PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão (3WA) Lorenza Moreno e David Sotelo (3WB)
Horário: 2as. e 4as. 15-17hs (3WA), 3as. e 5as. 19-21 (3WB)
16 de junho de 2010
Data da Entrega: 7 de julho de 2010
Período: 2010.1

ANÁLISE DE ALGORITMOS (INF 1721)

2^o Trabalho de Implementação

Descrição

Este trabalho prático consiste em desenvolver códigos para a resolução de um problema de Otimização Combinatória. A análise dos algoritmos considerará tanto o tempo total de CPU como a qualidade comprovada das soluções encontradas.

O desenvolvimento dos códigos e a análise experimental devem seguir os seguintes roteiros:

- Descrever os algoritmos informalmente.
- Demonstrar o entendimento do algoritmo explicando, em detalhe, o resultado que o algoritmo deve obter e justificá-lo.
- Explicar a fundamentação do algoritmo e justificar a sua corretude. Apresentar e explicar a complexidade teórica esperada para cada algoritmo.
- Documentar o arquivo contendo o código fonte, de modo que cada passo do algoritmo esteja devidamente identificado, deixando claro como o mesmo é executado.

O trabalho entregue deve conter:

- Um documento contendo o roteiro de desenvolvimento dos algoritmos (e dos códigos), os itens pedidos acima, comentários e análises sobre a implementação e os testes realizados (papel e eletrônico).
- Os códigos fonte (eletrônico).
- Um e-mail contendo os códigos fonte e os executáveis correspondentes deve ser enviado para **poggi@inf.puc-rio.br**. É **OBRIGATÓRIO** o uso do ASSUNTO (ou SUBJECT) AA101T2, **a falta do e-mail COM este** ASSUNTO **implica na NÃO consideracão do trabalho**.
- O trabalho pode ser feito em grupos de até 3 (três) alunos.

O Problema do Caixeiro Viajante (*Traveling Salesperson Problem*)

Dadas n cidades e as distâncias entre todos os pares de cidades, considerando-se uma cidade inicial, deseja-se determinar a ordem de visitação das outras $n-1$ cidades (cada uma exatamente uma vez) cuja distância total percorrida associada, incluindo o retorno para cidade inicial, é mínima.

Este problema pode ser visto como um problema em grafos. Seja $G = (V, E)$ um grafo orientado e completo, ou seja, existe uma aresta orientada $(v_i, v_j) \in E$ entre cada par de vértices $v_i, v_j \in V$. Um *circuito* em G em é uma sequência de vértices e arestas $C = \langle v_1, (v_1, v_2), v_2, (v_2, v_3), \ldots, (v_{k-1}, v_k), v_k \rangle$ tal que $v_1 = v_k$ e todos os demais vértices na sequência são distintos. Neste caso, dizemos que o circuito C *visita* os vértices $v_1, v_2, \ldots, v_k$ de G. Em outras palavras, apenas o primeiro e o último vértices se repetem em um circuito. Além disso, para que uma aresta (v_i, v_j) esteja presente em um circuito C a mesma deve aparecer imediatamente após o vértice v_i e imediatamente antes do vértice v_j em C.

Seja $p : E \mapsto \Re^+$ uma função de *pesos* positivos associados às arestas de G. O *custo de um circuito* C é definido como a soma dos pesos das arestas que o compõem. Em outras palavras, $custo(C) = \sum_{(v_i,v_j)\in C} p(v_i, v_j)$.

Um circuito é dito *hamiltoniano* caso visite todos os vértices de G. Um circuito hamiltoniano é dito *de custo mínimo* para um grafo G, caso possua o menor custo dentre todos os circuitos hamiltonianos possíveis para G.

O objetivo do PROBLEMA DO CAIXEIRO VIAJANTE consiste em, dados G e p, determinar um circuito hamiltoniano de custo mínimo para G. O PROBLEMA DO CAIXEIRO VIAJANTE é `NP-difícil`.

O propósito do presente trabalho consiste na implementação de métodos exatos, com complexidade de tempo exponencial, objetivando a obtenção de soluções ótimas para instâncias do PROBLEMA DO CAIXEIRO VIAJANTE. Neste sentido, três métodos devem ser implementados, os quais serão descritos a seguir.

1. Backtracking

O aluno deve implementar um algoritmo que enumere todos os possíveis circuitos hamiltonianos em G, retornando aquele que possui custo mínimo. Como G é um grafo completo, a enumeração de todos os seus circuitos hamiltonianos pode ser obtida a partir da geração de todas as permutacões possíveis dos vértices de G.

Uma permutação de V vai sendo construída a cada passo com a escolha do próximo vértice a ser visitado a cada nível de uma *árvore de enumeração*. Esta *árvore* é implicitamente representada quando uma implementação recursiva é utilizada. Assim, sempre que um vértice é inserido, o custo da permutação parcial pode ser calculado acrescentando-se o custo da aresta que o

conecta ao vértice anterior. Ao final, o custo do circuito hamiltoniano correspondente a esta permutação pode ser obtido somando-se o custo da aresta de retorno ao vértice inicial.

Em outras palavras, dada a permutação $\pi = \langle v_1, v_2, \ldots, v_n \rangle$ de V, onde $n = |V|$, o custo do circuito hamiltoniano correspondente pode ser obtido pela soma dos custos das arestas $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)$.

Ainda de acordo com o algoritmo proposto, em cada nível k da *árvore de enumeração* dos circuitos hamiltonianos de G, um novo k-ésimo vértice deverá ser escolhido dentre os $n - k$ vértices ainda não selecionados. Além disso, sem perda de generalidade, no primeiro nível árvore de enumeração, um mesmo (único) vértice inicial pode ser sempre fixado como o primeiro vértice no circuito hamiltoniano correspondente, resultando em apenas $(n - 1)!$ permutações a serem testadas. Desta forma, a complexidade de tempo do algoritmo de *backtracking* será $O(n!)$, onde $n = |V|$.

2. Programação Dinâmica

O aluno deve implementar um algoritmo de programação dinâmica para gerar caminhos de custo mínimo que partam de um determinado vértice w, visite exatamente uma vez cada vértice de um subconjunto $S \subseteq V - \{w, v_1\}$ e termine sempre em um mesmo vértice v_1 (previamente selecionado). Para tal, todos os 2^{n-2} subconjuntos possíveis de $V - \{w, v_1\}$ devem ser considerados, assim como todas as $n - 1$ possíveis escolhas de $w \in V - \{v_1\}$. Conforme dito anteriormente, o vértice v_1 é fixo.

A recursão da programação dinâmica associada à abordagem acima segue. Seja $CT(w, S)$ o custo total do caminho que se inicia no vértice w, passa exatamente uma vez em cada vértice de S e termina no vértice v_1 (através de uma única aresta ligando o último elemento de S visitado pelo caminho a v_1). A recursão se escreve:

$$CT(w, S) = \min_{v \in S} \{p(w, v) + CT(v, S - \{v\})\}$$

Observe e explique por que o número de estados nesta programação dinâmica é $O(n.2^n)$. Em seguida conclua que a complexidade de tempo do algoritmo de programação dinâmica proposto é $O(n^2.2^n)$. Uma referência para esta abordagem é o livro dos autores *Horowitz e Sahni*, presente na bibliografia do curso.

3. Branch-and-Bound

Alterar o backtracking do item 1) de forma a utilizar os seguintes limites inferiores (LB de *lower bound*), ou expectativa otimista para o valor do custo total da permutação de menor custo que pode ser obtida a partir da permutação parcial associada ao nó corrente da *árvore de enumeração*. Isto é, um LB é um valor que é garantidamente menor ou igual ao custo total do circuito hamiltoniano de custo mínimo. Quanto maior é um LB, melhor ele é. Dado que já

se tem uma permutação parcial, um LB será a soma o custo da permutação parcial com um LB para se completar o circuito hamiltoniano, o que corresponde a um LB para o problema do caixeiro viajante considerando que a permutação parcial é um só vértice de onde as arestas saem do último vértice e chegam no primeiro (vértice inicial).

Observe agora que se um (bom) circuito hamiltoniano é conhecido, se um LB possui valor maior ou igual ao valor deste (bom) circuito hamiltoniano conhecido, não é mais necessário continuar descendo na *árvore de enumeração*, pois nenhuma solução (circuito hamiltoniano) de custo menor que o (melhor) conhecido poderá ser encontrado. Ou seja, na implementação recursiva se faz um *backtrack*.

Implemente os limites inferiores (LBs) abaixo:

1. $Q-rota$: Seja S o conjunto de vértices que ainda **não** fazem parte da permutação parcial (inicialmente são todos!). O LB corresponde ao caminho mais curto com $|S|$ arestas que sai do vértice final da permutação parcial passa por vértices em S, não necessariamente por todos, e termina no vértice inicial. A seguinte recursão calcula esse caminho, onde $LB_1(w, 0) = p(v', w)$ e v' é o último vértice da permutação parcial.

   $$LB_1(w, k) = \min_{v \in S}\{p(v, w) + LB_1(v, k-1)\} \quad k = 1, \ldots, |S| - 1$$

   ao final deve-se somar $p(w, v_1)$ a $LB_1(w, |S| - 1)$ para todo $w \in S$. Finalmente o custo da permutação parcial deve ser somado para se obter o LB.

2. $1 - Arvore$: Seja S o conjunto de vértices que ainda **não** fazem parte da permutação parcial (inicialmente são todos menos o inicial!). Obtenha a árvore geradora mínima para o grafo induzido $G(V(S), E(S))$. Some ao valor obtido o custo da menor aresta que sai da permutação parcial para um vértice em S e o custo da menor aresta que sai de S para a permutação parcial (use o algoritmo que você preferir, *Prim* ou *Kruskal*, ou pense em algo ainda mais eficiente!)

O seu algoritmo de *branch-and-bound* deve ainda definir um critério para a escolha do próximo vértice que entra na permutação parcial em cada nível. Um circuito hamiltoniano inicial pode ser obtido e seu valor dado como referência para a poda na *árvore de enumeração*. Quando um circuito hamiltoniano de custo inferior é encontrado o valor dado como referência para a poda deve ser atualizado.

TESTES DOS ALGORITMOS

Os 4 algoritmos devem ser testados no conjunto de instâncias disponível no site do curso. Para cada um dos arquivos teste fornecidos, deve-se executar por até uma hora cada instância. O circuito hamiltoniano de menor custo encontrado deve ser reportado.