TRABALHO 0:
ÁRVORES E ESTRUTURAS DE PRIORIDADE
DRAFT 1

RAUL H.C. LOPES

1. Preliminares

Neste trabalho você exercitará seus conhecimentos de algumas estruturas fundamentais da arte de programação de computadores: seqüências, árvores binárias de pesquisa e de prioridade, ordenação. O objetivo consiste em explorar os recursos de linguagens de programação funcionais para obter código claro e compacto e, simultaneamente, buscar eficiência.

A seção 2 apresenta os fundamentos de teoria de grafos que serão usados no trabalho. A seção 3 apresenta os exercícios a realizar. A seção 4.1 define os critérios que serão utilizados na correção do trabalho e atribuição de pontos. A seção 5 define o que deve ser entregue e a forma de submissão do trabalho para correção. A seção 7 descreve alteracões feitas neste documento desde a primeira versão.

Algumas dicas importantes sobre a execução do trabalho seguem.

(1) Não deixe de ler este documento por completo antes de iniciar o trabalho.
(2) Siga estritamente as especificações deste documento: qualquer desvio delas pode significar a anulação de um exercício ou de todo o trabalho.
(3) Comece a trabalhar de imediato: o trabalho foi concebido para ser realizado em quatro semanas. Depois disso, você terá outro trabalho e nova prova.

O trabalho pode ser executado em grupos, mas, como estabelecido na seção 4.1, grupos maiores recebem menos créditos. Um elemento importante da avaliação do trabalho trata da deteção de mutualismo parasitário: situação em que um aluno atua de vampiro parasita sugando o resultado do trabalho do grupo. Fato: plágio em qualquer item de um trabalho implica na anulação completa do trabalho do(s) grupo(s) envolvidos. Qualquer elemento de um grupo pode ser, durante a execução do trabalho ou após a entrega do mesmo, avaliado

em relação ao conhecimento do que foi feito. Essa avaliação poderá ser feita de forma oral ou por escrito.

2. O Problema do Caixeiro Viajante

Os exercícios deste trabalho lidam com o *Problema do Caixeiro Viajante*, *Traveling Salesman Problem*, de agora em diante chamado **TSP**. Este problema demanda encontrar o caminho mais curto para visitar cada cidade de um conjunto dado exatamente uma vez e retornar ao ponto de partida. Para representar cidades, usaremos uma abstração: um grafo não dirigido.

Um grafo não dirigido é um par (V, E), onde V é um conjunto de pontos e E é um conjunto de pares de pontos, cada par de pontos sendo chamado de arestas. Para nosso trabalho valem as seguintes restrições:

- uma aresta (u, v) indica que existe uma conexão direta do ponto u ao ponto v;
- o grafo é geométrico: os pontos são elementos de um espaço Euclideano de duas dimensões;
- as coordenadas dos pontos são pares de inteiros positivos;
- para qualquer par de pontos u e v do grafo existe uma aresta (u, v), que tem um custo associado dado pela distância entre u e v;
- o grafo é não direcionado e a existência de uma aresta (u, v) indica a existência de outra aresta (v, u) com mesmo custo que a primeira.

Um caminho em um grafo é uma seqüência de pontos do mesmo. Um circuito Hamiltoniano C é uma seqüência de pontos com as seguintes restrições:

- cada ponto do grafo inicial figura exatamente uma vez no circuito;
- o circuito indica a seqüência de pontos a visitar para sair de um vértice inicial, o primeiro da seqüência dado, ir até o último vértice da seqüência, visitando todos os outros na ordem dada por C, e voltar ao ponto de partida.

Um circuito Hamiltoniano, que chamaremos simplemente de **tour**, consiste em uma permutação do conjunto de pontos de grafo dado. Assumindo que $C!i$ denote o i-ésimo elemento do tour C, o custo do tour é dado por:

$$cost.C = (+i : 0 \leq i < n - 1 : dist(C!i, C!(i + 1))) + dist(C!(n-1), C!0)$$

onde $dist(u, v)$ é a distância geométrica de u a v.

O problema do **TSP**, como colocado neste trabalho, consiste, então, em determinar o **tour** de menor custo.

3. Os exercícios

Esta seção contém exercícios sobre estruturas de dados e algoritmos usados freqüêntemente em bancos de dados espaciais e problemas sobre grafos como o **TSP**.

Em todos os exercícios represente o **tour** resultante como uma lista de pontos sem repetição. Por exemplo, o **tour**que parte do ponto p_0, vai para p_2, p_1 e volta a p_0 seria representado pela lista:

$$[p_0, p_2, p_1]$$

e não pela lista

$$[p_0, p_2, p_1, p_0]$$

Note que essa restrição refere-se apenas ao **tour** resultante. Sua representação interna dos diversos algoritmos fica a cargo de sua criatividade.

3.1. **Árvores binárias espaciais.** Neste trabalho, serão apenas considerados problemas em duas dimensões. As árvore binárias espaciais usadas armazenarão pontos de duas coordenadas. Uma árvore binária espacial, ver K-d trees for semidynamic point sets, apresenta as seguintes características:

- cada folha armazena uma seqüência de ao menos um ponto e até oito pontos.
- um nó interno particiona o conjunto de pontos de subárvore de que ele é raiz e define ao menos:
  - a coordenada usada no particionamento;
  - o valor da linha de particionamento;
  - a subárvore da esquerda contendo os pontos que têm na coordenada de particionamento valor menor do que (ou igual a) o valor da linha de particionamento;
  - a subárvore da direita contendo os pontos que têm na coordenada de particionamento valor maior do que o valor da linha de particionamento
- no caminho da raiz para as folhas os nós internos alternam o particionamento dos pontos pelas coordenadas dadas.

Defina os seguintes tipos em Haskell:

- *Point a* para representar pontos de um grafo, cujas coordenadas são do tipo a.

- *KDtree a* para representar tipo das *kd-tree* de pontos do tipo *Point a*.

Exercício 1. *Defina em Haskell uma estrutura KDtree a que implemente uma* kd-tree *sobre pontos de a. Sua estrutura deve apresentar ao menos as seguintes operações:*

- kdBuild xs*: constrói uma* kd-tree *a partir da lista de pontos* xs*.*
- kdSearch p t*: determina se o ponto p está na* kd-tree *t.*
- kdInsert p t*: insere p na* kd-tree *t.*
- kdDelete p t*: exclui p de t.*
- kdUndelete p t*: reinsere item p previamente excluído de t.*
- kdNN p t*: encontra o vizinho mais próximo de p em t.*

Exercício 2. *Apresente testes comparativos de correção, desempenho e uso de memória da estrutura do exercício ex. 1, usando dados dos arquivos da seção 3.5.*

3.2. **Árvores binárias de priodidades.**

Exercício 3. *Uma Árvore binária de prioridades é uma árvore binária ordenada da raiz para as folhas: a raiz é menor (ou igual) a qualquer elemento da árvore e cada uma das suas subárvore é uma Árvore binária de prioridades.*

Escolha uma estrutura para representar sua árvore de prioridades dentre as seguintes:

- Splay tree*, veja em Self-ajusting Binary Search Trees.*
- Binomial queue*, veja em A data structure for manipulating priority queues.*

Implemente em Haskell uma estrutura Priority a *para representar árvores de prioridades, de acordo com a estrutura que você escolheu. Sua estrutura deve ter, ao menos as seguintes operações:*

- join(q0,q1)*: realiza a união de duas estrutras de prioridadeq0,q1.*
- findMin(q0)*: retorna o mínimo da estrutura.*
- delMin(q0)*: exclui o mínimo da estrutura*

Exercício 4. *Apresente testes comparativos de correção, desempenho e uso de memória para a estrutura do exercício exrefbinq.*

3.3. **Heurísticas para STSP.** Nos exercícios a seguir, você trabalhará com as heurística **Farthest Insertion (FI)**, descrita na página 23 de Experimental Analysis of Heuristics for the STSP, e **Double Minimum Spanning Tree (DMST)**, descrita na página 22, do mesmo artigo. Implemente todos os algoritmos em Haskell.

Exercício 5. *Implemente um algoritmo para encontrar um **tour** para o **TSP**, usando a heurística **FI**.*

Exercício 6. *Apresente testes de correção e desempenho para o algoritmo do ex. 5.*

Exercício 7. *Implemente um algoritmo para encontrar a árvore geradora mínima de um conjunto de prontos.*

Exercício 8. *Apresente testes de correção e desempenho para a estrutura do ex. 7.*

Exercício 9. *Implemente um algoritmo para encontrar um **tour** para o **TSP**, usando a heurística **DMST**.*

Exercício 10. *Apresente testes de correção e desempenho para o algoritmo do ex. 9.*

Exercício 11. *Apresente estudo comparativo de desempho, levando em conta tempo e qualidade do **tour**, para os algoritmos dos exercícios 5 e 9.*

3.4. **Linguagem funcional impura.**

Exercício 12. *Escolha uma das heurísticas implementadas na sc. 3.3 e implementa em Scheme, buscando obter um código o mais rápido possível, com o obhetivo de bater o equivalente em Haskell.*

Exercício 13. *Apresente testes de correção e desempenho para os algoritmos do ex. 12.*

Exercício 14. *Apresente estudo comparativo de desempho, levando em conta tempo e qualidade do **tour**, para a solução em Scheme do ex. 12 e equivalente em Haskell.*

3.5. **Testes.** O *tarball test.data.tar.bz2* contém arquivos de dados que você poderá utilizar para testar seus programas. Interessa avaliar neste trabalho para cada arquivo:

- correção do **tour** calculado;
- tempo de execução;
- qualidade do **tour** quando comparado com o melhor **tour** possível.

Note que os arquivos de dados estão no formato TSPLIB[1]. Use o programa **fromtsplib.pl** para convertê-los para o formato do programa **tourlen**, apresentado na seçao 6.

Abre essa diretório com o comando:

```
tar -jxf test.data.tar.bz2
```

[1] Não deixe de visitar o *site* TSPLIB.

Leia o arquivo **README**, no diretório **test.data** criado, para mais informações.

4. Créditos

A definição dos critérios de avaliação A atribuição de nota para o trabalho ocorrerá em duas fases:

(1) Atribuição de créditos pelo trabalho submetido, de acordo com critérios apresentados na sc. 4.1.
(2) Definição do multiplicador de acordo com a tabela 2.

4.1. **Definição do número de créditos.** O número de créditos atribuídos a cada exercício está definido na tab. 1. Note que os créditos atribuídos aos exercícios de teste de desempenho e correção das diferentes estruturas dependerão da apresentação de artigo em LaTeX, apresentando tabelas comparativas e conclusões dos experimentos realizados por cada grupo. Não hesite em discutir com outros colegas (de outros grupos) o tipo de teste a realizar. Não será considerado plágio mais de um grupo apresentar extamente os mesmos testes e até estrutura de artigo. Será certamente plágio coincidência em excesso nos resultados de testes! Em resumo, dê desde o início do trabalho ênfase especial à escrita do artigo.

Oito dos exercícios referem-se a testes dos algoritmos a implementar. Dê ênfase especial a esses testes e à sua apresentação. Os testes de correção dos algoritmos devem ser realizados através de programas que testem automaticamente propriedades de correção das suas estruturas. Por exemplo, implemente um algoritmo que teste se a estrutura construída pelos algoritmos do ex. 1 é uma *kd-tree* válida. Apresente testes com esse algoritmo.

Os quatro itens finais da tab. 1 referem-se a testes de desempenho. O testes de desempenho serão realizados sobre um conjunto de dados, usando instâncias consideradas desafiadoras do **TSP**. Em cada teste, só poderão participar grupos que tiverem apresentado soluções sem erros para o respectivo exercício : por exemplo, só participarão do teste de desempenho do ex. 5 grupos cuja implementação da heurística **FI**não tiver erros detectados em nenhum teste. Os grupos serão classificados por tempos de execução. O primeiro colocado receberá quatro créditos. Os outros receberão pontos na razão inversa do aumento de tempo em relação ao tempo do vencedor.

O prêmio *cool hacker*, últimos quatro pontos da competição, será atribuído pela soma dos tempo dos três testes de desempenho. Novamente o primeiro leva total de créditos e os outros levam créditos na razão inversa da diferença em relação ao vencedor.

Tarefa	Valor
ex. 1	8
ex. 2	2
ex. 3	8
ex. 4	2
ex. 5	10
ex. 6	2
ex. 7	8
ex. 8	2
ex. 9	10
ex. 10	2
ex. 11	4
ex. 12	20
ex. 13	2
ex. 14	4
desempenho em ex. 5	4
desempenho em ex. 9	4
desempenho em ex. 12	4
cool hacker	4

TABELA 1. Tabela de Créditos por Tarefa

Fator	Multiplicador
Grupos de 1 aluno	110/100
Grupos de 2 alunos	100/100
Grupos de 3 alunos	75/100
Grupos de 4 alunos	60/100
Grupos de mais 4 alunos	0
Plágio	0

TABELA 2. Tabela de multiplicadores

4.2. **Definição de multiplicador.** A nota de cada trabalho é dada pelo número de créditos obtidos pelo trabalho multiplicado pelo fator da tabela 2.

Em caso de plágio todos os trabalhos de grupos envolvidos são anulados. Poderá ser enquadrado como plágio qualquer trabalho em que um mais alunos do grupo não tenham conhecimento de qualquer item do trabalho. Esse conhecimento deverá ser demonstrado em prova e/ou enttrevista.

5. A submissão para correção

Submeta sua trabalho por e-mail entre os dias 30/06/05 e/ou 06/07/05. Trabalhos submetidos fora dessa data serão considerados nulos.

A mensagem de submissão do seu trabalho deverá conter:

- **Subject: lpii.tp0**
- *Attachment*: **tp0.tar.bz2**

O corpo da mensagem será desprezado. O arquivo anexdo deverá ser obrigatoriamente denominado

tp0.tar.bz2

e conterá todo os fontes necessários para compilar seu trabalho prático, incluindo fontes em LaTeX, **Haskell**, **Makefile**e arquivo de identificação.

As seguintes regras devem ser rigorosamente seguidas:

- Nenhum compilado deve ser enviado.
- O *tarball do seu trabalho conterá a seguinte estrutura:*
  - *arquivo de* ***id***
    Conterá uma linha inicial com e-mail *de contato do grupo e, depois, uma linha de identificação para elemento do grupo, contendo número de matrícula e nome completo separados por* ***':'*** *(dois pontos.) Por exemplo:*
    ```
    jolero@gmail.com
    99900089:Ze' do lero
    ```
  - *arquivo* ***Makefile***
  - *diretório* ***doc***
    Conterá os fontes de toda a documentação do trabalho.
- *seu tarball será aberto e compilado com as seguintes linhas:*

  ```
  tar -jxf tp0.tar.gz
  make all
  ```

  Essas linhas gerarão os seguintes arquivos:
  - ***tsp.fi***
    Executável que implementa heurística do exercício 5.
  - ***tsp.dmst***
    Executável que implementa heurística do exercício 9.
  - ***tsp.scheme***
    Executável que implementa heurística do exercício 12.
  - ***artigo.pdf***
    Artigo em formato ***PDF****, que avalia qualidade de* ***tour*** *e desempenho dos algoritmos.*

6. Exemplo

O *tarball point.io.tar.bz2* contém um exemplo de programa para leitura e escrita de pontos. Abra o arquivo usando

```
tar -jxf point.io.tar.bz2
```

Isso criará um diretório de nome **point.io**. Leia o arquivo **README** desse diretório para instruções adicionais.

7. Alterações neste documento

As seguintes alterações foram realizadas neste documento quando comparado com sua primeira versão:

- A tab. 1 foi corrigida para incluir valor atribuído aos exercícios 3 e 4.
- A screfcredits inclui agora uma explanação sobre os testes de correção a realizar.