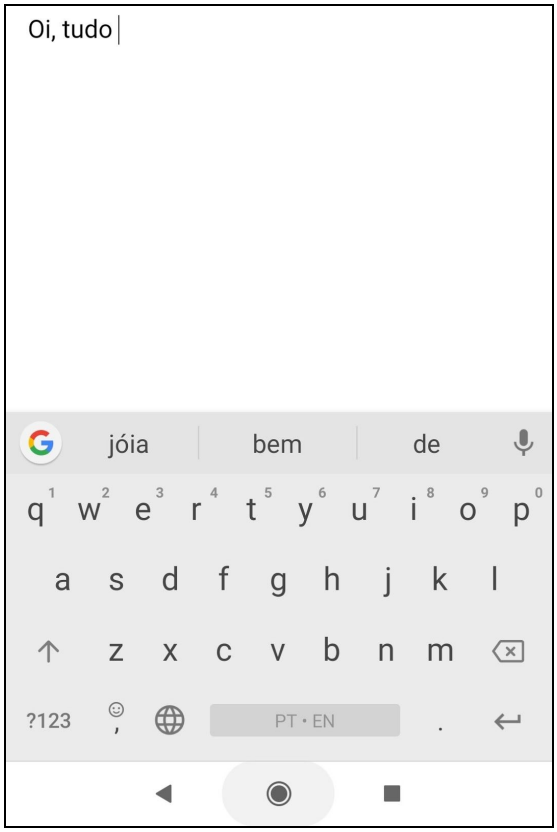
→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Classe MyMap.h:
https://drive.google.com/file/d/1ZhuStQCB7hBVStyeNM79cxhlosJCa52n/view?usp=sharing

Trabalho 3 - Predição de palavras



Uma técnica simples e bastante utilizada para prever qual será a próxima palavra a ser digitada em um texto é baseada no conceito de cadeias de Markov: a ideia consiste em se calcular estatísticas a partir de algum texto (vamos chamá-lo de *texto de treino* neste trabalho) e, com isso, determinar qual palavra é mais provável de aparecer após uma outra.

Por exemplo, se for determinado que após a palavra “eu” a palavra “estou” aparece 70% das vezes e “vou” aparecer 20% das vezes, então após alguém digitar “eu” seria interessante sugerir as palavras “estou” e “vou”.

O texto de treino pode ser um conjunto de textos na língua utilizada pelo usuário. Uma alternativa interessante consiste em atualizar as estatísticas com base no que o próprio usuário digita (assim, o teclado irá “aprender” com o usuário e, com o tempo, espera-se que as predições fiquem cada vez melhores).

Note que a sugestão de palavras também pode ser feita com base na palavra anterior e nas primeiras letras da palavra atual. Essa estratégia pode ser implementada de forma eficiente utilizando, por exemplo, árvores Trie (mas isso não será tratado neste trabalho).

Neste trabalho vamos desenvolver um sistema capaz de fazer sugestões de palavras em um texto. Além disso, vamos usar essas sugestões para tentar gerar pequenas frases de forma automática. Sua implementação **deverá utilizar** "dicionários" **MyMap** para processar os dados de forma eficiente.

Há vários detalhes importantes nessa especificação (pois há muitas formas diferentes de se implementar e muitos casos especiais) e, dessa forma, sugiro que a leia com bastante atenção (sugiro imprimir esta especificação e marcar as partes mais críticas com caneta vermelha). Apesar da especificação grande, o trabalho e' mais simples do que aparenta (foi muito mais trabalhoso escrever esta especificação do que implementar o trabalho).

Fazendo as predições

Considere inicialmente que a entrada seja um conjunto de várias sentenças escritas em algum idioma. Por simplicidade, não vamos considerar sinais de pontuação (eles serão considerados final de sentenca), acentuação e nem letras maiúsculas (seu programa deverá converter todas letras maiúsculas em minúsculas ao ler a entrada).

Para fazer as predições, vamos criar 3 maps considerando 3 "niveis" de predicao.

Considere o seguinte texto de exemplo (considere que cada sentença termine com um ponto final):
"Oi como voce esta.
voce vai viajar.
voce vai tambem.
como voce vai.". (note que o texto original provavelmente teria pontuação, mas isso foi retirado)

No nível 1, teremos um dicionário (vamos chamá-lo de map1) que contém a frequência de cada palavra no texto de treino. Se esse dicionario fosse criado para o exemplo anterior, teríamos o seguinte resultado (a coluna da esquerda representa a palavra e a da direita a frequência):

como	2
esta	1
oi	1
tambem	1

vai	3
viajar	1
voce	4

Ou seja, a palavra “como” aparece 2 vezes no texto, “vai” aparece 3 vezes, “voce” aparece 4 vezes.

Um dicionario com nivel 1 poderia ser usado para sugerir a primeira palavra de um texto. Por exemplo, com base no texto acima poderíamos sugerir “voce” como primeira palavra para um texto (pois e’ a que tem maior frequência).

No dicionário de nível 2 (chamado de map2), teremos a frequência com que, dada uma palavra X, cada palavra Y aparece após X. Para o exemplo acima, o conteúdo do dicionário de nível 2 será:

como	voce	2
oi	como	1
vai	tambem	1
	viajar	1
voce	esta	1
	vai	3

Nesse exemplo, após a palavra “voce” temos a palavra “esta” (com frequencia 1) e “vai” (com frequencia 3). Dessa forma em um texto com a palavra “como” deveríamos sugerir “voce” como próxima palavra. Em um texto com a palavra “você” deveríamos sugerir “vai” como próxima palavra.

Esse dicionario poderia ser implementado de forma similar a map1: poderiamos armazenar como chave de um map strings contendo 2 palavras. Por exemplo, a entrada “voce vai” indicaria a frequência com que a palavra vai aparece após a palavra você. Porém, essa estratégia não seria eficiente (por exemplo, ao armazenar “voce esta” e “voce vai” estariamos armazenando a palavra “você” duas vezes). Alem disso, para listar quais palavras aparecem após “você” teriamos que percorrer o dicionario inteiro procurando chaves que comecam com a string “você’.

Dessa forma, o dicionario de nivel 2 deverá ser representado usando um map de map. No exemplo acima, a chave “vai” seria mapeada em um segundo map contendo: “tambem” e “viajar” como chaves (ambos com o número 1 como valor associado a tais chaves).

Notem que uma estratégia melhor para sugerir a primeira palavra seria também contar no dicionário de nível 2 quais palavras aparecem após a string vazia no texto (ou seja, com qual frequência cada palavra aparece como primeira palavra de cada sentença). No exemplo acima, poderíamos ter uma linha onde a primeira coluna contém a string vazia (essa linha representaria a frequência com que as palavras aparecem após a string vazia, ou seja, são as primeiras palavras das sentenças). Porém, neste trabalho você não deverá fazer isso (deverá utilizar o dicionário de nível 1 nessa situação).

Por fim, no dicionario nivel 3 (chamado aqui de map3), teremos a frequência com que, dadas duas palavras X e Y, cada palavra Z aparece após X e Y. Para o exemplo acima, o conteúdo do dicionário de nível 3 será:

como	voce	esta	1
		vai	1
oi	como	voce	1
voce	vai	tambem	1
		viajar	1

Nesse exemplo, o map3 mapeia “como” em um mapa contendo apenas “voce” como chave (pois apenas essa palavra aparece após “como”). Esse mapa, por sua vez, mapeia “voce” em um mapa contendo duas chaves: esta (que, entao, e’ mapeado em 1) e “vai” (tambem mapeado em 1).

Intuitivamente, quanto mais palavras utilizamos para prever uma nova palavra, mais preciso e’ o resultado. Por exemplo suponha que alguém tenha treinado o sistema com a entrada acima e que, a seguir, tenha digitado “como você”. Se for usado um dicionario de nivel 0 para prever a proxima palavra, simplesmente pegariamos a palavra mais frequente do texto do treino (“voce”) e, assim, a sugestao seria “como você você”. Se for usado um dicionario de nivel 1, olhariamos a ultima palavra da frase (“você”) e sugeririamos “como voce vai” (“vai” aparece 3 vezes após “você”). Por outro lado, se for usado o dicionario de nivel 3, analisariamos uma janela com a palavra “como” seguida por “você”, o que daria como resultado “como você vai” e “como você está” (“vai” e “está” ficaram empatadas com frequência 1 no exemplo). Em situações que houver empate a palavra que for lexicograficamente menor deverá ser escolhida (ou seja, entre “esta” e “vai” a palavra escolhida seria “esta”).

Neste trabalho você deverá prever qual a proxima palavra de um texto **usando dicionarios do nível máximo possivel (3, 2 ou 1) (e' muito importante você seguir isso para que seu programa passe nos casos de teste)**.

Note que nem sempre é possivel usar o dicionario de maior nível. Por exemplo, se o usuário ainda não tiver digitado nada o sistema deverá sugerir a palavra mais frequente de map1. Se o texto a ser completado tiver apenas 1 palavra, então não poderemos utilizar map3. Por fim, pode ser que as palavras da janela utilizada não tenham aparecido no texto de treino. Por exemplo, se tentarmos completar "capivara vai" (considerando o texto de treino acima) não podemos utilizar map3 (pois não ha uma entrada para esses termos) e, assim, seremos obrigados a usar map2 (que possui uma entrada para "vai"). De forma similar, se tentarmos completar "vai capivara" teremos que usar map1 (pois em map2 não há uma entrada para "capivara").

Texto de treino

Você deverá pré-processar o texto de treino fornecido como entrada. Para garantir que os resultados obtidos em seu programa sejam iguais aos esperados pelo Submitty, as regras apresentadas a seguir devem ser seguidas.

Por simplicidade, assuma que não teremos acentuação em tal texto. Letras maiúsculas podem aparecer, porém, você deverá convertê-las para minúsculas (ou seja, após carregar o texto de treino seu programa deve trabalhar apenas com letras minúsculas).

As palavras do texto são separadas por um ou mais espaço em branco, tabulação ou newlines. Você deverá considerar que os caracteres ", ' e \n (aspas, apóstrofo e newline) são equivalentes a espaço em branco (ou seja, servem para separar as palavras). Dessa forma, o texto *I'm at \n "UFV"* seria considerado como uma sentença composta pela palavra "I", seguida de "m", seguida de "at" e seguida de "UFV".

As palavras serão compostas apenas por letras ou pelo hífen ("-"). Assim, guarda-chuva e' considerado uma palavra.

Qualquer outro caractere (sinais de pontuação e inclusive números) será considerado um separador de sentenças.

Por exemplo considere o seguinte texto de treino:

```
Esqueci meu guarda-chuva aqui. I'm at "UFV" and I
am 20 years old.

Espero que nao chova hoje! nao posso me molhar
```

Após sua leitura ele seria interpretado como sendo composto pelas seguintes 5 sentenças (as palavras estão separadas por vírgula):
esqueci,meu,guarda-chuva,aqui
i,m,at,ufv,and,i,am
years,old
espero,que,nao,chova,hoje
nao,posso,me,molhar

Assim, no dicionário de nível 2 teremos frequência 1 na ocorrência de “meu” após “esqueci”. Apesar de “i” e “am” estarem em linhas diferentes, consideramos que “am” aparece após “i” (pois não há nenhum sinal de pontuação os separando). Além disso, a palavra “nao” nunca aparece após a palavra “hoje” (na entrada ela aparece, mas desconsideramos isso porque elas estão separadas por um sinal de pontuação).

Por motivos de organização e para evitar erros, e’ extremamente aconselhável que a leitura da entrada e o pré-processamento sejam separados do resto do programa. Tente deixar seu código bem organizado. Ou seja, ao criar os mapas/dicionários você já deverá possuir a entrada pre-processada (por exemplo, sem letras maiusculas) e armazenada em um formato simples.

Passagem do texto de treino

O arquivo de treino poderá ser passado para o seu programa de duas formas: como primeiro (e único) argumento de linha de comando ou como parte da entrada padrão. Os comandos que seu programa deverá aceitar, por sua vez, serão sempre passados a partir da entrada padrão.

Se o treino for passado como parte da entrada padrão, a primeira linha da entrada deverá conter “COMECO_TREINO” (sem aspas). Tudo que estiver entre essa linha e uma linha contendo apenas “FINAL_TREINO” será considerado o texto de treino. Após a parte referente ao treino, uma série de comandos serão passados para seu programa.

Veja um exemplo de entrada abaixo (após o final do treino temos 2 comandos):

```
COMECO_TREINO
Esqueci meu guarda-chuva aqui. I'm at "UFV" and
I am 20 years old.

Espero que nao chova hoje! nao posso me molhar
FINAL_TREINO
consulta 3 nao
gerar 3 padrao não
```

Se o treino for passado separadamente, a entrada padrão conterá apenas os comandos (e seu programa será chamado usando uma sintaxe similar a “./a.out treino.txt”):

consulta 3 nao
gerar 3 padrao não

Comandos que seu programa deverá aceitar

Seu programa deverá aceitar apenas dois possíveis comandos: “gerar” e “consultar”. (cada comando estará completamente contido em uma linha). Após o resultado de cada consulta imprima uma linha em branco (para separar o resultado de diferentes consultas).

Um exemplo de entrada e saída se encontra no final deste documento.

Comandos do tipo “consultar”

Esse comando será utilizado para consultar estatísticas coletadas no arquivo de treino sobre palavras. Esse comando poderia ser utilizado, por exemplo, para se consultar quais seriam as palavras mais prováveis de aparecer após uma determinada palavra. Cada comando “consultar” será seguido por um número K e, entao, 1, 2 ou 3 palavras.

Se for fornecida uma palavra p0, seu programa deverá imprimir uma linha contendo “p0 (freq)”, onde freq e’ a frequência com que p0 aparece no texto. A seguir, deverão ser impressas as (até) K ocorrências mais frequentes de palavras que aparecem após p0 (sempre em ordem decrescente de frequência e crescente lexicográfica).

Por exemplo, considerando o texto de treino apresentado no início deste documento, a consulta “consultar 5 voce” deveria gerar como saida:
voce (4)
voce vai (3)
voce esta (1)

Note que, apesar de K ser 5, apenas 2 ocorrencias foram impressas (pois não ha mais ocorrências no arquivo de entrada).

Se forem fornecidas duas palavras p0 p1 para a consulta, seu programa deverá imprimir (no mesmo formato) a frequência com que p1 aparece após p0. A seguir, deverão ser impressas as até K ocorrências mais frequentes de palavras que ocorrem imediatamente após p0 seguida de p1.

Considerando o mesmo arquivo de treino anterior, uma consulta do tipo “consultar 5 voce vai” deveria gerar como resultado:
voce vai (3)
voce vai tambem (1)
voce vai viajar (1)

Finalmente, se forem fornecidas três palavras p0 p1 p2, seu programa deverá apenas imprimir a frequência com que p2 aparece imediatamente após p1 e p0 (nesse caso, o valor K ainda será fornecido, mas não fará diferença).

No exemplo anterior, a consulta “consultar 5 voce vai tambem” deveria gerar como resultado:

voce vai tambem (1)

Comandos do tipo “gerar”

Esse comando deverá gerar uma frase utilizando a técnica de predicao descrita neste documento. Cada frase gerada deverá estar em uma linha e após cada palavra deverá ser impresso um espaço em branco. A sintaxe do comando gerar deverá ser “gerar K MODO p0 p1 p2 p3 …. pm” (cada argumento separado por um espaço em branco), onde K e’ o número de palavras a serem adicionadas a frase, MODO e’ a técnica utilizada para a geração, podendo ser “padrao” ou “aleat” (essas técnicas serão descritas posteriormente) e *p0 p1 p2 … pm* são as m (0 <= m <= 1000) palavras que compõem o texto inicial.

Ao avaliar esse comando, seu programa deverá adicionar K palavras após a última palavra fornecida (*pm*), utilizando os dicionários para determinar qual palavra deverá ser adicionada em cada passo (sempre o de mais alto nível possível).

Modo padrao: No caso do modo padrão, seu programa deverá sempre escolher como próxima palavra a mais frequente de todas (em caso de empate, escolha a lexicograficamente menor). Isso seria similar a digitar algumas palavras no teclado do celular e, a seguir, sempre escolher a primeira sugestão apresentada pelo teclado para a palavra seguinte.

Por exemplo, considere o comando “gerar 3 padrao estou indo”. Nesse caso, a frase inicial e’ “estou indo” e você deverá adicionar 3 palavras a ela. Se a palavra mais frequente (no dicionário de nível 3) após “estou” seguida de “indo” for “agora”, tal palavra deverá ser adicionada a frase (“estou indo agora”). A seguir, uma nova palavra será adicionada. Suponha que a palavra mais comum após “indo” seguida de “agora” seja “nao”. Assim, a frase se tornará “estou indo agora nao”). Por fim, assuma que não há nenhuma ocorrência de palavra após “agora” seguido de “não”. Nesse caso, tentaremos um dicionário de nível 2 para ver qual a palavra mais frequente após “nao”. Se essa palavra for “estou”, a frase resultante que seu programa deverá imprimir será: “estou indo agora nao estou”.

Modo aleat: (aleatorio) o modo aleatório funciona de forma similar ao padrão. Porém, em vez de, em cada passo, escolher a palavra que ocorre com maior frequência após as anteriores, seu programa deverá adicionar um pouco de aleatoriedade a essa escolha (mas ainda assim espera-se que as palavras mais frequentes tenham maior probabilidade de serem escolhidas). Por exemplo, se após “eu vou” as palavras mais frequentes são “agora” (com frequência 10), “amanha” (com frequência 4) e “caminhar” (com frequência 1), então seu programa deverá escolher como próxima palavra uma dessas 3, sendo que “agora” será a palavra mais provável de ser escolhida e “caminhar” teria a menor probabilidade.
A forma exata com que essas probabilidades seriam distribuídas deve ser escolhida pelo aluno -- use sua criatividade (isso será corrigido de forma manual).

Complexidade das operações

A eficiência de suas operações e' importante nesse trabalho. Seja N o número de palavras presentes no texto de treino e assuma que o tamanho das palavras seja pequeno (ou seja, não vamos levar o tamanho das strings em conta na complexidade).

O pré-processamento deverá ter complexidade **menor** do que $O(N^2)$.

A complexidade da operação consulta deverá ser **menor** do que O(N) (assumindo que o número K de ocorrências a serem impressas e' muito menor do que N).

No caso da geração de frases no modo padrão, o custo associado a adicionar cada uma das K palavras a frase a ser gerada também deverá ser menor do que O(N) (ou seja, o custo total da geração deverá ser menor do que O(NK)). Não há restrições específicas sobre a complexidade do modo aleatório, mas tente implementá-lo de forma eficiente.

Arquivo README

Seu trabalho deverá incluir um arquivo README.

Tal arquivo contera:

- Seu nome/matricula
- Informacoes sobre todas fontes de consulta utilizadas no trabalho

Submissao

Submeta seu trabalho utilizando o sistema Submitty até a data limite. Seu programa será avaliado de forma automática (os resultados precisam estar corretos, o programa não pode ter erros de memória, etc), passará por testes automáticos "escondidos" e a qualidade do seu código será avaliada de forma manual.

Você deverá enviar os arquivos: README, seus arquivos .cpp e .h (com os nomes definidos neste documento).

Duvidas

Dúvidas sobre este trabalho deverão ser postadas no sistema Piazza. Se esforce para implementá-lo e não hesite em postar suas dúvidas!

Avaliacao manual

Principais itens que serão avaliados (além dos avaliados nos testes automáticos):

- Comentarios
- Indentacao
- Nomes adequados para variáveis
- Separação do código em funções lógicas
- Uso correto de const/referencia

- Uso de variáveis globais apenas quando absolutamente necessário e justificável (uso de variáveis globais, em geral, e' uma má prática de programação).
- etc

Regras sobre plágio e trabalho em equipe

- Este trabalho deverá ser feito de forma individual.
- Porém, os alunos podem ler o roteiro e discutir ideias/algoritmos em alto nível (nível mais alto do que pseudocódigo) de forma colaborativa.
- As implementações (nem mesmo pequenos trechos de código) não deverão ser compartilhadas entre alunos. Um estudante não deve olhar para o código de outra pessoa.
- Um estudante não deve utilizar o computador de outro colega para fazer/submeter o trabalho (devido ao risco de um ter acesso ao código do outro). Cuidado para não deixar seu código fonte em algum computador público (a responsabilidade pelo seu código e' sua).
- Crie um arquivo README (submeta-o com o trabalho) e inclua todas as suas fontes de consulta (incluindo pessoas que lhe ajudaram em algo do trabalho).
- Não poste seu código (nem parte dele) no Piazza (ou outros sites) de forma pública (cada aluno e' responsável por evitar que outros plagiem seu código).
- Se você estiver lendo isto (deveria estar…) escreva "Eu li as regras" na primeira linha do seu README.
- Trechos de código não devem ser copiados de livros/internet. Se consultar algum livro ou material na internet essa fonte deverá ser citada no README.
- Se for detectado plágio (mesmo em pequenos trechos de código) a nota de TODOS estudantes envolvidos será 0.
- Além disso, os estudantes poderão ser denunciados aos órgãos responsáveis por plágio da UFV. Lembrem-se que um conceito F (fraude académica) no histórico escolar pode afetar severamente seu currículo.
- O plágio pode ser detectado após a liberação das notas do trabalho (ou seja, pode ser que o professor reavalie os trabalhos procurando por plágio no final do semestre). Assim, sua nota poderá ser alterada para 0 caso algum problema seja encontrado posteriormente (essa alteração na nota vale apenas para casos de plágio).

Exemplo de entrada:

COMECO_TREINO
Oi como voce esta.
voce vai viajar.
voce vai tambem.
como voce vai.
FINAL_TREINO
consultar 5 voce
consultar 5 voce vai
consultar 5 voce vai tambem
gerar 5 padrao oi

Saida esperada:

---

voce (4)
voce vai (3)
voce esta (1)

voce vai (3)
voce vai tambem (1)
voce vai viajar (1)

voce vai tambem (1)

oi como voce esta voce vai
