Mô tả Kỹ thuật

**Tên phần mềm**: Vẽ và tìm đường đi của đồ thị thông qua hai thuật toán Kruscal và Prim.

(Nhóm thực hiện: BKK000)

**I. Sơ lược về công cụ thực hiện**:

\_ Ngôn ngữ lập trình sử dụng: C#

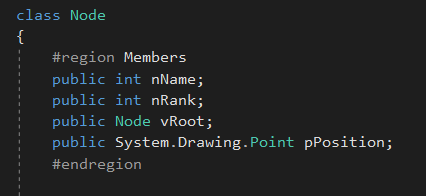
\_ Forms sử dụng: C# winforms application

\_ Phần mềm sử dụng: Visual Stdio 2017

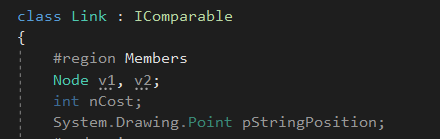
**II. Mô tả về giao diện :**

1. Tạo một số class cơ bản:

Bước 1: Tạo một class Node để lưu bậc của từng đỉnh trong đồ thị, tên của từng đỉnh. Ngoài ra còn lưu một giá trị pPosition dùng để vẽ Node đó.



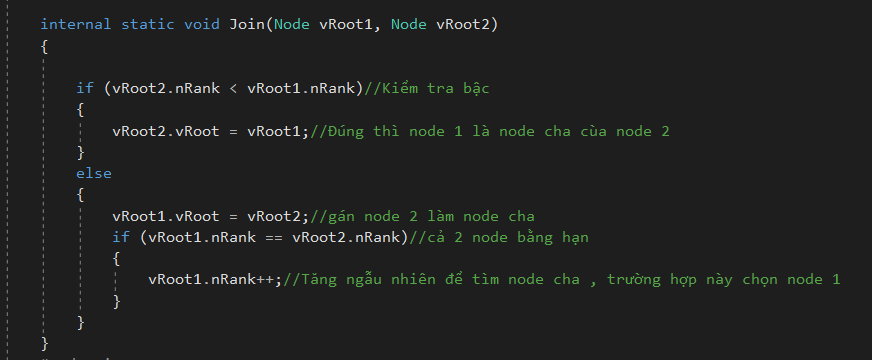
Bước 2: Tạo một class Link để lưu liên kết của hai Node với nhau và trọng số của chúng. Ngoài ra còn lưu một giá trị pPosition dùng để vẽ liên kết đó.



2. Thiết lập các hàm tương ứng trong mỗi class

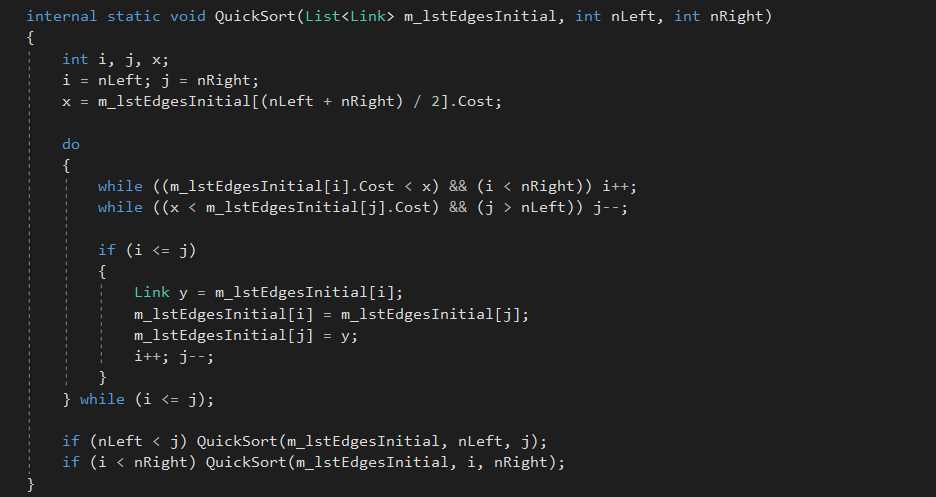
Đối với class Node:

* Tạo hàm để có thể tìm Node cha cho từng Node
* Xét trường hợp có 2 Node cha thì mặc định gắn cho một trong hai Node.



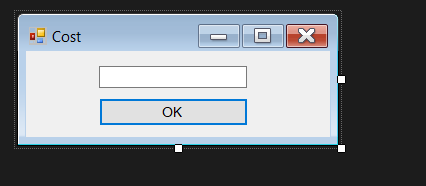
Đối với class Link:

* Tạo hàm để sắp xếp các cạnh tăng dần (Hàm này chỉ được áp dụng cho thuật toán Kruscal).



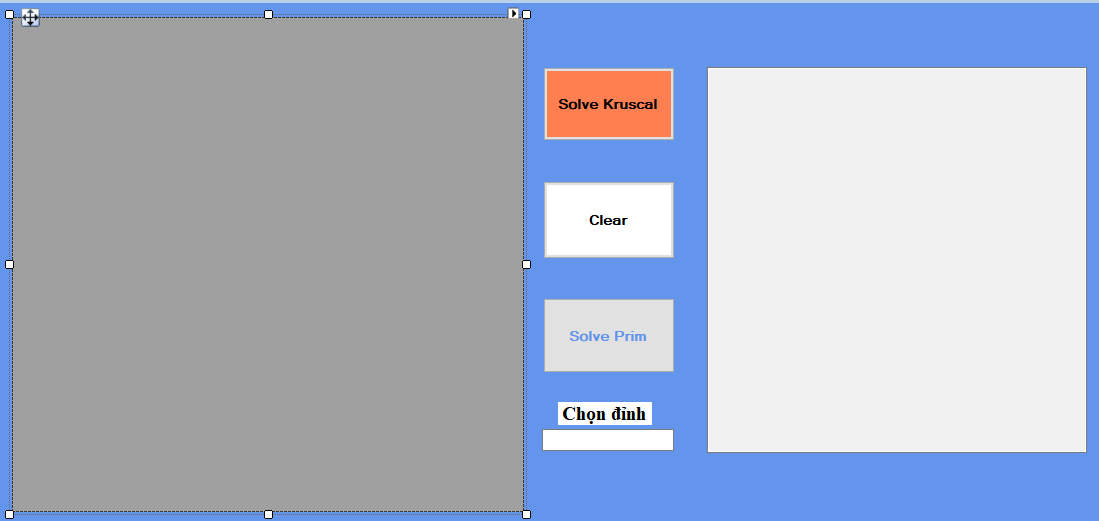
3. Thiết lập các form tương ứng và xử lí trên nó:

Bước 1: Tạo form để điền trọng số cho mỗi node.



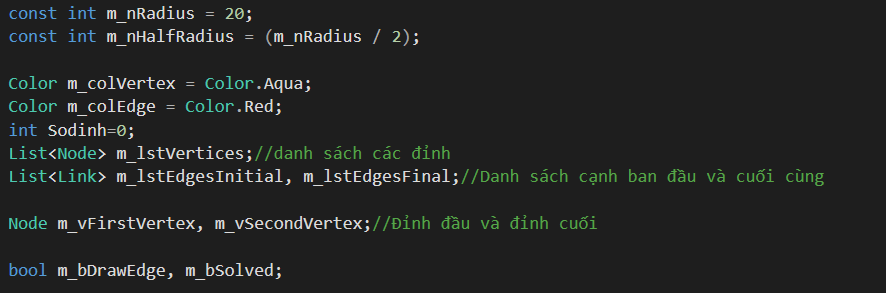
Bước 2: Đối với Form chính, tạo lần lượt các Richtextbox, Textbox và button tương ứng để xử lí trên giao diện. Cụ thể:

* Các Richtextbox để vẽ Node và Textbox để lưu mã giả.
* Các button để lưu các Node xử lí trong từng thuật toán và một số thao tác xóa, đóng form hay chọn đỉnh.

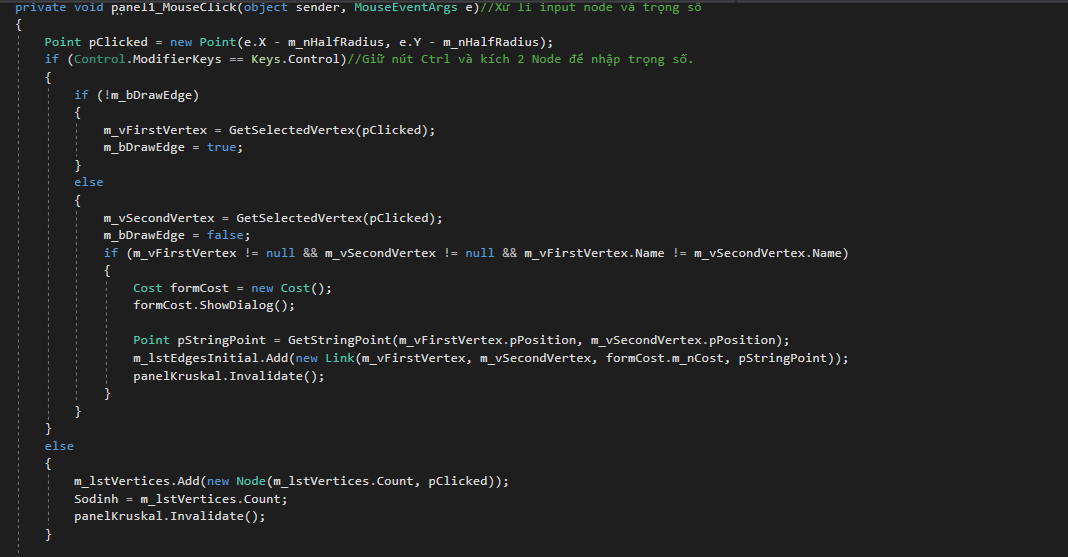


4. Bắt đầu xử lí trực tiếp trên các forms:

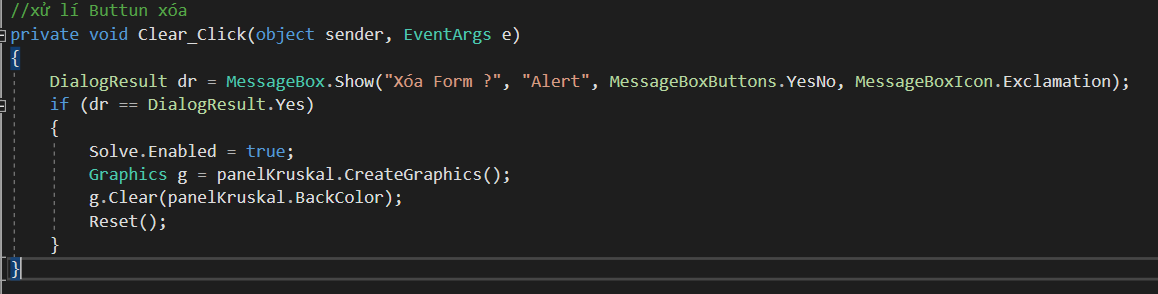
Bước 0: Khởi tạo các list gồm danh sách các đỉnh (Node), danh sách cạnh trước khi sắp xếp và sau khi sắp xếp. Ngoài ra còn khởi tạo đỉnh đầu và đỉnh cuối của đồ thị.



Bước 1: Tạo các event dùng để vẽ các Node cũng như nối liên kết giữa chúng. Ở đây event sẽ là mouse\_click (nghĩa là mỗi lần kích chuột nó sẽ vẽ một Node trên màn hình của Richtextbox). Ngoài ra còn thêm một hàm để chọn Node vẽ hình gì (Đối với giao diện này chúng ta chọn hình tròn).



Bước 2: Xử lí button ‘Clear’ nghĩa là xóa các Node để có thể vẽ lại trong trường hợp vẽ nhầm.



Bước 3: Thiết lập các hàm để chạy hai thuật toán Kruscal và Prim.Cụ thể:

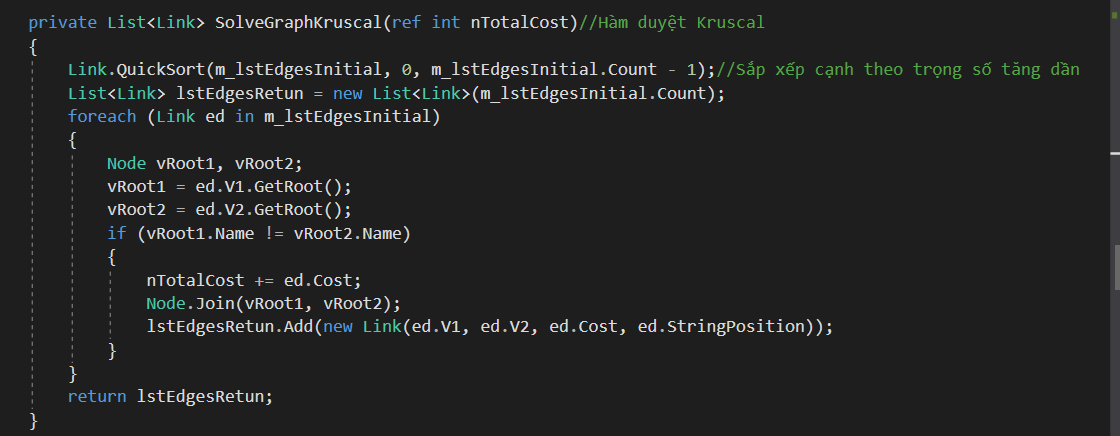
* Đối với Kruscal:

+ Dùng hàm Sắp xếp cạnh theo trọng số tăng dần như đã tạo ở class “Link”

+ Tạo một danh sách Link để lưu lại thứ tự các cạnh.

+ Sau khi đã lưu được bắt đầu xét thêm Node và tính chi phí tương ứng.

+ Trả về danh sách Link đã lưu thứ tự ở bước trên.



* Đối với Prim:

+Tạo danh sách cách cạnh để lưu vào cây sau khi duyệt đường đi.

+ Tạo danh sách để lưu các đỉnh đã duyệt.

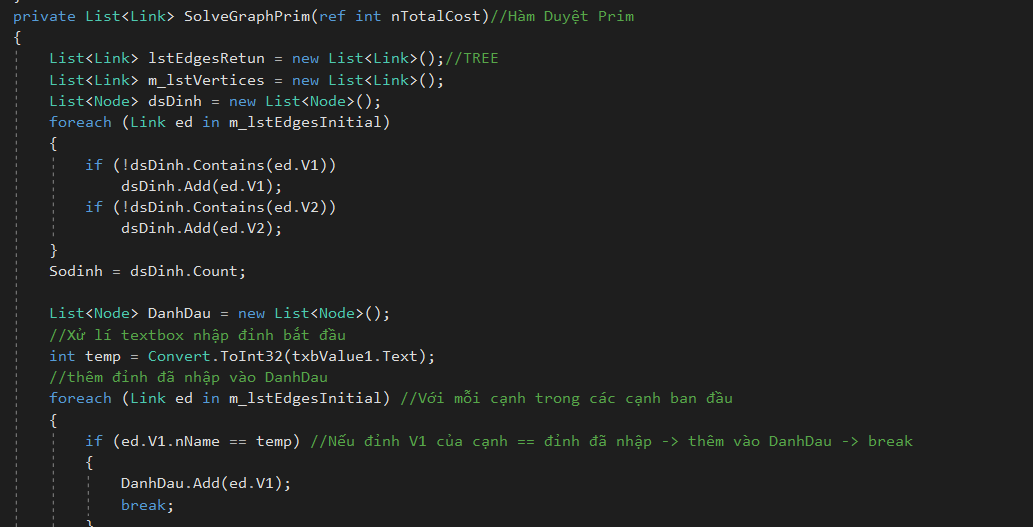
+ Tạo vòng for để có thể thêm vào các đỉnh trong danh sách các cạnh ban đầu.

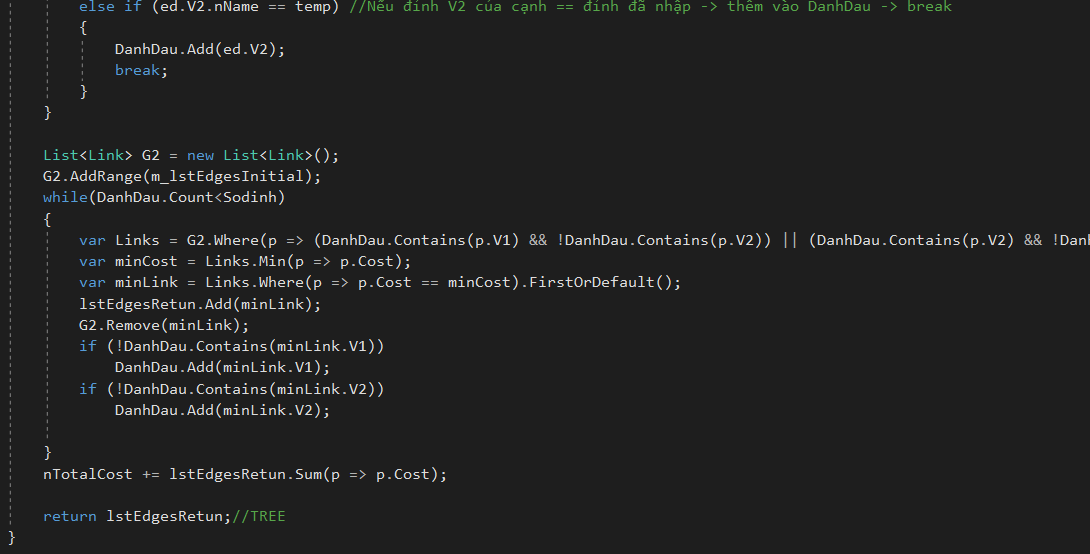
+ Tạo danh sách các đỉnh gọi là “DanhDau” (Link) để lưu danh sách các đỉnh bắt đầu.

+ Tạo danh sách “G2” (Link) để loại bỏ những đỉnh mà nó duyệt qua khi chạy từ đỉnh này đến đỉnh khác.

+ Chạy thuật toán qua mã giả.

+ Tính tổng chi phí và đường đi của nó sau đó trả các cạnh về cây đã tạo ở ban đầu.

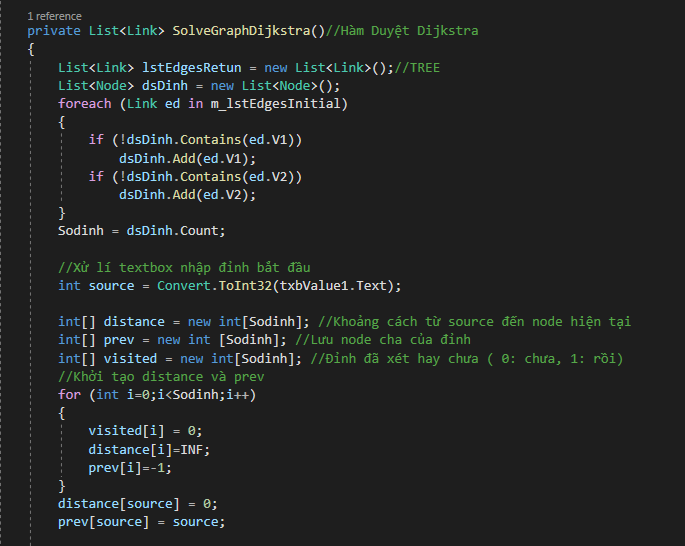




* Đối với Dijkstra:

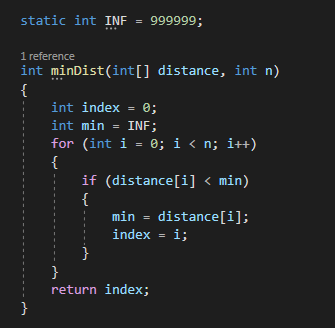
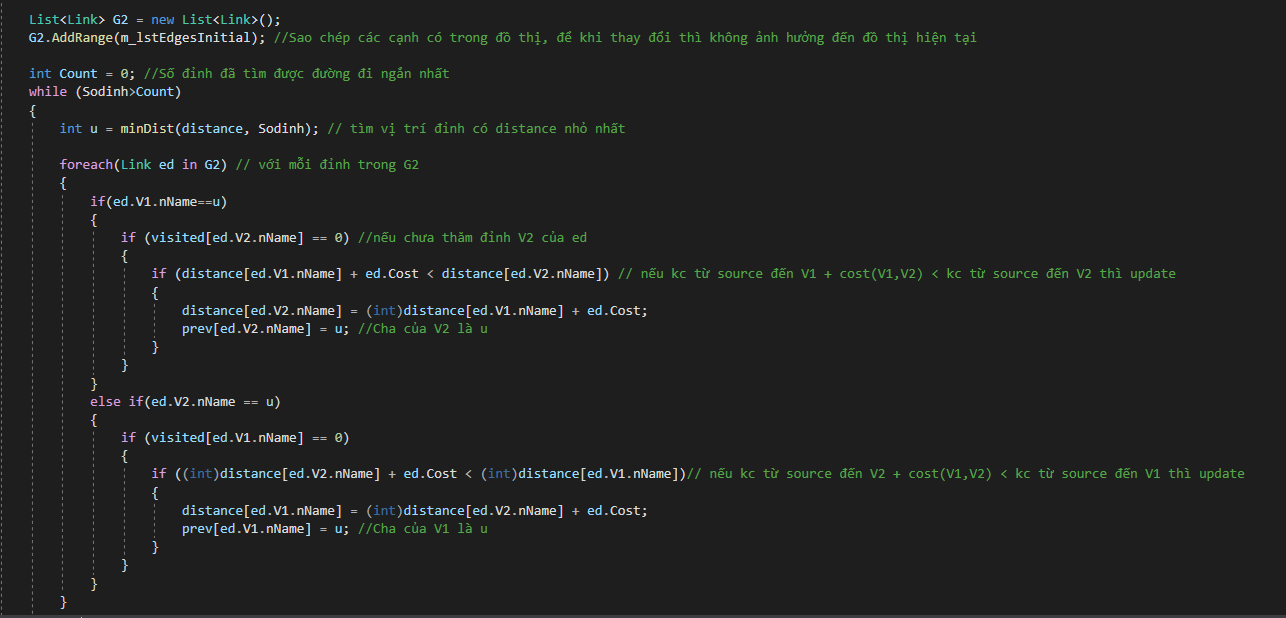
+ Tạo danh sách rỗng lstEdgesReturn để chứa các cạnh kết quả.

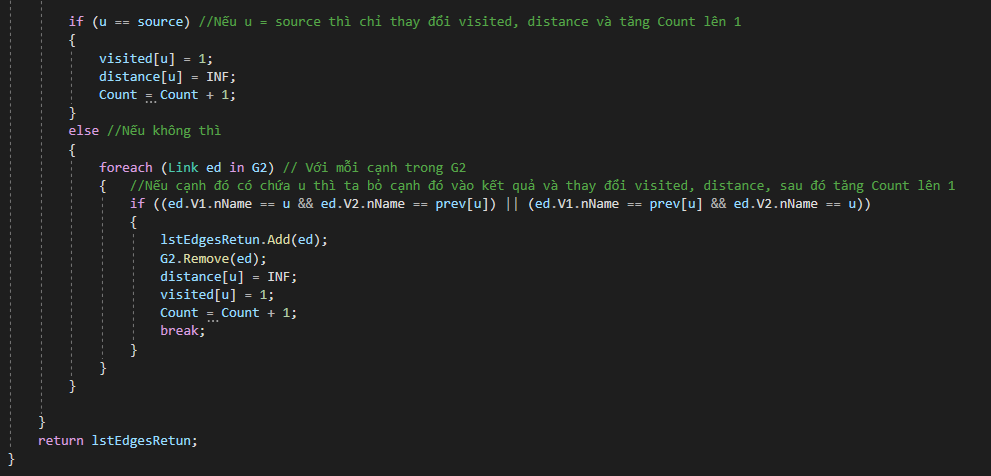
+ Đếm số đỉnh có trong đồ thị.

+ Khởi tạo các mảng distance[], prev[], visited[] để lưu lần lượt : khoảng cách từ source đến node hiện tại, lưu node cha của đỉnh, đỉnh đã xét qua hay chưa. 

+ Tạo mảng G2 sao chép các cạnh có trong đồ thị. Khai báo Count = 0 thể hiện số đỉnh đã tìm được đường đi ngắn nhất từ source đến nó.

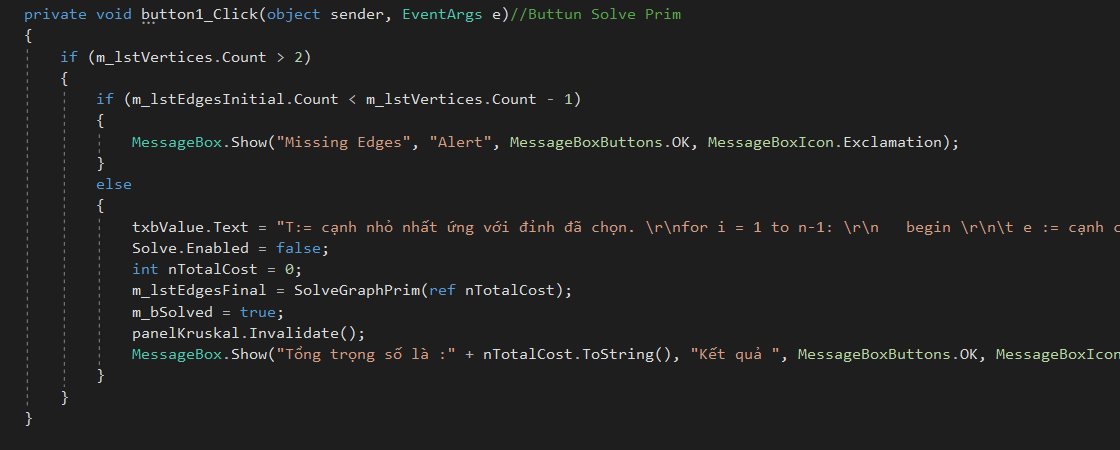
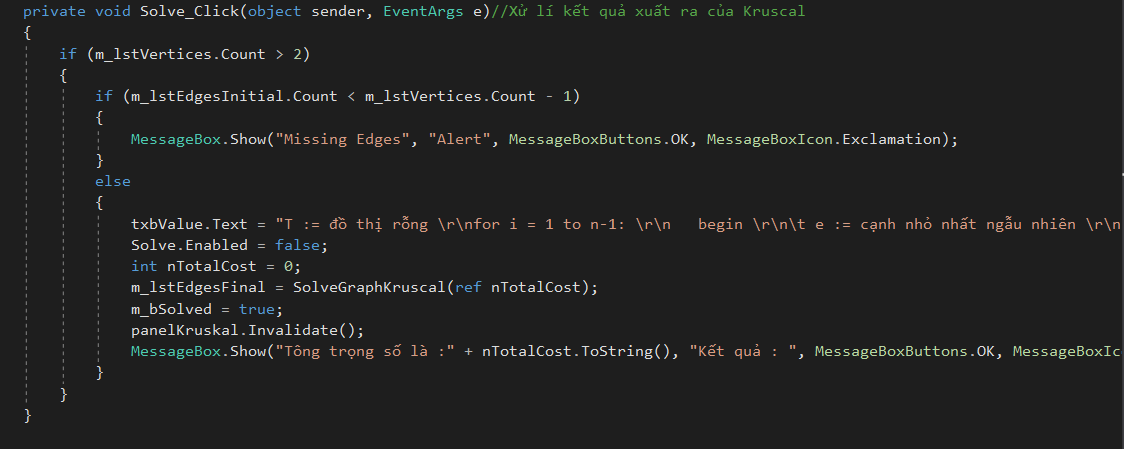
+ Tìm đỉnh u có distance[u] nhỏ nhất, sau đó cập nhật trọng số cho các neighbor của u.

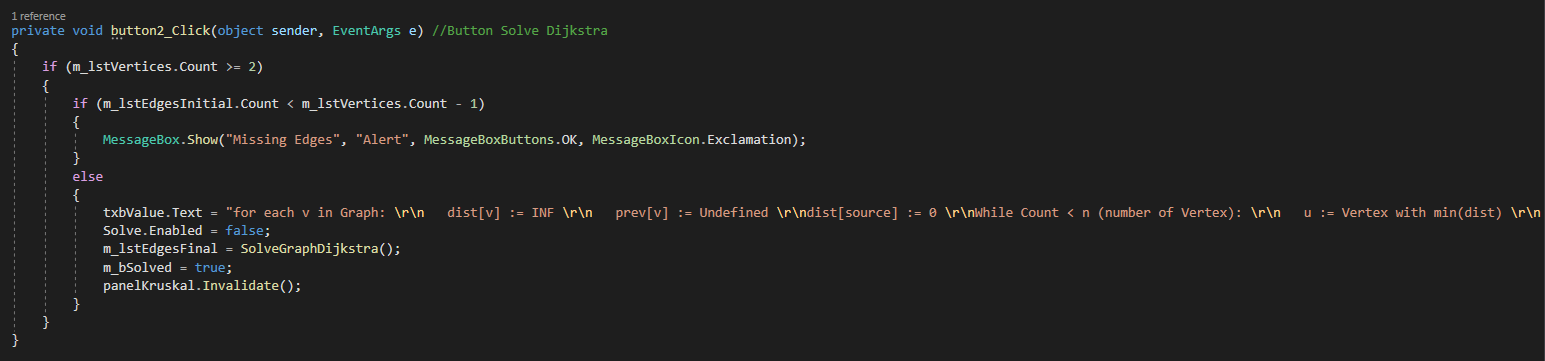


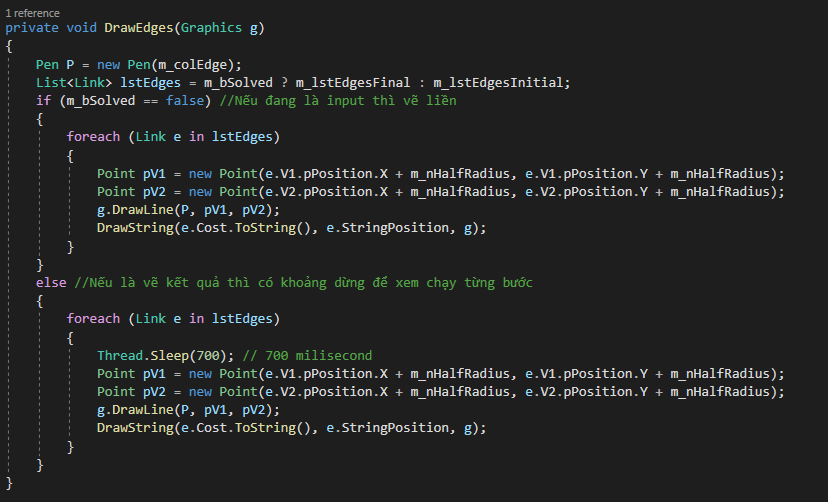
+ Cuối cùng, thêm cạnh chứa u và prev[u] vào danh sách lstEdgesReturn và cập nhật distance[u], visited[u], Count.

Bước 4: Tạo các button Solve tương ứng cho từng thuật toán.

Trong các button này chúng ta lưu dữ liệu Textbox khác nhau để khi Click vào nó để hiện ra mã giả khác nhau. Sau đó vẽ ra các cây đã duyệt sau khi chạy từng thuật toán.





+ Kết quả được hiển thị trên Form như sau: