AED III

Estruturas C++

Deque

Stack

Queue

Priority Queue

Deque

Um deque é um vetor dinâmica cujo tamanho pode ser eficientemente alterado nas duas extremidades da matriz. Como um vetor, um deque fornece as funções push_back e pop_back, mas também inclui as funções push_front e pop_front que não estão disponíveis em um vetor.

Um deque pode ser usado da seguinte forma:

DEQUES

As inserções e as remoções só podem ser feitas pelas extremidades.

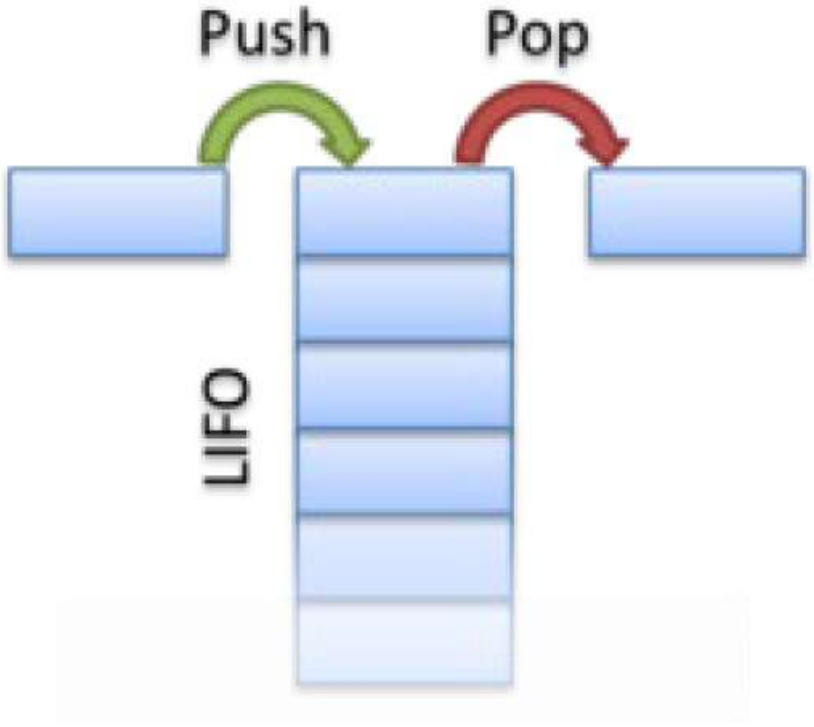
```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  deque<int> d;
  deque<int>:: iterator it;
  d.push_back(5);
  printf("d = {");
  for (it = d.begin(); it != d.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  d.push_back(2);
  printf("d = {");
  for (it = d.begin(); it != d.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  d.push_front(3);
  printf("d = {");
  for (it = d.begin(); it != d.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  d.pop_back();
  printf("d = {");
  for (it = d.begin(); it != d.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  d.pop_front();
  printf("d = {");
  for (it = d.begin(); it != d.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  return 0;
}
```

```
d = { 5}
d = { 5 2}
d = { 3 5 2}
d = { 3 5}
d = { 5}
```

Stack

Uma pilha é uma estrutura de dados que fornece duas operações de tempo de O(1): adicionando um elemento na parte superior e removendo um elemento da parte superior. Só é possível acessar o elemento superior de uma pilha.

O código a seguir mostra como uma pilha pode ser usada:


Push
Pop
LIFO

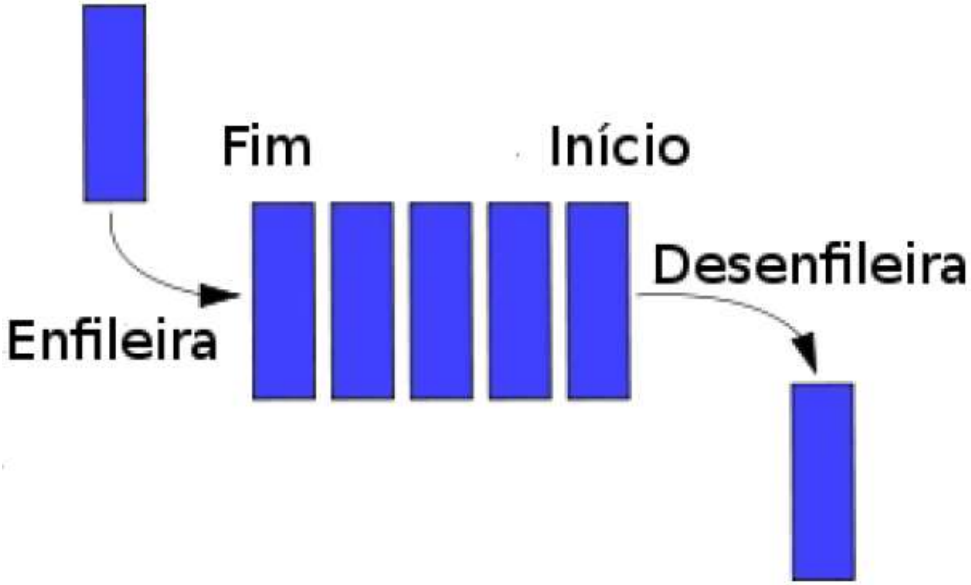
```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  stack<int> s;
  s.push(3);
  printf("Elemento do topo: %d\n", s.top());
  s.push(2);
  printf("Elemento do topo: %d\n", s.top());
  s.push(5);
  printf("Elemento do topo: %d\n", s.top());
  s.pop();
  printf("Elemento do topo: %d\n", s.top());
  return 0;
}
```

```
Elemento do topo: 3
Elemento do topo: 2
Elemento do topo: 5
Elemento do topo: 2
```

Queue

Uma fila também fornece duas operações de tempo de O(1): adicionando um elemento no final da fila e removendo o primeiro elemento na fila. Só é possível acessar o primeiro e último elemento de uma fila.

O código a seguir mostra como uma fila pode ser usada:


Fim
Início
Enfileira
Desenfileira

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  queue<int> q;
  q.push(3);
  printf("Elemento do inicio: %d\n", q.front());
  q.push(2);
  printf("Elemento do inicio: %d\n", q.front());
  q.push(5);
  printf("Elemento do inicio: %d\n", q.front());
  q.pop();
  printf("Elemento do inicio: %d\n", q.front());
  return 0;
}
```

```
Elemento do inicio: 3
Elemento do inicio: 3
Elemento do inicio: 3
Elemento do inicio: 2
```

Priority Queue

Uma fila de prioridade mantém um conjunto de elementos. As operações suportadas são inserção e, dependendo do tipo de fila, recuperação e remoção do elemento mínimo ou máximo. A inserção e remoção levam o tempo O(log n) e a recuperação leva O(1) tempo.

Enquanto um conjunto ordenado suporta de forma eficiente todas as operações de uma fila de prioridade, o benefício de usar uma fila de prioridade é que ele tem fatores menores e constantes. Uma fila de prioridade geralmente é implementada usando uma estrutura de heap que é muito mais simples do que uma árvore binária equilibrada usada em um conjunto ordenado.

Por padrão, os elementos em uma fila de prioridade C++ são classificados em ordem decrescente, e é possível encontrar e remover o maior elemento na fila. O código a seguir ilustra isso:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  priority_queue<int> q;
  q.push(3);
  printf("Elemento do inicio: %d\n", q.top());
  q.push(5);
  printf("Elemento do inicio: %d\n", q.top());
  q.push(7);
  printf("Elemento do inicio: %d\n", q.top());
  q.push(2);
  printf("Elemento do inicio: %d\n", q.top());
  q.pop();
  printf("Elemento do inicio: %d\n", q.top());
  q.pop();
  printf("Elemento do inicio: %d\n", q.top());
  q.push(6);
  printf("Elemento do inicio: %d\n", q.top());
  return 0;
}
```

```
Elemento do inicio: 3
Elemento do inicio: 5
Elemento do inicio: 7
Elemento do inicio: 7
Elemento do inicio: 5
Elemento do inicio: 3
Elemento do inicio: 6
```

Se quisermos criar uma fila de prioridade que suporte a descoberta e remoção do menor elemento, podemos fazê-lo da seguinte maneira:

```
priority_queue<int,vector<int>,greater<int>> q;
```

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   priority_queue<int, vector<int>, greater<int> > q;
   for( int i= 0; i< 10; ++i ) q.push( rand() );
   while( !q.empty() )
   {
      printf("%d\n", q.top());
      q.pop();
   }
   return 0;
}
```

Exercícios

1. Uma agência bancária estatal está sofrendo com o mau atendimento aos clientes. Suas filas são gigantescas! O gerente decidiu contratá-lo para que você crie uma simulação do atendimento da agência e permita que ele faça experimentos para melhorar o atendimento. O gerente vai informar a quantidade de minutos que ele deseja executar a simulação. Na agência existem 5 caixas. A cada minuto chegam de 4 a 16 clientes. Cada caixa atende de 1 a 2 clientes por minuto. Sempre o cliente dá preferência para alguma caixa vazia ou para uma fila com menor número de clientes. Para cada cliente que entra na fila deve-se registrar o momento (tempo) que ele entrou na fila. Quando o cliente for atendido deve-se computar a diferença entre o tempo atual e o tempo de entrada na fila. Este dado vai servir para calcular o tempo médio dos clientes na fila. O programa deve informar, por minuto, a quantidade de clientes sendo atendidos e esperando na fila para cada caixa. No término da simulação deve ser informado o tempo médio dos clientes aguardando na fila.

Exercícios

2. Manoel percebeu que seu estacionamento com uma entrada era um fiasco. Ele resolveu vender o seu terreno e comprar um novo que possui uma entrada e uma saída no fundo do terreno. Quando chega um novo carro, este é estacionado no terreno de Manoel, um atrás do outro. Quando um carro precisa sair, os carros do terreno são retirados pela saída, dão uma volta na quadra e são colocados no final da fila pela entrada do estacionamento. Faça um sistema que inclua carros no estacionamento informando o número da placa e retire carros usando o identificador (placa). Depois de ter informado a placa, cada vez que é pressionada a tecla S deve ser mostrado o estado do estacionamento.

Exercícios

3. Um hospital de cardiologia precisa de um sistema para efetuar o cadastro de pacientes que necessitam de doação de coração. Para cada paciente que é incluído no sistema deve ser informado o nome, telefone e o grau de urgência para transplante. O grau de urgência é definido na seguinte escala: (5) Muito urgente; (4) Urgente; (3) Médio; (2) Pouco urgente; (1) Sem urgência. Sempre que é o hospital recebe um novo coração o sistema é consultado para obter o próximo paciente que deverá ser operado. O sistema informa o nome e o telefone do paciente. Também a qualquer momento é possível visualizar o tamanho da fila de espera.