates. For example, if `graffe"` is the input, among the options `graf`, `graft`, `grail`, and `giraffe`, the task is to select the word closest to the input based on a certain criterion. Similarly, comparing a translated sentence to a reference translation helps evaluate the performance of translation models in machine translation. This process may involve calculating the minimum edit distance to determine the similarity of two sentences. This note gives a dynamic programming algorithm for MED.

> **Definition (Minimum Edit Distance, MED).**
> Given two strings $X = [x_1, x_2, \ldots, x_n]$ and $Y = [y_1, y_2, \ldots, y_m]$, the minimum edit distance between X and Y is the minimum number of edit operations required to convert X into Y.
>
> **Allowed edit operations:**
>
> 1. **Insertion:** add a character to X.
> 2. **Deletion:** remove a character from X.
> 3. **Substitution:** replace a character in X with another character.
>
> These operations are applied sequentially from left to right to obtain Y.

> **Example.** Suppose $X = \texttt{INTENTION}$ and $Y = \texttt{EXECUTION}$. One possible process of transforming $X \to Y$ is:

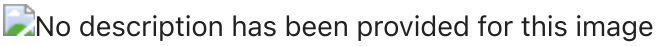
Step-by-step edits (one possible alignment):

1. $\texttt{INTENTION} \xrightarrow{\text{Del(I)}} \texttt{NTENTION}$

2. $\texttt{NTENTION} \xrightarrow{\text{Sub}(\texttt{N}\to\texttt{E})} \texttt{ETENTION}$
3. $\texttt{ETENTION} \xrightarrow{\text{Sub}(\texttt{T}\to\texttt{X})} \texttt{EXENTION}$
4. $\texttt{EXENTION} \xrightarrow{\text{Ins}(\texttt{C})} \texttt{EXECNTION}$
5. $\texttt{EXECNTION} \xrightarrow{\text{Sub}(\texttt{N}\to\texttt{U})} \texttt{EXECUTION} = Y$

We assume that **Ins** and **Del** each count as **1** operation, while **Sub** counts as **2** operations.
Consequently, the total number of edit operations above is $1 + 2 + 2 + 1 + 2 = 8$.

Let $O_k = [o_1, o_2, \ldots, o_k]$ denote the sequence of operations transforming X into Y, where each $o_i \in \{\text{Ins}, \text{Del}, \text{Sub}\}$.
We call O_k an alignment of X into Y (see the figure below).



- **Dynamic Programming for MED**

A naive approach would examine all possible edit sequences O_k and choose the one with the minimal cost. However, this method has **exponential** time complexity. Instead, we break the problem into smaller subproblems and exploit **optimal substructure**.

Problem setting. For $i = \{0, 1, \ldots, n\}$ and $j = \{0, 1, \ldots, m\}$, let

- $X_i = [x_1, x_2, \ldots, x_i]$ be the prefix of X of length i.
- $Y_j = [y_1, y_2, \ldots, y_j]$ be the prefix of Y of length j.

By default, $X_0 = \emptyset$ and $Y_0 = \emptyset$. Define $D[i, j]$ as the **minimum edit distance** from $X_i \to Y_j$. Clearly, $D[n, m]$ is the MED of $X \to Y$. There are two base cases:

- $D[i, 0] = i$ (delete all i characters from X_i).
- $D[0, j] = j$ (insert all j characters into $\emptyset$).

In the rest, we assume $i, j \geq 1$.

Optimal substructure. Let $O_k = [o_1, o_2, \ldots, o_k]$ be a minimum-cost sequence of operations that transforms $X_i \to Y_j$. Then

$$X_i = X_i^0 \xrightarrow{o_1} X_i^1 \xrightarrow{o_2} X_i^2 \to \cdots \xrightarrow{o_{k-1}} X_i^{k-1} \xrightarrow{o_k} X_i^k = Y_j.$$

If we remove the last operation o_k, the remaining sequence $O_{k-1} = [o_1, o_2, \ldots, o_{k-1}]$ must be an optimal process for $X_i \to X_i^{k-1}$.
Otherwise, suppose there exists another sequence O'_{k-1} that transforms $X_i \to X_i^{k-1}$ with strictly smaller cost: $\text{cost}(O'_{k-1}) < \text{cost}(O_{k-1})$. Then

> combining O'_{k-1} with the last operation o_k would yield a sequence for $X_i \to Y_j$ with total cost strictly smaller than $\text{cost}(O_k)$—a contradiction. This is the **optimal substructure** property.

Because operations proceed from the left of X_i to the right, there are only **three** possibilities for the last operation o_k:

- **Case 1 (Deletion).** Delete x_i from X_i.

$$D[i, j] = D[i-1, j] + \text{Del}(x_i).$$

- **Case 2 (Insertion).** Insert y_j into Y_{j-1}.

$$D[i, j] = D[i, j-1] + \text{Ins}(y_j).$$

- **Case 3 (Substitution).** Substitute x_i by y_j.

$$D[i, j] = D[i-1, j-1] + \text{Sub}(x_i, y_j).$$

Therefore, MED for $X_i \to Y_j$ can be computed from the three subproblems $D[i-1, j]$, $D[i, j-1]$, and $D[i-1, j-1]$ by taking the minimum:

$$D[i, j] = \min \begin{cases} D[i-1, j] + \text{Del}(x_i), \\ D[i, j-1] + \text{Ins}(y_j), \\ D[i-1, j-1] + \text{Sub}(x_i, y_j). \end{cases}$$

This recurrence is the core of **dynamic programming** for MED. In the example above, we can compute the DP matrix $\mathbf{D}$ using this recurrence, as shown in the following tables. The above formula is called dynamic procedure or dynamic programming. Given the above-mentioned example, we can calculate matrix $\mathbf{D}$ using this dynamic procedure as shown in the following tables.

- **DP Table: Initialization vs. Final Result**

Initial stage: $D[\cdot, \cdot]$:

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Final stage: $D[n, m] = 8$:

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

> **Remark.** We call the above method a **dynamic programming** method.
>
> 1. Try to prove or disprove the **uniqueness** of the optimal sequence of operations.
> 2. Can you find approximate solutions if n and m are large? (Hint: search for approximate string matching.)

5.2 Algorithm implementation

In [91]:
```python
import numpy as np

# define minimum edit distance algorithm via dynamic programming
def minimum_edit_distance(source, target):
    n = len(source)
    m = len(target)
    d_mat = np.zeros((n + 1, m + 1))
```

```
    for i in range(1, n + 1):
        d_mat[i, 0] = i
    for j in range(1, m + 1):
        d_mat[0, j] = j
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            sub = 0 if source[i - 1] == target[j - 1] else 2
            del_ = d_mat[i - 1][j] + 1
            ins_ = d_mat[i][j - 1] + 1
            d_mat[i][j] = min(del_, ins_, d_mat[i - 1][j - 1] + sub)
    trace, align_source, align_target = backtrack_alignment(source, target,
    return d_mat[n, m], trace, align_source, align_target

def backtrack_alignment(source, target, d_mat):
    align_source, align_target = [], []
    i, j = len(source), len(target)
    back_trace = [[i, j]]

    while (i, j) != (0, 0):
        # boundary cases first (avoid negative indexing)
        if i == 0:
            back_trace.append([i, j - 1])
            align_source = ["*"] + align_source
            align_target = [target[j - 1]] + align_target
            j -= 1
            continue
        if j == 0:
            back_trace.append([i - 1, j])
            align_source = [source[i - 1]] + align_source
            align_target = ["*"] + align_target
            i -= 1
            continue

        sub_cost = 0 if source[i - 1] == target[j - 1] else 2

        # prefer substitution/match when optimal (your tie-break rule)
        if d_mat[i, j] == d_mat[i - 1, j - 1] + sub_cost:
            back_trace.append([i - 1, j - 1])
            align_source = [source[i - 1]] + align_source
            align_target = [target[j - 1]] + align_target
            i, j = i - 1, j - 1

        # deletion
        elif d_mat[i, j] == d_mat[i - 1, j] + 1:
            back_trace.append([i - 1, j])
            align_source = [source[i - 1]] + align_source
            align_target = ["*"] + align_target
            i -= 1

        # insertion
        else:
            back_trace.append([i, j - 1])
            align_source = ["*"] + align_source
            align_target = [target[j - 1]] + align_target
            j -= 1
```

```
    return back_trace, align_source, align_target

# test the minimum edit distance
def test_med(source, target):
    med, trace, align_source, align_target = minimum_edit_distance(source, t
    print(f"input source: {source} and target: {target}")
    print(f"med: {med}")
    print(f"trace: {trace}")
    print(f"aligned source: {align_source}")
    print(f"aligned target: {align_target}")
```

In [92]:
```
test_med(source="INTENTION", target="EXECUTION")
```

```
input source: INTENTION and target: EXECUTION
med: 8.0
trace: [[9, 9], [8, 8], [7, 7], [6, 6], [5, 5], [4, 4], [4, 3], [3, 2], [2,
1], [1, 0], [0, 0]]
aligned source: ['I', 'N', 'T', 'E', '*', 'N', 'T', 'I', 'O', 'N']
aligned target: ['*', 'E', 'X', 'E', 'C', 'U', 'T', 'I', 'O', 'N']
```

In [93]:
```
test_med(source="AGGCTATCACCTGACCTCCAGGCCGATGCCC", target="TAGCTATCACGACCGCG
```

```
input source: AGGCTATCACCTGACCTCCAGGCCGATGCCC and target: TAGCTATCACGACCGCGG
TCGATTTGCCCGAC
med: 15.0
trace: [[31, 32], [30, 31], [30, 30], [30, 29], [29, 28], [28, 27], [28, 2
6], [27, 25], [26, 24], [26, 23], [26, 22], [25, 21], [24, 20], [23, 19], [2
2, 18], [21, 17], [20, 16], [19, 16], [18, 15], [17, 14], [16, 14], [15, 1
3], [14, 12], [13, 11], [12, 10], [11, 10], [10, 9], [9, 9], [8, 8], [7, 7],
[6, 6], [5, 5], [4, 4], [3, 3], [2, 2], [1, 2], [0, 1], [0, 0]]
aligned source: ['*', 'A', 'G', 'G', 'C', 'T', 'A', 'T', 'C', 'A', 'C', 'C',
'T', 'G', 'A', 'C', 'C', 'T', 'C', 'C', 'A', 'G', 'G', 'C', 'C', 'G', 'A',
'*', '*', 'T', 'G', '*', 'C', 'C', '*', '*', 'C']
aligned target: ['T', 'A', '*', 'G', 'C', 'T', 'A', 'T', 'C', 'A', '*', 'C',
'*', 'G', 'A', 'C', 'C', '*', 'G', 'C', '*', 'G', 'G', 'T', 'C', 'G', 'A',
'T', 'T', 'T', 'G', 'C', 'C', 'C', 'G', 'A', 'C']
```