Projeto e Análise de Algoritmos

Projeto por indução

Projeto por Indução

- Algoritmos que resolvem certos problemas podem ser projetados utilizando a técnica de indução
- Passos para resolver um problema *P*
  - Mostrar como se resolve os casos base de *P* (uma ou mais pequenas instâncias)
  - Mostrar como a solução para *P* pode ser obtida a partir da solução de uma ou mais instâncias menores de *P*

Projeto por Indução

- Projeto por indução resulta em um algoritmo recursivo que
  - Possui casos base (condição de parada) equivalente ao caso base da indução
  - A aplicação da hipótese de indução corresponde à chamada recursiva
  - O passo da indução corresponde aos passos necessários para produzir a solução do problema a partir de soluções de instâncias menores retornadas pelas chamadas recursivas
- Vantagens
  - Prova de corretude do algoritmo é dada no uso correto da técnica
  - A complexidade pode ser calculada a partir da recorrência do algoritmo recursivo

Cálculo de Polinômios

- Problema
  - Dada uma sequência de números reais $a_n, a_{n-1}, \dots, a_0$ e um número real x, calcular o valor do polinômio
  $$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
  - Podemos remover o termo de maior ordem e resolver o subproblema com os $n-1$ termos de menores ordem primeiro para então resolver $P_n(x)$.

- Hipótese de indução (primeira tentativa)
  - Para um dado $n > 0$, uma dada sequência de números reais $a_{n-1}, \dots, a_0$ e um número real x, sabemos calcular o valor de
  $$P_{n-1}(x) = a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
  - Caso base: $n = 0$, a solução é a_0.
  - Passo de indução: Considerando que sabemos como calcular $P_{n-1}(x)$, para calcular $P_n(x)$, basta calcular
  $$P_n(x) = P_{n-1}(x) + a_n x^n$$

- Solução recursiva usando a primeira hipótese de indução

```
Polynomial_Evaluation(A, n, x){
  if (n == 0)  P = A[0]
  else{
    P = Polynomial_Evaluatio(A,n-1,x)
    xn = 1
    for i=1 to n do xn = xn * x //xn guarda valor de xⁿ
    P = P + A[n]*xn
  }
  return (P)
}
```

- Análise de complexidade

$$T(n) = T(n-1) + \Theta(n)$$
$$T(n) = \Theta(n^2)$$

Segunda solução

- Muito tempo gasto para calcular x^n
  - Podemos calcular x^n em tempo constante a partir de x^{n-1}
  - Reforçar a hipótese indutiva para incluir x^{n-1}
- Hipótese de indução mais forte (segunda tentativa)
  - Para um dado $n > 0$, uma dada sequência de números reais $a_{n-1}, \ldots, a_0$ e um número real x, sabemos calcular x^{n-1} e o valor de

$$P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

  - Caso base: $n = 0$, a solução é $(a_0, 1)$.
  - Passo de indução: Considerando que sabemos como calcular $P_{n-1}(x)$ e x^{n-1}, para obter $P_n(x)$, basta calcular

$$P_n(x) = P_{n-1}(x) + a_n x x^{n-1}$$

- Solução recursiva usando a hipótese de indução mais forte

```
Polynomial_Evaluation(A, n, x){
  if (n == 0){
    P = A[0]
    xn = 1
  }
  else{
    P,xn = Polynomial_Evaluatio(A,n-1,x)
    xn = xn * x //xn guarda valor de xⁿ
    P = P + A[n]*xn
  }
  return (P,xn)
}
```

- Análise de complexidade

  $T(n) = T(n-1) + 3$ (2 multiplicações e 1 adição)

  $T(n) =$ 2n multiplicações + n adições

  $T(n) = \Theta(n)$

  – A inclusão de um cálculo a mais na hipótese de indução reduziu o custo final do algoritmo.

Terceira solução

- Nas hipóteses de indução anteriores, fazíamos a remoção do último termo a_n para resolver o subproblema P_{n-1}
- É possível criar o subproblema na ordem inversa, removendo o primeiro termo a_0
- Hipótese de indução (ordem inversa)
  - Para um dado $n > 0$, uma dada sequência de números reais $a_n, a_{n-1}, \dots, a_1$ e um número real x, sabemos calcular o valor de
    $$P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_1$$
  - Caso base: $n = 0$, a solução é a_n.
  - Passo de indução: Considerando que sabemos como calcular $P'_{n-1}(x)$, para calcular $P_n(x)$, basta calcular
    $$P_n(x) = xP'_{n-1}(x) + a_0$$

- Solução recursiva usando a terceira hipótese de indução

```
Polynomial_Evaluation(A, n, i, x){
  if (i==0) P = A[n]
  else{
    P' = Polynomial_Evaluation(A,n,i-1,x)
    P = x*P' + A[n-i]
  }
  return (P)
}
```

- Análise de complexidade

  $T(n) = T(n-1) + 2$ (1 multiplicação e 1 adição)

  $T(n) =$ n multiplicações + n adições

  $T(n) = \Theta(n)$

- Essa forma de calcular o valor de polinômios é chamada de Regra de Horner

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = ((\ldots((a_n x + a_{n-1})x + a_{n-2}) \ldots)x + a_1)x + a_0$$

Conferência de cientistas

- Você está organizando uma conferência de cientistas de diferentes disciplinas que deverão vir se houver oportunidade para trocar ideias com outros cientistas. A partir de uma lista de possíveis interações entre os cientistas, gerar uma lista de convidados de tamanho máximo de forma que cada convidado possa interagir com pelo menos outros k convidados.

Subgrafos induzidos maximais

- Seja um grafo não-direcionado $G = (V, E)$. Um subgrafo induzido de G é um grafo $H = (U, F)$ tal que $U \subseteq V$ e F contém todas as arestas de E incidentes nos vértices em U.
- Cada vértice representa um cientista e uma aresta conecta os vértices de 2 cientistas com potencial para interagir na conferência.

Subgrafos induzidos maximais

- Problema: Dado um grafo não-direcionado $G = (V, E)$ e um inteiro k, encontrar um grafo induzido $H = (U, F)$ de G de tamanho máximo de forma que todos os vértices de H tenham grau $\geq k$, ou então identifique que isso não seja possível.

- Solução:
  - Remover vértices de G com grau $< k$, pois estes não farão parte de H. As arestas que incidiam no vértice removido também são removidas, então os graus de outros vértices também podem ser reduzidos.
  - Quando algum outro vértice passa a ter grau $< k$, este também deve ser removido
    - Qual a ordem de remoção? Remover todos os vértices com grau $< k$ primeiro? Remover um vértice com grau $< k$ e prosseguir com os vértices afetados pela remoção?
  - Ao invés de pensar na sequência de passos do algoritmo, pensar em como mostrar que um algoritmo que resolve o problema existe para se obter um ideia de como projetar o algoritmo

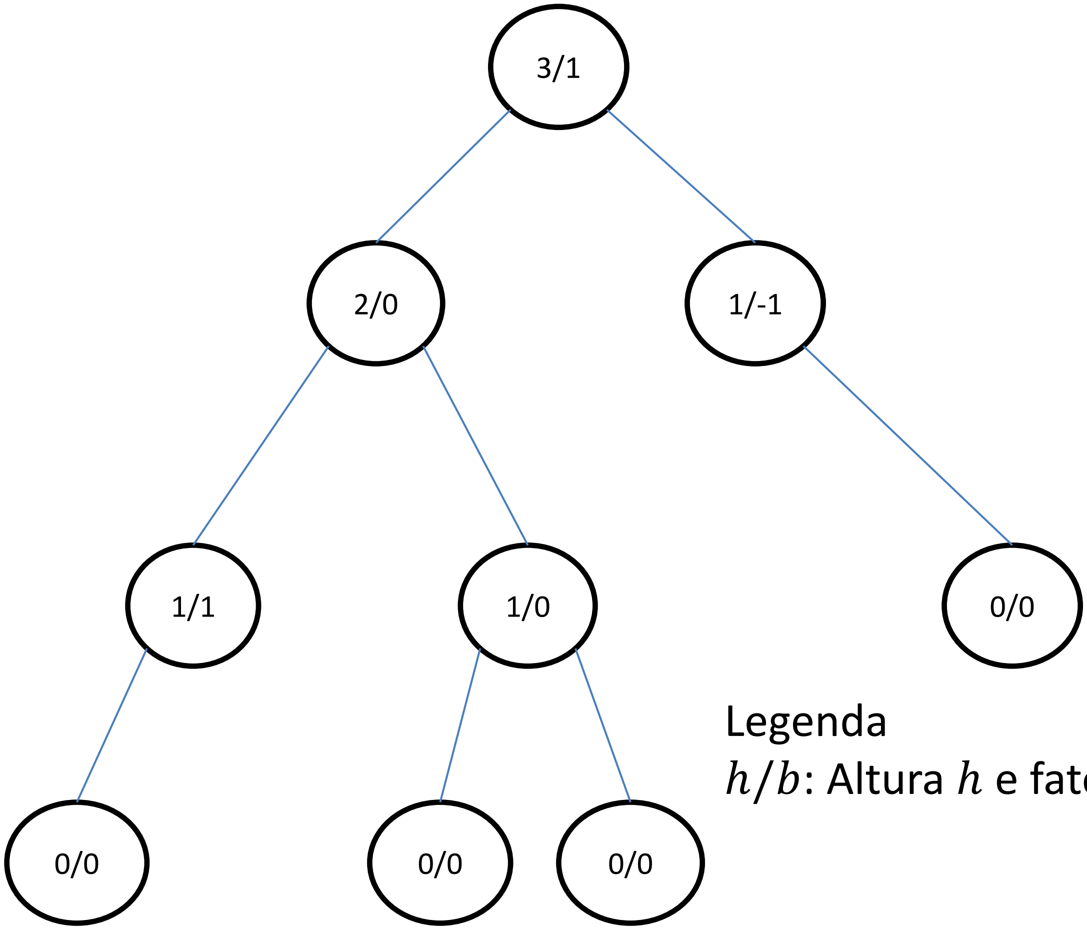
- Hipótese de indução
  - Sabemos como encontrar subgrafos induzidos maximais cujos vértices possuem graus $\geq k$ em um grafo com número de vértices $< n$.
- Caso base: O menor valor de n que faz sentido buscar tais subgrafos é $n = k + 1$
  - Para $n \leq k$, não é possível os graus de todos os vértices são $< k$.
  - Para $n = k + 1$, a única forma de se obter um subgrafo induzido com grau mínimo k é ter um grafo completo, ou seja, todos os vértices estão conectados.
- Passo:
  - Se todos os vértices possuem grau $\geq k$, então o grafo inteiro satisfaz o critério e a solução é o próprio G.
  - Caso contrário, existem algum vértice v com grau $< k$ e, portanto, este vértice permaneceria com grau $< k$ em qualquer subgrafo induzido de G. Então, podemos remover o vértice v e todas as suas arestas adjacentes sem afetar as condições do teorema. Após a remoção, o grafo possui $n - 1$ vértices e, por hipótese de indução, sabemos como resolver o problema.

- Solução recursiva

```
MaximalInducedSubgraph(G, k){
  if (n<k+1) H = Ø
  else if(every vertex of G has degree >= k)
    H = G
  else{
    v = vertex wih degree < k
    H = MaximalInducedSubgraph(G-v, k)
  }
  return (H)
}
```

Fatores de balanceamento em árvores binárias

- Altura de um nó
  - Comprimento do caminho mais longo do nó até as folhas
- Fator de balanceamento de um nó
  - Diferença entre a altura da subárvore esquerda do nó e a altura da subárvore direita do nó.



Legenda
h/b: Altura h e fator de balanceamento b

Fatores de balanceamento em árvores binárias

- Problema: Dada uma árvore binária T de n nós, calcular os fatores de balanceamento de todos os nós.
- Hipótese de indução: Sabemos como calcular os fatores de balanceamento de todos os nós em árvores com $< n$ nós.
- Caso base: $n = 1$, é trivial.
- Para $n > 1$ nós, podemos remover a raiz e resolver os subproblemas para as duas subárvores por indução.
  - Fator de balanceamento de um nó depende apenas dos nós abaixo dele
- Como calcular o fator de balanceamento da raiz com os fatores de balanceamento dos nós filhos?

Fatores de balanceamento em árvores binárias

- Para calcular o fator de balanceamento da raiz, precisamos saber as alturas dos nós filhos!
  - Solução: reforçar a hipótese de indução
- Hipótese de indução mais forte: Sabemos como calcular os fatores de balanceamento e alturas de todos os nós em árvores com $< n$ nós.
- Caso base: $n = 1$, é trivial.
- Passo:
  - Utilizando a hipótese de indução, posso calcular as alturas e os fatores de balanceamento de todos os nós das subárvores da raiz.
  - Fator de balanceamento: sabendo as alturas dos nós filhos (h_e e h_d) , podemos calcular o fator de balanceamento da raiz (f) de forma simples, $f = h_e - h_d$.
  - Altura: sabendo as alturas dos nós filhos (h_e e h_d), podemos calcular a altura da raiz (h) da seguinte forma: $h = \max(h_e, h_d) + 1$.

- Solução recursiva

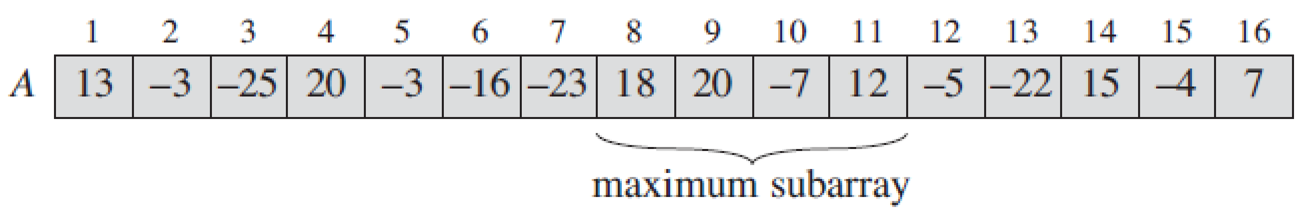
```
FatorBalanceamento(root){
 //entrada: Árvore binária com n nós e raiz root
 //retorno: altura e fator de balanceamento da raiz
  if (n==0){ root.h = -1; root.f = 0;
  else{
    he, fe = FatorBalanceamento(root.left);
    hd, fd = FatorBalanceamento(root.right);
    root.f = he - hd;
    root.h = max(he,hd)+1;
  }
  return (root.h, root.f)
}
```

Subarranjo Máximo

- Problema: Dada uma sequência $a_1, a_2, \ldots, a_n$ de números reais (não necessariamente positivos), encontrar um subarranjo contíguo tal que a soma dos seus valores seja máxima entre todos os possíveis subarranjos.
- Ex: 2, -3, 1.5, -1, 3, -2, -3, 3

  Subarranjo máximo: 1.5, -1, 3

  Soma = 3.5



Subarranjo Máximo

- Entrada com apenas números positivos
  - A solução é toda a sequência
- Entrada com apenas números negativos
  - A solução é 0 (subsequência vazia)

- Solução por força-bruta
  - Número de subarranjos é quadrático em relação ao tamanho da sequência, então calcular a soma de todos subarranjos custa pelo menos $\Theta(n^2)$
- Solução por divisão e conquista
  - Dividir o problema ao meio e resolver recursivamente
    - Solução (subarranjo máximo) poderia estar na primeira metade, na segunda metade ou entre as duas partes (início na primeira e término na segunda)
  - Custo de $\Theta(n\log n)$
- Projeto por indução
  - Como poderia ser feito o projeto por indução de outro algoritmo para resolver esse problema?

Subarranjo Máximo

- Hipótese de indução: sabemos como encontrar o subarranjo máximo em sequências de tamanho $< n$.
- Caso base:

  $n = 1$: Se o valor de a_1 for positivo, o subarranjo máximo contém esse valor, ou é vazio, caso contrário.

Subarranjo Máximo

- Passo:
  - Seja uma sequência $S = (a_1, a_2, \dots, a_n)$ de tamanho $n > 1$. Por H.I., sabemos encontrar o subarranjo máximo em $S' = (a_1, a_2, \dots, a_{n-1})$
    - Caso a solução para S' seja vazia, então todos os seus valores são negativos, então basta verificar a_n
    - Senão, a solução é da forma $S'_M = (a_i, a_{i+1}, \dots, a_j)$, para $1 \leq i \leq j \leq n-1$. Então,
      - Se $j = n - 1$ (o subarranjo máximo é um sufixo)
        - » Se a_n>0
          - então podemos estender a solução com a_n,
          - senão S'_M é o subarranjo máximo de S'.
      - Senão, $j < n - 1$ e temos 2 casos
        - » S'_M continua máximo
        - » Existe outra subsequência que não é máxima em S', mas é máximo em S adicionando-se a_n.

- A primeira tentativa de H.I. não pode ser aplicada para estender uma subsequência que não é máxima em S'.
  - Essa H.I. não é suficiente!
  - Ex: $S = (2, 3, -6, 2, 5)$
    - $S' = (2, 3, -6, 2)$
    - $S'_M = (2,3)$
    - Solução para S é $(2, 5)$.
- Como podemos reforçar a H.I.?

Subarranjo Máximo

- Hipótese de indução mais forte: sabemos como encontrar o subarranjo máximo e o subarranjo que é máximo entre os sufixos em sequências de tamanho $< n$.
- Caso base:

  $n = 1$: Se o valor de a_1 for positivo, o subarranjo máximo e o sufixo máximo contêm esse valor, ou são vazios, caso contrário.

- Passo:
  - Por H.I., sabemos qual é o subarranjo máximo (S'_M) e o sufixo máximo (S'_X) em S', então
    - Se adicionar a_n em S'_X gera uma soma maior que a de S'_M, então o subarranjo máximo de S é formado pela extensão do sufixo
    - Senão, S'_M é o subarranjo máximo de S
      - Sufixo máximo de S?
        - » Se a extensão de S'_X com a_n resultar em soma negativa, o sufixo máximo de S é vazio

- Solução recursiva

```
MaxSubarray(A,n){
 //entrada: Arranjo A e o tamanho n do subarranjo de A
 //retorno: subarranjo máximo (maxSeq) e sufixo máximo (maxSuf) de A[1..n]
  if (n==1)
    if(A[1] < 0){ maxSeq = 0;    maxSuf = 0;    }
    else        { maxSeq = A[1]; maxSuf = A[1];}
  else{
    maxSeq, maxSuf = MaxSubarray(A,n-1);
    maxSuf = maxSuf+A[n];
    if(maxSuf>maxSeq)
      maxSeq = maxSuf;
    else if(maxSuf<0) maxSuf = 0;
  }
  return (maxSeq, maxSuf);
}
```

Qual a complexidade desta solução?

Exercício

1) Dada uma árvore binária *T*, projete um algoritmo para decidir se *T* é ou não uma árvore binária de busca. A resposta deve ser sim ou não.

a) Faça o projeto do algoritmo por indução.

b) Escreva o pseudo-código da solução recursiva obtida.

Referências

- Manber, Introduction to Algorithms – A creative approach, 1st ed.
- – Cap. 5