51800263 - Từ Huy Vạn

Câu 1:

Chuỗi đầu vào:



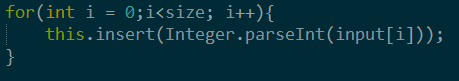
Gọi lệnh tạo cây:



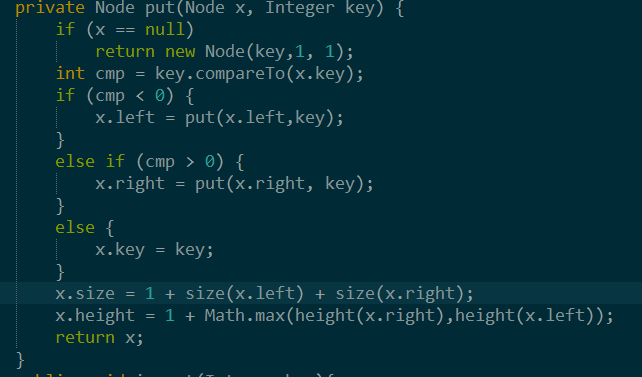
Tách chuỗi đầu vào thành mảng :



Lần lượt đưa các phần thử trong mảng để tạo cây:



So sánh giá trị của các phần tử được đưa vào để tạo thành cây:



Chuỗi đầu vào a = 8 9 5 20 1 45 88

Phần tử đưa vào: 8

8 trở thành root

tree:

8

Phần tử đưa vào: 10

10 > 8

=> gọi đệ quy để đặt 10 vào phía bên phải 8

bên phải 8 rỗng => 10 trở thành nút con bên phải của 8

tree:

8

10

Phần tử đưa vào: 5

5 < 8

=> gọi đệ quy để đặt 5 vào phía bên trái 8

bên trái 8 rỗng => 5 trở thành nút con bên trái của 8

tree:

8

5

10

Phần tử đưa vào: 20

20 > 8

=> gọi đệ quy để đặt 20 vào phía bên phải 8

20 > 9

=> gọi đệ quy để đặt 20 vào phía bên phải 9

bên phải 9 rỗng => 20 trở thành nút con bên phải của 9

tree:

5

20

10

8

hần tử đưa vào: 1

1 < 8

=> gọi đệ quy để đặt 1 vào phía bên trái 8

1 < 5

=> gọi đệ quy để đặt 1 vào phía bên trái 5

bên trái rỗng => 1 trở thành nút con bên trái của 5

tree:

8

5

10

1

20

Phần tử đưa vào: 6

6 < 8

=> gọi đệ quy để đặt 6 vào phía bên trái 8

6 > 5

=> gọi đệ quy để đặt 6 vào phía bên phải 5

bên phải rỗng => 6 trở thành nút con bên phải của 5

tree:

8

5

8

5

10

6

20

1

Phần tử đưa vào: 9

9 > 8

=> gọi đệ quy để đặt 9 vào phía bên phải 8

9 < 10

=> gọi đệ quy để đặt 9 vào phía bên trái 10

bên phải rỗng => 9 trở thành nút con bên trái của 10

tree:

8

5

10

20

9

6

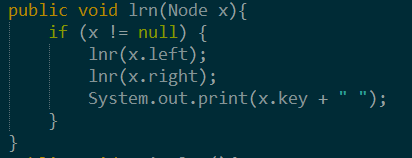
1

2:

Gọi hàm duyệt cây Left Right Node:



Sử dụng đệ quy để duyệt cây theo thứ tự Left Right Node:



\_ 8.left = 5 -> 5.left = 1 -> in 1

-> 5.right = 6 -> in 6

-> in 5

\_ 8.right = 10 -> 10.left -> 9 -> in 9

-> 10.left -> 20 -> in 20

-> in 10

\_in 8

\_Đầu ra: 1 6 5 9 20 10 8

3:

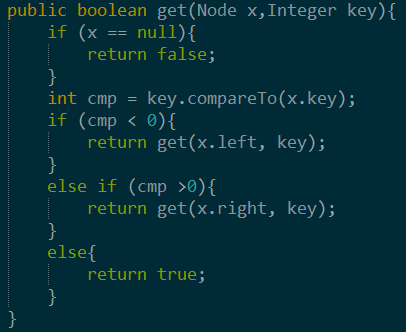
Lấy đầu vào từ người dùng:



Sử dụng đệ quy để tìm kiếm trong:

Nếu tìm thấy trả về True;

Nếu không tìm thấy trả về False;



Giả sử đầu vào là 10

10 > 8

=> Gọi đệ quy đi về phía bên phải của 8 (8.right = 10)

10 = 10

=> trả về true

\_ Gọi hàm in ở vị trí bên phải của node 8(8.right = 10):

\_ 10.left = 9 -> 9.left = null ;

-> in 9

-> 9.right = null;

\_ in 10

\_ 10.right = 20 -> 20.left -> null;

-> in 20

-> 02.right -> null;

Đầu ra: 9 10 20.

Giả sử đầu vào là 3

3 < 8

=> Gọi đệ quy đi về phía bên trái của 8 (8.left = 5)

3 < 5

=> Gọi đệ quy đi về phía bên trái của 5 (5.left = 1)

3 > 1

=> Gọi đệ quy đi về phía bên phải của 1 (8.right = null)

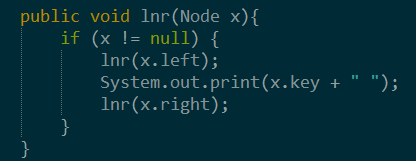
=> không tìm thấy trả về false

4:

Gọi hàm duyệt cây Left Node Right:



Hàm đệ quy duyệt cây Left Node Right:



\_ 8.left = 5 -> 5.left = 1 -> in 1

-> in 5

-> 5.right = 6 -> in 6

\_in 8

\_ 8.right = 10 -> 10.left -> 9 -> in 9

-> in 10

-> 10.left -> 20 -> in 20

Đầu ra: 5 1 6 8 9 10 20

5:

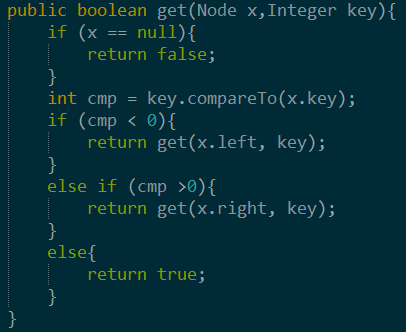
Lấy đầu vào từ người dùng:



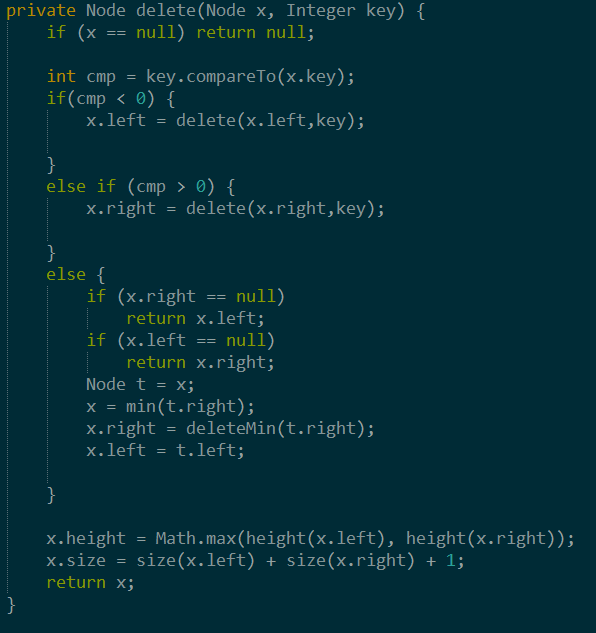
Sử dụng đệ quy để tìm kiếm trong cây:

Nếu tìm thấy trả về True=> gọi hàm xóa node đó;

Nếu không tìm thấy trả về False;



Hàm xóa node trong cây:



Giả sử đầu vào là 10

10 > 8

=> Gọi đệ quy đi về phía bên phải của 8 (8.right = 10)

10 = 10

=> trả về true

=>đưa 10 vào trong hàm xóa node

Đầu vào hàm xóa: 10

Gọi đệ quy tìm 10 trong cây để xóa

10 > 8

=> Gọi đệ quy đi về phía bên phải của 8 để xóa node (8.right = 10)

10 == 10

=> xóa node 10 khỏi cây

=> vì node 10 có 2 con:

\_ Lưu giá trị biến x cần xóa vào t = 10

\_Gán biến x bằng successor của t (min(t.right) = 20)

\_Gán vào bên phải x tất cả các node bên phải của t (t.right)

\_Gán vào bên trái x tất cả các node bên trái của t (t.left)

8

5

20

9

6

1

Giả sử đầu vào là 3

3 < 8

=> Gọi đệ quy đi về phía bên trái của 8 (8.left = 5)

3 < 5

=> Gọi đệ quy đi về phía bên trái của 5 (5.left = 1)

3 > 1

=> Gọi đệ quy đi về phía bên phải của 1 (8.right = null)

=> không tìm thấy trả về false

=> không gọi hàm xóa

Câu 2:

Chuỗi đầu vào:



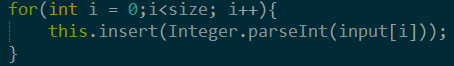
Hàm tạo cây từ chuỗi đầu vào:



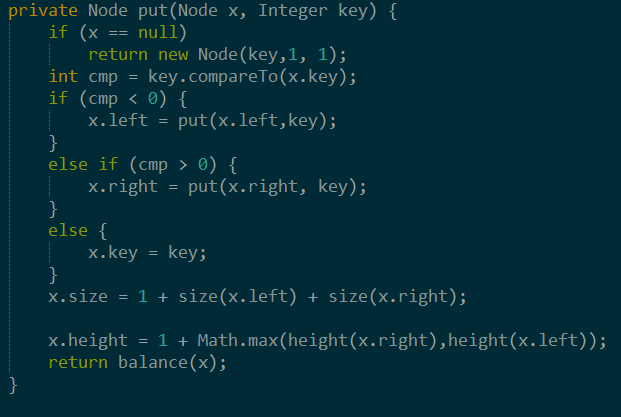
Tách chuỗi số nguyên cách nhau bằng khoảng trắng



Đưa các số nguyên vào hàm dựng cây:

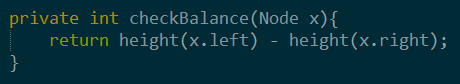


Hàm dựng cây AVL:



Hàm kiểm tra độ cân bằng của cây:

\_ Trả về kết quả là hiệu của chiều cao bên trái và bên phải của node đang xét.



Các hàm cân bằng cây:

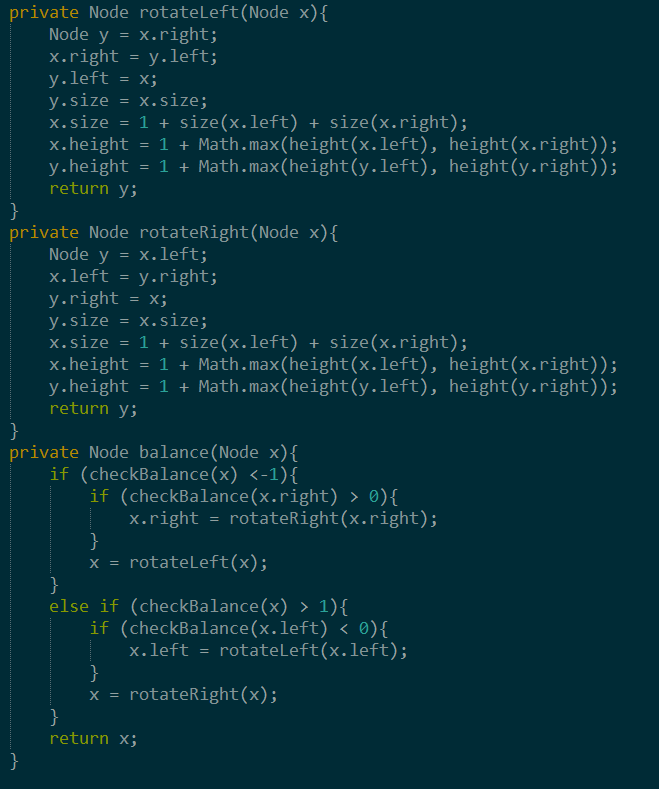
\_ Dùng hàm kiểm tra cân bằng để biết cây bị lệch theo trường hợp nào

+ Nếu checkBalance(x) < -1 => cây bị mất cân bằng Right Right => gọi hàm rotateLeft

+ Nếu checkBalance(x) < -1 và Nếu checkBalance(x) > 0=> cây bị mất cân bằng Right Left => gọi hàm rotateRight để cây mất cân bằng Right Right => gọi hàm rotateLeft

+ Nếu checkBalance(x) > 1 => cây bị mất cân bằng Left Left => gọi hàm rotateRight

+ Nếu checkBalance(x)> 1 và Nếu checkBalance(x) < 0 Nếu checkBalance(x) > 0=> cây bị mất cân bằng Left Right => gọi hàm rotateLeft để cây mất cân bằng Left Left => gọi hàm rotateRight



Chuỗi đầu vào: 5 3 7 44 12 87.

Phần tử đưa vào 5:

5

\_ checkBalance(5) = 0 => Cây cân bằng.

Phần tử đưa vào 3:

\_ 3 < 5

=> gọi đệ quy gán 3 vào bên trái 5.

5

3

\_ checkBalance(5) = 1 => Cây cân bằng.

Phần tử đưa vào 7

\_ 7 > 5

=> gọi đệ quy gán 7 vào bên phải 5.

5

3

7

\_\_ checkBalance(5) = 0 => Cây cân bằng.

Phần tử đưa vào 44

\_ 44 > 5

=> gọi đệ quy gán 44 vào bên phải 5.

\_ 44 > 7

=> gọi đệ quy gán 44 vào bên phải 7.

5

3

7

44

\_ checkBalance(5) = -1 => Cây cân bằng.

Phần tử đưa vào 12

\_ 12 > 5

=> gọi đệ quy gán 12 vào bên phải 5.

\_12 > 7

=> gọi đệ quy gán 12 vào bên phải 7.

\_12 < 44

=> gọi đệ quy gán 12 vào bên trái 44.

5

3

7

44

12

\_ checkBalance(7) = -2 => Cây mất cân bằng tại 7.

\_checkBalance(7.right = 44) = 1

=> cây mất cân bằng Right Left tại 7:

\_Cân bằng;

\_ gọi hàm rotateRight tại node 7.right (x = 44).

+ gán biến y = x.left ( y = 12).

+ gán x.left = y.right ( 44.left = null).

+ gán y.right = x ( 12.right = 44)

+ trả về y

=> 7.right = 12

5

3

7

12

44

\_ gọi hàm rotateLeft tại node 7( x = 7) .

+ gán biến y = x.right ( y = 12).

+ gán x.right = y.left ( 7.right = null).

+ gán y.left = x ( 12.left = 7)

+ trả về y

=> x = 12

5

12

3

7

44

Phần tử đưa vào 87

\_ 87 > 5

=> gọi đệ quy gán 87 vào bên phải 5.

\_87 > 12

=> gọi đệ quy gán 87 vào bên phải 12.

\_87 > 44

=> gọi đệ quy gán 87 vào bên phải 44.

5

3

7

44

12

87

\_ checkBalance(5) = -2 => Cây mất cân bằng tại 5.

\_checkBalance(5.right = 12) = -1

=> cây mất cân bằng Right Right tại 5:

\_Cân bằng:

\_ gọi hàm rotateLeft tại node 5( x = 5) .

+ gán biến y = x.right ( y = 12).

+ gán x.right = y.left ( 5.right = 7).

+ gán y.left = x ( 12.left = 5)

+ trả về y

=> x = 12

44

87

3

7

5

12

12

44

5

87

3

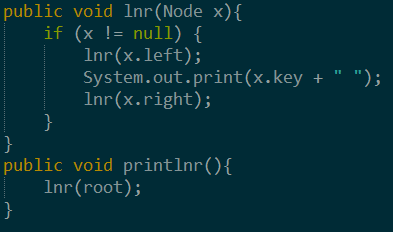
7

2:

Hàm gọi duyệt cây theo thứ tự Left Node Right:



Hàm duyệt cây theo thứ tự Left Node Right:



Duyệt cây:

\_ 12.left = 5 -> 5.left = 3 -> in 3

-> in 5

-> 5.right = 7 -> in 7

\_in 12

\_ 12.right = 44-> 44.left -> null

-> in 44

-> 44.left -> 87 -> in 87

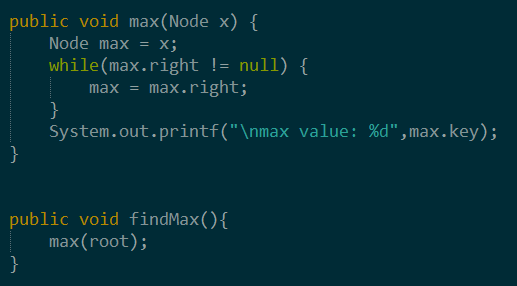
3:

Gọi hàm tìm node có giá trị lớn nhất:



Hàm tìm node có giá trị lớn nhất:

\_ Dùng đệ quy để tìm ở phía bên phải của cây



12

44

5

87

3

7

Node duyệt đầu tiên: 12

max = 12

Gọi đệ quy hàm tìm max với điểm khởi đầu là 12:

1. right = 44

max = 44

Gọi đệ quy hàm tìm max với điểm khởi đầu là 44:

1. right = 87

max = 87

1. right = null

=> trả về max = 87

Phần II:

Khởi tạo ma trận:









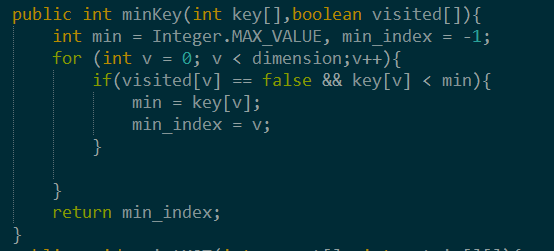
Gọi các phương thức Prim và Kruskal:



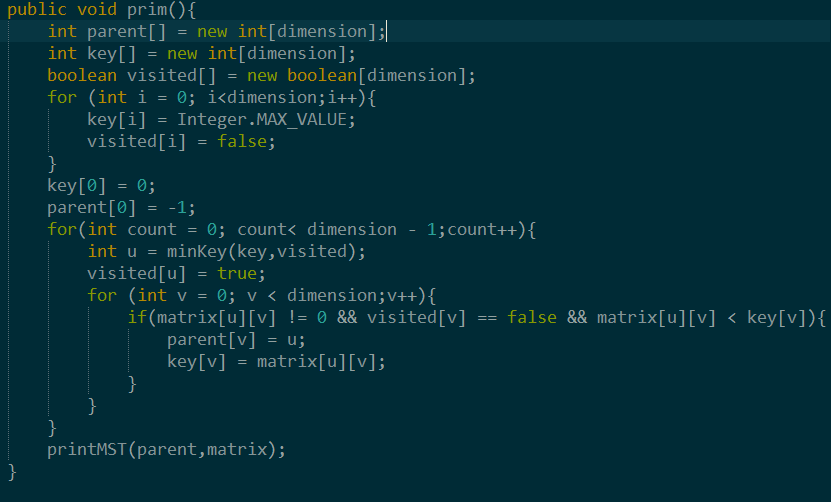


Prim:

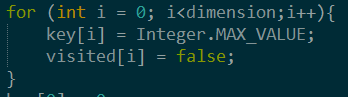
Hàm tìm các đỉnh có trọng số nhỏ nhất từ danh sách các đỉnh chưa có trong MST.



Hàm chính:



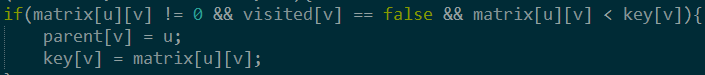
Khởi tạo mảng visited để xem node đó đã được xét hay chưa và gán giá trị ban đầu cho mảng key = inf (mảng key dùng để chứa giá trị cạnh nối tới đỉnh đó được thay đổi nếu tìm được cạnh nhỏ hơn cạnh hiện tại đang trong danh sách )



Chọn đỉnh có trọng số nhỏ nhất (bắt đầu từ đỉnh 0):



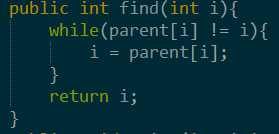
Nếu cạnh đang xét nhỏ hơn cạnh hiện tại đang ở trong key của đỉnh đó và đỉnh đó chưa được xét thì thay bằng cảnh nhỏ hơn:



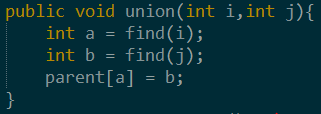
Lặp lại cho tới khi đi hết đồ thị.

Kruskal:

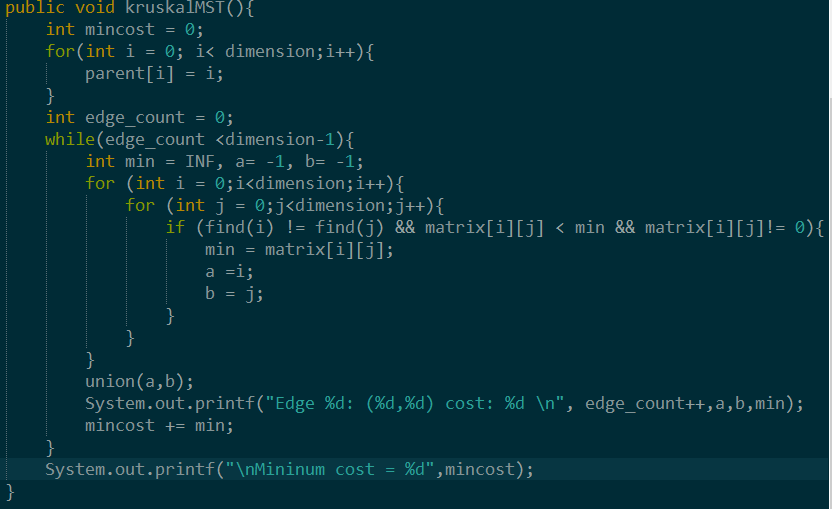
Hàm tìm cặp của các đỉnh



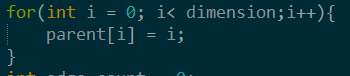
Hàm gộp 2 đỉnh lại thành 1 cặp:



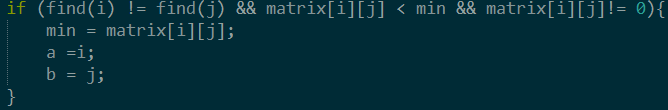
Hàm chính:



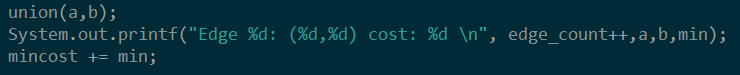
Khởi tạo các cặp của các đỉnh là chính nó:



Nếu 2 đỉnh chưa được chọn và cạnh đó nhỏ hơn cạnh đang trong min thì min bằng cạnh đó:



Ghép 2 cạnh được chọn với nhau để không xét nữa và in ra kết quả cạnh đã chọn và tính tổng các cãnh đã chọn:



Đồ thị thứ nhất:

1

3

5

0

2

7

9

2

3

Prim:

Khởi tạo mảng key với giá trị của từng đỉnh là infinite.

Khởi tạo mảng visited với giá trị của từng đỉnh là false.

chọn đỉnh 0 là đỉnh khởi đầu.

Khởi tạo parent[0] = -1.

\_ Chọn đỉnh 3 vì cạnh nhỏ nhất ( = 2) và visited[3] = false.

\_ Cho parent[3] = 0.

\_ key[0] = 2;

\_ Chọn đỉnh 1 vì cạnh nhỏ nhất ( = 3) và visited[1] =false.

\_ Cho parent[1] = 0.

\_ key[0] = 3.

\_ Chọn đỉnh 2 vì cạnh nhỏ nhất ( = 5) và visited[2] =false.

\_ Cho parent[2] = 1.

\_ key[2] = 5.

1

0

2

3

kruskal:

Khởi tạo mảng parent[] tạo cặp cho các đỉnh.

\_ Chọn cặp (0 - 3) vì cạnh nhỏ nhất( = 2) và bắt cặp (0-3).

\_ Chọn cặp (0 - 1) vì cạnh nhỏ nhất( = 3) và bắt cặp (0-1).

\_ Chọn cặp (1 - 2) vì cạnh nhỏ nhất( = 5) và bắt cặp (1-2).

1

0

2

3

Đồ thị thứ hai:

1

4

3

2

0

5 3

5

9

6

4

Prim:

Khởi tạo mảng key với giá trị của từng đỉnh là infinite.

Khởi tạo mảng visited với giá trị của từng đỉnh là false.

chọn đỉnh 0 là đỉnh khởi đầu.

Khởi tạo parent[0] = -1.

\_ Chọn đỉnh 1 vì cạnh nhỏ nhất ( = 5) và visited[5] = false.

\_ Cho parent[1] = 0.

\_ key[0] = 5;

\_ Chọn đỉnh 2 vì cạnh nhỏ nhất ( = 3) và visited[2] =false.

\_ Cho parent[2] = 1.

\_ key[1] = 3.

\_ Chọn đỉnh 4 vì cạnh nhỏ nhất ( = 6) và visited[4] =false.

\_ Cho parent[4] = 0.

\_ key[0] = 6.

\_ Chọn đỉnh 3 vì cạnh nhỏ nhất ( = 4) và visited[3] =false.

\_ Cho parent[3] = 4.

\_ key[4] =4.

1

5 3

2

0

6

3

4

4

kruskal:

Khởi tạo mảng parent[] tạo cặp cho các đỉnh.

\_ Chọn cặp (0 - 1) vì cạnh nhỏ nhất( = 5) và bắt cặp (0 -1).

\_ Chọn cặp (1 - 2) vì cạnh nhỏ nhất( = 3) và bắt cặp (1 -2).

\_ Chọn cặp (0 - 4) vì cạnh nhỏ nhất( = 6) và bắt cặp (0 -4).

\_ Chọn cặp (4 - 3) vì cạnh nhỏ nhất( = 4) và bắt cặp (4 -3).