Estruturas de Informação

Época Normal

5 de Fevereiro 2014

Duração 2,5 horas

Resolva cada exercício em folhas separadas

1. Várias abordagens de computação paralela baseiam-se no pressuposto de que partes de uma aplicação podem ser estruturadas em blocos de código, os quais podem ser executados em paralelo por múltiplos processadores. A abordagem de "work stealing" é uma dessas, tendo a particularidade de que os blocos de código são inicialmente distribuídos pelos processadores, onde são executados de forma LIFO (Last In First Out) mas, se um dos processadores não tiver nenhum bloco para processar, pode "roubar" blocos que estejam à espera em qualquer outro processador. Considere que os blocos de código são genericamente caracterizados por:

```
class Bloco {
   private:
       string codigo;
       unsigned tamanho_codigo;
   public:
       // métodos
        …
};
```

a) Estenda a classe Bloco considerando que existem dois tipos de blocos:
- Bloco Fixo que representa blocos de código "fixo" a um processador e que não podem ser roubados. O processador ao qual está fixo é identificado em atributo.
- Bloco Prioritário que embora possa ser roubado, apenas pode ser roubado em tarefas com uma prioridade específica, identificada por atributo.

Nota: o identificador de um processador e a prioridade são números inteiros não negativos.
Não é necessário definir os métodos das classes.

b) Defina a classe não template WorkStealing que guarda os blocos de código em espera nos vários processadores numa estrutura vector<stack<T>>, em que o índice do vector é o identificador do processador.

c) Defina na classe não template WorkStealing um método para inserir um novo bloco que recebe por parâmetros o número do processador a inserir e o bloco e retorna falso se tal não for possível.

d) Implemente na classe não template WorkStealing o método rouba_por_prioridade que recebe por parâmetros o número de um processador e uma determinada prioridade, e no caso desse processador não ter blocos para executar, rouba do processador seguinte no vector todos os blocos dessa prioridade. Considere que o processador seguinte do último elemento do vector é o processador de índice 0.

Resolva cada exercício em folhas separadas

2. Faça a análise da complexidade temporal T(n) do código abaixo. Justifique.

```
long int misterio (int m, long int n)
{
       if (m >= n)
          return 0;

       int parcial = misterio (m+1, n);

       return parcial + 1;
}
```

3. O autocarro escolar de uma região desloca-se diariamente a algumas casas para a recolha das crianças, percorrendo a rede de caminhos dessa região. As casas são caracterizadas pelo seu endereço e a ligação entre duas casas pela sua distância em Kms.

a) Defina a classe não template PercEscolar de forma a descrever um grafo dirigido que represente a rede de casas e as suas ligações (não é necessário indicar os métodos da classe).

b) O autocarro inicia sempre a sua viagem na "Rua da Eira, n.1" (EO). Implemente o método passaPor que receba um endereço da casa para onde o autocarro se deve dirigir (ED) e um outro endereço da casa por onde o autocarro pretende passar (EP). O método deve devolver o caminho mais curto entre o EO e o ED que passe pelo EP.

4. Defina na classe template tree<TN> o método **treeHas**, que verifica se a árvore contém todos os elementos de uma lista, que é passada como parâmetro.

5. Considere a estrutura de informação designada por fila de prioridade ou HEAP

a) Apresente graficamente a HEAP correspondente à introdução dos elementos: 65, 45, 90, 20, 120, 10. Deve redesenhar a HEAP a cada introdução de um elemento.

b) Apresente graficamente a HEAP correspondente à eliminação do elemento da raiz da HEAP. Deve redesenhar a HEAP a cada alteração da HEAP após a eliminação.