Documentação do Trabalho Prático I da disciplina de Algoritmos I

Cecília Kind - 2019054420

1. Introdução:

Esta documentação se refere ao Trabalho Prático I da disciplina de Algoritmos I. A proposta do trabalho é desenvolver um programa para auxiliar na alocação de pessoas para postos de vacinação em um dia seguindo os critérios de prioridade estabelecidos.

O trabalho foi implementado em C + + seguindo os princípios da programação orientada a objetos. As estruturas utilizadas foram vetores, para o armazenamento da lista de indivíduos e dos postos, e uma lista para a alocação de pessoas em cada posto. As demais seções presentes neste documento abordam a modelagem do programa (seção 2), as estruturas de dados e algoritmos (seção 3) e a complexidade assintótica dos algoritmos (seção 4).

2. Modelagem:

A implementação do programa foi dividida nas seguintes classes: Main, Relação, Pessoa e Posto.

A Main é responsável pela leitura dos arquivos, a criação dos vetores de pessoas e postos, o cálculo e ordenação das listas de preferência de postos para cada pessoa, e pela ordenação do vetor de pessoas para que as propostas sejam feitas antes pelas pessoas com maior prioridade.

A classe Relação contém o loop principal do programa, que para cada pessoa da lista ordenada de pessoas faz uma proposta para cada posto na ordem de preferência, até a pessoa ser alocada ou acabarem as opções de posto. Nestes dois casos a pessoa é removida da lista permanentemente. Uma pessoa é alocada em um posto se ainda há vagas disponíveis, e como as propostas sempre ocorrem do indivíduo de maior prioridade para o de menor prioridade, trocas na alocação dos indivíduos de um posto nunca ocorrem. O programa termina quando não há mais pessoas na lista.

A classe Pessoa armazena as informações sobre cada pessoa e uma lista de pares que representam a distância e o ID de cada posto na ordem de preferência.

A classe Posto possui as informações sobre cada posto e uma lista com as pessoas que estão alocadas naquele posto.

3. Estruturas de dados e algoritmos

- As principais estruturas utilizadas
  lista_pessoas: Vetor de pessoas de tamanho N, em que cada objeto de pessoa possui uma lista de pares de tamanho M (preferencia_posto), com as informações sobre cada posto (distância e ID) na ordem de preferência. O vetor de pessoas é ordenado de menor para maior prioridade.

  lista_postos: Vetor de postos de tamanho M, em que cada objeto de posto possui um vetor de pessoas alocadas de tamanho da capacidade do posto (pessoasAlocadas).

- Algoritmos auxiliares de leitura, armazenamento e ordenação:

  main, Le_Arquivo e caclula_lista_proximidade:
  for (indivíduo no vetor de Pessoas):
  for(posto no vetor de postos):
  Calcula distância entre o indivíduo e o posto
  Adiciona o posto no vetor de prioridade do indivíduo
  Ordena vetor de prioridade dos postos do indivíduo
  Ordena vetor de indivíduos na ordem de preferência dos postos de menor preferência para maior

- Algoritmos principais:

  Casamento(Lista de indivíduos ordenados por preferência, Lista de postos)
  While (vetor de indivíduos não está vazio)
  Chama função proposta para o indivíduo de maior preferência:
  if (proposta não é aceita)
  Remove posto que foi proposto da lista do indivíduo.
  if (não há posto na lista do indivíduo)
  Remove pessoa do vetor de pessoas.
  if (proposta é aceita)
  Remove pessoa do vetor de pessoas.

  for (cada posto na lista de postos):
  Imprime (Id do posto e Id das pessoas alocadas)

  Proposta(Pessoa que faz a proposta, Lista de postos):
  if (há vaga no primeiro posto da lista de preferência da pessoa)
  Remove posto da lista do indivíduo
  Aloca indivíduo na lista do posto
  Retorna verdadeiro
  else:
  Retorna falso

- Corretude do algoritmo:
  Observações sobre o algoritmo:
  a. Cada indivíduo propõe para os postos em ordem decrescente de preferência.
  b. Os indivíduos de maior prioridade sempre realizam as propostas antes.
  c. O indivíduo que faz a proposta para um posto com vagas é sempre alocado.
  d. Depois que uma pessoa é alocada a um posto, ele não é retirado mais.
  e. Se um indivíduo P não aparece na solução, P deve ter proposto para todos os postos.

  Uma alocação será instável e inválida quando alguma das seguintes situações ocorrer:

• Uma pessoa P1 está alocada para um posto B, mas há um posto A mais próximo que possui vagas disponíveis ou está alocando uma pessoa p2 de menor idade.

Provando a estabilidade do algoritmo por contradição:
De acordo com a afirmação a, se A é mais próximo de P1 do que B, então P1 deve ter proposto primeiro a A. Mas se P1 não foi alocado a A, então A não tinha mais vagas e todas as pessoas alocadas tinham preferência sobre P1, o que é confirmado pelas observações b e c. Logo, A não poderia ter alocado P1 e temos uma contradição.

• Há uma pessoa sem alocação mas ainda existem vagas em um dos postos.

Provando a estabilidade do algoritmo por contradição:
De acordo com a afirmação e, se a pessoa não foi alocada em nenhum posto ela deve ter feito proposta para todas as opções de posto. Mas de acordo com as afirmativas c e d, se a alocação em um posto é permanente e propostas feitas com postos com vagas são sempre alocados, então a pessoa não deve ter feito a proposta ao posto, o que contradiz a afirmação e.

• A alocação deve ser simultaneamente a melhor possível para todas as pessoas.

De acordo com a, b e c, o indivíduo de maior prioridade sempre realiza suas propostas antes, começando pelos seus postos de preferência, e uma vez alocado não pode ser retirado. Assim, a alocação é feita na ordem de prioridade de acordo com a melhor escolha possível para o indivíduo.

4. Análise de complexidade:

Seja M o número de postos e N o número de pessoas:

Complexidade de espaço:
Principais estruturas:

- Vetor de pessoas de tamanho N que armazenam cada uma uma lista de tamanho M de preferência dos postos: O(M*N)

- Vetor de postos que armazenam uma lista de pessoas alocadas do tamanho da lotação máxima. A complexidade é O(M + soma das capacidades máximas dos postos).

Complexidade de tempo:
Principais funções:

- Calcula_dist: retorna a distância entre dois pontos: complexidade de tempo O(1)

- caclula_lista_proximidade: Para cada um dos M postos presente na lista de preferências de uma pessoa, chama a função para calcular a distância e ordena com o sort da biblioteca padrão. M * O(1) + M*logM = O(M* logM)

- leArquivo: Além da leitura e armazenamento dos vetores de pessoas e postos, O(M + N), chama a função caclula_lista_proximidade para cada uma das N pessoas: N * O(MlogM) = O(NM* logM)

- proposta: Realizar apenas operações constantes como remover o primeiro elemento de uma lista, adicionar uma pessoa a um posto e comparar a capacidade do posto com o número de pessoas alocadas: O(1).

- Relacao: loop principal do programa, no pior caso devemos percorrer toda a lista de prioridades de postos de tamanho M para cada um dos N indivíduos: O(M*N).

- main: chama a função leArquivo (O(NM* logM)), ordena o vetor de pessoas O(N*logN) e chama a função casamento, O(M*N). Assim, a complexidade de tempo do programa é **O(NMlogM)**

5. Conclusão:

Este trabalho teve como base os conceitos do algoritmo de Gale–Shapley visto em aula. A partir dos conhecimentos de casamento estável e das variações do algoritmo estudadas foi possível desenvolver uma solução simples e estável para o problema de alocação de pessoas em postos de saúde. A complexidade de tempo do programa é O(NMlogM) e de espaço O(M*N).