Universidade Federal de Minas Gerais

Bacharel em Sistemas de Informação
Algoritmos e Estruturas de Dados 2

Trabalho Prático 2
Junho 2017

Gabriel Silva Bastos
Matrícula: 2016058204

1 Introdução

O trabalho prático consiste em ordenar um vetor de números naturais que representam as colunas na caixa mágica de Fernanda.

O tamanho do vetor está entre 1 e 100000 elementos. A faixa de distribuição dos elementos do vetor é de 1 a 100. Considerando a curta faixa de distruibuição, o algoritmo de ordenação *counting sort* foi escolhido para implementar a solução.

2 Visão geral da solução

Como sugerido no trabalho prático, o diagrama do percurso foi utilizado para se implementar a simulação. Para permitir maiores alterações na dinâmica do percurso, a variação do número de filas de bandejas, caixas, filas de bandejas, pilhas de bandejas e *buffets* de alimentos é permitida.
Define-se também diferentes pilhas de bandejas e *buffets* de alimentos como equivalentes. Ou seja, todas as pilhas abrigam o mesmo valor máximo de bandejas e todos *buffets* possuem o mesmo número de cubas, com a mesma ordem de alimentos.

As estruturas de dados fila e pilha, apresentadas durante curso de AEDS 2, foram escolhidas para representar respectivamente as filas de ficha/bandeja, e as pilhas de bandejas.
Para o caixa e os *buffets* de alimentos, foram utilizados simples ponteiros.

Desta forma foi modelado todo o diagrama, permitindo sua implementação na linguagem C.

2.1 Parâmetros

Os seguintes itens foram definidos como alteráveis na simulação:

- Número de usuários que adentram o percurso por minuto.
- Número de caixas. Este valor também determina o número de filas de caixa.
- Número de pilhas de bandejas. Este valor também determina o número de filas de bandeja e o número de *buffets* de alimentos.
- Número máximo de bandejas em cada pilha de bandejas.
- Número de bandejas repostas em cada pilha a cada reposição.
- Frequência de reposição de bandejas nas pilhas.
- Número de cubas em cada *buffet* de alimentos.

2.2 Cálculo do tempo médio

O cálculo do tempo de um usuário foi realizado subtraindo o horário da chegada deste no percurso pelo horário de saída do mesmo. Apenas os usuários atendidos dentro do tempo total definido no trabalho (4 horas) são incluídos no cálculo. Portanto, usuários remanescentes no percurso após decorrido o tempo total não são considerados.

Desta forma, torna-se trivial o cálculo do tempo médio de todos os usuários atendidos.

3 Implementação

3.1 Organização e estrutura de diretórios

A organização dos diretórios no projeto foi definida da seguinte forma:

bin:	binários resultantes da compilação.
data:	arquivos contendo os dados referentes aos resultados obtidos nas simulações.
doc:	arquivos referentes à documentação do projeto.
scripts:	*scripts* para manipulação da simulação, dos parâmetros e dos resultados.
src:	código fonte da simulação.

3.2 Compilação

Dois *makefiles* foram desenvolvidos para o projeto. Um para ambiente linux (makefile) e um para ambiente Windows (makefile.win). A diretriz simples de compilação é denominada *build* em ambos *makefiles*.

O *makefile* para ambiente linux possui também diretrizes para a execução dos *scripts* e armazenamento dos resultados destes em disco. Outras facilidades de debug também são definidas neste *makefile*.

3.3 Entrada

Foram implementadas duas formas de entrada dos parâmetros na simulação. Caso seja fornecido um único nome de arquivo na execução do programa, os parâmetros são obtidos no conteúdo do arquivo. Caso contrário, é requisitado ao usuário a entrada de cada um dos parâmetros.

O arquivo de configuração deve seguir o seguinte formato:
Um número respectivo à um parâmetro no início de cada linha, na ordem dos parâmetros descritos na seção 2.1.
Comentários podem suceder cada número em cada linha.

3.4 Estruturas de dados

Como base, foram implementadas uma lista encadeada e uma lista baseada em um vetor. Foram denominadas *linkedlist* e *vectorlist*.
A partir destas, foram implementadas as estruturas pilha e fila, denominadas *stack* e *queue*, e suas operações básicas. Ambas permitem utilizar tanto uma *linkedlist* quanto uma *vectorlist* como armazenamento.

Todas as estruturas foram implementadas de forma genérica, utilizando ponteiros para *void*. A responsabilidade de alocação de itens inseridos foi delegada para o usuário da estrutura de dados. Ainda, para uma maior flexibilidade, as estruturas são parametrizadas por um alocador de memória fornecido pelo usuário. Desta forma, torna-se trivial o controle do método utilizado para utilização de memória.

3.5 Alocação de memória e otimizações

Inicialmente, foi utilizado *malloc* para alocação dos usuários e dos membros internos das estruturas de dados. Após análise através do *valgrind*, foi constatado que um grande número de alocações era realizada. Para melhorar a performance nesse quesito, considerando que o número máximo de usuários é conhecido, a estratégia de *memory pool* utilizada.

Foi utilizado um grande vetor como *pool*, e uma *stack* para controle das alocações. Portanto, as operações de alocação na *memory pool* são extremamente rápidas. Desta forma, uma única alocação com *malloc* é realizada para os usuários, e uma única para os detalhes internos das estruturas.

3.6 Decorrência do tempo no restaurante

Foi utilizado um *loop*, iterando cada instante de tempo (em minutos) do estado do restaurante. Como na noção de uma máquina de estados, o código dentro do *loop* executa a dinâmica do percurso de um instante de tempo para o próximo. Desta forma, a partir de um estado inicial com o percurso vazio, deriva-se todos os estados até o tempo final desejado. Em cada instante, se um usuário sai do percurso, seu tempo é coletado para o futuro cálculo da média.

3.7 Coleta e comparação de resultados

Tendo pronta a implementação da simulação, é prático aplicar determinados parâmetros de entrada para se obter o resultado desta configuração.

Scripts foram desenvolvidos para gerar e aplicar na simulação as configurações interessantes, e também coletar e ordenar os resultados. Desta forma torna-se rápido obter a configuração que gerou a melhor performance do restaurante.

4 Análise de complexidade

Os algoritmos ótimos foram selecionados para a implementação das estruturas de dados. A maioria das operações implementadas possui complexidade $\mathcal{O}(1)$.

A documentação sobre o cálculo de complexidade referente a cada função, incluindo as relativas às estruturas de dados, foi incluída como comentários adjacentes ao código correspondente.

5 Parâmetros e resultados

5.1 Parâmetros interessantes

Foram consideradas como configurações interessantes as seguintes variações de cada parâmetro:

- Número de usuários que adentram o percurso por minuto: sempre 2.
- Número de caixas: 1–4.
- Número de pilhas de bandejas: 1–4.
- Número máximo de bandejas em cada pilha de bandejas: 30, 40 ou 50.
- Número de bandejas repostas em cada pilha a cada reposição: 10, 15 ou 20.
- Frequência de reposição de bandejas nas pilhas: 12, 14, 16, 18 ou 20.
- Número de cubas em cada *buffet* de alimentos: sempre 4.

Um *script* foi utilizado para gerar as combinações destas variações. Foram totalizadas 720 combinações.

5.2 Resultados obtidos

Através de *scripts*, cada combinação dos parâmetros foi aplicada à simulação, e o resultado coletado. Após a coleção de todos os resultados, estes foram ordenados de acordo com o seguinte critério:

- Tempo médio dos usuários (crescente).
- Número de caixas (crescente).
- Número de pilhas de bandejas (crescente).
- Número de bandejas repostas em cada pilha a cada reposição (crescente).
- Frequência de reposição de bandejas nas pilhas (decrescente).

O critério foi baseado na ideia de que alguns parâmetros apresentam maior custo para alteração. Por exemplo: é mais caro adicionar um caixa ao restaurante do que adicionar uma pilha de bandejas.

Os parâmetros que compõe o melhor resultado de acordo com o critério foram:

- Número de usuários que adentram o percurso por minuto: 2.
- Número de caixas: 2.
- Número de pilhas de bandejas: 2.
- Número máximo de bandejas em cada pilha de bandejas: 30.
- Número de bandejas repostas em cada pilha a cada reposição: 15.
- Frequência de reposição de bandejas nas pilhas: 14.
- Número de cubas em cada *buffet* de alimentos: 4.

6 Conclusão

Considerando os resultados obtidos, é crucial que o número de caixas filas e pilhas sejam 2. O número máximo de bandejas em cada pilha não necessita de ser aumentado, porém uma leve melhoria na reposição de bandejas é necessária: de 10 bandejas a cada 12 minutos para 15 bandejas a cada 14.

Essa é a alteração ótima para proporcionar a máxima performance do restaurante.

Caso uma performance média seja aceitável, é possível verificar qual alteração ótima é necessária para se atingir no arquivo *data/sorted_results.txt*.