Listas

Listas Auto-Organizáveis

Prof. Edson Alves - UnB/FGA

2018

Sumário

1. Definição

2. Eficiência das listas auto-organizáveis

3. Implementação

Definição

Listas auto-organizáveis

- Listas encadeadas e duplamente encadeadas não são eficientes ($O(N)$) em relação à busca de elementos
- Diferentes organizações dos elementos da lista podem melhorar o tempo de execução dos algoritmos de busca
- Estas organizações geram as listas auto-organizáveis, que são listas que se alteram dinamicamente a cada busca
- Em geral, estas listas não contém repetições de elementos (isto é, não há repetição de valores nos campos `info` dos nós
- Para efeitos de análise e implementação, se uma busca não localizar uma dada informação, a mesma será inserida na lista ao final da busca
- O comportamento dinâmico destas listas faz com que as buscas ganhem desempenho através do posicionamento estratégico dos elementos já localizados

Organizações possíveis

1. Move o elemento localizado para o início da lista
2. Trocar o elemento localizado com o seu antecessor, caso não seja o primeiro elemento da lista
3. Ordenar a lista de acordo com a frequência de acesso de cada elemento
4. Ordenar a lista de acordo com um critério pré-estabelecido (ordem alfabética, lexicográfica, etc)

Observações sobre as formas de organização

- As três primeiras formas incluem um novo nó ao final da lista se a informação procurada não for localizada
- Já a quarta forma adiciona o novo nó de acordo com o critério estabelecido
- As três primeiras formas visam fazer com que o elemento seja localizado o mais cedo possível
- A última forma é útil em buscas de elementos que não estão na lista, uma vez que sua organização subjacente pode suspender a busca quando for o caso
- Se uma informação não consta na lista, nas três primeiras formas a lista deve ser percorrida na íntegra, o que não acontece na quarta forma

Exemplo das formas de organização

Elemento acessado	Nenhuma forma	Mover para o início	Transposição	Frequência	Ordenação
A	A	A	A	A	A
C	AC	AC	AC	AC	AC
B	ACB	ACB	ACB	ACB	ABC
C	ACB	CAB	CAB	CAB	ABC
D	ACBD	CABD	CABD	CABD	ABCD
A	ACBD	ACBD	ACBD	CABD	ABCD
D	ACBD	DACB	ACDB	CADB	ABCD
A	ACBD	ADCB	ACDB	ACDB	ABCD
C	ACBD	CADB	CADB	ACDB	ABCD
A	ACBD	ACDB	ACDB	ACDB	ABCD
C	ACBD	CADB	CADB	ACDB	ABCD
C	ACBD	CADB	CADB	CADB	ABCD
E	ACBDE	CADBE	CADBE	CADBE	ABCDE
E	ACBDE	ECADB	CADEB	CADEB	ABCDE

Eficiência das listas auto-organizáveis

Ordenação estática ótima

- A eficiência das listas auto-organizáveis é computada através da comparação da forma de organização escolhida e a ordenação estática ótima
- Na ordenação estática ótima, os elementos são ordenados por frequência, e elementos com mesma frequência são ordenados alfabeticamente
- A eficiência é calculada através da razão entre o número de comparações feitas e o número de comparações possíveis
- O número total de comparações é a soma dos tamanhos da lista em cada etapa
- Este tamanho é o mesmos para todas as formas: o que difere é o número de comparações realizadas

Cálculo de eficiência: Ordenação Estática Ótima

Elemento a ser acessado	Lista	Tamanho/ Subtotal	Algoritmo	Número de comparações	Subtotal
A	-	0/0	-	0	0
C	A	1/1	A	1	1
B	AC	2/3	AC	2	3
C	ACB	3/6	ABC	3	6
D	ACB	3/9	CAB	3	9
A	ACBD	4/13	CABD	2	11
D	ACBD	4/17	ACBD	4	15
A	ACBD	4/21	ACDB	1	16
C	ACBD	4/25	ACDB	2	18
A	ACBD	4/29	ACDB	1	19
C	ACBD	4/33	ACDB	2	21
C	ACBD	4/37	ACDB	2	23
E	ACBD	4/41	CADB	4	27
E	ACBDE	5/46	CADBE	5	32

Comparações: 32/46 (69,57%)

Cálculo de eficiência: Mover para Frente

Elemento a ser acessado	Lista	Tamanho/ Subtotal	Algoritmo	Número de comparações	Subtotal
A	-	0/0	-	0	0
C	A	1/1	A	1	1
B	AC	2/3	AC	2	3
C	ACB	3/6	ACB	2	5
D	ACB	3/9	CAB	3	8
A	ACBD	4/13	CABD	2	10
D	ACBD	4/17	ACBD	4	14
A	ACBD	4/21	DACB	2	16
C	ACBD	4/25	ADCB	3	19
A	ACBD	4/29	CADB	2	21
C	ACBD	4/33	ACDB	2	23
C	ACBD	4/37	CADB	1	24
E	ACBD	4/41	CADB	4	28
E	ACBDE	5/46	CADBE	5	33

Comparações: 33/46 (71,74%)

Implementação

Implementação de listas auto-organizáveis

- As listas auto-organizáveis podem ser implementadas por meio da composição
- Assim, as listas auto-organizáveis contém uma lista duplamente encadeada, cuja interface fica inacessível ao usuário
- Usando as operações de inserção e remoção da lista encadeada é possível implementar as formas de organização desejadas
- Para manter um registro da frequência pode ser utilizada a classe `map` do C++
- Para computar a eficiência, basta manter a soma do tamanho da lista a cada busca, e também o número de comparações feitas

Exemplo de implementação: Mover para Frente

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  template<typename T>
6  class MoveToFront {
7      friend ostream& operator<<(ostream& os, const MoveToFront& mtf)
8      {
9          for (const auto& x : mtf.elements)
10             os << x;
11
12         os << " (" << mtf.comparisons << "/" << mtf.total << ")";
13
14         return os;
15     }
16
17 public:
18     bool search(const T& info)
19     {
20         auto it = find(elements.begin(), elements.end(), info);
21         bool found = true;
```

Exemplo de implementação: Mover para Frente

```
22
23          total += elements.size();
24
25          auto dist = distance(elements.begin(), it);
26          comparisons += (it == elements.end() ? dist : dist + 1);
27
28          if (it == elements.end()) {
29              elements.push_back(info);
30              found = false;
31          } else {
32              elements.erase(it);
33              elements.push_front(info);
34          }
35
36          return found;
37      }
38
39  private:
40      list<T> elements;
41      int comparisons = 0, total = 0;
42  };
```

Exemplo de implementação: Mover para Frente

```
43
44 int main()
45 {
46     const string message { "ACBCDADACACCEE" };
47     MoveToFront<char> mtf;
48
49     for (const auto& c : message) {
50         mtf.search(c);
51         cout << mtf << '\n';
52     }
53 }
```

Exemplo de implementação: Ordenação Estática Ótima

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
class Optimal {
    friend ostream& operator<<(ostream& os, const Optimal& opt)
    {
        for (const auto& x : opt.elements)
            os << x;

        os << " (" << opt.comparisons << "/" << opt.total << ")";

        return os;
    }

public:
    bool search(const T& info)
    {
        auto it = find(elements.begin(), elements.end(), info);
        bool found = true;
```

Exemplo de implementação: Ordenação Estática Ótima

```
22
23          total += elements.size();
24
25          auto dist = distance(elements.begin(), it);
26          comparisons += (it == elements.end() ? dist : dist + 1);
27
28          if (it == elements.end())
29          {
30              elements.push_back(info);
31              it = prev(elements.end());
32              found = false;
33          }
34
35          ++histogram[info];
36
37          while (it != elements.begin()) {
38              auto p = prev(it);
39              auto hp = histogram[*p], hit = histogram[*it];
40
41              if (hp > hit or (hp == hit and *p < *it))
42                  break;
```

Exemplo de implementação: Ordenação Estática Ótima

```
44              swap(*it, *p);
45              it = p;
46          }
47          return found;
48      }
49
50  private:
51      list<T> elements;
52      map<T, int> histogram;
53      int comparisons = 0, total = 0;
54  };
55
56  int main() {
57      const string message { "ACBCDADACACCEE" };
58      Optimal<char> auto_list;
59
60      for (const auto& c : message) {
61          auto_list.search(c);
62          cout << auto_list << '\n';
63      }
64  }
```

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference[1].

[1] https://en.cppreference.com/w/