TRABALHO 1:
ÁRVORES E ESTRUTURAS DE PRIORIDADE EM C
FINAL

RAUL H.C. LOPES

1. Preliminares

Este trabalho consiste em exercício de programação em uma linguagem imperativa sobre algumas das estruturas do trabalho anterior. Para tal você implementará em linguagem *C/C++* duas soluções heurísticas para o problema de encontrar o melhor **tour** para o problema do **TSP**. Você implementará as heurística **NN** e *Greedy* com fila de prioridade descritas no trabalho anterior[1] e uma heurística para melhora de **tour** obtido.

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.

O trabalho pode ser executado em grupos, mas, como estabelecido na seção 6, 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. As heurísticas

2.1. **Vizinho mais próximo e guloso.** As heurística **NN** e *Greedy* estão descritas no documento do Trabalho Prático 0.

[1] Para mais detalhes sobre o problema e as heurísticas veja documento do Trabalho Prático 0 e referências nele contidas.

2.2. **Busca local.** Uma das mais importantes heurísticas para obter o melhor **tour**possível para o **TSP** consiste em usar os seguintes passos:

- construir um **tour** inicial t_0 usando heurísticas como **NN** ou *Greedy*;
- melhorar o **tour** t_0 usando heurísitca de busca local realizam exclusão de algumas arestas, seguida de religação dos nós de grau um para obter um **tour**.

A heurística de busca local acima referida é implementada a partir de alguma variante do processo de permutação limitada de nós, conhecido na literatura como operação *k-opt*[2].

Um **tour** t_1 é obtido de um **tour** t_0 por *k-opt* quando t_1 é obtido pelos seguintes passos:

- geração de um grafo desconexo t' pela exclusão de k arestas não consecutivas de t_0;
- religação de nós de grau um presentes em t' para obter o **tour** t_1

3. Representação de **tour**

Escolha para representar seu **tour** uma das seguintes estruturas[3]:

- *array* com a seqüência de nós;
- lista duplamente encadeada;
- árvore binária com operações de *splay*;
- árvore em dois níveis.

As seguintes observações são importantes na escolha da representação do **tour**:

(1) Você deve escolher obrigatoriamente dentre uma das alternativas acima.
(2) Sua escolha afetará significativamente o desempenho do programa, que é o item mais importante da avaliação deste trabalho.
(3) Implementar as operações descritas na página quatro de **Fredman:tspds** deverá faciltar em muito sua tarefa de implementação. As operações ali descritas são:
   - $Next(a)$: Dado nó a, determina o nó que o segue no **tour**.
   - $Prev(a)$: Dado nó a, determina o nó que o precede no **tour**.

[2] Veja artigo, referenciado neste documento como **Fredman:tspds**, sobre alternativas para representação de **tour**, que facilitam a implementação da operação *k-opt*.

[3] Descritas em **Fredman:tspds**.

- $Between(a, b, c)$: predicado que determina se b está entre a e c no **tour**.
- $Flip(a, b, c, d)$: atualiza **tour**, excluindo as arestas (a, b) e (c, d) e inserindo as arestas (b, c) e (a, d).

Note que está implícito em todas essas operações o parâmetro que representa o **tour**.

4. Fila de prioridade

A heurística *Greedy* demanda uma estrutura de prioridade. Escolha sua estrutura de prioridade dentre as seguintes:

(1) *Binomial queue*[4]
(2) Árvore auto-ajustável[5]
(3) Árvores *splay*[6]
(4) Árvore 2-3.

Novamente sua escolha está restrita a uma dessas estruturas e ela afetará significativamente o desempenho do seu programa.

5. Os exercícios

Os exercícios desta seção demandam a implementação de quatro programas diferentes para a avaliação de **tour**. Todos usam o mesmo padrão de entrada e saída dos programas do Trabalho Prático 0.

Exercício 1. *Implemente um programa cálculo de* ***tour****, usando a heurística* ***NN*** *e uma estrutura de árvore binária espacial, como descrita no Trabalho Prático 0.*

Exercício 2. *Implemente um programa para cálculo de* ***tour****, usando a heurística* Greedy*, combinada com árvore binária espacial e árvore de prioridade.*

Exercício 3. *Implemente um programa que gere um* ***tour*** *inicial usando a heurística* ***NN****, como no exercício exercício 1, melhore o* ***tour*** *usando* k-opt*.*

Exercício 4. *Implemente um programa que gere um* ***tour*** *inicial usando a heurística* Greedy*, como no exercício exercício 2, melhore o* ***tour*** *usando* k-opt*.*

---

[4]Descrita no artigo vuillemin.
[5] Descritas em **sleator83** e no Trabalho Prático 0.
[6]Descritas em **sleator85**.

Critério	Créditos
Seção exercício 1	10
Seção exercício 2	10
Seção exercício 3	10
Seção exercício 4	10
Seção 6.4	10
Seção exercício 6.6	10
Seção exercício 6.7	10
Seção exercício 6.8	10
Secão exercício 6.5	10
Seção exercício 6.9	10

TABELA 1. Tabela de atribuição de créditos

6. CRÉDITOS

A atribuição de nota para o trabalho ocorrerá em duas fases:

(1) Atribuição de créditos pelo trabalho submetido, de acordo com a tabela 1.
(2) Definição do multiplicador de acordo com a tabela 2.

6.1. **Definição do número de créditos.** Cada trabalho submetido será avaliado para receber de zero a cem créditos de acordo com a tabela 1.

6.2. **Critério fundamental da atribuição de créditos.** Os critérios que definem atribuição de créditos colocam uma ênfase especial na eficiência (50% dos créditos vão para desempenho) e correção: programas que apresentam qualquer erro não podem participar de nenhuma competição de desempenho.

6.3. **A avaliação não comparativa.** Os itens das seções exercício 1,exercício 2,exercício 3 e exercício 4 serão avaliados em relação:

- Correção do **tour**;
- Correção do uso de memória;
- Melhoria no **tour** no caso das seçoes exercício 3 e exercício 4.

6.4. **Estudo de qualidade.** Escreva um artigo curto (duas ou três páginas em LaTeX) comparando os programas implementados com respeito a:

- qualidade do **tour**;
- tempo de execução;
- demanda de memória.

Importante: Não use os pacotes **listings** e **noweb** no artigo deste trabalho.

6.5. **O melhor tour.** Neste item os **tour**s de todos programas serão comparados para definir o melhor. Em cada teste realizado, o grupo com o melhor programa receberá dez créditos e os outros grupos receberão créditos razão inversa da piora da qualidade do **tour**, quando comparado com o melhor grupo.

A nota de cada grupo neste item será a média das notas dos diversos testes.

6.6. **O mais rápido NN.** Usando a mesma metodologia de atribuição de créditos da seção exercício 6.5, buscar-se-á aqui a implementação mais rápida para o exercício exercício 1.

6.7. **O mais rápido *Greedy*.** Usando a mesma metodologia de atribuição de créditos da seção exercício 6.5, buscar-se-á aqui a implementação mais rápida para o exercício exercício 2.

6.8. **O mais rápido *k-opt*.** Usando a mesma metodologia de atribuição de créditos da seção exercício 6.5, buscar-se-á aqui a implementação mais rápida para os exercício exercício 3 e exercício 4, obedecendo à seguinte restrição:

- Só podem participar programas que tenham produzido **tour** que seja no máximo 10% pior que o vencedor do teste da seção exercício 6.5. Eliminam-se aqui programas que são rápidos porque não produzem um **tour** minimamente respeitável.

6.9. **O mais econômico *k-opt*.** Usando a mesma metodologia de atribuição de créditos da seção exercício 6.5, buscar-se-á aqui a implementação com menor uso de memória para os exercício exercício 3 e exercício 4, obedecendo à seguinte restrição:

- Só podem participar programas que tenham produzido **tour** que seja no máximo 10% pior que o vencedor do teste da seção exercício 6.5. Eliminam-se aqui programas que são rápidos porque não produzem um **tour** minimamente respeitável.

6.10. **Definição de multiplicador.** O 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

Fator	Multiplicador
Grupos de 1 aluno	11/100
Grupos de 2 alunos	10/100
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

do trabalho. Esse conhecimento deverá ser demonstrado em prova e/ou enttrevista.

7. A submissão para correção

Submeta sua trabalho por e-mail até 07/03/05. Trabalhos entregues até 18/03/2005 poderão, a critério do **Senhor do Castelo**, ser corrigidos.

A mensagem de submissão do seu trabalho deverá conter:

- **Subject: tbo.tp1**
- *Attachment*: **tp1.tar.bz2**

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

tp1.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.
- Não use os pacotes **listings** e **noweb** no artigo deste trabalho.
- 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 **kdtree**
    Conterá a implementação da *kd-tree*.
  - diretório **pq**

Conterá a implementação da fila de prioridade.
- diretório **nn**
  Conterá a implementação da heurística nearest-neighbor, do exercício 1.
- diretório **greedy**
  Conterá a implementação da heurística *Greedy*, do exercício 2.
- diretório **opt** com implementação da heurística *k-opt*.
- diretório **nnopt** com implementação do programa usando **nn** e *k-opt*, referido no exercício exercício 3.
- diretório **greedyopt** com implementação do programa usando **greedy** e *k-opt*, referido no exercício exercício 4.
- diretório **doc**
  Conterá os fontes de toda a documentação do trabalho.

Adicionalmente seu *tarball* poderá conter outros diretórios relevantes para a sua implementação.

- seu *tarball* será aberto e compilado com as seguintes linhas:

```
tar -jxf tp1.tar.gz
make all
```

Essas linhas gerarão os seguintes arquivos:
- **tsp.nn**
  Executável que implementa heurística do exercício 1.
- **tsp.greedy**
  Executável que implementa heurística do exercício 2.
- **tsp.nnopt**
  Executável que implementa heurística do exercício 3.
- **tsp.greedyopt**
  Executável que implementa heurística do exercício 4.
- **artigo.pdf**
  Artigo em formato **PDF**, que avalia qualidade de **tour** e desempenho dos algoritmos.

8. 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.

Importante: você não é obrigado a usar essas rotinas de entrada e saída. Seu programa deve apresentar o mesmo padrão de entrada e saída do Trabalho Prático 0, mas a implementação é decisão sua.