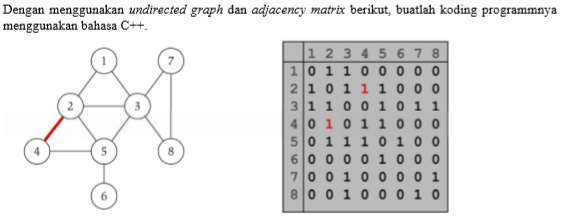
1. **Studi Kasus 1:**



/\*

NAMA    : SURIADI VAJRAKARNA

NPM     : 140810180038

KELAS   : B

TANGGAL : 5 APRIL 2020

STUDI KASUS 1 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA

\*/

#include "graph.hpp"

#include <iostream>

void print(int data) { std::cout << data << ' '; }

int main(int argc, char const \*\*argv)

{

    const size\_t graph\_size = 8;

    Analgo::Graph<int> g(graph\_size);

    g.add\_edge(1, 2);

    g.add\_edge(1, 3);

    g.add\_edge(2, 3);

    g.add\_edge(2, 4);

    g.add\_edge(2, 5);

    g.add\_edge(3, 5);

    g.add\_edge(3, 7);

    g.add\_edge(3, 8);

    g.add\_edge(4, 5);

    g.add\_edge(5, 6);

    g.add\_edge(7, 8);

    try

    {

        std::cout << "Adjacency Matrix dari Graf tersebut:\n";

        for (const auto &node1 : g)

        {

            for (const auto &node2 : g)

            {

                std::cout << g.is\_edge(node1.first, node2.first) << ' ';

            }

            std::cout << '\n';

        }

    }

    catch (const std::exception &e)

    {

        std::cerr << e.what() << '\n';

        return EXIT\_FAILURE;

    }

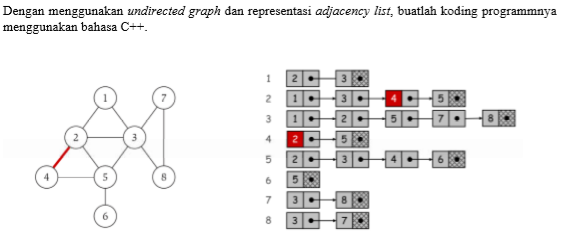
    return EXIT\_SUCCESS;

}

![A picture containing drawing

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4REGRXhpZgAATU0AKgAAAAgABAE7AAIAAAAUAAAISodpAAQAAAABAAAIXpydAAEAAAAoAAAQ1uocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFN1cmlhZGkgVmFqcmFrYXJ1bmEAAAWQAwACAAAAFAAAEKyQBAACAAAAFAAAEMCSkQACAAAAAzM4AACSkgACAAAAAzM4AADqHAAHAAAIDAAACKAAAAAAHOoAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyMDIwOjA0OjA3IDE1OjI1OjU4ADIwMjA6MDQ6MDcgMTU6MjU6NTgAAABTAHUAcgBpAGEAZABpACAAVgBhAGoAcgBhAGsAYQByAHUAbgBhAAAA/+ELJmh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjAtMDQtMDdUMTU6MjU6NTguMzc1PC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPlN1cmlhZGkgVmFqcmFrYXJ1bmE8L3JkZjpsaT48L3JkZjpTZXE+DQoJCQk8L2RjOmNyZWF0b3I+PC9yZGY6RGVzY3JpcHRpb24+PC9yZGY6UkRGPjwveDp4bXBtZXRhPg0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICA8P3hwYWNrZXQgZW5kPSd3Jz8+/9sAQwAHBQUGBQQHBgUGCAcHCAoRCwoJCQoVDxAMERgVGhkYFRgXGx4nIRsdJR0XGCIuIiUoKSssKxogLzMvKjInKisq/9sAQwEHCAgKCQoUCwsUKhwYHCoqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioq/8AAEQgAvgJ3AwEiAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A8l0nQH1TTb2/e/tLG2smjWWS58w5L524CIx/hNXbjwXeaes8mr39jp8Mc4gjmlaR1nJQPlPLRiRtKnJA+8O+QHaBrVhpvhPWrW8ggu57mW2aG2nEm1wpfccoVIxkdx+NaWkeJZru/ku9a1zTI7K6uEe60y5s5JUKKAo2IImVSEAUEMDwMmm92l5fp/wRHFOoSRlV1cAkBlzhvcZwfzrorvwTd2sdwF1HT57q2tVu5bON5BIIiobcNyBThTkgMTjNYeoPayaldPp0bRWjTOYI3OWSPJ2g8nkDHc12up6voZ1K91iLVVuZJtJFlDaQxSq4ka3ETFyyhdo5PBOeKSty6hrf+u6/S5xa2UqC1mvY5re0uW+S4MRIZQcMVzgNj0BrZh8GX914wn8P200PnRhnSaUlUkQDcrDGfvAgj61nXzxnRdNVdWlu3USbrJ0cLZ/NwFJODu+8duPeus0zXbFdDTVZ7yODU7bTptMCHl3OMwOoHPTcpboNq+oy3bXyv/X9dwd+n9f0jGs/BF/dadb30l1Z2sE9vPcZnMmY0hKhiwVCed4IAzkVXg8Mm5W5nh1awNhaqpmvj5yxKzEhUAMe9mOOgU8d66jS/GlveLcTard2lndzxXiKr2zSQRl0t1jBXa+VPlNxg9OetU01myl0m60q71XS1n+0xXttdwaeVtWZQVMbxiEHODnPlkHuaJaPTy/r5AtTG03wfe6vPfx6ddWc6WcYfzVdtsxZSyqny53EA8EDBBBwapS6Rt8N2+rwTeajTtbzx7MeS4AK855DDPOB90iu4vvFnhnS5bZtKR7pvt0V5J/Z5+yRq0cSIBtaLkFvMOAB196wbC702fWtf0hJ44dL1NpPss0h2LG6MWhYk/dH8Jz2Y0mu39bf8FAnfV/1/W5mXvh1tNsYpdQ1Gzt7mWBbhLJhK0pRvu8qhQEjnBYcHnFXpfBEsWtx6P8A2zpjajJIkf2ceflSwByWMW3AByeauQ6pZjw1cWmvavY6uiWRjsIEtpTcW8pI2YlaNcKvcbiMDABrcuPE2lv42i1X+2dJewR1YRjTXFyMR7eZPIBPzf7fSh6MNbaHMWvgDVbqQp51pERqD2GZHcfOqO5bhT8mEYZ657VTt/DP22eRbHV9PuIIIWnublRMsdugwMtujDEkkABQxrpPDnjiKS+tZtfuYrc280CqywswKJDcKXYAHJLSrn1z04rNj1Ce31WO5tPFOiWsixMhkttPkhjdTjKOi2wD59GUjim9H/XYLlHS/Cqazqv9n6fr2mSTMCY8rcKJMKWOCYuMAHrj2zTY/C6Pp0uoNrumx2SXAtlnZbjEj7A/AERbGDjkDkH2rdtNc0Cy+IenahA8MNtFZlLye2t2SKScxOpZI8ZCklR90cgnAqpoOrWsPgmTTv7S02zuzqHn7dRsTcIY/LC5H7qQA5+hpP8Ar7/8hlS08E3V9bWslrqNg8t6kz2lvmUPOIiwbBKbRnaSNxH4VzVehaf4r06307TNLmvIUQ293BcXkFqVe1d5GKSIwUMFPGVXAwSCM8VwMDRpcRtPGZYlcF4w20uueRntn1o6/wBd3/wANlvCl5beHbTXL+WO3srmdYwvLShW3Yk2cDb8jY5GcenNVr/Qbmy8TNoqETTGdYoXAwJQxGxh14YEH8a6LXfFek634Vv41t7q3vZ72GSKB7kSJGiIyjbiJQFAO0LnPIOTg5n0bWdJWPSdf1O4UXmkwyWrW4bEk7KhNuy8H1KliMDYvtl6Xv0X+X9feLp5/rf/ACMtvh9q/wDad9YxS2sslnafa90bsVnTpiP5cls8YIHIqvp3gvUdSvrO2hntYzd2f21ZJHYJHHvKDcQvB3DHGeo5roYfGGkQeFx9gNxa6jbwGGFbl/PdttxFKnzhFGP9ZwRxj3FWta8VeFv7IP8AZ5uZpLi3ggNvbubd7ZVlkmZd7RspAZkHHXFO1lr/AFv/AMATu3p/W3/BOEl0O7h0q6vptiJa3YtJYyTvEhDHpjGPlPerMHhmV9Ntr281Cx09bwt9lS6dw0wBwSNqkKueMuVH4c10esatoevWerRW+pQac17fW96q3McpGfIIkGUjPIdj2APUVNp2taDNdaJLqOoWKppsP2G8hu7Fp0uIlYlXizE2M577CMVNmtH5fl/mO+l/X89PwObPhC8Wzd3vLJbyO0+2tp5dvPEWM5+7szt+bbu3Y7VHc+GpU8QafptncLMmpLC1rcSLsVxJgZIGcYbIPX7prbvtW0a71K78Q/2jie5014P7PEL+YszQ+Sfmxs2DJbO7OMDFR+HNUsWsNJk1K5jgl0HUopVLnl7Z5AXAHVirDdgdmarSjzJeev3/APDCu7GXfeEb22h8yxng1VRctauLESM0coUttKsiseFY5AI+U81RbQNYS6gtn0m+We4UtDEbZw0oAySoxkjHpXWaX42t4PGELiO207Sor2a6Z7eORjK5R1R23Fmz83QYA3Hisvw/rcN1LNb+Jr8tbpBO9us7S+W00hUt5hiHmFTtyR6j3NZRbau+36f5lPRlJPDF0ujaveXwms59MaANazQFWbzSQM5wVxjPQ5zU1t4F8SXNjd3K6ReJ9lKAwvbSCSQsf4F2846n0BFdynjrw7piyGF4rsJZ2UCRwwygBo3k3NGZcnKBlZS57Dv05K1udMS38SaTJraOt+YXt9Rmil2ylX3HeApcMQx7EZB56E099AMyLw5OPD+oalfreWhtHWOJGsZCkrbtrqZMbUKnsep4pdV8M3Vg9qtoJr7ztNj1CXyoD+5Rhk5xngf3jitDSBpdv4X1uyuNesY574RJEDFcEfu5d2SRFwCBkd+ecVra34m0jVvCNtpEF59lmtrC2ZpVifbdSRqQYH4zx1U/dyTnHWnp+X9fr+ALexxaaNqklxLbx6bdtNCQJY1gYtGT0BGMjPbNXbTwvf8A9pWsGs215pVvcTrB9quLVgqOwyoOcD0PXpzXfaf4p8N2niPVNXbWlI1Ge2lSEW0u6JUZS247cZHPC5HHXtXNeJ/Edlqnhu3toLxp7hI7IMrK/DIlwJOSOxkX65780R2TZN27nPWWg3t74hGihVhu/MaN/NJCxlc7i2AeAAT+FSXHhnV4tXudOgsZrye2kCObSJpAcglSMDoQCR7V1X9t6KqXHiK8lM17qOm/ZZLSCUJKJ8iOV87WCAp8wJHJZsd8R6r4q0q78MX0enzT211d2lnDJFIzO7mJ5Fbc4VQ3yCM9BnPqDS1S/ruVdP8Ar+v6RydvoOsXlxPBaaVfTzWx2zxxWzs0R54YAcdD19Ks2mi2134Y1HUkvJludP8ALMlu1uNjB32jD78574212za/4Xutck1Ga/ieVb+CYG6W62hEhQFo1jwPM3gjL8cCsg3ej3Fr4q/4ntnC2r3G+3R4bjKqs5fLYiIGV6Yz15xQ9P69BXurnLxaDrE+nG/h0q+ksgrOblLZzGFXOTuxjAwcntiprXwxrFzPYI9hcW0WoTRwwXNxC6RMX+6d2OR34zxXodxLFBZmG0ED6jD4fhjubeaWQf6OiLK/7sRFAxHGfMI56ZOKoW/iXRbXxFqGqtrLXEWqX1rMLfyZA1siTLIxkyu35ApQbC3XjitElzpPa/6iu+W/9anHXXhbULSCNm2SzzSyRxWsQZ5ZFjyGlCgfcypGTgnB4wCaZo2gT6vcpCfMtzcxSmzZosrcSIM+WCSBz0yM4JAxzXWf8JJol14ytPEE16YH3S2txD5TsAhjZEmTAwFwRlOoOcA54x9W1Oz0/R/DkGjaml5d6ZLNM8scUiKrs6MuN4BI49B06VlG7Sv/AF/W5UtNjK8P+HrvxHqZsrOSGFlQyPJcMVRACByQCepA6d6jh8PaxczSxWmlXly8UjRuILdnwy43Dgdsj8xXXXmq+HdOtdVuLdpLh9beCZLezuBFJbJzI6lijBcSADb1wAenWXVfF2lXDQSafdSQLLrcWoTw4f5R5UZckgAEiQP09Mgc0/6/Ffl+gr9fuOIt9G1O7s5bu1027ntoM+bNHAzJHgZO5gMDA55q5NotsPCMetW15NI/2oWssEluECsULZVg53DjHIFdzpviDwpb65Fqb30O9bu7kkadbppEV3bYIlXEYUqcndk8niuZgXTpPAg0tdZtGvpL9LsW4inLEeWV2Z8vbvyfXb/tUK7/AA/MG7Py1MOXQNYg08X82k30dmVVhcvbOIyGxtO7GMHIx65q3a+ENan1OCxubGbT5LhJHja9heJWCIXbGV54H6ivStT1e007WdXvrZYZ47TULafVYvNkd0WNhGuxGiVFO4g4Dt0wDisHQvEOj+H5gkuuDUfNv5rszLBKUjH2eVBuDqCWdpBkAEfLyaelm/X8v69bD1vZ/wBb/wBfM4y88P3tlbwO4WWWS3+1SQRBme3i42tJxhQQQRz0IzjIpbHw9e3geSfZYWscIme6vAyRhGyFIwCzbiCBtBzz6Gukj1fRbjUtSvm1H7M+uadLFNHNHI4tJyynBYKSUbadpXJGcEDHOjP4o0+LSf7J0bxF9ilSxtY01JYZYwzxPJvTKqXUEOCCBzyDjJFL+vz/ACF/X5HCSaPc/Zrq7tCt7Z2jqktzAG2ru+6SGAYA4IyQBniqFdw3ibS7XxZqfiCGaS5kKLDFaiLy0vWMYWSSUdPLLAsV+8xYZxya4u4lE9zLKIkhEjlhHGCFTJzgZzwKB9C7D4e1q40w6lb6Pfy2Kqzm6S1dogq53HeBjAwcnPGDVSS0uYbaG4mt5Y4Ljd5MrIQsm04baehweDjpXr17LHBpj21iLaTVYfC8UNxb3M8wH2ZIxK+Y0i2BiMgHzu/YmuT1jxZotz4Q8P2kGiabLPa/aPOti12Ft90oYBT5ozuHJyWx2xXlUcbVqSXuaN/hZtPfy3/4BfKrX/rocbcWlzaeV9rt5YPOjEsfmIV3oejDPUHselQ113jnxFpetwaNHpllao1vptvFLLF54aJlDAwjexBUZHOCf9o1ykTrHIGeNZQP4XJwfyINd9CpKpTUpxs+xL0FhgmuH2W8TysBkqiljj8KYysjFXUqynBBGCDV6KaCa1ngJjtDI6OOGKkAEY7nvmob+Zbi9d4yWXCqGPVsKBn8cZrYQ+20fU7yylvLPTrue1hz5k8UDMiYGTlgMDA5rR0jwpf6pp17qDwXNvZW1pJcJcm3YxylP4A3AyefXp0rpfB+r+GtLt9Iur28hS6t/O+0faBdPJHuJCiJU/dhSDk7snk8Uxta0R4by+OrBZJtBGnw2Agk3RSBACM42hdykjBOd3OKJaJ2/rcFq/67o5+bwjqFvokd1c2mow30t0IIrJ9OlXzFK53ByME5BG3r3qjP4f1m18v7TpF9D5kghTzLZ13uRkKMjliOg612PgzULbSPCdnfahIYbePxDGzSbScAQtnoMnGR+dWB4n0G+/s2LU9SmMUcdoJivmqQyRXIcFlG7G6RAdvJDHHeqtq/66IV9P67tHJ6b4K8QalrCaaNMuradk3sbm3kQIuDgt8uQCVIBxyeKqHw3ri3Jt20bUBOApMRtX3AMSFOMZ5IOPXBrsx4k0Gx1Dwr5Nxb+Tp7XK3X2KKfy4hKAAV8752HJJ+hwOmcKzv7XQvDfiHT7bWIria/jgWJ7VJVDqHO9SXRSPl69iDjnmoGUtO8L3VwmpPqMV9YpYQO7N9gkkHmAAiN8D93kHOW4FZ8ekalM1ssWn3Tm7BNuFhY+cB12cfNj2ro/D+qQXVzqt/r+t28Nxd6fJZKJopSxJjVEY+XGRjAAJznjoauaJrWlaZ4cl0O41os175w+1xROUsMrgYDJuIkxhto4BHfNN6fcJNvc4Srlxo+p2ljFe3WnXcNrNjy55IGVHyMjDEYORzVMdeeleqW/iXwnpLXclrcwzxzXNpIoC3Mk8qxyKWaUyfJuHzEBRj36U0rg3Y42TwTrEWhxXstjfJdzXf2aKya0cSSDy9+9e5HBHA7daZqPhHULa6traxtNRvLmS1SeeH+zpY3hYkgrgjLAEfeHBq9rl5pieEXsLXWV1S8m1VryR1hkUbShXOXAJPQn69+tdOuuWeiW+krfXLWzyaNY4ARiSoud7dB02An3H1pLX+vL/Mb00X9a2/I88Xw9rT3gtE0i/a5aLzhCLZy5jzjftxnbnv0q7ZeCvEF/p15eW+mXRW0cI0X2eTe7FtpCgLyVI+Ydq6i61vw5qsl095fRSSlCIhefaRA2bueQlhEAxYI6EZ4+Y96g1bXdG1fU/FEEepJbw6mlq1tdTQy7CYtu4EBWcZ55wenXvQ9HoHU5GPw9rUyu0Wj38gTduK2rnbtOGzxxgjB9DU8vh+SDwvDqs/2qKa4uRFDA9lIElQrkOsv3TzkbRz3rUufE0mneHvD9poupk3GmT3Er+WjbAxf5HwygNlSeo6EggZxTU/sw/D86cdcslu2vVvPKMVxwvlFdhPlY3ZPrj3oWwupgS6VqFubkT2FzEbTb9o3wsPJ3cLvyPlznjPWqqK0jqiKWZjgKBkk+ldt4k17TdS8LDS4NUlln08w7ZnjIGo/LtJ+6CPLyQu4jK571Q0S50fw34t0q9i1Jr+3EW6eRbVkNtIysuAGPzbCQcjrjihB0MqTw1rsM6wzaLqEcrjcsbWrhmGQMgY55YD6ketNXw9rT3j2i6RftcxkB4RbPvUkEjK4yMgEj2BrqPCtrFbXOum38QWsxl0iYtdRC4XyvnTlsxhuc/wg1p/bLPUfBmp2dprEcUNnZ2FnLeyJKscpDysQAqlyvzBRlR93oKNl934tofX+vI83ngltrh4LmJ4Zo2KvHIpVlI6gg9DRWr4t1K31fxPdXdk7yQFY41kkXaZNkaoWx2yVzzzzRQBX0zQb/V7e4uLJIfJtiomknuYoVQtnaMuwHODVkeEtZD3Kz20dr9ml8mRru5igXfjO0NIwDHHPBPBB6EVoeHb3TLfwXr8Oq7pfNmtSlvFcrDJJgvkqSrZAyM8d+1a1jrVx4jumGr2ujNok9yhNvdahHDJaKqrHuQmQSZ2KueCGx0pu12l5fp/wRHAuhjkZG2kqSCVYMPwI4P1FbNz4P1u0tJbiW1jKwxLPKkd1FJJHGwBDtGrFgMEHJHeszUEtY9Suk06RpbRZnEEjjDPHk7SeByRjsK7/AFWS0i1m/wBabVbL7NJootYkhuo5ZJpWthHs2KSygNySQOlJfDdh1t/W6/zPO4IxNcRxGRIg7BS752rk9TjJwK1E8L6tN4juNDtrbzr6BnDIrBQQvJILY4xyPXijUpbx/DujJcaja3Fsgm+z2sTAyW2X+bzAACNx5GSePSux0jUUNhH4pE8cV3b6XNYyl3Ck3CKPJPJ+Ysnb/YNN2V/K/wDX9dwbf9f12/I5Cz8J6zfWlvdW9qnkXEcsscstxHGpSMgOxLMMAFh1xTYvDGpz3E0UKW0ggjEk0yXsLQxqTgbpQ+wEnsTmu40vxHZatbzs0dhbK9vfRQWFzdiONQUtgqFiyYDFW5yMnd6VmwtbSeHr3SPJ0WzvBdQ3i2qagDb3UahlKGbziAQTnaXB9KJaP7vx/wAgWpzEHhfWLq4vILa0Er2UXnT7JUICEbgQc4bI5G3OR0qtNpU8Oi22qbo3t7iV4fkJzG64O1uMAkEEYJ4r0aS+8NeFZIPsN+sImv4bmSLTtt2uyOJQ0bMZchS7yd2/Suc0+G2u9Q8QeF7OVZYLmV3051bcGliZimD0+dNy59xSen9en9eqBNPX+v6/QwpPD2owacl7cLbwRSReciTXcSSunZhGWDkHHGBz2q3J4L1uO8js2itPtcjqi2y6hbmXLYx8gfcOueRwOa2mMWo+Hp28V/2X5lrYbbG9tr6NrlnGPLjaONzuHYllBA6kVv3NzYt8RItT8vQhZrIj/wBoJqimVsRY5TziByMfcFD0fkF9NDgrbwhrl2xWCx3EXjWRzMigTKpZlyW7BSc9OOtMXwxqcl2LeAWk7mNpWaG/gkSNF6s7q5VB7sRXb+G/F9vqeoWsl81rYiO6hLrLOFWV/IufMlJOMbmdQfTKjNYlq+oafqLHT7Hw5bRzQNHcWw1WKSG4QkcOXuGwc4I2sDxTas7f1t/XyC5j2nhHVr+9NpZfYZ5wM7I9StiW4J4/efNgA5xnHekj8J6pJDLMv2EQRSiFpm1K3WMuVDbQ5kwxwexPf0NdNpZ0PSfiXpktlPbWsH2NmuwLoSQwTtC4KLIThh93+I8kjJqpoMkE3w/ktPI0m8uBqnm/Z9Rvhb4TygN4/exk88dTSf8AX32GYsPg/Wri1S4gt4ZUkEjRKl3CXlEZIcogbc+Np+6DntWJXpmlavp9tp+i2RlsLW/+y3sUN4LjcLGVpG2gncV2MOMkE4IINebweUbiP7SXEO8eYYwCwXPOM8ZxRu/67sC+nh7UTpNvqssSwadcTiBLqRwF3HPOB82ODyB2Peob3Sbuw1qXSpo911FN5O1f4mzgY9Qe31rs/EWuaNq3gq8Wxvp12XsC2ljLbpGYokR1CgCRiVAJJbHLHpzkSaRJY3s+jeK9RnSNNOhaG83EFmmhTMJAJG5mXbgdzGfenpfyX5W/r7xX083+dzlZPB+uxXt7aPYHz7C3+03CiVDtjxncCDhuvbJqOz8L6xqF5a2tpZ+ZNd2/2mFfNQbo8kbsk4HKkYOD+YrurbxLptr4e/tKx1RrjUYLQwMLtVhlm2XMUigoHbIKu468hT6Gp9Vv/Dum6StxYazsMtlb2tsbNUnmhTzpJiGQyLjjy1OTTtZXf9b/APA/EV3fT+tv+CeZnTLxdPlvmhIt4ZxbyOWAKyEEhcdeint2q1ZeGtVv7FLu3gjEMrlITNcRxGZh1EYdgXPOMKDzx1rr9cWw1jTtbi0O9sv9K1O3vUS4u4oCQ0BZ/vuBwz4IBODx2qS0i0jVpfD6ahJpUtpZW5sb+G5vxCbdlckyIRIocNngruFTZrfy/L/PQd01f1/P/LU44eFtYOmG/wDsi+SIPtJTzk80RZx5nlbt+3/axjHPSmXXh+9t9bt9LQR3E90IjbtE2EmEgBQgtjg5xzjnNdVfzWk+s3viOPUrRbK40l4khE6+cJGt/JEPlZ3jDc5xtwM5qPw5NFqFpoN7PKkcugajDHO8jAAWzyhlYk9Ar7h/wMVaiuZLz/W3+QruxzGp+HdS0iBZ72GPyWlMPmQXEcyrIOSjFGIVsdjg1mV6NovijTLfxlFaWkSWtj/aM95Pc3d0sglfy5FXB2qqp8x45J3dao6F4ju9dvx/bOq/Z5rG3uJLWZbiO2lkZyv7pZpMrGBzjAGBuA61lF8yv5foU9HY5a00e4vNF1DU4njEFgYhKrE7m8wkDbxjtzkioYNOurmwur2GLdb2mzz33AbN5wvGcnJ9K9ck1rw5B9rbVrq2vFks7ATA3SXTPKryAs5XHnFPkLYHIA6g889Z65rQg8S6bL4pD6k7QyWtwuqCOKTD/OY5CwVcqR8uRwMY4xVPeyA4i1077Tpl5efbLSH7Js/cSy7ZZtxx+7XHzY6n0FO1bR7jR2tBdPG5u7WO6j8sk4R+gOQOeK6rTP7QuPBev2V9q9q7XLRtBDPq8PzSLNmRgGk4JwTu/i6gnNaetarYXHhG3t9G1C2t9Tj0m2W7kMyg3EAUhoFbPysDy0fVwcc4xTsn96Bb2PM6t6Vp0mrarb2EMsUUtw4jRpSQu49ASAep4/GvWtNvdMtvFms6rNrGli2v7m0e3/0yMsyqyliVzlcdw2DweK5zxj4i/tLwxagapHcXCixlKpKCwk2XHmNgdx8gPp8o9KI6pNk3vexw9ppl5faomnW0DNdu5jERIU7h1BJwBjHemX1lcadfTWd7GYriBykiEg4I9xwfqK9BWTTUu7rxfc3X2VNR05vLEKq0iXb4il2IWXcRln6jAcfi/UPEVnb6DqN/oWrFb67tLHMh2xTmRDLE5ChiVbaqk4PR+uDS1Su/61t/XqVdN6f1/X6Hmlalvob3Xh+71aG9tmWz2edbfvBKoZtqn7u08+jV6VBe6eviO71G011IvO1CB5o7fU4LRCghRnkckEzAsWGwd93c1zy2UL6d4wSzu9LjivbhRZRnUII96pcE8KXGBtGRnGRjGaHp/XoK91dHIvr2ryacNPk1W9eyChRbNcOY9o6DbnGBiq1naT399BZ2ieZPcSLHGmQNzE4AyeBya9Za6bTvDgtraSaWFtCijn02O5hjjVSBJLNsMol3bS2f3Yzk8kVFFrESa5cS32t2M2kSahZtpMC3KMtuqzK2RHnMAWIMrZC9cc1cYpzs+/6/1b/gCv7t/wCv67nmd1ot/ZWIvLuDyoDO9urNIuWdfvADOSB0LAYzxmk0rSp9YuZLe1aMTJC8yo5OZNi7iq4By2AcDjpXoH9o6ddeObDWY9TtYbP9/ZNbyTKv2Z/LcAgd43Lbt/TLHJ7nF1G8Ph7R/Cn2a9s7i+0+eeaRLW4WURkujBWKkjkemR1rOLbSv/X9blS02/r+tjmtI0a/12/FlpUHn3BUtt3qvA75YgCqbo0cjJIpVlJDA9Qa9EePSPDkOr38eoSWqaq0DacbWJJZUgY+a3yF1+UMoQnPVSKvXniW0sJY5fD+qxwR3eupcy7JFVxFJFE7hucqu7IIzjK4OcU7f153X+f4Cut/u9DyutSbQ3i8OR6zHe208DTi3eKPzBJE5UthtygdB1BNelaLNpFpqBKa1CNPnu73z7YalBBbohdlRWhwWm3LghsgAY5wK5CKwZvhsbb7Vp/nvqKXQh+3wmTy/KIzsDbs5IG3G72oWq+78wej+85+917V9Rtlt9Q1W9u4FIZYp7h3UEcA4JxUWnabdateraafF5s7KzBNwXIVSx5JA6AmvYtV1G2srm9truSS60ZL21+02zTwtFYQxsFwIkkZ9pfblSi+4zWZpOsm2vIT4p1yxv7gX1xJbub1JlSH7NKrDep+RXYxgJkHjgCnZWb9fyX9fIfWz/rf+vmeY3emXdjaWlzdxeVHeIZIMuu5lBxu25yAT0JAz2zS6bpV7q0zxWEO8xoXkdnVEjUfxM7EKo9yRzxXbJNaXerateQ6pamPWtJkjtY57iONoHBT9w+SAmAuFJwCAMdxWjdX9vDoP9k6de6LcakmnWa755YZIW8t5Q6BpP3e4bgeeoyR2NL+vz/Kwv6/L/M81uNPubXzDJGGjjkEbTROJI9xGQA6kqTgHoe1Vq75dQ0nTfEes3YuLVtGZEin0y25S+lKDcqL/CgkDESfw8bc5ArhbhonuZXt4zFEzkpGW3FFzwM8ZwO9A1qrl1/EWtS6YNNk1i/exChBatdOYto6DZnGBgcYpIdHnl0C51gyRR21vMkADk7pZGBO1QBzgAk5xxQdIx4dXVv7QsTmfyfsYm/0gcZ37Mfd7Zz1rptajh0/UvD3hKWdLWKxkjk1Cd0DKtxMVLswPDBE2ryP4SK451IwajS3bd9O2/6L5la2u9v6X9ehxNaF5otxY6JpuqSvEYNR83ylUncvlsFbcMY6njBNewSalpAit1vtagubyLU52sLjUNXgvTDmCQRS5RQsSeYqHbyFOCccVyHirxBqthoPhvyvE73GsQx3K3U9nqRlkQO6MqNIrHPGO5HHHSuSnj6lapGMYW17+UvLulr8iuVJXucZrGmf2Rqcln9ts77YFPn2UvmRNkA8NgZxnB96o19A/wDCRWumeLr5bvVLa3ae6tPPE8yAvH9hYnO7tu2Z9yB3rA0XVLeW8j1Z/EhS5lmsPtaJq0NpuVLSIvJKzKzTDfuUxjuG7msaWZ1HC8qfRO99728gcVZPueUf2Xef2N/avk/6F9o+zebuH+s27tuM56c5xiqqsVYMpIIOQQeleo3Wv311oOs6b4d8Rray22uyPbp/ai2y/YyrBViZnVSmR91T6HFZ3iDxteWWlaPpulXlvNZSeH0tbqBSGXe6lW37T99cAjP3fTk56qeLrTduRXv3tZWvroJxX9etjmNZk1K80nTtQ1TXl1Mz+YEt3vGmmttrYO9W+5uwCPUVi16Pqx1a58OeFWGv2L6rp88++Z9bgd4Nzp5Z3eYTgAds4A5xUfxE1Sx1LSYjouoWbQR38n263gURtcXW3m5CbjmNgCAf4cnn5qdLGSlKMWr3cte1m7dOq27it/X3/wBf1r55RRRXpEhRRRQAUUUUAT2d9d6dcrcafdTWs6ghZYJCjDPXkc0l1d3N9dPc3txLczyfflmcuzcY5J5PAqGigAooooAKKKKACiiigCWG5ntxILeaSISoY5Ajld6nqpx1HA49qEuZ47eW3jmkWGYgyRq5CuR0yOhxk4qKigAooooAns7K61C5FvYW011MwJEUMZdiB14HNNtraa7vIrW3TdNNII0TIGWJwBz05rrvBmNL0PVNbN/b6fOXis7Wa4WQruLeY/3FY52oB0/irql0dbbXorrwxZ216txrCTXVy9ukot4GSOVMFx+7XEj/ADYByoGcjFVbX+v66k30/roeZ2+g6ldXl9awW2+bT45JblfMUeWqHDHJODg+mazq9BsJxe+JPFNhd2tjNBDb380ZeyiMiuCxDeZt35BPHNXrLw3Gng+c3ttFLCdEa7jlj0xFRJMb1xdFt7SDHKgY6joKiOsU/L/P/IvS9vP/AC/zPMKK9E1TTAniO0tP7Mht/CJmt1GorZIVkjO0lmudu7LEnPzDHTtWsmnWR1y2t9S8PTCQXNwVkn0WOzt2jEEh8sFHYSYZVKt1wCc09OXmf9WEtTyWivStOtrLVtN0zU5NMsX1SSzvjb20VukcdxNG6+WDGAFchWbjGW2jOaztL0y7165vtA1vToNO1WeOOezd7BLVlCMd4wqrwULn32Cj+v6/MVzhquR6RqUzWyxafdObsE24WFj5wHXZx82PapvEN3Z3viC7m0y3jt7PfsgjjXaNigKDj1IGT7k16Lp2uW+n3HhHT3gka88i1NpNgYjE0uJgfqoAHHc046peYNtOx5RUs1tPbeX9ohki81BJH5iFd6Howz1B9a9M0/w8tv4SllurWGaMaKbyKRdKj8tZcb1/0lm3u/HKY24yMYFQ6m+oarceFZjp1vcaTNBZpcXEemxbFfzdrR+YE+QZ42Agc4xzQ1Z28/8AP/IL6X/rp/meaAZOBVzVtJvdD1KTT9Uh8i6ixvj3q2MgEcqSOhFdNrOqPd67daVo3h2y+1W+oubQWtijt5ahlMZTYfMzw2WzjBwMYxtXuhPafES+F3oUUOiy2oa4eW0EccUYhXe8ZC4Vw3Tb1Y471Kd1cb0dv63PM6ltreW8u4ba3XfNM6xxrkDLE4AyeOprc8a2q2mvJHbW0EFibaNrJoFGJoSPlkJwCzE5yTznI6AVv+CbCc6bpd3o9hBdTtqwW/nlgSU20ShSvLgiMHLncMEkAZ4xVRV/6+Qm7L+u1zhLm3ltLqW2uF2TQuY5FyDhgcEZHvUVew6V4bS71aR7u2iubO+vb3zdmmJKFAdlHmXLMGibP3Qg9O5rltc1Ky0nTtLsk0TT5Eu9DRpZfIUSmZ1IWTfgkFSAcDGec9sQm7L+u/8AkN72RylzpF9Z6XZ6jcQbLW+3/Z5N6nfsOG4ByMH1xVSON5ZFjiRndyFVVGSxPQAV2PijR9RTwZol/eaK1pMpnW6lTTxb4G5BHv2qBz2J61U+H0kcfiZx5qQ3clpPHZSO20LcMhCcnoew9yKvuK+iMK/0rUNKkWPVLG5sncblW4haMsPUBgM1Ur03TNCg0jTFvdR0i683+zr6ZrDV/nQSx+TiRV2KV3Z9zgAZqhoCHxKLrULXTbWGeztoYJUtNJiu3ndnb94sB2xrxgMe2OMZNK2tgutzjdM0m81eeSKwjV2ijMsjSSrGiIOrMzEADkdT3qtNE0E8kTlC0bFSUcOpIOOGBII9wcGvUtQGnaRdeNbW20SxaOC2gkIJlUPveJipCSABQTkBcdByRXlsrrJM7pGsSsxIjQnag9Bkk4HuSaOw1s352/L/ADHPbTxW8U8sMiQzZ8qRkIWTBwcHvg9cVFXfXt1qep/D3RpLSzt57aMXMd9LBpkL/ZhuBySE/dnac5GCeuc81talpmmQ36Wy6LNLZNe2i2N0mkRpbqnmKMm5VyZg6E/ezkkcDpTsK+lzyeiu8n1ayn8Ytpo8KW11Ha6qzRQ2FuokaJNymPAXMnOH+Y4+UjoeNODQIl1gzXwSS6l02WSysX0WGG4VxJtJa1BVHbbuZck5xnB24pLVX/ra4+tmecw6ddXGn3N9FFutrVkWZ9wG0uSF4zk5welVq9W8qddI1xdE8O3BvvJshLb3WkBDJJulBlW2BZQNuO2M5OKZe+HbbULhF0fS7WV4dbi+2rbIrLCpgjMiseixhxIMH5QQRR1fy/T/ADE3ZHmP2af7J9q8mT7Pv8vzth2b8Z256Zxzioq9PvZtRm8PaxZaDptrcvZa9KhtoNLhlMcO1grFNh7gjdjPGM1w5ivP+EMWU6daix+3lRf7R5xk8v8A1Wc52Y5xjGe9Ja3+X42GZFFFFMAooooAKKKKANmTxZrMto9u9zFiSAW7yi2iEzxAAbDKF3kYAHLcjisaiigAooooAKKKKAClR2jdXRirKchgcEH1pKKANi/8VavqVtPBdTw7Llg9wYbWKJpyDkb2RQW555J5561j0UUAFFFFABRRRQAUUUUAFW9O1OfS5mkto7V2ZdpFzaRXAx7CRWAPuOaqUUpRUlZoCzqOo3erajPf6lO9xdXD75JH6sf6DsAOAOBVaiihJRVlsAUUUUwCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoruPB3lWvg3WdRS+uLCaG4gSa5s4y1wsJz8qcjbuYLkkgYHfgHL8fhP+E4vmS3FuZFhleLj5XaJGfOAOdxOeBzmm1YSdylZ+IG0/TJ7Wy0+zhmuYGt5bweY0rRscsMM5QZAxkKDisiuz0G+eX4c+JrQRQxxwwQEsiYaRjOOWbqcDgDoB0GSSeMoe41sFFdl4ovX1DwD4bneKGEGa7CxQJsSNQyYAH9TknqSSSarfDq2S68YxqfLE6QTSWxkQuFlVCUbaMkkHnA9KA6J/wBb2OWrS03WpNJtrhbS1txczo0YvGDGWJGGGCfNtGRkZ255ODXQeNLiO+8N+H7oS3d3LuuYje3wAmuEUoQTySFDM4ALN069hN4O8q18G6zqKX1xYTQ3ECTXNnGWuFhOflTkbdzBckkDA78AltxX1Rw9FdJ4/Cf8JxfMluLcyLDK8XHyu0SM+cAc7ic8DnNdx4ZFvaWfhO3WadrTUYZVmsYI/wB1cuWZZXnY8EKmMDDH5f4eDRbQLnkdFTWixvewLcHETSKHOcYXPP6V6h4wZF0zxXp0stxcrZNbvBG0ey3sR5irGkIOTkxs2SAo4P3utFtAvqeU0V1Pw6tkuvGManyxOkE0lsZELhZVQlG2jJJB5wPSrPjS4jvvDfh+6Et3dy7rmI3t8AJrhFKEE8khQzOACzdOvYDQXONooopDCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAs2OpX2lzmfTLy4s5Su0yW8rRsR6ZBHHAqCSV5pXlmdpJHYs7scliepJ7mup+HQsZvGFnaX+nQ3puJNoNx8yRrtYn5OjE8cnIHPGcEctIMSuBwNxoYLqaFp4j1ywtVtrHWdQtrdc7Yobp0UZOTgA461m16h4e0fS47Xw3bXiWDW+rQuZ0ktxLc3MjsUwjYzEqAKc7l74DHIry+m1ZiTTRpXfiPW7+0a1vtZ1C5t2xuhmundDg5GQTjqBVCGaS3mSaCRopY2DI6MQykcggjoa6rxKthL4K0C8sNNgsfNmukIjG52VWUDe55Y8nk8cnAA4rmrGaC3vElu7UXcSgkws5QMcHGSOcZwSBgkcZHWjqPoLfaje6ncefqV5cXc20L5lxK0jY9Mk5xRY6lfaXOZ9MvLizlK7TJbytGxHpkEccCt34g28Fr40uY7S3htovKhYRwxhFBMSk4A4HJNS/DoWM3jCztL/Tob03Em0G4+ZI12sT8nRieOTkDnjOCEgeiuctJK80ryzO0kjsWd2OSxPUk9zVmDVtRtbGWytr+6htJs+bBHMyxyZGDlQcHj1qtIMSuBwNxr07w9o+lx2vhu2vEsGt9WhczpJbiW5uZHYphGxmJUAU53L3wGORQloEtJWPL6t3GrajeWcNpd391PbQY8qCWZmSPAwNqk4HHHFRWkS3F7BC7bVkkVS3oCcV6L4p0zTbfTPElmItPEumtE1pFaW4D2yCRY/3swALs4fOCXwQclSMU7aCvqebwzSW8yTQSNFLGwZHRiGUjkEEdDU19qN7qdx5+pXlxdzbQvmXErSNj0yTnFbfgTTYNT8ULHcxRTiGCWdIZ2CxyOiEqGJ425wT2wOeKs+L4LJtC0W9tWtZrmRriG4ubO1FvFMUKMCqAL08wru2rnHsDQ0FzkqKKKQwooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKANHR9ev9AuTcaW8Mc+QRJJbRysmM/dLqdvU9MZqpd3cl9dPcTiJZHxkQwpEvTHCoAo/AV0HgLT9I1TxPbWOs281z9ok2Rxq+xMbWJLEfNxgYAx1yTxg83IAsjAdASBQwXU1bPxTrFhp6WVrebIY9/lExIzw7xhvLcgsme+0isiuo0rTtHu/A+tXJtppNRs4on855MLGWl24VR1+XqWzyeAMZPOW0JubuGAHaZXVAfTJxTs3K3UTaUbs0r7xRqeo6WmnXRtDaxkmOOOwgj8skgkqVQFckDODz3rOtLqSyuknhWJnTOBNCkq9Mcq4Kn8RXeeJPCumafpOuIllFZ3GltEbZhcs89wm9Y3eVNxVQxcMvCE9gRmub8GaNFrniNLe5hkuIooZJ2t4jhp9ikhAe2TgULyB6aMpazr2oa/cJPqjwySoMCSO2jiYjAAyUUbsAADOcdqNH16/0C5Nxpbwxz5BEkltHKyYz90up29T0xmtjxZpFlZaVpN7aw2tvcXBmiuoLK4aaFHQoRtYs2TtkGcMwyOxyKZ4C0/SNU8T21jrNvNc/aJNkcavsTG1iSxHzcYGAMdck8YID21Ofu7uS+unuJxEsj4yIYUiXpjhUAUfgK0bPxTrFhp6WVrebIY9/lExIzw7xhvLcgsme+0isqQBZGA6AkCvRdB8H6bNZaJHqFpC0erRO0l9NcMkiOzFI0gQN85BALZVh83JApJXQ3pK3U84rWv8AxRrGp2bWt9d+ZG+zzWESK82wYXzHADPjtuJrJrqPEWnaPD4T0S/0e2miNzLcJJJPJueUIVAJA+Vep4HTpk4yToC3MDT9Qu9K1CG90+dre5hbdHInUH+o7EHgipdS1m+1cw/bpUKQKViiiiSKOME5OEQBQSeScc1DYmzF4h1ITNbAEssBAduDgAngZOBnBwOcHpWx4406z0rxZcWumwfZ7ZY4nWMOzbS0aseWJPUmhgjn6KKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA1vD2vv4c1Jb+3sLS6uYzmJ7nzP3ZwQcBHUHOe+aoXlwl1dyTRW0VqrnIhhLlV47b2Y+/JPWt7w14bj1XTbq+ntr2+Mc0dvBZ2LBZJZGyxJYqwCqqk9PxFUPFGkQ6D4kutPtZ/tEEex4pCQdyOgdeRweGAyOD1pu4k+xPpfif+y9HudOXR9PuI7tQtw8xn3ShW3LnbIAMH0A981hqxRgykhgcgjtXRadoWmXfg7VdSN3O9/ZRxyCFFCxx7pdmGJGWJHPGAOOTkgc/BC1xcRwx43yMEXPqTijXm0DRRu9jc1Txfe6ra3UctrZwzX3l/bLmGNhJc7ORuyxUcgE7QuSOaztH1e60PVIr+xKebHkFZF3I6kYZWHcEEg11WueCbPTtF1W4t1vopNLdEE9yy+XffOI5DGm0MArsOcsOxwTXPeGdHj1vWltrmR47aOJ552jGX2IpYhR/eOMD60LyGyLVdbk1SC1tltbaztbQN5NvbBtqljlmyzMxJwOp7DGKk8Pa+/hzUlv7ewtLq5jOYnufM/dnBBwEdQc575q94p8ORaLYaZeRQ3Vm16JFlsryRXlgdCp5IVcZV1OCoI/EUzwXoem6/r0Njql3NCJnCRxW6je5wxJ3HhQMehJyAB1IFcTtYxby4S6u5JoraK1VzkQwlyq8dt7MffknrWzYeMb6wsbSFbazmmsA4sruWNjLbB+oXDBTgkkbg2CeKwXG2RlHQEiu50HwHbalaaalyt6Z9RhaY3cTKtvZgkrEH3L85ZlPAZTyAMmkthvSVmcJW5qPib+0NCg0r+x9PtobZmaB4TPvjLEFsbpCDnHcH2xWNBC1xcRwx43yMEXPqTiu11zwTZ6douq3Fut9FJpbognuWXy775xHIY02hgFdhzlh2OCaethX1ONtJ47e6SWa1iu0XOYZi4VuO+xlb34IrR8ReIH8R3y3lxY2lrPtCu1t5n7wABVyHdhwF7Y980eGdHj1vWltrmR47aOJ552jGX2IpYhR/eOMD61d8U+HItFsNMvIobqza9Eiy2V5IrywOhU8kKuMq6nBUEfiKGgT1ObooopDCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA3NF8RLp2j6hpF9bS3Nhf7GdYLjyXVkOQQ21gR6gqe3TFUtb1Vta1ia+aFIA4VUiQkhEVQijJ64VRz3q1o+hRX+nXWo391Ja2lvJHCvkwCaSWVzwqoWXPAJJz271D4i0SXw74gutKnkErW7DDhSu5WUMpweQcEZHY035iVi/o+v6XpugX+nXGl3dw2oIiTyx3yxgBX3LtUxNjsDkn8K5+ORoZUkjO10YMp9CK3LPw0l14Vv9YOowiS0RJBaIu9irSbPmOcJzyBySByACCcGjVPzHa6Oq1zxmmsQ6jIunNDfaosS3k73JkTCEHEakAoCyqcFmwBgYrK8Na9P4a12HUrdPM2BkePeV3owwwDDlTg8EdDg1Z1zw0mjaLp18mow3pvHlRhApKRtGQCA+fn+91AA44JGDWTY2ovbxIGuIbZWBLTTttRAASScAk9OgBJPABNC3sgequX9Y1mC/sbKxsLN7W0szI6iWbzZHeQgszMFUdFUAADp3qTwrrdp4d1qLU7mxmvJrdt0Kx3IiUHBB3ZRievbH41F4m0RfD2vzaclz9qWNEcSmPZuDIG+7k46+tP0fQor/TrrUb+6ktbS3kjhXyYBNJLK54VULLngEk57d6FcTt1M29ktpbyR7KGWGBjlY5pRIw45ywVQec9hXS2Hjf7Lp+lJdWDXN3o3mfYZftJWNd3ILx7TuKnkEFc8A5xWN4i0SXw74gutKnkErW7DDhSu5WUMpweQcEZHY1bs/DSXXhW/wBYOowiS0RJBaIu9irSbPmOcJzyBySByACCUth7sw45GhlSSM7XRgyn0IrqNc8ZprEOoyLpzQ32qLEt5O9yZEwhBxGpAKAsqnBZsAYGK5eONppUjjG53YKo9Sa6jVPBa2Ol6pcW2oNczaPJFHfKbbbEGc7SI5Nx3FWIByF9afQWlzK8Na9P4a12HUrdPM2BkePeV3owwwDDlTg8EdDg07WNZgv7GysbCze1tLMyOolm82R3kILMzBVHRVAAA6d6i0HR31zVktBMtvGEaWadxkRRqCzNjvgDp3NWdc8PrpWm6ZqNrcSz2epJIYjPb+TICjYbK7mGOQQQxzn2oYaGJRRRSGFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB1HhjxRHpGjajpk895ZfanSWG9sUVpYXXIIALLwVJGQwI9wSKyvEWpw6vrs13axNDAVjiiRyCwREVFJx3IUE/WjStCl1O1urt7m3srO12iW5ud+0MxwqgIrMSeTwOgOcVDq+lXOiatPp18E86BgCUbcrAgEMD6EEEfWm7iVrG1oWp6HZeGdVsL651BbjUo40bybNHWLZJuBBMqlsgegx71zFdBp3g6+1K0tZI7i1hnvlkaztZmYSXIQclcKVGSCBuZckcVgKpZgqgkk4AA60dR9DptX1PRLjwfp2l2NzqD3GnvK6mazREl8xlJBIlYrgA9jn2rnrRbZ7pFvpZYYDne8MQkYcdlLKDz7itnUvB97ptjdzyXVpNJYGMXttE7GS2L9N2VCnng7S2CazdI0q51vVIrGyC+ZJklnOFRQMszHsAASaXUOljU8Z6tpmua5/aOlvdnzI0SSO5gWPZsRVGCrtnOCe2Pep/DHiiPSNG1HTJ57yy+1OksN7YorSwuuQQAWXgqSMhgR7gkVlatoculW1ndC5t7y0vVYwXFuW2sVOGXDqrAjjqO4qXw74ZvvE199nsGhiG4KZbiTagYgkL3JJ2ngAngnoCQ1fUWmj7EXiLU4dX12a7tYmhgKxxRI5BYIiKik47kKCfrWpoWp6HZeGdVsL651BbjUo40bybNHWLZJuBBMqlsgegx71zLDaxU9QcVv6d4OvtStLWSO4tYZ75ZGs7WZmElyEHJXClRkggbmXJHFLoPZmHBM1vcRzJjdG4dc+oOa7PxJ41s9Ys9SMDX7S6isai1nCrDZDesknllT8250HO1T3OTXEqpZgqgkk4AA61val4PvdNsbueS6tJpLAxi9tonYyWxfpuyoU88HaWwTT1sLS5H4S15PDviGK+nhM0BR4pkUKW2OpBIDAgkZzg8HGKl8S69DqtpZW0Fxe3rW7yyy3l8AJJnfaMbQzYCrGoHzH8KzNI0q51vVIrGyC+ZJklnOFRQMszHsAASam1bQ5dKtrO6Fzb3lpeqxguLcttYqcMuHVWBHHUdxQ7gZlFFFIYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHXeFNds7Pw/qOm3E9raXTzRXNpc3lsZ4kdcqQVCt82GOCVYdehwayvFupw6x4muby1llniKxxrNMDul2Rqhc555K5555qppujX2recbKNCluoaWWaZIo0BOBl3IUEnoM5NQ39jdaZfzWV/C0FxC2ySNuoP9fr3ptsSSWx3+h+MNNtdP0O4a8isp9MjaK7iFqzXFyisXRY5ACFUk4IynfO4YFee2s32a7hnA3GJ1fHrg5rRsfDGr6jYrd2dqHifeY1MyK82wZfy0JDPgddoNZNPmalf5i5Vy26bHoXibxTpl5putjT76JodVMbQ2MFq0bxOZFkkeZiMOwK4BDP9442jiuc8F6tZaP4kSbVV3Wc0MlvMSpYKrqVyQCCQM8gHOM45qhqGg6npVjaXeo2j28N4GMBkIDNjGflzkdQeQMggiq9jY3Op30NnYQtNcTNtRF7n+g9zwKSutBvXU6DxZrMWoaXpdil/Ffy2jTO8ltAYYED7AqRoVXAAjyflAyx68mo/AdzYad4rtNT1TUYbOC0fcRJHIzSZUj5QiN0464rJ1LRr7SPIN7EgjuFLwyxTJLHIAcHDoSpwRgjPFGm6Nfat5xso0KW6hpZZpkijQE4GXchQSegzk0K9waVrEN9FHDfSpBcxXUYORNCHCtnngOqt7cgdK7/AEPxhptrp+h3DXkVlPpkbRXcQtWa4uUVi6LHIAQqknBGU753DArgL+xutMv5rK/haC4hbZJG3UH+v171esfDGr6jYrd2dqHifeY1MyK82wZfy0JDPgddoNJPTQb1ldmdazfZruGcDcYnV8euDmu98TeKdMvNN1saffRNDqpjaGxgtWjeJzIskjzMRh2BXAIZ/vHG0cV57WtfeGNX06xa8u7UJFHsEoWZGeHeMr5iAlkyOm4Cnd2FZXLPgvVrLR/EiTaqu6zmhkt5iVLBVdSuSAQSBnkA5xnHNWfFmsxahpel2KX8V/LaNM7yW0BhgQPsCpGhVcACPJ+UDLHrya5y3tp7y4S3tIZJ5pDtSONSzMfQAdan1TSrzRdRksdSh8m5jClkDq2AQCOVJHQihtgktSnRRRSGFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB2fhK8sX8K6pp0yWct59ohuYLe/ufIhmC5UhmyoON2dpYZ98EHK8a30Wo+Lru4guI7ldkMbTRLhHZIkRioAAxuU4wMY6VlWOm32qTmDTLO4vJQu4x28TSMB64APHIqGWKSCZ4p42jkjYq6OMMpHBBB6Gm3qJKx6doGtaZFZ+GtQSXT0bTonhvZLq4Hm26qxb91CSNxcNjcA55x8pGa8uHXirlvpGpXdjLe2un3U1rDnzZ44WaOPAycsBgYHPNU6G9bglZHaeI4pZPAekC4v7K5u7ae5kulTUYZpP3jrtOFclieScZx3rP8B3tjZeKF/tSUQ29xby25lL7AhdCoJbB2jJxuxxnNYMtndQ2sNzNbTRwXGfJleMhZMHB2nocHrimQwyXEyQwRtLLIwVERSWYngAAdTQtwey/rrc6rxfeRPoejWC/2fHNbNcObbT5RLHAjlAo8wM25iUZjlifm7cATeEryxfwrqmnTJZy3n2iG5gt7+58iGYLlSGbKg43Z2lhn3wQeVvtOvdMuBBqVncWcxXcI7iJo2I9cEdODRY6bfapOYNMs7i8lC7jHbxNIwHrgA8cihMLK9zV8a30Wo+Lru4guI7ldkMbTRLhHZIkRioAAxuU4wMY6V2ega1pkVn4a1BJdPRtOieG9kurgebbqrFv3UJI3Fw2NwDnnHykZrzGWKSCZ4p42jkjYq6OMMpHBBB6GrNvpGpXdjLe2un3U1rDnzZ44WaOPAycsBgYHPNF9AsrkNpKsF7BM43LHIrMPUA5r0XxVqmnHTvEj2U+nm31UxPbvFcCa4uXMiSEyDOYwgVhtKqOcfMea80qaWzuobWG5mtpo4LjPkyvGQsmDg7T0OD1xRfQLaj9PjuZb1YrKZYJZFZd73CwrgqQwLsQACMjk85x3roviKqSeKnu7e5tLmCaKJUe2uo5eUiRWyEYlefXGe2a5eGGS4mSGCNpZZGCoiKSzE8AADqamvtOvdMuBBqVncWcxXcI7iJo2I9cEdODSGVqKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA7jwdBb33g3WbRba6vrkXEEz2VkwWa5iGRjOCdoYgnCk9OmcjK8fz/afG97ITGZNsKy+U25RIIUDgHJzhgw6npXOUU27iSset+Gvs62XhPUre3luILSGWO4mEgS2sTuJmaXHO5kPGSo6fezivJB14ooob1uCVlY77xdaa7P4A0O71ezvRLFNcm4eW3ZBEGZAmRgBR0CjgdhWV8Omg/4TGOOebyHmgmiglGNyyshC7ckDdk4GSOSK5aikHRLt/nc7LxmBbeG9AsWsZdOe3a5K2lzIWnSMlMM+QMbmEhGFAx27mbwdBb33g3WbRba6vrkXEEz2VkwWa5iGRjOCdoYgnCk9Omcjh6KdwsrnR+P5/tPje9kJjMm2FZfKbcokEKBwDk5wwYdT0ruPDX2dbLwnqVvby3EFpDLHcTCQJbWJ3EzNLjncyHjJUdPvZxXklFF9AtqA68V33i6012fwBod3q9neiWKa5Nw8tuyCIMyBMjACjoFHA7CuBopdB9bnU/DpoP+Exjjnm8h5oJooJRjcsrIQu3JA3ZOBkjkirPjMC28N6BYtYy6c9u1yVtLmQtOkZKYZ8gY3MJCMKBjt3PG0U2xWCiiikMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKK9B1L4M+IdL0q71C4vNMaK0ged1SWQsVVSxAyg5wK8+oAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAP/2Q==)

1. **Studi Kasus 2:**



/\*

NAMA    : SURIADI VAJRAKARNA

NPM     : 140810180038

KELAS   : B

TANGGAL : 5 APRIL 2020

STUDI KASUS 2 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA

\*/

#include "graph.hpp"

#include <iostream>

void print(int data) { std::cout << data << ' '; }

int main(int argc, char const \*\*argv)

{

    const size\_t graph\_size = 8;

    Analgo::Graph<int> g(graph\_size);

    g.add\_edge(1, 2);

    g.add\_edge(1, 3);

    g.add\_edge(2, 3);

    g.add\_edge(2, 4);

    g.add\_edge(2, 5);

    g.add\_edge(3, 5);

    g.add\_edge(3, 7);

    g.add\_edge(3, 8);

    g.add\_edge(4, 5);

    g.add\_edge(5, 6);

    g.add\_edge(7, 8);

    try

    {

        std::cout << "Adjacency List dari Graf tersebut:\n";

        for (const auto &node1 : g)

        {

            std::cout << node1.first << "\t";

            for (const auto &node2 : g)

            {

                if (g.is\_edge(node1.first, node2.first))

                {

                    std::cout << node2.first << " -> ";

                }

            }

            std::cout << '\n';

        }

    }

    catch (const std::exception &e)

    {

        std::cerr << e.what() << '\n';

        return EXIT\_FAILURE;

    }

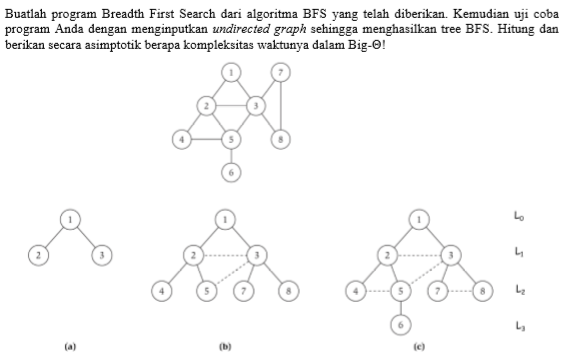
    return EXIT\_SUCCESS;

}

![A screenshot of a cell phone

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4REGRXhpZgAATU0AKgAAAAgABAE7AAIAAAAUAAAISodpAAQAAAABAAAIXpydAAEAAAAoAAAQ1uocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFN1cmlhZGkgVmFqcmFrYXJ1bmEAAAWQAwACAAAAFAAAEKyQBAACAAAAFAAAEMCSkQACAAAAAzQ0AACSkgACAAAAAzQ0AADqHAAHAAAIDAAACKAAAAAAHOoAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyMDIwOjA0OjA3IDE1OjI4OjI2ADIwMjA6MDQ6MDcgMTU6Mjg6MjYAAABTAHUAcgBpAGEAZABpACAAVgBhAGoAcgBhAGsAYQByAHUAbgBhAAAA/+ELJmh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjAtMDQtMDdUMTU6Mjg6MjYuNDQxPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPlN1cmlhZGkgVmFqcmFrYXJ1bmE8L3JkZjpsaT48L3JkZjpTZXE+DQoJCQk8L2RjOmNyZWF0b3I+PC9yZGY6RGVzY3JpcHRpb24+PC9yZGY6UkRGPjwveDp4bXBtZXRhPg0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICA8P3hwYWNrZXQgZW5kPSd3Jz8+/9sAQwAHBQUGBQQHBgUGCAcHCAoRCwoJCQoVDxAMERgVGhkYFRgXGx4nIRsdJR0XGCIuIiUoKSssKxogLzMvKjInKisq/9sAQwEHCAgKCQoUCwsUKhwYHCoqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioq/8AAEQgAvgJgAwEiAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A8Vs7Se/voLO0TzJ7iRY40yBuYnAGTwOTWrJ4O1yO4ghWzWdp5jAht7iOZRIBkqzIxCEDk7iOAT0BqLwrNFb+MNHmuJEiijvYWeR2CqoDgkknoK6P/hJnh1aaz8KQwW+b+W7lmv72JkuDtZNuXCIEKs3HJO7rwKemnz/T/MWpyN/ps+mui3D2rlxkfZ7uKcD6mNmx+NWNN8O6jq1nJdWaQCCORYmknuooRvYEhRvYZJwelaHiu30uKHTpbKO0ttQmSQ3trY3PnwR/N8hVgzAEjOVDHGB0zVnRoE1HwDfadFeWUNy+pQyhLq7jg+QI4LfOwyASOmaS2f8AXUOqX9bNnN31jdaZfTWV/A8FxC22SNxyp/z371LNpU8Oi22qbo3t7iV4fkJzG64O1uMAkEEYJ4rpNQv5J/EetPomtWqQDS1t5ZrghftiJFGjogKnLMVyMYOB1ql4W/4mdlqXh1z897GJrTP/AD8R5Kgf7yll/EUPbT+v6QLzKcnhPW4pniksSHjsTqDDzE4gAzv6/p19qL7wrqumtsu47YTblX7PHewyTEtjaPLVy+Tkdq7LWPFttJql7pkJtzCtnNGl4s4IdPsTBYQeh/eMT15JAqzqGqadZ/ECDW5LbRpLCOeFmvre/EtwRsC7vKEx6H0jzx+NNrYm7t8jgb3wzqunuqXMERkaYQGOK5jlZJDnCMqMSpODw2Oh9KlPhTVI9Zn0m5jjgvobdpxCz7vMCrvwhXIJK5I57HnNdlo1hoWlMuo6tc6V9p0+9aeO5tb9ZpLuNY5GU+X5pGS/l4GFPY96xNX120jTw5q+lX013f2LyRyvcRLFIwRw6FlDtwRIy5zztPpSt0/r+rlX3Od03Rr7Vlnezjj8u3UNLLNMkMaZOBl3IUEnoM5PPpVmHwtqc8Es6fYhBFN5DTPqNukZfG7CsXAbg54JrpZILjTdTv4tMi0q70LVFjuktdRu4oFZGyVKlpFYMh3LlT2Oau3Eeix6HqOnaEuk6jBHq/mRQ6hqIhCr5IDMreZGWAbKg85HPPWh3/r5DuuhyEfg/XJbmwgishJJqXmfZNk8ZEoT7xBDYAGOp69qim8NalbWEd5cC0iilhE8avfQLI6HoRGX3nPpjNdxF4rt9Nv/AA9piCz2IlruuI7kMlmfOJmXdk8EBQctwB3zWZrpjvNF0mSyg0W4eHSoI2uZNTVLiFxnKiPzl5Huh603tdd7Epvr2OcuPC+p2hRbsWdvI5UeVNfwJIm4ZG5C+5OD/EBjvU8ngnXI7q1tjBatPeANBGl/AxdSpYNgOflwp+Y8e9a+vGDUNAvb/wAQjS49cDxC3n0+8jla7OcP5iRuyjC87sLzgc5rQtNTsB498KTm8tfJg0iKKV2mUIjiJwVZs4U5IGCR1pPb5/oM5KLwpqk9w0MH2GVkiaZ2j1K3ZERSASzB9q8sOpFUdS0u80i6FvqEPlyMiyKQ6urowyGVlJDA+oJFdjpc9rpUuqve2Gh2yyaTKiW9tqJmS4benykidmB9ACCcH0rG8bXcF/rNtc2NxHJZvZQ/Z4UYE2qAY8lsd1IPXkgg96T6f13/AMv60H/X5f5mXo+iajr9/wDY9ItjcT7S5UMFAA7ksQB1A/EUk+kzwaPb6mWjeCeV4fkJJjdMEq3GASCCME8V2PgDXNF01rC0lvLixuJr0S3krQJ5UiL9xDIZAVUHLH5Tzj0rJ0DytSXWPDsUjSJe5msWdQpM0ZJQYyQC6ll6nkjrTf8AX9f1qJPv/W/9ehjroGpvoserLak2UkwgWXevL+mM5xxjOMZ70++8Natps9/De2hik05Ue6BkU+WHICng85LDpnrXcrqXh+GP+wzqrNHDpKxhRGn2c3C/v9wl8zqXyuNvfGa0/wC0tD1681yPUdUs49t6sBeS4UC4tvtSygqSfmwBIOOxWq5dV8/8iXJ8ra8vx/r7zzebwprUGpWWnzWRW6v1D28ZkT5wffOAfY4I71T07Sr3Vbw2thAZZVVnf5gqoq9WZiQFA9SQK9Gs/EOi6nq+m3zapIZrfXGlzexJb+XDKrEgfvGyqmMcnGNw9awtFtV0+z1vTb6806GfVLYC1lW+hkRijhzG7o5CbhwCxAOOtTrbUq+tvT82YSeFdWeaRFit9kcaytP9shEAUsVH73fs5IIxuzkH0NNvPDeo2FneTXcaxPY3C29xASS8ZYEq3AxtODgg88eorrtYtdLufCcWjaTJpVjqBMN9cQx6iphdv3iMoleRl3KCh2l+hJFUor+zl8bXOmSXqT2OoWUOmS3IyU8xYY0WQZ6gSIDn0+tVbddfz/rYSlom/wDhv63MG18JazeafBeW1tHJHcI7wILmISyhCQ22PdvbGDwB2rGrv5tatPDWk+HfMtvtOs6Wt0gjF0oS2l85wPMjClie4G5c47ioNb8X3Vrp2l6fp91BLZyaIlvcwqQy72Uhi2D99cAjP3fTk5l2vp3t+f8AkVrexyGnWUmpana2MDKstzMsKFyQoLEAZx25p8mmXKa0+lIFkuluDbAK3DPu28E44z64r1aO80yzhME2uR3cEN7YNaPPqkEi4WVN7xwoB5AClgQT0+hrI1rxHDba7p1/4f1ZbTSY9RP2u0t59shk8zMkrBTmVXXo3IA+XjoXZXX9dhX0bPP006X+2BptxJDazef5EjzuAkTbtpLMM4APU1bbw9cfY9Wuorm1mg0qVI5XikLCXexUMhxgjjOTjiupuLvW2+Icdz/wkUUlsLt3hm/tuLaluZASuTJ8oIx8hwTjpxWnpl9oumaj4rbXp7K4try+ingWK4jnDqZpCr7VY7wuQxTrjqOeUtYpv+tV/wAEb3aPLaK9Iid72KOHU9f064vbXXhd3NzLfJteLy0w6HPzDC42rkjgYFdS/iyyivfLg1u0WEXFuw23CY+a9k8w5z/zzxn/AGT6GnbVLu/0v/wBX1f9dbHjd/pU+n21lcSNHJDfQ+dE8ZJH3ipU5A+YEcj6U680PUdP0y01C7tjHa3oJgk3Kd2PYHI698Z7VtaJENf8L3OjNIiXFjML23d2AAjOFmGTwABtf/gJrrv7b8O3GsT28WqP5Vlf2Mlos8aRwxrDIsLbJPMO4FGLdBwCafLd2XX8v6/IL9f6v0PKKuaVp41TUYrP7XBaNKQqPOH2licAfIrHnPpivQrbV7fW2RNW1yKFrfXJGtpTJGGhhELYEe75UUsqAHG0HB61HrBh1C88KsNQtpbq1kd7uS71mCeSNBKjKGm+UNwWIAz3xnBqY628wlonY89v7OTTtSubKcq0ltM8LlDlSVJBxntxVevU9NgFtfeLLyPUo7Rrm9jNvf2N5AWWNpZH2h2lVBuCjK793TKkVPqesXtxqzy6dqFvod+dUE1+s99HG0kBRPJYvu2yoEzlFJHzdDkURV0r/wBbf5/gN6P+vM86tvDWrXi2r29ruS7ikmidpUVdkZIdmJOFAIPLYrKr1HWdT0fV/DS6RpN/a2b3DXUsDGRY18sXMjrBJn/VhlKsucDKgHtius1nJbPqI1PTxF/wi/2NY2ukErTCPBTy87gcjuADxjOaHpf1f6/5WD+vy/zOD1bSp9HvRb3DRybokmjliJKSI6hgwJAOMH06g07VdD1HRHgXVLYwG4iEseWVtyn6E4PseRXT6HbWniTSdJOozxxDRrpYrt5XCg2jtuBJJHRty/8AA1roLDxJomoahDftqbNPBqFzIv2+GOELHNBIdoG9gyhkUc4+8Bjmh6Xfa/8AX9dhJ3/r+v6aPKa1NA0N/EOoixt722trh/8AVLceZ+8OCcAqrY4HfFdtp2qR6vY6TeajrpTVYLW9zILyKGdzvXy4/Okz5WQzYPpkDrVoyWc3xJ0XUU1GwMdtYxi9uJtThdjJiRfmkJXzGGFBIHoehBo1X9ev+QX3seV0V6n4MRtC8PwO159muBq++Q2d5bBbtEjXERkeVVZcv/CXAycjOKfFqzx6pYySajbadYwyXP8Ab2ny3KhnmLuZA0ZOZtylVUgN07Yptf18v6/phf8Ar52PPm8Naskckj2u2OK0S9d2lQKsT/dOc4yc8L94+lZaqWYKoJJOAAOteheJb6x1jwvY2NlqFtFcWFhbTyp5yhbrEQUrnP8ArE5wncM2Bnrsf2jpVjrut6zc6lp81neXdjNDHFcq8jokqMzbASwx6EA8Hinb3+Xpdr8f6+4G7Rutzzi78OapZSwwz26/aJiFW1jmSScEjOGiUl1OOzAVl13etkXEaLJd6JBqr6tus7uwlij2QYOXd4zwN2wgud/3qxPF+qabqupRyadGZJkXbdX+3yxevx+88vHyZ59z1IBzUFaXOfor0L4cKsPh3X7lb6TTZ2ktYY76CSKKSMbmdlDyyRrhggBAfPA4IFbU2uaU/jPxhca/ZXFneXWlSrh9Uh2zRlY/LRMIw3sgUghmHXANebUx0o1Z04wvy+evTp8/w8xqN7eZ5HRXe6JqfheLwL4mge0vYjM1r/o8mqReZOBIxHl/uRjaOW4bP+zXBV2UqsqkpJxas7euif6/10TQUVctpLFbm3MkUgCuplZ5Aykd/lCg/qanv5ZWsSl5cpcTGbdGVkEm1cHPIPAJxx7VuIzKK6PwSIRrFxLLqDWMkVpI0LJcx27yPwAizSAiMnJ+brgEd67XWL6CWOT+wdbtLXWpLSzSa9fU4/MZAZBIpuBt3MCIy2MFgB1FNqyuJbnAS+GZ7fxSdCu76xtpl+9cTzFIV+TfyxHHHHTrUcGhPc6BearDe2rLZFfOt/3nmKGfap+7tIJ9GrvrDX7iX41rFp+ryy6dczqzrBcMYZj5ABJAOD07+lc5odmZPCHiBPtenxvfCEW8cuoQRs+ybLfKzgjgZ5xntmp6XGc5/Z3/ABI/7S+2Wn+v8j7L5v7/AO7nfsx9ztnPWqVdpMLu5+FyWt1qdrNJDercQ28mpws8cAiYYVC+RyfuAZ9qua9qk8mgrenVpNMvbRoFtrKx1hLiByEKs0ccRzBgDOSSPmI71T0b+QLVHEajp11pV/JZX8XlXEWN6bg2MgEcgkdCKrV2fxC1G91TUmu01hb3SZjG9vCL9ZNjeWM5h3bkOdwOVHP1rP8ABIhGsXEsuoNYyRWkjQslzHbvI/ACLNICIycn5uuAR3pLzA5ytmXwzPb+KToV3fWNtMv3rieYpCvyb+WI4446da7/AFi+gljk/sHW7S11qS0s0mvX1OPzGQGQSKbgbdzAiMtjBYAdRUVhr9xL8a1i0/V5ZdOuZ1Z1guGMMx8gAkgHB6d/Sj7VhN6XOBg0J7nQLzVYb21ZbIr51v8AvPMUM+1T93aQT6NVf+zv+JH/AGl9stP9f5H2Xzf3/wB3O/Zj7nbOetdHodmZPCHiBPtenxvfCEW8cuoQRs+ybLfKzgjgZ5xntmpJhd3PwuS1utTtZpIb1biG3k1OFnjgETDCoXyOT9wDPtR0G9JWOLqzqOnXWlX8llfxeVcRY3puDYyARyCR0Irt9e1SeTQVvTq0mmXto0C21lY6wlxA5CFWaOOI5gwBnJJHzEd6p+Pr+91fVvtCayl3pNw0TW8X9oK4jYxjJMW4shB3ZJUd/WjW9kHQ4uiuw8P29v4f8UahavqWmy3i2TrYX0U4aBZyAVIkIABxuAJwAe9a2kS+IRoPiF31iN9SBtAl4+qxPsUmQYExchTgngMDz70f1+Ng9TzmivYLjUZ5LVrzQdWtLWM62gubz7SkCybYIfNILYLKXViQoO7rg15brk1rceINRm08AWkl1K8AC7QELkrx24xxQ97f1sn+oJ3X9eZUhhluJ0ht43llkYKkaKWZieAAB1NXLrQtXsriCC80q9t5rhtsMctu6NKcgYUEZJyR09ateD/+R30T/r/g/wDRgrrIfEB0zVptO8M2V9rNz/ak11NHJblcDY8bKixszdHY78g8LxT00+f4W/zEcLf6VqOlOiapYXVk0gyi3ELRlh6jcBmn2GiarqqO+l6ZeXqxnDtb27SBT6HaDitXxXoVrpEOnXFvDd2Et6kjS6bekGW32tgHOFO1ucZUH5T1q/oem6hqnw4v4NJtZ7q4GqwMFgQsyjy3546DJHNJbP8ArrYOqX9bNnHyRvDK8UyNHIjFXRhgqR1BHY1PLYXMGn297JHi3uWdYnDA5K43AgHIIyOuOtddrRvLvxZrrWVtY6u8emILyeTa4i2wxrLKhJHzhweRk8nrWZ4f/wCJtoGqaCfmm2/brIH/AJ6Rj51Hu0efxUUPRDWpjtpGpK5RtPug4g+0lTC2RF18zp93/a6VLd+HtasLQ3V9pF/bW4xmaa2dEGenzEYr0bWtct11i/0pIJFvo9MmtmmIGGtxZM4X1z5hz06AU/VLKztPibBqmp6LqUNqtzAX1Od/9E+4oU48oYG7HO88j8KbW3mRzaHmV3ouqafHE9/pt5apMcRNNAyCQ/7ORz+FOXQ9TN5cWj2UsVzbQtPLDMPLdUUbidrYJ45wOcV6LpHhWWO4Or+II9QtZbTUTcXkl3KWt7kIksglUBMkbgBkFuGOOuKx9Z1YWd14b8SxX9vqNypkgu3t1cLL5bAkEOqnmOUKeMUWe39eX4lXWpxljpt9qk5h0yyuLyVV3GO3iaRgOmcAHjkVbh8M69cSSpb6JqMrwvslVLSRjG2M4bA4OCODXVRabd6deazo9poN9rWiX7RTJJYq4kVD88TBwrDo3IYEZHbFaFx4bt7XwzqOlWdjqWvQ2utfKNPcLIn7gE78JIPlJKnAHI7dKl7X/roPqef/ANiaqJ4YTpl55twWEMf2d90hXhgoxzjHOOlP/wCEe1r+z/t39kX/ANj2eZ9o+zP5ezru3Yxj3r0q21iDSdR8K6XJaXEd6kdt9lMmMwiWUiZW6HO1QOnc9KxPEelvcaLot3DoOrXbpo9vtvIGzboRnhl8o5x3+Ydap7X87Ep337XOSXw3rjQQzro2oGKfb5UgtX2ybvu7TjnPbHWnS+FvEELRLNoWpRtM+yMPZyAu2CcDjk4BOPY103ifTn1XSdQ8RazpV9oWqI8IZLgMIb1m+U+WrgMpAG7ALAAdq1bMbviL4OG0tnRIRtXqf3MnApPRX8/0uM4Q+F9fW5W3bQ9SE7qXWI2km5lBAJAxnAyOfcVQurS5sbp7a9t5baeP78UyFGXjPIPI4Nd1o2nQ6ZJrP2vQdY06B9GmLxXkwDygSR/cYwqB/wB8tWP4+ffrVmYSGsP7PgFg/wDE0AXA3f7QIYHpyOlJ9P67/wCQ/wCvy/zOes7K61C5FvYW011MwJEUMZdiB14HNOm0+6t7C3vZY8W9yzrE4YHJTG4EA5BGR1x1rvvhnPp9pcWMdtqNrHqd7dhbiOWOXesKnIjQhCuWYZJLDgAetYWkoNS0zWvDyyrPIrNe2LrnDyR53hQQD80eTggH5RnFN6L+v62Etf69f1/A50afeGyW8FpObVpPKWcRnYX/ALobpn2p0+mX9tJcJc2VzE9qAZ1kiZTEDgDcCPlzkYz6ivTUtrWDTh4Zk1K1DW+lrObTbJ5oulP2hjnZs+6Sv3s4rTuNPi8U3/iNCwEy3aWF3jg+Wt2rI/A/ublz/wBMxVcuq+f+X5kuXut+n4/8A8gbSdRS5gt3sLpZ7lQ8ERhYNKp6FRjJB9RUNta3F5dJbWkEs87nCRRIWZj6ADk161b3VrrfiDR9Rh1G1vZLbXGhjW2SRfLglVmjU70Xp5bdMjmub8O6Pqdha69aT2F3ZapfWZWySaJonuEDgzJHuHzMU7DJqel3/XUrrb0/No5VfD+stqDWC6RfG8VN7W4tn8wL/eK4zjkc1FLpN9BZy3U9u0cUM/2aXeQGjkwTtZfvDoeo7GvQNX8PzReCU0vQrTUhfzeRd3GnXD+ZcJDulUKFCqSqv8xG3gvk1TknN74z1HQb+4SVtUsoLWSTeGUXkcKbWJ9RIrKT/tGm1uv6f9beolJOz/r+upx0ehavNpx1CLSr17IKWNytu5jAHU7sYwMH8qoV6PFqNn4c0TwlqN+bz7ZZRXXl2scShZG81xtdywKjPBAVsjNUNb1Ky0rTtLsU0TT3S70RGllMCiUyupCyb8EgqQDxjPOe2E97Lvb8/wDIrW9jjbS1mvr2G0tU8yeeRY40yBuYnAGTx1NLJZXEWoPYtCxuklMJiT5iXBxtGOpzxxXr1r4bTTyv2q1jMllfaf8AZ549LS2TJlQN5coYvMuG5LDrg9aytakh0jxRp17pEFmdKfVCLu+nto5JBceb+8R2dT5eByoGOPmyTkh21Xn/AMD/AD/IV9Gzzi30+6udTj06KFvtckohWFvlO8nG05xg545qxLoOpQR6i8tttXTJFiuz5inymLFQOvPIPTNdheWniCX4kCOfQVeJr2TaP7GiKywGUBnP7vDjBHznJGevNa2k6RY2dx4xs/EVs9hp730DRo6GFGTzpPL5xxGSACw7Zx60lrFP+ug3o2jyiivSIvD17qMUa3OiWyahBrwF7Fb2yIsMHloRnAwIyASCeD1yScnqX07SoL37OmiaWVFxbjLWUZOJL2SJhyOmxQAO3XrjDs7pd3b71cV9Wv63seL3dhc2K27XUexbmETQkMGDISRng+oIx1GKbNY3dvaw3NxazRQXAJhleMqsmOu0ng49q6LTreTxJ4POnQjzL7S7gSQL1LQSsFdR/uvtb/gTV200Gm6hfNpAv7K9g0i/sVgtVjkLKiukEqvuQKQxYE4JoUW3Zf0v+B/mF+r/AK/4c8gq1p2nXGq3q2tn5Rmf7olnSIMfQFyBn2zmvRLbTYPEDIkdjplrLb65JbQlLMbfKWFm2sq4MpygIDE5JwTgmm6p4embUfCc1npdwb15na5CaYtrJsSVMM8MRIXAbrxxjNKPvW8/11CTsnY4mDU9U8PPeaaPJAE2J7e4t4rhBIhK5w6suRkjIqlqGoXWqX8t7fzGa4lOXcgDPGAABwAAAABwAMV6RZ+Hp213xbLNYKl/9tU2pvNOe6HlvM5ZhEEc4IXAfbjtkZqW9sxcai8/hfw5bSSvqvlXttcaahNvCFURgxMv7pGG9i2AfVsiiOtv67f5jen9ep5pDpWoXDW629hcytdBjAEhZjMFzuK4HzYwc46YqpXreq28MvhWLTvCzL9skW8WzCMWM9otzJujjP8AeKhT3LKCB75q+Ho5LZ9Sj0yE6ePC+8TiJfLNwI+SD0MgIOccjHND0v5N/r/l94f1+X+Z5/f2Fzpl41rex+XMqqxAYMCGUMCCCQQQQeKbdWN3YtGL21mtzKgkjE0ZTeh6MM9QfWuw07S/+Ey0/QTnMtncrp963cQEl43+gUOuf9la6eNtN8S6jbXk1zYaqttfXUawpHJtWCSCR41YOi/dMRwR+eaHpfyv939MSf8AX9eh5DV/SNEvtcuTb6YkUkwGRG9xHGzf7odhuPHQZru9O01desdJ1OKzsYbs2t68i2+mJKZBG6hQluNqO434G7tyckCr0Hh+WD4o6FLYabMFFlHLeGKzEQRysi5dEJWMnbyAcZzRqv69f8gutbHnun+JdU0y1jtrWWBoopTLEs9pFN5TnGSpdSV6DpjpWbPPLc3Ek9xI0s0rl5Hc5LMTkknuSa9J8GeHjbaOkt9pS/a49UC3kN1pT3cht1jU7AgjYxk7s5O3PHOBU1pYQT3umyWuh2h0W5a4fV5Z7RCbVt7b0d8Zg2Jt2hdvPTNN/wBfcC2/ruebjStQLMBYXJKwfaWHktxFjPmdPu4/i6VUr07xRHLd+ErSDSD/AKRFplnPqMQ+/LbiMbWH+wjZLD3DHgcXl8P2Fjr+tajqGm28GkfbLFrWcxKsewypuaM9NuDyRxg4ot7/AC+bX3MG7RueXXOk6jZWkN1eWF1b284BimlhZUkBGRtJGDxzxVSvRNcstR+1rqUekyaZ4ibWPLtD5zu12mCS+JSQwUhBuGEIfpiub8X/ANjf2lH/AGOY/tG3/TRaj/RRLxnyT125z7f3eMVNyutilpniHUdItZra0eBredleSG5tYrhCy5AbbIrAHk8jmmyS6p4p8QBpC97qN9KqLwAXY4VQBwAAMADgADsBTdF0PUfEWqJp2j2/2i6dWZY96pkAZPLEDpW54VX+xNL1XxLMNston2Ox/wCvmUEbh2+RN7c+1ctWVOlzSglz/jror9bNr8PIFdqy/r+tzl54Xt7iSGXbvjYo21gwyDg4I4P1HFWLDS7zU/tP2GHzfstu9zN8wXbGuNzcnnGRwOa9P+HvhaO90jSZL20hurLUGufOCaUk20DKDzbl2DRNn7oQeh5JNS2enT2Gi6kdO063j0J/C7NHfC3j33EzxhnzMRvY7t4KA4UKOBxXFUzOMZSpx3X3btffpt+Jaim/67r/ADPMIdD1CfRhqscK/YTdC089pUUCUjdtOTkDHOTx70mr6JfaHcQw6lHGjTwieIxTpKrxkkBgyEgjIPeuqctrnwukuW0+zfUBrMVtC9np8UMjKYidg8tAWyccc1a8WaLr9tceFJLbRLuSdNIhgWOXTzKDKDKxjKMpBYLk7SM45xWyxclU5ZWWrX3K+/8AwBcuia7frY4rWdGv/D+qS6dq8H2e7iCl496tjIBHKkjoR3qtb2lxdeb9lglm8mMyyeWhbYg6scdAPWvTNV0RtM+Kt0ZPCd3PaTWmbeOz0wSLG/kIDKkJUJIFduVPGTzzip9JjvtH8U6zpcZsLm6vtCeeC3/sm3hm80xqRC8IU7XwCTGCQepBzWSzB+yUkk3yqX+fe1vNj5U2ec6f4g1DSrdotPeGAsGHnLbRGZQwwdspXevHoRWbV3WFvk1aYatafY7sEeZB9lW22cDH7tVULxg9BnOe9Uq9aL5kmZ7BRRRTAKKKKACtLT/EGoaVbtFp7wwFgw85baIzKGGDtlK7149CKzaKACiiigAooooAKsw6jdW+n3NjFLttrpkaZNoO4oSV5xkYyelVqKALJ1G6OlLppl/0RZjOI9o4cqFJzjPQDjOKrUUUAFFXtF019Y12y06M4N1Osef7oJ5P4DmvTdW0q28S3Fve3i20sGn3FzBHb2d3HKZoEiaWGMmNmK58tx69cU+lwWrseUvbTxwRTyQSJFNnypGQhZMHB2nvg+lLdWdzY3LW99by20y43RTIUYZGRkHnpXZa7ew2/hXwvcTaFZiOSK6KWjtcCMAyKQ4Pmb+fdiOfpWhrWi28nizWbueC2ltraG1y+p3Vy6o0iLgfuy0rk8gc4AHPaktQPN6kt7ia0uEntZpIJozlJI2Ksp9QRyK73xH4e0fwxHql5Fpq323UEtooLiSUR26mFZSTtZWJJYqMnopzk1HZaR4ag0XS7rXPsdvHqaTSySlroyQ/MVVYQisny4BIcsTnnHBJ/X4XA4KivQJNE0NrU2SaWqT/APCPjUjeefIXEoRW4XO3acHIIPXgjgVQ1yLRPD/iiPR5dEintLZ4fPupJZhNOpAZmGHCgEE4wvTHNP7Vv63sK+l/66f5nHUV1Xinw/Z+G9P+zOm+9uL2SS3mEmf9EUYQ46fOTnP+zUPgj/kJal/2Cbv/ANFGktVf+trj6pedvxsYsUeoatcQWsC3N7MqbIYUDSMFGW2qBk4HJwPeqpGDg8Guo+HrRN4ytLeWDcZyyiZJpYpIvlbJVo3Xr05zxSeFrbSL+3ktbiO0l1ie5jS2TUDOIXUgjCmEghyxHLfLj8ae+wtNjmKK7priOw+G0kF3oVjJJbayYJUeSYgOIiNxKyj5uCODt46d64iJBJMiMwQMwBY9uetJe87Ib0V2OS2nlt5Z4oZHhhx5sioSseTgZPbJ6ZqKvT20eJdO8TaHpmj+VHDe2dms3muHum83HzMxKAnO75VAAYdaks/COhXc9g9xaWZQanLZzrp8tzsdVhdsFpcEsrJ1U4OaV9L/ANdP8w2dv6/rQ8vhhluJ0ht43llkYKkaKWZieAAB1NE0MttO8NxG8UsbFXjdSrKR1BB6GuosdS0SbXNHbTtEFtN9qMcqm4m27S6+W6uJAwkA3ZIwM4IHYY/iVBH4r1aMM7BL2ZQXcuxAcjljkk+55p9gXUzKkt7ia0uEntZpIJozlJI2Ksp9QRyK7nSGex8EaTJpt9dab9u1CWG+vbFN0quAvlRk7lIXBJxn35qXVbLw9oEdk+uaY2pT3N9eRXl0jtESscpXckasFDdDj7uBjHOQPRiumv68/wDI89q2ulag+mtqKWNy1ih2tdCFjEpyBgvjA5IH412un+F9ObQZhqVnarPHpB1EGG4nN1j7ysx2+SqsOMYJAI6mqfjGTSx4d8PRWtneRsdP3wb7tWSMGZ925RGNzHB5BXtxxy3p/Xr/AJDWv9en+ZxdFbHhKeO38W6Y01tFcKbqJQJSwCEuPmG1hyPfI9Qa6f7Fp2seL/EL3mnWVqumC4mZS9ztmbzQu6QKXfC5LHYF/AdF/wAH8A/4H4nAUqO0bq6MVZTkMDgg+td6bHwoumavqlraRaj9ltbZhFFLOkMc7uyMAX2uU4VsHnnAPeq2mx+F9VvLiHTbK3ivLm4hSztdTe48oAphlVomzu8zoXONuOhzQr9BHGzTSXEzzTyNLLIxd3diWZickknqTTp7ae2KC5hkhMiCRBIhXcp6MM9QfWu7t/D2gaVpenDxMbSBrwT/AGmV3uWliZXZAsXlqYyVKgncTknsME2dStdP1OO3s7jTQ00HhlLoXwlfdGyRAqNudu0kbTkEkt1HFO3bv+j/AMh9df61X+Z5tRXfan4e03S7rxZcLpKy2+lyW62yTPL5YLsu4EhgTkH1ql8SbqGTxhdwJptvDIGiY3CNLvlBiXghnK457KOg96mL5reYHHUVo68iJrlwsWky6OgK4sZnZmi+UdSwB568+tZ1MAooooAkhuJrZnNvNJEXRo2KMV3KRgqcdQRwRUdFFABRRRQBq6brrafptxp81jaX1pPKkrRXPmDDqCAQY3U9GPBJFVtW1SfWdUlvrpY0kkCgJEu1UVVCqoHoFAHPPHOap0UAFFFFAEkdxNFDLFFNIkcwAlRWIDgHIBHfBAPPeo6KKACiiigDb0/xM9lp1vZT6ZY38VrO08H2kSAxs23P3HUEfKOGzWXe3c2oX9xeXTb57iVpZGAAyzHJOB05NQUUAFFFFABRRRQAVKbq4NmLQzyfZhIZRDvOwOQAW29M4AGfaoqKVk9wClUhWBKhgDkg9DSUUwNnUfEb3ujR6Ta6fZ6bYrcG5aK18w+ZJt2hmaR3bgZAAIHJ4rGooqIQjBWiAUUUVYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAS29zPZ3Cz2k0kEy52yROVYZGDgj2NOtr67smDWd1NblXWQGKQrhwCA3Hcbjg+59aZbvFHcxPcxGaFXBkjV9pdc8jODjPriul8bx2qroM1nZW9kLjSo5Xjt02rks4z6k4AGSSTjkmnsrhvdGUusfbrjf4lk1LVFUHYBfbWVjjJy6P1A9B25qzqHi7VbrX7zVNPurjTGudqlLW4dMIoCqpIIJwAPxz0rCopAaEGv6zbXUtzbatfQ3EwAlljuXV5ABgAkHJx70tt4h1qzSVLTV7+BZnMkqxXLqJGPVmweSfU1nUUAWf7Svt277bcZ8n7PnzW/1WMeX1+7jjHSrUOu3j28VjqV9qNzpaYBs0vCi4HQDIZRg4P3TWZUkEohuI5WiSYIwYxyZ2vg9Dgg4PsaAL2u6w+t6kLgxCCGKJILeAMW8qJBhVyeTx1Pck1Vsr+7065Fxp91PaTgECWCQowB6jI5rpfFSpH4e00anZWlprcsjzGK1tlh2WzAbBIqgDdnJHfGM1zNnZyX1ysELQq7AkGedIV4/2nIA/OhXu/6/rz8we39f16F4eKfEC3DTrrupiZ1CNILyTcygkgE5zgZPHuaE8Ua/E0rR65qSNM26UrdyDzDgDJ55OABz2Aq1omkwS+IDoup27XMl0BHFNp8yztA5wQ42MVcY+8M9M8gisvVLA6Xqk9kbm3uvJbb51tIHjf3DDrQAthq+paX5n9mahdWfmgCT7PM0e/HTOCM9T+dU67DwaItQjk0+5/s243wzLDYvZqLmZ9hKlZ9ny4PrIDxjHSuPoe4dDQl8QazPEkU2rX0kcYUIj3LkLtIIwM8YIBH0FPm8S67cSI9xrWoSvGdyM907FTgjIyeOGI/E+tRz6VcWVtFdztZSxMVwkd9FIxzzgojlh78DHtWn4k0O1tNPstZ07NrbagNyafcSAzQ9eR3eI4O1/wADzyR+YLyMax1K+0ucz6ZeXFnKV2mS3laNiPTII44FF9qN7qdwJ9SvLi8mC7RJcStIwHpknpyarUUAXLDV9S0pnbS9QurIyABzbTNHux0ztIz1qO5v7u9Ci8up7gKzOvmyFsMxyx57k8n1NV6KAND+39Z+wCx/ta++yBNgt/tL+WFxjbtzjGDjFVJrme4SJJ55JVhTZEHckIuc4GegyTwPWoqKACtKTxFrc11Bcy6xfyT2+RDK105aLIwdpzkZHpWbRQBduta1S+837bqV5c+cqrJ507PvCnKg5PIBJI9M06x13VtMt2g03VL20hdtzR29w8ascYyQD1wKoUUAXrLXNW023e307VL20hclmjguHRWOMZIBweKJtc1a409bCfVL2WzUBVtnuHMYA6DaTjjtVGigC/ca7q11Zi0utUvZrYKEEMlw7IFGMDaTjAwPyFMuNX1K7sobO61C6ntoMeVBJOzJHgYG1ScDA44qnRQBNdXdxfXL3F7PLcTvjfLM5dmwMck8ngVDRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAPhlaGZJUClkYMA6B1JHqpBBHseK0tV8R6lrVvDBqDWzpbgLF5VnDEUUZwoKKDt5Py9Pas6BokuI2uI2liVgXRX2llzyAcHH1wa6PX7PTx4Y07UYtNj0m8uppNltHK7iS3GNsuHZiOcjOQDjIFD2BbnMUUUUAFFFFABUtrcy2d3Dc27BZYXEiMVDAMDkHB4PPrUVSW4hNzELsyLBvHmGIAsFzzgEgZxTW4nsaGr+ItQ1xmbUzaySM/mNLHZQxSO2McuiBj16E1l10njDT9Osf7Gl0m1a2ivNNSd1eUyMWLMMknvgDoAPQCsKzsbvUblbfT7Wa6nYErFBGXY468Dml5Dfcdaajd2EdwlnM0IuY/KlKABmTOSueoBxzjqODxVatrTdEjfV5NJ137Vpd7KoW3MsJASUkbRIhG7awOMjpkHBFUNT0y70bU59P1GEw3MDbZEJBwevUdeDmgCxZ+INQsLT7PZNbwjayiZLSITANnOJdu8cEj73TjpWZXWeFdNtNWs54bvTLORUhmIniumF4zhCyhIfM+bnA/1eMZyeK5Oh7/15h0FRijqwxlTkZAI/I9anv7+61O+lvL+d57iU5d3PX29gOgA4A4FTz6LqdjbRXmoaXewWchXbNJAyK4PIwxGOR0q7rnh4WNjBq+lySXOjXjFYJpF2ujDOY3H94YPI4OMj0AwMOitfw5pMOqagz38hh060Qz3ko6rGP4R/tMSFHuan8aWNlp/iiaDS7f7NamGGRIt7Pt3xKx5YknkmgFr/AF/XcwaKu2Gi6pqqu2l6beXqxkBzbwNJtz0ztBxWjo/h1fEGnzRaW0p1m23SPaOBtniGOUPZl5yp6gjBzxQBg0UV2HirS7aC2nl0PSdNfTYjEPt9petPIhZR99RKwXLbhyg9BQHWxx9FSQW811cJBaxSTTSNtSONSzMfQAck1pQ6SNP1aK18V29/pkUqHDtAVdM5CvsYAsoPUDBxnBzQBk0Vpa7oV74d1M2WoqofaJI3Q5WRD0Yex98H1FZtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBLazi2u4ZzDHOInD+VKCUfBzhgMZB71q+IfEj+I7g3N1p1nBcswLTQNNkqBgKA8jKF9gB0rHiKCZDMrPHuG5UbaSO4BwcH3wa6LxHoFvpej2N4tpqGmXFzJIhstQcO5RQuJAQiYBLYwR260PYFuc3RRRQAUUUUAFPhdY5kd4llVWBMbkhXHocEHB9iDTKkt445bmKOaYQRs4DyspYIM8tgcnHtTW4GtrniM65a2kMmmWVr9jjWGF7cy7hGM4T55GBGWznGfesWt/xVo1ho50ttLluJYr2xS5LXAUMSWYdB90fLnGTj1NYFLqMv6VrFzosss9gI0uZIzGlwVy8OepQ/wsRxnqATjHWqJJJJJyT1JrQ0bSk1i5ktBeRW10yZtkm4Sd8/c39FJGcE8E8ZGap3NtPZ3UltdwvBPE2145FKsp9CDQI0tL8QHR40ex02zW9RWCXzGUyruBGQN+zIBwPl/Xmsiul8N6Ro+uK1lJ9vS+FvNO90HQQQbFLDcuCSpxgtuXBYDHc81Q9/68w6CoQrqSocA5KnOD7cVd1bWLvWrwT3jKAiiOKGNdscKDoiL2Uf/XOTk1SQKXUOSq5+YgZIH0rU1rQZtJMc8Uq3mnXHNtewg7JR6HP3XHdTyPpgkBFaw1fUtKZ20vULqyMgAc20zR7sdM7SM9am17X77xHqQvtTk3yiJIh8zEBVGOMk9Tkn3JPerfhPRYdc1G6huIbu48mzknSC0YLJKy4woJVuufQ1Q1e3S01J4I7C808xgBre9fdIpxnn5E7EcYofYEUavprN1Doj6Vb7IIJpN9w0Yw8+Puq7d1XqFGBk5OTiqFa1loMmp6PLd6ZOtxdW5JuLEKRIsfGJF/vr1zjleDjBzR0DqZNbFx4hZ9Km06x06y06C4ZTcfZhIzTbTlQTI7EAHnAxWQuNw3AkZ5AOM10k2n6FL4Ru9VgtdQs5VnS3tRNeJMsz/efIESkBVxznqwoewdTmquaXqUmk6gl5bxQyTRgmIzJvEbY4cDoSOozkZ7VTq5pVnb3+oJbXd9HYrICFnlUlA+PlDEcqpPBbnHXFAmV555rq4knuZXmmkYs8jsSzE9SSepqOrWpabeaRfPZ6jA0E6dVbuOxB6EHsRwa6Lwr4ZsdZ08XFwl1eSG7WGWG0uEiNrFgEzvuVsryRngDacnkULbQbvfU5OirF/BFbajcwW04uIYpWSOYDiRQSA34jmq9JO6uN6MKK1tR0GS102DU7Gdb7TpQA08akGGTHMci/wtnp2Ycg+mTTEFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQA+ExrMhnRniDDeqMFYjuASDg++D9K19Q1u2l0GPR9Ls5ra0W5N05uLkTOz7dowQiADHbGT61jKpZgqgkk4AA610PiTwk/hvTdPnuLtZbi5LpPAqY+zuoUld2TuPzgHgYII5oewLc52iiigAooooAKfCY1mQzozxBhvVGCsR3AJBwffB+lMqSBEluI45ZVhRmAaVgSEGeSQMk49qa3A2/EWvWGtWenxWmnXNrJY262yPLdrKGjUseQI1+bLdc446Vj2d9d6dcrcafdTWs6ghZYJCjDPXkc1q6roVpa6JBq2lahJe2klw9s3nW/kurqA3Tc2QQc5z+FYdK+odDYsvEM9vqcuqXynUtR2AQT3khl8p+AHIbO8gdATgHBwcYrNu7y51C7kur64kuLiQ5eWVyzN25J9qs6To11rUk0NgY3uY4zIluWxJNjqEH8TAc7epAOM1RIIJBGCOoNAHSWHiPS7Xwq2jy6TdBpn33VzbXqxtcYOVUhomwo44B5PNc1W6NF0238O2WpajqN1HJemURQwWayAbCByxlXrkdAawqOodC/Prep3trFaahqN7c2cRUrbyXLMqgcDAOQOOBxxU+s+ILjVUjtYkFnpkH/AB72MTHy4+vzH+85ycueTk9BxWUihnVSwQE4LNnA9+Ku6vo93ol79nvVUhlDxTRtujmQ9HRhwyn1/A4PFAIisZLGOcnU7e4uItvCW9wsLA+uSj8deMfjVvXtaOt3cDrB9ngtbdLW3iLl2WNM43Mep5POB9BWXRQBdsNa1TSlddL1K8slkILi3naPdjpnaRmrdh4hn0rTJIdLT7NezsfPv1c+aU4wiH+AZySRy3HOBg49FABWrq+tDUbHTrG3tza2lhCUSMyby8jHLyE4HLHHHYACsqigCSC4mtbhJ7WWSGaNtySRsVZT6gjkGtGLXp5tWiv9d83WmhQiNL2dnXPO3dnJKgnJXIz681lUUAXNT1W+1m9a71O5e4mIwCx4Vc5CqOiqM8AcCtrQ/FkOl6Tb2NzY3Egtb37bFJaXn2dmfAGH+RtwwOMYIyeea5mijYHruWdRvX1LVLq+lREe5meZljGFUsxJA9ualOt6qdM/s46neGxxj7L9obysZz9zOOvPSqNFHSwdbmvfeIJp9Li0qwj+w6cijfBG5JuH4JeRuN5yOB0UYAHc5FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAXNJ1J9I1a21CKCGeS3cSJHOpKEjpkAg8Hnr2rY13xNa6z4bsbKPTo7a6huZp5ZEaRgd+OheRiSSMnPoMd84NpbPe30FrEVDzyLGpboCTgZ/OrusaPHpEzwf2pZ3k8crRSxW6zZjIODkvGoPIxwTQLr/XmZlFFFAwooooAKltfs5u4ftpkFvvHmmIAuEzztzxnHTNRVJBCbi4jhVkQyMFDSOFUZOMkngD3NNbiex0PivV9J1OO2i0Sa8itLQeXb2MtqsaRIeWYuJWLOTjJwM/gBWDZ3kljcrPCsLOoIAngSZef9lwQfyrQ1Tw9Lpunw38V7Z39nLK0AntGchZFAJUh1U9CDnGD61kVK8hs2dO1m2i1h9V1O1W4uIgHtoIIkghMoxtLhNuFHXCjLEckcmqOp6ldaxqc+oag6yXM7bpGVAgJ6dAAKSz0281CK5ksoGmFrF50wTBZUyAWx1IGRnHTqaq0wOr8OeJoNDs18/UdUuUCSBtI2AWkhYEYYmQ5HOT+76/nXKVr6X4fOrwN9j1GzN2I3kFk/mLKwQEnB2bOgJ+9WRQ9w6F+fVp7y1itbhLRIUK/PBYwpJgccsqhm/E896u634hW+sLfSNNhNvpFmxaCOTDSux6yO3Ytn7o+UcdcZrERS7qi4yxwMkAfmelWL/T7vS7+Wy1CB7e4hba8bjkf4g9QRwRzQwK1FWdPsLnVNRgsbGMy3Fw4RFHcn+nvVjXdFn8P6xLp11LDNJGqN5kDEowZQwwSAehHagDOooq02m3i6WmpGBjZySmETLgqHAB2nHQ4ORnGe3SgCrRRWrqWgvpNupvr60S8IDNYKXaZAem4hdgOOcbsjPIzxQBlUUVa0/TbvVbr7Np8DTz7GcRIRuYKMnA7nAPA5NAFWigjBweDRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAT2UjRX9vJHOtu6SqyzOCVjIP3iACSB16H6Gui8SalYXmjxJJc2Ooau100sl5Y2f2dRGRyrfIm9i3zZK8evNcuqlmCqCSTgADrV/UtDvtIVP7QSGF2xmH7RG0qZGcPGGLIcf3gKHsHUz6KKKACiiigAqW1jimu4Y7icW8TuFeYqWEak8tgcnA5wKiqS3gkurmK3gXdLK4RFzjJJwBzTW4nsdN4sudN+wWun+HtTtZtMtWOyBEmE0rkfNNIXjVcnAGATgYA7mucs7iO2uVlntIbxADmGcuFP/AHwyn9atapod5o523xtQ4coyQ3sMzKw6hlRiR+NZ1Shs39E1OwtfEH9sTBtNW0AkhtNPaTdK44CB3LFQeSxJPGQAc4rL1S/Gp6pPei1t7Pzm3eRbKVjT6Ak49agS3mlhlljid44QDI6qSEBOASe2ScVHTC51nhXUbLSbd5NQ1GxNpNHKs9klq5upAyldiy+WNoPB4kAxnI6g8nRRR1uBfnvra6tYrePS7KzcFd11G05ZuxyGdl9zhfp6Vp+I9ctLvTrLR7BTdQaeNiajcKRNL1yo/uxZPyqckeoyRXO0UbhsbXhzW7XRDetcWc88lzAYElguRC8Kn72CUYZI4zjgE+tTeNdS07VfEX2nSPMMH2aFC0jZyyxhTj5VxjAHuQT0IA5+igNn/Xl/kXbC/t7NXFxpVnfliMG4eYbPp5ci/rmtfQtbs9A0i9lwL66vgYDp8it9nRBg75P77Z+6AeOST0Fc3RRuGw6N/LlR9obawOD0Nd74n8VWuqWGtMdae/i1FoXsrBonzZMGDMTuUIuBuTKE7t2T3rgKKd9LASQSLDcJJJCk6q2TFIWCv7HaQcfQitS1utOu9bt57iNNEtoRvY2Hmu7FeRs3uxDk4AOQB1rHopAaviTWx4g1uW/WzhtN4xtjzl/9pz/E57nAz1xWVRRQNu4UUUUCCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKANPw3eW2n+KNMu74ZtoLqOSXgnChgScDrjrit/wAWSQPo832xtHk1BtSZraTTfKJa32tuLmPsW2Y3nf8Ae965GC3murhILWKSaaRtqRxqWZj6ADkmrd/oOr6XCs2p6Ve2cTNsV7i3eNS2M4yQOeD+VN7f15B1KFFFFIAooooAKs6dLLDqlrLb+X5scyMnmsFTcGBG4kgAZ6kkVWp0UUk0yRQo0kjsFREGSxPQAdzTW4nqjqfFK2EumJeTxafb63cXbvLFpt2biN4iMlmO9wrbugDDgnjpXN2b2iXKtqEM08GDuSCYRMfT5irD9Kkv9K1HSnRNUsLqyaQZRbiFoyw9RuAzVSpXkNnS+HLy0tPEMl/bXb6Xp0MWZ4riRbiS4Q4BhC7VEm49iAAOSeM1i6pcWd1qk8+mWZsrV2zHbmUyeWPTcQM1UopgFFXbbRtUvbKS8s9Nu7i1iz5k8UDMiYGTlgMDA5qlQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAKqlmCqCSTgADrXS+J1/sXTLDw0jYktx9qvx/08OB8v/AEwv1LVzNFHSwBRRRQAUUUUAFPhikuJ0hgjeWWRgqIi5ZieAAB1NMooA7PxHamy8DWFu1leaSY76Qiy1Bt00u5FzKDtTCjAXG3GT1Pbk7NLR7lV1CaaCDB3PBCJWHp8pZR+tQUUdbh0sdT4YWOPXLlbOaOfRPJH9oPqUIjQw5GcqrMQ27Gzad27GO9YOqf2eNUn/sY3Bsd37n7SAJMe+OKqUUAereFLRzo/h25aJ5HgiuvJ1GJCbewyWJ+0fMAepPVMAj7/SvKaKKOtwWxoSRackMDaZeX0t8WX929osaqf9lxIxJzjHyjPtW34y+wlbc3Jj/4SP8A5iQtFHkZ5+9jjzem7b8uc981ylFD1BaBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGn4bge58UaZDFbxXLvdRgQzHCSfMPlbg8Hvwfoa6/xrc/bfCryR31xqMcOsyRvNesxkt2KH91Hkcx4XOcjoMotefI7RuroxVlOQwOCD61c1DWNT1by/7U1G7vfLzs+0ztJtz1xuJx0FD1Vv66f5Arp/15lKiiigAooooAKdFK8MySwu0ciMGV0OCpHQg9jTafDNJbzJNBI0UsbBkdGIZSOQQR0NAHYeI7o6p4GsLwX91qQhvpIXutRXbPkorBF+Z8oACfvdT0FcnZ28dzcrFPdw2aEHM04cqP++FY/pUl/q2o6q6Pql/dXrRjCNcTNIVHoNxOKqUdQ6WOl8OWEMniGTTDDa6zZTRYnuIy0QgQYJmWSRVKbe5YYPI5yKxdUtrS01SeDTr0X1qjYjuRGU8weu08j0qBLiaKGWKOV0jmAEiKxAcA5AI74IzUdAHa+Edb1jSdHvtWbVLxbHT4jDa2puH8qS4kBCrszghRucj2HrXFVMbu5azW0a4lNsjl1hLnYrHgsF6Z96hoeruHQvz2VvaW0VzDqtldSkqTbxxy7l7874wpx0OCfxrZ8W6bZW1vaXoSPTdTulD3WkICRDkZEg/uBuD5ZOVyO3TmEdo3V0YqynIYHBB9adLLJPM8s7tJJIxZ3c5ZieSST1NDBf1/X9f5sooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD//Z)

1. **Studi Kasus 3:**



/\*

NAMA    : SURIADI VAJRAKARNA

NPM     : 140810180038

KELAS   : B

TANGGAL : 4 APRIL 2020

STUDI KASUS 3 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA

\*/

#include "graph.hpp"

#include <iostream>

void print(int data) { std::cout << data << ' '; }

int main(int argc, char const \*\*argv)

{

    const size\_t graph\_size = 8;

    Analgo::Graph<int> g(graph\_size);

    g.add\_edge(1, 2);

    g.add\_edge(1, 3);

    g.add\_edge(2, 3);

    g.add\_edge(2, 4);

    g.add\_edge(2, 5);

    g.add\_edge(3, 5);

    g.add\_edge(3, 7);

    g.add\_edge(3, 8);

    g.add\_edge(4, 5);

    g.add\_edge(5, 6);

    g.add\_edge(7, 8);

    try

    {

        std::cout << "Jalur Traversal Breadth First Search/BFS (dimulai dari simpul 1): ";

        g.bfs(1, [](const int &data) { std::cout << data << ' '; });

    }

    catch (const std::exception &e)

    {

        std::cerr << e.what() << '\n';

        return EXIT\_FAILURE;

    }

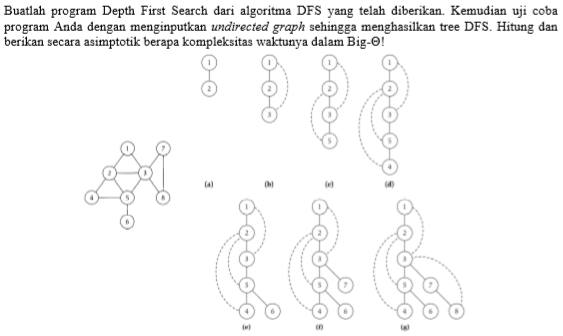
    return EXIT\_SUCCESS;

}



*Breadth First Search* atau BFS merupakan metode pencarian secara melebar dengan mengunjungi node dari paling kiri ke paling kanan pada tingkat yang sama. Setelah itu, baru melanjutkan ke tingkat di bawahnya. Untuk memperkirakan *worst case*, BFS harus mempertimbangkan semua jalur yang mungkin untuk semua simpul sehingga nilai kompleksitas waktu dari BFS adalah **O(|V|+|E|)** di mana V adalah jumlah simpul dan E adalah jumlah ujung yang ada.

1. **Studi Kasus 4:**



/\*

NAMA    : SURIADI VAJRAKARNA

NPM     : 140810180038

KELAS   : B

TANGGAL : 4 APRIL 2020

STUDI KASUS 4 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA

\*/

#include "graph.hpp"

#include <iostream>

void print(int data) { std::cout << data << ' '; }

int main(int argc, char const \*\*argv)

{

    const size\_t graph\_size = 8;

    Analgo::Graph<int> g(graph\_size);

    g.add\_edge(1, 2);

    g.add\_edge(1, 3);

    g.add\_edge(2, 3);

    g.add\_edge(2, 4);

    g.add\_edge(2, 5);

    g.add\_edge(3, 5);

    g.add\_edge(3, 7);

    g.add\_edge(3, 8);

    g.add\_edge(4, 5);

    g.add\_edge(5, 6);

    g.add\_edge(7, 8);

    try

    {

        std::cout << "Jalur Traversal Depth First Search/DFS (dimulai dari simpul 1): ";

        g.dfs(1, [](const int &data) { std::cout << data << ' '; });

    }

    catch (const std::exception &e)

    {

        std::cerr << e.what() << '\n';

        return EXIT\_FAILURE;

    }

    return EXIT\_SUCCESS;

}



*Depth First Search* atau DFS adalah metode pencarian mendalam dengan mengunjungi semua simpul dari paling kiri lalu ke bawah, baru melanjutkan ke kanannya hingga paling kanan. Kompleksitas algoritma DFS apabila dinotasikan dalam Big-O adalah O(|V|+|E|) di mana V adalah jumlah dari simpul dan E adalah jumlah dari ujung karena kita hanya memerlukan satu jalur tunggal dari akar sampai daun terakhir, ditambah dengan simpul-simpul saudara kandungnya yang belum ditelusuri.

1. **Library Graph (Tambahan): *graph.hpp***

#pragma once

#include <algorithm>

#include <list>

#include <map>

#include <queue>

#include <stack>

#include <string>

#include <unordered\_map>

#include <vector>

namespace Analgo

{

template <typename K, typename W, typename T>

class \_base\_graph

{

protected:

    \_base\_graph() {}

    \_base\_graph(size\_t n);

    \_base\_graph(size\_t n, const K &k);

    template <typename F>

    \_base\_graph(size\_t n, F f);

public:

    void add\_node(const K &k);

    void add\_node(const K &k, const T &t);

    size\_t order() { return node.size(); }

    size\_t size() { return this->edge.size(); }

    T &operator[](const K &k);

    virtual bool is\_edge(const K &k1, const K &k2) = 0;

    // graph iterator

    class iterator

    {

        friend class \_base\_graph<K, W, T>;

    public:

        iterator &operator++()

        {

            ++it;

            return \*this;

        }

        iterator operator++(int)

        {

            iterator i = (\*this);

            ++it;

            return i;

        }

        iterator &operator--()

        {

            --it;

            return \*this;

        }

        iterator operator--(int)

        {

            iterator i = (\*this);

            --it;

            return i;

        }

        std::pair<const K, T> &operator\*() { return \*it; }

        std::pair<const K, T> \*operator->() { return &(\*it); }

        bool operator==(const iterator &i) { return it == i.it; }

        bool operator!=(const iterator &i) { return it != i.it; }

    private:

        typename std::map<K, T>::iterator begin, end;

        typename std::map<K, T>::iterator it;

    };

    virtual iterator begin();

    virtual iterator end();

    virtual iterator find(const K &k);

    // edge

    class Edge

    {

    public:

        Edge(const K &k1, const K &k2, const W &w = 0) : ky1(k1), ky2(k2), wt(w) {}

        bool operator==(const Edge &e) const

        {

            return key1() == e.key1() && key2() == e.key2();

        }

        bool operator!=(const Edge &e) const

        {

            return key1() != e.key1() || key2() != e.key2();

        }

        const K &key1() const { return ky1; }

        const K &key2() const { return ky2; }

        const K &key(const K &k);

        K &weight() { return wt; }

    private:

        K ky1;

        K ky2;

        W wt;

    };

    class edge\_iterator

    {

        friend class \_base\_graph<K, W, T>;

    public:

        edge\_iterator &operator++();

        edge\_iterator operator++(int);

        edge\_iterator &operator--();

        edge\_iterator operator--(int);

        Edge &operator\*() { return \*it; }

        Edge \*operator->() { return &(\*it); }

        bool operator==(const edge\_iterator &i) { return it == i.it; }

        bool operator!=(const edge\_iterator &i) { return it != i.it; }

    private:

        typename std::list<Edge>::iterator begin, end;

        typename std::list<Edge>::iterator it;

        K key;

    };

    edge\_iterator begin(const K &k);

    edge\_iterator end(const K &k);

    // traverse algorithm

    template <typename F>

    F bfs(const K &k, F f);

    template <typename F>

    F dfs(const K &k, F f);

protected:

    std::map<K, T> node;

    std::list<Edge> edge;

};

template <typename K, typename W, typename T>

\_base\_graph<K, W, T>::\_base\_graph(size\_t n)

{

    K key(1);

    for (size\_t i = 0; i < n; i++)

        node[key++] = T();

}

template <typename K, typename W, typename T>

\_base\_graph<K, W, T>::\_base\_graph(size\_t n, const K &k)

{

    K key(k);

    for (size\_t i = 0; i < n; i++)

        node[key++] = T();

}

template <typename K, typename W, typename T>

template <typename F>

\_base\_graph<K, W, T>::\_base\_graph(size\_t n, F f)

{

    for (size\_t i = 0; i < n; i++)

        node[f()] = T();

}

template <typename K, typename W, typename T>

void \_base\_graph<K, W, T>::add\_node(const K &k)

{

    if (node.find(k) != node.end())

        throw std::string("\_base\_graph::add\_node - node already exists");

    node[k] = T();

}

template <typename K, typename W, typename T>

void \_base\_graph<K, W, T>::add\_node(const K &k, const T &t)

{

    if (node.find(k) != node.end())

        throw std::string("\_base\_graph::add\_node - node already exists");

    node[k] = t;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::iterator \_base\_graph<K, W, T>::begin()

{

    iterator i;

    i.begin = node.begin();

    i.end = node.end();

    i.it = i.begin;

    return i;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator

\_base\_graph<K, W, T>::begin(const K &k)

{

    edge\_iterator i;

    i.begin = this->edge.begin();

    i.end = this->edge.end();

    i.key = k;

    i.it = i.begin;

    while (i.it != i.end && i.it->key1() != i.key && i.it->key2() != i.key)

    {

        i.it++;

    }

    return i;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::iterator \_base\_graph<K, W, T>::end()

{

    iterator i;

    i.begin = node.begin();

    i.end = node.end();

    i.it = i.end;

    return i;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator

\_base\_graph<K, W, T>::end(const K &k)

{

    edge\_iterator i;

    i.begin = this->edge.begin();

    i.end = this->edge.end();

    i.key = k;

    i.it = i.end;

    return i;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::iterator \_base\_graph<K, W, T>::find(const K &k)

{

    iterator i;

    i.begin = node.begin();

    i.end = node.end();

    i.it = node.find(k);

    return i;

}

template <typename K, typename W, typename T>

template <typename F>

F \_base\_graph<K, W, T>::bfs(const K &k, F f)

{

    K current;

    std::queue<K> q;

    std::unordered\_map<K, bool> visited;

    q.push(k);

    visited[k] = true;

    while (!q.empty())

    {

        current = q.front();

        q.pop();

        f(current);

        for (iterator i = begin(); i != end(); i++)

            if (is\_edge(current, i->first))

                if (!visited[i->first])

                {

                    q.push(i->first);

                    visited[i->first] = true;

                }

    }

    return f;

}

template <typename K, typename W, typename T>

template <typename F>

F \_base\_graph<K, W, T>::dfs(const K &k, F f)

{

    K current = k;

    std::unordered\_map<K, bool> visited;

    std::stack<K> s;

    s.push(current);

    while (!s.empty())

    {

        current = s.top();

        s.pop();

        if (!visited[current])

        {

            f(current);

            visited[current] = true;

            for (iterator i = begin(); i != end(); ++i)

            {

                if (is\_edge(current, i->first))

                {

                    s.push(i->first);

                }

            }

        }

    }

    return f;

}

template <typename K, typename W, typename T>

T &\_base\_graph<K, W, T>::operator[](const K &k)

{

    if (node.find(k) == node.end())

    {

        throw std::string("\_base\_graph::operator[] - node does not exist");

    }

    return node[k];

}

template <typename K, typename W, typename T>

const K &\_base\_graph<K, W, T>::Edge::key(const K &k)

{

    if (k != key1() && k != key2())

    {

        throw std::string("Graph::Edge::key - key supplied is invalid");

    }

    return k == key1() ? key2() : key1();

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator &

\_base\_graph<K, W, T>::edge\_iterator::operator++()

{

    ++it;

    while (it != end && it->key1() != key && it->key2() != key)

    {

        ++it;

    }

    return \*this;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator

\_base\_graph<K, W, T>::edge\_iterator::operator++(int)

{

    edge\_iterator tmp = \*this;

    ++it;

    while (it != end && it->key1() != key && it->key2() != key)

    {

        ++it;

    }

    return tmp;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator &

\_base\_graph<K, W, T>::edge\_iterator::operator--()

{

    --it;

    while (it != begin && it->key1() != key && it->key2() != key)

    {

        --it;

    }

    if (it == begin)

    {

        while (it != end && it->key1() != key && it->key2() != key)

        {

            ++it;

        }

    }

    return \*this;

}

template <typename K, typename W, typename T>

typename \_base\_graph<K, W, T>::edge\_iterator

\_base\_graph<K, W, T>::edge\_iterator::operator--(int)

{

    edge\_iterator tmp = \*this;

    --it;

    while (it != begin && it->key1() != key && it->key2() != key)

    {

        --it;

    }

    if (it == begin)

    {

        while (it != end && it->key1() != key && it->key2() != key)

        {

            ++it;

        }

    }

    return tmp;

}

// undirected unweighted graph

template <typename K, typename T = void \*>

class Graph : public \_base\_graph<K, void \*, T>

{

    using Edge = typename \_base\_graph<K, void \*, T>::Edge;

public:

    Graph() : \_base\_graph<K, void \*, T>() {}

    Graph(size\_t n) : \_base\_graph<K, void \*, T>(n) {}

    Graph(size\_t n, const K &k) : \_base\_graph<K, void \*, T>(n, k) {}

    template <typename F>

    Graph(size\_t n, F f) : \_base\_graph<K, void \*, T>(n, f) {}

    void add\_edge(const K &k1, const K &k2);

    bool is\_edge(const K &k1, const K &k2);

    void remove\_edge(const K &k1, const K &k2);

};

template <typename K, typename T>

void Graph<K, T>::add\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

    {

        throw std::string("add\_edge: this->node does not exist");

    }

    // Store lower key value in first key of edge

    if (k1 < k2)

        this->edge.push\_back(Edge(k1, k2));

    else

        this->edge.push\_back(Edge(k2, k1));

}

template <typename K, typename T>

bool Graph<K, T>::is\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

    {

        throw std::string("is\_edge: this->node does not exist");

    }

    if (k1 < k2)

        return std::find(this->edge.begin(), this->edge.end(), Edge(k1, k2)) !=

               this->edge.end();

    return std::find(this->edge.begin(), this->edge.end(), Edge(k2, k1)) !=

           this->edge.end();

}

template <typename K, typename T>

void Graph<K, T>::remove\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    if (k1 < k2)

        this->edge.remove(Edge(k1, k2));

    else

        this->edge.remove(Edge(k2, k1));

}

// weighted graph

template <typename K, typename W, typename T = void \*>

class WeightedGraph : public \_base\_graph<K, W, T>

{

    using Edge = typename \_base\_graph<K, W, T>::Edge;

public:

    WeightedGraph() : \_base\_graph<K, W, T>() {}

    WeightedGraph(size\_t n) : \_base\_graph<K, W, T>(n) {}

    WeightedGraph(size\_t n, const K &k) : \_base\_graph<K, W, T>(n, k) {}

    template <typename F>

    WeightedGraph(size\_t n, F f) : \_base\_graph<K, W, T>(n, f) {}

    void add\_edge(const K &k1, const K &k2, const W &w);

    bool is\_edge(const K &k1, const K &k2);

    void remove\_edge(const K &k1, const K &k2);

    W &weight(const K &k1, const K &k2);

};

template <typename K, typename W, typename T>

void WeightedGraph<K, W, T>::add\_edge(const K &k1, const K &k2, const W &w)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    // Store lower key value in first key of edge

    if (k1 < k2)

        this->edge.push\_back(edge(k1, k2, w));

    else

        this->edge.push\_back(edge(k2, k1, w));

}

template <typename K, typename W, typename T>

bool WeightedGraph<K, W, T>::is\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("is\_edge: this->node does not exist");

    if (k1 < k2)

        return std::find(this->edge.begin(), this->edge.end(), edge(k1, k2)) !=

               this->edge.end();

    return std::find(this->edge.begin(), this->edge.end(), edge(k2, k1)) !=

           this->edge.end();

}

template <typename K, typename W, typename T>

void WeightedGraph<K, W, T>::remove\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    if (k1 < k2)

        this->edge.remove(Edge(k1, k2));

    else

        this->edge.remove(Edge(k2, k1));

}

template <typename K, typename W, typename T>

W &WeightedGraph<K, W, T>::weight(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("weight: this->node does not exist");

    if (!is\_edge(k1, k2))

        throw std::string("weight: Edge does not exist");

    if (k1 < k2)

        return std::find(this->edge.begin(), this->edge.end(), Edge(k1, k2))

            ->weight();

    return std::find(this->edge.begin(), this->edge.end(), Edge(k2, k1))

        ->weight();

}

// directed graph

template <typename K, typename T = void \*>

class DirectedGraph : public \_base\_graph<K, void \*, T>

{

    using Edge = typename \_base\_graph<K, void \*, T>::Edge;

public:

    DirectedGraph() : \_base\_graph<K, void \*, T>() {}

    DirectedGraph(size\_t n) : \_base\_graph<K, void \*, T>(n) {}

    DirectedGraph(size\_t n, const K &k) : \_base\_graph<K, void \*, T>(n, k) {}

    template <typename F>

    DirectedGraph(size\_t n, F f) : \_base\_graph<K, void \*, T>(n, f) {}

    void add\_edge(const K &k1, const K &k2);

    bool is\_edge(const K &k1, const K &k2);

    void remove\_edge(const K &k1, const K &k2);

};

template <typename K, typename T>

void DirectedGraph<K, T>::add\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    this->edge.push\_back(edge(k1, k2));

}

template <typename K, typename T>

bool DirectedGraph<K, T>::is\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("is\_edge: this->node does not exist");

    return std::find(this->edge.begin(), this->edge.end(), Edge(k1, k2)) !=

           this->edge.End();

}

template <typename K, typename T>

void DirectedGraph<K, T>::remove\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    this->edge.remove(edge(k1, k2));

}

template <typename K, typename W, typename T = void \*>

class WeightedDirectedGraph : public \_base\_graph<K, W, T>

{

    using Edge = typename \_base\_graph<K, void \*, T>::Edge;

public:

    WeightedDirectedGraph() : \_base\_graph<K, W, T>() {}

    WeightedDirectedGraph(size\_t n) : \_base\_graph<K, W, T>(n) {}

    WeightedDirectedGraph(size\_t n, const K &k) : \_base\_graph<K, W, T>(n, k) {}

    template <typename F>

    WeightedDirectedGraph(size\_t n, F f) : \_base\_graph<K, W, T>(n, f) {}

    void add\_edge(const K &k1, const K &k2, const W &w);

    bool is\_edge(const K &k1, const K &k2);

    void remove\_edge(const K &k1, const K &k2);

    W &weight(const K &k1, const K &k2);

};

template <typename K, typename W, typename T>

void WeightedDirectedGraph<K, W, T>::add\_edge(const K &k1, const K &k2,

                                              const W &w)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    this->edge.push\_back(edge(k1, k2, w));

}

template <typename K, typename W, typename T>

bool WeightedDirectedGraph<K, W, T>::is\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("is\_edge: this->node does not exist");

    return std::find(this->edge.begin(), this->edge.end(), Edge(k1, k2)) !=

           this->edge.end();

}

template <typename K, typename W, typename T>

void WeightedDirectedGraph<K, W, T>::remove\_edge(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("add\_edge: this->node does not exist");

    this->edge.remove(Edge(k1, k2));

}

template <typename K, typename W, typename T>

W &WeightedDirectedGraph<K, W, T>::weight(const K &k1, const K &k2)

{

    if (this->node.find(k1) == this->node.end() ||

        this->node.find(k2) == this->node.end())

        throw std::string("weight: Node does not exist");

    if (!is\_edge(k1, k2))

        throw std::string("weight: Edge does not exist");

    return std::find(this->edge.begin(), this->edge.end(), Edge(k1, k2))

        ->weight();

}

}