Lista de exercícios
Prof. João B. Oliveira

1 Algoritmos gulosos

1. Escreva um algoritmo que resolve o problema de achar o menor número de moedas para dar troco, em um país que tem moedas de 11, 5 e 1 centavos e depois confirme que ele funciona corretamente (tente o troco para 15 centavos). Provavelmente seu algoritmo **não será** guloso.

2. Explique por que o algoritmo de SelectionSort é um algoritmo guloso.

3. Um novo algoritmo para encontrar Árvores de Cobertura Mínima em um grafo G é assim:

   (a) Inicie em algum nodo u de G;

   (b) Siga a aresta de valor mais baixo ligada a u e que leva a um nodo ainda não visitado;

   (c) Repita o passo anterior até visitar todos os nodos de G.

   Agora responda duas perguntas:

   (a) Ele é guloso?

   (b) Ele acha uma Árvore de Cobertura Mínima?

4. Você tem uma série de atividades bem legais que podem ser realizadas no sábado, mas todas tem hora marcada para início e fim. Você gostaria de escolher o máximo de atividades possíveis para fazer, já sabendo que quando iniciar uma delas terá que ir até o final sem mudar para alguma outra. Você pensa em alguns algoritmos para escolher o máximo de atividades:

   (a) Escolher as atividades dando preferência para as mais curtas (por que ocupam o mínimo de tempo);

   (b) Escolher as atividades dando preferência para as que terminam o mais cedo possível;

   Teste seus dois algoritmos com as atividades abaixo:

Atividade	Hora Início	Hora Fim
1	2	4
2	1	4
3	2	7
4	4	8
5	4	9
6	6	8
7	5	10
8	7	9
9	7	10
10	8	11

5. Apresente um algoritmo guloso e de tempo $O(n)$ que opera sobre uma lista de inteiros $[a_1, a_2, a_3, \ldots, a_n]$ e encontra o local onde a soma dos números de um lado da sequencia é igual à soma dos números do outro lado (ou o mais parecida possível). **Seu algoritmo tem que respeitar uma regra extra: cada elemento da lista pode ser acessado somente uma vez.**

6. Uma coloração de um grafo não dirigido é a atribuição de cores aos nodos de forma que nenhum par de nodos vizinhos tenha a mesma cor. No problema de coloração de grafos, desejamos encontrar o número mínimo de cores necessárias para colorir um grafo não dirigido. Proponha um algoritmo guloso para resolver o problema e comprove que ele sempre funciona (se você conseguir, pode ganhar um milhão de dólares).

2 Divisão e conquista

1. Implemente pesquisa binária sobre um vetor de inteiros.

2. Teste sua pesquisa binária:

   (a) Crie um vetor de 20 inteiros inicializado com $(0, 1, 2, 3, 4, 5, 6, \ldots, 19)$.

   (b) Para cada um dos 20 elementos, teste sua pesquisa binária para ver se ela encontra o elemento corretamente.

3. Modifique sua pesquisa binária para que ela seja **ternária** e teste outra vez.

4. Implemente Mergesort, dividindo o vetor que deve ser ordenado em duas partes (Mergesort convencional).

5. Modifique seu Mergesort para que ele divida o vetor em três partes.

3 Programação dinâmica

1. Você pode construir uma calçada da sua casa até a casa da vovó usando pedras verdes, azuis e amarelas. As pedras são quadradas, tem 1 metro de largura e 1 metro de comprimento, e a casa da vovó fica a 50 metros de distância. Antes de comprar as pedras na loja de material de construção, você gostaria de responder a estas três perguntas:

   (a) Quantas calçadas com coloridos diferentes são possíveis?

   (b) Quantas calçadas com coloridos diferentes são possíveis, se a vovó não quer que duas pedras amarelas fiquem uma ao lado da outra?

   (c) ... e se a vovó também não deixa que duas pedras azuis fiquem lado a lado?

   Você decide encarar esse problema com uma recursão simples e (se não der certo) pode tentar uma versão com memorização para acelerar as coisas.

4 Backtracking

1. Escreva um algoritmo que seja capaz de resolver o problema das 4 rainhas usando *backtracking*. Altere seu algoritmo para que ele seja capaz de resolver o problema para um número qualquer de rainhas.

2. Adapte o algoritmo produzido para o problema 1 para produzir outro algoritmo que resolva o problema do passeio do cavalo no xadrez. Procure uma descrição deste problema em `http://en.wikipedia.org/wiki/Knight%27s_tour` e tente fazer versões que achem caminhos abertos (mais fácil) e caminhos fechados (mais difícil).

3. Escreva um algoritmo que seja capaz de resolver um Sudoku como o que está abaixo, usando *backtracking*. Preveja a entrada e saída de seu algoritmo.

	2		6		8			
5	8				9	7		
				4				
3	7					5		
6								4
		8					1	3
				2				
		9	8				3	6
			3		6		9	

4. Dado um conjunto de inteiros $S = \{a_1, a_2, \ldots, a_n\}$ escreva um algoritmo baseado em *backtracking* capaz de testar se existe um subconjunto de S que tenha a soma igual a um outro inteiro k.

5. Escreva um algoritmo baseado em *backtracking* que recebe um inteiro n e produz uma sequencia contendo todos os inteiros de 1 a n com a condição de que a soma de um inteiro com o seguinte na sequencia seja sempre um quadrado perfeito. Por exemplo, ao receber o número $n = 15$ seu algoritmo poderia produzir

```
8 1 15 10 6 3 13 12 4 5 11 14 2 7 9
```