UNIVERSIDADE FEDERAL DE SÃO PAULO
CAMPUS SÃO JOSÉ DOS CAMPOS
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA (DCT)

Exercícios de Projeto e Análise de Algoritmos
Lista 4 e 5

UC: **Projeto e Análise de Algoritmos**

Aluna: **Thauany Moedano**

RA: **92486**

Professor: **Dr. Reginaldo Massanobu Kuroshu.**

Entrega: **12/10/2015**

Resumo

Resolução da lista 4 de exercícios da aula de Projeto e Análise de Algoritmos no 2º semestre de 2015.

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.

Resolução Para projetar por indução devemos seguir três passos: Caso baso, hipótese indutiva e passo indutivo.

Hipótese de indução: Para um dado número de nós k ≥ 0 de uma árvore qualquer, sabemos verificar se esta é uma árvore binária de busca observando se seus filhos alocados à esquerda são menores que a raiz de cada subárvore e que seus filhos alocados a direita são maiores que a raiz de cada subárvore. Queremos mostrar para k+1 nós que sabemos verificar se é uma árvore binária.

Caso base: Um nó folha não possui filhos e portanto é uma árvore binária.

Passo indutivo: Temos uma árvore que sabemos verificar se é binária ou não. Alocando um nó a mais, este nó é um nó folha e portanto sabemos que a subárvore que contém o nó alocado como raiz é folha e portanto uma árvore binária. Como também podemos verificar para k nós se seus filhos a direita são maiores e os filhos a esquerda são menores, é possível saber se o nó alocado foi colocado no lugar corretamente. Portanto, é possível verificar se uma árvore com k+1 nós é binária ou não.

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

```
int verificaABB(ArvBin T) {
   if(T->esq == NULL && T->dir == NULL)
      return(1);
   else if(T->dir.Valor < T || T->esq.Valor > T)
      return(0);
   else {
      return(verificaABB(T->dir) && verificaABB(T->esq));

   }
}
```

2. Seja P: N → N uma função definida da seguinte forma: P (0) = P (1) = P (2) = P (3) = P (4) = 0 e, para n ≥5,

$$P(n) = P(\frac{n}{2}) + P(\frac{n}{2}+1) + P(\frac{n}{2}+2) + n$$

a) Projete um algoritmo recursivo que recebe um número n como entrada e retorna o valor de P (n).

Resolução:

```
int Polinomio(int n) {
   if(n < 5) {
      return(0);
   else {
      return(Polinomio(n/2) + Polinomio(n/2+1) + Polinomio(n/2+2) + n);
   }

}
```

b) Escreva a versão deste algoritmo recursivo com memorização.

```
int Polinomio(int n, int v[]) {
   if(v[n] != -1) //Inicialmente o vetor e preenchido com -1 quando ainda
       nao ha solucao para o polinomio
      return(v[n]);
   else if(n < 5)
      v[n] = 0;
   else
      v[n] = (Polinomio(n/2) + Polinomio(n/2+1) + Polinomio(n/2+2) + n);
   return (v[n]);

}
```

c) Escreva o algoritmo de programação dinâmica bottom - up para calcular P (n).

```
int Polinomio(int n, int v[]) {
   for(int k = 0; k < n; k++)
      if(k < 5)
```

```
      v[k] = 0;
    else
      v[k] = (v[k/2]) + v[k/2 + 1] + v[k/2 + 2] + k);

  return v[n];
}
```

3. Considere o problema de retornar n centavos de troco com o número mínimo de moedas. a) Seja a ser retornada e o seguinte D = (25, 10, 5, 1) o conjunto de diferentes valores de moedas disponíveis, para esse conjunto de possíveis moedas, é possível projetar um algoritmo guloso que encontre a quantidade mínima de moedas para retornar o troco T ? Justifique. Caso possível, forneça um algoritmo guloso que encontre a solução ótima.

Resolução: Sim pois este conjunto de troco possui dois elementos que fazem com que o algoritmo guloso funcione:

Subestrutura ótima: As soluções ótimas do problema incluem soluções ótimas de subcaso. Para um dado subcaso de tamanho k ($k \leq n$, n o tamanho do problema), a solução para k+1 inclui a solução ótima de k incluindo um novo elemento. Assim, estendendo as soluções é possível encontrar a solução de n.

Propriedade gulosa: De acordo com pesquisas e um artigos, conjuntos que possuem a **propriedade canônica** sempre funcionam com a escolha gulosa. Como o conjunto (25,10,5,1) tem essa propriedade, podemos afirmar que esta a escolha gulosa sempre leva ao melhor resultado.

```
int retornaTroco(int D[], int valorTroco) { //O conjunto D de troco e
   passado em ordem descrecente
  int soma = 0;
  int min = 0;
  while(soma < valorTroco) {
    for(int i = 0; i < D.size; i++) {
      if(D[i]+soma <= valorTroco) {
        soma += D[i];
        min++;
        break;
      }
    }
  }
  return(min);
}
```

b) Forneça um outro conjunto D para o qual o algoritmo guloso não encontra a solução ótima. Mostre um caso para o qual este algoritmo falha.

Resolução: D = (5,4,1) para um troco de 8 centavos. O algoritmo guloso escolheria a solução (5, 1, 1, 1) quando a melhor solução seria (4,4)

Outro exemplo é D = (25,10,1). O algoritmo guloso retorna (25,1,1,1,1,1) enquanto a resposta ótima seria (10,10,10).

c) Projete um algoritmo por programa ção dinâmica qu e encontra a quantidade mínima de moedas necessárias para devolver n centavos de troco para qualquer conjunto D que inclua a moeda de 1 centavo.

Para projetar um algoritmo por PD para o problema do troco precisamos pensar na recursão do problema. Seja uma matriz C[D.size][N] cujo N é o valor de troco que gostaria de ser recebido, podemos montar a seguinte recursão:

C[i][j] = 0, se j = 0. //Se o valor do subproblema é zero não há o que retornar de troco (caso base)

C[i][j] = j, se D_i = 1 //Se no conjunto de troco tem apenas a moeda de um, o valor do troco vai ser ele mesmo (caso base)

C[i][j] = C[i-1][j], se j $\leq D_i$ //Caso o valor da moeda exceda o troco do subproblema não é possivel inclui-la na solução

C[i][j] = min(C[i-1][j], 1+C[i][j-D_i], para outros casos //Escolhemos se vale a pena incluir a moeda na solução ótima

Desta maneira basta construir o algoritmo que preencha a matriz:

```
  int troco(D[], N, C[]) {
    for(i = 1; i < D.size; i++) {
      C[i][0] = 0;
      C[0][i] = 0; //Caso 1
    }

    for(j = 1; j < N; j++)
      C[1][j] = j; //Caso 2

    for(i = 2; i < D.size; i++) {
      for(j = 1 j < N; j++) {
        if(j < D[i]) {//Caso 3
          C[i][j] = C[i-1][j];
        else
          C[i][j] = min(C[i-1][j], 1+C[i][j-D[i]]) //Caso 4
      }
    }

  return(C[D.size][N]);
}
```

4. Seja um dado conjunto S = a 1 , a 2 , ..., a n de tarefas, onde a i requer p i unidades de tempo para ser finalizado, uma vez iniciado. Seja c i o tempo de término da tarefa a i , o objetivo é mini mi zar a média dos tempos de término das tarefas. Por exemplo, seja p 1 ,=3 p 2 ,=5. Se realizarmos as tarefas na sequência a 2 e depois a 1 , então te mpos c 2 = 5 e c 1 = 8, logo o tempo médio é 6,5. Caso a sequência seja na ordem a 1 e depois a 2 , então tempos c 1 = 3 e c 2 = 8, logo o tempo médio é 5,5.

a) Forneça um algoritmo guloso que minimize o tempo médio de término das tarefas.

```
  //A struct eh um resgistro que contem os valores de $P_i$ e $C_i$ e eh
     passado ordenado por $P_i$

  int MinimizaTempo(Struct Tarefa, int qntTarefas) {
```

```
    int Ordem[qntTarefas];
    Ordem[0] = Struct.P[0];
    int media = 0;

    for(int i = 1; i < qntTarefas; i++) {
      Ordem[i] = Ordem[i-1] + Tarefa.P[i];
    }

    for(int k = 0; k < qntTarefas; k++)
      media += Ordem[k];

    media = media/qntTarefas;
    return(media);
  }
```

b) Prove que esse algo ritmo obtém a solução ótima.

Temos que novamente provar dois aspectos de algoritmo guloso: **Propridade Gulosa e Subestrtutura Ótima**.

Subestrutura Ótima: Para um dado número de atividades podemos escolher a solução ótima para aquele problema. Para estender a solução basta incluir uma nova atividade a anterior.

Propriedade Gulosa: Como queremos obter a maior média, é interessante escolher sempre a atividade que gasta menos tempo. Assim a próxima atividade espera menos tempo para ser concluída e melhora a média de atividades.

5. 5. Para cada uma das afirmações abaixo, diga se ela é verdadeira, falsa, verdadeira se P $\neq$ NP ou falsa se P $\neq$ NP. Justifique suas respostas.

a) Não há problemas em P que são NP - completos.

É **verdadeiro somente se P $\neq$ NP.** Um problema para ser NP-Completo deve estar em NP. Portanto, um problema P só pode não ser NP-Completo se ele for $\neq$ de NP.

b) Existem problemas em P que estão em NP.

Verdadeiro pois se conseguimos resolver o problema em tempo polinomial (Conceito dos problemas P), podemos verificar uma solução em tempo polinomial também (Conceito dos problemas NP).

c) Existem pro blemas em NP que não estão em P.

Verdadeiro se P $\neq$ NP. Isto seria equivalente a dizer que problemas em NP não possuem soluções em tempo polinomial. Se um problema desse existir, então P $\neq$ NP.

d) Se A pode ser polinomialmente reduzido a B e B é NP - completo, então A é NP - completo.

Falso se P$\neq$NP Pois fazendo a redução de A para B, determina-se um limite superior para A. Se B não é resolvido em tempo polinomial (suponha um limite O(x) para B) e a transformação ocorre em tempo polinomial P'(x), um limite **superior** para A é O(O(x) + P'(x). Entretanto não temos como garantir o limite inferior não terá execução polinomial a menos que P = NP.

e) Se A pode ser polinomialmente reduzido a B e B $\in$ P, então A $\in$ P

Verdadeiro pois determina-se um limite superior para A. Como B $\in$ P, o limite superior de B é P(n). Como a redução de A para B ocorre em tempo polinomial P'(n).

O limite superior de A é O(P'(n) + P(n)) o que continua sendo polinomial e $\in$ P.

6. Seja Independent Set (IS) o problema de decidir se existe um conjunto independente de no máximo k vértices em um grafo G . Um conjunto de vértices S do grafo G é independente se não existe uma aresta (x , y) onde x $\in$ S e y $\in$ S , ou seja, não existe nenhuma aresta entre qualquer par d e vértices em um conjunto independente. Mostre que IS está em NP - completo. Dica: tente reduzir Vertex Cover para IS.

Para mostrar que IS está em NP-Completo devemos seguir os passos:

Mostrar que IS está em NP: Para verificar se um conjunto de vértices é uma solução de IS, para cada vértice k, teríamos que conferir quem são as arestas ligadas a k, o que daria em torno de O(k^2). Como a solução de IS é verificável em tempo polinomial, IS está em NP.

Selecionar um problema NP-completo conhecido: Selecionaremos o problema Vertex Cover.

Construir uma transformação de Vertex Cover para IS: Dado um grafo G(V,E) e uma constante k, o Vertex Cover procura o menor conjunto de vértices $\in$ V em que cada aresta e $\in$ E contém pelo menos um vértice de S e que o conjunto tenha pelo menos k vértices. Basta notar que se existe um subconjunto S' com |V|-k vértices que é um *Independent Set*, então também existe um conjunto S com k vértices que é um Vertex Cover. Isso porque Vertex Cover é um problema complementar a Independent Set.

Provar que a Redução é polinomial: A redução apenas altera a entrada k' de Independent Set para k' = |V| - k, o que é constante.