Hàng đợi ưu tiên

Trong bài này chúng ta sẽ cùng nhau tìm hiểu về hàng đợi ưu tiên.

Nếu bạn đã đọc qua loạt bài học về cấu trúc dữ liệu vun đống -heap, thì bạn sẽ biết hàng đợi ưu tiên là một ứng dụng của heap.

Tuy nhiên bạn có thể thực hiện hàng đợi ưu tiên mà không cần dùng heap. Nhưng dùng heap để xây dựng hàng đợi ưu tiên là cách hiệu quả, tối ưu và hay được dùng nhất.

Trong một số ngôn ngữ như C++ thì hàng đợi ưu tiên đã được cho vào trong bộ thư viện chuẩn và ta chỉ việc gọi ra để dùng,các bạn có thể tham khảo ở đây. Tuy nhiên trong rất nhiều các trường hợp khác ta phải tự tay xây dựng hàng đợi ưu tiên và cải tiến nó để giải quyết được các bài toán phức tạp hơn.

Định nghĩa hàng đợi ưu tiên

Hàng đợi ưu tiên cũng có những tính chất giống như hàng đợi đó là chèn phần tử vào phía cuối và lấy ra từ phía đầu. Nhưng có điểm khác đó là thứ tự các phần tử trong hàng đợi ưu tiên phụ thuộc vào độ ưu tiên của phần tử đó.còn hàng đợi bình thường thì tuân theo tính chất FIFO (vào trước ra trước).

Phần tử với độ ưu tiên cao nhất sẽ được xếp lên đầu hàng đợi và phần tử với độ ưu tiên thấp nhất sẽ được chuyển xuống cuối.

Do vạy, khi bạn chèn một phần tử vào cuối hàng đợi ưu tiên, no có thể được chuyển lên đầu tiên nếu độ ưu tiên của nó là cao nhất.

Ví dụ về hàng đợi ưu tiên

Giả sử ta có một mảng với 5 phần tử: {4, 8, 1, 7, 3} và bạn phải chèn các phần tử này vào một hàng đợi ưu tiên theo giá trị lớn nhất.

Bước 1: Ban đầu hàng đợi rỗng, do vậy 4 được chèn vào.

Bước 2: Chèn 8 vào hàng đợi, do 8 lớn hơn 4 lên 8 được đẩy lên đầu hàng đợi.

Bước 3: chèn 1 vào hàng đợi, 1 nhỏ hơn 4 và 8 nên 1 giữ nguyên vị trí ở cuối hàng đợi.

Bước 4: Chèn 7 vào hàng đợi, 7 lớn hơn 1 và 4, nhỏ hơn 8 nên 7 được đẩy lên vị trí nằm giữa 4 và 8.

Bước 5: Chèn 3 vào hàng đợi, 3 lớn hơn 1 và nhỏ hơn 4 lên 3 được đẩy lên vị trí nằm giữa 1 và 4.

Tất cả các bước được mô tả dưới hình sau:

Hình 1: Mô tả hoạt động của hàng đợi ưu tiên

Có nhiều cách để xây dựng hàng đợi ưu tiên.

Cách tiếp cận đơn giản

Giả sử ta có N phần tử và phải chèn chúng vào hàng đợi ưu tiên. Chúng ta có thể sử dụng danh sách liên kết để thực hiện chèn các phần tử với độ phức tạp O(N) (trong trường hợp ta chèn phần tử vào cuối danh sách liên kết)và có thể săp xếp lại chúng để duy trì các đặc tính của hàng đợi ưu tiên với độ phức tạp O(NlogN).

Cách tiếp cận tối ưu

Sử dụng heap để xây dựng hàng đợi ưu tiên với độ phức tạp là O(logN) cho việc chèn và xóa phần tử khỏi hàng đợi.

Dựa vào cấu trúc heap, hàng đợi ưu tiên cũng chia ra làm hai loại là hàng đợi ưu tiên theo giá trị lớn nhất (max -priority queue) và hàng đợi ưu tiên theo giá trị nhỏ nhất (min - priority queue).

Trong phạm vi bài học này chúng ta sẽ đi sâu vào tìm hiểu hàng đợi ưu tiên theo giá trị lớn nhất. Cách xây dựng hàng đợi ưu tiên theo giá trị nhỏ nhất cũng tương tự như vậy.

Hàng đợi ưu tiên theo giá trị lớn nhất có thể thực hiện các thao tác sau:

- Trả về phần tử lớn nhất trong hàng đợi ưu tiên.
- Xóa phần tử có giá trị lớn nhất ra khỏi hàng đợi ưu tiên và trả về giá trị của nó.
- Chèn phần tử vào hàng đợi ưu tiên.

Cài đặt hàng đợi ưu tiên - priority queue

Ta cùng xét cách viết các hàm thực hiện các thao tác trong hàng đợi ưu tiên.

Giả sử ta có A là mảng để lưu các phần tử của hàng đợi ưu tiên.

N là số phần tử hiện có trong hàng đợi.

Hàm chèn phần tử vào hàng đợi ưu tiên

Khi chèn phần tử vào hàng đợi, nó có thể vi phạm các quy tắc của max heap, nếu vậy ta phải thực hiện đổi chỗ giá trị node cha và của node đó cho tới khi ta có được giá trị của node cha là lớn hơn.

```
void insert_element(int A[ ], int val)
{
    N = N + 1; /* Tăng kích thước của hàng đợi lên 1
                  khi ta chèn thêm phần tử vào */
    int i = N;    /* N là biến toàn cục, không nên thay đổi giá trị của nó,
                  ta sử dụng biến tạm i ở đây, để có thể tùy ý xử lý */
    A[ i ] = val; /* Trước tiên phần tử được chèn vào vị trí cuối cùng của hàng đợi */
    while( i > 1 and A[ i/2 ] < A[ i ]) /* Nếu giá trị node cha nhỏ hơn giá trị của nó */
    {
        swap(A[ i/2 ], A[ i ]); /* Đổi chỗ hai node */
        i = i/2;                /* Tiếp tục kiểm tra tại vị trí của node cha */
    }
}
```

Độ phức tạp về thời gian: O(logN)

Hàm trả về giá trị lớn nhất trong hàng đợi:

```
int max_element(int A[ ])
{
    return A[1];  /* A[1] là node gốc trong max heap
                  và node gốc trong max heap là node có giá trị lớn nhất */
}
```

Độ phức tạp thời gian: O(1)

Hàm xóa phần tử lớn nhất ra khỏi hàng đợi ưu tiên và trả về giá trị của nó:

```
int pop_max_element (int A[ ])
{
    if(N == 0)
       {
          cout<< "Khong the xoa phan tu, vi hang doi rong";
          return -1;
       }
    int max = A[1]; /* Lưu lại giá trị của phần tử lớn nhất trong hàng đợi */
    A[1] = A[N]; /* Phần tử cuối cùng sẽ được đặt vào vị trí của node gốc,
                    Điều này làm cho node gốc vi phạm các dặc tính của
                    max heap, và hàm max_heap() sẽ chạy lại từ vị trí này
                    để duy trì lại trật tự mới sau khi node có giá trị lớn nhất
                    bị xóa ra khỏi hàng đợi */
    N = N -1;
    max_heap(A, 1); /* Chạy hàm max heap tại node gốc mới */
    return max;
}
```

Độ phức tạp thời gian: O(logN).

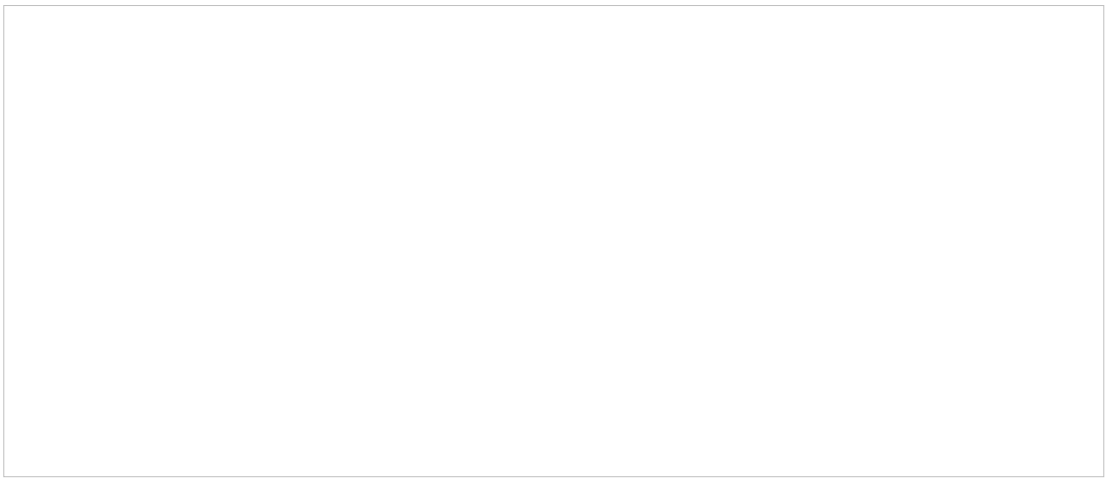
Ví dụ về các thao tác với hàng đợi ưu tiên

Gỉa sử ban đầu ta có 5 phần tử trong hàng đợi ưu tiên như sau: {8,7,4,3,1}

Ta xét chuỗi các thao tác sau:

Chèn phần tử mới vào hàng đợi ưu tiên

insert_element(A, 6). Khi chèn 6 vào hàng đợi trên, các quy tắc của hàng đợi ưu tiên bị vi phạm (vi phạm quy tắc của max heap). Do vậy ta phải thực hiện đổi chỗ với node chacủa nó, node có giá trị 4. Sau khi đổi chỗ, các quy tắc của max heap được duy trì. Cùng xem hình mô tả dưới đây.

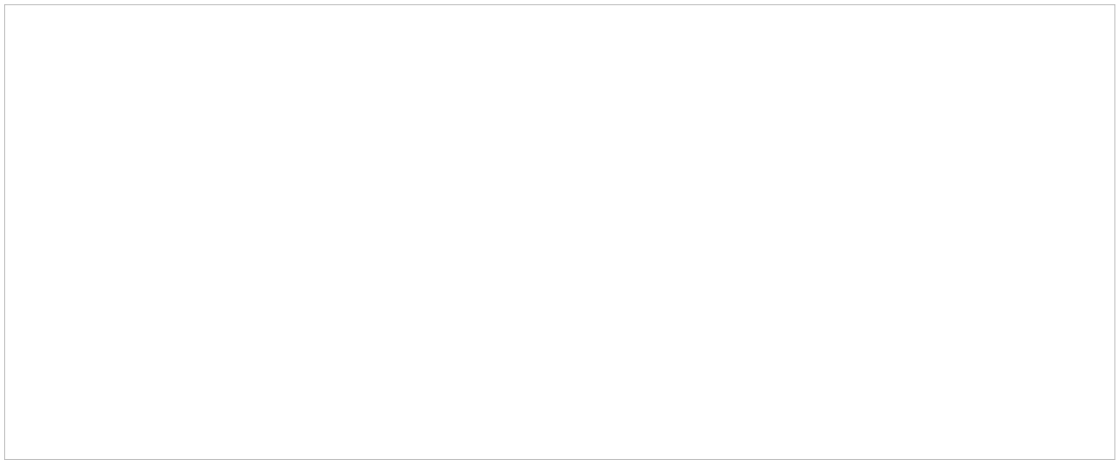


Hình 2: Chèn phần tử vào hàng đợi ưu tiên

Xóa phần tử lớn nhất khỏi hàng đợi và trả về giá trị của nó

pop_max_element() sẽ lấy phần tử có giá trị lớn nhất ra khỏi hàng đợi (node gốc của max heap).

Như ở hình dưới đây phần tử có giá trị 8 sẽ được lấy ra, và phần tử cuối cùng trong hàng đợi có giá trị 4 sẽ được chuyển lên thế chỗ của phần tử vừa được lấy ra. Lúc này các quy tác của max heap bị phá vỡ nên ta cần thực hiện hàm max_heap() để duy trì lại các quy tắc này.



Hình 3: Xóa phần tử lớn nhất khỏi hàng đợi ưu tiên theo giá trị lớn nhất

Mã nguồn đầy đủ của chương trình cài đặt hàng đợi ưu tiên

```
#include<iostream>
using namespace std;

int N;

void insert_element(int A[ ], int val)
{
   N = N + 1; /* Tăng kích thước của hàng đợi lên 1
                 khi ta chèn thêm phần tử vào */
   int i = N;    /* N là biến toàn cục, không nên thay đổi giá trị của nó,
                 ta sử dụng biến tạm i ở đây, để có thể tùy ý xử lý */
   A[ i ] = val; /* Trước tiên phần tử được chèn vào vị trí cuối cùng của hàng đợi */
   while( i > 1 and A[ i/2 ] < A[ i ]) /* Nếu giá trị node cha nhỏ hơn giá trị của nó */
   {
      swap(A[ i/2 ], A[ i ]); /* Đổi chỗ hai node */
      i = i/2;                /* Tiếp tục kiểm tra tại vị trí của node cha */
   }
}

void max_heap (int A[ ], int i)
{
      int largest;
      int left = 2*i;              /* Vị trí của con bên trái */
      int right = 2*i +1 ;         /* Vị trí của con bên phải */
      if(left<= N and A[left] > A[i] ) /* N là số phần tử trong mảng, biến toàn cục */
          largest = left;
      else
          largest = i;
      if(right <= N and A[right] > A[largest] )
         largest = right;
      if(largest != i )
      {
         swap (A[i] , A[largest]);
         max_heap (A, largest);
      }
}


int max_element(int A[ ])
{
   return A[1];  /* A[1] là node gốc trong max heap
                 và node gốc trong max heap là node có giá trị lớn nhất */
}

int pop_max_element (int A[ ])
{
   if(N == 0)
      {
         cout << "\tKhong the xoa phan tu, vi hang doi rong\n";
         return -1;
      }
   int max = A[1]; /* Lưu lại giá trị của phần tử lớn nhất trong hàng đợi */
   A[1] = A[N]; /* Phần tử cuối cùng sẽ được đặt vào vị trí của node gốc,
                   Điều này làm cho node gốc vi phạm các dặc tính của
                   max heap, và hàm max_heap() sẽ chạy lại từ vị trí này
                   để duy trì lại trật tự mới sau khi node có giá trị lớn nhất
                   bị xóa ra khỏi hàng đợi */
   N = N -1;
   max_heap(A, 1); /* Chạy hàm max heap tại node gốc mới */
   return max;
}
```

```
void print_element(int A[]){
    if(N <= 0) return;
    cout << "\t";
    for(int i = 1; i <= N; i++){
        cout << A[i] << " ";
    }
    cout << endl;
}
int main(){
    N = 0;
    int A[10];
    insert_element(A,4);
    insert_element(A,8);
    insert_element(A,1);
    insert_element(A,7);
    insert_element(A,3);
    cout << endl;
    insert_element(A,6); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);
    pop_max_element(A); print_element(A);

    return 0;
}
```

Kết quả chương trình sau khi chạy và kiểm tra tại https://www.onlinegdb.com.

Source code trên gitlab: https://gitlab.com/thevngeek/basic-data-structure/blob/master/hangdoiuutien.cpp

Ta có thể thấy với cách thực hiện như trên các phần tử của hàng đợi ưu tiên tuy không luôn luôn sắp xếp theo đúng thứ tự lớn trước, bé sau. Nhưng mỗi lần lấy phần tử ra khỏi hàng đợi ta luôn luôn được đảm bảo rằng phần tử đó là lớn nhất, như vậy quy tắc phần tử có độ ưu tiên cao hơn sẽ được ra khỏi hàng đợi trước (được phục vụ trước) vẫn được giữ đúng. Và ưu điểm của cách làm này cho ta lợi về tốc độ chạy chương trình.

Như vậy ta có thể ứng dụng hàng đợi ưu tiên để lập lịch công việc. Khi ta có một hàng đợi với N công việc, mỗi công việc đều có độ ưu tiên của riêng nó. Nếu công việc có độ ưu tiên cao nhất sẽ được hoàn thành trước và xóa bỏ khỏi hàng đợi, ta có thể dùng hàm pop_max_element() để thực hiện việc này. Và nếu tại mỗi thời điểm ta phải thêm một công việc mới vào hàng đợi, ta có thể dùng hàm insert_element() để thêm phần tử với độ phức tạp về thời gian là O(logN) và duy trì các quy tắc của max heap (node cha luôn luôn có giá trị lớn hơn các node con).

Tham khảo

1.https://vi.wikipedia.org/wiki/%C4%90%E1%BB%91ng_nh%E1%BB%8B_ph%C3%A2n

2.https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/tutorial/