TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**BÀI TẬP LỚN MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Giảng viên hướng dẫn: Nguyễn Việt Hưng

Sinh viên thực hiện: Mai Văn Hiếu - Lớp CNTT4 – K61

**Hà Nội, tháng 12 năm 2021**

**Hà Nội, tháng 12 năm 2021**

|  |  |
| --- | --- |
| **Sinh Viên: Mai Văn Hiếu** | **Mã sinh viên: 201200123** |
| **Môn: Cấu trúc dữ liệu và giải thuật** | **Lớp: CNTT4 – K61** |

**Báo cáo bài tập lớn**

**I**, Đề bài:

1. Cho cây nhị phân trừu tượng chứa các phần tử so sánh được với nhau thỏa mãn các tính chất sau:  
   a. Phần tử gốc lớn hơn hoặc bằng mọi phần tử còn lại của cây  
   b. 2 cây con trái và phải cũng thỏa mãn tính chất trên
2. 2. Hãy cài đặt cấu trúc dữ liệu cây nhị phân trên có các thao tác push(x) thêm phần tử vào cây và pop() lấy phần tử gốc khỏi cây, top() lấy giá trị của nút gốc sao cho luôn luôn đảm bảo tính chất cây heap.  
   Áp dụng cấu trúcA cây này như một hàng đợi ưu tiên thực hiện các việc sau  
   - Nhập vào 1 dãy số xuất ra dãy giảm dần  
   - Giải quyết bài toán nối thanh kim loại http://laptrinhonline.club/problem/tichpxnoidam  
   Công việc cơ khí thật là mệt nhọc, muốn nối một thanh kim loại độ dài a với một thanh kim loại độ dài b thì kinh phí để thuê nối tốn mất a+b đơn vị tiền tệ. Hiện nay Tèo cần nối n thanh kim loại lần lượng có độ dài là a1, a2, ... an thành một đoạn theo bạn Tèo nên bố trí thế nào để tổng số tiền phải trả là ít nhất  
   Input:  
   Dòng đầu chứa số nguyên dương n (1<=n<=105)  
   Dòng tiếp theo là n số nguyên dương tương ứng là độ dài các thanh muốn nối  
   (1<=ai<=103)  
   Output:  
   Một số nguyên dương là số kinh phí ít nhất phải trả  
   Ví dụ  
   Input  
   3  
   8 4 6  
   Output  
   28  
   Giải thích: Nếu ta nối thanh 8 với thanh 4 tốn chi phí là 8+4=12 sau khi nối xong còn 2 thanh độ dài 12 và 6 nối lại với nhau tốn 12+6=18 tổng chi phí nối là 12+18=30. Nếu ta nối 4 với 6 trước tốn 10 và còn 2 thanh 10 và 8 nối lại với nhau tốn 18 do đó tổng kinh phí ít hơn chỉ còn 28  
   Gợi ý: Bằng cách nhập vào dãy a là độ dài thanh mỗi bước lấy ra 2 thanh ngắn nhất nối với nhau rồi lại thêm thanh nối được vào.

**II**, Phân tích bài toán:

1. **Yêu cầu của bài toán.**

**1.1,** Xây dựng cây nhị phân có các phần tử so sánh được với nhau, thoả mãn điều kiện, các phần tử cha luôn lớn hơn hoặc bằng các phần tử con của nó.

**1.2,**

**a,** Cài đặt cấu trúc cây nhị phân trên có các thao tác:

* push(x): thêm một phần tử vào cây.
* top(): lấy giá trị của phần tử gốc.
* pop(): lấy phần tử gốc ra khỏi cây.
* Các thao tác trên phải luôn đảo bảo tính chất của cây heap.

**b,** Áp dụng cấu trúc cây đã xây dựng như một hàng đợi ưu tiên(priority\_queue) để giải quyết vấn đề sau:

* Nhập vào một dãy số và xuất ra dãy giảm dần.
* Bài toán nối thanh kim loại.

**2, Xác định lớp.**

* Xây dựng lớp Heap(Binary Heap) có các chức năng thỏa mãn yêu cầu bài toán.

**3, Mô tả các chức năng của lớp.**

* Sử dụng template class: khai báo khuôn mẫu lớp, cho phép định nghĩa lớp cho nhiều kiểu dữ liệu.
  + template<typename T>: T là kiểu dữ liệu trừu tượng.
* Xây dựng một hàm swap(hoán vị) để hoán đổi giá trị.

**void swap(T \*x, T \*y)**{

T temp = \*x;

\*x = \*y;

\*y = temp;

}

**a,** Lớp **Heap** bao gồm:

* Các thuộc tính:
  + T \*node: con trỏ trỏ tới mảng các phần tử trong cây.
  + unsigned int capacity: kích thước tối đa của cây.
  + unsigned int num: số phần tử hiện tại có trong cây.
  + bool reverser: xác định cây là Max heap hay Min heap.
* Các phương thức:
  + Heap()
  + Heap(int capacity, bool reverse)
  + ~Heap()
  + int parent(int i)
  + int left(int i)
  + int right(int i)
  + void heapify(int i)
  + bool empty()
  + bool full()
  + unsigned int size()
  + void push(int value)
  + T top()
  + void pop()

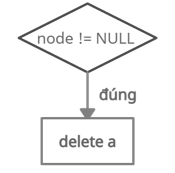
**b,** Chức năng, biểu diễn, đánh giá thời gian của các phương thức.

**Heap()**: hàm tạo không đối có chức năng khởi gán giá trị cho các thuộc tính, xác định tính chất (Max heap hoặc Min heap) của class Heap.

* + node = NULL;
  + num = capacity = 0;
  + reverse = false;
* Thời gian chạy: O(1).

**Heap(int capacity, bool reverse = false)**: hàm tạo 2 đối khởi gán giá trị cho các thuộc tính, cấp phát bộ nhớ động lưu trữ các phần tử của Heap.

* + num = 0;
  + this->capacity = capacity;
  + node = new T[capacity];
  + this->reverse = reverse;
* Thời gian chạy: O(1).

**~Heap()**: hàm hủy để giải phóng bộ nhớ.

* + if(node) delete [] node;
* Thời gian chạy là O(1).

**void parent(int i)**: lấy vị trí của nút cha tại vị trí i.

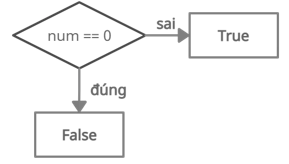
* + return (i – 1) / 2;
* Thời gian chạy là O(1).

**void left(int i)**: lấy vị trí của nút con trái tại vị trí i.

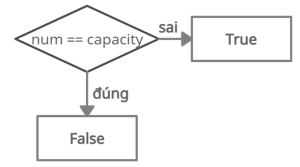
* + return 2 \* i + 1;
* Thời gian chạy là O(1).

**void right(int i)**: lấy vị trí của nút con phải tại vị trí i.

* + return 2 \* i + 2;
* Thời gian chạy là O(1).

**bool empty()**: kiểm tra cây có phải là cây rỗng.

* + return num == 0;
* Thời gian chạy là O(1).

**bool full()**: kiểm tra cây có phải cây đầy.

* + return num == capacity;
* Thời gian chạy là O(1).

**Unsigned int size()**: trả về số phần tử hiện tại trong cây.

* + return num;
* Thời gian chạy là O(1).

**Void heapify(int i)**: biến đối heap để duy trì tích chất heap.

int l = left(i);

int r = right(i);

int root = i;

if(reverse){

if(l < num && node[l] < node[i])

root = l;

if(r < num && node[r] < node[root])

root = r;

}

else {

if(l < num && node[l] > node[i])

root = l;

if(r < num && node[r] > node[root])

root = r;

}

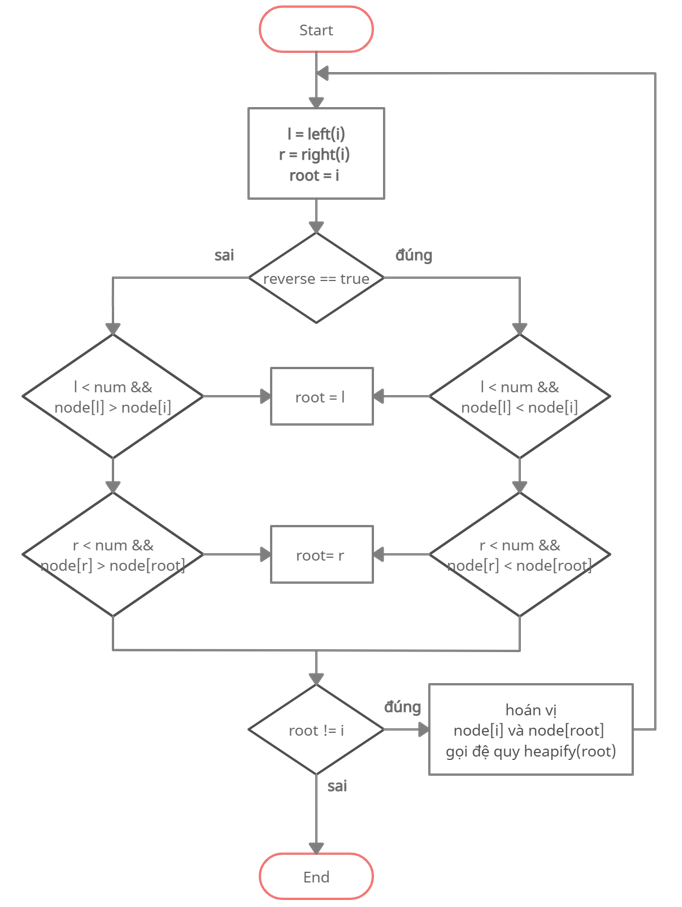
if(root != i){

swap(&node[i], &node[root]);

heapify(root);

}

* Biểu diễn bằng lưu đồ thuật toán:



* Thời gian chạy là: O(log(n)).

**void push(T value)**: thêm một phần tử vào cây.

if(full()){

capacity = capacity \* 2 + 1;

        T \*temp = new T[capacity];

        for (int i = 0; i < num; i++)

            temp[i] = node[i];

        if (node)

            delete[] node;

        node = temp;

}

int i = num;

    node[num] = value;

    num++;

    if (reverse){

        while (i != 0 && node[parent(i)] > node[i]){

            swap(&node[i], &node[parent(i)]);

            i = parent(i);

        }

    }

    else{

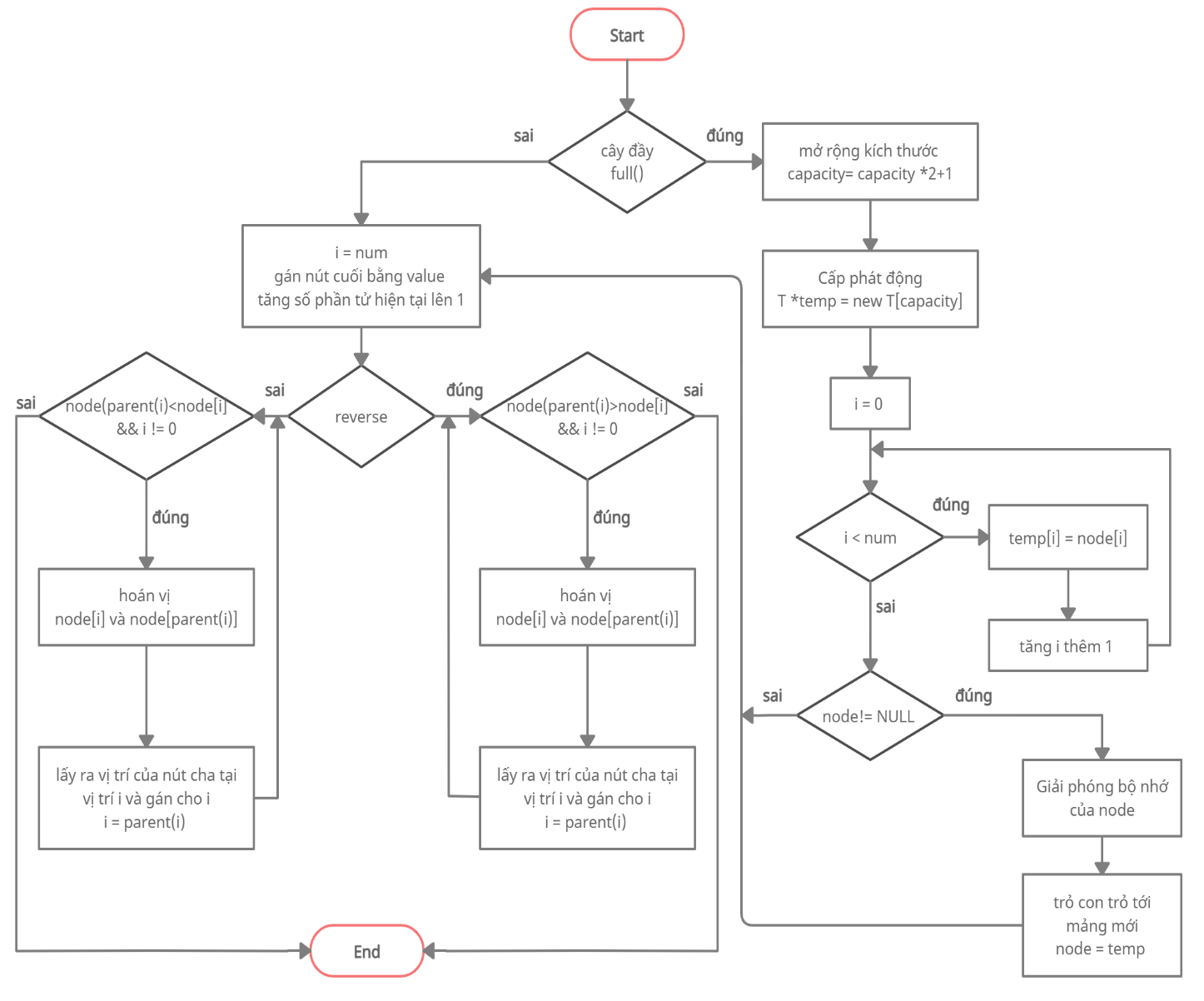
        while (i != 0 && node[parent(i)] < node[i]){

            swap(&node[i], &node[parent(i)]);

            i = parent(i);

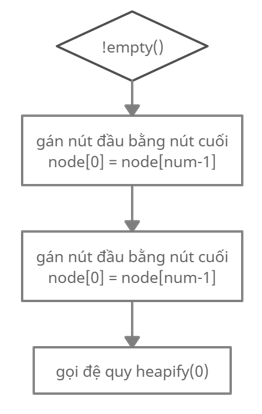
        }

    }

* Biểu diễn bằng lưu đồ thuật toán:
* Thời gian chạy là: O(nlog(n)).

**T top()**: trả về giá trị của phần tử gốc.

* + return a[0];
* Thời gian chạy là O(1).

**void pop()**: lấy phần tử đầu ra khỏi cây.

* + if(!empty()){

node[0] = node[num – 1];

num--;

heapify(0);

}

* Thời gian chạy là O(log(n)).

**III**, Cài đặt các lớp và hàm MAIN bằng C++:

Code:

// Mai Văn Hiếu - 201200123 - CNTT4 - K61

#include <iostream>

using namespace std;

template <typename T> // khai báo khuôn mẫu hàm

void swap(T \*x, T \*y);

template <typename T> // khai báo khuôn mẫu lớp

class Heap

{

private:

    T \*node; // con trỏ trỏ tới mảng các phần tử trong cây

    unsigned int capacity, num;

    bool reverse; //biến xác định Max heap - Min heap

public:

    Heap() // Hàm tạo không đối - khởi gán

    {

        node = NULL;

        num = capacity = 0;

        reverse = false; //  khởi gán false <=> Max heap

    }

    Heap(int capacity, bool reverse = false) // Hàm tạo hai đối-khởi gán

    {                                        // reverse mặc định là false

        num = 0;

        this->capacity = capacity;

        node = new T[capacity];

        this->reverse = reverse;

    }

    ~Heap() // Hàm hủy - xóa vùng nhớ

    {

        if (node)

            delete[] node;

    }

    int parent(int i) { return (i - 1) / 2; } // lấy vị trí nút cha tại vị trí i

    int left(int i) { return (2 \* i + 1); }   // lấy vị trí nút con trái tại vị trí i

    int right(int i) { return (2 \* i + 2); }  // lấy vị trí nút con phải tại vị trí i

    bool empty() { return num == 0; }         // kiểm tra xem cây có phải cây rỗng

    bool full() { return num == capacity; }   // kiểm tra xem cây đã đầy hay chưa

    unsigned int size() { return num; }       //trả về số phần tử hiện tại có trong cây

    void heapify(int i);                      // vun đống

    void push(T value);                       // thêm một phần tử vào cây

    T top() { return node[0]; }               // trả về giá trị phần tử gốc của cây

    void pop()                                // lấy phần tử gốc khỏi cây

    {

        if (!empty()) // nếu cây có chứa nút

        {

            node[0] = node[num - 1];

            num--;

            heapify(0);

        }

    }

};

template <typename T>

void Heap<T>::heapify(int i)

{

    int l = left(i);

    int r = right(i);

    int root = i;

    if (reverse) // nếu đúng thì là Min heap

    {

        if (l < num && node[l] < node[i])

            root = l;

        if (r < num && node[r] < node[root])

            root = r;

    }

    else // Max heap

    {

        if (l < num && node[l] > node[i])

            root = l;

        if (r < num && node[r] > node[root])

            root = r;

    }

    if (root != i)

    {

        swap(&node[i], &node[root]); // hoán vị

        heapify(root);

    }

}

template <typename T>

void Heap<T>::push(T value){

    if (full()){

        capacity = capacity \* 2 + 1;// mở rộng kích thước tối đa của cây

        T \*temp = new T[capacity];

        for (int i = 0; i < num; i++)   // sao chép các phần tử của cây

            temp[i] = node[i];

        if (node)

            delete[] node;  // xóa vùng nhớ của node

        node = temp;

    }

    int i = num;

    node[num] = value;

    num++;

    if (reverse){

        while (i != 0 && node[parent(i)] > node[i])

        {

            swap(&node[i], &node[parent(i)]);

            i = parent(i);

        }

    }

    else{

        while (i != 0 && node[parent(i)] < node[i])

        {

            swap(&node[i], &node[parent(i)]);

            i = parent(i);

        }

    }

}

template <typename T>

void swap(T \*x, T \*y){ // Hàm hoán vị

    T temp = \*x;

    \*x = \*y;

    \*y = temp;

}

/\* MAIN \*/

int main(){

    /\* Nhập vào một dãy xuất ra dãy giảm dần \*/

    int n, a, b, x, res = 0;

    cout << "Nhap so phan tu: ";

    cin >> n;

    Heap<int> Q1(n); // khai báo Max heap có n phần tử

    cout << "Nhap day:\n";

    while (n--)

    {

        cin >> x;

        Q1.push(x);

    }

    cout << "Day giam dan: ";

    while (!Q1.empty())

    {

        n = Q1.top();

        cout << n << " ";

        Q1.pop();

    }

    /\* Giải quyết bài toán nối thanh kim loại \*/

    cout << "\nNhap so thanh kim loai: ";

    cin >> n;

    Heap<int> Q2(n, true); // Khai báo Min heap

    cout << "Nhap cac do dai: \n";

    while (n--){

        cin >> x;

        Q2.push(x);

    }

    while (Q2.size() > 1){

        a = Q2.top();

        Q2.pop();

        b = Q2.top();

        Q2.pop();

        res += a + b;

        Q2.push(a + b);

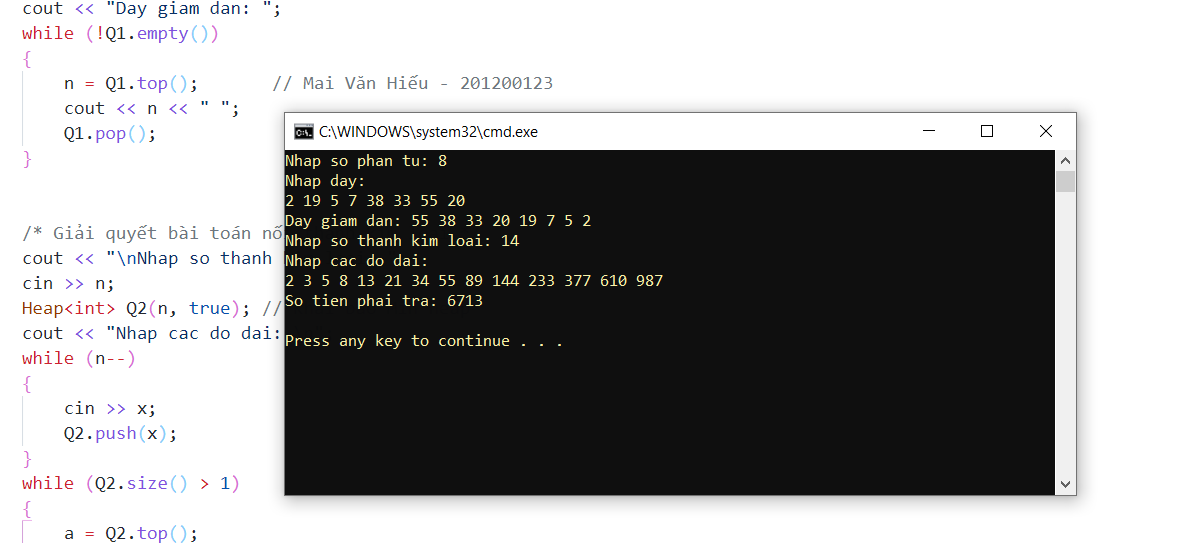
    }

    cout << "So tien phai tra: " << res << endl;

    return 0;

}

Test:



**IV,** Danh sách tài liệu tham khảo:

Các trang web:

* <https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/>
* <https://en.cppreference.com/w/cpp/container/priority_queue>
* <https://www.cplusplus.com/reference/queue/priority_queue/>
* <https://stackoverflow.com/>
* <https://algorithmtutor.com/Data-Structures/Tree/Binary-Heaps/>
* <https://vi.wikipedia.org/wiki/%C4%90%E1%BB%91ng_nh%E1%BB%8B_ph%C3%A2n>

**Hết**