Vetores

Fundamentos

Prof. Edson Alves - UnB/FGA

Sumário

Vetores em C/C++

Características dos vetores

- Os vetores (*arrays*) tem suporte nativo na maioria das linguagens de programação
- Em geral, a capacidade (número de elementos) do vetor tem que ser conhecida em tempo de compilação
- Os elementos do vetor são armazenados em memória de forma contígua (linear)
- O acesso aleatório é imediato: qualquer posição do vetor pode ser lida ou escrita em $O(1)$
- Porém a inserção e remoção aleatória é lenta: os elementos tem que ser transpostos no momento da inserção/remoção (complexidade $O(N)$)

Declaração estática de vetores

Sintaxe para declaração de vetores

```
tipo_do_dado nome_do_vetor[N];
```

- Na sintaxe acima, N é o número de elementos do vetor
- Os vetores armazenam elementos do mesmo tipo que foi especificado em sua declaração
- O primeiro elemento do vetor tem índice 0 (zero) e o último tem índice $N - 1$
- Um erro comum é usar tentar acessar o elemento de índice N (*off-by-one*)

Sintaxe para acesso do i-ésimo elemento

```
nome_do_vetor[i] = valor;              // Escrita
tipo_do_dado x = nome_do_vetor[i];     // Leitura
```

Exemplo de uso de vetores

```c
/* Compile com a flag -lncurses */
#include <ncurses.h>
#include <ctype.h>

int found[4096], i;
char word[] = "TESTE", uppercase[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
typedef enum { STARTING, RUNNING, WAITING, VICTORY, GAME_OVER } State;

void print(int tries, State state) {
    const char header[] = "Descubra a palavra secreta:";

    clear();
    mvprintw(1, 2, header);

    for (i = 0; word[i]; ++i) {
        addch(' ');
        found[i] ? addch(word[i] | A_BOLD) : addch('_');
    }

    mvprintw(3, 0, "Chances restantes: %d", tries);
```

Exemplo de uso de vetores

```
21      mvprintw(4, 0, "Letras disponíveis:");
22
23      for (i = 0; uppercase[i]; ++i) {
24          if (uppercase[i] != '*')
25              printw(" %c", uppercase[i]);
26      }
27
28      mvprintw(6, 0, "Digite uma das letras disponíveis: ");
29
30      if (state == VICTORY)
31          mvprintw(8, 10, "Você descobriu a palavra secreta!");
32      else if (state == GAME_OVER)
33          mvprintw(8, 10, "Game Over!");
34
35      refresh();
36  }
37
38  State update(State state, int *tries) {
39      int left = 0, hits = 0, c;
```

Exemplo de uso de vetores

```
41      if (state == STARTING)
42          return RUNNING;
43
44      c = getch();
45
46      if (c < 'a' || c > 'z' || uppercase[c - 'a'] == '*')
47          return WAITING;
48
49      uppercase[c - 'a'] = '*';
50
51      for (i = 0; word[i]; ++i) {
52          if (!found[i] && word[i] == toupper(c)) {
53              ++hits;
54              found[i] = 1;
55          }
56
57          left += (!found[i] ? 1 : 0);
58      }
```

Exemplo de uso de vetores

```
60      if (left == 0)
61          return VICTORY;
62      else if (hits == 0)
63          *tries -= 1;
64
65      if (*tries == 0)
66          return GAME_OVER;
67
68      return RUNNING;
69  }
70
71  void init() {
72      initscr();
73      noecho();
74  }
75
76  void close() {
77      getch();
78      endwin();
79  }
```

Exemplo de uso de vetores

```
81 int main()
82 {
83     int tries = 7;
84     State state = STARTING;
85
86     init();
87
88     while (state != VICTORY && state != GAME_OVER) {
89         state = update(state, &tries);
90         print(tries, state);
91     }
92
93     close();
94
95     return 0;
96 }
```

Declaração de vetores dinâmicos

- Também é possível alocar um vetor dinamicamente, postergando o conhecimento da sua capacidade para o momento da execução do programa
- Em linguagem C, um vetor pode ser alocado dinamicamente de duas maneiras:

```
tipo *nome = (tipo *) malloc(sizeof(tipo) * capacidade);
tipo *nome = (tipo *) calloc(capacidade, sizeof(tipo));
```

- Em linguagem C++, um vetor pode ser alocado através do operador **new**:

```
tipo *nome = new tipo[capacidade];
```

- A memória alocada para o vetor em C deve ser liberada através da função `free()`
- Já em C++ deve ser usado o operador **delete** `[]`

Exemplo de uso de vetores dinâmicos

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

int * novo_jogo(int N) {
    int *ns = new int[N];

    if (!ns) return nullptr;

    for (int i = 0; i < N; i++) {
        ns[i] = (rand() % 60) + 1;

        for (int j = 0; j < i; j++) {
            if (ns[j] == ns[i]) {
                i--;
                break;
            }
        }
    }

    return ns;
}
```

Exemplo de uso de vetores dinâmicos

```
22 int main() {
23     srand(time(NULL));
24     int N;
25
26     do {
27         printf("Quantos numeros serão sorteados (entre 6 e 15)? ");
28         std::cin >> N;
29     } while (N < 6 || N > 15);
30
31     auto ns = novo_jogo(N);
32
33     if (!ns) return -1;
34
35     std::cout << "Numeros sorteados: ";
36     for (int i = 0; i < N; i++)
37         printf("%d%c", ns[i], " \n"[i + 1 == N]);
38
39     delete [] ns;
40     return 0;
41 }
```

Capacidade de um vetor

- Um vetor, seja alocado estaticamente, seja alocado dinamicamente, não guarda em si informações sobre sua capacidade
- Como consequência da afirmação anterior, ao passar um vetor como parâmetro de uma função é necessário ou informar a capacidade ou definir um elemento delimitador (sentinela)
- Uma forma de evitar este problema é criar um novo tipo de dado que armazene os elementos do vetor e seu tamanho
- Importante notar que, no caso de vetores (*arrays*), o tamanho físico (capacidade, número máximo de elementos que ele comporta) coincide com o tamanho lógico (tamanho, número de elementos efetivamente preenchidos)
- Esta superposição de conceitos pode gerar problemas de interpretação (por exemplo, ler um elemento que não deveria ser considerado) e alinhamento de memória (pois as alocações precisam indicar a quantidade exata a ser usada)

Exemplo de uma estrutura para vetores

```c
#ifndef FLOAT_VECTOR_H
#define FLOAT_VECTOR_H

typedef struct _FloatVector {
    float *data;
    int size;
} FloatVector;

extern float max(const FloatVector *v);
extern float min(const FloatVector *v);

#endif
```

Exemplo de uma estrutura para vetores

```c
#include <float.h>
#include <float_vector.h>

float min(const FloatVector *v) {
    float m = FLT_MAX;

    for (int i = 0; i < v->size; ++i)
        m = v->data[i] < m ? v->data[i] : m;

    return m;
}

float max(const FloatVector *v) {
    float m = -FLT_MAX;

    for (int i = 0; i < v->size; ++i)
        m = v->data[i] > m ? v->data[i] : m;

    return m;
}
```

Exemplo de uma estrutura para vetores

```c
#include <stdlib.h>
#include <stdio.h>

#include <float_vector.h>

int main()
{
    FloatVector v;
    int i;

    printf("Informe a quantidade de valores a serem inseridos: ");
    scanf("%d", &v.size);

    if (v.size < 1) {
        fprintf(stderr, "Nada a fazer!\n");
        return -1;
    }

    v.data = (float *) malloc(sizeof(float)*v.size);
```

Exemplo de uma estrutura para vetores

```
21      if (!v.data) {
22          fprintf(stderr, "Sem memoria\n");
23          return -2;
24      }
25
26      for (i = 0; i < v.size; i++) {
27          printf("Informe o valor %d: ", i + 1);
28          scanf("%f", &v.data[i]);
29      }
30
31      printf("O maior valor informado foi %3.2f\n", max(&v));
32      printf("O menor valor informado foi %3.2f\n", min(&v));
33
34      free(v.data);
35
36      return 0;
37  }
```

Tipos de dados abstratos

Tipos de dados abstratos

- Os tipos de dados abstratos (*Abstract Data Type – ADT*) são definidos pelas operações que agem sobre estes dados, e não pela implementação destas operações
- Conhecidas as operações, é possível escolher a implementação mais eficiente para cada caso
- Em alguns casos também é preciso determinar as restrições que cada operação possui, sejam restrições ou de comportamento ou de complexidade

Operações típicas de tipos de dados abstratos

Operação	Descrição
`create()`	cria uma nova instância `S`
`initialize(S)`	prepara a instância `S` para o seu estado inicial, permitindo seu uso em operações subsequentes
`free(S)`	libera a memória utilizada pela instância `S`
`empty(S)`	retorna verdadeiro se a instância `S` está vazia
`size(S)`	lista o número de elementos contidos em `S`
`compare(S, T)`	retorna verdadeiro se as instâncias `S` e `T` tem os mesmos elementos, na mesma ordem
`print(S)`	produz uma representação visual da instância `S`
`copy(S, T)`	faz com que a instância `S` fique com o mesmo estado que a instância `T`
`clone(S)`	cria uma nova instância `T` com o mesmo estado de `S`

Contêineres

- Contêineres são tipos de dados abstratos que representam coleções de outros objetos
- Além das operações `create` e `size`, os contêineres devem fornecer operações para:
  - remover todos os elementos de uma só vez (`clear`)
  - inserir novos elementos (`push`)
  - remover elementos (`pop`)
  - acesso aos elementos armazenados (`element`)
- A classe `vector` de C++ é um contêiner
- É possível criar um contêiner semelhante em C, implementando as operações listadas anteriormente
- Para tal, é preciso definir uma estrutura que contenha as variáveis necessárias para a implementação das funções que agirão sobre esta estrutura

Exemplo de uso da classe vector

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

int main() {
    std::vector<int> ns { 1, 1, 2, 3, 5, 8, 13, 21 };

    std::cout << "ns tem " << ns.size() << " elementos\n";

    ns.clear();
    std::cout << "ns tem " << ns.size() << " elementos\n";

    for (int i = 1; i <= 10; ++i)
        ns.push_back(i);

    ns.pop_back();

    std::cout << "ns: ";

    for (size_t i = 0; i < ns.size(); ++i)
        std::cout << ns[i] << (i + 1 == ns.size() ? "\n" : " ");
```

Exemplo de uso da classe vector

```
21      std::vector<int> xs(9, 123);
22
23      std::cout << "xs tem " << xs.size() << " elementos, todos iguais a " << xs.front() << '\n';
24
25      std::iota(xs.begin(), xs.end(), 1);
26
27      std::cout << "xs:";
28
29      for (auto x : xs)
30          std::cout << ' ' << x;
31      std::cout << '\n';
32
33      std::cout << (ns == xs ? "Sim" : "Nao") << '\n';
34
35      return 0;
36  }
```

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/