| Estrutura | Implementação | Inserção | Remoção | Acesso | Busca | Uso de Memória | Vantagens | Desvantagens | Aplicações |
|---|---|---|---|---|---|---|---|---|---|
| **ArrayList** | Array dinâmico | $O(1)$ **amortizado** (fim) $O(n)$ (meio) | $O(n)$ (devido ao deslocamento) | $O(1)$ (índice direto) | $O(n)$ (busca linear) | Usa um array fixo e redimensiona conforme necessário | - Acesso rápido por índice- Gerenciamento automático de tamanho | - Remoção e inserção no meio são custosas- Resize consome tempo e memória | Estruturas dinâmicas de listas, editores de texto (lista de comandos) |
| **LinkedList** | Lista encadeada (nós com ponteiros) | $O(1)$ **no início e fim** $O(n)$ no meio | $O(1)$ **no início e fim** $O(n)$ no meio | $O(n)$ (precisa percorrer a lista) | $O(n)$ (busca linear) | Usa mais memória (ponteiros para próximo e anterior) | - Inserção e remoção eficientes em qualquer ponto- Cresce dinamicamente sem precisar realocar memória | - Acesso a um índice é lento- Maior uso de memória | Implementação de listas dinâmicas, manipuladores de histórico (navegadores) |
| **Fila (FIFO)** | Array ou lista encadeada | $O(1)$ (fim) | $O(1)$ (início) | $O(n)$ (se precisar percorrer) | $O(n)$ (se não usar hash auxiliar) | Depende da implementação, pode usar array fixo ou lista dinâmica | - Estrutura ideal para processamento por ordem de chegada | - Busca e acesso aleatório são lentos | Gerenciamento de tarefas, sistemas de impressão, escalonamento de processos |
| **Pilha (LIFO)** | Array ou lista encadeada | $O(1)$ (topo) | $O(1)$ (topo) | $O(n)$ (se precisar percorrer) | $O(n)$ (se não usar hash auxiliar) | Depende da implementação, pode usar array fixo ou lista dinâmica | - Fácil de implementar- Estrutura ideal para operações sequenciais reversíveis | - Difícil acessar elementos antigos diretamente | Algoritmos recursivos, desfazer (Ctrl+Z), backtracking |
| **Tabela Hash** | Array de buckets + função hash | $O(1)$ (na maioria dos casos) | $O(1)$ (se não houver colisões) | $O(1)$ (com boa função hash) | $O(1)$ a $O(n)$ (se houver muitas colisões) | Usa mais memória que arrays simples (slots extras para colisões) | - Busca e acesso muito rápidos- Ideal para armazenar chaves únicas | - Pode sofrer colisões- Implementação mais complexa | Banco de dados, caches, tabelas de símbolos em compiladores |
| **Políticas de Cache** (FIFO, LRU, LFU) | Array, lista encadeada, hashmaps | $O(1)$ com lista encadeada + hash | $O(1)$ com lista encadeada + hash | $O(1)$ com hash | $O(1)$ com hash | Depende da implementação, pode usar hash + lista dupla | - Acelera o acesso a dados frequentemente usados | - Requer estrutura de dados eficiente para boa performance | Sistemas operacionais (gestão de memória), navegadores (cache de páginas) |