Algoritmos e Estruturas de Dados II
Lista de Exercícios

Exercícios do livro Introduction to Algorithms: 16.1-2, 16.2-5, 23.1-1, 23.1-5, 23.2-5, 24.3-3.

Questão 1

1. Estrutura de dados que permite extrair rapidamente o menor elemento, usada em Prim e Dijkstra.
2. Processo de armazenar resultados de sub-problemas já resolvidos para evitar recomputações.
3. Critério que permite afirmar que sempre existe uma solução ótima que começa com a melhor escolha local.
4. Algoritmo que enumera candidatos recursivamente e retrocede quando encontra um beco sem saída; é uma técnica que explora o espaço de soluções de forma exaustiva, mas abandona caminhos inviáveis precocemente.
5. Método que tenta todas as possibilidades, sem qualquer técnica de poda ou heurística; é particularmente útil para problemas pequenos e simples de implementar, que sempre encontram uma solução se ela existe.
6. Estrutura que representa todas as configurações possíveis de um jogo de dois jogadores por turnos.
7. Um algoritmo "míope" que toma decisões localmente ótimas sem considerar as consequências futuras.
8. Estratégia de avaliação de árvores de jogo que busca maximizar o ganho do jogador atual e minimizar o ganho do oponente.
9. Técnica usada para compactar dados, atribuindo códigos curtos a símbolos frequentes e longos a símbolos raros.
10. Uma otimização em árvores de jogo que permite descartar ramos sem a necessidade de explorá-los por completo.
11. Uma estratégia de poda em árvores de jogo que explora a relação $T(x,MAX) = -T(x,MIN)$ para configurações simétricas.

Termos:

A) Backtracking

B) Código de Huffman

C) Força Bruta

D) Propriedade da Escolha Gulosa

E) Guloso

F) Heap Binário

G) Max-Min

H) Memorização (ou Sobreposição de Sub-Problemas)

I) Poda Alfa-Beta

J) Poda por Simetria

K) Árvore de Jogo

Questão 2

Considere 12 arquivos cujos tamanhos (em MB) são fornecidos em um vetor T. O custo de mesclar dois arquivos é a soma de seus tamanhos, e o arquivo resultante tem tamanho igual a essa soma. O objetivo é mesclar todos os arquivos até restar apenas um, minimizando o custo total acumulado.

A) Proponha um algoritmo guloso para resolver o problema.

B) Analise a complexidade temporal e espacial do seu algoritmo.

C) Demonstre que o algoritmo é ótimo (ou forneça um contra-exemplo).

D) Escreva um algoritmo de backtracking que explore todas as ordens de mesclagem possíveis e devolva o custo mínimo. Utilize poda por limite superior para acelerar a busca.

E) Compare, em um breve parágrafo, o desempenho e a qualidade das soluções obtidas pelos métodos guloso e backtracking para instâncias aleatórias.

Dica: O critério guloso clássico consiste em sempre mesclar os dois menores arquivos restantes (similar à construção de Huffman).

Questão 3

Modifique o backtracking gerador de subconjuntos. Utilize o algoritmo base abaixo, que imprime todos os subconjuntos de V[1..n], e faça APENAS as modificações mínimas para que:

A) Sejam impressos apenas os subconjuntos em que (número de elementos pares) <= (número de elementos ímpares).

B) Sejam impressos apenas os subconjuntos em que (número de elementos ímpares) = ½ x (número de elementos pares).

```
GeraSub(ns, t):
  para i ← t .. n incl.:
      S[ns] ← V[i]
      escreve(S)
      se i < n então
          GeraSub(ns+1, i+1)

ler n
para i ← 1 .. n incl.:
    ler V[i]
GeraSub(1, 1)
```

Questão 4

Construa uma árvore de Huffman para os símbolos a seguir e suas frequências:

A = 11

B = 24

C = 16

D = 18

E = 9

A) Apresente a árvore resultante.

B) Escreva o código binário atribuído a cada símbolo.

C) Calcule o tamanho médio da codificação (bits/símbolo).

Questão 5

Considere o problema da Mochila 0/1: Dada uma lista de n itens, cada um com um peso w_i e um valor v_i associado, e uma mochila com uma capacidade máxima de peso W, o objetivo é selecionar os itens a serem colocados na mochila de modo a maximizar o valor total, respeitando a capacidade da mochila. Cada item pode ser escolhido ou não escolhido (não pode ser fracionado).

A) Escreva um algoritmo **guloso** para solucionar o problema, baseando-se na ideia de "valor por unidade de peso". Ou seja, calcule v_i/w_i para cada item e ordene os itens com base neste valor.

B) Analise a complexidade de tempo do algoritmo proposto no Item A).

C) Prove ou dê um contra-exemplo de que o algoritmo proposto no Item A) produz uma solução ótima para o problema da Mochila 01.

D) Escreva um algoritmo de **backtracking** para solucionar o problema da Mochila 0/1. A cada passo, decida se inclui o item atual ou não, e retroceda se a capacidade for excedida.

E) Compare o desempenho, a eficiência e os resultados obtidos pelos algoritmos propostos usando o método guloso (Item A)) e backtracking (Item D)) para o problema da Mochila.

Questão 6

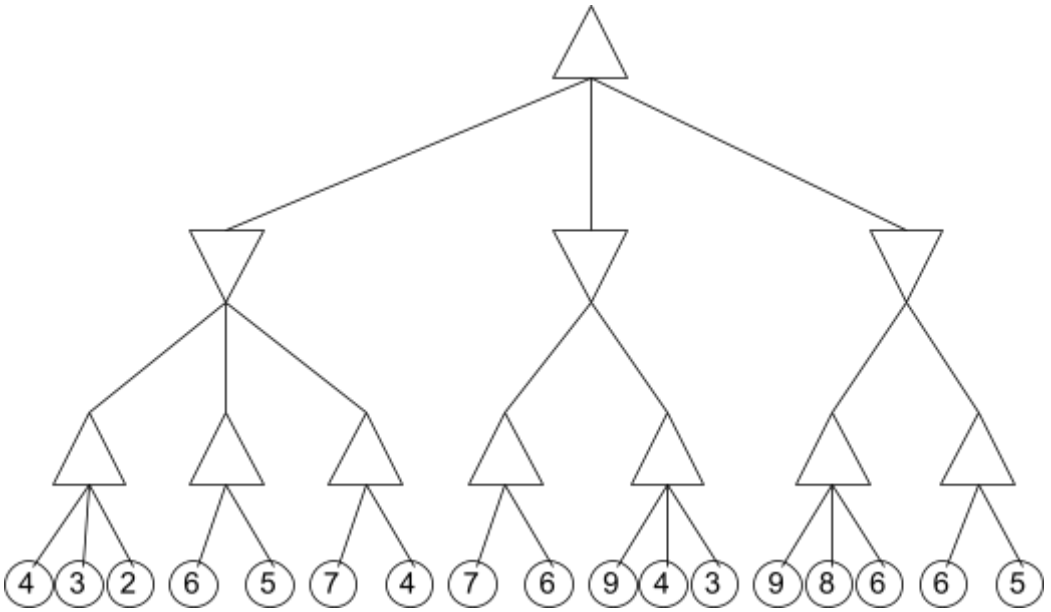
Explique a diferença entre resolver e verificar um problema de decisão. Qual é o papel do “certificado”? Por que essa distinção leva à definição da classe NP? Inclua um exemplo concreto dos slides (por exemplo, mostrar que um subconjunto soma S em SUBSET-SUM).

Questão 7

Descreva, em linguagem acessível, o que significa “reduzir A a B em tempo polinomial”.

Questão 8

A árvore de jogo a seguir mostra dois jogadores, MIN e MAX, jogando, onde os triângulos apontando para baixo denotam decisões tomadas por MIN, os triângulos apontando para cima denotam decisões tomadas por MAX e as folhas mostram a pontuação final.



A) Preencha na árvore os valores para todos os nós MAX e MIN, de acordo com o algoritmo visto em aula. Lembrando que os triângulos apontando para cima são nós MAX, enquanto os triângulos apontando para baixo são nós MIN.
B) Qual o valor que MAX espera alcançar?
C) Qual(is) nó(s) seria(m) podado(s) pela poda alfa-beta?

Questão 9

Escreva um algoritmo de backtracking para resolver o seguinte problema: dada uma string de entrada não vazia s e um dicionário dict[] contendo uma lista de palavras não vazias, o objetivo é encontrar e retornar todas as possíveis frases que podem ser formadas pela quebra de s em sequências de palavras do dicionário. É necessário que cada palavra resultante da quebra deve ser uma palavra válida contida no dicionário dict. Além disso, as palavras no dicionário podem ser utilizadas zero ou mais vezes. Cada quebra de palavra válida deve ser retornada como uma string, com as palavras separadas por um único espaço. Exemplo: Entrada s = “gatosecao”, dict = ["gato", "gatos", "e", "cao"]. Saída: "gato e cao", "gatos e cao"

Questão 10

Escreva um algoritmo de backtracking para resolver o seguinte problema: dada uma string não vazia s composta apenas por dígitos, determine se ela pode ser particionada em uma sequência de pelo menos três substrings numéricas tal que, a partir da terceira substring, cada número seja a soma dos dois números imediatamente anteriores. Como restrição, considere que a sequência deve ter um tamanho N >= 3 e todas as substrings numéricas, exceto o número 0 (zero) isolado, não podem conter zeros à esquerda. Exemplo: s = "12243660" pode ser particionada como [12, 24, 36, 60], onde 36 = 12 + 24 e 60 = 24 + 36. (Resultado: true)

Questão 11

Escreva um algoritmo guloso para resolver o seguinte problema: Você recebe uma fechadura composta por n anéis circulares diferentes, e cada anel possui dígitos de 0 a 9 impressos em série. Inicialmente, todos os n anéis juntos exibem um número inteiro de n dígitos, mas existe apenas um código específico que pode abrir a fechadura. Você pode girar cada anel quantas vezes quiser em qualquer direção. O objetivo é encontrar o número mínimo de rotações realizadas nos anéis da fechadura para abri-la.
Exemplo: Entrada = 2345, Código de desbloqueio = 5432
Saída: Rotações necessárias = 8, já que o 1º anel é girado três vezes como 2->3->4->5, o 2º anel é girado uma vez como 3->4, o 3º anel é girado uma vez como 4->3 e o 4º anel é girado três vezes como 5->4->3->2.

Questão 12

Escreva um algoritmo guloso para resolver o seguinte problema: Dada uma coleção de intervalos de tempo, representados como um array onde cada elemento arr[i] = [starti, endi] define um intervalo. O objetivo é mesclar todos os intervalos que se sobrepõem e retornar um novo array contendo apenas intervalos mutuamente exclusivos que cobrem exatamente a mesma duração que os intervalos originais. Por exemplo, se a entrada for [[1, 3], [2, 4], [6, 8], [9, 10]], os intervalos [1, 3] e [2, 4] se sobrepõem e devem ser mesclados em [1, 4], resultando na saída [[1, 4], [6, 8], [9, 10]].