Estruturas de dados

Prof. Rafael Alves Bonfim de Queiroz
rafael.queiroz@ufop.edu.br

Conteúdo

Introdução

- As estruturas de dados são o coração de qualquer programa sofisticado
- A seleção da estrutura de dados correta pode fazer uma enorme diferença na complexidade da implementação resultante
- Escolha a representação de dados correta e sua tarefa será fácil de programar
- As operações abstratas das estruturas de dados mais importantes: pilhas, filas, dicionários, filas de prioridade e conjuntos
- Os usuários de linguagens modernas orientadas a objetos, como C++ e Java, têm implementações dessas estruturas de dados básicas disponíveis usando classes de biblioteca padrão

Pilhas

- Pilhas e filas são contêineres, onde os itens são recuperados de acordo com a ordem de inserção, independente do conteúdo
- As pilhas mantêm a ordem de último a entrar e o primeiro a sair
- As operações abstratas em uma pilha incluem:
  - Push(x,s): Insira o item x no topo da pilha s
  - Pop(s): Devolva (e remova) o item do topo da pilha s
  - Initialize(s): Crie uma pilha vazia
  - Cheio(s), Vazio(s): Testa se a pilha pode aceitar mais pushs ou pops, respectivamente.
- Observe que não há operação de busca de elementos definida em pilhas e filas padrão

- A implementação mais fácil usa um array com uma variável de índice para representar o topo da pilha
- Uma implementação alternativa, usando listas vinculadas, é melhor porque não pode transbordar
- Pilhas modelam naturalmente pilhas de objetos, como pratos de jantar
  - Depois que um novo prato é lavado, ele é empilhado para o topo da pilha
  - Quando alguém está com fome, um prato limpo é retirado do topo

- A ordem da pilha é importante no processamento de qualquer estrutura aninhada corretamente
- Isso inclui fórmulas entre parênteses (empilhe um "(", retire ")"), chamadas de programa recursivas (push em uma entrada de procedimento, pop em uma saída de procedimento) e percursos de grafos em profundidade (push() na descoberta de um vértice, pop() ao deixá-lo pela última vez)

Filas

- As filas mantêm a ordem de entrada e saída
- Isso parece mais justo do que o último a entrar, o primeiro a sair, e é por isso que as filas nas lojas são organizadas como filas em vez de pilhas
- Baralhos de cartas podem ser modelados por filas, uma vez que tiramos as cartas do topo do baralho e as adicionamos de volta ao fundo.
- As filas FIFO serão usadas na implementação da busca em largura em grafos

Filas

- As operações abstratas em uma fila incluem:
  - Enqueue(x,q) - insira o item x no final da fila q
  - Dequeue(q) - retorna (e remove) o item da frente da fila q
  - Initialize(q), Full(q), Empty(q) - análoga a essas operações em pilhas
- As filas são mais difíceis de implementar do que as pilhas, porque a ação acontece em ambas as extremidades
- A implementação mais simples usa um array, inserindo novos elementos em uma extremidade e movendo todos os elementos restantes para preencher o espaço vazio criado em cada fila

- No entanto, é um grande desperdício mover todos os elementos em cada desenfileiramento
- Podemos manter índices para o primeiro (cabeça) e último (cauda) elementos no array/fila e fazer todas as operações localmente
- Não há razão para que devamos limpar explicitamente as células usadas anteriormente, embora deixemos um rastro de lixo para trás dos itens anteriormente retirados da fila
- As filas são uma das poucas estruturas de dados mais fáceis de programar usando listas vinculadas do que matrizes, porque eliminam a necessidade de testar a condição de limites do tamanho

- As filas circulares nos permitem reutilizar esse espaço vazio.
- O ponteiro para a frente da lista está sempre atrás do ponteiro para trás
- Quando a fila estiver cheia, os dois índices apontarão para elementos vizinhos ou idênticos
- Existem várias maneiras possíveis de ajustar os índices para filas circulares
- A solução mais fácil distingue o cheio do vazio mantendo uma contagem de quantos elementos existem na fila

Dicionários

- Os dicionários permitem a recuperação baseada em conteúdo, ao contrário da recuperação baseada em posição de pilhas e filas
- Eles suportam três operações principais:
  - Insert(x,d) - Insere o item x no dicionário d
  - Delete(x,d) - Remove o item x (ou o item apontado por x) do dicionário d
  - Search(k,d) - Retorna um item com a chave k se existir no dicionário d
- Exemplos de maneiras de implementar dicionários: listas vinculadas classificadas/não classificadas, matrizes classificadas/não classificadas e uma floresta cheia de árvores aleatórias, esparsas, AVL e vermelho-pretas - sem mencionar todas as variações em hash
- O principal problema na análise de algoritmos é o desempenho, ou seja, alcançar o melhor trade-off possível entre os custos dessas três operações
- Geralmente queremos na prática é a maneira mais simples de fazer o trabalho dentro das restrições de tempo dadas
- A resposta certa depende de quanto o conteúdo do seu dicionário

Dicionários estáticos

- Essas estruturas são construídas uma vez e nunca mudam
- Assim, eles precisam oferecer suporte à pesquisa, mas não à inserção na exclusão
- A resposta certa para dicionários estáticos é tipicamente um array
- A única questão real é se deve mantê-lo classificado, a fim de usar a pesquisa binária para fornecer consultas rápidas de associação
- A menos que você tenha restrições de tempo, provavelmente não vale a pena usar a pesquisa binária até n $>$ 100 ou mais

Dicionários semidinâmicos

- Essas estruturas suportam consultas de inserção e pesquisa, mas não a exclusão
- Se soubermos um limite superior para o número de elementos a serem inseridos, podemos usar um array, mas, caso contrário, devemos usar uma estrutura vinculada
- As tabelas de hash são excelentes estruturas de dados de dicionário, especialmente se a exclusão não precisar ser suportada
- A ideia é aplicar uma função à chave de busca para que possamos determinar onde o item aparecerá em um array sem olhar para os outros itens
- Para tornar a tabela de tamanho razoável, devemos permitir colisões, duas chaves distintas mapeadas para o mesmo local

Dicionários semidinâmicos

- Os dois componentes do hash são
  1. definir uma função hash para mapear chaves para inteiros em um determinado intervalo
  2. configurar um array tão grande quanto esse intervalo, de modo que o valor da função hash possa especificar um índice
- A função hash básica converte a chave em um inteiro, e toma o valor desse inteiro mod do tamanho da tabela de hash
- Selecionar um tamanho de tabela para ser um número primo (ou pelo menos evitar números compostos óbvios como 1.000) é útil para evitar problemas

Dicionários totalmente dinâmicos

- As tabelas de hash também são ótimas para dicionários totalmente dinâmicos, desde que usemos o encadeamento como mecanismo de resolução de colisão
- Aqui associamos uma lista encadeada a cada local de tabela, portanto, inserção, exclusão e consulta reduzem ao mesmo problema em listas encadeadas
- Se a função hash faz um bom trabalho, as m chaves serão distribuídas uniformemente em uma tabela de tamanho n, de modo que cada lista será curta o suficiente para pesquisar rapidamente

Filas prioritárias

- As filas de prioridade são estruturas de dados em conjuntos de itens que suportam três operações
  - Insert(x,p) - Insere o item x na fila de prioridade p
  - Maximum(p) - Retorna o item com a maior chave na fila de prioridade p
  - ExtractMax(p) - Retorna e remove o item com a maior chave em p
- As filas de prioridade são usadas para manter agendas e calendários
- Elas governam quem vai em seguida em simulações de aeroportos, estacionamentos e afins, sempre que precisamos agendar eventos de acordo com um relógio

Filas prioritárias

- As filas de prioridade são usadas para agendar eventos nos algoritmos de linha de varredura comuns à geometria computacional
- Normalmente, usamos a fila de prioridade para armazenar os pontos que ainda não encontramos, ordenados pela coordenada x, e empurramos a linha um passo de cada vez
- A implementação mais famosa de filas de prioridade é o heap binário, que pode ser mantido eficientemente de maneira top-down ou bottom-up
- As pilhas são muito espertas e eficientes, mas podem ser difíceis de acertar sob pressão de tempo
- Muito mais simples é manter um array ordenado, especialmente se você não estiver executando muitos inserções

Conjuntos

- Conjuntos (ou subconjuntos mais estritamente falando) são coleções não ordenadas de elementos extraídos de um determinado conjunto universal U
- As estruturas de dados de conjunto se distinguem dos dicionários porque há pelo menos uma necessidade implícita de codificar quais elementos de U não estão no subconjunto fornecido
- As operações básicas em subconjuntos são:
  - Member(x,S) - Um item x é um elemento do subconjunto S?
  - União(A,B) - Construa o subconjunto união de todos os elementos no subconjunto A ou no subconjunto B
  - Intersecção(A,B) - Construa o subconjunto intersecção de todos os elementos do subconjunto A e do subconjunto B
  - Insert(x,S), Delete(x,S) - Insira/exclua o elemento x no/do subconjunto S

Conjuntos

- Para conjuntos de um universo grande ou ilimitado, a solução óbvia é representar um subconjunto usando um dicionário
- O uso de dicionários ordenados torna as operações de união e interseção muito mais fáceis, basicamente reduzindo o problema de mesclar dois dicionários ordenados
- Um elemento está na união se aparecer pelo menos uma vez na lista mesclada e na interseção se aparecer exatamente duas vezes
- Para conjuntos extraídos de universos pequenos e imutáveis, os vetores de bits fornecem uma representação conveniente
- Um vetor ou matriz de n bits pode representar qualquer subconjunto S de um universo de n elementos.
- As operações de inserção e exclusão de elementos simplesmente invertem o bit apropriado.
- Intersecção e união são feitas aplicando operações lógicas "and" ou "or"

A biblioteca de modelos padrão C++

- Os templates são o mecanismo do C++ para definir objetos abstratos que podem ser parametrizados por tipo
- A biblioteca de Modelos Padrão C++ (STL) fornece implementações de todas as estruturas de dados definidas acima e muito mais

Biblioteca de modelos padrão C++: STL

- Stack - os métodos incluem S.push(), S.top(), S.pop() e S.empty()
  - top retorna, mas não remove o elemento de cima
  - pop remove, mas não retorna o elemento
  - As implementações vinculadas garantem que a pilha nunca fique cheia
- Queue - os métodos incluem Q.front(), Q.back(), Q.push(), Q.pop() e Q.empty() e têm as mesmas idiossincrasias da pilha
- Dicionários - STL contém uma ampla variedade de contêineres, incluindo mapa de hash, um contêiner associativo com hash que vincula chaves a itens de dados
  - Os métodos incluem H.erase(), H.find() e H.insert()

Biblioteca de modelos padrão C++

- Filas de prioridade - Fila de prioridade declarada<int> Q
  - os métodos incluem Q.top(), Q.push(), Q.pop() e Q.empty()
- Conjuntos - representados como contêineres associativos classificados, declarados set<chave, comparação> S
  - Os algoritmos de conjunto incluem união de conjuntos e interseção de conjuntos, bem como outros operadores de conjuntos padrão