Push_swap

Porque Swap_push não é tão natural

Resumo:

Este projeto fará com que você classifique os dados em uma pilha, com um conjunto limitado de instruções, utilizando o menor número possível de ações. Para ter sucesso você terá que manipular vários tipos de algoritmos e escolher a solução mais adequada (entre muitas) para uma classificação otimizada dos dados.

Versão: 7

Conteúdo

EU Prefácio 2

II Introdução 4

III Objetivos 5

IV Instruções Comuns 6

V Parte obrigatória As regras 8
Exemplo V.1 V.2 9
V.3 O programa "push_swap" 10

VI Parte bônus VI.1 O programa "checker" 12

VII Submissão e avaliação por pares 14

Capítulo I

Prefácio

- C

- ASM

- LOLCODE

- PHP

- BrainFuck

•C#

•HTML5

• YASL

• OCaml

Capítulo II

Introdução

O projeto **Push swap** é um projeto de algoritmo muito simples e altamente direto: os dados devem ser classificados.

Você tem à sua disposição um conjunto de valores inteiros, 2 pilhas e um conjunto de instruções para manipular ambas as pilhas.

Seu objetivo? Escreva um programa em C chamado push_swap que calcula e exibe na saída padrão o menor programa, feito de instruções *da linguagem Push swap ,* que ordena os inteiros recebidos como argumentos.

Fácil?

Veremos...

Capítulo III

Objetivos

Escrever um algoritmo de classificação é sempre uma etapa muito importante na jornada de um desenvolvedor. Muitas vezes é o primeiro encontro com o conceito de complexidade.

Algoritmos de classificação e sua complexidade fazem parte das questões clássicas discutidas durante entrevistas de emprego. Provavelmente é um bom momento para examinar esses conceitos, já que você terá que enfrentá-los em algum momento.

Os objetivos de aprendizagem deste projeto são rigor, uso de C e uso de algoritmos básicos. Especialmente focando em sua complexidade.

Classificar valores é simples. Classificá-los da maneira mais rápida possível é menos simples. Principalmente porque de uma configuração de inteiros para outra, a solução de classificação mais eficiente pode ser diferente.

Capítulo IV

Instruções Comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deverá ser escrito de acordo com a Norma. Se você tiver arquivos/funções bônus, eles serão incluídos na verificação de norma e você receberá 0 se houver um erro de norma.
- Suas funções não devem encerrar inesperadamente (falha de segmentação, erro de barramento, liberação dupla, etc.) além de comportamentos indefinidos. Caso isso aconteça, seu projeto será considerado não funcional e receberá nota 0 na avaliação.
- Todo o espaço de memória alocado no heap deve ser liberado adequadamente quando necessário. Sem vazamentos será tolerado.
- Se o assunto exigir, você deve enviar um Makefile que irá compilar seus arquivos fonte para a saída necessária com as flags -Wall, -Wextra e -Werror, use cc, e seu Makefile não deve relinkar.
- Seu Makefile deve conter pelo menos as regras $(NAME), all, clean, fclean e ré.
- Para entregar bônus ao seu projeto, você deve incluir uma regra bônus em seu Makefile, que adicionará todos os diversos cabeçalhos, bibliotecas ou funções que são proibidas na parte principal do projeto. Os bônus devem estar em um arquivo diferente _bonus.{c/h} se o assunto não especificar mais nada. A avaliação da parte obrigatória e da parte bônus é feita separadamente.
- Se o seu projeto permite que você use sua libft, você deve copiar seus fontes e seu Makefile associado em uma pasta libft com seu Makefile associado. O Makefile do seu projeto deve compilar a biblioteca usando seu Makefile e depois compilar o projeto.
- Nós encorajamos você a criar programas de teste para o seu projeto, mesmo que este trabalho **não precise ser enviado e não receba notas.** Isso lhe dará a chance de testar facilmente seu trabalho e o de seus colegas. Você achará esses testes especialmente úteis durante sua defesa. Na verdade, durante a defesa, você é livre para usar seus testes e/ou os testes do colega que está avaliando.
- Envie seu trabalho para o repositório git designado. Somente o trabalho no repositório git será avaliado. Se Deepthought for designado para avaliar seu trabalho, isso será feito

após suas avaliações por pares. Se ocorrer um erro em qualquer seção do seu trabalho durante a avaliação do Deepthought, a avaliação será interrompida.

Capítulo V

Parte obrigatória

V.1 As regras

- Você tem 2 pilhas chamadas a e b.
- No inicio:
  - ÿ A pilha a contém uma quantidade aleatória de números negativos e/ou positivos que não pode ser duplicado.
  - ÿ A pilha b está vazia.
- O objetivo é classificar os números em ordem crescente na pilha a. Para isso você tem à sua disposição as seguintes operações:

sa (swap a): Troque os 2 primeiros elementos no topo da pilha a.
Não faça nada se houver apenas um ou nenhum elemento.

sb (swap b): Troque os 2 primeiros elementos no topo da pilha b.
Não faça nada se houver apenas um ou nenhum elemento.

ss: sa e sb ao mesmo tempo.

pa (pressione a): Pegue o primeiro elemento no topo de b e coloque-o no topo de a.
Não faça nada se b estiver vazio.

pb (pressione b): Pegue o primeiro elemento no topo de a e coloque-o no topo de b.
Não faça nada se a estiver vazio.

ra (girar a): Desloca todos os elementos da pilha a em 1.
O primeiro elemento se torna o último.

rb (girar b): Desloca todos os elementos da pilha b em 1.
O primeiro elemento se torna o último.

rr: ra e rb ao mesmo tempo.

rra (rotação reversa a): Desloca todos os elementos da pilha a em 1.
O último elemento se torna o primeiro.

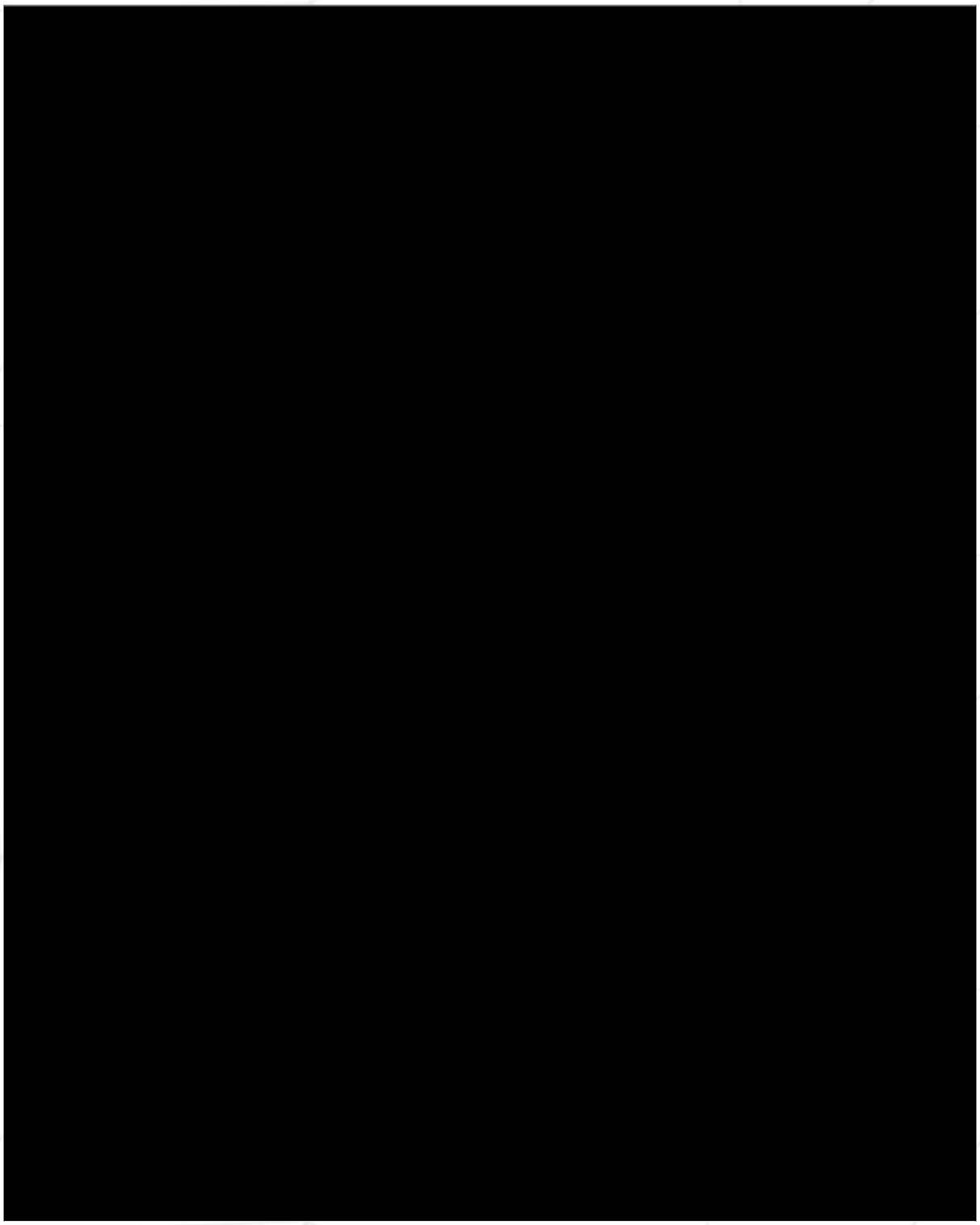
rrb (rotação reversa b): Desloca todos os elementos da pilha b em 1.
O último elemento se torna o primeiro.

rrr: rra e rrb ao mesmo tempo.

Exemplo V.2

Para ilustrar o efeito de algumas dessas instruções, vamos ordenar uma lista aleatória de inteiros.

Neste exemplo, consideraremos que ambas as pilhas crescem da direita.



Os inteiros de a são classificados em 12 instruções. Você pode fazer melhor?

V.3 O programa "push_swap"

Nome do programa	push_swap
Entregar arquivos	Makefile, *.h, *.c NAME, all,
Argumentos	clean, fclean, re stack a: Uma lista de
Makefile	inteiros
Funções externas.	• ler, escrever, malloc, grátis, saída • ft_printf e qualquer equivalente VOCÊ codificou
Libft autorizado	Sim
Descrição	Classificar pilhas

Seu projeto deve obedecer às seguintes regras:

• Você terá que entregar um Makefile que irá compilar seus arquivos fonte. Não deve vincular novamente.

• Variáveis globais são proibidas.

• Você tem que escrever um programa chamado push_swap que receba como argumento a pilha formatada como uma lista de inteiros. O primeiro argumento deve estar no topo da pilha (tenha cuidado com a ordem).

• O programa deve exibir a menor lista possível de instruções para ordenar a pilha a, estando o menor número no topo.

• As instruções devem ser separadas por '\n' e nada mais.

• O objetivo é ordenar a pilha com o menor número possível de operações. Durante o processo de avaliação, o número de instruções encontradas pelo seu programa será comparado com um limite: o número máximo de operações toleradas. Se o seu programa exibir uma lista mais longa ou se os números não estiverem classificados corretamente, sua nota será 0.

• Se nenhum parâmetro for especificado, o programa não deve exibir nada e fornecer o solicitar de volta.

• Em caso de erro deverá exibir “Error” seguido de '\n' no erro padrão.
Os erros incluem, por exemplo: alguns argumentos não são inteiros, alguns argumentos são maiores que um número inteiro e/ou há duplicatas.

Durante o processo de avaliação, um binário será fornecido para verificar adequadamente seu programa.

Funcionará da seguinte forma:

Se o programa checker_OS exibir "KO", significa que seu push_swap surgiu com uma lista de instruções que não classifica os números.

O programa checker_OS está disponível nos recursos do projeto na intranet.

Você pode encontrar uma descrição de como funciona na parte bônus deste documento.

Capítulo VI

Parte bônus

Este projeto deixa pouco espaço para adicionar recursos extras devido à sua simplicidade. Porém, que tal criar seu próprio verificador?

Graças ao programa verificador, você poderá verificar se
a lista de instruções gerada pelo programa push_swap classifica a pilha corretamente.

VI.1 O programa “verificador”

Nome do programa	verificador
Entregar	*.h, *.c
arquivos	bônus
Makefile	pilha a: uma lista de inteiros
Argumentos Funções externas.	• ler, escrever, malloc, grátis, saída • ft_printf e qualquer equivalente VOCÊ codificou
Libft autorizado	Sim
Descrição	Execute as instruções de classificação

- Escreva um programa chamado checker que tome como argumento a pilha formatada como uma lista de inteiros. O primeiro argumento deve estar no topo da pilha (tenha cuidado com a ordem). Se nenhum argumento for fornecido, ele para e não exibe nada.
- Ele irá então esperar e ler as instruções na entrada padrão, cada instrução será seguida por '\n'. Depois de lidas todas as instruções, o programa deve executá-las na pilha recebida como argumento.

- Se depois de executar essas instruções, a pilha a estiver realmente ordenada e a pilha b estiver vazia, então o programa deverá exibir "OK" seguido de '\n' na saída padrão.

- Em todos os outros casos, deverá exibir "KO" seguido de '\n' na saída padrão.
- Em caso de erro, deve-se exibir "Error" seguido de '\n' no **erro padrão.** Os erros incluem, por exemplo: alguns argumentos não são inteiros, alguns argumentos são maiores que um inteiro, há duplicatas, uma instrução não existe e/ou está formatada incorretamente.

```
$>./verificador 3 2 1 0
rra
pb
sa
rra
pa

OK $>./verificador 3 2 1 0
sa
rra
pb

KO $>./checker 3 2 um 0 Erro
$>./
checker ""                  1
Erro $>
```

Você NÃO precisa reproduzir exatamente o mesmo comportamento do binário fornecido. É obrigatório gerenciar erros, mas cabe a você decidir como deseja analisar os argumentos.

A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a parte obrigatória foi feita integralmente e funciona sem mau funcionamento. Se você não passou em TODOS os requisitos obrigatórios, sua parte do bônus não será avaliada de forma alguma.

Capítulo VII

Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes dos seus arquivos para garantir que estão corretos.

Como essas atribuições não são verificadas por um programa, fique à vontade para organizar seus arquivos como desejar, desde que entregue os arquivos obrigatórios e cumpra os requisitos.

arquivo.bfe:VABB7yO9xm7xWXROeASsmsgnY0o0sDMJev7zFHhwQS8mvM8V5xQQp
Lc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZ0