Cash

Algoritmos Gulosos

Ao fazer uma venda, é provável que você queira minimizar o número de moedas que você entrega para cada cliente, para que não acabe (ou irrite o cliente!). Felizmente, a ciência da computação deu aos caixas em todos os lugares maneiras de minimizar o número de moedas devidas: algoritmos gulosos.

De acordo com o Instituto Nacional de Padrões e Tecnologia (NIST), um algoritmo guloso é aquele "que sempre pega a melhor solução imediata, ou local, enquanto encontra uma resposta. Algoritmos gulosos encontram a solução geral ou globalmente ideal para alguns problemas de otimização, mas podem encontrar soluções piores que as ideais para algumas instâncias de outros problemas."

O que tudo isso significa? Bem, suponha que um caixa deva a um cliente algum troco e na gaveta desse caixa tenha moedas (R$ 0,25), dez centavos (R$ 0,10) cinco centavos (R$ 0,05) e centavos (R$ 0,01). O problema a ser resolvido é decidir quais moedas e quantas de cada uma entregar ao cliente. Pense em um caixa "guloso" como alguém que quer tirar o maior proveito possível desse problema com cada moeda que tira da gaveta. Por exemplo, se algum cliente tem 41 centavos de troco, a maior primeira (ou seja, melhor "mordida" imediata ou local) que pode ser feita é 25 centavos. (Essa mordida é "melhor" na medida em que nos deixa mais perto de R$ 0,00 mais rápido do que qualquer outra moeda faria.) Observe que uma mordida deste tamanho reduziria o que era um problema de R$ 0,41 a um problema de R$ 0,16, já que 41 - 25 = 16. Ou seja, o restante é um problema semelhante, mas menor. Desnecessário dizer que outra mordida de 25 centavos seria muito grande (supondo que o caixa prefere não perder dinheiro), e assim nosso caixa guloso passaria para uma mordida de 10 centavos, deixando-o com um problema de 6 centavos. Nesse ponto, a ganância pede uma mordida de R$ 0,05 seguida por uma mordida de R$ 0,01, ponto em que o problema é resolvido. O cliente recebe uma moeda de R$ 0,25, uma de R$ 0,10, uma de R$ 0,05 e uma de R$ 0,01: quatro moedas no total.

Acontece que esta abordagem gulosa (ou seja, algoritmo) não é apenas ideal localmente, mas também globalmente para a moeda dos EEUU (e também da União Europeia). Ou seja, enquanto o caixa tiver o suficiente de cada moeda, essa abordagem do maior para o menor renderá o menor número possível de moedas. Quão poucos? Bem, diga-nos você!

Detalhes de Implementação

Implemente, em um arquivo chamado `cash.c` em um diretório `~/pset1/cash`, um programa que primeiro pergunta ao usuário quanto dinheiro é devido e depois imprime o número mínimo de moedas com o qual esse troco pode ser pago.

- Use `get_float` para obter a entrada do usuário e `printf` para exibir sua resposta. Suponha que as únicas moedas disponíveis sejam vint e ecinco centavos (R$ 0,25), dez centavos (R$ 0,10), cinco centavos (R$ 0,05) e um centavo (R$ 0,01).
    - Pedimos que você use `get_float` para que possa lidar com reais e centavos, embora sem o cifrão. Em outras palavras, se devemos R$ 9.75 a algum cliente (como no caso em que um jornal custa 25 centavos, mas o cliente paga com uma nota de R$ 10), suponha que a entrada de seu programa será de `9.75` e não de `R$ 9.75` ou `975`. No entanto, se devemos R$ 9.00 a algum

cliente, suponha que a entrada de seu programa será `9.00` ou apenas `9`, mas, novamente, não `R$ 9` ou `900`. É claro que, pela natureza dos valores de ponto flutuante, seu programa provavelmente funcionará com entradas como `9.0` e `9.000` também; você não precisa se preocupar em verificar se a entrada do usuário está "formatada" como o dinheiro deveria estar.

    - **Observação:** Note que você deverá usar um ponto (.) como separador de decimal em vez de uma vírgula.

- Você não precisa tentar verificar se a entrada de um usuário é muito grande para caber em um **ponto flutuante**. Usar `get_float` sozinho garantirá que a entrada do usuário seja realmente um valor de ponto flutuante (ou inteiro), mas não garante que seja não negativo.
- Se o usuário não fornecer um valor não negativo, seu programa deve solicitar novamente ao usuário um novo valor até que o usuário forneça um valor válido.
- Para que possamos automatizar alguns testes do seu código, certifique-se de que a última linha de saída do seu programa seja apenas o número mínimo de moedas possível: um inteiro seguido por `\n`.
- Cuidado com a imprecisão inerente aos valores de ponto flutuante. Lembre-se de `floats.c` da aula, em que, se `x` for `2` e for `10`, `x/y` não será precisamente dois décimos! E assim, antes de fazer a mudança, você provavelmente desejará converter os reais inseridos pelo usuário em centavos (ou seja, de um valor em ponto flutuante para um inteiro) para evitar pequenos erros que poderiam se acumular!
- Tome cuidado para arredondar seus centavos para o centavo mais próximo, como acontece com round, que é declarado em `math.h`. Por exemplo, se o valor em real for ponto flutuante com a entrada do usuário (por exemplo, 0.20), codifique como

```
int cents = round(dollars * 100);
```

isto seguramente converterá `0.20` (ou mesmo `0.20000000298023223876953125`) para `20`.

Seu programa deveria se comportar como abaixo

```
$ ./cash
Change owed: 0.41
4
```

```
$ ./cash
Change owed: -0.41
Change owed: foo
Change owed: 0.41
4
```

Como testar seu código

Seu código funciona conforme prescrito quando você insere

- -1.00 (ou outros números negativos)?
- 0.00?
- 0,01 (ou outros números positivos)?
- letras ou palavras?
- nenhuma entrada. Isto é, quando você apenas pressiona Enter?

Você também pode executar o comando seguinte para avaliar a exatidão do seu código usando `check50`. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2020/x/cash
```

Execute, também o comando `style50` como abaixo

```
style50 cash.c
```

para garantir que seu código segue adequadamente as regras de **estilização de código**.

Informações adicionais e OBRIGATÓRIAS

As primeiras linhas do seu código devem consistir de um comentário de várias linhas contendo sua matrícula, seu nome completo e seu nome de usuário do github como no exemplo abaixo.

```
MATRÍCULA: ................
NOME: ................
USUÁRIO: ..............
```

substitua os pontos com suas informações.

Enviando seu programa

Uma vez que você tenha verificado o funcionamento, `check50` e a estilização do código, `style50`, execute o comando abaixo, logando com sua conta do GitHub através de **nome de usuário** e **senha**. Por questões de segurança, você verá asteríscos (*) em vez dos caracteres da sua senha.

```
submit50 cs50/problems/2020/x/cash
```