Ata da resolução de exercício

Exercício[1]: 16.1-3
Aluno: Paulo Gurgel Pinheiro
pinheiro@ic.unicamp.br
Disciplina: Complexidade de Algoritmos - MO417
Data: 02 de Maio de 2009

1 Enuciado exercício 16.1-3

Vamos supor que temos um conjunto de atividades para programar entre um grande número de salas de conferências. Desejamos programar todas as atividades usando o mínimo possível de salas de conferências. Forneça um algoritmo guloso eficiente para determinar que atividade deve usar casa sala de conferências.

(Isso também é conhecido como o ***problema de colorir grafos de intervalos.*** Podemos criar um grafo de intervalos cujos vértices sejam as atividades dadas e cujas arestas conectem atividades incompatíveis. O menor número de cores necessárias para colorir cada vértice de tal modo que dois vértices adjacentes nunca recebam a mesma cor corresponde a encontrar o menor número de salas de conferências necessárias para programar todas as atividades dadas.)

1.1 Organização

Esta ata está organizada da seguinte maneira: a primeira seção descreve a solução proposta, a segunda seção um exemplo demonstrativo e a terceira seção contém o algoritmo da solução.
Para entender a solução o leitor pode simplesmente escolher uma das seções desta ata. Elas foram escritas de forma que também permitissem o entendimento caso fossem lidas separadamente.

2 Solução proposta pelo Prof. João Meidanis

A descrição da solução proposta segue abaixo:

- Entrada do problema
  - A entrada é um vetor S de n **atividades**, $S = \{a_1, ..., a_n\}$.
  - Cada atividade possui um tempo de início i_n e um tempo de fim f_n.
- Estruturas de dados utilizada
  - Um vetor M com $2n$ posições para armazenar os **marcadores** tempos iniciais e finais das atividades (esse vetor pode ser visto como uma linha do tempo). $M[indice] = (flag, atividade)$, onde $flag$ pode ser $incio$ ou fim.
  - Um **heap mínimo** H com n elementos, $\{1...n\}$. Cada nó do heap representa uma sala. No pior dos casos as n salas serão utilizadas (atividades nunca compatíveis).
  - Um **vetor** A que armazenará as **alocações**, $[indice] = (sala, atividade)$.

[1] Exercício extraido de [1]

- Solução

  1. Ordenar as atividades do **vetor** S por tempo de início ou assumir que já estejam ordenadas.;
  2. Criar o **vetor** M com os marcadores. Exemplo: M=$\{(inicio, 1), (inicio, 2), (fim, 1), (inicio, 3), (fim, 2), (fim, 3)\}$
  3. Criar o **heap mínimo** de n nós identificados de 1 a n;
  4. Criar o **vetor** A de alocações.

  5. Pecorrer o **vetor** M
     (a) Se o elemento for de início: $M[indice] = (inicio, a_i)$
        - Então **atividade** em questão é a_i;
        - A **sala** que será alocada para essa atividade será a **raiz do heap** H.[2]
        - Adiciona-se no **vetor** A a alocação (sala,ai).

     (b) Se o elemento for de fim: $M[indice] = (fim, a_i)$
        - Então **atividade** em questão é a_i.
        - A **sala**, quando anteriormente alocada, foi armazenada com sua atividade correspondente no **vetor** A. Procura-se nesse vetor essa associação, passando como entrada a atividade a_i.
        - Desaloca a sala apagando sua associação em A.
        - Adiciona no **heap mínimo** H o número da sala que estava associada a essa atividade que acabará.

  6. Ao final do **vetor** M todas as atividades terão sido alocadas no menor número de salas possível.

3 Exemplo

Para auxiliar no entendimento da solução, essa seção descreverá um exemplo prático para um conjunto de 5 atividades.

3.0.1 Início

- Conjunto de atividades: $S = \{a_1, a_2, a_3, a_4, a_5\}$.
- Número de atividades: $n = 5$.
- Número de salas: 5 (No pior caso, 5 salas serão utilizadas).

A Figura 1 mostra as atividades dispostas em uma linha do tempo, tal que i_j e f_j são os tempos de início e de fim de uma tarefa j, respectivamente.

3.0.2 Procedimento

1. Ordena-se o **vetor** S por tempo de início. $S = \{a_1, a_2, a_3, a_4, a_5\}$. Ou assume-se já estar ordenado, como é o caso;
2. Cria-se o **vetor** M. M= $\{(inicio, 1), (inicio, 2), (inicio, 3), (fim, 1), (fim, 3), (inicio, 4), (fim, 2), (inicio, 5), (fim, 4), (fim, 5)\}$;

[2]Como o heap é mínimo e as salas numeradas de 1 a n, sempre estará na raiz aquela sala de menor número.

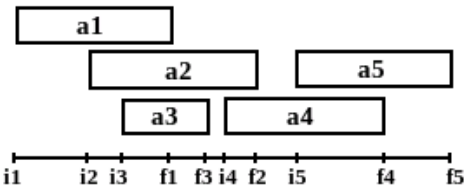
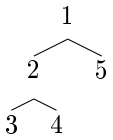


Figura 1: Atividades com marcações de tempo de início e fim.

3. Cria-se um **Heap mínimo** com 5 nós. $H = \{1, 2, 3, 4, 5\}$ referente às salas.



4. Pecorre-se o **vetor** M

1.Verifica primeiro elemento no vetor M. Acha $(inicio, 1)$).
2.Retira o nó raiz do heap. Nó retirado = 1.
3.Adiciona ao vetor A o elemento associaçao (atividade 1, sala 1).

1.Verifica próximo elemento no vetor M. Acha $(inicio, 2)$.
2.Retira o nó raiz do heap. Nó retirado = 2.
3.Adiciona ao vetor A o elemento associaçao(atividade 2, sala 2).

1.Verifica próximo elemento no vetor M. Acha $(inicio, 3)$.
2.Retira o nó raiz do heap. Nó retirado = 3.
3.Adiciona ao vetor A o elemento associaçao (atividade 3, sala 3).

1.Verifica próximo elemento no vetor M. Acha $(final, 1)$.
2.Verifica no vetor de A que sala estava associada à atividade 1. Acha (atividade 1, sala 1).
3.Apaga essa alocação do vetor A.
4.Adiciona o nó 1 no heap mínimo.

1.Verifica próximo elemento no vetor M. Acha $(final, 3)$.
2.Verifica no vetor de A que sala estava associada à atividade 3. Acha (atividade 3, sala 3).
3.Apaga essa alocação do vetor A.
4.Adiciona o nó 3 no heap mínimo.

1.Verifica próximo elemento no vetor M. Acha $(inicio, 4)$.
2.Retira o nó raiz do heap. Nó retirado = 1.
3.Adiciona ao vetor A o elemento associaçao (atividade 4, sala 1).

1.Verifica próximo elemento no vetor M. Acha $(final, 2)$.
2.Verifica no vetor de A que sala estava associada à atividade 2. Acha (atividade 2, sala 2).
3.Apaga essa alocação do vetor A.
4.Adiciona o nó 2 no heap mínimo.

1.Verifica próximo elemento no vetor M. Acha $(inicio, 5)$.
2.Retira o nó raiz do heap. Nó retirado = 2.

3.Adiciona ao vetor A o elemento associaçao(atividade, sala)= (atividade 5, sala 2).

1.Verifica próximo elemento no vetor M. Acha $(final, 4)$.
2.Verifica no vetor de A que sala estava associada à atividade 4. Acha (atividade 4, sala 1).
3.Apaga essa alocação do vetor A.
4.Adiciona o nó 1 no heap mínimo.

1.Verifica próximo elemento no vetor M. Acha $(final, 5)$.
2.Verifica no vetor de A que sala estava associada à atividade 5. Acha (atividade 5, sala 2).
3.Apaga essa alocação do vetor A.
4.Adiciona o nó 2 no heap mínimo.

4 Algoritmo

Nesta seção, será apresentado o algoritmo da solução. Tal algoritmo recebe como entrada o conjunto S de atividades.

Algoritmo 1: Algoritmo para o exercício 16.1-3

```
1  S←ORDENA-ATIVIDADES(S); /*Ordena as atividades de S por tempo de início */
2  n←S.tamanho;
3  M←CRIA-MARCADORES(S); /*Vetor que contém os marcadores (inicio, n) e (fim, n)*/
4  H←BUILD-MIN-HEAP(n);/*Cria Heap min com elementos de 1 a n*/
5  A←A[n];/*Vetor onde vão sendo registradas as alocações (atividade, sala)*/

6  for j←1 to M.tamanho do
7      if M[j].flag='inicio' then
8          atividade←M[j].atividade;
9          sala←REMOVE-RAIZ-HEAP(H);
10         associacao←(atividade, sala);
11         A.adiciona(associacao);
12         print('atividade:'+ atividade + 'na sala:'+ sala);
13     else
14         if M[j].flag='fim' then
15             atividade=M[j].atividade;
16             sala←BUSCA-SALA(A,atividade);
17             A.remove(sala,atividade);
18             ADICIONA-ELEMENTO-HEAP(H,sala)
19
20     end
21 end
```

- Na linha 1, as atividades de S são ordenadas por tempo de início;
- Na linha 2, n é a quantidade de atividades do vetor S;
- Na linha 3, é criado os marcadores da linha do tempo, tal que $M[j] = (inicio|fim, atividade)$. Exemplo: $M[3] = (inicio, 2)$, significa que a terceira marca no tempo é o início da atividade 2;

- Na linha 4, cria-se um heap mínimo com n elementos de chaves de 1 a n. Cada nó representa uma sala;
- Na linha 5, inicializa o vetor que irá armazenar a alocação de uma atividade em uma sala.
- Da linha 6 a 20, o for varre o vetor M (que representa uma linha do tempo).
- Da linha 7 a 12, Se o marcador é de início:
  - Linha 8, inicializa a variável *atividade* com o número da atividade do marcador.
  - Linha 9, inicializa a variável *sala* com o valor da raiz do heap mínimo que foi removida (era a primeira sala que estava disponível).
  - Linha 10, inicializa o elemento *associacao* com a atividade da linha 8 e a sala da linha 9.
  - Linha 11, adiciona no vetor de associações A a *associacao*.
  - Linha 12, imprime a alocação.
- Da linha 13 a 19, Se o marcador é de fim:
  - Linha 15, inicializa a variável *atividade* com o número da atividade do marcador.
  - Linha 16, busca no vetor de alocações A, a sala que esta associada a atividade referida e inicializa a variável *sala*.
  - Linha 17, desaloca a sala da atividade, removendo o elemento do vetor A.
  - Linha 18, adiciona no Heap o número dessa sala que agora ficou livre;

4.1 Procedimentos

- ORDENA-ATIVIDADES(S) Pode-se assumir que o vetor S de atividades já esteja ordenado crescentemente pelo tempo de início das atividades. Se não, é possível ordená-lo em tempo O(nlgn).
- CRIA-MARCADORES(S) Pecorre o vetor S já ordenado criando marcadores $(inicio, a_n)$ e (fim, a_n) em ordem de acontecimento para as n atividades.
- BUILD-MIN-HEAP(n) Constroi um heap mínimo em O(nlgn).
- REMOVE-RAIZ-HEAP(H) Remove o nó raiz do heap H e executa um procedimento para reorganizar o heap. Tal procedimento é semelhante ao MAX-HEAPIFY descrito no Capítulo 6, página 106 de [1], porém adaptado para um heap mínimo. Essa operação é executada no tempo O(lgn).
- BUSCA-SALA(A, *atividade*) Procedimento que retorna a sala em que a atividade estava alocada. Tal procedimento deve ser implementado utilizando um hash e ser executado em média em O(1).
- ADICIONA-ELEMENTO-HEAP(H, *elemento*) Adiciona um elemento ao heap H e executa um procedimento para reorganizar o heap. Tal procedimento é semelhante ao MAX-HEAPIFY descrito no Capítulo 6, página 106 de [1], porém adaptado para um heap mínimo. Essa operação é executada no tempo O(lgn).

Referências

[1] T. Cormen, C. Leiserson, R. Rivest, and Stein C. *Algoritmos: teoria e prática.* Editora Campus, 2 edition, 2002.