Aula 11: Fechando Programação Estruturada

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA

Sumário

Um último tópico em C

- Ponteiro para função ou Ponteiro de função!

Ponteiro para função

- Em termos práticos, a utilidade de um "ponteiro para função"é semelhante a delegates do C# e algumas diretivas do javascript.
- Um programa é um conjunto de instruções armazenado na memória, assim como seus dados. Por este motivo, é possível referenciar o endereço de uma função.

Ponteiro para função

```c
int Add(int a, int b){
    return a + b;
}

int Subtract(int a, int b){
    return a - b;
}

int Multiply(int a, int b){
    return a * b;
}

int Divide(int a, int b){
    return a / b;
}

int main(){
    int (*pfun)(int, int) = Add;
    printf("%d", pfun(2, 2));

    return 0;
}
```

Ponteiro para função

```c
void Print(int a, int b, int (*pfun)(int, int)){
    printf("%d", pfun(a, b));
}

int main(){
    Print(2, 2, Add);
    Print(2, 2, Divide);
}
```

Ponteiro para função como parâmetro.

Ponteiro para função

```
1 int main(){
2     int (*pfun[4])(int, int) = {Add, Subtract, Multiply, Divide};
3
4     for (int i = 0; i < 4; i++)
5         Print(8, 4, pfun[i]);
6 }
```

Vetor de ponteiros para função: Interessante para composição.

Complexidade

- Mais alguns detalhes de complexidade!

Complexidade

Notação	Nome
$O(1)$	ordem constante
$O(logn)$	ordem logarítmica
$O((logn)^c)$	ordem poli-logarítmica
$O(n)$	ordem linear
$O(nlogn)$	ordem linear-logarítmica
$O(n^2)$	ordem quadrática
$O(n^3)$	ordem cúbica
$O(n^c)$	ordem polinomial
$O(c^n)$	ordem exponencial
$O(n!)$	ordem fatorial

c é uma constante, n é variável.

Complexidade

Exemplos:

- Busca: $O(1)$, $O(n)$
- Ordenação: $O(nlogn)$, $O(n^2)$
- Multiplicação de matrizes: $O(n^3)$
- Caixeiro Viajante: $O(n!)$

Complexidade: P vs NP

- Classe de problemas P: Podem ser resolvidos em tempo polinomial
- Classe de problemas NP: $O(c^n)$, $O(n!)$. Não podem ser resolvidos em tempo polinomial (somente em tempo exponencial, será?).
  - Problema da mochila, Torre de Hanoi... Existem muitos problemas interessantes nessa classe!
- Em qual classe o seu trabalho final se encaixa?

Complexidade: P vs NP

Os erros a seguir são bastante comuns.

- "Problemas NP-completo são os problemas mais difíceis conhecidos."Uma vez que problemas NP-completo estão em NP, seu tempo de execução é no máximo exponencial. Porém, alguns problemas requerem mais tempo, como por exemplo a aritmética de Presburger.
- "Problemas NP-completo são difíceis porque existem diferentes soluções."De um lado, existem vários problemas que possuem um espaço de solução grande, mas podem ser resolvidos em tempo polinomial. Do outro lado, existem problemas NP que no máximo uma solução que é NP-difícil sob redução em tempo polinomial aleatória.

Complexidade: P vs NP

- "Resolver problemas NP-completo requerem tempo exponencial."Primeiramente, isso implicaria P $\neq$ NP, que ainda é uma questão não resolvida. Além disso, alguns problemas NP-completo efetivamente possuem algoritmos que rodam em tempo superpolinomial, porém subexponencial.
- "Todas as instâncias de problemas NP-completo são difíceis"Geralmente algumas instâncias, ou quase todas, podem ser facilmente resolvidas em tempo polinomial.

Complexidade: P vs NP

A pergunta de um milhão de dólares:
P versus NP: P = NP ou P $\neq$ NP

Os sete problemas do milênio

- P versus NP
- A conjectura de Hodge
- A conjectura de Poincaré (resolvido por Grigori Perelman)
- A hipótese de Riemann (LEIAM: A música dos números primos)
- A existência de Yang-Mills e a falha na massa
- A existência e suavidade de Navier-Stokes
- A conjectura de Birch e Swinnerton-Dyer

Computabilidade

- Até aqui comentamos problemas que são difíceis de serem resolvidos.... Mas todo problema pode ser resolvido?
- Existe algoritmo para tudo?
- A computabilidade de um problema é intimamente ligada à existência de um algoritmo para resolver o problema.

Computabilidade - Exemplo

Problema da parada: Um problema de decisão.

- Dadas uma descrição de um programa e uma entrada finita, decida se o programa termina de rodar ou rodará indefinidamente.
- Alan Turing provou em 1936 que um algoritmo genérico para resolver o problema da parada para todos pares programa-entrada possíveis não pode existir. Dizemos que o problema da parada é indecidível nas Máquinas de Turing.

Turing e outros grandes mestres

- Os fundamentos matemáticos da ciência da computação moderna começaram a serem definidos por Kurt Gödel com seu teorema da incompletude (1931).
- Essa teoria mostra que existem limites no que pode ser provado ou desaprovado em um sistema formal; isso levou a trabalhos posteriores por Gödel e outros teóricos para definir e descrever tais sistemas formais, incluindo conceitos como recursividade e cálculo lambda.
- LEIAM: Gödel, Escher, Bach An Eternal Golden Braid

Turing e outros grandes mestres

- Em 1936 Alan Turing e Alonzo Church independentemente, e também juntos, introduziram a formalização de um algoritmo, definindo os limites do que pode ser computado, e um modelo puramente mecânico para a computação.
- Tais tópicos são abordados no que atualmente chama-se Tese de Church-Turing, uma hipótese sobre a natureza de dispositivos mecânicos de cálculo.
- O matemático húngaro John Von Neumann (1903-1957) formalizou o projeto lógico de um computador.
- Turing e Von Neumann participaram da criação do ENIAC.

Exercícios

- 1. Crie uma Agenda de Contatos (Nome, Telefone, Endereço) com uma Busca Binária baseada no nome do contato.
- 2. Crie uma Agenda de Contatos (Nome, Telefone, Endereço) baseada em Tabela Hash.
- 3. Exemplos de uso de ponteiro de função.

Na próxima aula...

Trabalho, entrega do trabalho e férias!!!