UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO

Alexandre Galocha Pinto Junior (10734706)
Eduardo Pirro (10734665)
Fernando Gorgulho Fayet (10734407)

Tema 2: escritor automático para smartphones

18 de Junho de 2019
São Carlos, SP

1. Introdução

O sistema desenvolvido para o trabalho tem como objetivo principal popular frases descritivas (apenas algumas palavras, como: “sair trabalho”) com expressões utilizadas pelos usuários do sistema (populando a frase como: “hoje não vou sair porque estou fazendo trabalho”). Também é possível conseguir sugestões de palavras a partir de uma palavra dada (onde se entra uma palavra e recebe no máximo 3 sugestões de palavras para adicionar na sequência).

Para o funcionamento do sistema, foram utilizados algoritmos de caminhos de maior peso, com penalidades para evitar repetições.

2. Explicações

2.1. Decisões de Projeto

Para realizar o projeto percebemos que precisaríamos de um algoritmo que encontra caminhos com maior peso possível (o equivalente a pegar as palavras que possuem a maior quantidade de ligações entre si).

Então, como ponto de partida no projeto decidimos implementar o algoritmo de Dijkstra, que nos daria o caminho mínimo de uma palavra até outra palavra. Fizemos isso pensando em utilizar o algoritmo e através de algumas modificações chegar ao resultado esperado, que seria de percorrer o maior número de palavras, porém, não permitir que o algoritmo ficasse preso em um *loop*.

Porém, após implementar Dijkstra, pensamos que a adaptação do algoritmo não traria resultados muito bons, pois apenas realizar as “punições” caso o algoritmo entrasse em um *loop*, poderia ainda assim trazer resultados não tão satisfatórios.

Por isso procuramos um algoritmo que fosse atender melhor à nossa necessidade. Foi então que chegamos a solução que utilizamos no projeto final.

O algoritmo que utilizamos, que foi idealizado por Eduardo Pirro, é um algoritmo guloso que busca sempre pelas ligações de maior peso já criadas, mas que permite que um nó tenha apenas dois antecessores sendo que esses antecessores devem ser diferentes. Dessa forma, a execução não fica presa em um *loop*. O algoritmo idealizado inicialmente por Eduardo, foi discutido e testado pelo grupo em diversos e ao final chegamos em uma versão dele com um funcionamento desejado ao início do projeto. Em homenagem ao aluno resolvemos chamar o algoritmo de “Pirro’s Algorithm”.

Esse algoritmo é uma alteração do algoritmo de dijkstra que busca o maior caminho “seguro” a partir de um vértice inicial para todos os demais, guardando sempre um máximo de 2 antecessores de cada vértice, a idéia é, verificando os antecessores, encontrar loops que seriam infinitos e impedir que o programa

continue para esses casos e, ao mesmo tempo guardar e interpretar o antecessores de modo que seja possível entrar no loop sem perder o caminho anterior.

O algoritmo funciona da seguinte maneira:

1. Criação de um vetor de distâncias e dois de antecessores, iniciados com valores pessimistas.
2. Distância ao vértice inicial = 0 e o insere num vetor.
3. Enquanto o vetor não estiver vazio, selecione o elemento com maior distância contido nele.
4. Para todos os adjacentes ‘v’ do vértice atual ‘u’, verifica se a distância de v é menor que a distância até u + peso da aresta u->v e se é seguro* adicioná-lo no vetor, caso seja, ocorre um “tensionamento**” e o vértice é inserido no vetor.
5. Ao final teremos a distância e ambos os vetores de antecessor de cada um dos vértices com valores definidos.

*seguro significa que, ao chamá-lo não estaria gerando um looping infinito, a verificação desse caso é bastante simples, tendo dois vértices: ‘u’, ‘v’, tal que exista uma aresta u->v, será seguro continuar o algoritmo para v se o antecessor primário de v for diferente de u, isso garante que não está entrando infinitamente em um loop pois não insere duas vezes com o mesmo antecessor (exceto no caso de o antecessor secundário de v for nulo ou igual ao u, pois isso não caracteriza ciclo infinito).

**tensionamento é o nome do algoritmo que faz exatamente o oposto do relaxamento, tendo dois vértices: ‘u’, ‘v’, tal que exista uma aresta u->v, a distância de v será atualizada com a distância de u + peso da aresta u->v, o antecessor primário de v passa a ser u e o que era primário anteriormente passa a ser secundário.

Ao finalizar o algoritmo, percebemos que a utilização da estrutura “Heap” assim como no algoritmo de Dijkstra, poderia otimizar o tempo de execução.

Ao final obtivemos um algoritmo com complexidade O(A*log(V)) em tempo e complexidade O(A) em espaço, sendo A o número de arestas e V o número de vértices do grafo.

Outra decisão que tivemos que tomar foi relativa ao tratamento de sinais de pontuação e de letras maiúsculas. A decisão tomada pelo grupo foi de não realizar um tratamento (remover pontuações e colocar todos os caracteres em letra minúscula por exemplo), pois ao refletirmos sobre, percebemos que, no geral, as palavras que possuem pontuação atrelada vão aparecer em situações que as que não possuem pontuação, não vão aparecer. As letras maiúsculas também, que estão fortemente ligadas aos inícios de frase.

2.2. Funcionalidades

O sistema finalizado possui 4 funcionalidades que serão descritas abaixo.

A primeira funcionalidade é de inserir palavras no grafo de palavras do usuário. Essa funcionalidade funciona da seguinte maneira:

O usuário escreve o número de palavras que a frase a ser inserida possui e a frase que deve ser inserida no grafo.

O funcionamento da inserção é o seguinte: sempre que vai ser inserida uma ligação entre palavras, cada uma das palavras é buscada; se não existem, são inseridas no grafo. Então é feita uma busca procurando a ligação entre as palavras; se ela existe, é apenas incrementada; se não existe, é inserida.

A segunda funcionalidade é a de imprimir o grafo na tela. Ela imprime palavra a palavra (incluindo as "palavras" pré-definidas, que são "__BEG__" e "__END__", para indicar inicio e fim de frases, respectivamente), e todas as palavras que estão ligadas a palavra impressa.

A terceira funcionalidade é a de completar uma frase com palavras descritivas. Para ela, o usuário deve inserir o número de palavras a frase descritiva deve ter, seguido da frase descritiva. Então o programa irá rodar o algoritmo de caminho máximo para cada par de palavras e retornar, sempre que possível, o caminho entre essas duas palavras, contendo as palavras com maior quantidade de ligações entre si mesmas. Caso não seja possível encontrar uma ligação ou uma palavra, o programa retorna um erro, dizendo que não foi possível realizar a operação.

Antes de finalizar o projeto terminar o projeto, vimos que uma outra função interessante para o sistema seria o de sugerir "próximas" palavras, quase um equivalente aos teclados dos *smartphones*.

Então a quarta funcionalidade é exatamente essa, a sugestão de palavras. Ela busca por palavras que possuam o maior número de ligações com a palavra inserida e retorna no máximo 3 sugestões de palavras para adicionar na sequência.

3. Demonstração

Nessa seção do trabalho faremos uso de imagens da execução do sistema para exemplificar e demonstrar o seu funcionamento.

Começaremos mostrando o menu inicial do programa, que tem as opções que podem ser utilizadas no sistema.

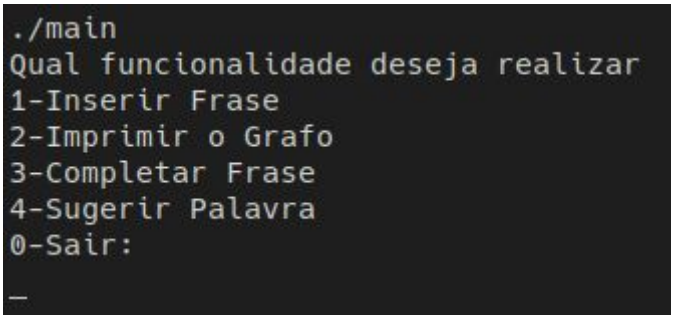

Figura 1: Menu principal do sistema

Como podemos ver na figura 1, o sistema possui 4 opções em seu menu: inserir uma frase nova no grafo (inserindo 1); imprimir o grafo atual, que contém todas as frases (ligações entre palavras) já inseridas pelo usuário no grafo (inserindo 2); completar uma frase, onde o usuário insere palavras que deseja que esteja na frase e o programa monta a frase (inserindo 3); e sugerir palavra, onde o usuário insere uma palavra e o sistema sugere até 3 opções de palavras para "seguirem" a palavra inserida (inserindo 4). Além disso, caso o usuário queira sair do programa ele pode digitar 0 e o programa será finalizado.

Partiremos agora para a tela de inserção de frases no grafo interno ao sistema de sugestões.

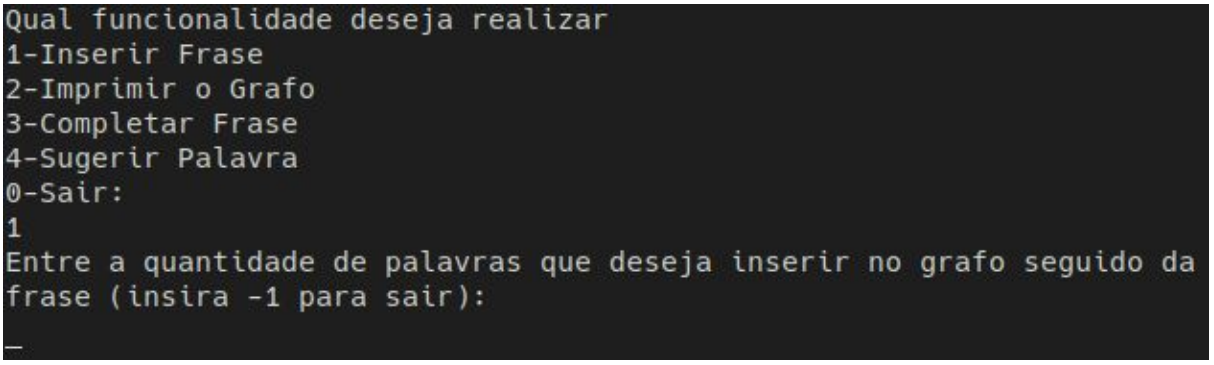

Figura 2: Opção 1 - Inserir frase(s)

Nessa primeira opção o sistema solicita que o usuário insira as frases que serão inseridas no grafo no seguinte formato:

N PALAVRA_1 PALAVRA_2 PALAVRA_3 .... PALAVRA_N

Um exemplo seria o seguinte:

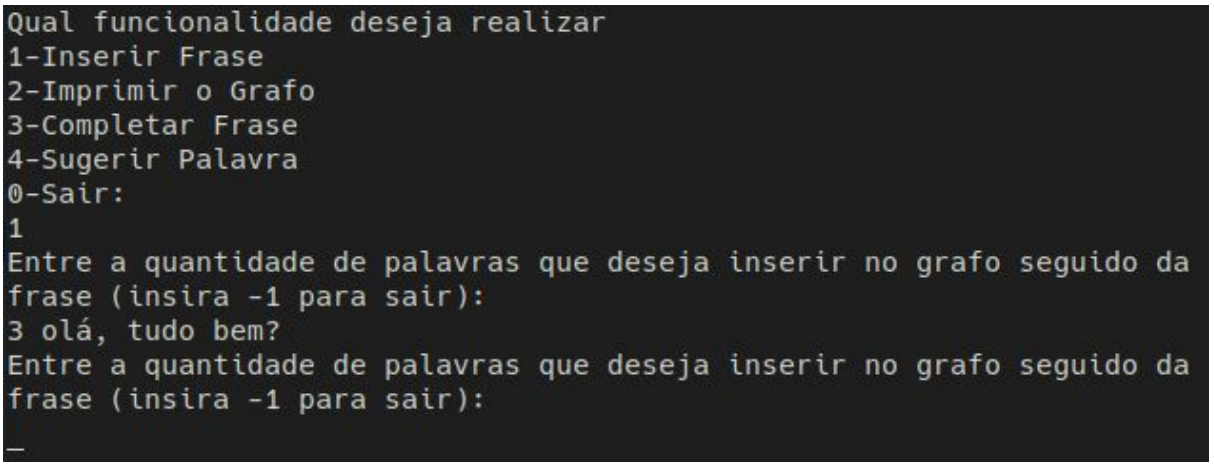

Figura 3: Exemplo de frase sendo inserida no programa

Nesse exemplo as palavras adicionadas ao grafo foram as seguintes (incluindo os sinais de pontuação):

- olá, (incluindo a vírgula)
- tudo
- bem? (incluindo a interrogação)

Para sair da execução da função 1, basta inserir -1 como na imagem a seguir:

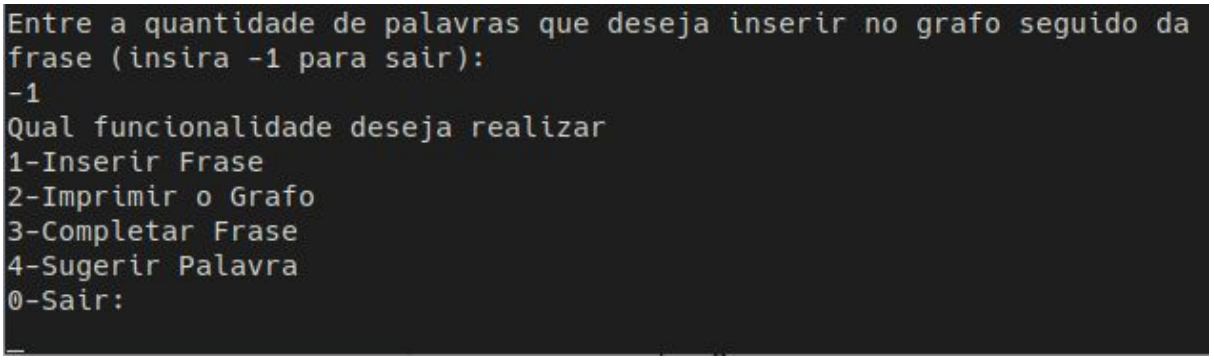

Figura 4: Saindo da função 1

Partiremos agora para a execução da função 2, que será demonstrada através de imagens.

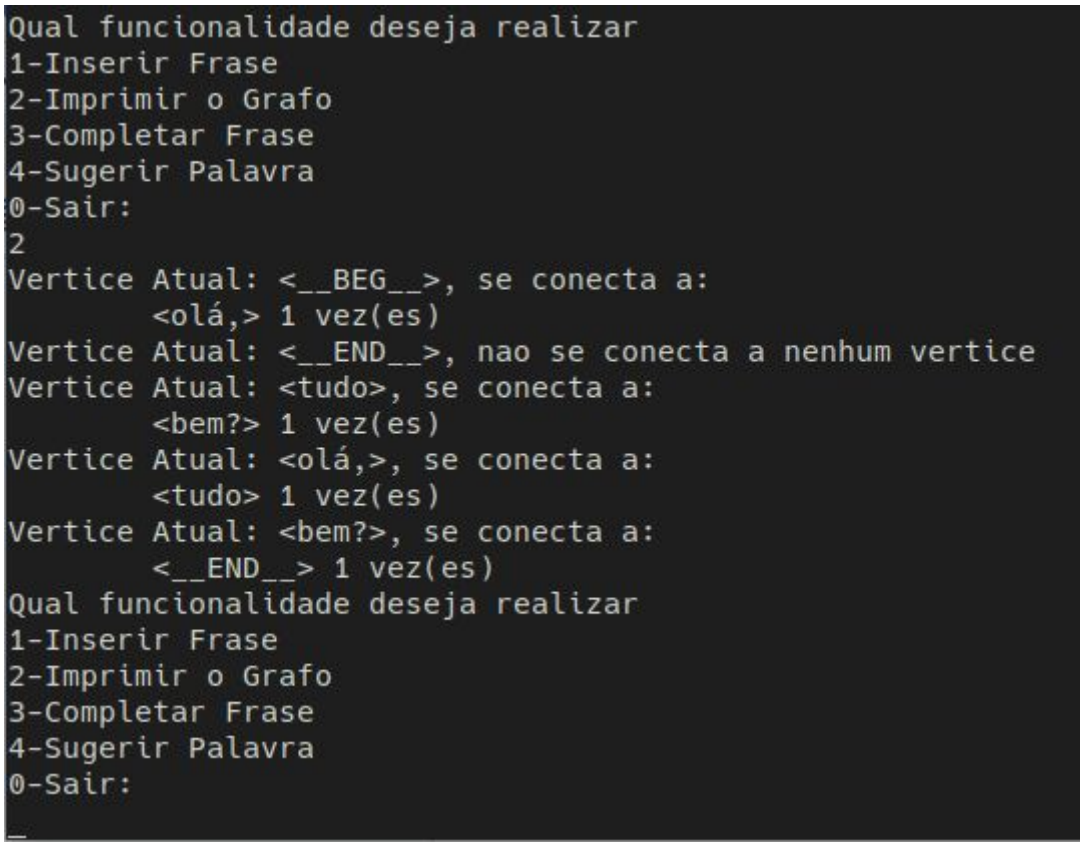

Figura 5: Execução da funcionalidade 2

A execução da funcionalidade 2 se limita a imprimir na tela todos os nós existentes no grafo atual e todas as ligações que possui. Essa impressão se dá da seguinte forma:

Vertice Atual <PALAVRA>, se conecta a:

<CONEXAO_1> PESO vez(es)

ou

Vertice Atual <PALAVRA>, não se conecta a nenhum vertice

Uma observação importante a ser feita é a presença de dois nós que não são palavras que foram inseridas pelo usuário explicitamente: __BEG__ e __END__. Essas “palavras” são utilizadas para indicar quais palavras começam uma frase e quais palavras terminam uma frase.

Partindo agora para a funcionalidade 3, funcionalidade principal do sistema, que é a de completar uma frase. As imagens inseridas abaixo são para exemplificar o funcionamento dessa função:

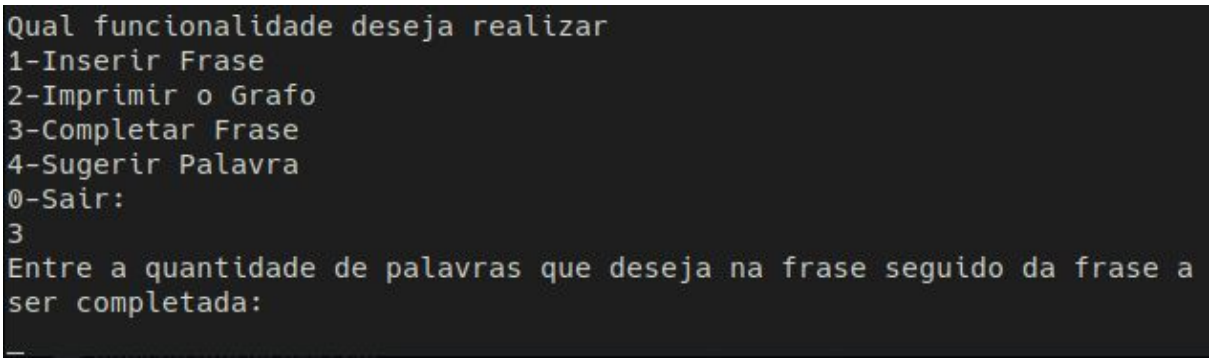

Figura 6: inserção da frase para ser completada

Nessa terceira opção o sistema solicita que o usuário insira as frases, que serão utilizadas pelo algoritmo de completar frases, no seguinte formato:

N PALAVRA_1 PALAVRA_2 PALAVRA_3 .... PALAVRA_N

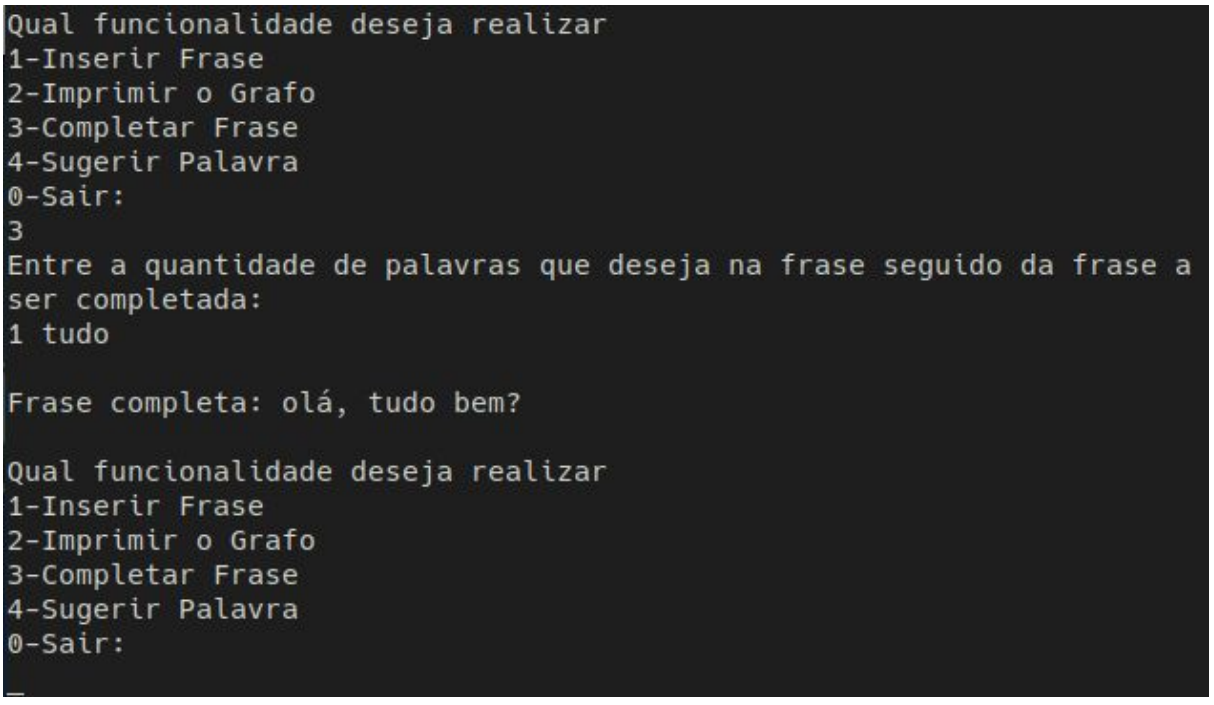

Figura 7: Execução da funcionalidade 3

A execução utiliza as palavras inseridas pelo usuário para percorrer o grafo e achar o caminho de peso máximo que passe pelo nó de início, por todas as palavras, e pelo nó de final de frase.

Após rodar o algoritmo, o sistema vai imprimindo as frases, parte a parte. Caso em alguma das partes ocorra um erro, o sistema para a execução. Esse caso de execução é demonstrado pela imagem a seguir:

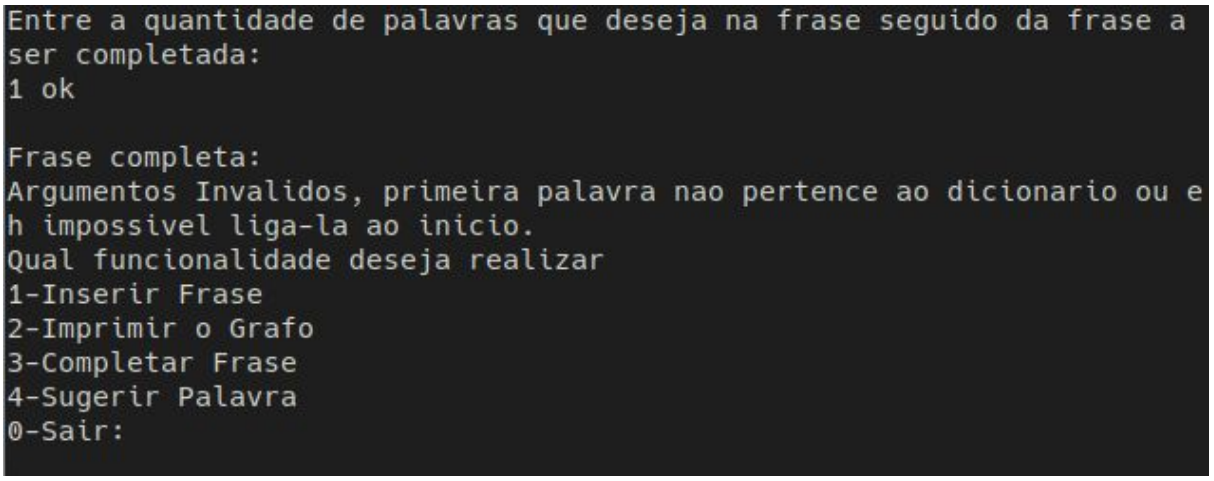

Figura 8: Erro na execução do algoritmo de completar frases

A última funcionalidade do sistema, a de sugerir próximas palavras, é exemplificada pelas seguintes imagens:

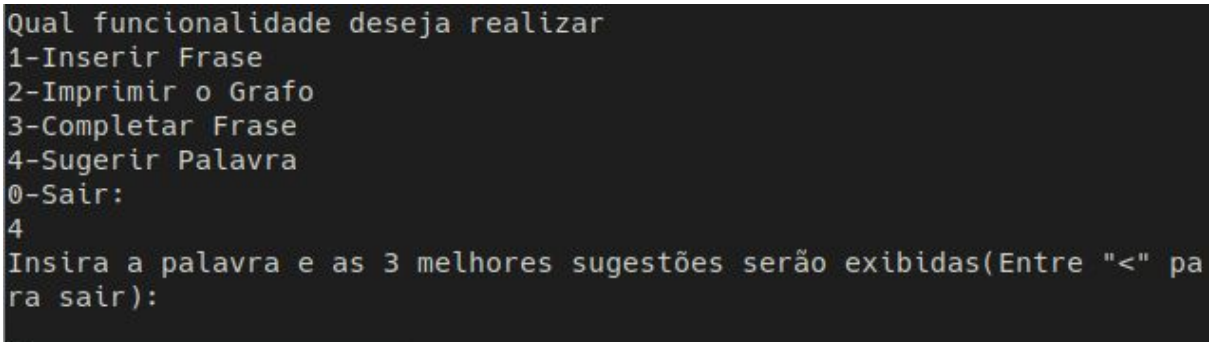

Figura 9: inserção da palavra para sugestão

Na última funcionalidade o sistema solicita que palavras sejam inseridas para que procure sugestões de palavras.

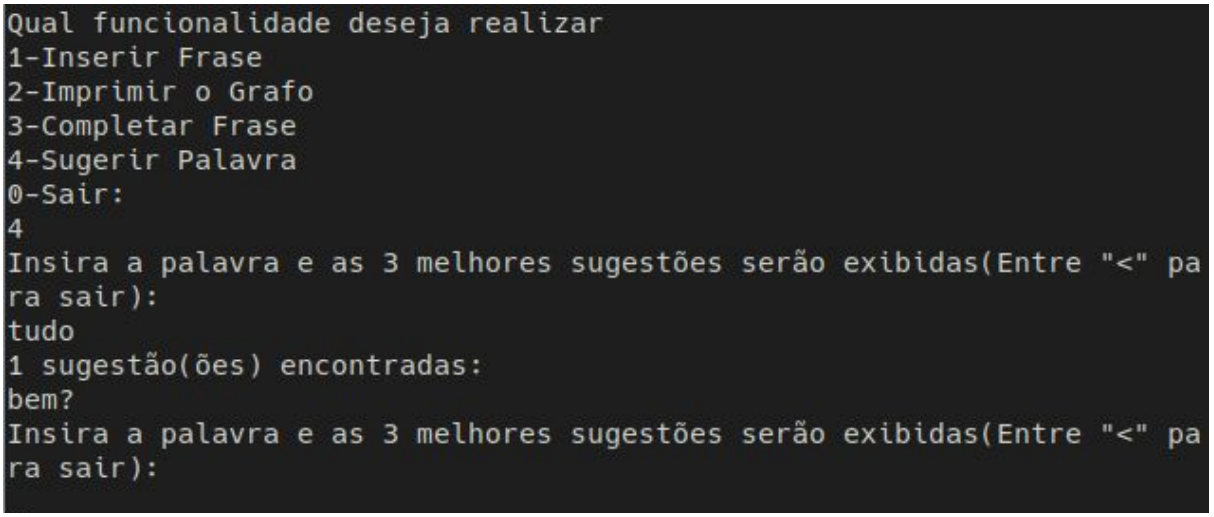

Figura 10: Execução da sugestão de palavras

Podemos ver evidenciada na imagem anterior a palavra sugerida para a palavra inserida pelo usuário.

Caso o usuário deseje parar a execução desta funcionalidade, basta que ele insira o caractere “<”, e o sistema volta para o menu inicial.

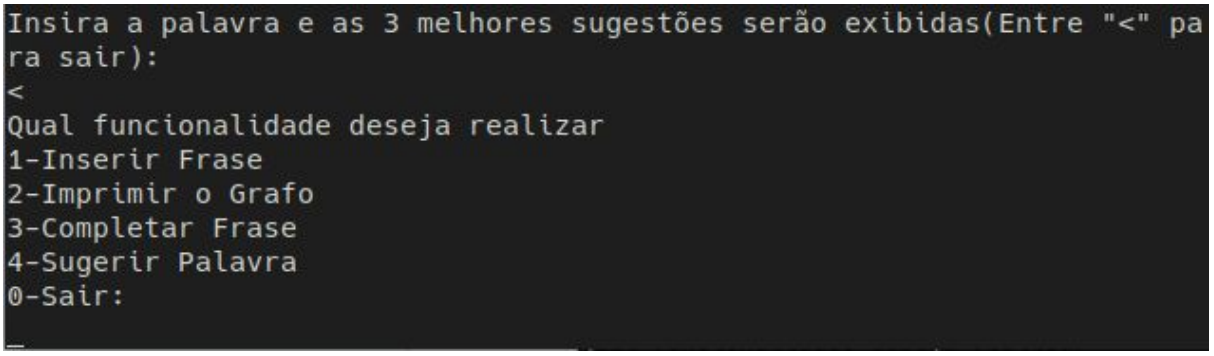

Figura 11: Saindo da funcionalidade 4

Caso, ao executar a busca por sugestões, o sistema não encontre a palavra ou não encontre sugestões para a palavra inserida, uma mensagem é exibida para o usuário, indicando que não existem sugestões

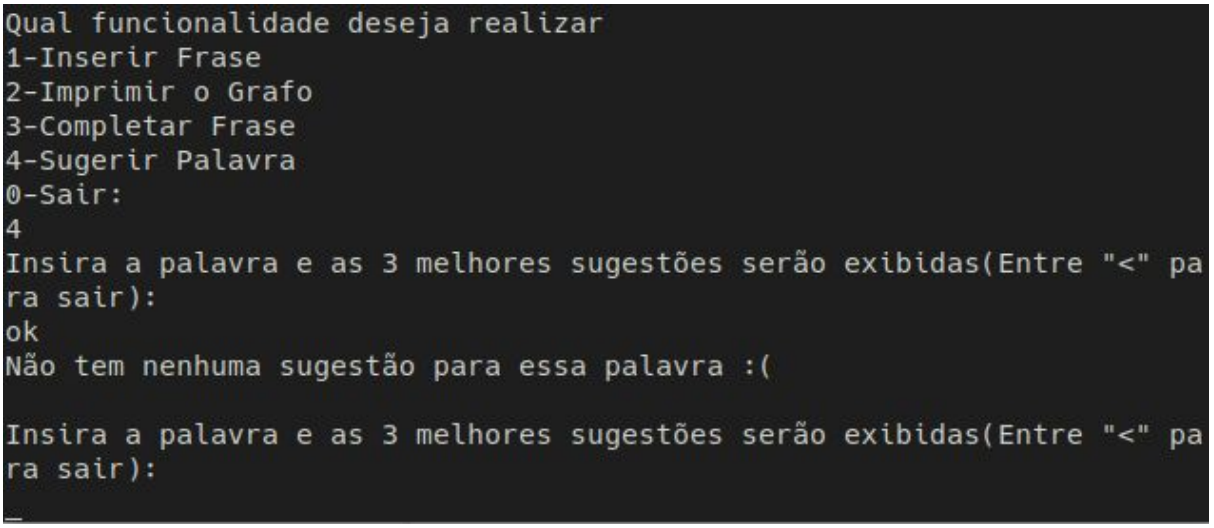

Figura 12: Erro ao processar a palavra

4. Conclusão

Com esse projeto o grupo pode discutir e aplicar bastante algoritmos mais avançados de grafos, com enfoque nos algoritmos de caminhos mínimos, que foram usados como base para chegar aos algoritmos de caminhos de maior peso.

Algumas conclusões que chegamos quanto ao manuseio de “textos” foram que as pontuações e letras maiúsculas, no geral, importam. Por isso, uma mesma palavra, mas que possua uma letra maiúscula, ou um sinal de pontuação a frente, podem determinar significados semânticos bem diferentes em sentenças, e portanto deve se posicionar diferentemente dentro da frase.

Outra conclusão que chegamos foi de que não faria sentido deixar o algoritmo englobar ciclos completamente mais de uma vez, afinal, isso apenas faria com que ficasse se repetindo as mesmas palavras por várias vezes.