MC658 - Análise de Algoritmos III
Lista de Exercícios de BackTracking e Branch-and-Bound

Instituto de Computação – UNICAMP[1]

1. O problema das n-rainhas consiste em determinar se é possível distribuir n rainhas em um tabuleiro $n \times n$ tal que nenhuma das rainhas se ataquem. Escreva um pseudocódigo de um algoritmo de *backtracking* que imprime todas as soluções possíveis para o problema.

2. Faça a execução do algoritmo de backtracking visto em aula para o problema *Sub-Set Sum* usando como instância $S = \{5, 7, 10, 12, 15, 18, 20\}$ e $W = 35$. Mostre a árvore de execução do algoritmo.

3. Seja S o conjunto de sete números inteiros dado por $S = \{11, 8, 7, 5, 4, 3, 2\}$. Deseja-se saber se S contém um subconjunto S' tal que soma dos seus elementos seja exatamente igual a metade da soma dos elementos de S. Ou seja, trata-se de saber se o **problema da partição** para S tem uma solução e, em caso afirmativo, apresentar um subconjunto S' que responde ao problema.

   Você deve resolver este problema usando um algoritmo de *backtracking* onde: **(i)** as soluções são representadas por tuplas de tamanho variável e, além disso, **(ii)** o valor do k-ésimo elemento da tupla é sempre menor do que o valor dos $(k-1)$ elementos que o antecedem na tupla para todo $2 \leq k \leq 7$.

   Para explicar o seu raciocínio, a execução do algoritmo deve estar representada por meio da porção da árvore de espaço de estados que foi explorada. No interior de cada nó da árvore escreva qual é a solução parcial (tupla) construída até aquele momento. Além disso, coloque ao lado de cada nó da árvore um número que indica a ordem em que ele foi visitado, iniciando com o valor zero para a raiz. O algoritmo pára quando encontrar a primeira resposta ou concluir que ela não existe.

4. Escreva em um pseudocódigo de alto nível, o algoritmo **recursivo** que implementa a solução da questão anterior.

5. Dado um grafo $G = (V, E)$ direcionado e dois vértices u e v, considere o problema de se determinar todos os caminhos de u para v. Escreva um pseudocódigo de um algoritmo de *backtracking* para este problema e explique o funcionamento do algoritmo (que tipo de tupla usada para representar as soluções etc). Argumente sobre a complexidade deste algoritmo.

6. Considere o problema de emparelhamento máximo em grafos bipartidos: Entrada é um grafo bipartido $G = (V_1, V_2, E)$ com peso nas arestas; deve-se achar um emparelhamento que maximize a soma dos pesos das arestas que fazem parte do emparelhamento. Escreva um pseudocódigo de um algoritmo de *backtracking* para este problema e explique o funcionamento do algoritmo (que tipo de tupla usada para representar as soluções etc).

7. Considere o problema 3CNF-SAT com pesos: neste problema temos uma fórmula booleana $f = C_1 \wedge_2 \wedge \ldots \wedge C_m$ sobre um conjunto de variáveis $x_1, \ldots, x_n$, onde cada cláusula é um "ou" de exatamente três literais e possui um peso $p(C_i)$ associado. Deve-se encontrar uma atribuição para as variáveis que maximize a soma dos pesos das cláusulas verdadeiras. Escreva um pseudo-código de um algoritmo de *backtracking* para este problema. Descreva qual tipo de tupla utilizada.

[1] Material preparado por vários docentes que ministraram a disciplina nos últimos anos.

8. Mostre que utilizando a estratégia de melhor limitante (best-bound) para problemas de maximização, a solução devolvida pelo algoritmo de *branch-and-bound* é ótima.

9. Execute o algoritmo de *branch-and-bound* para o problema da mochila considerando a seguinte instância: $n = 5$, $c = (10, 15, 6, 8, 4)$, $w = (4, 6, 3, 4, 2)$ e $W = 12$. Mostre a porção da árvore de espaço de estados que é gerada.

10. Seja $G = (V, E)$ um grafo completo não dirigido de 5 vértices com comprimento nas arestas dados pela matriz (simétrica) D abaixo. Nesta matriz, o elemento d_{ij} representa o comprimento da aresta (i, j).

$$D = \begin{pmatrix} - & 8 & 12 & 14 & 10 \\ - & - & 7 & 4 & 16 \\ - & - & - & 18 & 15 \\ - & - & - & - & 9 \\ - & - & - & - & - \end{pmatrix}$$

Deseja-se encontrar o caminho hamiltoniano de comprimento mínimo neste grafo que tem o vértice $\underline{1}$ como uma de suas extremidades. Para tanto, projetou-se um algoritmo de *branch-and-bound* onde as soluções (parciais ou completas) são representadas por tuplas começando com o número 1. Por exemplo, a tupla $(1, 2, 3)$ representa a solução *parcial* formada pelas arestas $(1, 2)$ e $(2, 3)$. **Note que uma tupla de tamanho 4 já é suficiente para definir por completo um caminho hamiltoniano !**

Lembrando que todo caminho hamiltoniano é também uma árvore geradora do grafo, pode-se usar como função classificadora a árvore geradora mínima do grafo. Na verdade, estando algumas arestas já incluídas na árvore geradora *a priori*, é possivel construir a árvore geradora de comprimento mínimo da seguinte forma:

Passo 1: Seja F as arestas já incluídas na árvore *a priori*.
Passo 2: Ordene as arestas de $(E - F)$ (ou seja, aquelas não incluídas *a priori* !) em ordem crescente do valor dos comprimentos.
Passo 3: Percorra as arestas nesta ordem e a cada iteração, inclua a aresta corrente na solução caso ela não forme um ciclo com as arestas anteriores.
Passo 4: Repita o passo 3 até que $|V| - |F| - 1$ arestas tenham sido aceitas na solução.

Usando a *função classificadora* acima no algoritmo de *branch-and-bound* construa a porção da árvore de espaço de estados explorada pelo algoritmo. Ao lado de cada nó desta árvore indique qual o valor do limitante calculado pela *função classificadora* e no interior deste mesmo nó escreva qual a tupla que corresponde a ele.

Lembre-se: para uma tupla $(x_1, \ldots, x_p)$ as arestas (x_1, x_2), (x_2, x_3), ..., (x_{p-1}, x_p) já devem ser consideradas como parte da árvore calculada pela função classificadora !

11. Considere o seguinte problema de escalonamento: Temos um conjunto J de tarefas, onde cada tarefa $j \in J$ possui um tempo de processamento p_j, tempo de liberação r_j o qual representa o tempo em que a tarefa pode começar a ser executada, e uma previsão de término d_j. Se tivermos por exemplo uma tarefa j com $p_j = 5, r_j = 100, d_j = 110$ significa que a tarefa deve ser executada de forma ininterrupta por 5 segundos, mas sua execução só poderá ocorrer após o tempo 100. A previsão de término da tarefa é o tempo 110.

Devemos escalonar as tarefas em uma única máquina e esta máquina só processa uma tarefa por vez. Em um escalonamento definimos o tempo de término da tarefa j como C_j. Definimos o atraso de uma tarefa como $L_j = C_j - d_j$.

O problema consiste em escalonar as tarefas em uma máquina de tal forma a minimizar o atraso máximo, ou seja, minimizar $L_{max} = \max(L_1, \ldots, L_n)$.

Faça um algoritmo *branch-and-bound* para este problema. Este problema é para o caso não preemptivo, ou seja, após começar a execução de uma tarefa, esta deve ser executada de forma ininterrupta. Em um escalonamento preemptivo, uma tarefa pode ser interrompida e continuada depois.

Para calcular um limitante inferior de um nó na árvore de *branch-and-bound* use o seguinte algoritmo para o caso preemptivo do problema: Dentre as tarefas que podem ser executadas, ou seja, tem seus tempos de liberação menor do que o tempo atual, mantenha em execução a tarefa com menor previsão de tempo de término d_j. Esta regra de execução é um algoritmo ótimo para o caso preemptivo.

12. Para cada um dos problemas abaixo, diga como você representaria uma solução (total ou parcial) usando tuplas e como você computaria um limitante dual (função classificadora ou limitante) em um algoritmo de *branch-and-bound* para cada um dos problemas abaixo. Considere que os grafos dados na entrada são sempre não direcionados.

    (a) Coloração de grafos: encontre o menor número de cores para colorir todos os vértices do grafo de modo que não existam dois vértices adjacentes com a mesma cor.

    (b) Caminho mais longo: encontre o caminho mais longo no grafo (sequencia de arestas que toca cada vértice no máximo uma vez).

    (c) Clique máxima: encontre um subconjunto S de vértices de maior tamanho possível, de forma que todos os vértices em S sejam adjacentes uns aos outros

    (d) Conjunto independente máximo: encontre um subconjunto S de vértices de maior tamanho, de modo que nenhum dos vértices em S seja adjacente um ao outro (note que se você fez o item anterior, resolver este item é bem trivial!).

    Sugestão: para encontrar limitantes duais, procure usar as informações sobre os graus dos vértices (no grafo original ou nas soluções parciais que forem geradas durante a execução do algoritmo de *branch-and-bound*).