Estruturas de Dados - Pós
Estruturas de Dados Avançadas - Graduação
Lista de Trabalhos - 2019.1

Prof. Victor Campos

Para todas as estruturas, devem ser feitas perguntas interativas para utilizar as operações implementadas junto com uma operação para imprimir o conteúdo da sua estrutura. Não é necessário uma interface grafica.

1. **Heap compacta:** Implementar uma heap compacta para armazenar valores no intervalo $0, \ldots, 2^b - 1$. O espaço total usado pela sua estrutura de dados com n elementos deve ser de $nb + O(1)$ bits dentro de um vetor que está entre 25% e 100% cheio, mantido usando table doubling.

   Operações:

   (a) INCLUIR(Heap H, inteiro x): inclui x em H

   (b) REMOVERMIN(Heap H): remove e retorna o elemento mínimo de H

   (c) CONSTRUIRHEAP(Heap H): cria um vetor aleatório H com n elementos e espaço $nb + O(1)$ e constrói uma Heap sobre estes elementos.

   (d) ALTERARPRIO(Heap H, inteiros i e x): altera a prioridade do i-ésimo elemento da heap H para x.

2. **van Emde Boas:** Implementar uma van Emde Boas para armazenar valores no intervalo $0, \ldots, 2^b - 1$. O espaço total usado pela sua estrutura de dados com n elementos deve ser de $O(n)$. Podem usar implementações prontas de tabela de dispersão para esta estrutura, mas precisa simplementar o table doubling.

   Operações:

   (a) INCLUIR(van Emde Boas E, inteiro x): inclui x em E

   (b) REMOVER(van Emde Boas E, inteiro x): remove x de E

   (c) SUCESSOR(van Emde Boas E, inteiro x): retorna o sucessor de x em E (x não precisa estar em E)

   (d) PREDECESSOR(van Emde Boas E, inteiro x): retorna o predecessor de x em E (x não precisa estar em E)

3. **Persistência parcial em árvores balanceadas:** Implementar uma árvore rubro-negra com persistência parcial.

   Operações:

   (a) INCLUIR(árvore T, inteiro x): inclui x em T e retorna um identificador da versão da árvore após a inclusão.

(b) REMOVER(árvore T, inteiro x): remove o nó com chave x de T e retorna um identificador da versão da árvore após a inclusão.

(c) SUCESSOR(árvore T, inteiro x, versão v): retorna o sucessor de x em T na versão v (x não precisa estar em T)

(d) PREDECESSOR(van Emde Boas E, inteiro x, versão v): retorna o predecessor de x em E (x não precisa estar em T)

4. **Manutenção de arquivos ordenados:** Implementar um vetor para manter arquivos ordenados em tempo $O(\log^2 n)$ amortizado por operação. O seu vetor deve ser dinâmico e deve estar entre 25% e 100% cheio, implementando table doubling and halving. Cada elemento deve ser mantido por um nó que mantém o seu índice no vetor sempre atualizado.

   Operações:

   (a) INCLUIRDEPOIS(Vetor A, inteiro x, nó v): inclui x em A imediatamente antes de v.

   (b) INCLUIRANTES(Vetor A, inteiro x, nó v): inclui x em A imediatamente depois de v.

   (c) REMOVER(Heap A, nó v): remove v de A

5. **Rotulação de listas:** Implementar um lista encadeada em que podemos verificar a ordem entre dois elementos da lista em tempo $O(1)$.

   Operações:

   (a) Temp

6. **Bit 1 mais significativo:** Dado uma palavras de computador, calcular o índice do bit 1 mais significativo desta palavra em tempo $O(1)$.

   Operações:

   (a) Temp

7. **Range Minimum Query (RMQ):** Pré-processar um vetor de tamanho n usando espaço $O(n)$ para responder perguntas de encontrar elemento mínimo em subsequências do vetor.

   Operações:

   (a) Temp

8. **Link-cut trees:** Implementar link-cut trees para verificar a conectividade entre vértices em árvores.

   Operações:

   (a) Temp

9. **Split e concatenar em árvores binárias de busca:** Implementar link-cut trees para verificar a conectividade entre vértices em árvores.

   Operações:

   (a) Temp