**Team Members**

* Nada Abdallah (leader)
* Lubna Ibrahim
* Dalia Yehia
* Yousra Emad
* Yara Gamal
* Hend Hussein
* Dina Badruddin
* Esraa Adel
* Reem Assad

**Abstract**

The protein structure prediction problem is one of the most (if not the most) important problem

in computational biology. This problem consists of finding the conformation of a protein with minimal energy.

Since the latter half of 20th century, a growing number of researchers from diverse academic backgrounds are devoted to bio-related research. Protein, as one of the most widespread and complicated macromolecules within life organisms, attracts a great deal of attentions. Proteins differ from one another primarily in their sequence of amino acids, which usually results in different spatial shape and structure and therefore different biological functionalities in cells.

This problem consists of finding the conformation of a protein with minimal energy. the inference of the three-dimensional structure of a protein from its amino acid sequence—that is, the prediction of its tertiary structure from primary structure. Structure prediction is different from the inverse problem of protein design. Protein structure prediction is one of the most important goals pursued by computational biology; and it is important in medicine (for example, in drug design by finding inhibitors) and biotechnology (for example, in the design of novel enzymes).

Drug design based on structure has become a highly developed technology and is used in large pharmaceutical companies. Firstly, the structure of the protein of interest must be known. Therefore, molecular modelling plays an important role in the discovery of new drugs.

If the structure of the receptor is known, then the application is a problem of structure-based drug design. These methods have specific goals, such as attempting to identify the location of the active site of the ligand and the geometry of the ligand in the active site. Another goal is to select a number of related binders in terms of affinity or evaluation of the binding free energy.

**Acknowledgment**

Thanks to everyone who helped us to carry out our project.

We would like to express our deep gratitude and respect to Prof. Mostafa Al-gayar and Dr. Sara El-Sayed El-Metwally, our supervisors, for their professional guidance and valuable support, encouragement and useful criticism of this project. Finally, our thanks go to our families for their love, prayers, caring and sacrifices for educating and preparing us for our future.

**Table of Contents**

Team Members

Abstract

Acknowledgment

Table of Contents

List of Figures

List of Tables

Chapter 1 (Introduction)

* 1. Overview
     1. Protein structure

1.1.2 Protein inhibitor

1.2 Problem Statement

1.3 Main Project Objectives

1.4 Project Scope

1.5 Project TimeLine

Chapter 2 (Literature Review)

2.1 Introduction

2.2 Background

2.2.1 Conventional Approaches for Protein Structure Prediction

2.2.2 strategies of computational method

2.2.2.1 Homology Modeling:

2.2.2.2 Protein Threading

2.2.2.3 Template-Free Modelling

2.2.3 Conventional Approaches for Protein inhibition

2.2.3.1 molecular docking

2.3 Related works

Chapter 3 (Software Engineering Analysis and UML Diagram)

3.1 Introduction:

3.2 Agile Development Life Cycle:

3.3 System Analysis:

3.4 System Overview

3.5 UMLs work

3.5.1 use case

3.5.1.1 use case Diagram

3.5.1.2 use case Scenario

3.5.2 Sequence

3.5.2.1 Diagram

3.5.2.2 scenario

3.5.3 activity

3.5.3.1 Diagram

3.5.4 Class Diagram

3.5.4.1 diagram

3.6 database design

3.6.1 Database design diagram

Chapter4 (design)

4.1 introduction

4.2 System architecture

4.3 logo

4.4 system design

4.4.1 Home

4.4.2 Enter primary sequence

4.4.3 How it works

4.4.4 About us

**List of Figures**

Figure 1: Four levels of structure

Figure 2: protein 3d structure

Figure 3: A Structure-Based Drug Discovery Paradigm

Figure 4: the basic concept of protein structure prediction has been illustrated schematically based on different protein modelling stated earlier

Figure 5: Homology modelling

Figure 6: threading modelling

Figure 7: templet free modelling

Figure 8: the basic concept of protein structure prediction has been illustrated schematically based on different protein modelling stated earlier

Figure 9: Diagram illustrating the docking of a ligand to a receptor to produce a complex.

Figure 10: inhibition

Figure 11: Agile life cycle

Figure 12: system use case diagram

Figure 13: system sequence diagram

Figure 14: activity diagram

Figure 15: class diagram

Figure 16: database diagram

Figure 17: protein structure prediction system architecture

Figure 18 protein inhibition system architecture

Figure 19 st-inpro logo

Figure 20: home page

Figure 21: insert amino acid sequence

Figure 22: How tool works

Figure 23: About us

**List of tables**

Table 1: project timeline

Table 2: Lipinski’s Rule of Five

Table 3: use case user scenario

Table 4: use case admin scenario

Table 5: sequence scenario

**Chapter 1**

**Introduction**

* 1. **overview:**

1.1.1 Protein structure

Protein is the basic building block of life. This is the key component of the body which is responsible for various physiological biochemical reactions. Protein is an important chunk in bioinformatics field to understand the possible biological process of life. It is very important to predict the protein structure to pursue the following challenges like drug design, medicinal application as well as in bio-industrial applications. Over 25 years, the way out towards prediction of protein structure has been continued. Multiple kinds of approaches have been taken for protein structure prediction through computational approaches which have been developed as the most popular and useful in recent times. To know about the different protein structure prediction approaches, we should first focus on the overview of the structure of protein.

Four types of protein structure have been discovered so far (Fig. 1). Those are primary structure, secondary structure, tertiary structure and quaternary structure. The primary structure of protein is based on the simple linear arrangement of amino acid residue sequences. The secondary protein structure is generally based on the binding pattern of the amino hydrogen and carboxyl oxygen atoms between amino acid sequences throughout the peptide backbone. There are two kinds of protein secondary structure, and those are alpha helices and beta strands. In these structures, the amino acid sequences are linked with each other by hydrogen bonds. The alpha helix structure is generally composed of 3.6 amino acids per turn along with hydrogen bonds which are formed between every fourth residue while in beta strands there are two portions of the chain—one is upward with 5–10 consecutive amino acids and another is downward 5–10 consecutive amino acid sequences. H-bond interactions are formed mostly in between adjacent amino acids and short loops between them. This secondary structure prediction is more likely related to the pattern of alpha helices and beta strands amino acids residue structure. The prediction of secondary structure is mainly focused on to know about the linear amino acid sequences, i.e., primary protein structure [8]. The pattern of the amino acid residues arrangements, their size and shape direct the ligands to fit with the protein in a better way

Diagram

Description automatically generated

Figure 1: Four levels of structure

The tertiary structure is about the three-dimensional structure of monomeric and multimeric molecules. In this structure, alpha helix and beta strands formed a globular structure together. The folding structure of this kind of protein is initiated by the hydrophobic bond, di-sulphide bond, salt bridge and H-bond. The texture of this structure is not so rigid, as it is fluctuated minutely in continuous manner. The quaternary structure is built up through dimeric and/or multimeric molecules stabilized by the non-covalent bonds. The structural annotation of a protein is key understanding towards the function of a protein. It is also an important thing to know whether the structure of a protein is in its correct conformation or not, if so then is that correct conformation results the efficient function. The pattern of amino acid residue sequences determines the protein structure. Generally, a rough sequential structure of amino acid residues is the key to predict the complex protein structures. However, this experimental prediction is hard to find about the particular function of proteins. The knowledge about the primary structure, i.e., linear amino acid sequence, is not enough as the conformational configuration is getting fluctuated continuously. Recently, a number of techniques have been developed to determine the three-dimensional structure of protein, namely electron microscopy, spectroscopy, X-ray crystallography and nuclear magnetic resonance (NMR). However, there is a wide technical slit between the known sequential structure and the predicted structure that has been found which is also a challenge towards protein structure prediction. Computational method is to resolve the protein structure prediction challenges directly from the amino acid sequences.

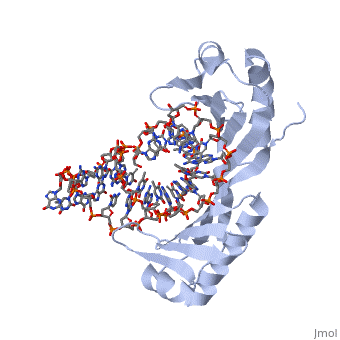


Figure 2: protein 3d structure

**1.1.2 Protein inhibitor**

The ability to identify protein targets for different ligands would have enormous impact on drug discovery, both in the repurposing of known drugs and in the identification of off-targets. Moreover, identifying small molecule binding sites and associating potential ligands with those sites is important in protein function annotation. The methods described in this paper are designed to accomplish these goals. They rely heavily on the use of protein structural alignment to detect relationships not available from sequence alone and use a measure of protein–ligand interaction similarity to determine whether proteins with similar shapes are likely to bind similar ligands. We present results that compare favorably with existing methods using an approach that can be applied on a genome-wide scale.

* 1. **problem statement:**

A protein’s biological function is dictated by the arrangement of the atoms in the three-dimensional structure. This could be the arrangement of catalytic residues in an active site or how a protein interacts with other proteins for structural or other regulatory purposes. Having a protein structure provides a greater level of understanding of how a protein works, which can allow us to create hypotheses about how to affect it, control it, or modify it. For example, knowing a protein’s structure could allow you to design site-directed mutations with the intent of changing function. Or you could predict molecules that bind to a protein.

A protein inhibitor is a molecule that binds to an enzyme and decreases its activity. By binding to proteins' active sites, inhibitors reduce the compatibility of substrate and protein, and this leads to the inhibition of protein-Substrate complexes' formation, preventing the catalysis of reactions and decreasing (at times to zero) the amount of product produced by a reaction. It can be said that as the concentration of protein inhibitors increases, the rate of enzyme activity decreases, and thus, the amount of product produced is inversely proportional to the concentration of inhibitor molecules. Since blocking a protein's activity can kill a pathogen or correct a metabolic imbalance, many drugs are protein inhibitors.

Briefly We will try to solve this problem by:

1. Find the best template for protein in Database
2. Predict protein structure quickly and efficiently
3. finding the conformation of a protein with minimal energy
4. Finding an efficient protein inhibitor
   1. **Objectives section:**

**Do the following using AI and deep learning Algorithms**

1. Search for the sequence entered by the user in the DB.
2. finding the conformation of a protein with minimal energy.
3. Display the 3D protein structure:
   1. If the protein is found in the DB, the 3D structure of the protein will be shown.
   2. If the protein is not found, it will search for templates in the DB.
   3. It will align between the target protein and the template.
4. Search for active sites in protein.
5. Predict protein inhibitor for these active sites.
6. Display the protein inhibitor.

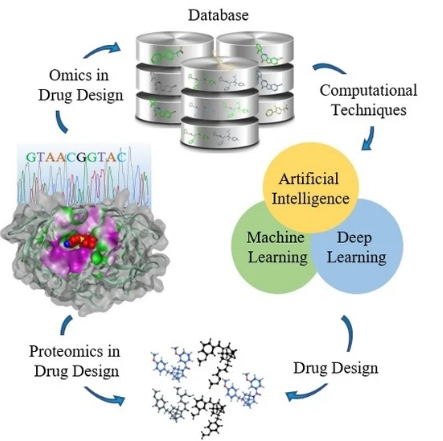


Figure 3: A Structure-Based Drug Discovery Paradigm

* 1. **Project Scope:**

"ST-INPro" is an application that enables its users to obtain protein structure and its inhibitor from its amino acid sequence, by using the technics of Artificial intelligence (AI) and Deep Learning (DL). Firstly, the user will be able to insert protein primary structure, then the system will display the target protein structure and its best inhibitor. All of this will be done with the need of research and paper documents on NCBI and ScienceDirect.

**1.5 Project TimeLine**

|  |  |  |
| --- | --- | --- |
|  | DURATION  (MONTHS) | DESCRIPTION |
| 1 | 3 | PROJECT PLANNING $ ANALYSIS |
| 2 | 3 | IMPLEMENTATION |
| 3 | 1 | Design |
| 4 | 1 | Testing |
| 5 | 1 | Evolution |
|  | TOTAL: | 9 months |

Table 1 project timeline

**Chapter 2**

**Literature Review**

**2.1 Introduction:**

In this chapter, we provide a background about the tools and techniques needed to build our system. We also review the previous related work to our system, and the common technologies that are used.

**2.2 Background:**

**2.2.1** Conventional Approaches for Protein Structure Prediction

Protein structure prediction has a long history, starting from 1894, when Emin Fischer suggested that a unique rigid 3D structure defines a protein’s function. In 1961, Christian Anfinsen proposed that the native conformation has the lowest energy among all possible conformations. Later in 1972, his hypothesis (later known as Anfinsen’s dogma) stating that the native conformation of a protein in a particular environment is determined by its amino acid sequence alone, motivated more bioinformaticians to work on structure prediction problems. Protein structure prediction techniques search for the optimal conformation by applying heuristic algorithms. Template-based and template-free are the two main approaches for protein structure prediction. The method to be chosen depends on the availability of template structures in the protein repositories. Template-based methods use a template structure to build the model, whereas template-free modeling has to start from scratch. Templatebased methods utilize the knowledge of structural similarities with other known proteins. These approaches search for homologous proteins or proteins with similar folds and use them as template/templates for building the target model. Homologous sequences are those sequences that share a common evolutionary ancestry. However, this is not always the case. Some sequences that do not share any significant sequence similarity still have a common ancestor. Template-based methods use homology modeling, comparative modeling and threading (fold recognition) techniques. The terms homology modeling and comparative modeling are used interchangeably as both follow a similar approach to structure prediction. The only difference is that when the query sequence and the templates chosen to share an ancestral history, it is defined as homology modeling. When there is only a shared sequence similarity without any evolutionary relationship, it is called comparative modeling. After finding potential templates, model building techniques are applied to construct the 3D structure. When a reliable evolutionary relationship or significant sequence similarity is not detected, threading techniques are used, as proteins without any evolutionary relationship (but having similar folds) also show structural similarity. Ab initio methods are applied when template-based methods fail to detect a suitable structural template. Template-based approaches can generate high-quality models for those target sequences, showing a high level of sequence similarity to the known structures. Although accurate models can often be generated using TBM, it does not help us understand the physicochemical principle behind protein folding.

**2.2.2 strategies of computational method:**

Three major strategies of computational have been taken to predict the protein structure and those are as follows:

• Homology modelling techniques or comparative techniques.

• Protein threading or protein fold recognition.

• Ab initio or de novo techniques.

Diagram

Description automatically generated

Figure 4: the basic concept of protein structure prediction has been illustrated schematically based on different protein modelling stated earlier

**2.2.2.1 Homology Modeling:**

Homology modeling is the most accurate and successful approach to creating reliable protein structures when a suitable template exists in the PDB. Homology modeling methods search for homologous protein sequences with known 3D structures and choose a sequence with significant sequence similarity to the target as the template/templates. It then aligns the target sequence with the template and starts building the structural model. Multiple sequence alignment techniques (e.g., CLUSTALW, Clustal Omega, and MUSCLE) add more information about structurally conserved regions in the sequence and hence can detect more distantly related templates. Instead of relying on a single template, the information from multiple templates improves the homology modeling technique’s accuracy and efficiency. Homology modeling applies to those targets where sequence identity is above 25%. When sequence identity is less, even one residue’s misalignment will significantly reduce the resultant model’s quality. Among different protein modeling techniques, homology modeling renders high-quality structures

1. Template searching, the first step in homology modeling, compares the query sequence against the sequences of known protein structures using tools like BLAST or FASTA.
2. Template selection chooses the most suitable template/templates from among the results returned by the search methods.
3. Further steps build target structures using the template-target alignment information.
4. From the template-target alignment, one can distinguish two regions: regions well aligned with the template and regions with no alignments.
5. The next step builds the 3D model for those aligned regions by utilizing the template’s structural information.

Diagram

Description automatically generated

Figure 5: homology modeling

Template-dependent modeling can be adopted here. The unaligned regions correspond to the gap in the sequence: modeling these gaps is referred to as gap modeling or loop modeling.

Whenever there is an unavailability of structural information for a segment, those regions apply template-independent modeling; two commonly used approaches are:

1. The fragment database search approaches utilize a library of loop fragments. It selects candidate fragments according to the length, secondary structure and other geometrical constraints. It assesses the ft of the selected fragments by various criteria like RMSD and the number of rigid body clashes.

2. The second method is based on the conformational search among all possible conformations of the segments.

Side chains are the branches added to the backbone of a protein to complete its structure. The feasibility of fitting side chains to the backbone structure relies on the fact that side chain conformations are limited in number. Rotamer libraries are used to obtain side chain conformations. Most times, the initial model generated by the homology modeling method needs refinement. Protein structure refinement aims to improve the quality of the structures predicted by protein modeling methods. When it is difficult to find a suitable template or a good alignment, the comparative modeling pipeline steps (except template searching) are iterated until no more improvements are observed in the resultant model.

**2.2.2.2 Protein Threading:**

Threading modelling is generally used when the template and target sequences share less than 30% identity. Thus, structures that do not share an evolutionary relationship with the target protein can be used as templates. However, the target protein has to adopt a fold similar to that of the protein that has had its structure solved. The method can be classified as a pairwise energy-based method. Using the sequence of the target protein as input, a search is conducted on a database of structures in order to find the best structural match using the criterion of energy calculation. The process is accomplished through a search for solved structures that are most appropriate for the target protein. The comparison highlights secondary structures because they are evolutionarily conserved. A model is constructed by placing aligned residues between the structure of the template and the target residues. In the next step, the energy of this model is calculated. This is done on various structures in the database. In the end, the models obtained are ranked based on the energy. The model presenting the lowest energy constitutes the most compatible folding model

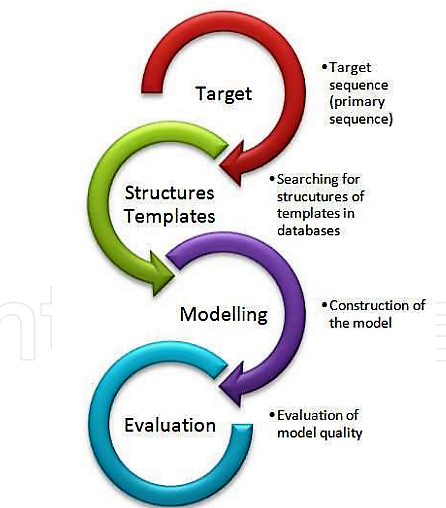


Figure 6: threading modelling

**2.2.2.3 Template-Free Modelling:**

One of the biggest problems in comparative modelling is the lack of templates. Template free methods generate models based on the physicochemical properties and thermodynamic chain of the primary protein target. The processes are iterative. The conformation of the structure is altered until a configuration of lower potential energy is found.

Predicting the nature of protein structure from its amino acid sequences is really a tough job. We have discussed earlier about protein homology modelling and threading. These two popular methods are used for protein structure prediction based on the sequence and/or structural fold compatibility. But if homologs do not exist in the resource or existing homologs cannot be identified, then another way out must be found. In this case, ab initio modelling has been found. It is used to predict comparatively complex protein structure such as tertiary structure. This method requires a large number of computational resources to predict the structure of complex one. This modelling is very useful in the research field of medicine and drug design [26]. In this modelling, a conformational search has been done based on the designed energy function which generates possible compatible conformations (decoy structure) and the most suitable one should be picked up. Ab initio modelling generally depends on three key factors, those are as follows: energy function design, conformational search engine and model selection strategy.

1. Energy function design: A possible decoy structures have been generated but the most suitable one is being chosen on the basis of thermodynamically stabled native protein. An accurate energy function has designed with which that native protein structure contacts in a condition. This is classified into two groups:

(a) physics-based energy function and

(b) knowledge-based energy function.

In case of physics-based energy function, the atom’s alike physical and chemical properties are calculated while knowledge-based energy function statistically solves the protein structure.

1. Conformational search engine: It is an efficient search method. It can quickly identify the low-energy states through conformational search. There are two popular computational methods which are good in search of conformational space those are Monte Carlo (MC) and molecular dynamics (MD). MC and MD both need large computational resources. Another two categories are genetic algorithm and mathematical optimization. MC simulates the random sample parameters to explore the complex structural behavior. MD simulation is being used to understand the movement of atoms of protein. Genetic algorithm solves the natural selection problems by repeatedly modifying the population while mathematical optimization selects the best element set from the pool of available related alternatives.
2. Model selection strategy: That can select near-native models from a pool of decoy structures. It is both energy-based and free energy-based. It helps to select the decoy with the lowest energy. The initial strategy of ab initio modelling is to elucidate the secondary protein structures from its primary structure, i.e., linear amino acid sequences, and then it is followed by the tertiary structure prediction based on physicochemical parameters. Whenever the structure prediction cannot be done by homology modelling or threading, ab initio will resolve the problem generally. However, this modelling has limitation related to the exploration of locations and orientation of amino acid side chains. Another major limitation of this modelling is very time consuming to get a successful solution.

**Diagram

Description automatically generated**

Figure 7: template free modelling

Diagram

Description automatically generated

Figure 8: A generic pipeline for ab initio Protein Structure Prediction, in which evolutionary information in the form of alignments, 1D and 2D PSA are intermediate steps.

**2.2.3 Conventional Approaches for Protein inhibition**

**2.2.3.1 molecular docking**

In molecular docking, a ligand is usually placed in the binding site of a predetermined structure of a receptor. In other words, this is a method based on structure. The receptor is typically a protein, and the ligand is a small molecule or a peptide. The optimal position and orientation of the ligand are determined using a search algorithm and a scoring function that ranks the solutions.

Diagram

Description automatically generated

Figure 9: Diagram illustrating the docking of a ligand to a receptor to produce a complex.

The first step of the process of molecular docking is to determine the binding sites of the protein.

The metaPocket method predicts binding sites using 4 methods: LIGSITEcs, PASS, Q-Sitefinder, and SURFnet – which in combination increase the success rate of prediction. The methods LIGSITEcs, PASS, and SURFnet use only the geometrical characteristics of the protein structure, detecting regions that have the potential to be binding sites. Such methods do not require prior knowledge of the ligands.

In Q-Sitefinder, the surface of the protein is covered with a layer of methyl probes for the calculation of Van der Waals interactions between the protein and the probe. Probes with favourable interaction energies are retained and are classified based on the number of probes per group. The largest and most energetically favourable group is ranked first and considered the best potential binding site.

Another step is to define the position of the ligand in the pocket. This can be predicted by molecular docking algorithms.

The search algorithms must be able to present different configurations and orientations of the ligand in a short time. Search algorithms, such as those used in molecular dynamics, Monte Carlo simulations, and genetic algorithms, among others, are all suitable for molecular docking.

Scoring functions must be able to discriminate between different ligand-receptor interactions. These can be grouped into field-force, empirical, and knowledge-based methods.

The algorithms can be classified into rigid body docking and flexible docking algorithms. In rigid-body docking, both the ligand and receptor are rigid. These methods are faster, but do not allow ligand and receptor to adapt to the binding. In flexible methods, the computational cost is higher compared to rigid methods. However, in these cases, the flexibility of the ligand and/or receptor is considered.

Another important factor to be considered in ligand-receptor interactions is the presence of water. Some methods allow water molecules to be positioned. In cases where this is not possible, the position of water molecules can be predicted using a software program such as GRID.

GRID calculates the interactions between chemical groups and small molecules with known 3-dimensional structures. The energies are calculated using Lennard-Jones interactions, electrostatic and hydrogen bonding between the compounds, and 3-dimensional structures, using a position-dependent dielectric function.

GOLD uses a genetic algorithm that seeks solutions through docking that propagates multiple copies of flexible models of the ligand in the active site of the receptor and recombining segments of copies at random until a converged set of structures is generated.

The process of searching the databases can be time consuming; a way to reduce the search space is filtering databases by performing a search with the fastest algorithms, selecting the best candidates ranked. Subsequently, within this selection, a search algorithm slowly generates a new ranking of the ligands. Another way to reduce the number of ligands being studied in the database is to perform a search for ligands that offer the greatest possibility of being used in drug design. In this case, it is possible to filter the database by using the ADMET (absorption, distribution, metabolism, excretion, and toxicity) filter.

Lipinski´s rule of 5 can be used. The rule of 5 is a set of properties that characterise compounds that exhibit good oral bioavailability. It states that, in general, an orally active drug has no more than 1 violation of the rules (Table 1):

|  |
| --- |
| **Lipinski´s Rule** |
| **Not more than 5 hydrogen bond donors (nitrogen or oxygen atoms with one or more hydrogen atoms** |
| **Not more than 10 hydrogen bond acceptors (nitrogen or oxygen atoms)** |
| **A molecular mass less than 500 Daltons** |
| **An octanol-water partition coefficient log P not greater than 5** |

Table 2: Lipinski’s Rule of Five

**2.2.3.2 Ligand-Based Virtual Screening (LBVS)**

Other methods can also be used for screening databases of compounds, such as those based on ligands (LBSV). In this case, a similarity search can be made between known bioactive compounds and molecules contained in databases. LBVS techniques include methods based on the pharmacophore and quantitative structure-activity relationship (QSAR) modelling.

In pharmacophore-based virtual screening, a hypothetical pharmacophore is taken as a template. The goal of screening is to identify molecules that show chemical similarities to the template.

QSAR is based on the similarity between structures. It is a quantitative relationship between a biological activity and the molecular descriptors that are used to predict the activity. QSAR searches for similarities between known ligands and each structure in a database, investigating how the biological activity of the ligands can be correlated to their structural features.

Diagram

Description automatically generated

Figure 10: inhibition

**2.3 Related works:**

PHYRE2 - Protein Homology/analogy Recognition Engine - this is a site for the prediction of the 3D structure of proteins. In each case bioinformation use this site it provides him with a model. Phyre2 uses the alignment of hidden Markov models via HHsearch to significantly improve accuracy of alignment and detection rate. It also incorporates a new ab initio folding simulation called Poing to model regions of your proteins with no detectable homology.

ORION - is a web server for protein fold recognition and structure prediction using evolutionary hybrid profiles. Various databases such as PDB, SCOP and HOMSTRAD can be mined to find an appropriate structural template. For the modeling step, a protein 3D structure can be directly obtained from the selected template by MODELLER and displayed with global and local quality model estimation measures.

I-TASSER ONLINE - 3D models are built based on multiple-threading alignments by LOMETS and iterative TASSER simulations; function insights are then derived by matching the predicted models with protein function databases. I-TASSER was ranked as the No 1 server for protein structure prediction in recent CASP7 and CASP8 experiments.

ESyPred3D - this automated homology modeling program derives benefit from a new alignment strategy using neural networks. Alignments are obtained by combining, weighting and screening the results of several multiple alignment programs. The final three-dimensional structure is built using the modeling package MODELLER.

Robetta - is a protein structure prediction service that is continually evaluated through CAMEO. It features include an interactive submission interface that allows custom sequence alignments for homology modeling, constraints, local fragments, and more. It can model multi-chain complexes and provides the option for large scale sampling. It uses the PDB100 template database, which is updated weekly, a co-evolution-based model database (MDB), and provides the option for custom templates

(PS)2: protein structure prediction server predicts the three-dimensional structures of protein complexes based on comparative modeling; furthermore, this server examines the coupling between subunits of the predicted complex by combining structural and evolutionary considerations. The predicted complex structure could be indicated and visualized by Java-based 3D graphics viewers and the structural and evolutionary profiles are shown and compared chain-by-chain.

InterProSurf - predicts interacting amino acid residues in proteins that are most likely to interact with other proteins, given the 3D structures of subunits of a protein complex. The prediction method is based on solvent accessible surface area of residues in the isolated subunits, a propensity scale for interface residues and a clustering algorithm to identify surface regions with residues of high interface propensities.

GeNMR (GEnerate NMR structure) - generates 3D protein structures using NOE-derived distance restraints and NMR chemical shifts.

P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure

metaPocket - Identification of cavities on protein surface using multiple computational approaches for drug binding site prediction

POCASA (POcket-CAvity Search Application) is an automatic program that implements the algorithm named Roll which can predict binding sites by detecting pockets and cavities of proteins of known 3D structure.

**Chapter 3**

**Software Engineering**

**Analysis and UML Diagrams**

**3.1 Introduction:**

In this chapter we will provide the system analysis and design process that included in our system that includes the functional and non-functional requirements and UML Diagrams.

There are 4 methods for developing a system:

1. Agile development methodology.
2. DevOps deployment methodology.
3. Waterfall development method.
4. Rapid application development.

In this project we will use.

* 1. **Rapid application development:**

Rapid Application Development (RAD) is a development model that prioritizes rapid prototyping and quick feedback over long drawn out development and testing cycles. With rapid application development, developers can make multiple iterations and updates to a software quickly without starting from scratch each time. This helps ensure that the final outcome is more quality-focused and is in alignment with the end-users’ requirements.



Figure 11: RAD

**In our project we use RAD phased methodology:**

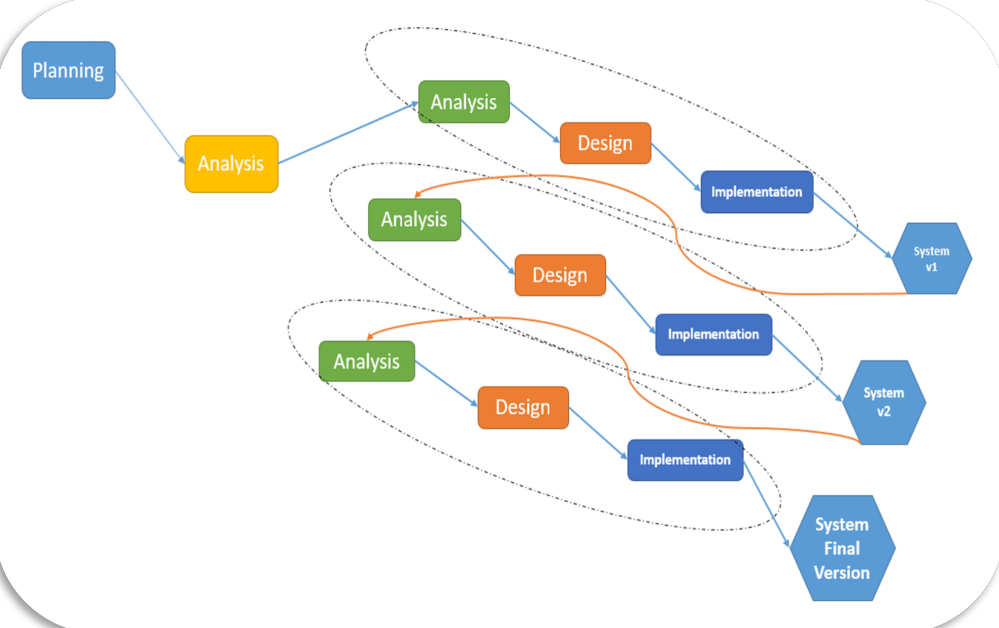
This methodology breaks the overall system into a series of versions that are developed sequentially. The team categorizes the requirements into a series of versions, then the most important and fundamental requirements are bundled into the first version of the system.

The analysis phase then leads into design and implementation; however, only with the set of requirements identified for version 1.

As each version is completed, the team begins work on a new version.

Our project was divided into 4 versions:

1. Homology modelling
2. Threading modelling
3. Templet free modelling
4. Protein inhibitor

****

**3.3 System Analysis:**

Design is the planning that lays the basis for the making of every object or system. It is the process of originating and developing a plan for a product, structure, system, or component with intention. Is used for either the final (solution) plan (e.g., proposal, drawing, model, description) or the result of implementing that plan in the form of the final product of a design process.

The person designing is called a designer, which is also a term used for people who work professionally in one of the various design areas, usually also specifying which area is being dealt with (such as a fashion designer, concept designer or web designer).

Designing often requires a designer to consider the aesthetic, functional, and many other aspects of an object or a process, which usually requires considerable research, thought, modeling, interactive adjustment, and re-design.

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group.

UML includes a set of graphical notation techniques to create visual models of software intensive systems. It is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

• actors

• business processes

• (logical) components

• activities

• programming language statements

• database schemas, and

• reusable software components

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (workflows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML aims to be a standard modeling language which can model concurrent and distributed systems

**3.4 System Overview:**

**Scenario:**

There are many elements that affect and use the system:

* visitor scenarios:
* Insert protein primary sequence (Text or FASTA file).
* Get protein structure.
* Get protein inhibitor.
* Management scenarios
* Update database.
* System development.

**3.5 UMLs work:**

UML (Unified Modeling Language) is a specification language that is used in the software engineering helps to simplify the process of software design, making a model for construction with a number of different views.

➢ Classifications of UML diagrams:

* Behavior diagrams: specify behavioral features of a system, such as Activity, State

Machine, and Use Case diagrams.

Interaction diagrams: A subset of behavior diagrams, such as Communication, Interaction Overview, Sequence and timing diagrams.

* Structure diagrams: that depicts the elements of a specification that are irrespective of time, such as Class, Composite Structure, Component, Deployment, Object, and Package Diagrams.

**➢ Goals of UML:**

The primary goals in the design of the UML were:

* Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
* Provide extensibility and specialization mechanisms to extend the core concepts.
* Be independent of programming languages and development processes.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of the OO tools market.
* Support higher-level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**3.5.1 Use case:**

* Diagram that shows the relationships among actors and use cases within a system and describes a sequence of actions that provide a measurable value to an actor. It is a description of a system’s behavior as it responds to a request that originates from outside of that system. In other words, a use case describes "who" can do "what" with the system in question. The use case technique is used to capture a system's behavioral requirements by detailing scenario-driven threads through the functional requirements.
* Use cases describe the interaction between one or more actors (an actor that is the initiator of the interaction may be referred to as the 'primary actor') and the system itself, represented as a sequence of simple steps. Actors are something or someone which exists outside the system ('black box') under study, and that take part in a sequence of activities in a dialogue with the system to achieve some goal. Actors may be end users, other systems, or hardware devices. Each use case is a complete series of events, described from the point of view of the actor.
* Model the functionality of a system using actors and use cases.
* Use cases are services or functions provided by the system to its users.
* Use cases may be described at the abstract level (business use case, sometimes called essential use case), or at the system level (system use case). The differences between these are the scope.
* Be at the appropriate level of detail.
* Not include detail regarding user interfaces and screens. This is done in user interface design.

**3.5.1.1 use case Diagram**:

Diagram

Description automatically generated

Figure 12: System use case diagram

**3.5.1.2 use case Scenario:**

* **User scenario:**

|  |  |
| --- | --- |
| Use case: | Predict protein structure and inhibitor. |
| Description: | Users use the tool with aim to find protein structure and inhibitor. |
| Actors: | * Bioinformaticians * Alchemists * Pharmacists |
| Success Scenario: | Users get protein structure and inhibitor as output. |

Table 3 use case user scenario

* **Scenarios of admin:**

|  |  |
| --- | --- |
| Use case: | Tool management. |
| Description: | All processes of tool management and privileges of admin. |
| Success Scenario: | Update database.  Tool development |

Table 4 use case admin scenarios

**3.5.2 Sequence:**

➢ Definition:

* An interaction diagram that shows the objects participating in a particular interaction and the messages they exchange arranged in a time sequence.
* Are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario.
* Are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.
* A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

➢ Advantages of sequence diagram:

✓ Are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.

✓ Help you discover architectural, interface and logic problems early.

✓ Sequence diagram editor makes it so easy to edit your sequence diagrams that you could even make the corrections in real time during the meeting and instantly see the result of the changes as you make them.

✓ Documentation. Sequence diagrams can be used to document the dynamic view of the system design at various levels of abstraction, which is often difficult to extract from static diagrams or even the complete source code. The diagrams can abstract much of the implementation detail and provide a high-level view of system behavior.

**3.5.2.1 Diagram:**

Diagram

Description automatically generated

Figure 13: system sequence diagram

**3.5.2.2 scenario:**

|  |  |
| --- | --- |
| Sequence: | Protein sequence map in the tool. |
| Description: | All protein steps in the tool during prediction. |
| Actors: | * Bioinformaticians. * Alchemists. * Pharmacists. * Admin. |
| Success Scenario | * Users insert protein primary sequence (text or FASTA file). * Search in the database if protein structure exists or not. * If it exists, then return it and start predicting its inhibitor. * If it doesn’t exist, then start predicting it. * Search in the database for best templet. * If templet exists, start alignment between it and target. * If templet doesn’t exist, start predicting with templet free. * After alignment, start constructing the target by adding loops and chains in their place. * Return protein 3d structure. * Start predicting its inhibitor. * Find the active site in it. * Start predicting the inhibitor for this active site. * Return the inhibitor. |

Table 5 sequence scenario

**3.5.3 activity:**

* Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.
* Illustrates the dynamic nature of a system by modeling the flow of control from activity to activity.
* Diagrams describe the workflow behavior of a system and can show activities that are conditional or parallel.
* Activity diagrams are constructed from a limited repertoire of shapes, connected with arrows. The most important shape types:
  + - * rounded rectangles represent activities.
      * diamonds represent decisions.
      * bars represent the start (split) or end (join) of concurrent activities.
      * a black circle represents the start (initial state) of the workflow.
      * an encircled black circle represents the end (final state).

✓ Arrows run from the start towards the end and represent the order in which activities happen.

✓ Hence they can be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

➢ Why we use activity diagrams:

✓ To model the workflow behind the system being designed.

✓ Analyzing a use case by describing what actions need to take place.

✓ Describing a Complicated Sequential Algorithm.

✓ Modeling applications with parallel processes.

**3.5.3.1 Diagram:** Diagram

Description automatically generated

Figure 14: activity diagram

**3.5.4 Class Diagram:**

It is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.

**3.5.4.1 diagram:**Diagram

Description automatically generated

Figure 15: class diagram

**3.6 database design:**

➢ Phases of Design Process:

1. Determining data to be stored

* The data to be stored in the database must be determined in cooperation with a person who does have expertise in that domain, and who is aware of what data must be stored within the system.
* This process is one which is considered part of requirements analysis
* Data to be stored can be determined by Requirement Specification.

2. Conceptual schema

* Determine where dependency is within the data. Sometimes when data is changed you can be changing other data that is not visible.

Ex:

In a list of names and addresses, assuming a situation where multiple people can have the same address, but one person cannot have more than one addresses, the name is dependent upon the address, because if the address is different than the associated name is different too. However, the other way around is different. One attribute can change and not another.

3. Logically structuring data

* Once the relationships and dependencies amongst the various pieces of information have been determined, it is possible to arrange the data into a logical structure which can then be mapped into the storage objects supported by the database management system.
* In this case of relation database, the storage object are tables which store data in rows and columns
* Relationships between tables may then be stored as links connecting child tables with parents.
* Complex logical relationships are themselves tables they will have links to more than one parent.
* The relationships may be defined as attributes of the object classes involved or as methods that operate on the object classes.

4. Physical database design

* Specifies the physical configuration of the database on the storage media.
* This includes detailed specification of data elements, data types, indexing options and other parameters residing in the DBMS.
* Includes modules & the database's hardware & software specifications of the system.

**3.6.1 Database design diagram:**

Diagram, schematic

Description automatically generated

Figure 16: database diagram

**➢ Defining Keys**

Keys are used to describe the relation between columns. These keys uniquely identify a record. There are several types of keys:

o Primary key is a column in the table which is used as the unique identifier.

o Composite key is a primary key having more than one attribute

o foreign key is a key in table which is also the Primary Key in another table.

**➢ Normalization:**

Normalization refers to the process of structuring data to minimize duplication and inconsistencies.

There are several types of normalization:

First Normal Form

* The rules for the first normal form are as follows:
* Eliminate repeating information.
* Create separate tables for related data.
* Breaking the flat table into two tables

Second Normal Form

* The rule for the second normal form is as follows:
* No non-key attributes depend on portion of the primary key

Third Normal Form

* The rule for the third normal form is as follows:
* No attributes depend on other non-key attributes.

➢ Relationships:

A relationship works by matching data in key columns — usually columns with the same name in both tables. In most cases, the relationship matches the primary key from one table, which provides a unique identifier for each row, with an entry in the foreign key in the other table. For example, book sales can be associated with the specific titles sold by creating a relationship between the title\_id column in the titles table (the primary key) and the title\_id column in the sales table (the foreign key).

There are three types of relationships between tables. The type of relationship that is created depends on how the related columns are defined.

❖ One-to-Many Relationships

* A one-to-many relationship is the most common type of relationship. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A. For example, the publishers and titles tables have a one-to-many relationship: each publisher produces many titles, but each title comes from only one publisher.
* Make a one-to-many relationship if only one of the related columns is a primary key or has a unique constraint.
* The primary key side of a one-to-many relationship is denoted by a key symbol. The foreign key side of a relationship is denoted by an infinity symbol.

❖ Many-to-Many Relationships

▪ In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa. You create such a relationship by defining a third table, called a junction table, whose primary key consists of the foreign keys from both table A and table B. For example, the authors table and the titles table have a many-to-many relationship that is defined by a one-to-many relationship from each of these tables to the title authors’ table. The primary key of the title authors table is the combination of the au\_id column (the authors table's primary key) and the title\_id column (the titles table's primary key).

❖ One-to-One Relationships

* In a one-to-one relationship, a row in table A can have no more than one matching row in table B, and vice versa. A one-to-one relationship is created if both related columns are primary keys or have unique constraints.
* This type of relationship is not common because most information related in this way would be all in one table. You might use a one-to-one relationship to:
  + Divide a table with many columns.
  + Isolate part of a table for security reasons.
  + Store data that is short-lived and could be easily deleted by simply deleting the table.
  + Store information that applies only to a subset of the main table.

✓ The primary key side of a one-to-one relationship is denoted by a key symbol. The foreign key side is also denoted by a key symbol

**Chapter 4**

**Design**

**4.1 Introduction:**

In this chapter, we will introduce our system design.

**4.2 System architecture:**

Diagram

Description automatically generated

Figure 17: protein structure prediction system architecture

![Diagram

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDcRXhpZgAATU0AKgAAAAgABAE7AAIAAAAGAAAISodpAAQAAAABAAAIUJydAAEAAAAMAAAQyOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAERUUC03AAAFkAMAAgAAABQAABCekAQAAgAAABQAABCykpEAAgAAAAMwNQAAkpIAAgAAAAMwNQAA6hwABwAACAwAAAiSAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxMjoxMDoyOSAwODo1NzoyMwAyMDEyOjEwOjI5IDA4OjU3OjIzAAAARABUAFAALQA3AAAA/+ELGGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMTItMTAtMjlUMDg6NTc6MjMuMDUxPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPkRUUC03PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAzkFEwMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpGiivHfj38SPEngCXw7H4Xe3V9SNwsomhD5KeVtxnp980AexUV87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzt/wkf7Rn/Qv2/8A36g/+Lo/4SP9oz/oX7f/AL9Qf/F0AfRNFfO3/CR/tGf9C/b/APfqD/4uj/hI/wBoz/oX7f8A79Qf/F0AfRNFfO3/AAkf7Rn/AEL9v/36g/8Ai6P+Ej/aM/6F+3/79Qf/ABdAH0TRXzt/wkf7Rn/Qv2//AH6g/wDi6P8AhI/2jP8AoX7f/v1B/wDF0AfRNFfO3/CR/tGf9C/b/wDfqD/4uj/hI/2jP+hft/8Av1B/8XQB9E0V87f8JH+0Z/0L9v8A9+oP/i6P+Ej/AGjP+hft/wDv1B/8XQB9E0V87f8ACR/tGf8AQv2//fqD/wCLo/4SP9oz/oX7f/v1B/8AF0AfRNFfO3/CR/tGf9C/b/8AfqD/AOLo/wCEj/aM/wChft/+/UH/AMXQB9E0V87f8JH+0Z/0L9v/AN+oP/i6P+Ej/aM/6F+3/wC/UH/xdAH0TRXzt/wkf7Rn/Qv2/wD36g/+Lo/4SP8AaM/6F+3/AO/UH/xdAH0TRXzt/wAJH+0Z/wBC/b/9+oP/AIuj/hI/2jP+hft/+/UH/wAXQB9E0V87f8JH+0Z/0L9v/wB+oP8A4uj/AISP9oz/AKF+3/79Qf8AxdAH0TRXzdpPxU+KmnfFHw/4Z8a29rZf2jcQ74fs6bmidyuQVY45Vh+FfSNABRRRQAV87/tT/wDIV8Df9drr/wBCt6+iK+d/2p/+Qr4G/wCu11/6Fb0AfRFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzv8AFX/k6/wL/wBcbT/0pmr6Ir53+Kv/ACdf4F/642n/AKUzV9EUAFFFFABXzv8AtT/8hXwN/wBdrr/0K3r6Ir53/an/AOQr4G/67XX/AKFb0AfRFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRVS51bT7Pi5vIYyP4S4z+XWsufxpo8Odksk2P+ecZ/rispVqcPikkK6N+iua/4TBpT/omj3sw7HbjP5ZpP+Ei1qX/UeH5E/wCujn+oFZfW6PR3+TFzI6aiuZOp+KWXKaRbjPTdIP8A4qg3vi0jAsbIe4P/ANlS+tQ7P7mHMjpqK5g3Xi1SD9msn9s//ZUp1DxYpGdNs3Hfa2P5vR9aj/K/uDmR01Fcz/bHiSI/vtFjYf8ATOT/AOuaP+Ep1CP/AI+PD90o7spJH/oNP63S63XyYcyOmorml8bWSNtu7S7t2/2kGP55/Sr9v4o0e5xtvUQntICmPxPFXHE0ZbSQ+ZGtRTIp4p13QSpIvqjAj9KfW97jCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAK2oalY6RYSX2q3tvY2kWPMuLmVY40yQBlmIAySB9SKxP+Fj+B/8AocvD/wD4NIP/AIqukr4h+PsMVv8AHLxCkEaxput22qMDJt4yT+JJP40AfY+leLPDmvXTWuh6/pepXCIZGis72OZ1UEAsQpJxkgZ9xWvXMfDOGOD4U+FVhjVA2kWrkKMZYwqSfqSSa6egCC+vrXTLCe91C4jtrW3QySzSttVFHJJNZvhfxdoPjPSjqPhnUY7+1VzGzKrIyMOzKwDD15HI5o8XeF7Hxn4Tv/D+qmRbW9QKzRNhkKsGVh9GUHHQ45rmvhT8KbH4XaVew29/JqN5furXFy0flqQm7Yqpk4xubnJJJ+gAB31FFFABRRRQAUUUUAFFFFABRRRQBkar4s8OaDdLa65r+l6bcOgkWK8vY4XZSSAwDEHGQRn2NWdK1vStetWutD1Oz1K3RzG0tncLMisACVJUkZwQce4rx/8AarVT8LtMcqNw1mMA45AMM2f5D8qX9lT/AJJZqX/Yal/9EQUAe20UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzv8Vf8Ak6/wL/1xtP8A0pmr6Ir53+Kv/J1/gX/rjaf+lM1fRFABRRRQAV87/tT/APIV8Df9drr/ANCt6+iK+d/2p/8AkK+Bv+u11/6Fb0AfRFFFFABRRRQAUUUUAFFFFABRRRQAVQ1LWrHSkzeThWIyI15Y/hVDXdZuIbmPS9JUPfTDJY9Il9frTNN8OW1o/wBovCby8blpZecH2B/nXHUrvmcKa1W76Ily7EB13WdUONH07yIT0nue49QP/wBdJ/wjt/fc6xrE0gPWKH5V/wAP0roaKwcHL+I2/wAvuI33Mi38LaRb4/0XzWHeRi36dK0YbS2thi3t4ov9xAP5VNRVRhCPwoBc0ZpKKu4xc0ZpKKLgLmjNJRRcBc0ZpKKLgDAMMMAQexFUp9G025/11jASe4QA/mKu0Umk90IwJfCFjv8AMspZ7SQdDG+cfnz+tNEHiXTOba7j1GIfwTDDH8T/AI10NFZeyitY6egGLbeLoBMINXtpdPmPdwSp/Hr/AJ610EciTRrJE6ujDIZTkH8aqXNrBeQmK6hSVD2YZrnZrW88LSG70x3n0/OZrZznYPUf4/nmtFXqU/j1XfqUpNbnX0VDaXUV9ZxXNu26ORdympq7001dFhRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK+Jf2g/8Aku3iH/t2/wDSaKvtqviX9oP/AJLt4h/7dv8A0mioA+tvhx/ySzwp/wBgWz/9EJXSVzfw4/5JZ4U/7Atn/wCiErV1zV7bQNAvtXvm229lbvPIfZQT+fFAHG/GrVtb8N/Da98Q+G9Zn02608x/u0ghkScSSxp8wkRiMBiRtI6nOeMcn+zp8QfE/jv/AISP/hK9T+3/AGL7L9n/ANHii2b/ADd33FXOdi9c9K2PjVcXN3+zZqVxf+X9qmtrJ5vKUqm8zwltoJJAznAJJ964L9kb/mbv+3L/ANr0AWPib8Y9ev8A4m23gPwNfLpi/borG41BFWR3mdghUZzhVLYOMHKnkVP8VYfF3we8M2GreGPHmu3qXN19muE1d47w7ihYMhdDtHyNxz169c4HxS+HWq+CvizF8Q7Cxe/0FNSi1S6FsAXt2WRXkDDPQkEhunPOOp0PjL44074t+HNL8O/Dm3vtduhfJczmCykVYBsdFVyyjGS556DacmgDv/gnqnibxVoMPibWvGbaxaTxPA+mvpcFubacMuT5kZywABwMDIcEgHiuY174vT6/8aovAthrE+gaTHdPZT39rEhnmuBldgaQEIu/5chSScdjmvQfhB4Dk+Hnw+g0m8dHvppWursxklRIwA2j6KqjjqQT3ri/G/iLwbpvxXtbfwt4Xtde+IE7iNJdxWK3fA+aUg4LKozwMgA8igDifiJ4s8WfBb4m2lnpfinVdb0+e2jvZLfWJxPvDO6smcDaPk424xmvo2G5PijwlbXukahcacNRtoriC6gRGkjVwHGBIjLyDg5XoT0PI+Sv2irHVbH4g6cNf1QalfS6TFJI6QrFHHmWUeXGo52DH8RJJJJPYfU/w4/5JZ4U/wCwLZ/+iEoA+b/hz8bviRrfxA0zTbnUBrCXJkjFkbeCFZG8ttpZ1jDKqthmI5wp4PQ6nxh1X4kfDTxFpGqS+Opr/wDtDzWSCGDyIIihQlPK3MGX5wATz1rhf2fP+S7eHv8At5/9Jpa9I/a5/wCZR/7ff/aFAHuXgfxMvi7wHpPiAiOM3luHlVGyqOMq4z7MCPwrxi3+Iev/ABg+LknhjwxrNxoXhu1jlaa4sWC3FxGh2+Yr4JXLFcAYwDk5NdN8LHu4v2UY5NNXdeJp2otbr6yCWfaPzxXlP7KrAfFTUATydGlA/wC/0NAHRftDaLrnhz4e6bp9xrNzrmjNqMbRXGpMrXcEqxSjYXVQJFYEnJG4FTyc8dR+yp/ySzUv+w1L/wCiIKP2q/8Aklmm/wDYai/9ET0fsqf8ks1L/sNS/wDoiCgD22iiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/ydf4F/642n/pTNX0RXzv8AFX/k6/wL/wBcbT/0pmr6IoAKKKKACvnf9qf/AJCvgb/rtdf+hW9fRFfO/wC1P/yFfA3/AF2uv/QregD6IooooAKKKKACiiigAooooAKKKKAOP0qZT431QTEea2VTPoCOPyA/KumzXGnS31HxDq8ltMYbq3mDRP2zzwfyrRtPEjW0otNeha1nHAkx8j+/+eK8SnU5W1Lu9fmY3OhzRmmRypNGHidXQ9GU5Bp2a6rjFzRmkzRmgBc0ZpM0ZoAXNGaTNGaAFzRmkzRmgBc0ZpM0ZoAXNGaTNGaAFzRmkzRmgBc1FdyRR2cz3OPKCHfn0xzVa/1iy01CbqZQ3aNeWP4Vi+Xf+KHUzo1npgO4L/FL/n8vrWU6iXux1YXL3gV2bw+4Y5CzsF9hhT/Mmukrm/AwxoUwH/Py38lrpK7cL/Aj6Fx2CiiiukoKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAqarqMWkaRdahcAmO3jLkDvjt+NeI6p4w8RajI15/ak9qu75YrZ9ip6D3+pr2nXtLGtaBe6cz+X9piKB/7p7H88V86ajpmu6dqq6RdaXcrdu4jj2xkpIScAq3Qj3oA9U+GHju71+5udG1lxLd28fmxT4AMqZwQR6gkc9816PXm/gr4c3PhC5bWJbtbnUZoPKaLb+7jBIJA7k8Dmu6S5vJYNwgjjdfvBmJGfagC9RWVYa2tzfNY3KCK5C7lAOQ4HXH+FatAFbULqaysJLi2sLjUJUxttrZo1kfJA4MjqvGc8sOAepwK+WPiZ8KPiN44+IuqeIbLwjLb2940flxzX9rvASJE5xKRk7c9e9fWFFAHkPhfXvil4d8J6XosvwsW7OnWkVqJ18QW0e8IoUHbzjgDvXMfE21+M3xE0caPF4Ji0jTGcPNCmq20rzEHKhmLjgEA4A6ivoWigDzv4xaJrWvfC658M+GtGn1K5vViUPHPDHHAI5Y3+YyOpOQpA2g9OccZ4z9nzwR4v+H2o6xb+JvDk9vBqgh2XUd1byJEYhKTvCyFud4A2g89cDmvd6KAPmv4xfEbXfB3x50y5u7V7rRtLhE1rYmYxx3JeNlaQnB+ZWYgcHGwepqWP9rhDKgm8FssZYbmXVNxA7kDyhk+2RW7qGdN+LOvXfxa8O3Gt6NNtXQ7wacby1tIhkspjAbaxygJxklCehFXLXxT8CtTLxaToOl6jKBzFaeF5Hfn6QUAdh8OPiroPxLtbltHW4trq02me1uVAZQ2cMCCQw4PTkdwMivnrQfA3xI8DfGyPVYfDN3r1xbXE032hR5cF2sgdC3mn5UJDk4PI9K9h+Ffw+i0rx3r/jO30aTQbDUolt9N0yZdkkUWEMjOgJ2bnQEKeQMjA6V61QB8vfHvwB488R63YeJP7F+2BrNbd7PS1a4a0CsSFY4DSEl2O4IAOmOMn2f4f3XiTTfhVp0es+FpYb7TrSG1hsILuNpp1RVTed5RIycbtpY4HvxXdUUAfKPwv+E/xC8EfEnSvEOp+Ep57WzMvmR299aFyHidOAZQOrA9R0rt/wBoPwR4v+IOo6Pb+GfDk9xBpYm33Ul1bxpKZRERsDSBuNhB3Ac9Mjmvd6KAPPfgxo2s+HvhnZ+G/E2iTafcWHmAtJNDLHOJJZH+Xy3Y8BgDuA68Z5x45efDjxr8HPib/wAJJ4G0eXW9HDsEhtw0rmF8bonUZfI7MAegJ7ivqWigD53+KumfEz4n+DLBj4VGlwpfIY9JWcTTufLkzNI5ChFX7oHUl+cYGev+Ang3xX4E8M3WleJNPs7eC4uGvFdLvfMjssabGQKVxhCdwfrxg5yPWaKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/wAnX+Bf+uNp/wClM1fRFfO/xV/5Ov8AAv8A1xtP/SmavoigAooooAK+d/2p/wDkK+Bv+u11/wChW9fRFfO/7U//ACFfA3/Xa6/9Ct6APoiiiigAooooAKKKKACiiigAooooA5bRv+Rm1v8A66r/AOzVsXNrBeQmK6iSVD2YZrG0f/kZtb/66r/7NW9Xjw1TT7v8zEwJPC5tnMmjX01mx/gJ3Kf8++ab9r8R2HFzZxXyD+OI4Y/5+ldDRS9kl8LsBgr4ttUbZfWtzav6OmR/j+lXYfEOlT42X0a/7+U/nV9kV12uoYHsRmqM2h6ZPzJYw59VXb/Ki1VbNMNS5HdW82PKnjkz/dcGpawpPCOkuTtjkj/3ZD/XNRf8IhbJ/wAe97dxfRx/gKOeovs/iF2dFRXPDw3drwmu3ijsNx/+KpB4f1NRgeILj8UJ/wDZqPaT/l/FBc6KiudXQNV/i16YfRSf/ZqP+Ed1Akl9fuz/ALuR/wCzUe0n/L+QXOipGYKMsQB6k1z/APwi8kn/AB8aveSf8C/xJoXwdp2cySXMh/2nH+FHPU/l/ELmtLqlhB/rryBfYyDP5VQn8WaTDkLO0pHaND/M4FOi8L6RF/y67z6u7H+uKvwWFpbY+z2sMZHdUANH719kGpj/APCR3t1xpmkTyZ6PL8o/w/WkNh4g1H/j9vUsoj1jg6/mP8a6Gij2bfxSb/ADKsPDen2LCTyzPL18yb5jn6dK1qSitIxjFWigMfwP/wAgOb/r5b+S10lc34H/AOQHN/18t/Ja6SuzC/wI+hpHYKKKK6SgooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoopk0giheQ9EUt+VAEMs5LtHE6IV6sx6fSsm91ufSHEt0VuLXPzsgwyj1GOv0rTto4Wt97bWd+WPqTWfqWhW09tIZAfmHIBwKANpHWWNZI2DIwBVh0INV7i8MSMYo9+04LE4ANc/oWsy2+7SWtmkW0iHlyoc/L0AIPerkF9Hf2baeu5bnbypHKnPX86ANCe0jmty85LtjOSeB9K4TWfFF1oGp2tm0+6ylly+/kqB2B9Oh/Cup1Rb+zsyyTLJgfdIxn8a4UeFLvxq4u7qc2sEcmIkXDNJg4bPoMZGfWgD0jS9VTUoBLCC8fZlGQag1PVRYOzsrKhGDkdKj094dEh+weUIUjH7tR029qxPFWtwC0kyw6HvQBR0i7k1vx9bPZ5MVoGkmcdACpAGfcn9K9GrzLwT4iNrpL20aW8UsbeZsZNpmDZIJb1/+tXaaL4ltdYbygphn5/dsc5x1wfagDZooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKK5q91lYtRjgaO8S5jJMnlQNIMfQcEGgDpT0rmZ/EW7VLO309mnuXk2zW4/hX+I47Y9a5nX/iHePdz6dpTpbmMgG6MZDEEdlYcfX2rnfC3jWKw8WzL4gt4iL9VjSaFjuVl/LhsnP4UAezWt2t15m1SoRscnk++O3SrFYFv4hsMR3FvgwzzLCxySysThR6Yye3rW/QAUUUUAFFJkZxkZ9KWgAormhqlyPGNrYPE8SkOsj7srKQm4YHb17elad7s+3/6SZQuxfs/l5zvy27GO+NvX/GgDSrKvNdgtL62t2dC084g8vOHBPQ/Sl0+S5fWL03EU8alE2B8bAAzDjB6kYJrkfH2qW1rNZXllamW+3/uXGBu2nIbd1C54zjvxQB3N7NJEsSQkK80gjDkZ28E5x36VRF/crqPkySAxxFFd0jyrFiQMnPBzgYGcE+lZmieMdO8U3RsYYmLFCZYnGGiIAPJ7g54I7iuhGn2gkjcQIGjGFI7f40AWaKyr258u/cTTyxBEVoFjPMh53cfxduO2ak0nE3m3cU0zwSnEQkkLcDqeemTn8hQBo0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/wAVf+Tr/Av/AFxtP/Smavoivnf4q/8AJ1/gX/rjaf8ApTNX0RQAUUUUAFfO/wC1P/yFfA3/AF2uv/Qrevoivnf9qf8A5Cvgb/rtdf8AoVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFAHK6R/yM2t/9dV/9mrdrC0j/AJGbW/8Arqv/ALNW5XjQej9X+ZgLRSUVdwFopKKLgLRSUUXAWikoouAtFJRRcBaKSii4C0UlFFwFopKKLgLRSUUXAyfA/wDyA5v+vlv5LXSVzfgf/kBzf9fLfyWukrtwv8CPoax2CiiiukoKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKa6CSNkb7rAg/SnVj+JPEtn4ZsFuLwNJJIdsUKfekP9B70AYtxd6joV2sN3BLJag/LcopZcf7WOh+tSX3imA2pIlUjHrWDa/GK2F4I9Z0trW2c486OTzNv+8MDj6V5lqHimGW9uLn7OJPMkLiFPljUE9BzVKLlsdOGwlfFS5KEHJ+R7d4TKGwuNSmOGu2+TP9xeh/E5ot7nPiXbbbS5iYkE4yBivMrLx7eXnhsvaBISkgt8BhmM4yDt+gOPpWDeNIMXP2mQzNz5gkO8H60mrOzMqtKdGbp1FZrc9k17XJGX7MkErTudqxhDkn0FWvDen3Hh7Sx9vC+bKzSSAHOzJ6Vx3wx8VNqaXCavO015YsESRz95GBwT6twRmvRZll1SIiP5I+m4jrSMzD8Uapay2bb2AKjKsDgisHQvCA1iwTVtcupJIH+aC2Hyhhnhm9j6UzxD4ZEN7HJd3shtN37xCOPbOOcZ6+1b1k2q6iq20dobaFMIZDjYo/2cdePSgDmL7St0+pLYSGE7EbeOSCCcAelcKNY1Dw/qMd/Y3LefA+/bIxZWPQgj3HFe9y6dZWOnGKJQSeXduSx9TXifirw5caz4rt9N0Z4IZLx9rNLIEVff3PsOTQB7roOrRa9oFlqkClUu4VlCn+Ekcj8DkVoVQ0PSYdC0Gy0u1JaK0hWJWPVsDqfr1q/QAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRQTgZNRR3UMsrRI/7xRkqRg49aAJGYIuTUMN7DPcPApKyxjLIwwcev0qlr99FZaeSbhIbhiPIVm5ZvQDvXO+E5LvVLu51m8dixLW0SwcgAHLbs/hjPpQAnxN8PQajoq6oJXhubIhQV5DozAFT+eQf8aw/hr4C024tv+Ei1XbfyTGSOCCZAyQqrlScH7zHb17Cug8aanDbeEJrS8mP2m4YeTGQd7KHBzjqBgd6h+HMyXPhG0igcSPbSTLLEH2tExlZlJHptP60AdFD4X0qC/S6igK+Ud0cIOIkb+8E6A1r1V0+O4igcXjBpDIzZDZGD/npVqgAooryvxB8WJ/C/xSl0bVEgfRg0UZdEPmw741beeu4Ak8AZx9KznUjTs5dTrwuEq4tyjSV3FX+Wi089TroLOX7XCgtZU1BbpnuLryyN8JLYHmfxcFRtzxjoMVf0lTpfh8JJE6lJpFVGJyd0rBeT65HNR2/jLw7dab9vt9YtZLf1V/m+m3rnnpjNY+teOtPuLGO18O41TUbt9kECAgqwOdzA4IA65PpnpRGKWqCvXnJOnJW1/FX/AM9uhyviXxPfp4vN5ZPDbnTpdhixuErLlWySAcEHHHYCvQfCviqx8U6Nb3dtNEtw6fvbYOC0bDhhjrjPf0xXLD4ULfCO61fV7g3srGS78lV2MxOTtyMjvyc5647VW1z4S2Vhpz3nhS4vLW7to9yQ+YXEmOoB+8GPPQ9e1aHIehz6rYW0U8k15Cot13S/OCUHuOteW6jp2q+N7g3nhy1AsbOPyoZLpzGbgFtx28Hpk89OnfNcVZ7da8TaZpVjbl55LhFliYEHYpzJv7gbQc5r6He4W1dLa3ty4RASsYAEadBgfh0HpQB5TZ+FfG3hIHVNPSGSWUDz4bRvNYgdA6sBv6n7vPXFa138StbsrSGW60SGFiuWErOvmehUEcD65rrk1iedZRHHuaRjHFCw2ZbBxhu4wpzx9K8v+JkGr3/h+01hrsPAUAUJbkbdwOctnA5AHvnNAHc6Dc6h410y31KQvp9nKpJZT+8kPQ7M8Kme/U4rrbOzhsLOK1tV2RRLtUZz+tZek20Fn4Is4dBB8lbJDbEdSCoIPPc5z+NQXph8pPsckgg+bBk3ld/GAOc7uvt1oA6Kio4C32WIybg2wZ3dc47+9U9X1SDSrXzrqUxRcgyAZ2nHFAF55Y42VZJFUucKGONx9B60jzxRKzSSogTG4swGM+tYmnz3Go2cF9c2Rm+1QBVXgbQGYhjnpuUqePyqW509vOunEcoR5o5f3Gws5C+jcdcfkKANhHWRA8bB1YZDKcg0tQWfnCzj+0gCXHzAY/pU9ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wAC/wDXG0/9KZq+iK+d/ir/AMnX+Bf+uNp/6UzV9EUAFFFFABXzv+1P/wAhXwN/12uv/Qrevoivnf8Aan/5Cvgb/rtdf+hW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAcppP/ACM2t/8AXVf/AGatzNYWlf8AIza3/wBdV/8AZq2814kXv6v8zAdmjNNzRmquIdmjNNzRmi4Ds0ZpuaM0XAdmjNNzRmi4Ds0ZpuaM0XAdmjNNzRmi4Ds0ZpuaM0XAdmjNNzRmi4Ds0ZpuaM0XAzPA/wDyA5v+vlv5LXSVzfgf/kBzf9fLfyWukr0ML/Aj6G0dgooorpKCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooqC9vbfTrKW7vJVighXc7t0AoAnorzm4+MNlDckJpF1Jbg/wCt3qGI9dv/ANeuhtPiF4XvPJC6tFE8oBCTAoQT2JIwD+NAHS0VFcXdva2j3VzMkUCLuaRmwoHrmuXX4neFjd+Q988YJwJnhYRn8ccfjQB1tFMiljnhWWCRZI3GVdDkMPUGn0AVtQ1G00qwlvdRnS3t4V3PI54H+J9q8+b44eHxqAhWx1BoC2BOEXn325zVL473VxFpOkQKSLWWd2kx0LKo2g/gW/KvF5LuOMxuq4Kcsc9ea6adJShzM+0yrI8PicBLFVW3J3sk7bf5s9d1b4na9cXDyaV5NnbA/IrRB2I9WJ/pXH+J/Hl/r91aG9t41ubeJk3ISEfJzux2NY/9uRta4WX5SMkZosbFLsG9uPT92M9vWsI2T1PlMNOlTrxlXjzRT1WxQvNTurt9k4jAfglVxioHYqhRQGx1PpWje6Q7fvIjsxyAeay0P7zy3H3eo9TXbSlB6R0P0/I8bltVSp4SPJJ6tdf+CRxXYtJw0asrHrzw31qzca/ujKhW3DqMdKivliC4Rg4x6Y/CudeZ/NZI8scEfhU1aabucmd5TQxFVVJNp7adfl3PUfgpdpc+MrhLhgE2eYFP8RGAPy3GveLPV44InhkIDxsQQa+U9E1C70G/srzTgRJFjdx94H7wNexQeLPDGruftviGSxuAuAzQFA34nINYVKbT0R8pmmUVaE4ujBuNlsm9fM3vGOuQmzkXcCSMAetbXhXW4YfDttDKw82KMK4J5BArznV00eOye7t9UOptghZN42qfoO/1ril1vUXvZ7izuHQP8xUHggDr+lTCm53scmXZTiMwlONPRxXXT5fmev8AifxfFZ7gJBg56GvIdV197+7aZYd2DlGZsYx3FZl9fXV+hknlLgc4qIXaC3ZdqndjDHqPpXRCgl8R9flvC9KneWOfM+iTdvv0v/W56l4B+Lctrp95p/iBpLqeFQ1llsvJ2KFvbrk9s9a0j8YNUtrkPd6VbPbZ+ZInYOB7E8E/hXhAvPK1i0dGwVlUk+2ef0rptT1hGhJZwcDArCpFRlofKZ1gqWDxPLR+Fn1Bo+rWmu6Rb6lp0nmW1wm5DjBHqCOxByDV2uJ+EemXml/DqzTUUeKWeSS4WJxgortlQR245/Gu2rI8QKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAI7iIzQPGr7GI4bGcGsJNP1ySeWa5bTSeFRSjPkeu7gr9Bmti4uWQSCIoGjXJ31FpOppqtj56LtKu0brnOGBwaAPEvEn2yy1u6tPEUnm3Q5VhIxGw8qVJ5A/qKoeEY/Fl/qF9D4GnMUUYVrktNsUsc7eSDluteo/FG1t20W1umgjNws+xZSo3BSjEjPpxWT4N8T6P4Y+H8M801qu2IySxINsjPk5ycYJPHWk3ZXZUYuUlGO7OL19fE+h6ks3jRlZ54tsUvmCRXx1GQBgjP61f8AhxqU9rrVxqSRXkVlKnlyXC2++N8EYGOvU9R61y3xI8fXXizWLF2s5LfSbdh+6VwzupILknoDgYFe8xRaJfaHbX0cM01jJCksGMgRIQCAACMcHpWMKyqX5Oh6WIy6pg+V4nRS7a/8Asza+kWlSX6yxOkeSVKMhYD2PIP1rXt5luLaKZAQsiBwD1AIzVK40nS9XkhvJ4EuMAMjZOGHUZHQ/jWjW55YVzN98O/C2qa9daxqekx3d5doEladmZcAKowpOFOFHIGfzrpqKTSe5cKk6bvBteh5b4o+G8GjWq33g6wlcq582zV9+FOTuQMc9eNoPToPXP8AhDaXN74t1LVZUEUVvbCDYw+Zmdg2fbAT9a9iry/RriHwx8XdVt7yVILa+U4kkPAYkSKCeAM73H5ChK2iFKTk3KTu2ejzXnl3HlRwvKVAaQpj5Ac4OO/Q8CiG8L3TQSwNC20upYg7lBxnjp1FJLamWfz4Lhow6hX2gHeoyRg9up5rO1JV03T7u51DUIwkhHzSLt/izsB78cAf/Xpknmnie5Xwz8XLTWbYpNFInmZQg7gyMhBPbJA5PtXWf8LC0O41PDz3Wn702iV4siZRk4GMlTyea888QXt3f6ldXcWm3ssN/KxtHW2ZhIucKqkDB7Cs7w1a6fp3ii1/4STzzaWrGR7SSH5w56b0J+4OvAPT60Aex395Yxm4vrJ5DOWUxtDEZQZACAvAwcg9jWB4wjhHgeO3uLWKNpGW3QHOYpAdzH0wMMAP1rcm+JXhq1DATy+WqbkdYGCNxwozjB7c4FY0eiar4/uHvNTM2j6MX329qEAllOCBIc/dzn05/UgFb4e+Lrm31Ox8KXnlzQrbsLe4LHeNn8J7HjpjHSvUK80f4PxRWTm31u4e83bleeNfKwBwNq4I+oNZ41vxromny2d3MY1tcYuHVZcjHAEh4Yc9xu9aAPStVMUZimvVDWcYbzQRkA8bSR+fPbNeQ+KPFE1/e33k3gitcfZ0Q4kHljvnuTyc+9WtOXxP8S4p57rUorfTLWQxymPqxUBiFj6ZwR8zE89uMVuJ8PvC50VpjBFdyQRecWluZAzEgHD4OHHp0HUUAdN4AuWvPAumzv8Axxkg88jceeefzroiQqknoOa81tdTuPAOvtp0kj3Hh5pdiMRua2JGQPp7enI5yD2dl4r0LU7prS01KB5gCTGxKkgdSM4yPpQBZ/tGYWEl2bQtGIvNjCuMsMZwc9D+dEepiTUfswEQxgEGTD52huFx05HfPtUcSWgdrEzynzYSsSMwwExyF+nHXJ6VYGnRC5Eu+TAYPsyNpYLtDdM9B64oAt0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/wCTr/Av/XG0/wDSmavoivnf4q/8nX+Bf+uNp/6UzV9EUAFFFFABXzv+1P8A8hXwN/12uv8A0K3r6Ir53/an/wCQr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFAHJ6V/yM2t/9dV/rW1msXS/+Rm1r/rqv9a2q8FPf1f5nO9zifi34v1HwR8P7jWdHSBrpJoo189SygM3PAI7V5zpXjf46avo1vrGm+GtKurG4j82JwYxvX12+cG/DGa6j9oz/kj9z/19wf8AoVdF8I/+SReG/wDryX+Zrqi1GnzWvqVsrnP/AAy+NNv411WXQddsP7H12IsFgZjtmK/fADAFWGDlTk4BOeDj1FnVF3MQoHcmvm34kKNH/ap8OXWl4inu5bJptgxvLymJgfqgA/Gu5+IvgHw7rHid9W+IPjaaz0p/L+x6W90kEaMq4bG7O7dyeAGGTz0wThG6a0TQNI9ZV1cZRgw9Qc15/wDFD4sQfDOXSUuNJk1D+0GfJScR+Wibdx5B3H5hgcfUV5F4Sm07wZ+0Dpmk+AvEEuo6BqIxNGJxIhLK3ynHBKlVIOM9vXNL9pLwpa6J4vtdbguJpJ9bMjzI+NsZjWNRtwM9D3qoUo+0UXsxqOp9V5prSojBWdVJ7E1w/g74YW3gTwvq+k6Bq14JNRDFLqYKXgcoVDKAAOODz6V5drHwv+GdnaXsfiP4gvd+ITH+8uJr+Mssqr/cOTg8DDMTgDBFZRjFu1/wJSR9F5rwb9m3xPrniG88TPr+sXuo+WLdkF1cNIsZYy7toJwoOB0x0HpWt+zZ4g1LWfA1/bapdy3f2G72QPM25lRkB25POAc4+uOleRfBXwNdePZtZ05tbudM0hFha+iteHuifMCLk8YHzE5BHTjoRtGCipxk9rFW3Pr9XVxlGDD1BzQzqi7mIUDuTXzR4ft7z4NfH+y8MW+pTT6JqzRqEkyd4lyiEgDAcSDGR2/TvviL4B8O6x4nfVviD42ms9Kfy/selvdJBGjKuGxuzu3cngBhk89MZOmk1roxWPWVdXGUYMPUHNKWCgliAB3Jr5h8JTad4M/aB0zSfAXiCXUdA1EYmjE4kQllb5TjglSqkHGe3rnU+N+oLJ8XtF0rxneX1r4PNuski227bIctuOADuO4ID1IHTGar2PvJX6XDl1PolXV13KQwPcGhnVF3MQoHcmvPvhf4R8J6EbzU/AetTX2nXyqGt/tIljicdwMZVj3Dc/yryOfVtG+JvxX1sfEPxIdN8P6XI0VlYPdCJJCrFARnjPBYnr8wGcCpjT5m7PRCSPp5WDLlSCD3BozXzd4N1Wx8E/HWx8PeB9fOq+GNWT54fPEqROwfgEcBgVHPXaec9a+kampHkYNWM3wP/wAgOb/r5b+S10lc34G/5Ac3/Xy38lrpK78L/Aj6GsdgqO4nitbaW4uJBHDEheR26KoGST+FSVk+KtC/4SfwpqOiG8lslv4GgaeEAsqtweD6jIPsTXSUQeG/G/hrxfB5vhzWrS/4yY0fEij/AGkOGH4it2vkPxJ+zd428NT/AG3wzcR6zHEdyPav5FwuO+wnr/usTVPRfjl8SfAd4NO18y3qxcPaazCwmUf75w+eO5I9qAPsiivGfCn7TXg/WtkOvxXGg3LcEyjzoc+zqMj8VA969b03VdP1myW80i+t762f7s1tKsiH8QcUAWyQASTgDvUFpe2t/CZrG5huYgxXfDIHXI6jI71n+LNEfxJ4P1XRYrprSS/tZIFnX+AsuAT6j1HpmvkTwF4x1z4H/EW60vXYJRZmUQ6lZg5yP4ZY+xIByD/EDjuCAD7Soqvp+oWmq6db3+nXCXNpcxiSGaM5V1IyCKsUAFFFFABRRRQAUUUUAFecfFHX7C50SHTbK/hlna5UyxxSBiFUE8498Vf+Ker3GnaBb2ts7RC9lMcjqcHaBkjPv/LNeIate2VsqCO5jSZedhOPwppN7FRjKTtFXLk8UUU8WSG3DPXNN1B4DB8vORyCK5yTWJLxlS2UtIDxyMD8atmy1CcotzKkELEB5FBcqO5C8Z/OkJprc6W08SX974JtNOupne1trySNCx6gKpVT643HH4elUNRuITD8oxxzk966XU38KQ+C7bQfDiS3zKPOe9fKMkp6kjH3vUdMcVx+naDJrGvW2l3WqJZpcMEWV16n0z0BPagR6N8Ftbvzbvpx3SWrvIIC+dqlcHj8zx7V6Ve61NpUga/jRrfOGePOU98dxWIvg+Dw9pllHpLGFbAh4x/eI5JY9885+tZGteKrLWrN4re/skujlTbvcKGyO1Fi4wlO/Kr2LHjzxL4Q1fTJtE1Rri8UkN5lkoYwuOjBiQMj8euK+fZtPjttRlQ3P2mBWPlEoVYrngkdjjtXXzSi1SSC4j2SqcEH+Guanmjkvnbg44rooN81j6nherVljFR5vcs210KU1vFKMABT2A4rUsdT8uGGKbIAwAexxSWsFvdagF5MS84bqa2b63t1tgBsII6AdKddxva2p1cU18L7X2EYe+tW1pv0fcZdakhtwvy8dx1rkZ5nmu5JIAWA4470rxySXcsQdjGuNo3YA/GnJaNHbPJBKvmKf9WeN30opU3fmIyDKa3PDGt+7ra2/VGfc3NyVI8hx9RSaLCHnaSXBNJP9suX8oREHvyK1tM0y3a2toImkW/aRmmJHyLHgYA9TwfzrZuzuz6SrWVKqq9RNxju2ti4PKEi5bgjnjpVLUDHhwhyueCRiuiuNFhitgdrDI4bPWuL1OR7e78lz8g53e1CrRloRS4kwuMbpQTT6Xtr9xPcXD2+0xuyB8BwD94VasZHuLkWv8QOPqKp6fp91rVwskabbdGGXY9T9K6FNC/s25+1pK0kgGCp4FTKoo7HLjc5p4NzcH7z2VvLd/h/wxck0jyoNwcgkYyOhrnLsRRuySLtZf7pxmugl1jzbXC7yFz8uOh71zM/k3Ury3aOFbhDkisqc5uVmeLk2aY/EYvknLni0732Xn5a6GcHiWd5F6KpGSc8mtnwozR+KNAlZk5v4c+byuDIOoPbFRaZo93rupx6XoFgbi4cgDYvCA/xMew9zXryfAVbLTYJ7nWpftsJVwYEARWBzxnnr34q5yitGenmOJwlGLp12m7OytezfU9f1bxBpWhqp1W+itt/3VY5Y/QDmnaTruma5A0uk3sV0qnDbG5X6jqK8E1Oe6OpznXLh7m8A2tK2MtgYH4VV8La6+heOrC9idkt9/l3ZUEgxHrkDrjgj3Fch+eH0pRWTovijRvEBddJv455EGWj5VwPXacHFa1ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAMkk2YCruY9BnFVYtVtmQmdvsxUkETfKPqCeCPel1KxkvYQLe5a2mXO2QLu69iO9YWp+E9Qv8AQpbD+37g+YuG8yJSrc5xjqBxjrQBzPjfxhp0sdw2m3rwTQIVMg+7P7L64Oea6DQzpn/CNWbade2scT4lWdpsM75yd2e4PBz+leO6vNeaZ59lf2kkUts2HUpnB7EHuD2xW/4d+Durarpa3uqan/ZRuP3i232cSOgP97JGMjBx2oA3PGmuQ3lt/Y9jLHdGGY3Etyj7l3EHKjH+9+HSrXw91DS9X0dvDt5BFHNZxBWV1VluUJPzYI6gnn8K4fxl4Vn+HskEq6g+oQXURBLKFZGXr8o7cirmieB/EF7o630k1nbm4BmhhlBYlFwDuYdDkjAGaTaW5cISk/dLep/ACa41lmg19INILAlWiJmRO6g52/j+ldPqGoGxt7XRPC8sM9i+y0tyH3bCAFxkdcYrnNV0TxV4d0l57+b/AIl7ACYwXLkYbsyHtnAJrjNYvo0CS2skkc0ZBjEUhHz9iAO/pUQpwhflW504nG4jFKKrSvy7H0PpkB06zs7WOZ3toYyheRNpOAME+nOa1QQQCDkHuK890S88ZpYWdxr2lTSr5f74CRWcnB58sfhXVeGYr2PTZG1CMwmWZnjhbrGhxgH8cnHvWhxmxRRRQBBNeww7gWBZRuIHYepPQD3NeIXE/wDbHiqQanaXj2d/eOizwRNIrjdtAV8YKjgZ6cV6zceHG1G8H9rXX2mxi/1dqFKhz28w5+bHp0rcVFRAqKFUDAAGAKAPItf8R6n4USw0rw3qEt5poX7O9zIgZoe4G/pkL0GM4FV/Ct0/jHxmul+ILq41K1t7d50R2woO4D5sDnO79MdOK9X1XRNO1vTnsdTtUmt3O7b0w3ZgRyD71zHhXwzZ+DdduUtInEF3+7aeY5IKnKDd7hyMeq+9AHZ+X5dv5duqptXaigYC4HA+lcqdJa7u7ma70CJ7aPaywTbGaSQH7yenHuM5rraKAPAfFmk6ld+MbXRZbJdLh1KVEhZ2XaiMcH7ueeox617Prs09vp0NvaswaV1R5FyWWMEFyAOSduelcL8XC0epaNdaZtOpWjGYAZBwrKVyegGQw/GtrwP42j8cxX1lqVh9lvrFlaVEYlCrZ2srdQflIP8A9epb6dzanF/xF9m3UhvbuER+TBc/6HKhFo/myeWJx947s52jg85wcgZOax/Glzqkfghob6YCOXKb4UyJHyRkN26E4Iz6Z6V6kIY1ZGVACilVx2BxkfoPyriviKyL4Zu7VvkgVoiEU43l3JJP0xnj8aUY8pVat7RJW/G/6ff3ZgfDvxjo1n4VsfD8dtNFfOZI3Kxkq75OXyMkk8cYzntiuoOjajOtvFBYw20SRfPM7fPJhcBNv8I6cdvrXkWh6hZ+HfFunXlsZCq+YXKuoYKwIZhu4Jr2PSviBoF5ZK1xqKQTKvzrMhQkj07H6DmrOcofES0I0G6ulUwNHtIYzHbMc/d2ZxuPTdjp3rl/A93qMlteTr4YM0PlOyahMoRSBwVUty3Q8j8aPG+vjxpq+naFpUkkNo7lpppF2bVAJd+eyoDj3Ndx9v0XQtDnEtvd21rBb7RHMDnyQMZXJ4HT0OSO5FS5JbmtOlUqfArkNlpkugyC8nlF9LBaPK+5iPLGMkL25IxzzhavXOs3Okx7r+a3lM8Blh48oK+5FCEk8gmQc8dDWV4Z8XaR4ja8Fjqckdwqr58UsIVlwMYQEEMvbufzFaOn2i20gLRKpe4EEkTAMCuzcOucHnoCB7VPxq8GdCSw8nTxEHfqv62+W5saPdvf6JZXcrRtJPAjuYjldxUE49s1cqOCCK1t44LeNY4o1CoijAUDoKkq1e2pyzcXNuOwUUUUyAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8AJ1/gX/rjaf8ApTNX0RXzv8Vf+Tr/AAL/ANcbT/0pmr6IoAKKKKACvnf9qf8A5Cvgb/rtdf8AoVvX0RXzv+1P/wAhXwN/12uv/QregD6IooooAKKKKACiiigAooooAKKKKAOS0v8A5GbWv+uq/wBa2qxdM/5GbWv+uq/1rZr5++r9X+ZzPc8u/aJVm+D12VUkLdQFiB0G/HP4kVwvgn4/aboXgfS/D9p4f1LUdTtLYRIkW0LKwyeCMtj/AICfpX0XRW8asVDlkrjUlazPBPhz4O8UeMfii/xG8bWUmmxRsWtLSVSrEhdigI3Kqo5ycZPPc1heKpdL0H9o2+1D4o6fPeaNPEfsDSxmWIDaoU7ejKPnBXBwxzjvX0xTXRJBh1DY9Rmmq/vXa6WHzHzBaSadqH7QfhTVfDvh19E0W6YCzDWywC5Cht0qoOxyOe4FdJ+1Lo95caXoOrwQtJaWbzRXDqCfLMmzYT6A7SM+uB3r3zoMCggEEEZB6g0e399StsHNrc8xh8eP8UfhZ4lm8GWd/a3sNu0EayFVdpCmSEKk84OB0OT2ryb4ea34BsPBT6LqfhCXV/F7zSRi3awE0krkkLtLZ2BRjI4wQTg19TKqoMIoUegGKBGgfeEUMf4sc0o1lFNJaeouY8R/Zit5bTw74ht7hDHLDqAjdT/CwTBH515x8GfF998O4tY1m70G7vtEmdLa7ntgC9vKgZlypIwMMc5wOnPavrekIBBBGQeoNV7dNyutx82582+Gri9+MXx+tfFUGmyQaHpBQq8y/dEYLIpPILl23YHQfTJi8VS6XoP7Rt9qHxR0+e80aeI/YGljMsQG1Qp29GUfOCuDhjnHevpZVVF2ooUegGKHRJBh1DY9Rmj2+u2lrBzHzBaSadqH7QfhTVfDvh19E0W6YCzDWywC5Cht0qoOxyOe4Feq/EP4h+ENH1l/Dfj7RJ57OSJZobh7YTQyHuB3DA+mfqK9L6DApGRXGHUMPQjNTKspNNrbzFzXPnr4EaZ5vxQ13WfC1ne2XhJ4GjgFzuxIdy7QCchiNr9yVBwTzzjfZtN+FfxW1v8A4T/wumraFqkxe0vJbNJ1j3OWBXfxnBIYA7vlzg8Z+nwAoAUAAdAKRlVxh1DD0IzVfWLyba0Y+Y8j8F+Mvh1r/ji0sPBfgqMTrudtUh0mKFLX5GIywG5Sdu0dMk16/TFRUGEUKPQDFOrGclJ6Etmf4G/5Ac3/AF8t/Ja6Sub8Df8AIDm/6+W/ktdJXqYX+BH0N47BRRRXSUFZ2teHtH8R2RtNe0y11CDslzEH2+4z0PuOa0aKAPDPFf7LnhvU98/ha/uNFmPIhkzPD9Bk7h9dx+leSal8Kvil8MrxtQ0dLtkTreaJMzgj/aUYfHrlcV9nUE4GTQB8neFv2oPFGkOtv4qsLfWYlO1pFH2ecfXA2nHptH1rX+IevfD/AONmgpdaRqKaR4rs4z9nh1ICD7SvXyTJnYcnO35up7AmvUvHtp8JdYV/+E1udBW5wR532lI7kY9ChDnHpyPavmPx74e+HWnNJL4G8YXGoMOfsdxZvx7CbaoP/fP40Add8Bvi2/gzVv8AhFfE8rR6PcylYpJTj7FMTg5z0Qnr6HnjmvrYHIyK/Nyvp79nj4wfb4YPBXia4/0qNdum3Mh/1qgf6lj/AHgPu+o46gZAPoWiiigAoorH8ReLNF8KWguNdvo7YNnYnLPJj+6o5NNJt2RUYym+WKuy/qWo2mkabPf6jOtva26F5JG6KK8f1j9oi1iaWPRNEkmxkRz3UuwE+pQAnH4iuU+Jfxfg8aaUmj6ZYz2tstwJGkmcbpQAcDaOnJB6npXmsLxsSZs4weg6elenh8JFq9Tc+kwGVQcObELXt/wx2PiD4u694ksHstVjsXj3h4miiKPC3qDn0JGDXIExyRkszGUt35z6nNZ9wwMyhRkk44p/7y3G5zuX19K7IRhTb5UevRp0qEpckbd7HaeGfCVzc6Hc+JdgFlYyqhyOWYnGR7DI5962ri8iNsAAMj+LNdb8F9ctLzwJPpFxbefGJnikQplZA/OPyNLq3wZllmaXTdQntbRjnyplDMo9Aev514+Jm51XfofJZjWlVxMr9NF6Hnmk6kIb67CkbSAcEZGeagvbk3d/bwW5zK0q4x2wc5/CvV4fAXhebw9HpaWLfbYHKtOJCs0jH+IkdQfToMVoaf8AB/StIiE+6SWbq5ds7h12564+mK5zzznfHvja7uPC2laZFK8b3MW65cHBZVwAM+/U15jcwxlFVCHLDoB09q9i+JtgviTw6ksUC2l7pLdQP3bxtgFc9ugx9Md686j0aKK08wgsRwXz3rqp1YwhY+5yrPMFl+XxpuLc23e35u/kczdXNzGiCa4kkVBt5PIFRJa3T7mjRh33tnD/AI1pfZ4jrMUMoDITnnvjtXR3JgFsNp56EY6UpVIp3ijPF51haM4VMHSWqve1mtdVp107/fc4mOTUbC8DyRqoH3vnByPbFXn1ia7j2wROWPrxTSYjfSIxAUYI9qa8YeX/AEdxuU8MtaezjUSkepHKsLnNKGMbab3V1fTTt9xWkEtpGzMdzNy2Kk08rMokuGYRsc7V64puoieRHZmDMeSfWqukQXV3KsETLjOCT/CO5rTSGnQ9hypZfaFmqcV9xr2y26ud2VUnkgZNRx3yWV2sp6dDWrLoCQW/EjliM5NcxNp95dXTwjiJD80nt9Kj2kai5UcDzfCZtSlg6Sd2trW+a/4J0V1rsRhxv57CsaSPJE91EpMrYVXXJA9aux3E0MIxEsuwbQ+Buqk6z302+dvLVenNc8ac1NaHyOCynMKOPhH2bVnv0t1121Re0jU4obsxuQpIwK1dR1ZGhyzj5VwK5U20dpN5yt5o7gnNXVsXuYFuILdmQ8rk8H8DWlWnd8zZ6mdZTKrXeJnVjG/STtsrKzOh0yCOPRhJIcSSsXKkdj0rmtaZI5WUY2sCcVr2sPiC+sfPt9HunhGQZgo8sY65bOBVK30YvqcNzreyW1Rwz2yOVMo/u7scZrCL5Xc+YwVaWExKm3a2/mu3me2fDy0s/CHwxs9SFn5D3cC3dzL1aX0BJ9ug6c1h6x488RamzzQXZsrfPyxRKDgdskjk1vXPi/TPFng+6srKP7FIkPkm2OP3XHy4x1XivKNRv5tMke11CJ7eZOCjjr7j1qW7s4qs3Um5vqdf4YubHxI2p2GuW4nv9qyRSouDtHBxjkHJHSsU6b/YdxLb3UUoG44d1wXHbmun+DOg3X9t3PiLVImtbcwGC1SYbWlyQWbB7YAHvmup+IcdldaVMjbehww6g+opGZ5VoOoSWvj/AEJ7ORkle/hiO0/eR3CsD7EE19M15z4H+E2n+H7+HWr+9fU75V3QFkCRw5HULk5OD1Jr0agAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDkvGWl2F5DLdXNr508IiCSAH5AHLHp+dF34vtrLTtS1CyklvFt0LLF5bMGYDIAYDgn0qzrcOuyLNZ6faxyxTE7ZmkChM/3h149s1uafa/YdNtrUvvMMSxl8feIGM0nsVFpSTauj5S8deK9W8Z3cd9fSwoIwVihiG0RKTkjPU/j6V9KeGpbDxP4U0/V2sUj+3QJMyY+63t+NcjqXwH8N6jrjXou763tpXLy2cTjaSSSQGIyo9vyxXpNpaQWFlDaWkaxQQRrHHGvRVAwB+VclClUi37R3PoM1zDCV401hIuNvlby+8fLBFPbvBNGskTqVZGGQwPYivPtP8AAmlaP4z067RUkTzZ1htzGNsRGSpBzk4AxzXob7vLbZ97Bx9a42+1YXWp2NrpltINQSZd+YyCgz85Y+mM12Hzp2dFFFABRRRQAUUUUAFZd9opv7yJ5LyVbZZFla3UDDsvI56gZA49q1KKACo5riKBcyttHrgnH+FSVl6tozaoPL+2ywQPgTxoB+8HpntQBwHi3QNd8T6tPq3h9LeS2ESwr5kuzfjOSDjBAzyenp0NYWjweKvAN3LOLBHa4jEt0Ix5wZV/vYG5cbsZGRXtwRRH5YUbAMbccYqnZaNZ2F1Lc26MZpRtLu5Yheu0Z6D2FAHFr8V4fscVwdGnMZ4lZZVIU/7Jx83HrijRtPuviBOniLWgq6YAf7O09jlW7b5B35HTv9MCl+IOlalD4WmsdGhtPs00jPuckMjHLE4xzz0+orrPC+m2+keFNNsbOQyQw26BZCc78jJb8SSaAMbQPBdhZald3MuhafaxSKFSEKJSTkktk/dBz0FM8U+AtNv9OuLnSbOC01JF3xsh8tHI7MBxg88475rsa4v4r2mr3ngK4i0KOWWUSo00UIJeSIHkLjnrg/QGonJxi5JXOjDUo1q0aUpcqbSu+nmeKjULyZxr0FpJdaVptwkV1Mh2jLKRgH6Hr0yVz1rt7vXbr4lQR6DGsVhPfjYk8o4WBWWQnaTkyEouFHGMnPpyHw8sdWnu9QguLK6ttDv7do55poJFjEikFGU7SN4bHXjBNd94a8It4c1qx1TV9Rjub24Jhs1gX93Fu4LnPU7d2O3NY0/31O80enjF/ZuMccNJNLVPR/09CXwB8Jr3QNfk1bxNfW168cRhtreHLIoOMs24DngYAr01LO2jkV47eJXRdqsEAIHoKz7y5ktLfedQDNG20LsX5+md3Hv2x+ZxTbbXoAJTdSyMWuDHEi2sm/GwNjZjd6nJFaxjCmuVHBXq4jGVHVnq32X+RsUUA5GaK0OMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/ydf4F/642n/pTNX0RXzv8AFX/k6/wL/wBcbT/0pmr6IoAKKKKACvnf9qf/AJCvgb/rtdf+hW9fRFfO/wC1P/yFfA3/AF2uv/QregD6IooooAKKKKACiiigAooooAKKKKAOR03/AJGbWv8Arqv9a2Kxbg/2N4unNwdsF+AySHoG9D+OfzFbNfOSvGck+7OaW4tFJRSuIWikoouAtFJRRcBaKSii4C0UlFFwFopKKLgLRSUUXAWikoouAtFJVTU9Qj02yeaRvmxhF7s3YUOVgG+Bv+QHN/18t/Ja6SsXwnYyWGgRrOCskzGUqeozjH6AVtV7mGTjRin2OmOwUUUV0DCiiigArjvir4T1Pxt8PL7Q9DvUs7qcowMjFUkVWBKMRyAfoa7GigD5R0z9lPxTOQdV1rSrND/zx8yZh+G1R+tdnpf7KHh6DadZ8Q6jeEdRbRpAD+e8/rXvdFAHm+l/AD4caXtb+wftkg/ju7iSTP8AwHO39K7PS/C3h/RMf2NoenWBXoba1SM/mBWrRQAUUUUAFfJfxD1ibWfiPq0uos+2C5a3jXr5caHAAH4Z+pr60rxT4qfBrUNc1ubXvCbRNPcYNzZyvs3Nj76seOe4OOea68JUjTneR6uV4inh6zlU6rc8CvGUAlT06GoZJWaNZUYBmALA9zXo+lfALxtrEpGora6RCOrzyiRj9FTP6kVy2s6DY6R4ivNLsp2nSzkMBnnwvmMvDMB2BIOB9K9GNRVZtRPoaeIjiarjTfQyreNUhLsdznnNXra2a+ZYkdQ7EAKe9Z7IQxVGAHoalsNQbTdRhllG5FPLD0racnCDsjqq1JUqUuVapfie7fB+1i0C6uYDKxWYK/zdCw4OK9Xv9ShFsfmHSvmmHxYtoY7i1uQrLyuG4NekaAvirxTosV4bZbC3mbaklwSC4/vBcZxXz0pOTuz4Kc5Tk5Sd2zU0/UEPjtJEGY44nMh7DoAT+JrtrzWIRbnDDpWVpvgu10TTZi073N1OP30zDG72A7CsO20/TvMnbWNYkgggfBR2VFI7fMaRKTbsjR1SyTUPCF7EXCSXqlkY9sfd/l+teF3mptZO9vdqYpYzhlb1/rXq3ivxDpltoU11pWrxXSphI4o2Vsk8AcH/ADivHL2/1C6jkEty2yU7mQdDWsKUpnu5dkWKx6cl7sVo2/0Ri3epF79Jlk8vyzkZHJq+dTvJ7fKxEAjgk4qgGaSVo5UV2QZBNaSXMUllkL82cZz09RiuhUYtan1dDhvBTgoVJPTqtG+990NsViVGadiWflmA79hUQuFivRj+LipV0bUXQyxBERuQrnmqdroWpX18wYpEI+rZziq9pCKsuh1xzjAYWkqNGStD+vmTX92r7mwq57KMAVo+G4/K0J70KoPnk9OWXpn35qnJ4bWKcfb7tjD32Lj8znpVyS/NgotIYyURcIEHQVhUqc+iPmc6zaGYqNKgm/kXbzWka3GZMgDjnpWTbrM5llR2XzRyF9KrQxfa7lpZECDoFx+tWo2uIQVXIIBBIOK0o07e8z1+G8mlQf1zEKzafKu3m/UhFz9llCXDn7OzfMcZK+/vVmCKOVSWZVJBPzdPpWTexT3AKomPUk9Kt2FtfXUBMMJKKdoZjjNbuSjufSzx1PDSbru0ejfchuisUoI6E4IrS0jV/scAtLpcxO37p8/cz2Ptmse7trmO8RbtPLjByTnr7VNb2s2u6rb6fZ8NK4Bc9EHcn6CuetOmqcpzdktTxsc8Lj8NOTacej8zpdN1SaO8urKOdltpwrSRA8MQeDVzUJYDD8v93nPrXXWfgXSrPw7LbeY8k8xDvdNgSbh0IHYD0rk/+EL1G81P7GdSi8rBYusZ3Y+nTNfJ0OIsDV57y5bd09V3/wCBufnc8DWjayvc6H4GWMt54m1S+kXNnbweTuYcF2YEAfQA/mK9dTTbL+2RNPbxnC5QugI69RmuH8N3I8E2Edmbbz7KM5whw+T1Y/3jXay6tY6rYia2kDBk3Qupx+Felgsww2Oi5UJXt95hVoTou00aNwhnHmLAjxgcBjjPvXF+JobG5t2cQqJIWDPEx64OcHHUGugHia2Nh98AgYIPavN9d1W41PV3i0uGW4coVYRIW6+uK7zE7Xw1rn/E+hs4mJtb6EyRoTny2Azx7YzXb14/omnalp/iTRrq/U2sccyQpEx+chht5HbOa9goAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiqt3eG3hkkRQ/lDLAtg/QUAWqKhs7uK+sobq3O6KZA6n2NTUAFFUtT1nTNGiWXVr+2skc4VriUIGPtmhryW5t4brSDbXsEg4IlwG9GDDII68Y/HildGipyaTto+vT7y7RVE/2ruMam0GVyJyrEA46bM88853dOPep7MXYtV/tBoWny2TApC4yccEk9MfjQncJQsr3RPRRRTMwooooAKKKKACiiigAooooAKKKKACiiigCjq9lJfWBiiWN2znZKSFb6kcik0Wxn0/TEgupVkl3Mx2Z2pk52rnsOlX6KACiisbUodVmvoo7KOFU3hjdO2fLXHK7eCfz70AV/GOvwaNo8sOXN3dxPHCsbYZTtI3n0APesH4X/2Xe6EyGJZNSsy0Nw0nzHazFlxk9CuB/wABIq34l+Hx19ftKapJDqKptSQxL5eOeCowcZPXJPFcV4t8D6j4Z0641nwz4jn05ooVjv7dHwLjAyzKf4TzwO3ODUyfKm7G1CmqtSMHK1+rPV9Q0eC5tY1gggSWB0eIlAANrhtuewOKy7vQLu6u5tQ8iCO8nGweXdOghwAA+VUbzx0IHQD1rkfgdr2va3pWpf2tLPdafDIos7q4yzOx3b13nlgPl+mcew9UrONqsVI7KvtMBWlRTTs91/Xp/VxsSskSK7b2CgFvU+tOoorY83cKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/wAnX+Bf+uNp/wClM1fRFfO/xV/5Ov8AAv8A1xtP/SmavoigAooooAK+d/2p/wDkK+Bv+u11/wChW9fRFfO/7U//ACFfA3/Xa6/9Ct6APoiiiigAooooAKKKKACiiigAooooAq6jpttqlobe8j3KeQRwVPqDXMSWms+H+I1Oo2K9MffQf5+o+ldjRXNWw0K2r0fcmUUzlrPX7C84Ewif+5L8p/wrRByMg5FWb/RNO1LJu7VGc/xr8rfmKxn8IS2xJ0nVJ4B2jk+Zf0/wrzJ4OtHbUydN9DQzRmsprXxPadY7a9Ud1baf6VE2tX1vxfaNcx/7Sgkfy/rXNKM4fFFr5EOLRtZozWInirTycOJoz3DJ0/I1OniLS3/5esH0ZGH9Kz50I1M0ZqiNZ04jIvYfxbFPGq2BGRe2/wCMoH9aOZAW80Zqp/adh/z/AFt/3+X/ABpp1fTwcG9g/CQGjmQF3NGaz31zTU63kZ+mT/KoH8TaYvSdn+kZ/rRzIDXzRmsP/hJ4ZTi0s7qc/wCyn/66et3r91/x6aOYge85x/PFVG8vhVx2bNnNRT3UFqm64mSIf7TYzVFdD8QXf/H3qUVqh/hhGSP5fzq3a+DdOifzLtpbyTuZW4/If1zXRDC159LepSg2Zz6+93IYNEtZLuX+9twq+5/+vir2meGpGulv9dlFzcDlIh9yP/H+X1roIYIreIR28SRIOiooAFPr0KOCjB803d/gaxgkFFFFd5YUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfOnxV+FWvW/ia71nw5YyajY30hmeKAZkhc8sNvUgnJBHrX0XRWtOrKk7xOnDYmphp80D4i1Hwz4m050N/4f1O380fu91q/zfpXUaB8EfGPiLRn1GeGPTIww2Q3m5JJV7sFxxj3xmvrWuY+IOsXGi+EZprRik0zrAsg6pu6n64BrWWJmzqnmVaascpf3vh7wv4LsrHTbXTrjUII0hhDxI7p6ueM+p+prh9Q1jWXdbuTV7vzScqwmIAI9AOB+VZt8qLbCSBgZVOSPX8ayDqNxet5Fna3FxMTtEcUTO2fTAFcp5rbbuz1HQPidLf6LLb6vIDe2x2FwMeap6Nj19a821jVH13Wpp7iTbGXKxhuigV6/8L/B8/hrw7dXWuxLHqOoOshgYAtCgHyg+/JJ/CvIfF1nHaeOdTitXXyDMXAH8JPJH55rpw9uY+w4SUHi53jd20fbXUx/I/fhhgdqfdyyRReVKuNnPTB5p148sUSJIoAUbhjqc1kXuohlJZssfU812NpH6LWrQpRd9CBJt+o7l6Kp3VNpdyg1Ta/3C65HvXU+HdGgg01ZJVTz5V3uzdvas7XoEhUyW6qrn0GM1x+1Tdj8+nnsKlR0LNJtrm9Vbb/gnRPeRfZduBnqGzWdp2oJHqM+COgOD9aw44rv7Ivm3GD3AGcVOlikds0isSc4LbvmJqVQmzjpcLZhNtStFLu9/u1+8va3fxtE/T5uwrLt4JpYmuZx91VUKTz9aqFM3iqzs/BwSehqe2uibgxTEEqMg+ta0qXK7yPayXIvqtZVsS/eTsktr+f6EsjPbHzGB8pjjeBwD6VVkvx5uA2dx65q3fXMYU+XwuBkE5yayWvfOUrbwlyeMItbt2PqsRW9m+VSNGa8RbdUyMKOoHJrp9IuYY9IhBAbMeeuME964M2oii/0onex4TPArbTKIkcEpQYxgdKxqRlNaHzucYLF5nSUaSV4u9r2epc1J4rm+t4WRZA8gBVhkEd63ZLS2hslWIIqjjYBjFczLbi32Th90hGVbdkj/CkOvSSAQ7SZSQox0JPSuedJwV2fKY7IsZgqUZT95PdK7sdTo/ia+hWbT8idYgGjaRjlV6Yra8Pa9B9ul+0/JdHsem32NcZpLfYEuHuWX7RL8rr/AHAO31qXSbe413xPFBYfwKzSP2RSMc/nXy2Z5VgZYepWsou17r+ranm4fE1VOMb3PQNc12H7MxLqOKy7TVfsfhy3SJ5BeOzsqIc8FiQMVq6boFloMi3GoWUesspzi5YgD6L0/MGvTfDUfh/ULFdR0bT7aEklHxCoeNh1U1wcNU8NFylTneXY3x8qjspKyPDftslxG8s1y+9hk4JAPtUvg/xRfaH4sjtrMyzQXwKSQICxJAJDAeox+VdB4p+E+vQalLJ4WEN3ZSsWSGSURvFn+Hngj0NbXw2+Ft5oGr/294llia+RSlvbwtuWLIwWLdzjjjgZr7M8ovaVpur+IfEVrfXtrLZ2NpKJszLtaRlOVAU84zjJr0SiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK5nxdYiWASLNNFvR1cRvgPhSVH1+nPFdKzKilmYKo6kngV5T8QfEMeq6m2lwzobSzIYmNs7pcHuPTOMeuaAO8sp1GgaUY5Vgt3gTfJFgBfkGADyBzSS318rzRoxIMilJDHjYoCbhj3z+GT6V5p4I+JVt4eZtE1uSRrGKMfZJETcYgONhxyR0xx610WpfFe3lbZ4ahS62ruke4VkA9gvX8aAPNPitdufE7Talcq8fkRpE+d4TIJKkjGDkEkYyMjrXb/Axb/TvCVzLqAdNPurrfabxgD93uYqD/CcdfUHjvVT4b+HbLxL4o1TxBrLrc3MDhfszHcN7A5lYEdxlR6AGvWjptqSQ0e6MjiJjlB9F6CuWnQ5ajqPdnu4vNHVwUMHH4Ypa2S1/rrfUfZXJvLKO4MbRFxnY3Veehqeore3itIFht0CRrnCjtk5qWuo8IKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiqE2qrFcPGIJHCt5e9SMGTbuC8n0PXpUthfLfwebHG6J2LYwexwQeeRU8yvY0dKajzNaFqioL65NnYy3ATeY1ztqOwuZLhZkm8tnhk2F4vuv8oOR+ePwp3V7CUJOPP0LdY+u6NpeoxFtQt4XZsIGmGV/EdD+NbFMlhjnhaKeNZI3GGRxkEfSmQnYyfCptk0JLWyVFis5GgHlqFQ4OflA7c/zrZpkEEVtCsNvEkUSDCoigAfQCn0A3cKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8nX+Bf8Arjaf+lM1fRFfO/xV/wCTr/Av/XG0/wDSmavoigAooooAK+d/2p/+Qr4G/wCu11/6Fb19EV87/tT/APIV8Df9drr/ANCt6APoiiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAY8Mcv+tjR/wDeUGq0mkadL/rLC2b3MK5/lVyipcYvdAZh8OaOTk6fD+AqP/hFtF/58E/76b/Gteio9jS/lX3CsjHXwnoi5xYjn1kc/wBaevhjRl6WEf4kn+tatFL2FJfZX3ILIz00DSY/u6dbn/ejDfzqzHYWkX+qtYE/3YwKnoq1TgtkFgAAGAMAdqKKKsYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXmXxnmm/s7TbXeyW0sjlmH98Abc/ma9NrA8b2OnX3g+/GrwGaCGIygKdrKwHBU9jmgD5i1S9urJAshRw/ClG6/h2rvfhV4ytfDPh26N1Z3D3FxcFyUA+ZQAByfTmvJLm582/bzHLKpwu70zXe2l1AlkqqAV2YXnGK6qtJUqcb7s9PFYWOGoQb1lL7keh638TLG80y4msneG5iQkRyrhge3sRXid5duZvtLsXLkl2PJyTnNTa7dxK6kkD5sH6VRitXvZBFZuxY9uoFFB2i2fUcM1IUcLWqtpO617JIikvftDCKH5nbgAVY/stbWJZWGZWGQ5H8q0bTw5NHMiu0akD5mReTU+paW0UB8uQkgd609rDqep/b+XKV6suZvsnZf18yhaa+YoDHN1j4JFUbnU/7Qu0jXO0k/oM1n3TJA42nJOdxPevd/hf8AB7SG8N2eta/E11d3cQlSF8hIlbkDHckY61jKMYu585i8JgcHXjiJNtXvZW16/cePG9DQEBsHuKgbUk2csBXs/j74W6GbWSfTLf7HOgJHksQrexFeQ22k2qI6vGu8Zz5hyciuiE+fY+ty7NJZnf2OjW9/wK9hBLePJdDCxovBY43Z9B3pVt4xdksSzbTgntTprsW/yMQAOnpSWC/aJmnn3pCFIUgfeP8AhV6bHpJQTVP4pXuysLN7qd1aVhGv6+2auJCbdRHA67c4wO1WbaKNmPRQTjmob9FjLbWBKnhlp2tqbKhGlFz6kklgjQ73TcM43Ed6pILj7aIoFMp2sQo64AycevAqRdTD2wy2PVc8Zqz4TcTeLre6ckQ2eZXIXOTjAX8c1zYqq6VGVSCu0nZd32OHMMXSo0vawaTK8TXF+wS3QkscZPAqC7shBGQWO4HLH3rW1yWGz1ec20bRWsjl4gRjAPUfgaxLi6a7kWC3HmSOcACqjVjOkqktLq+vT1IeKw1bDe1cr3R6zb2nh/XltZNRslcXEIeGWNzGzDH3W2nkj+lb2jaXY6VDI2mW8dvCD91eS59SeprzuG7FpDpllAxcWoVWdecnvj8zXp8WoW62IC4xt456V+LZh7WCVNSbg27Jt2300PhoKm5txXzKuq6pGYTkgYGK1/hA000OtXGD9medFQ9iwU7sfgVrznxBdRXGoCPzmjTBZyh7CvZfhrayWnw/01JY/LLq0irjB2sxKk++CK+h4cwqjW5/L/gHBjql42Oqooor7s8cKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACo7gyC2kMI3OFJA9fapKKAOTaa/1bUpI9L0/wCzBFw9xexsERs9l/ib9OOtZt78LbC4uX1K61S6a7aYTSttURMoOSmwDgH1zmu+pHVXQo4yrDBHqKAOCn8C+EpNfaDUtHtZW1Bt8WzcpjZV5UEEHBAJ4wK4/wAbfD6bwm9zqnhu236XIF8yLzSWtzkDA3HlSefUZ9K9ZsvDtlZakb5WnmmClYzNIWEYPUKP6nJqp48ntbfwFrDX05hia1dAy9d5GFA9yxApNpK7Lp05VJqEd27HL/CHSk0vwjda/dXIeTUT5sipyIUj3AL7t94n647V1914jisbKe4vbWaJo1V1iBVmkVjhSMHHXt2ry34P+N7RLEeEPEAkW4uZ5DbyMB5cgcZMZI5BzuxnrnGeleheJrDRrPRJptcmmuAFAj3ShZMDoqYxk8n1PNYxqe0jzUz0auEWCrOji000+nVeW25rRa1B9uubS4ZVkgVXLLyCGJA988Gr6zxNKI1kUuU3hQedvr9K5bwlJa6tpP8Aa2m7BdyXEjyrK2WHzMqq2ORhcY/+vW9b6fJDqb3bzq4ePZsEYGOmMH04PHvW55ZfornvEWtRWD/ZpIJZ5Jl2wJCpZi/JwAO+BmoRqF7BplmLu7MkhQtcNCybgwGdgJGD3468deaAOnorC0bV7u+sEbydzPI/ktJ8nmRKR831OePXrWhZagbuaRCqoFJAGTu4OD2x19CaALtFFFABRRRQAUUUUAFFFFABXn/ij4nW/hPWPs95azz7EzPCu0Fcn5WXPUYB4/lXoFcL8Q/hjp/jWGS9R5bfVY4CkTxsAkmMlVcemTjIwRms6nPy+5udmCWGddLE35fL+tjprJdO1yxt9VtC7Q3kSzxsHZQdyjDYB4bGBnrVid4tH0eeZI3dLaJ5ioJZnwCx5PUk5/E1y/wx18ah4QtNPvkitNR08fZZLYYU4QYBC/Qc+4NXPEGoRaFpLSXcReeRisW1smV8Eg4PQep6jt2px1VzOq3Gbg3dJ97rTzON8DfF6+8S+ME0jUrC3EF1E8kb2wYmLaM4fOcjHGeOcV6lby2wLW9qFURMVKouFUjBI9P4hXm3wb0LToNJv71PKGryXkguJI0AdIyQVTnopxu4x19q9Jt7aCxEmx2Alfcd7556cZqKMZxh77uzpzGrh6uIbwseWHbX79SxRTHljjxvkVc9MnGafWx51gooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8AJ1/gX/rjaf8ApTNX0RXzv8Vf+Tr/AAL/ANcbT/0pmr6IoAKKKKACvnf9qf8A5Cvgb/rtdf8AoVvX0RXzv+1P/wAhXwN/12uv/QregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAqpq1va3ej3kGoMEtZIXWVicBVxyc+3Wrdc58Qra6vPh3rkGnhjcPZvtVerADJA+oyKcVdpFwipSUX1PkXU7aFtUuvsFwZbdZW8qUrt3rng47Z64qJNTv7CLy96yIOB2xS2lyyEGLBYccjIqOdRtPmL+Ne9OjCUEmfcVcJRq0VF62IJmudQl/0nKIOcDvXV+EzHaGXIwSABnniuYgfeCpbJFTfb5IF2q7Jg5xnvWUsPB0nGJz1MFT+qypU9LnoM2pIl5u4UMoxj1FUNW1ZXRmdwSaxfB+n6j458VW+jW0vlq2Wlmx/q0HU/574r6Dh+CPg+KGBbm0numTG55bh/nPuAQK8mcHDc+UqUHSaU3+p4x8OfBkPizX1vtXRv7Jt5NrDoJn67c+g4z9a+iYNajsI/skhVGiG0DoMdiPapLnw9pmn6UlrYWkVtBEMRxxrtC/SoLCbTb/AE3yb+2hlkgJRi65Psc1Ddx4jEOs12Ssvkcl4y8TwJZyZkXketeG3Ukr6jIITvjkYsHxjGa3/EOoRa1r13NGkdvbq7LDEM7VUcD8TXO+asF0BnAbjFdtKnyK7P0fJcmeAgq1WfvStdLou3mRz6eS6u7liDnaRwat+U8h2N0xRPeB2iB2gLxkDr9aRroJIGzjjg1vofTxjSi20QzI9qpMbcf3TWbNePcny4EZnPb0+tXNQv8Afvd3yzck+tSaahj08eYuHc7sHjgnioerscFSUatZUITsmv6sV7Dw9Lc3Uccsw2Zy20dRXaf2amm2ii0HkqRxt6H/ABrGieW2Kz4JXpwOKln1tJUCR7nc8BFBJJ+lcleNpHwfE9GdDExgk1C2nm+vzINRvxPbOkqq0qMBjGdx7cV6d/YKXVp5WmaJBC7oDLsgCKpxzk4pfh/8MJkt4PEevWZ+3+aJbW0nztRQPlZ1/vZ5A7cV2mu+NtJ0uR7e/s5pr0DJhiwQAfUnp/OvEx+XRxsoc0mlG+i67HzlHESoppdTy0+GpbO7NvbThrzGCQm7Oc8L6dDWelvrsut2uiWEUjXFy4QK2TtHdyewA5NdpYeJNEv/ABhp8y28tpL5wUxvj585AwRxwTXpVr4ft1u21FQ0d06FA6HBCnt+grGeT0JVVNbdV/wSlipqNupxOj/BWCLUFudf1Rr5FYEwRJsWTHZjknHsK9RRFjRURQqqMAAYAFcxLrk2i6lFHeSma0lcIWf70ZJwDnuK6ivTo4enQVqasc8pynrJhRRRW5AUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRTJJPLA6ZY4XJwM0AMu/MFsxhUMwwdpOARnnmuD8VaBe+KPCviC0hkSFkuUNqh+7vjwTn0Bzj2xmrvinx7b6RZXNsqL9tUrHhmBQBjjd1yfpjrXI+EfGU2ga01jqUjXNldyGSadtzvE23GfdeBnjvmplFSTi+prRqzo1I1YOzTuvkcVpXwg8SazrPk+IHj0ayjBY3XmpMXK9AgVue3UjitHV/D174FvoINQ1p9VsPLYWzvnC9PlI3ELgenXHtXtlp4m8OXgjjt9Rs+G+SNyEII9A2K8x+IGuaZ4g8UaNZwSeXpyXKm4uCm1Wy4LHOM9FxkjHNZ0qMKXwnbjczxOOd6z+RmeC7/wARJNfX3hexuZYX2iWYIpjcDJ4DEZI55HriurPxM1y1Xy7/AEdYpWXKtJBLEAD90kHIxkj+Kun8Q+IdE8JWqx6nqa2yTA+RbJB5ny55+QDkflTmsdI8Z+GI303UvNspYTAZ48MSoIJGD91gVBz2x0q+dXsnqcv1eooKpOLUXs7O1/U81Pjm+03xVZatrN0t1axOTIixgFAw2krjrgdjXTa/8StO1PTY7bwrPvupAZGd4seUMc8Efe57Zrk/CWg6Pr2q6reTO9/b6GqskBGYrhiXwX4+YYUHA45qbxl4Tk0vxNF/wjVhau9yscbm3URAySMwAC5wvA5I46Uc8eg/qtVNqSs10ej6f5o9A8FeIrDxFoyWU4MV3pyJFIGk2l/lxuUg/dODkf8A1q6uK0t7eUtEu1mBAG44APJAHb8K898E/DzV9IWS+1bUIYruUELbJCsyRqcHBLDOcjnHFdBqVvqqWy6fZ2KPcSMCt5CBHHEf7xBJIx7ZzVnMdTRSKCEAY5IHJpaACiiigAoprlljYoNzAEgeprK0bVn1K9voirrHblAvmJtbJBzx6cfzoA16zl1iJ9WjslwzPuyVPKFezemeajn1SaOPeoiJJb5D95QuevPfH/1jXN/EDWY9FNnNZyLDfu6vuaPKlB3J9e30JqoxcnZK4m7HZXFyYXRFTez5OC2BgdefxqL7fmGeUIBHDuy7OMcKCD9DmuBg+J66npnlxadcPq8bHbHax+YoPYnuAc4ri7XXL3U9cs9IuLmW4gv9QUXNhGxRWUt8wI6jGCT9KTTTsxjr621y91L+0LGzmmSa5Z4NQj+UPl8blOc967L/AIV9Hrmltd2evXz3od1WW6IdMg4IwPp1Br0KWK0gsDFLHElqibdhUbAo7Y/pVePUtNtoEjiYQxKCAoiZQmOcEY4PsetS2luXGE5fCrnDeE7aL4d67LpWuzDdqwEsWon5InZFOYzk/KRkkZ65+ldrfRJqUMNzavb3MOx1G58od2MOCM5Ix+p5rnvin4Y1PxT4VittFEclxBcCbyJWCiUBSMZPQ85rN+GXgrxB4b8NTQ6repaTzXBmS2jAkWIbccnOMnqQPbmsbzdRwa07noqnh4YSOIhU/eJ25f18tP8AhzqIdD81VaRre4XzY5AzDdgK3zAZ9ep9ya36p6Xp/wDZtl5LTNO7O0jyMMbmY5OB2HtVytVFR2OCpWnU+IKKKKoyCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/ACdf4F/642n/AKUzV9EV87/FX/k6/wAC/wDXG0/9KZq+iKACiiigAr53/an/AOQr4G/67XX/AKFb19EV87/tT/8AIV8Df9drr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK4f4j/EhPBC2Gn6bp0mseINWk8uw0+M438gFmbsoz+PsASO4rxDSX/tz9sbWftsaONF0cLaZHKZEWT9czyfgaANYQ/HqVBc/afB0JK7vsmJiR7Z2nn/AIF+NaPw1+J954o1jUPC/i7Sl0XxRpihp7ZXyky4GWTk46jjJGGBBOTj0ivGPHcK6b+014C1K1JSa+gmtZwrY3qoYDI/7afoPSgDW+MXxA8ReD9U8K6T4UisTd69dPb+ZeqzIhDRKvQ8AmXk89OlZ2r+JPjL4O0ibW9d03wvq2nWamS6isJJkmEY6sC3Ax1PB4zxWP8AtGXq6b41+G188M1wtrqEsxht03SSBZLY7VHdjjAHrU3iz41QeK9OuvBGgeGdZtdc1y3eziTWI47RFWRSpckuT93djjkjFAHrnhPxNZeMfClhr2l7hbXse8K/3kYEhlOO4YEfhWxXMfDnwifA3w/0zw+8/wBoltUYyyDoXdy7Y9gWIHsK6egAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD5s+N3g608MeIbfVdIsfs1jqAPnGPOxZwcnj+HI5x04NeU3t55pLNgsfQYr7f1LS7HWdPlsdVtIru1lGHilXcpr5z+LHw80zwX4k0zU9FtEg025O0xGRm2SqckjcScEEflXpUMS3H2fU+iwWYSlTVC2utvM84bRLrSLGC7uxgXHUY+53APvVW9uUlXLKnAA4GK9CuL62utOaCcJLC45VhnNZvgPQdA1T4o6Vpt/ZNd2s7vuhaQ7cqhYE+oyORSp4tqPK0Th81apezmrvU9K+EGhx+GPAttrU9h5V1qEhllnblvKyQg/2R3/GvSbnxLCLfcJF9etVNZ0DV4lePRBDNat92B32GP2HYiuY1Hw/qUICzaDcSORn/AESUshP4Hj9K4JScndnhVJupJyfU19W8X2/2UnzV6etcFH4jKzXc6XPlM5ARCM7gM80viHwv4g0zT01NtHItzkNFB+8eEdmZRn9M471ymk6Hrfi/V4rTRbOZfmxJdSRlY4V7lif5DmpIMLWkmtLqWaMB4ZGL/L1TJzjFZf2HUrm3k1A2Uxs7dgJJAPuk9M966rV/BviDStUNpqemX1xIrYjEEDPHLzwQwzkV634K8CaZF4MdPEViRfXx/wBLSRiGjHOE4PGB+tbKrLZn0dLiDFKMadTVK3rY+cRHdTyfLuSMc7nGKjMkz3C27yDlgNwBOPeu38WfDjXvD+qvHHuutLL/ALu+RchVJ43jsf0qa00CysNOwgWRyfnkfG5j/hWkqiS0PXxmb4enBeyk5OXmVLDw7pVtEJbjzLqdRlWdvlz/ALtU9TLTX0a2y75OflHcAZqO8uLm3uTBb7pFIJACliAOvStDTY4bMSXEsomuHGNxGNg9AK5+Zp8x8ksTiIVYYlyu09PkYcupKIypbGOx7Gu9+DVndQard+JpYB9jij8iNmXl2JBZl9hjk+9ZXhjwtb+NvFjQTKy2lrEZrmaNRkDoq59z/WvoGXwrostlEsKvZrFGEUWz7BgDgY6VrOrzKx7mZZ/LHYf2drN7mvalby1EszZ3DOM8AV4Dq9y1v4i1JLtQ0q3MgYMfc4/TFeoS3mo6XutbO3nvoYxhXiXcwHuBXB+KPC+r69ef2jJplzZErgynapfHTcpP61gfLHET3Dy6xYx2g3XDXMYjA6ltwxX0DpOvSNeTaTu3yR8xsT/CfX6VwXhL4bXek2y60yR3V7IpMLzPgRKe4GOp9au+HdL1m68WzRrIsE6RGSWRgSuOiqPxoA7jW7SzWyczYlkI5Zu30q14P1OXVfD6S3DF5IpHhLn+LaeD+Vc1eeHvFepy/Z5Ta20J4afzd3HqFxn+Vdno2lQaJpMFha5McQ+83VieST7k0AXqKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKgvLSK9tmgm3BW6Mpwyn1B7Gp6KAOL8VeEdJ/4RK4gjAjud4ljvJfnkDg5yWPOMcY9K5zwX8LIrqG31jxPdG682ItFZxgoihhwxbOSccjGMe9ei6zZXl5blLGSBS6lJFuELKVP0qxpdiNN0q2shIZPIjCbyMZ/Dt9KAOS1H4WaNPby/2dLd2khX92vnl4wfdWzwe/6V5XfaXr15epocOmu2oMWQwZH7sDGWY9NvzA59xX0XXm0rpafHGDy5ABMrCQsR/FDnHtyi/5NAHlnjT4aaj4S1WD+zbqbWhcR7HCooaJyeFK5zgjofr7V2nh1tV+FHwnuL3VIUa/1O9BgtWfckO5cDcVPohJAPoM13d5pGqy65bzQ2UB+zXDzi6abHmK2fl2+uDjnt0NVfiVpVpc/CvUlvonD20ZnhKfMyzZ+U8DuWwfYmuSVBQ5pw3sfQUM0nX9jhcQr01JX7s8z+D/AItOh+K/+EfmiSW31eXiVfvJIE49ipAxXql7p8raxZQWGkeRLBciU3KKFhC9z0wW28dM5rzn4Q/D+D7PY+MNXe5mmfzRaQxpgRMG2q/GckjdjOAO9e12qyrc3XmGQqXUpv7DaMge2aeFhKNNKRnnmIo18bOdDRdfNotUUUV1HhhRRRQAUUUUAFeNa149vND8Walf2jxhI5PJks5V+8qnrnqCTk/jXovizxppXhCy8zUZs3EiM0FuiktIR9OgzgZNcb4I8IxeK9LTxZ4lRH1C/mNzAEHyxoMhcg8En73PTCjsa6o4dqn7WppF6LzZDnryrc15vi54LS0E32xppXwxto4CZfqQeOMdc1RtLeH4r30mpTxzweHbf91bYfZJdSKTuYjqqjJHvVb4weDLCfwfLqmk2qQ6lpwXb9nRQZFYqpD9yMcj6V0vgrR7bw14Q0qx0yWRreWL7Q0xG8yyPhjnHbk9MdBW8nh4YdTpX52++yX9fmQudztLYXw74DtvC98p0qZkttxkfeS0srbSAGbpgZJ6elVfiL4YF5oFxq2iQxW+s2J+1RzRoA8m3kgnucDIz3Arq9OnuJ45muo2jIlIVWXGFwMfX60+/s49R025sp8+VcxPC+04O1gQcfnXnTblfudVKUYTi2tE0cl4R8daT450qKwefydVNuslxbbSpDDGWUngjOD+IrpI9LkW8SeS7MmGMkiGMYd8BQ3tgAD86818NfB7UfDPi+31aLWEukhJWM7PLKIeDkZOTt49Oa9brGkpuKdRanfjp4enWlHBSbpvuu+/QKKKK2PMCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8AJ1/gX/rjaf8ApTNX0RXzv8Vf+Tr/AAL/ANcbT/0pmr6IoAKKKKACvnf9qf8A5Cvgb/rtdf8AoVvX0RXzv+1P/wAhXwN/12uv/QregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArwvxdPH8Nf2kLHxhqKGPRPENp9iurvkrDKAoyeOOI4z9Nx7GvdKp6rpOn65p0lhrFlBfWkuN8FxGHRsHI4PcHmgBkeu6RLpy38eqWTWbIHFwLhfLK4zndnGMc14/ot4nxS/aHj8RaQ8kvh7wrbNBFchcJPcMGB2nHI+b8kU9G56cfs//AA0F0Jx4c5D79hvJ9mc5xt34x7dK7zStJ0/Q9OjsNHsoLG0izsgt4wiLk5PA7k80AeNfHr/kpXwr/wCwu3/o62rt/jB4OXxj8Or+K3i3apZIbuwkTiRZU+bap7bgCv4g9q39e8H6D4m1DS73XLAXVzpE/wBospPNdDC+VOflIzyi8HI4rboA474U+Lz43+G2l6vK2678vyLv185PlY/jw30YV2NY3hzwjofhKO8j8PWIso724NzOiyuytIepAYkL0HAwK2aACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvPPjN4PvfFfhCN9IjM19p8vnJAOsqkYZR79CPpjvXodFVCThJSRpSqypTU47o+G5Zr2O5Nl5Fwtznb9n8tt+fTbjNe+/BL4W3egzf8ACU+JEaLUZoiltauPmgQ9Wb/aI7dgT3PHsZghM3nGJPM/v7Ru/OpK1qVufZWOrEYz217RSvvbqFFFFYHCFFFFABWNfWaQatFdNIRDMcPERwWA4P6Vs1T1aw/tPTZbYOY3PKOP4WHQ0Acb8VL5V8HmG3YDzpkjdgPur1/mBXg13rD22Yp0YOPbg+4r1XX9K8QXCvp+oWtzLE/H7mAvv+jDivPNV8K3llqrWGpXDW20KxR0DOqsMjkHGcUAP+HCXZ8Ur4gaKT7NZh1DKM5crjGPocn8K67U9J0XxXqxvLi1a3XOZ54nMfme2O5964hbiPw/PbyWJcJE43BWIMi55B9zXvkHw30b7T59zLeXKHnyZJcL+O0AmgDlfDN7o2i2V7p+mWotrRnCtIuTvYjoXPVsA1sSahqUaiKzjN8rfcMJG4/UVc8e+F4ZfArw6PFDa/YH+0pGMIrYBDDPqQTye9eceGvF9zocLrdQko/IcHLKPT6UAdPcaprXhvzNR1CGOISkRx25ky7N6nHAFcxqvjTxDcAvNPF5bf8ALIRYXHpnrV7WhrXi7Q/7Y02zaextWZyVcFmxw2F6kivPL3XYmi2eZuYcBc8/TFAH0H4d8YWHiHwpFd20f2cp+5lgJz5TKOme4xgg+hqfRdMS5a41NmZTL+7j2sR8o6n8/wCVeafD3TpdG0dpteEkC3cpmW3YYxwAA3oeM4ruz4ys4oWjWVECjAAOABQBfi1qXTdcgsrqUy29y/loznLIx6c9weldRXl+lG48WeK7WS2VjZWcqzTT/wAOVOQoPckgV6hQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFNkLLE7INzBSQPU06igDj77XtQuNRWDw3FNdOo/wBJQgARdurYAOe3t04rxbU9TjFpKl/lbnzHEoZsvu3Hj3b+tfS7sI42c9FBNcXFp+nJ8Rob6SG0upbyJvLcRgvGw+bdnHoMZ96AMW017x/YWdu8ujXFxD5aqgeBXduOCwR9wPrmsTxN4n1jWkksNbWTTFVQWtljMe/uCwJJP8uK9rrhvifounS6E2tXFuPtNpsXztxG1C2DkdxzQ3bVjjFyajFXbOU+HnizU7XTptGgtnn0+xdcXQjJ8hWJLBmGAOhOevP0r0rw9qUupm8kUO1ksoFtK45f+9z3APesDwKtlpvw30ubRI0ja/ZPOmIJzKx2O5z1wRgDgcCtiTV7xPOjMiA2gZpJVhL+cFIHCg8decZ6VDmkdEMNUn5HQ0VS0u8a+t5JH25EzoNoxwDgVdqk7q5hOLhJxfQKKKKZIUUUUAcD8SPhvJ41kgvLDURZXtvE0WJE3JIpOQDyMHOefeqfwr8R2emeFofDOuGPTNU0p2gkimkwHJYncCfUk5/TgivSq8T8daFq0vjibTbOKKeXViZoZXYKFUKc7z6KFPrwBXRLE1Z0lRk7xW3kQoRUuZbmp4/8WacynS7W7hnhZ2kuZI+Sfmyqk+gP8hXQ/CnXJdZ8Iskzo32GY20QHDeUFXZuH5gHuBWl4e0rStA0y1063sYxJJGollKKDOQB85JOTk847elcL8ZI7nw3c2PiHQmudOMyvbXtzZuUDjAMYcD6Ng9eMUYag8RVjSTSv3CcuSLkej6q+pJeFrIStB5aKyooJBL8sM9wByPQ1Z0kzm2l+0GRgJT5ZkBDFMDHXnrnr/LFc18NdZ1m/wDAltceJYbk3m5wrSREPJGGIUsPXH07V2UbrJGrocqwBB9QadV8l6Vlp1QR194dRRRXMWFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzv8AFX/k6/wL/wBcbT/0pmr6Ir53+Kv/ACdf4F/642n/AKUzV9EUAFFFFABXzv8AtT/8hXwN/wBdrr/0K3r6Ir53/an/AOQr4G/67XX/AKFb0AfRFFFFABRRRQAUUUUAFFFYninxPa+F9LFzOpmnlOyCBTzI39AO5pNpK7LhCVSSjFXbNuivNAnjbXh9ovNXXR4n5S3t0+ZB79/zP5Uv/CN+I/8AodL7/v2f/i6z9o+iOz6pFaSqK/zf6HpVFea/8I34j/6HS+/79n/4uj/hG/Ef/Q6X3/fs/wDxdHtH/KH1Wn/z8X3P/I9KorzX/hG/Ef8A0Ol9/wB+z/8AF0f8I34j/wCh0vv+/Z/+Lo9o/wCUPqtP/n4vuf8AkelUV5r/AMI34j/6HS+/79n/AOLo/wCEb8R/9Dpff9+z/wDF0e0f8ofVaf8Az8X3P/I9KorzX/hG/Ef/AEOl9/37P/xdH/CN+I/+h0vv+/Z/+Lo9o/5Q+q0/+fi+5/5HpVFea/8ACN+I/wDodL7/AL9n/wCLo/4RvxH/ANDpff8Afs//ABdHtH/KH1Wn/wA/F9z/AMj0qivNf+Eb8R/9Dpff9+z/APF0f8I34j/6HS+/79n/AOLo9o/5Q+q0/wDn4vuf+R6VRXmv/CN+I/8AodL7/v2f/i6P+Eb8R/8AQ6X3/fs//F0e0f8AKH1Wn/z8X3P/ACPSqK81/wCEb8R/9Dpff9+z/wDF0f8ACN+I/wDodL7/AL9n/wCLo9o/5Q+q0/8An4vuf+R6VRXmv/CN+I/+h0vv+/Z/+Lo/4RvxH/0Ol9/37P8A8XR7R/yh9Vp/8/F9z/yPSqK81/4RvxH/ANDpff8Afs//ABdH/CN+I/8AodL7/v2f/i6PaP8AlD6rT/5+L7n/AJHpVFea/wDCN+I/+h0vv+/Z/wDi6P8AhG/Ef/Q6X3/fs/8AxdHtH/KH1Wn/AM/F9z/yPSqK81Ph3xMnzR+MrxmHQMhx/wChmpLTxbrvhi8it/F4S8sJWCJfwrgof9oADP5Z69aPaW+JWE8Hdfu5qT7ap/iejUUiOsiK8bBlYZVgcgj1pa1OEKKKKACvnXxj+0/e6L4o1LSdH8OQMlhcSW3nXczbnZG2k7ABgZB4z09K+iq5+98BeEdS1OXUNR8MaRd3c3+smuLKORnPqcg5Pv1oA+Z7r9qfxvMSLfT9Et1zwRBIzfmZMfpWNc/tG/Emf/Vaxb23/XKxiP8A6Epr1H4kfGLwb4KM2leA9F0i91dSVe5htYxBbNjB5UfO3sOB3PGK8F1TTvFHiTR7vxt4glle03rCt5dHaJ3JwIoRjBwAxwuFUKenAoA2Ifiz8TfEGpW+n23ie/e5u5VgiSDbGWZmwB8oHc19o6Bp0+keHrGwvL2a/uYIVSa6ncu8z4+ZiTzyc8dhx2r5f/Zh8Ff2x4xuPE95Hm10hdkGejXDjH47VyfqymvrGgAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACuH8f8Aw9Pit477TLpLTUY12EyAlJV7Bscgj1ruKKAPHfC/wWvY9cgv/Fl7bTQ2ziRLS23MJGByNzEDjPYDmvYqKKAOT+Ji3B8B3jWwYhGR5Qv9wMM/4/hXhV1qUZtgPl47jvX1A6LIjJIoZGGGVhkEehrhbv4OeDru/Ny1lPGpO428VwyxH8Ow9gRQBifAcXT6HrFw4YWct2ogz0LBcOR/46Pwr0G807Q7ATavc6bZpJAhle4+zrvAAznOM5q7Y2FppljFZafbx21tCu2OKNcKo+lVvEGmtrHh2/0+Ngj3EDIjHoGxxn8aAPF/EHinUdavZr2ALZwPwIkUHI7Fs9TWt8KTo2r6pe22qWiTapEm9DIoMbR5wcL2YHGfrx3rzvU76bSJ5bHVI3tLiI4eKTjn29R713fwT0C/uNdufEtzC8FkIDBblxgzFiCWA/ugDr6mgD2mGGK3iEcEaRRr0VFAA/AU+iigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACqltpVjZ3Utza2kUU03+skVcFuc/zq3RQAV5t8aPEQ07w1HoiRgy6sSDIx4jRCpJ+udo/OvSa5Px74CtPHWmQwy3L2d3bMWguUUNtyOVI7g4HcdKxrRlKm4x3Z6OWVqNDGU6tdXjF3/y+52OD+DvjcTxp4N1K1SSKONvssiLneMszK46HqTmvVrs6dbWaNPZxm2tjkZiGIfcA9Me1cX4J+DmleFZprvUbk6tfSjakrR+WIV/2ACSCe5zXU3vhaG6jW3ju7iCzP+tt1bcJPX5jkjPTilQhOMEp7mmaYihVxUqmFTUXr8+r8jVhtLRJmure3hWWUZaVEAZweeT1NWKREWNFRBhVGAPQUtbnltt7hRRRQIKasqO5VXUsvUA8inVw/jG81HwZYTeJLedb4RBbf7NKm0BWcAEsPTjmrpwlUmoR3egm0ldnV6lfnT0gk8vejy7ZOeVUKzFh64C5xXltt4ytdR+Jmn6tfx+VZmI2kPO4BmO0P9CSR9DVPR/F3jP4n30mk6ZLYaTb2gD3t5ErMSrZAVQTnJAPQj69q33+DNuswmtdevEkh2yQb4o2CyKchm4+YZHTiuuvRWG/dVEufyexnGXP7y2O/ksLeNPNwxMQyMueg5A+gIFeW/EHxNaa9Fp+l3cbQQW15bz30gYkR4yWXGPmAHNY8fxk8W3NxBo1vYabJqVxILJG+YZm3bd5BOAD1xjiuof4QT6rGZte8QTG5uCWultolCknspPP4/yq6tCpgWvaL3ntZ7CjJVdtj0MW0N1AXhuJBFOA42MMEHnI471bUbVA9Biuf0rUZrCa00W6aGaZcxDyidwRV4Zhjjpj8RXQ15xsFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wL/1xtP/AEpmr6Ir53+Kv/J1/gX/AK42n/pTNX0RQAUUUUAFfO/7U/8AyFfA3/Xa6/8AQrevoivnf9qf/kK+Bv8Artdf+hW9AH0RRRRQAUUUUAFFFFABXm+oj+1vi/JHcfNFpVqrRqem4hTn6/P+gr0ivOIv+Swa9/17Rf8AoEdZVenqd2D0c3/df5pHTUUUUCCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKpaxYR6no11aTKCssZAz2PY/gcGrtNk/wBU/wDumh6ocW4tNFb4Z30t94GtfPYs1uzQgn+6Dx+QIH4V1lcR8Jv+RIH/AF8yf0rt6dP4EZ4xJYiaXdhRRXmHxP8AjhoPw+SSxtSmq65jAs43+WE+srDp/u9T7A5rQ5TuvEfibR/CWjy6p4gv4rK1j/ic8uf7qr1ZvYc18r/Ef46+IPiFdnQfCEFzYaZO/lLDCCbm8ycANt5AP9xfxJrDstN8f/HzxY1xLI9wkbbXuJMpa2Sn+FQOB9Blj1OeTX078OPhF4e+HFoHso/tuquuJtRnUbz6hB/AvsOT3JoA8t+Fv7NiR+TrHxEUO/DxaSjcD081h1/3Rx6k8iuW/aP8VLq/jKz8H6KgFjoqiPyIFwpnYD5VUf3V2qB2JYV9ayb/ACm8rbvwdu7pntmvl/4W/BrxTL8XX1rx3pzxwafcPdSTTMGF3cbsqV5+Ybjvz04x3oA90+F/g5PAvw803RioF0E867YY+aZ+W+uOFB9FFddRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBXudPs7xla7tIJ2T7pljDEfTIqcAKoCgADgAdqWigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACqOs2NvqejXVheRrLDdRNEyNjByMd/zq9UdxbxXUDQ3CB426qaabTugOc8A6Bpnh3w0LTSbdI8SyCZxy8jByBub+IgYFdPUNraQWNstvaRLFEnRVHFTU5Scm5Sd2JK2iMaLwh4eg8QNrkOj2iamxJNyIxuyep9ifXrWzRRQ5OW7C1hNo3bsDPTOKWiipGFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/AJOv8C/9cbT/ANKZq+iK+d/ir/ydf4F/642n/pTNX0RQAUUUUAFfO/7U/wDyFfA3/Xa6/wDQrevoivnf9qf/AJCvgb/rtdf+hW9AH0RRRRQAUUUUAFFFFABXnEX/ACWDXv8Ar2i/9Ajr0evOIv8AksGvf9e0X/oEdZVOnqd2D/5ef4f1R01FFFAgrxP4ofFfxX4Y+IB0Hw1a2cyLZifEsLO7fKzMeGHRV6e1e2V4L4ljSX9rTRo5FDI9uqsp7gwyU0RO9tD034Z+Mz478D22rzpFFeB2huooSdqSKe2eRlSrYPTd361znxY+JGseFNU0rQ/CVtbXeq3qvNIsw3bI16cZHXDHJPAQ/hzHwguP+EK+K/ibwHdSBIJJGms1Zv7o3DGeSTEwJ/3Kr+F2Pj74peMvGEn7yx0u0ltbBuq8oyKynp9xXYj/AKaU7E8zat1O7+DXjvVPHvhi9vtbjt0nt7wwqbdCqldityCTzkmvQ2ZUGXYKPUnFeJfs3XCWnw91y5lzshvWkbHoIlJrn/Bvhm6+O2o6tr3jDVb2GwgnEVtaWkoAjJGcDcpAABXnGSSaLDUnZH0aCGGVIIPcUMwVcsQAO5NfPGi/bfhB8cbDwrbapcXWg6r5arDOd2PNJVTgcBhIOoAyDzUnjGLUfin8dZPBst/La6LpqZkSJuu1AWfHQsWYKM9B+OSwc5q/tF+Jda0BfDX9g6td6eJ2uXkNrM0fmbfK25weQNx4PHNe2QktBGSckqCfyr5Q+M3gSXwH/YtlBrN3qGlSidrSC7ILWzDy94BGBg5ToB0/E+l/HXxbf6ZZ6D4c0zUG0waqwa6vEcoY4gVXGRyFyxJwRwuOhNFhKVm2z2UOjMVVlJHUA9K8u/aC1vVNC+HtpPouoXNhNLqUcTy20pjcp5cjY3DkDKj8q808W6B4E8J+GotY+HvjA/2/YsmPJvlZrgFgGO0dMdcDjGcg10Pxi1qbxF+z74X1e7Ci4u7y3kl2jA3+RNuwOwzmiwOV00W/iN4k1vTv2efCmoWGrXtvfXZtFnuop2WWQG3djlwd3LKCTnmuouF8a6z8GPDUng3UVj1iW2tpLi5uXy8iGL5juYHLFtpJPvXCfFP/AJNo8Ff79l/6SyU74oSOn7M3g1UdlDmyVgDjcPs0hwfXkA/gKBX3PcvD0WpweG9Pi1+ZJtTS3QXUiYw0mPmPAA6+1aDOqY3MFz0ya+fvinNKf2afBxMr5layEh3H5/8ARpDz68gH8KqXnwuv/E3wql8aeKvEl3d6mmmNfWsK48mKFYtypjHUqoztwMnv1JYrm6JH0bSF1DhSw3HoM814t8M/GusTfs/65qMkzXF9oq3EdvNKd5IWIOpbPXbuP4AVxul+B/D3iT4M6n4017V7mfxA0VxO0sl0CVkQsI4yD13BV9/m4xxRYObsfTlFeO/s5+G7aw8FS6/G9x9o1NzHIjsPLxE7BSoxnuQck9K9ipFJ3VwooopFBTZP9U/+6adTZP8AVP8A7poBGX8Jv+RIH/XzJ/Suu1HUrLSNPmv9UuorS0gXdLNM4VUHuTXI/Cb/AJEgf9fMn9K8/wD2gvh34y8Y6po8nhuee+0928mTT94SO3k5ImPQYIyCTkjGB97FOl8CJxv+8z9Tk/ij+0hd6qZtH+HzSWloco+pkFZpf+uY6oPf73+7Vb4Yfs66l4jkj1rx759hYOfMWzJIuLnvl88oD/30fbg16p8LvgNovgQRalq/l6trowwmZf3Vuf8Apmp7/wC0efQLXrFaHIU9J0jT9B0uHTtHs4rOzgXbHDCuFH+J9SeTVyiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8AJ1/gX/rjaf8ApTNX0RXzv8Vf+Tr/AAL/ANcbT/0pmr6IoAKKKKACvnf9qf8A5Cvgb/rtdf8AoVvX0RXzv+1P/wAhXwN/12uv/QregD6IooooAKKKKACiiigArziL/ksGvf8AXtF/6BHXo9ecRf8AJYNe/wCvaL/0COsqnT1O7B/8vP8AD+qOmooooEFeEeIgR+1xoeQeYEI9/wB1JXu9FNEtXPnn9ofTLzQfFGkeL9Hkkt5Z4ntJZo8fI4Bx+LIzD6LXcfDDww3hn4IOs8Xl3moW015OOc5dDsBB6EIEyPXP1r06ii4uXW54j+zhbC7+HWu275CT3rRkj3iUH+dcx4F8Yz/BDUdX8PeM9Jvmt5pvMt5raMHewGCy7mAZSNp4PHf2+laayK4w6hseozTuLl0Vj580NL34vfG6x8XQ6VcWnh/ShGY5rgFd5jLMgBHBbzDkgEgAc+7PFt1f/Cv49y+LruwkuNG1QFd8XJYMgDKCcAOGXOD1H6fRAAAAAwB0ApGVXGHUMPQjNFw5D5V+MvjC78f2Ok6za6HeWGh2ryQwXV2ArTyyBWOACeAIxyMjrz2r0D48eFtRvYdA8Sabp51JNLO26tfL37kJVgSvdcgg8d/SvawAAABgDoBRRcOTe54JD8TfhDPYRPH4Gt5L6UhRYx6NAz7icYB6H+ft2q58dwk/wR0Gay0xtOgN5bS/YxFs+yq0EnyFQMLgsFx68V7YIo1bcsahvUDmn0rj5dLHz58UY3f9mbwayIzBDZMxAztH2aQZPoMkD8RR8UY3f9mbwayIzBDZMxAztH2aQZPoMkD8RX0HRTuLkPnz4oxu/wCzN4NZEZghsmYgZ2j7NIMn0GSB+Ir0C3Rn/Zr2IpZm8KsAoGST9lNeh0Urj5dT54+E0t7b/s9eL5tMuEtrpLiYxyyOEC/uY8/MeAcZwT3xXKeFofBsvhm0udY+HXiDUHRGE19ZSStDMQTk8EAe4HTFfRXxH8K3HjTwHf6HZXKW1xPsaN5M7CyMGAbHODjGecdcHpXAaI3xi8NeHrfw5Y+F9InS0VYLfUPtSBFTj5mTeCxHPOB9D3dyHG1kdx8OfGfhzxf4d/4paI2cNjthexdAjW4528AkYODgg9j3Brr686+EHwzl+HekXh1G5juNRv2QzeTnZGqg7VBPU5ZiTgdfavRaTNI3tqFFFFIoKbJ/qn/3TTqbJ/qn/wB00AjL+E3/ACJA/wCvmT+ldvXEfCb/AJEgf9fMn9K7enS+BE43/eZ+oUUUVocgUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABUD3trHL5clzCj9NrSAH8qnr5q8BeAfDvxB+LnxKHi2ye/+xas4gP2iSPYDNMCPlYZ4RRz0xQB9KKyuu5CGB7g5pa+e9f0WH4L/FrwYvgm4urfS/EN0bW80uS4aSI/PGhcBsnOJAQeuV64JFe+X9/aaXYTX2pXMVrawKXlmmcKqD1JNAFiivPYvjn4AluIkOsSRQzSGOO7ms5kgdh1xIV2/j0r0BJEkiWSN1eNhuVlOQR6g+lADqK871X48fDvR9Vl0+618STQtska3t5JUU+m5VIP4E13Glatp+u6XDqOj3kN7ZzgmOeBwytg4PI9CCCOxFAFyiuD1P40+BtLv7m0k1Z7l7T/AI+Xs7WWeOHnHzOilR0Peut0XXNM8R6VFqeh30N9Zy52TQtkEjqD6Eeh5oAv0VyNx8U/BtppupX93rcUEGm3j2Nx5iOreen3kRSMuf8AdBpvhD4q+D/HN9JZeHdWE13Gu828sTxOy+qhgN2O+M470AdhRXinxf8AjFZ+Hte0vR9E1yW1vrLVYf7XiW3YgWxUMwyVIPBH3ea9E8HfEXwx49a8XwtqDXhsghn3QSR7d+7b99Rn7p6elAHT0Vw/if4x+B/CGqNpus60ovU4kggieYxn0baCAfY81reEvHvhrxxbSTeGNVivPKx5sWCkkeehZGAIHHXGKAOiorH1XxXo2ia5pmkaneeRear5n2RCjEP5ahnywGFwD3IrI8M/FPwj4w8RXOieHdUN5eW8bSMFhcIyqwUlXIwRlh0POeKAOvooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/wAnX+Bf+uNp/wClM1fRFfO/xV/5Ov8AAv8A1xtP/SmavoigAooooAK+d/2p/wDkK+Bv+u11/wChW9fRFfO/7U//ACFfA3/Xa6/9Ct6APoiiiigAooooAKKKKACvOIv+Swa9720X/oEdej15n4wmHhr4i2usyA/Zb638mcgZ2kcZ/RT+BrKrsn5ndgtZSh1aaX4P9Dq6KZDNHcQpLBIskbjKupyCPY0+gQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABTZf9U/+6adWD4t16HRtGlXeDdzoUhjB5yeN2PQUm0ldl04OclGJY+E3/IkD/r5k/pXb1zvgPSzpPguwgf8A1kiec/1c7v0BA/Cuiq6atFJnPipqdeco7XYUUUVZzhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFfMvgTR/F2rfFj4l/wDCF+KItAeLV5PP8yyjuBPmefaPmB24w3T19q+mq8d+D+g6vpPxS+Jd1qmmXdpb3up+ZazTwsiTqZp2yjEYYYZTx6igDH+H1reSfGmex+LN1NqHjCwg83SpSy/ZDCRy0Sqq/P8Ae5I6A8ArVz9oXU5f7X8FaDJaXd/pt7ftc3tlZRGWa7WEpiNVBG7IduPXB7Vq/Gnw/q0epeGfG/haxub7VNCvAstvaRGSSa3c/MNoBJ7j2DsavfFvwXqPivT9B8Q+GYFk1vw/dpfWtvOdhmXKs0ZyQA2UQ8+hHGaAMnWfiNa634autDu/hV43ayuIGh8r+xMKoIwMYbjHUY6Y4rBXU/Evhn9j65/tKC70/U4ENki3UbRSxwvcCMcHBHyMQPoDXXW/xmvWjjguvhr4yTUThHhi0/fEH74lJGVz/Fgcc1u+IdCvfiR8JrzS9bsP7FvtRhJW2abzfs7q+6Lcy4z91CQOmSOaALnw88NWHhv4daPplpBEF+xxtOwQDzpGUF2Prkk9e3Fc/wDFqZPA3wP10+GII9PG0RosIKiPz5gJGXHQ/OxGOhNc94W+JPifwpoFr4a8T/D3xJd6np0aWsdxYW/nw3KqoCuZOgz3I3dM9cgdhqOhat8RvhLe6T4usIdH1HUo2K26SeaLYh90JZgeSNqk49x7UAcT4E8fweE/A2l6PZfDPxpIsNsglmh0T5biQrl5Pvc7iSfoatfBaHUIPG/jOSHw5q+gaBfyxXdpbalaNBskIPmbQeOT2BIAC9OBSeF/iF4g8DeHLPwz4u8AeI57rS4EtYrnR7UXUM8aKFRtwIAOB059eM4HofhDxJqviaO6udQ8M3ug2i7Pso1BgJ5s53Fox9zHy9SScnpjkA8t+C/hfSr/AOIfjrX762FxfWWvXMNqZfmWAM7FmVezHgbuuBgYyc2/jlZQaN4t8DeLbBfI1KLWI7SSSNQDNE3JVj3wAwHs7VzXw913xR4Z8ZePNT0zRH13Qf7euEvbWzwbuFw7kSIp++McFc56HjBzv3SeIvjF8QfDtxJ4c1Pw/wCF9Auftkr6pGIZbmZcFVCHJxwBkcYLc5wKANP49Rpu8DtsXLeJbYE469a9A8ZajJ4e8Ca9qunpGtzZ6fPPFleN6xsVzjqM1xvx00LWdU8M6RqXh2wbUbnQ9Vi1B7SPJeVEByFA5Jzjgc4zW1oniBfiboGs6bqHhnW9Es5rc20h1SDyWl8xWVggzk4Hf3FAGP8AATw7a6T8LdP1IqJdR1ffeXd02WeRmY4BY88KB+OT3NYXje1t/Cv7RvgjWNJh+zy6751nfLAnE4G0BmAH+2pJ/wBgE9KreDfEPir4RaQ3hHxP4Q1nW7Szkf8As7UdFt/tCyxli2GGflwW4yc84xxk62h6P4g+IXxT0/xv4h0m50LRdFgZNL0+8AE80jghpHXqo56Huq47kgGb8cNEt/EfxR+HWj3ryJbXk11HN5TbWZD5W5c9sjI+hr2Sx0rT9Mt4YNPsre2it4/KiSKMKET+6MdBwK85+IukalffGL4c3tlp91cWtnPcm5nihZkgBCYLsBhc4OM9cV6jQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXk/7PPi7XfGXgbUb7xLqD39zFqTRJIyKu1PLjbGFAHVj+desV4d+yn/AMk11X/sLv8A+iYqAPbpkaSB0jfy2ZSFcDO0461zPw98P614a8OSWXiPxLL4juWuXlS7lBBRCF/d5LEkAhj143Y6AV0tx/x6y/7h/lXi/wADtQuNJ/ZuvNRsYDcXNmt7PDCASZHQFlXA9SAKAPaJLmCF1SWaONm+6rOATSpLG5wkisfQHNeLfDX4aeF/HPw7t/EXi6L/AISDV9ZV5Lq+mncuh3sNiYbCbenGOR6YFV/2dfCmiWE3iPUbO28y80/U7jS4dQ81z59uCjDK52Z4ByAOMfiAe4TXENuAbiaOIHoXYLn86erK6hkYMp5BByDXluvfCr4bR3dzq3xAvTeXNxK0hutX1d4RGrOSI0w6gIM4ArnPg7qNjo/xl8VeC/C2oLf+F47VL6xaO6E8cTfut6owJyN0xB56pzzmgD3Xcu7bkbsZxnnFMS4hklaJJUaRPvIGBI+orwvx1pOqa9+01ZaVpGrSaR9r8PBLq6hH70QebIXWM/wucABu2SfY1fiz8LNB+Hvgz/hMvA/2rSdZ0m5ikFyt1JIZdzhDneSM5bJ7EZBBBoA+gqjFxA0xhWaMyjqgYbh+FeZ/GLxzqHh34RQX2jiRNT1lobW3aJctG0iFiR3ztUgYzyRUGn/s+eFbLSoZEkv4vESL5h1uO8k84TnkuF3bTyTxjkd6AIfgaSdc+ImT/wAzLP8A+hNXrMtzBAyrNNHGzfdDuAT9K8b/AGeILu2bxxBqNwLu9i12RJ5wNolkGQzY7ZOT+Nc38OdP8G+MtR1q2+K8aTeN5b2SK4tdSmaMomPlSDDAYHP3TuHXpgkA+jKKy/Dnh+z8LeH7bRtMac2lqGWETymRlUsWC7jzgZwPQACtSgAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/J1/gX/rjaf+lM1fRFfO/wAVf+Tr/Av/AFxtP/SmavoigAooooAK+d/2p/8AkK+Bv+u11/6Fb19EV87/ALU//IV8Df8AXa6/9Ct6APoiiiigAooooAKKKKACsHxb4eXxDpPlKqtNFlow3RvVfbPH5Ct6igabTujwn+z9V0Sd4LK/urDB+aFs4H4U/wC1+If+g7N/3zXt81tBcAC4hjlA6B1DfzqH+y9P/wCfG2/78r/hWXsonX9dr9Wn6pf5Hi32vxD/ANB2b/vmj7X4h/6Ds3/fNe0/2Xp//Pjbf9+V/wAKP7L0/wD58bb/AL8r/hR7KP8ATH9dreX3L/I8W+1+If8AoOzf980fa/EP/Qdm/wC+a9p/svT/APnxtv8Avyv+FH9l6f8A8+Nt/wB+V/wo9lH+mH12t5fcv8jxb7X4h/6Ds3/fNH2vxD/0HZv++a9p/svT/wDnxtv+/K/4Uf2Xp/8Az423/flf8KPZR/ph9dreX3L/ACPFvtfiH/oOzf8AfNH2vxD/ANB2b/vmvaf7L0//AJ8bb/vyv+FH9l6f/wA+Nt/35X/Cj2Uf6YfXa3l9y/yPFvtfiH/oOzf980fa/EP/AEHZv++a9p/svT/+fG2/78r/AIUf2Xp//Pjbf9+V/wAKPZR/ph9dreX3L/I8W+1+If8AoOzf980fa/EP/Qdm/wC+a9p/svT/APnxtv8Avyv+FH9l6f8A8+Nt/wB+V/wo9lH+mH12t5fcv8jxb7X4h/6Ds3/fNH2vxD/0HZv++a9p/svT/wDnxtv+/K/4Uf2Xp/8Az423/flf8KPZR/ph9dreX3L/ACPFvtfiH/oOzf8AfNH2vxD/ANB2b/vmvaf7L0//AJ8bb/vyv+FH9l6f/wA+Nt/35X/Cj2Uf6YfXa3l9y/yPFvtfiH/oOzf980fa/EP/AEHZv++a9p/svT/+fG2/78r/AIUf2Xp//Pjbf9+V/wAKPZR/ph9dreX3L/I8W+1+If8AoOzf980fa/EP/Qdm/wC+a9p/svT/APnxtv8Avyv+FH9l6f8A8+Nt/wB+V/wo9lH+mH12t5fcv8jxb7X4h/6Ds3/fNH2vxD/0HZv++a9p/svT/wDnxtv+/K/4Uf2Xp/8Az423/flf8KPZR/ph9dreX3L/ACPFTd+ISCDrs3PoK0vDPg2fV9TW6u2mnXdmW6mJOfYZ6n+VesDS7AHIsbYEf9MV/wAKtABQAAAB0ApqnFO5MsZWlHlvZPskvyERFjRUQbVUYAHYUtFFaHIFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBwXw08F6p4S1PxbPqrW7Jq+sS3tt5Llv3bEkbsgYPPSu9oooAKKa7pFG0kjKiKCzMxwAB1JNUNP8Q6Lq1w0GlavYXsyrvaO2uUkYLkDOFJOMkc+9AGjRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRWZeeJdC067NrqGtada3C4zDPdojjPT5Sc0AadfNv7OXjzwv4W8B6jZ+IdbtNPuJNSaVI53wWTyoxn6ZB/KvpKuem+H3gy4nkmuPCOhSyyMXeR9NhZmYnJJJXkk0AO0bxl4d8WW16PDer22om2QecIHzs3A4z9cH8q8v+Cj6nH+zTevoEayamovDaI2OZedvXjrjrxXrmk+GtC0Dzv7C0XTtM8/Am+x2qQ+ZjON20DOMnr6mue8c+FLh/hRrWg+ArW30u5mgP2eCziSFWO4F1AGFBdQy54+9nNAHgfhm8+G1p4M0v8AtLxH4z8OybPK1C1tJJkt7iYcSdFYcnPAIODyAa98+GuseCLzwoLH4bXNtJZ2QwYEDLIjHPzOr4bLEH5j1x1rlPA3xFs/C/gjTNB1Xwj4l0+90y2S2mij0iR1lkVQGkVlGCGbJyfX8am+HOhatqfxT1/x/faPP4esNQtUtrSwuFCTTY2EyyoCcN8nGefmPpkgHF/C2bwFqNnrOo/FW60ubxW1/Kl5Hr0qZiCnhI1kOMA7h8vPboBVj4VtpE37TniS58MWS2eiz6LusljtvIjkQPboXRcD5WdXII69a9r1DwZ4Y1bURqGqeHtLvLwY/wBIns43fjp8xGeKvx6Tp0Wp/wBpRafapfeSLf7UsKiXygc+XvxnbkA4zjigDy29/wCTutO/7Fo/+jJK0P2h/wDkhut/79t/6Pjr0JtJ05tXXVW0+1OorF5K3hhXzhHnOwPjdtyScZxzTtR02x1exkstWsre+tJMb7e5iWSNsEEZVgQcEA/hQB5J8ZtC1HV/gdpN7oodrvRmtdQCxruYqsZUkD23bu/Cmukj+NvgZ/DMWrDXLd5ZI1I0+I77oyHgRiIfMTk46Y75xzXfJGkcSxRoqxqu1UUYAHpj0rHs/BnhjTtVOp2Hh7S7a+JLG5hs41kyeSdwGcmgDzH4BXksj+Pr2+tZLOVtdmlmtn5aFvmLIcdSDkfhWh4n1n4M/EDw817ruraJKTDlJmnWG8jHXAHEmePu459DXp9pplhYS3MtjZW9tJdSebcPDEqGZ/7zkD5j7nmsmfwH4RutRN/c+GNHluy4czvYxlyw6EnHX3oA5L9n+61+8+ElnN4leSRjM4snlz5jWwxtLE9ed+D/AHdtemUiqqIERQqqMAAYAFLQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/ABV/5Ov8C/8AXG0/9KZq+iK+d/ir/wAnX+Bf+uNp/wClM1fRFABRRRQAV87/ALU//IV8Df8AXa6/9Ct6+iK+d/2p/wDkK+Bv+u11/wChW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGd4hUv4Y1RV6tZzAf8AfBr5X/ZXkVPitfK3WTR5VH182E/0r62miWeCSJ/uyKVP0IxXxz+zhMbH4221tJ8rzW1xAR7hd2P/ABygD7JooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr4z+Pw+0fH6+ij5f/AEVMe5iT/EV9mV8ZfEM/2z+1HPCvz+brFpbYHfHlR4/SgD7NooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8nX+Bf8Arjaf+lM1fRFfO/xV/wCTr/Av/XG0/wDSmavoigAooooAK+d/2p/+Qr4G/wCu11/6Fb19EV87/tT/APIV8Df9drr/ANCt6APoiiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACvi+ykHgH9qHNyfJgttcdCzcBYZmIDH22SA19oV4f8aPgRe+O/EEfiDwxdWsF88axXcN0zKsu3hXDAH5sYBB4wB+IB7P/aVj/wA/tv8A9/V/xo/tKx/5/bf/AL+r/jXyT/wy949/57aP/wCBT/8AxFH/AAy949/57aP/AOBT/wDxFAH1t/aVj/z+2/8A39X/ABo/tKx/5/bf/v6v+NfJP/DL3j3/AJ7aP/4FP/8AEUf8MvePf+e2j/8AgU//AMRQB9bf2lY/8/tv/wB/V/xo/tKx/wCf23/7+r/jXyT/AMMvePf+e2j/APgU/wD8RR/wy949/wCe2j/+BT//ABFAH1t/aVj/AM/tv/39X/Gj+0rH/n9t/wDv6v8AjXyT/wAMvePf+e2j/wDgU/8A8RR/wy949/57aP8A+BT/APxFAH1t/aVj/wA/tv8A9/V/xo/tKx/5/bf/AL+r/jXyT/wy949/57aP/wCBT/8AxFH/AAy949/57aP/AOBT/wDxFAH1t/aVj/z+2/8A39X/ABo/tKx/5/bf/v6v+NfJP/DL3j3/AJ7aP/4FP/8AEUf8MvePf+e2j/8AgU//AMRQB9bf2lY/8/tv/wB/V/xo/tKx/wCf23/7+r/jXyT/AMMvePf+e2j/APgU/wD8RR/wy949/wCe2j/+BT//ABFAH1t/aNj/AM/lv/39X/GrBIVSzEAAZJPavi/xD+zz418N+Hr3Wb06bLbWMRmlWC4Zn2DqQCo6Dnr0FctdfEXxDefD618HzX0rabbztLy53OmBtiJ/uKQxA/2vYUAfeP8AaVj/AM/tv/39X/Gj+0rH/n9t/wDv6v8AjXyBp37NfjrUtLtb6JtLiS5hSZUluWDqGAIDDZweeas/8MvePf8Anto//gU//wARQB9bf2lY/wDP7b/9/V/xo/tKx/5/bf8A7+r/AI18k/8ADL3j3/nto/8A4FP/APEUf8MvePf+e2j/APgU/wD8RQB9bf2lY/8AP7b/APf1f8aP7Ssf+f23/wC/q/418k/8MvePf+e2j/8AgU//AMRR/wAMvePf+e2j/wDgU/8A8RQB9bf2lY/8/tv/AN/V/wAaP7Ssf+f23/7+r/jXyT/wy949/wCe2j/+BT//ABFH/DL3j3/nto//AIFP/wDEUAfW39pWP/P7b/8Af1f8aP7Ssf8An9t/+/q/418k/wDDL3j3/nto/wD4FP8A/EUf8MvePf8Anto//gU//wARQB9bf2lY/wDP7b/9/V/xo/tKx/5/bf8A7+r/AI18k/8ADL3j3/nto/8A4FP/APEUf8MvePf+e2j/APgU/wD8RQB9bf2lY/8AP7b/APf1f8aP7Ssf+f23/wC/q/418k/8MvePf+e2j/8AgU//AMRR/wAMvePf+e2j/wDgU/8A8RQB9atqlgilmvrYKBkkyrx+tfHnw5z43/aXg1JQWil1SfUiSPuqpaRf1Cj8avf8MvePf+e2j/8AgU//AMRXsHwS+C8/w4mvNW165trnVrqMQotsSyQR5BI3EDLEhc8cbe+aAPX6KKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/ACdf4F/642n/AKUzV9EV87/FX/k6/wAC/wDXG0/9KZq+iKACiiigAr53/an/AOQr4G/67XX/AKFb19EV87/tT/8AIV8Df9drr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigCnq9gmq6JfafKAUu7eSBge4ZSp/nX58aJpran4l0/S2BDXV3FbkdDlnC/1r9E6+HPCGngftD6bZbeIPEI4/65zZ/9loA+4lUIoVQAoGAB2paKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8nX+Bf8Arjaf+lM1fRFfO/xV/wCTr/Av/XG0/wDSmavoigAooooAK+d/2p/+Qr4G/wCu11/6Fb19EV87/tT/APIV8Df9drr/ANCt6APoiiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAorn9c8eeFPDcpi1zxDp1lMOsMlwvmf98D5v0rnv+F6/Dkvsj8RGYgZPk2NxJj67YzQB6DRXn//AAvH4ff9Bq4/8Fd3/wDGqP8AhePw+/6DVx/4K7v/AONUAegUV5//AMLx+H3/AEGrj/wV3f8A8ao/4Xj8Pv8AoNXH/gru/wD41QB6BRXn/wDwvH4ff9Bq4/8ABXd//GqP+F4/D7/oNXH/AIK7v/41QB6BRXn/APwvH4ff9Bq4/wDBXd//ABqj/hePw+/6DVx/4K7v/wCNUAegUV5//wALx+H3/QauP/BXd/8Axqj/AIXj8Pv+g1cf+Cu7/wDjVAHoFFef/wDC8fh9/wBBq4/8Fd3/APGqP+F4/D7/AKDVx/4K7v8A+NUAegUV5/8A8Lx+H3/QauP/AAV3f/xqj/hePw+/6DVx/wCCu7/+NUAegUV5/wD8Lx+H3/QauP8AwV3f/wAao/4Xj8Pv+g1cf+Cu7/8AjVAHoFFef/8AC8fh9/0Grj/wV3f/AMao/wCF4/D7/oNXH/gru/8A41QB6BRXn/8AwvH4ff8AQauP/BXd/wDxqj/hePw+/wCg1cf+Cu7/APjVAHoFFef/APC8fh9/0Grj/wAFd3/8ao/4Xj8Pv+g1cf8Agru//jVAHoFFef8A/C8fh9/0Grj/AMFd3/8AGqP+F4/D7/oNXH/gru//AI1QB6BXx54Y028H7WrwfZpBJHr1zM67fux7nbd9NpBz7ivoP/hePw+/6DVx/wCCu7/+NUwfGr4crcNOuqyiZlCtINJutxA6AnyulAHodFef/wDC8fh9/wBBq4/8Fd3/APGqP+F4/D7/AKDVx/4K7v8A+NUAegUV5/8A8Lx+H3/QauP/AAV3f/xqj/hePw+/6DVx/wCCu7/+NUAegUV5/wD8Lx+H3/QauP8AwV3f/wAao/4Xj8Pv+g1cf+Cu7/8AjVAHoFFef/8AC8fh9/0Grj/wV3f/AMao/wCF4/D7/oNXH/gru/8A41QB6BRXn/8AwvH4ff8AQauP/BXd/wDxqj/hePw+/wCg1cf+Cu7/APjVAHoFFef/APC8fh9/0Grj/wAFd3/8ao/4Xj8Pv+g1cf8Agru//jVAHoFFef8A/C8fh9/0Grj/AMFd3/8AGqP+F4/D7/oNXH/gru//AI1QB6BRXn//AAvH4ff9Bq4/8Fd3/wDGqP8AhePw+/6DVx/4K7v/AONUAegUV5//AMLx+H3/AEGrj/wV3f8A8ao/4Xj8Pv8AoNXH/gru/wD41QB6BRXn/wDwvH4ff9Bq4/8ABXd//GqP+F4/D7/oNXH/AIK7v/41QB6BRXn/APwvH4ff9Bq4/wDBXd//ABqj/hePw+/6DVx/4K7v/wCNUAegUV5//wALx+H3/QauP/BXd/8Axqj/AIXj8Pv+g1cf+Cu7/wDjVAHoFFef/wDC8fh9/wBBq4/8Fd3/APGqP+F4/D7/AKDVx/4K7v8A+NUAegUV5/8A8Lx+H3/QauP/AAV3f/xqk/4Xr8ORJsk8ReS2M4msbiPI/wCBRigD0GisHQ/HPhbxLJ5eheINPvpevkxXC+Z/3x1/St6gAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/AMnX+Bf+uNp/6UzV9EV87/FX/k6/wL/1xtP/AEpmr6IoAKKKKACvnf8Aan/5Cvgb/rtdf+hW9fRFfO/7U/8AyFfA3/Xa6/8AQregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiuO+IHjSbw3a22l6DAt/4m1ZjDptl795X9I16k+2PUgAl8ZfECy8KS2+nWtrNrGv3vFnpNocyyf7THoiDux9D1wa56LwP4u8aN9q+Iuvy6bZtymhaFMYo1HpNMPmkPqBxkZBroPA3gK18I28t5dztqfiC+AbUdVn+aSdu6qT91B2UdgK62gDmtB+HXhDwzGF0bw9YwMP+WzRCSU/WR8sfzrpaKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKQgMpVgCCMEHvS0UAcvr3w08G+JVP9r+HbGSQnPnxR+TKD/wBdEw361zUng3xn4Ib7T8P9cl1qwUfPoWuzF+PSGfqh9AePXNem0UAcr4N8e6f4u+0WbQTaXrVidt7pN3xNAfUf3kPZhxyOma6quP8AHXgKHxQkWp6VOdK8TWA3afqkPDKRz5b/AN6M9CDnqfcF/gHxo3irTri01W3/ALP8Q6W/kapYHrHJ2dfVG6qf1PWgDraKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/AMnX+Bf+uNp/6UzV9EV87/FX/k6/wL/1xtP/AEpmr6IoAKKKKACvnf8Aan/5Cvgb/rtdf+hW9fRFfO/7U/8AyFfA3/Xa6/8AQregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAGTTR28Ek07hIo1LuzdFAGSa83+GFtL4o1rVfiRqkRV9VP2XSI3HMFjGxAOOxdgWP6cGtD4y6ncWPwyvrTT/APj91d49Lt+cfNOwQ/8Ajpauu0bS4NE0Ox0qzGILK3jt4/8AdRQo/lQBdooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiisbxR4s0XwbosmqeIr6O0t04UHl5G/uovVj7D+VAGzRXj2gfH6HUGuLrWPDt9YafKQ2nOuGknjyQXYMVAGRwVJ5BHbJ1m+Ovhletjq3/fmL/wCOVapyaukenTynHVaaqQpNxezPS6K85tPjb4cvJCkNhq24djDHz+UldLZ+L4L1FdNL1KNW6NJGg/8AZ6lpp2Zw1aU6M3TqKzXQ6GiseXxLbxAlrW7OPRF/+KrFvPiZpVi2LnT9TX38pMH/AMfpGR2VFcA/xk8PR9bPUj9Io/8A4ur3h/4laX4lv2tdO0/UhsGXlkiQIv1Icn9KAOxorEu/FFtZXQgns7wE/dYRqVb6HdWla3q3UW8RSxf7MgAP6E0AWaKQEGhmCjJoAWiqz30Uf3g34AVUm8QWkOd6THHoB/jQBqUVVg1CG4tEuIw2xxkAgZpkuqwRD5lf8AP8aALtFYdx4rsLYHzIrg49FX/Gs3/hZOjfbobUwXoeZgisUTGT/wACoA66iuL8R/E/SvDFwI9R07VGRjhZoooyhPpkuD+lZA+O/hc9LPVf+/Uf/wAcoA9LorzJvjz4XXrZar/36i/+OVF/w0B4W8wImn6y7E4AWCI/+1KAPUqK4WL4taJJCsjWGpxFv4HijBH5PSN8XdBTraaj/wB+o/8A4ugDu6K4O1+LuiXl9FaW2n6pJNK21VEUf5/frtY7tZFBEbj2IHH60AT0Vz+t33iiNwPDumaTOmRukvr+WI47jakLfnu/CugGcDPWgAooooAK8y+JET+DvFGlfEiwT9zbYsNcRRzJaSMAJMDqUbB/EdhXptZviLRofEXhnUtGuseVf20luxI+7uUjP1Gc/hQBooyuiujBlYZBHQilri/hHq8+sfC7SGvci8s0axuQTkiSFjGc+5Cg/jXaUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/ABV/5Ov8C/8AXG0/9KZq+iK+d/ir/wAnX+Bf+uNp/wClM1fRFABRRRQAV87/ALU//IV8Df8AXa6/9Ct6+iK+d/2p/wDkK+Bv+u11/wChW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAee/FNfP1fwFbMMq3ieCUjsdkcjD9a9Crz/4lf8jN4A/7GBf/AETJXoFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFIzBFLMQoHJJPSgCO6uoLK1kubuVYoYlLO7HhQK4iT4uaJHeeWbW9NvnHniMY+u3OcVJ8TboXPgaaTTp0njhnja48lw2Ez3x2zivGJtQikeJflAB6jvQB7FqHxe0Kyn2w295dxD700cYC/gGIJ/Kuo0XxLpWv6R/aem3aNbDhy52mM9wwPQ186X95CbfAABA5561T8OXBNvdx+e4t5ZgWiDfKxUcEjv1NAH1FaapYagzLY3tvcsv3hFKrEfkatV8uXWoy6NeJqGlTmC4t23RyLxnHY+x7ivSPGXxpj03QtOHh1IrjUdRtFuDI3KWwYdx3bOePbmrhTlUlyxNqFCpXmoU1dnrdFfHd74m8Sa60iajrd9dBssY3uCE4/2QQB+VQ6Z4w8Q6LFPZ6bq9zFbXcbRSxGQsuCOSM/dPuOa7XgJpbnsvJKqjfmVz6Z1b4r+DNGv2s7zWEaZG2uIInlCH3Kgiuj0jWdO17TkvtGvIry2fgSRNkZ9D3B9jzXxbdQeREJJfkUruDE4GPWr/wAOtb8Y6j4jm8NfD/UPsT6qhSe5c4WNV5Mg/usACMgZ545wamvhoUo6PUzx2X0sNBNS1/M+iviJ8YdN8HXK6Lo1u2ueJpyEh062y2xj08zHI/3RyfYc1z3hf4Q6r4o1mPxZ8ZLr+0r/AO9b6OD/AKPajqAwHB/3Rx6lq6/4d/CjQvh9bma3B1DWJgTc6ncDMjk9Qv8AdUnt1Pcmu5rhPEPBvjbGsfjGySJAqJpsaqqjAAEkmABXm8dtNd3CW9vG0ksh2qoHJNfU2taTb3t6s8sCSOIwm5lB4yTj9a5+68O2izpPHbok0ZyjqoBU10xr8sbWPs8JxOsNhYYdUr8qte/6W/UzvAPw5ttCtUu7xRNfSqCxYcR+wrvhYqF4XFc1ZeIbrTZxFfr58LHG8feX/wCtXWRXcMwG1xz0zxXO227s+Sr16mIqurUd2zOubNSp4rl9Y0uKaJg6A59RXbXIwDXP6iBg0jE8R8S6S+nSGSIExMenpXqvw30FdK8Ows64nuP3khPXmub8QWkdyfKkHysQDXomisqWsKr0CAD8qANe8tRNYsqgFl5UmqFpc/IATzWrHIMetYuqwm0mNzEMxMfmx/CaANMXA9ailuMjrxWOt+CvDUyS+GOTQBaubgDPNYGo3ICnmn3V/wAHmud1C+zkA557UAd7prmLRLZGPIXrVW8uBg81XjuhDZRR7s4QVmX198p5oAp6pc/ewa5GMtc+JrFY+Ssyt+ANaGqagArfNXO2moTWeoi7gRXkXIUP0Ge9AHpfiJbW4t5EvUjeEnkSdK8f1/w7pscjSadOYif4Q2R+ArTvJrvUZzNe3EkrnuTx+VUJrQEHIoAx4fC/mkGS9JXuAtb+maVY6Y4eGLLj+N+TVBJHs2x96Pv7VoJcK6AqQQe9AGrqN1bzLGII9pA596yJm60rzCqc03BoA9E+EmiC5v7nV5UDCL91C3o3f9K9jijCrXG/DqxXT/CNmpG2SYeZIM9zXaKwxQAMoKmnUhIA5NLQAUUUUAFFFFAHnnwiIWPxpAudkHi2/RB2UHY2B7fMa9Drzv4Rf6zx3/2OF9/6DFXolABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/5Ov8AAv8A1xtP/Smavoivnf4q/wDJ1/gX/rjaf+lM1fRFABRRRQAV87/tT/8AIV8Df9drr/0K3r6Ir53/AGp/+Qr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHn/xK/5GbwB/2MC/+iZK9Arz/wCJX/IzeAP+xgX/ANEyV6BQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXlPxT1qd9ah0YSPHapEJZFU48xmJxn1Ax0969WrA1fQdMvPEunandxbrmBWVCfukcEZHcgk4+poA8ZtfD3ii3vUuvD2nTBzn76hY5V7h1PBUjsa1tK+Fem3NxNN4gnkt7nDSrZQNsQY5OD/QV7DPf2li7yXMscSBQd7MAAPrXAeKvEVhdsjaZeQyyrINpjcEg5oA8pkbT7CxMsiylScLtheZvx2gnp36Vx134x0i2u2NldYBOHUwuvP4itfWdbWLVLqLR3Etosh8uR16jPb29K8j1NzJq947Y3NO5OPXca2lSnTSlJbnXVwtbDxjUqR0Z3s+tyakhKrII/dGXP5gZpLULIPklC7RxuOMY7Uy53xZWRcFeCCMEVBb287LJOqOIz0O04Ne1CMaVoo+yo06eFShFINQ1AW8cl1ctkE9hyx9AKoDWbQQfamc+ip/Fn0qPU76O7h/s22t/tdxIcAD+BvXPrUMOg/YU8y6w04/h7If6msZ1KjnanqkcdbEV5VnGhZxS1fZ/r6DJGudWIkvWMNqOViB5P1r6H/Zv8Fy2ovPFF3aeTDLH9nsNy4LLnLuPY4Az35ryz4aWsN18SvD8ElpHerNdASRToHUrg5ODxwMn8K+zkRY0VEUKqjAUDAA9K4a8nHTdvqeLjpuHuPVvd9RaKKK4zyCOVA3btWdcxDB4rUb7prPuXHIoA5nUrcMjYFaelH7Ro8RJyyfKT9Kp6g2VNTeHHH2KdM9HJoAllu5rfIJLp6HrWdd3sc0ZKN06juKvXxHOK5LU3aFzJGenWgDK1u42gsDypzXYaDqKzWcTq2RtFeZaxqHnAoOD3FafhbWfs6C1kbG37uaAPXIrvK8GlluFdCrgFSMEGuZt9VUqDuqY6gMfeoArapbSWRMtqS8WeV7rWHJrAII3citm41EY5aua1VbSYsSmxvVDigCG61dcElqzbS8+2atEu792h3MfXHasi8URykCYv8AWq8V+bbIjIGepoA9CuNWGD83FY19q6hT81ctLrLEfM5/KrFp9kuSHmudw/ug0ANubtrl+Tx6U6GPpTdQlgbUglsirGiDBHepI2wM0ASmPA45qvKoxWlZ3dvDG4nj3kjA9qzJnDE44oAz7leDWal0bWbY33GP5VpXLcGsHUG+VqANV7setVJbkuwCgnkcCqmmQ3N9sG1lj/vnv9K6a20+KFAFXn170Aepad4+0DS7G2t0mZ1SNR8g6HHIrb0/4k+HL6QRfbxBIWwqSjG6vGGgUDpVWe3Ug5AoA+khexzxCSGRZEPRlORWpXy9pPinVvDFyHspmkt+j28hypHt6GvqGgAooooAKKKKAPO/hF/rPHf/AGOF9/6DFXoled/CL/WeO/8AscL7/wBBir0SgAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8AJ1/gX/rjaf8ApTNX0RXzv8Vf+Tr/AAL/ANcbT/0pmr6IoAKKKKACvnf9qf8A5Cvgb/rtdf8AoVvX0RXzv+1P/wAhXwN/12uv/QregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPP/iV/yM3gD/sYF/8ARMlegV5/8Sv+Rm8Af9jAv/omSvQKACiiigAooooAKK4HXPina6feyW+mWLX4iba8xk2ISOoXg5+tbPhHxxp3i6OZLdHtryAAy20hBIB6MD3HvQB0tFFFAEFzLs2Rq21pDgH0Hes/VbKD7MzZIfH3txz+dS65HMbJZrXmeJsomcb89V/z6VxOq+Jb47rYafefaOnl+Q2f5UAS2+v3OmQPcrK0i20uyWNjkSL1z7HHf2r0CKVZoUljOUdQyn1BrxOI3V9LFoiBlvL+b95uBATPr9AK9rt4VtrWKBPuxIEH0AxQBW1jUk0fRbvUJV3LbxF9vqR0H514XqfibXNQkN7Nq9zG+cqkExjVPoor0n4keKrPTNMk0Z4lnuL6FgwZ9qxIeNxPrnoPavn6W6uJo8GUhBx8vGfetIU3N6Hq5ZlWIzKbjSskt29jv7b4zeIbTSVtfItrq4jYj7XMDl17ZUY59+9dZ4D+Ly+I9Xj0bXLWO0vZgfIliJ8uUgZ24PKnHTk5rxJt9vCGjYYcFccH/wDVWZDqs2leINPvo4z5lrcJMAcgHawOM+nGK6Z0oKNlufb47IsBRwns4R9+2jvq2fZxOBk9BWXqVrNqMCPCwi8tt0bYyT/9Y1xJ+MNjc6RFJFpN3500Z8xGZVEZ6cH+L16Cuq8P602t6bHPYgNFjBZumfSuI/NZRcZOL3R5f46vLtvEZ03UZQUtUUhV+6Swzn3rgdedYreV7Zyrqp2uvBrt/iVfWus62JdOYG6tx5Ms6D93IB29yDnmvPGsru6kkkuuba2Iaby8khc4BPoCeM007O4RfK0zmY7sJEwAUh1xyM4+lcJf/wDISuf+uz/zNen6ppdpqF+v9lh/tU7gJBCNwkY9AB7+1eba5Z3GneIdRsr1PLuba6lhmTOdrq5DDI9wa7cTXjWirHsZhjoYulG26O4sLm0l1ES6nOqW8YMkjSHrjt7844qe81bUvGLtaaDG1hpKfLJduMNJ7D/AfiR0rO0jwXf6mf7Q1+Jra3xuigYbS/uR1A/Wu31fw54i0vwhZX8drbWttfSR2tiomVXd5G2oFQ/99fQE1jXq+1nc5MdiXiavN0Wx5nr+mJoMSnTZJk+cI0yyEFuCeccVBpl3PcWLiWR5ZDIfmdixxgdzV3xlFd2itaajby211HP+8ilUqwOD2qv4SspNTu7XT4yR9rvEhygy3zFRwPxrWF4TSv0OujzUqyg39np/X4nsH7OGnTXHxAu71YQ9vZ2TK0rLnazsNoB7EgN+Ga+nqxfCvhPSfBuhx6XoduIol5kc8vK+OWY9yf8A9VbVc1SfPK55mIq+1qOSCiiiszAq3lx5JCkcEdazLicMpIORVzUyARn+7XNXszRElT+FAEd/OArc0/w3Nm3uDjjd1rBv78Mjc4PcVraDIV0dAqcyMWG3kmgC/fS8HJrlNQ33MpihUu57CuvTRri7Ja6zBH+ppZLG3s02wxgED73c0AeVan4dmhieV2zKeQo6CuZFy0MnUq6n8q9U1mMMpOK8m8Tr9hvPNBwrnBoA6HT/ABTsAS4bafX1rYGvoygh+PrXksmoLj71QjVrpWH2aRhzx3oGk27I9Yn1xccPWRd6q0oODgeprhF1nUIzmefcP7uBVgap5i5L5z71Ti47nXiMFXw0Yyqxtf8ArU0b/VfKlMag8/xnvVM3wI61Vmu1lGw4bPY0x7JEt946H3rkr4mFGyfU78ryXEZmpSptJR6vv2JJb4c/NWe93LJKBAzBvUUWlqtz5oZyWToPWprWNhkBNpBwTXbSh7WKmtmd+G4ZxM5r2zUYvtq9P66mppF1Nlo7ly8i9z6VvRzcda5XBgcTK3I+8PUVpQXwdAc06kOSRwZ1lzwGJ5Yr3Ht/Xc2zNxUEkvHWqZuxjrUEl3x1rI8QfczcVh30pb5R1Y4qe6uxg81Z8OeG73xG9zfRqVsrFfMllI4z2A96ANnR7X7NYRRHnaM/nWvGoIrPtXygPrV1JMAUASTRlQCVwD3qjMBWneal9ptki8sKE7jvWTK4xQBnXa8HNfWNfJl0/B5r6zoAKKKKACiiigDzv4Rf6zx3/wBjhff+gxV6JXnfwi/1njv/ALHC+/8AQYq9EoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/ACdf4F/642n/AKUzV9EV87/FX/k6/wAC/wDXG0/9KZq+iKACiiigAr53/an/AOQr4G/67XX/AKFb19EV87/tT/8AIV8Df9drr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDz/4lf8jN4A/7GBf/AETJXoFef/Er/kZvAH/YwL/6Jkr0CgAooooAKgvo5JtPuIoG2yvEyofRiDip6jll8vAUbnb7q0AfMkl8bRZbW4Xy5Y2KSI4wysDyK3/g75198SpLi3OILe0kExzwdxGF9znn8K9O1rwb4a1/UJZ/EWlxtd5H72N2jLr2yVIz+NZ7aVaafc2en+ErOGzMUokVY+AMHksepyOOaAPQqgluHBYQReYV6nOAKq3V1fwYaOKJsnhOcn8arR6nJYQgahbuikks6HdjNAEjG6vVE4h2GNvlXdkH1qnqHiBIYWSQ+W46qeDXnfj34jahBfDRPDdyLdY1BluYyCzk8gKewwRz15rgL288azxJcytqlzbg/ekjZ1x65I5FbKk3G59FR4fxFTDRxEpxipbJvU9BTxlp+keJpNWvd0q28TJFFGAWkkbHA9OM5NZWp/HXXvt4OnabZQW27iOUNIzfUgj9BXDPY307G6kEaOwGU5x/9ajT7P7deO92gTy2+6OhP+FVFU1HXc78HSybDYZyxEueot189l0fqTa7rF54n1S41TVHUzyfwDhUUcBVHtWbMES3BRxk9VHatPV7COGNggCsvoa5NryWWQQxozOzbRj1reFWLVkfS5bnuDr0nTpw5OVbeXyOn8PxxTq002CQ20Z7VJr0Nu8DqQrgdDjrTdP0W4trQ/ZJvMl5Z1Y8HjnHpVC4hvLq8itrkeWsrhfkO4sT2H1rjnLmlc/OsxxcsVi51uZvXT06Hb+CdOfxZex2Kgwx+WJJZMcqv+z7mvYJtBOh+GJrPRJpItkLiNd2csQf1zXO+BPBuqeG4pL2YxK00aqLfBJQDpk11UOojUrlrCdjBcAFgnZx32moPOPn5dQRLZo2AB6Hd1Uir/w51GOL4hG5ldFsobOY3Zf7mwjgH8cflXe/Erwr4C0rTJNd8Qzvph6Freba07egTncx9h7mvBp73UfGOljTNFsBonhhZC7ybc3F8c8F3PJ47D5R7kUAdV4r+Kdjcay9l8KtBto75H3NqiQIqx4PVRjH4nj2NchB8EfF/iiC41iz1TTNVuJmeacCeUzNITlt26MfMSSevNdFYR6ZpGkiz02BIFQ/OT99j6se9d18GtY+xXWsSytttZXjVCehcA5/QigDy8p45nlazuYdNhljGHE8cyN+oq1qc3jPVPEvh/UPFN/ZTafol1DLHZWm5UVUdS2FK8sQMZJr0n4n63DP4jtZo5Nyrb7Gx0U7iR+def6nqv2j5I8zSyEKqryWPQACgDsfitb6h8TvDlnY6E2mi2juFuxczyuJWOx12BFjPGHBzu68Y4zWB8OPhF4g8Kazb6vrdtC0UL74XimbchOBu2FB0GSDnOR07j3bwBoY0DwNptrewpHdrDunO0A7iScE+2cfhWnqmoW4s2G4dKabWxSk47GboPjOwuFktNU1C2hu4X2AySBfNB6EZ7+tdSCCMg5Br5y8Ty20mp3LKF2559K1vAXxq8OeG/Dd3ZeKtay1pNttERWmkdCM7RtB4B9SOuKRJ7vRXip+OfiDxOTH8Nfh/qepK3C3t8PLhB98fL+bik/4RD40+MRnxJ4vtPC9o/W20lMyr7blIP8A5ENAHqmsSbZQP9j+prlNRmAU4yfpT/C3w5/4Q3TJNNstQvNS+0TG6ubu9cF2kYBTjjphF4JJ6810CaXb2i5wJJB/G1AHn82jXF0DLKWhTtxywr0XRbey0/ToDbRkx7Btc8msXUQChGOK1NFk8zRYwf4MpQBenuVkBKtmsi8l6067YxMWXrWNdagD8rcN6UAZeqyjDYNea+J0SYNvAYAjGa7bVbsbWOa4DW7jcrHP3mq6avJHpZVBzx9GKV/eX3dfwOYktYRyEqMyIgwFC+4qeVqsWPh261aNpUIji7Mw612y5Y6n6Zi5YTAxdWyj8jDuJOvNV7JJZ7hlhR5DjOFGa6L/AIQrULi4ePzUEa/8tMf0rf8AC2lx6BJc2s11iS7TY5xhXXrjmuec00fIZnmWHr0nGMr3OWtNLeS2kmZWJQbuP4frVd3vXQCO3dww4YDiuw1rZZW7Q24VI3YFwO/tWUr7cYrCWDp4mXNLSx9Fw5KliMPONBckYu3m9N309Cna6dcabZGa6QiSY5HoPalVGb2+lXrzUZtRMaSYEcIwoHr60xEr0qdNQiox2R9NQpe6ovZFbyB6VC8LKcxHafT1rQZOKgkFOUFJWZGMwNDFUnTqxuivbxajds62trJOY13P5YzgetTponiG7lWG20e7kkc4VQnU10Xw+1Iad4rjDEBZ18s5H4173Z3qbVICg+oArz6kOWVj8izjARwGKdKF+W11c8a8J/BDV9XkjuvFEh020zk245lb2Pp9a9ju/D+l6f4PuNF0q1WGAwsEROCzY4JPetEXm7AzknpQp3XuGPKdfrWZ5B82wu8DNDLxJGxRh7jrVoTe9dN8VvCM+iau2uWERbT7o5m2j/Uv7+x9a4NL0EdaANRpuKqzTcdaqtdj1qpNd9eaAFup+DzX1/XxVdXechTzX2rQAUUUUAFFFFAHnfwi/wBZ47/7HC+/9Bir0SvO/hF/rPHf/Y4X3/oMVeiUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wAC/wDXG0/9KZq+iK+d/ir/AMnX+Bf+uNp/6UzV9EUAFFFFABXzv+1P/wAhXwN/12uv/Qrevoivnf8Aan/5Cvgb/rtdf+hW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAef/Er/kZvAH/YwL/6Jkr0CvP/AIlf8jN4A/7GBf8A0TJXoFABRRRQAVk6venTZ4bqQfuCCjN2U9s/WtamyRpNG0cqK6MMMrDII+lAHnninx1FAgt7SCO6uJBkbz8qD1OOazPAGvvLrl0L/wAvznVfLCgjIzyOSfas34naH/YOu297p9p5djdIE/dDhZRnIPpkYP4GuRsL6e28Q2V3NN9mjWYGSdwWCr3JA5oA+jXkWOaOWQ5BUgegNZmu30ItW5HSsaLUr3UwtnHAxdlDrKD8m3s4boRTpfDN8txBJqd9DLbBxujjUjd+PpQBy3hXwfDqHjeXWdRtx9kVcwRyLxI/9/HcAfrXp2oxQC1IwAMdKZqEKNZq9qypLEPk7A+1cpc6pql6Tb2un3EkvTheB/wLpTbb3NqtepVUVN35VZeSPPfGRhsb9xGqqshyeO9cHFqxivJtgLIOSR616l4n8DS3GkTz6tPi8YZjWJuIsdie+e9eGS3MlvJJBGhYxkhtozjHXNXCKluelleBpYubdaVortv5GxqOuCVPLQ8txzWLbec14J4F3JE4JPqRTbW1m1C5UybwpOBt6n6V6/pfwb1afRY5JGisCRkROCzY98dDW0Yxhq2fS4TA4PAQdTEVLRlpru/RLp3focdBrDC18zy5PLJIDBT19KueFLxZ/HWlXF3EVtY7gEtIMAtg7evvivavC3g7w9pmiC2EAe6j4mknHJbrkdgPSsH/AIRzRbHxwt9fyQw2FrE88izsBGpGPmJPAAyTXM7X0PhqqgqklT1jd29Oh6T/AGlHJAFi+diOAvNePfET4iR+GPEtpaaTG2o+IRJ+50+2+dwxGAHxnGc/d6n0Gc1m6z8TdY8W6hceGvg/F9msQ2y51+VSixg9kJ5HsfvHsBjNa/gjwfoHgW/ge2kN/q0rFrnUp+ZHOCWx12gn8T3JpGZxmq+B/EOos3if4ozm/wBRZf8AR9OVv3VoD2Kjj8Bx6ljTnvoVswiBAo6Edh6V33i7X4pbiNFYtiRWIXk4Bya87161sLnU5LmG3lsbZ2LNGknA98Y4/CgChpcOmalrrf2w9xHYrtEkkHb5hnP4Z6V9H2mk+Frbw3bwWFtaNpyxb4lTBDjGc56kn1r5sufE3hjTBDavdxRQhgJBBmRsZ5JxnnHrXoEPxh1DVtOSw+F/w61C9tkXZHd3iiGFcfT5fzcUAYkzxXEk8syCJnJIjIPy8/d59Kz/AA7qmi+H/H+m3uqvbW9urtuklYKEO04bn3q5f/Df4k+Kw+qeJdW0/QkkBxb2KFn/AEP67zUHgn4N6Al49z4qnOqXAcBYGdgmM4LEjBJz26fWgDv9X/aK8Hwz/Y9Ct9R8Q3R4SKxtyFY/VsH8ga5XVfE/xQ8awyHSfDFj4Zsehlv5d0yfUHkf9+69s0zTNC0HTPI0TTrOwi28pbQrGD9cDn8a898YakLe+zbPhnBVgO/H+NAHil38PtRvrmM674ilu95JlRAQqkHHA6H8hWgPC2meHCk+lRLHdwfMlw2JGz6/NkfpUFz4nmj1WQSxlYUJUeuc8k1T1PxKkkRWMlmbgAdTWsqU4vVHTPC16TV4vU+lvhV43k8beE2mvURNQspTb3IjGFYgAq4HYEHp6g129eafA7whf+F/Bs9xrMTQXuqT+e0DfeiQDCgjsepx716XUStfQyqpKbsRTyhBt7kVlXMo5qbVciUFTg7ev4msK4vsEh+Pf1qTMr6hKNpzVzw9cF9Hfc2dspA+lc9qN4Npwa1dEkSLRU2ZG8ljnuaALd9Jwa5LVpsZIOK27+7GDzXIavdgKxz1oAwdR1FpCYyeR1NcdqV4JpyqHKLwPermu6gYRsQ4MmSfUCuaef3rpoxt7zPt+HMEqa+uVN3pH9WSSyYrstB1uBNJSNWAZV2sK8/mn96rRyzyTiO2ZgzHHBq6i5tDszmj9ehGEXqnoep2etwq0oDrknpWDrOqRyXUZQbirZwKyo41hgCFssB8zZ5JqCXABxUewfVnnx4VqRknVqK3W2/p/wAEu3N7PqjkorbVI6DPNa97pJg09J03I5TJRuorG0DWY9LuXWbASTByexrZvtZTUpFgSUAycbmPArNuVN2ieVVqYrKsTKnhZNLf1MeAjFWlbFV3t5bKcwTD5h0PZh6inhj6V3xkmrn6vg8TCrQjUTWqNWWTTk0fLNi4J4FYvnxyEhGyR2pqXEZmYycj09qzbubN3H5JwxcDj0rxf7Uk8R7NR929vMqpNUoOV73ZoCd7edJojtdCGBr1/wALeL49SsUYtiVRh1J7145NE3PNO06/lsp1eJyrKecd69SpDnPm85yxY+KW01s/0Z9P6NOJgJmPX7tW7W43XMhJz83JrjfC2vx3+lQTRkDKjIz0PpWxb3givZEzwTla4WrH5fOEqcnCSs1odbKlve2cltdxrNBKu10YZDA15D4p+CQ8x7nwldiJWOfsk3IX2U/416VDegqPmqRr0YPNBB8133gnxdp83lXWjyhyMjawYEfhUNr4E8U6i3GntDEDtaSRsBfwr6NuL3g/NxWFqF9kH5qAPMtP8AWmjAXGpyC7uV5C4+RT619K14LrN8AjYNe9UAFFFFABRRRQB538Iv8AWeO/+xwvv/QYq9Erzv4Rf6zx3/2OF9/6DFXolABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/5Ov8AAv8A1xtP/Smavoivnf4q/wDJ1/gX/rjaf+lM1fRFABRRRQAV87/tT/8AIV8Df9drr/0K3r6Ir53/AGp/+Qr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHn/xK/5GbwB/2MC/+iZK9Arz/wCJX/IzeAP+xgX/ANEyV6BQAUUUUAFFFMlkWKMu35DuaAPOvjDPNDp+ljn7M07b/Tdt+XP/AI9XkupXsRh4AXC889a+g/EOkf25pk1tqUUUttIP9Uc5U9iCOh9680074U+Hbq/kXUdT1BxActZllUOO3zgZI/I0AdB8GdQJ8BILtyQs8gh3HpHngD2zmuyVU1O5dC58leoB71z174euTbx/2GqWEUKBI0C/LtHQYq5psq2NuVt3keeNc3Eb/eJ/vL6igC7rFk9vbMbe4ZeOj81h6P4mitLI200qiZGO8Z75rSnmuNUh3Z8mEj779T9BXM2Fnp7+JZ7FrVLqJ4TK/mDJLKQM57dTQBW8Q65Jq0y2GnKZrmc7VRefxPtWxaW2nx2J077NFb3ttEAXRR+9AGC3uc9c+tdNZ6LaaRbs+nWFrbbh8wRPmI9261g38dve6lbtCPLk8zY4HUAjkfSgdzmtC8EWGu+L4NYWNLd9PlJmSNAEmcfdJHb1r1O7uBbW58wcY6iuN8PSnSNaudPgjeVpSZgRyewP9K6O6iuL2M+cwgQdc/eP0FNtsudWdRJTd7aI+Yvhz8PdN8ex6/eavqmrwy22oPFElncKoYHnkMrEnmn+J/g9p2iYlu77Vnti2CslwhYfX5MV2v7PEy22heKimA66s3JHJG2pfiNrcVxbSQPjJOCpoSu7FUKMq9WNKO8ml955knhDwStuI4NY1yNv+eX2qMDP/fusi78LaTDK3kXuqso6MZ1OfyWtmHRE1Gz80ahb2yktlCPmwKleysl/e2aTSWyEDczEAsemfXH9KqUHfli9T3aOU4aWLWGlW7p6W16LXT/PoYFh4S0y6kIuLvUov7p+0Lz/AOOcVNfeCNGihJN/qDAc4eZCP/Qa2Xt0W33lh1x97nNel/Djw14Q1W1t767t5Ly/h/1kNxMWjVh/Fs7+vORV1afJsaZ3kTy5KpTd4PTzT/4J5FpPwx0u8t0knur2KRuQBIi4HbOVqSXwTpdhcQmfUtWNr5i+cI7lQ2zPODtxnHtXrnxA8K6rY6pNrGlWkl7Y3f7xxbruaF+4KjnHcEVwUWlahq17CLy0uLWxMqiWWVCnGeQoPU1ifMHd3f7PHg+K0EsXiLxK4K5Gb2LGP+/VeQeP/AVp4Y0u5vdHv9SmS3mSNmuJ1I+bOBwo54r3nVpNXigjgS7tmEgVEAJ3AngDH/16wPj14ftfD/wQWCFjJM+oQtNK3WRiGyfp6CgDoNRu9V0qNbaSZANo2GdihK44OehrlVWW51I3l5MssMOWZhnbnsBnrzXqmvXdpNo6Bwjr5YI3AHtXiHivXY7TTtjNN5kpZIV6KecZH0qoxcnZF06cqk1CO7OF8SXcM2r3bxYClz07nvWdoF2+l61aauII5/sc6zRxTDKOVORkelOOnwGEvJK7Pn/Vnpj1zUs8sPkoIk2ALgjPevd9ndJS6I+4WHvFRqbJWPq/4b/Eay+IOkSyxxfZb+1IW5tt24DPRlPdTg/TFdnXyx+z5f3UPxVNtbZNvc2konGOAFwwP54H419T14tWKjNpHx2LpRpVXGOxkaw+JAP9nP6muS1OQYPvXSeIZvLuFH/TMH9TXF6pdA55rI5Tn7++aOQRcnccDFdQl0LaxhhyPkQDiuB1C6/0+M5+62TV641tcYD54oA2b/URyd1chqmoBy25gqLyxNQ6lrKJGXnlESDqSeTXD6trz6mskFmdkIPJPV6qMXJnbhMHUxU1GK07lPVtSe61CSTGUzhcelZzXRPSrMOnPOoYzjaeoA5q3b6WsLbiN7die1daiz9BoYOtyRhBWijPisbm5I+Xap9a2LTTVtl+U5Y9WqVWkj6KpqRL5AcTIU9+1axikezhsJh6L5pPXzGNbLg5FVpYCB8pIrUyrruQhge4qCRappHbVoRa0Oeu0ccHnJxWxbJ5KwqeoABqvdRgg021uC8Zjf78fQ+orNaM8mlGNGs79djvPD0SXNv/AKeyymJ8RsRyopNWmg093miiR3xtXI4+tc3ZanNbu/lHhjnFWLgXF5bi6mceWrYZB1H1rlnB83kz4XMMsrwx2itCcrJ9NXt5BaeHI9VnkPntFI3zZzwPwqiukRWF05ZzLIpwGPQfSriztHyjFfpUTvnJJ5rojhqMZc0Yq5+orDU4Sv22/wAyGU1SkhdiZIuSOo9asyvS2x4rZ6mU4qpLlZp+EfFr6RdiGVibeQ8j+6fWvUf7X8xEkRsOnbuRXjlxp8Nwd6jZKP4l712vhXV7bVbX7BqOYr2AYEgPLD1rkrU2vePguI8qnSf1pLfe35/5nodpriyIMNVo6sp/irk00SfzA1rqCKvcODmo7iDVLYnaqzKP4kPX8K5j4w6S51YYODXP6hqygH5qy5ZNRl48kjPqaqTWroN95KBj+EdaAILid7mQsfuivpqvl+SRpDiFMIOme9fUFABRRRQAUUUUAed/CL/WeO/+xwvv/QYq9Erzv4Rf6zx3/wBjhff+gxV6JQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzv8Vf+Tr/Av/XG0/8ASmavoivnf4q/8nX+Bf8Arjaf+lM1fRFABRRRQAV87/tT/wDIV8Df9drr/wBCt6+iK+d/2p/+Qr4G/wCu11/6Fb0AfRFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB5/8Sv8AkZvAH/YwL/6Jkr0CvP8A4lf8jN4A/wCxgX/0TJXoFABRRRQAVWmHn3SRBseX85/oKs1ga5dy6Rc/bdjNbSKFdgM7CPX0BoAt6teNaW7bsMuOSOtef6fqsFz41Vt2VEZGe3JHB/WneIvGkEtsyxSh2YYAU5rzYau0N0zh9kiOW49fWgD6Oe+hFt1HSuD1DUIx4r04IeXn2MAcZU9R+VcnpPjPU9Xt5orSEzSQAbwhycHocdcUlrbatdapHcFDFLuwss4KRp+fJNAHSeKNVnsH2WMwZXYKEc8gn0NZWja3aaRBNdXEhkvJpNsj44AVuFX271pjQrSyVtQ12+W8mQErGmVjj9+eSa5q11zRLW/nn1TRJL6JnzHtkChFx12HgmgD0a48ZWxsypkw2OVPBH4Vyum64h1i71C4by4UXbGG43Nzz+A/nXoWjJps2jw3UCxyCaMOJGALEEcc1HF4csrm6a9lhQuRtXKA4H40Ac/4KvkvNZvtQlBCbRFEzDG7nJI/St7XtSiW2YhgMDg5rN1iX+x8tIPOts/OvRlHqp7GuR8VpqIs5Z9Pma4tVTduIwSOuAehoA8y+FPia30PQPEImlVGm1J2+Y442isXX9XuvEGptLaD9wOA57+9cr4bs5L2K8bf8v2hsj1NdjZLBFAqNlVA/hHOa66NL7TPv8gyRNRxU29Vp+Rn2tnHCrASlpm+9uGPyqSyumgufs13K72xbd5Y6e5HvUWpSBV3KcMvIIqlGZb+5DRjBQcn39K1klse/iqdGVqHLdrb/P1W9zoL6K1mZzEZoo+Ni5B3DrzU3h3WrrQdZF7ZEFD8k0Z/iU9ce9Zl5eXp0y3tWhiUQg/vF6n61TsvOJZlfcCRk44/CohCKvpq9/MzpJ1qH1XEwfVO7u2uj/XyPf7TxlcnTlkiTzoiPlkicNkfzrkdc1y61C7DuGGw7goOTntmuIEbRWxaOVlBPO18HPriuhs7qFbMZ+bK9c859a56lPkPg84yaWVuN58ylfp27/eRWvjC/wBO122vL5DOsD7ih/mPcVc+Nfi5PEnwrie2WRoJLuNlkKEA43A8/WuX1mYSXEaRcsTjFYPirULyHwE2kzSMbf7YJkUHgHB4/n+dNrmjzJbGlXDxxuB+t0aSi4O0rbW3v66nrtzq9xLFDb304EZTJS3YO+P7pPQfWs3VtSj1DSP7H1CAQ6c0gKyKPMe2HdlB6nv271mfareF4pIHVopEG1x0NV9W1WNomZmXpjjisk2ndHz6bhK60aKl/wDCfxnFJGdL04avZzgNb3lnKpSVD0bBIK8etYt/8OPHFje/Y7jw5fmQnCmCEyqfoy5H619TfDKyu9P+HGkQagrJN5RfY4wUVmLKD+BFdXXT9aqPc9L+1K7+LU8j+B/wwvvBsF1rHiBBFqN5GIo7cMGMMecncR/ETjgdMV65RRXPKTk7s8+pUlUlzSOL8aXXk6hGucfuQf8Ax5q8+1TUQAxzXRfFDUBa+III88m0Ugf8DevL9V1QW9s9zcE7V6L6mpJjFyait2Vta1M26gq2JHPHsK564127ZSPNx9BWXe6q13cNK55PQegqmXlnOI1JrqjFJeZ91g8Jh8PQjCUFKe7dr6/8AkvLyW5kw8jOe2TmpbK1d7hVUcAfMabHbC3BaT5n/lWzp8ISIHu3JrWMbs9nB4TnqLm08ieC2SNcIuKn8sYp6ACpxA8i5jUsB6Cuix9XGEYKxRdKqzRgirziq0tS0YVoKxnLPJZy7kOUz8y1o+YssYdDlT0NZ9yMioLK6MTtC5+XqPas07Ox5UK/sZ+zls/wLdx0NZm/y7tT68VeuHFZcjFp1xzg1MmcWLnaSaNeOTaEPYjFdHoFvLfrPCFzCw+Y+9ccJtsOGyOe9dV4R12K2geKRgrZyD61FSXunmZziL4GUUr7fLzJtT0qTT03F94HXisd5fetrxBrMc8TKGDFq5V7j3opVJNajyPNcTiKU1iHe1rP+uxNLL70+0lyOvesyW596datKgMjDCk8A1pzanrRxX73Q31k4pkilpFmikMUycq69RVSO4BHWrMStLz2rTfQ9bnhXh7OSumdBpXjW7gxBfxFio/1qdD9a0z46hHHkzfXiuVWEAdKa0YxWP1eJ4MuFME25O+vmdS3i2C5bYsm1j0Dd6YGe6ffKTjsK5i2t1kvEDDgHNdXBgKK5akFB2R8NnOApYDEKlTbel9SRYfl6V9K18626iaVYy2Ae57V9FVmeKFFFFABRRRQB538Iv8AWeO/+xwvv/QYq9Erzv4Rf6zx3/2OF9/6DFXolABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/5Ov8AAv8A1xtP/Smavoivnf4q/wDJ1/gX/rjaf+lM1fRFABRRRQAV87/tT/8AIV8Df9drr/0K3r6Ir53/AGp/+Qr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHn/xK/5GbwB/2MC/+iZK9Arz/wCJX/IzeAP+xgX/ANEyV6BQAUUUUAFBAYEEAg9QaKKAPPviF8PptbjivvDaW8N7CpV4CBGs65z1HAYHPJ65rwnW9PvrLVZbHWIDbXkJCvGHBIyARyOOhFfW9eH/ABj8Daq2unxHo1pJeQTIouo4V3PGyjG7aOSCAOnTFa0uXm9493InhFjF9aScbO19r/8ADXPPNHuJNAvFurd2iMi4356itXVvF95Pb5kuWwOhFYOlj7XeqtxGwRRnDqQCfSt3UY7f7PhcEbeRjpTrcvN7prxEsKsb/s21ldLZeny3Ohhtr/UvhmNeuVuA8cgEil8qYwcFyuMjsa5y61GM24HyjHfua2/BfjO/0rQp7KW0F3blmEfmNwynqD6io9E8OeHPFXiI2ckcumTOC8VqJyYpiOSoPVeO1ZO19DwqygqjVPbp/wAHz7+Z1Hwj1qabR5Rf7vscM5jtWb7r8AsM98E/rXfP4jt7V5EkmRBnILMBkGuF8ReKbXT9HTw7oOmQQw2i4YspHksOygfxepPrXEyyG+jM15N5smP4z+gpGR3vizxAl1EYLdhJJL8qAHqTUPiXX7ez8PQ2SyKxigWPjuQMZryS+nkt7yNYJnQM2MK2MfSq18Lgcyyy7iMjcxOa3p0udXufS5VkLzCl7dVEop2a6/L5HPN4Q0h7h3kNxy5JAkA/pVG+8KadEp8h5x6ZcHH6VtRFp3IaXy3J2g44NQXNu8Tb5pwSpyFArocI22Pr62XYKVK8aC9dDlLjw1e2WJZLc3EAPIRsPj6c/wBat6Tp/h2/JjnnngmzgRyOF/XGDW+jXV6xRCq4AJOc4zTz4esLxDFeRbmHSVflf86hUlf3V9551LJYKpz4aClHtPb5Nar53RWm8GaQZMWxucHpukB/pTp/AenJEG33K5HDbhz+lRXOlavoYB0q9F/bAZ8mYYZPYf8A1iPpT4PGEM6CC/V7WdONsn3fz7fjVWprSSsdcaeVQfs8Vh1Tl5rR+klp+vkZU3gwbS1tejg4KyL/AFFUv+ET1HbuhWC4GcYWQgn88V0tstzqd0y2Q3gjqD1+lXreSS0Yi4j8oD5Rkg1PsoPocyyXL8Q7qDjHum7fjdHGHR2sX2apot8DjO6KTjH/AHyf509l8LuxG3ULYjqJiDz+ArtpZvtsqqJCHA4YnOBUd1pKTQr9ouFkznK+X0/E0eytsU8hUF+4tJL+aKf4pxZBo+o2l7CYm1K2jhSTIjeXY7jqQAf519HeAfCPgTW9NtvEelaOplDEeXPM0wgkXqACSD6g47ivmi20LS7q7ezGmG4kcfulhDb92ewXrXsPhX9nER+HYL1vEGs+H9YmBdktJhtiB+6CBg7sdfmrmmmpanxGZUp0sVONS199Nj6Aorxf/hXXxg0IH/hHPiamoqOiatASSPQlhIfxzR/wkvx10H/kJeENI12BTzLZTBHb8N+f/HKg889oorxf/hoSfSQP+Ez+H3iHRgPvOsXmL9cuE4rd0r9oP4c6ptDa21jIf4Lu2kTH/AgCv60AM+JmjSalr9vLGoUJaqDIf99+P1rxrXZbix1P7FJGPLI4JGdxr3rVtb0nxDCt5oepWuo2wTYZbWZZFDDkqSDwcEcdeRXkvjixWeHzVH7yJty1cHaSZ35dWdHGUqi6Nf5HDmG2cE+Sue/FRmAMcR8D2FKgZ5eRj1q7HGABXopXP2SFNVelip9iQKSRkgd6ktz8oq4U46VSYGKYqenUU7WNXTjSalEuK2BWhY6u1jBIgRW3jGSOlZCyUGSmaycJxtIklfcxJ7nNVZGpzyVWlkqWznrVFYr3Ddazd2Lse9W55ODVKFTLOSOwrGT1PnMTPmmkiU3GUIJ5FSaXCZblpmHyoOp9arR20txOUQd+T2Fb9parFCsa9B+p9aIptl4OjOvVUpLRFeWz+0cY4oTTRHHtBNayxACgpW3Ij3HgKb1ktTCmsJBnbKT/AL1VxaMTiWbHsBW9IlUbiIEdKhwSPOq4CnTXur5FRIYIeQNzDu1SwMJbkKeg5qpsledYVP3j1NbFrpywrnlmPU0oq5GGpzqytCNktyGWzVZ1aKQbGPK+lacKgKMVXmjEe0gY5qeN+K0Ssz2aFONObsrFyKJpnEca7mPQVHcQvBIY5VKsOoNLBdPbSrLEcMvQ0y7u5LqYyzHLGrOptp+Q22YJdKTW/DLwK5ZpNrAjsc1q2t4HQYrgrr3rn5bxXRlHGRrdJL8UbglxyCc19L18ri44619UVznyIUUUUAFFFFAHnfwi/wBZ47/7HC+/9Bir0SvO/hF/rPHf/Y4X3/oMVeiUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wAC/wDXG0/9KZq+iK+d/ir/AMnX+Bf+uNp/6UzV9EUAFFFFABXzv+1P/wAhXwN/12uv/Qrevoivnf8Aan/5Cvgb/rtdf+hW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAef/Er/kZvAH/YwL/6Jkr0CvP/AIlf8jN4A/7GBf8A0TJXoFABRRRQAUUUUAQXl5b6fZS3d5KsUEKlndugFeXeK/ieuoWL6fo8FxbCZtpumYKSncADkZ/lmul+Kqz/APCDSyw58uGeN5v9zOP5kV4XqWoxSWo+ZVK8hhQBY1IRG3BifDgZzjof61jRQ6lqkR8vZHHnHmMf5CpE0jxPqtlDPZaVcS284zHMi/KRnGfb8a0Y/tHh4JY34WO4jQZ2sGByPXvQAWd4tlB5Dbd0Y2HI/Wk0VrvUvHGkW+kqXuhdJIMH7qqcsSewwDWeix614gtbYBihkBnePOViB+Ykjpx/OvYfC+laL4Xjlu9Ls5LZrpdheclnK9Ry3ODz+VAHFeM0vNC8T3Z1FWjW7dpoZf4ZFbrg+o6VxbayyOYY8y46beeK+l9QvdJ1TSfJ1C2t7uEjOyZA4/I1414u8PR6peMPCtlbxNZxs7wwgIHXPQepqopN2Z1YOnRq14wry5Yvdnnj3DSz+fOdu37q55+tR3OoB1AB3MeAKq3E4mcxmIiTvkYx9aaDawjaB+8A++Dya7YpRVkfpuEp08FR9hh37vVvdsJyUjjjQ5kLA8epNWbyNGB87lu3pUenoI73zpRn5SED8496bqc4cbVPJOBVdLnRdKjKcvkvQv6KsCWysc/Ocue9SXKuzmS2GQvGD3H+NZkMrJYxtEVG3KkdN2D1qxHfL9nAOQTzVJq1jrpV4eyjTlpoV7i7nIKCGQN7rimWOkWWoxtJqkfmLuIwpwR+Nbnh63TUrqR7sZjixhT0Y1s6vaW6QkIE4HBSueVRKVnqfJ47OcPRxXs6sHUit72/Lqeef2NqGk3BPh+9cKTnyJDlT/T/AD1qe28Rxx3P2fxJaNZyHjzCpaP69z+WazNcutT0gW80Wo7xOzqFKoWG3byfl77v0NO00apr2kyzT6koQSFDE9qjg4AOf1oUtbQHRxsVWdHAqalvyvlcLWvs5K2/Rr0Ogf7IqfadPdHB6MjZBp5XUJoUaOzlZZFyrAcEfWuG1G0vfDl+oin2h1yrx8B/XIP8q39I8Vajb6ajX9u4sXk2C8VCUQnqDj88dfaplVa0tY58Xn9alemqfJNbrp6r+vme5fs9Wy2+u60tzCrXZt42EoOfLXccr+Jwfwr3qvOPg1Y+Gbbw5LceHdcttau7oq93PEdrJj7qeWfmQDJ+8ATnNej1zt3d2fG1q069R1JvVhRRRSMgrC1XwR4W1wN/a/h3TLxm/jltEL/g2Mj863aKAPN73wvofgyJ7Hw3YJYW05+0PEjswMh+Un5iccIvA44rgPEcgcMM9Qa9G+IV0IdTiUnH+jg/+PNXkes3nmB2B9hTirtI6MLSlWrwpw3bSOWQYkNWkqqTtmIqZW4r1UfutNpXRq6dp51FmRXVNo7msu+t8O8eQWQ8EU+O4eIkxuVJ44qF3JySeaGEk7u70ZQExUkNwRTjN70l1GJBlThhWa85jba/BFZt2PIq1nRdnsXnm96rSzVVe6HrUIaW4bbCjMT6CocjzquL5tI6sJ5txwOc1etbYw2u5xh35+lS2el+QRLc4Z+y+lTTtxQo9WaUcNKN6tXfoitZ7vtpVT8rD5hW5EoArGsSPtTfStmNuK0gerlyShfzLKAHjvRPBJEAXQgHoSOtNSUo4YdjmrWpas9/HGrKqiMY4rQ9KTkpKy0Mx6qTdKsu1VyvmSBB3qGclbXRFaC0MsvmngKeKvFJP77D6GrSRBVAA4oZaajYqnhVTjZFGQylCpbI96I5ccHrViRRVOVcHIpPQyqKUHe5Z83imNLUflOyBomDA9vSoJFmGcoeKTZM6s0tUOll461BFqJt5AOSGOMUx4p26Lj6mlsLQG+RpjuK8he1ZyXNoeJjMP8AXkqM1o393mav9oFflfKn0NfY9fH0iRyf6xFb619g1y1afJ1Pic6yZ5XOPv8AMpXt30tv94UUUVkeAFFFFAHnfwi/1njv/scL7/0GKvRK87+EX+s8d/8AY4X3/oMVeiUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wL/1xtP/AEpmr6Ir53+Kv/J1/gX/AK42n/pTNX0RQAUUUUAFfO/7U/8AyFfA3/Xa6/8AQrevoivnf9qf/kK+Bv8Artdf+hW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAef/Er/AJGbwB/2MC/+iZK9Arz/AOJX/IzeAP8AsYF/9EyV6BQAUUUUAFFFFAHEfFlZ/wDhC/MiBaGK4RrgD+5yMn2yRXgd/Dpe3zRCoIGcbjj8s4r6wlijnheKeNZI3BVkcZDA9QRXH/8ACpPBn9o/bDpAYhtwhaVzFn/czj8OlAHIfDjw1NqHg6zlv726s1uWkeFUI/1W7CnBHAPOK6jUvAnhtNO8q5t2vMEtm4k3Et68Yrd1C136hHbQqYSIh5BQYVQOCMenTis7xBpV1HYMVu2JxzxQBheB7TSdJg1n7BaRQOkiLuVR9w84z+ddZq8WmarpwS7XOBwytg/TNct4V0W31PSLqGxvJIL6OUmWVl3CUN0yvtgjj0pt/ZHRm+zatqM6AD5XCgLIPYn+VAHK+IzbabObbS7y5kBGShkzs9jxXMJrF3ZebHbSQwyyDA87OG/Ed/rU+oX8VvrV15bb0MhaNn53L2+tc1qV0Lm6ihhOXZxgjtz1qZJtWTsVFqLu1cguvD+ptB5yRxXAb5mVTjZk+/asUJPayPC9o4nb+BUycfhXVtrawo0TttYcEZ61V06d7jWBOxaNApVc8bs9cVuqslue/R4gxULc1pWKejeHVu5G/ti8m095CBARGHA/3+eM8Y/Wi/8ABd1FdTfYpXv0gAMhRNrAH8a6m9Nv9mAXqRyCOlavgm6hbw/eqyDelwQZD1YbRgfhXgZtjMVgl9apyvG6Ti7W169/xKwua16tW0/XyPJ7p4IcxGF0ZeNjZBH4U2yEzqw3+UOqqRXsOqado+taRHLqFqjzICgmUlXwCccjr+Nc7a+AtIvtNS+k1d7RmnaNIS6ZKggAgnn19a6cszynj24uDjJb9vv/AOAe482owanV09P+AcvpTvaF3imMrv1B4FXWe+1SYW8aeWD952PAFddffCG4tbae50LWYr116WsqbGf6NnB9uBmuV06DVbHUp0utOnj8pCZMj7oHf3/CvZnySXMtzz8xWV4yhLE0pWqLpffzaeu3Y5j4g+HLmFNNGn2txc4WTzDHGWx93GcDjPNZ/hcS2+myQXETxfvyxDqQcbV9fpXeajrsbQZMgbAwOawo4tQltDcpDiJm+QscEj157VNNrmu2ceS4iEcS6+Inaytfp2t92xzPiaC71W9tYbC0nuZAGwsMTOT07Aexr0n4Z+HruLwfHY30bm4vrxvJspI+QWCr8wP+7nHvUGj6Bc2TnUryVRLgrGsZzsyOTn1xXsvwdudLv7W7H2KNNVsmCyT5LF0bOCM/d6EHH9aio05XRwZtXp18ZOrTd0/+GOV+JPwp0PRJLXU/BFh4h07xFKwEJ8O2ryRbuAWfkLEOn8S9zg81V0r4ofFPwERbfEnwfe6pYRRh31C1iBeJOmWkjzETx0JU+p5r6EoqDyzk/B3xO8J+Ool/sDVYnucZazm/dzr6/Iev1XI966yvOvGHwP8AB/i2VryO0bRtUzuW+03ETbvVl+63PfAPvXJ7/jD8Lv8AWKnj7QYu4yLtFH5tn/v5+FAHuNFee+Dfjb4O8YSLaLenStTztax1HET7vRW+63PYHPsK9CoA8g+Lt2YfEVvHnav2NST/AMDevJ7y6858D7o6V3Px4uZYvG9mqcp/ZyEj38ySvMVu0k4Pyt6GuyjBL3mfovDmXUKEI4uo7zktOy/4I6dsOG/A0qy8dagmfKkZqstxtODwa3bPqJ11CZpGSo2k4qr9oHrUbz+9HMKWJViaWWqE8inlgDRLPx1qqjGa5RR0zk1m2eTiMRzPlXU17Wwh2iSZBkjhaujZGMRqFx6Cq4mGODxTWm960VketB06UbRRJJJxVGeTrSyTe9UJ5+1RKRwYnEKxYspgLsg9xW1HJxWPpNn5yyTP6YX61ethI5KsCuDg04XsbYGVSME2t9i/5lNLk9qekQAp5QVqe0oya1KpPPzAge1TwiIH93jP60joKqyKVbcvBFLYxbdN3aua0YywXHXirF9ps1lEkkoAV+lZsFx5iBhww6j0qxc301wqrK5YL0zVXN3JtqUXoVpD1qnN0qw7VUmbioZy15aDLefy5DGejdKneX3rJuJNp3A9Kl+1bow2eoqFLoeZDFct4PoWJZabZMTcluwHWqZkMrhE5Jq/EBBCFGM9zQndk06jqVOboi8ZPevsevjO2G/536dhX2ZWGI6HzXFrclQb/vf+2hRRRXKfChRRRQB538Iv9Z47/wCxwvv/AEGKvRK87+EX+s8d/wDY4X3/AKDFXolABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/wCTr/Av/XG0/wDSmavoivnf4q/8nX+Bf+uNp/6UzV9EUAFFFFABXzv+1P8A8hXwN/12uv8A0K3r6Ir53/an/wCQr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHnvxV/cX/ga9JwkPie2R2zjAkWRc/TJH516FXGfFrRpta+GOrJZEre2aC+tWXqssJEgx7naR+NdB4a1uHxJ4W0zWrbHl39rHOAP4SyglfwOR+FAGnRRRQAUUUUAFFFFAGbrLTW8Md5BG0hgJ3qoydp6kD2wK4nX/G1vNamOB/MkYYCLySfpXpFc7qfgrTNSumuFae0kc5c2zBd59TkH9KAOP8Ahit2niO8EmQr2+6ZOyHd8oPvy1enywxTpsnjSRP7rqCP1qlo2h2Gg2ht9Oi2Bjud2O5pD6k960KAMXXPCGheI7SO31fTopUi/wBWVyjR/Qrgj6V53rPwItzdNN4Y1P7CrjDRXKGXH0bOcexzXr1FAHzLL4Y/4RzVGtNThk+2KMySSjhueCnbbx1qhrRiRGaJuV5VsYr3fx/4FPi23huLC4W21G2BVHkBKSKedrY5HPINeV3fwZ8a3UEhd9PG08RrcHMn0O3A/Gs6cOSNr38xyd3c4S1nudUvrez89IlnkSMykE7dxx0rr10i90gW+j2ckLCaYIJQTl2Y43Ef56Vf0X4MeKrW+guJ47KLym3KpuNwDerEDnHtXW2fgC4sNUS617U1+1BS1qLdcojjud3X6cda8bMMNiMViIU0v3a1fr/wx10KkKcG38Rzes+CdQ0fTJftOr2zwQqWHkq25+/fpXOEWsdiFUcgY+or0v8A4RifXZTDd64oA6osHp65Ncj4u+HSaPGZNP10tDjlJofmB9iD0+tenh8JRw1/ZRtc551Zz+JmLol9eNpOpeUbk29o8YEqH5Ig5I2k9unH1pt3PEkAaNir9zmmadqH/CKFoEka6sLlgl/bufluUPB47EDoRyK9BuvgSlxdnyPEc8dk3IRrcNIB6bsgH64rqMzyGzOnS3Jmu7aBnVyFYrgN7kdCat6rqH2oiGBfMllISOONeWJ4AAFe2+F/gx4f8PztcXzvrEu0ogukXy0B6kIO/ua6XSfAvhjQ777bpWi2tvc9pQpZl+mScfhQU5SluzwO98N+K9K8qwudBvpZgoUNBEZVbjsy8V6x8JPBd/4Y0y8vtbQRX+ospMAOfJjXOASO5JJP4V6HRQSFFFFABRRRQByHjL4XeEvHUbHXtKjN0Rhb23/dzr/wIfe+jZHtXnv/AAhvxU+GPz+BtaXxVosfP9lal/rUX0Qk9h/dYc/wmvcaKAPmTxNrGtfE3V1vofCeq2N5YW62l5Zm3eRonDM2eFBwQ3GQOhrnrjwP4ob7vhvV8+1jL/8AE19eUVuqzStY+mocQVKOHjQVNNRVr6nx2PBvi4go3hnWSR0P9ny//E1dHgDxLPbr5mgaqjAf8+MmR/47X1vRQqz7GkOJKkb81NP5s+PpPh54uQHy9B1OT0/0KQf0qs3gPxmX2f8ACM6rn1+xSY/PbX2VRSdXyMp5/KW1O3zZ8aj4ceL3/wBb4f1RfYWUh/8AZam/4QDxNCn7rw1rBf8AvfYJf/ia+xKKPa+Qo5847Ulf1Z8bjwf4uBwfC2t/+C6X/wCJpG8H+Lv+hW1v/wAF03/xNfZNFHtWL/WCv/Kj4vfwZ4wbp4V1v/wXTf8AxNTW3w68VyfPceHNWUf3TYy5P/jtfZVFHtX1REc8nzc04J/NnyHB4J8VWrsU8Oauyn+EWMv/AMTV+Hwd4m6t4d1UE/8ATjJ/8TX1bRVKu10PQpcVVqWipq3qz5cXwj4jHXw/qv8A4BSf/E09vB/iNlyNB1Pj/pzk/wAK+oKKr6y+x0/65Yj/AJ9L72fK7eEPEh/5l7Vf/AKT/wCJqvJ4O8THp4c1Y/8AbjL/APE19YUUvrD7Gc+L68v+XS+9nyPH4R8VQzZ/4RrWCp4OLCX/AOJq0fCPif8A6FzVv/AGX/4mvq6ihYhroZw4rxEFb2a+9nyY/hDxQf8AmW9X/wDAGX/4mqkvg3xURx4Z1g/9uEv/AMTX19RSdd9iJ8UV5/8ALtfez4xuPBPi5uF8La0fpp8v/wATUa+BfGJAX/hF9aA9f7Pl/wDia+0qKj2rPPlndSUruKPjmDwR4pgHHhfWSx6n+z5f/iaV/B3i09PC+tf+C+X/AOJr7Fop+2fY3XENZR5VBHx5Lpep6X5cep6beWbtgKtxA0ZJ9ACBX2HRRUzqOdrnFmWa1MwUFOKXLf8AG3+QUUUVmeOFFFYvjLX08LeCtX1uQj/QrV5UB/ifGEX8WIH40Acr8H8S2Xi68QfurzxVfzRN/eXKqD/47Xolcl8LdAk8N/DLRbC4z9qaD7RcluvmykyPn3BYj8K62gAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8nX+Bf+uNp/6UzV9EV87/ABV/5Ov8C/8AXG0/9KZq+iKACiiigAr53/an/wCQr4G/67XX/oVvX0RXzv8AtT/8hXwN/wBdrr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigBCAylWAIIwQe9eYeBZP+EB8dah8PbxymnXRfUfD7u3BjY5ltx7o2SB1wSa9Qrm/HHg218aaELWSVrO/tnFxp9/FxJaTryrqfT1Hce+CADpKK4PwZ49uLnUR4V8b240vxVAmdh4h1BR/y1gboQcZK9Rz6HHeUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFVdR0221S1MF2pK5yrKcMp9Qe1WqKAOUi8BW9vLLNBql8szjCszKQPqMDNYevfDPUtT09zDroe7BzGskO2Mj0OCTn3/AEr0eigDxXw78GdWl1yC78VXVqtnbyCT7NbsXaYg5AJIGF9e9e1UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV5d4vcfEP4jWPgi2Pm6NpLJqOvMhyrsDmG2P1I3EenuK0vGPju7bU5PCPgGIaj4mlXEso5g0xD/wAtZm6AjsvUnHHQHe8FeD7LwV4fXT7R2uLiRzNeXsvMl3O3LSMfUn8hQB0NFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/wCTr/Av/XG0/wDSmavoivnf4q/8nX+Bf+uNp/6UzV9EUAFFFFABXzv+1P8A8hXwN/12uv8A0K3r6Ir53/an/wCQr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBheLPBujeNNLWy1y3ZvKcSQXELbJrdx0aNxyp/T1rj4x8SfAb+UsY8eaKv3HaVYNQgHoxPyy/X7x9q9NooA89tfjZ4RE4tfED3/hq9P8Ay66zZvA3ud2CuPxrbj+JfgaWMOvjHQQD/e1KFT+RbNdFcW0F3CYrqGOeM9UkQMD+BrJPgvwsSSfDWjknv9gi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVR/wALH8D/APQ5eH//AAaQf/FVZ/4Qrwr/ANC1o/8A4ARf/E0f8IV4V/6FrR//AAAi/wDiaAK3/Cx/A/8A0OXh/wD8GkH/AMVTJPiX4GijLt4x0Egf3dShY/kGzVz/AIQrwr/0LWj/APgBF/8AE0q+DPC6sGXw3pAIOQRYRcf+O0AcvcfGrwrJMbbw2NQ8TXv/AD76PZPMR6ZYgKB754qnJb/Ejx6/lXePAeiH76wSrPqE49nHyxD6fMPevSoLeG2iEVtEkMa9EjUKB+AqSgDE8K+EdG8G6T/Z+g2vkxs2+WV23yzueru55Yn/APVituiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/ACdf4F/642n/AKUzV9EV87/FX/k6/wAC/wDXG0/9KZq+iKACiiigAr53/an/AOQr4G/67XX/AKFb19EV87/tT/8AIV8Df9drr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKptrGmrrK6Q2oWo1J4vOWzMy+aY843BM5xkHn2oAuUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB87/FX/k6/wAC/wDXG0/9KZq+iK+d/ir/AMnX+Bf+uNp/6UzV9EUAFFFFABXzv+1P/wAhXwN/12uv/Qrevoivnf8Aan/5Cvgb/rtdf+hW9AH0RRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHzP8GvCOofE3w1f65rfjbxTbXUWoPCBZ6iUUjYjZwQcHLnpgdK6bxLH43+CscGv2fiK98U+Fo5VS/tNUYPPAHYAOsnU8nA6AEjIIOQv7Kf8AyTXVf+wu/wD6Jir0b4pKjfCXxSJQCv8AZVwRn1EZx+uKAOg0nVLXW9Gs9U06TzLS9gSeFyMEowBGR2OD0rK1Xx74S0O+NlrHiTTLO6X70M10iuv1Gcj8a8r0bxNqvhH9jy21eMuuoR2rRW7kgGMSXLRxuP8AdRlI+grqfh78JvC1h4G01tb0Kw1bVLu3W5vLu/t0uJHlkG5vmYHgE4H05yeaAPQ7DULLVbKO80y7gvLaUZSaCQOjD2I4NM1PVtO0Swe91i+trC1QgNPcyrGgJ6DJIGTXjvhW0t/h3+0he+E9Hf7Pouvad9uisi3yxTDOdgzxwj8emOyin3VhH8Rv2kNS0rxFCt1ofhaxjaKxkOY5Z5FRt7L0PDsOf7q+pFAHqWieLvDviV5E8P63Yai8QBkS2uFdlB6EgHIFedXv/J3Wnf8AYtH/ANGSV2sPw28I2ev2GtaZoNlpt9YOzRSWMKwA7kZCGCgBhhj1rzHx14kt/CP7SC67dwT3EVl4VeQxW8ZZm/eSYHA4Hqx4A5NAHs2pa/o+j3VpbatqlnZT3r7LaK4nVGmbIGFBPzHLAceorl/GfhrQtZ8ceFL7VvEMumajp9w0thZx3KR/bGBUsu0jLDAwcdmI71hfDLw43im7i+Jniu4hv9Uv492nW8TbodMhPREz/H/ePY5HrVf4of8AJcPhf/183X8o6APUbnV9OstRtLC8v7aC8vdwtreSUK8+0ZbYpOWwDzist/HvhKPWP7Kk8S6Ut/vEf2c3iB954C4z97PGOteY/G/S5db+KPw70yC/uNOa7ku4mubY4kRCIw209iVyM9s961PiN8KPBVj8ItZGneH7K0n07T5J4LqKICcNGu4bpPvNnbg5J6mgD1quau/iN4LsL6SzvfFWkQXEZw8cl4gKn0PPB9q4O+1rXJf2Sl1PTppZNT/saLfNvPmbAVWV92c7hGHbOc5Herei+Gvhz/wo6DzbXS30t9MWW4unRFkZ/LG5y3BEmfxB4oA9B0XxRoPiMzDQNZsNTMG3zRaXCy+XnOM7ScZwfyrVryz9njS4bX4R6dftptva3t4HEs6W6xyXEayOIy5ABbg8E9Rz3zXqdABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/5Ov8AAv8A1xtP/Smavoivnf4q/wDJ1/gX/rjaf+lM1fRFABRRRQAV87/tT/8AIV8Df9drr/0K3r6Ir53/AGp/+Qr4G/67XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB8vfAr4o+GPh14Q1PSPFlzcWd+2pPL5AtXYgeWi4OBwcqRg+ldj4u8Q+IvjHZf8Ip4K0LUdN0e5kX+0Nb1KEwJ5QOcRrnLZI5HU9CADmvcKKAOH8afD6LV/g1deDNFHlCKzjiswzYy0RVkBJ9SgBPuTXHeEfjnoXh/wxZ6L8RBe6Jr2mxLbTwz2kjmbYNokUqp4bHfHOcZGDXtNNaKNyC6KxHQkZoA8i8B2mp+Nfi9qHxGvLGfT9Hishp+kRXMZSWdM5MpGeBndj13j0JOf4nu7j4UfHC78a6hbXdx4Z1+1jgvJ7ePf9kmXYqlh2Hyj67jjJAFe30EAggjIPUGgDgNC+MGieLvEVnpfg60v9XikZvtd8ts8UFmoUkFmcDknAAx361z+oxJN+1rYxyoHjfwywZWGQwMkgII9K9eVFQYRQo9AMUtAHhtpNJ8B/iCun3cznwF4gmZraRwSNMuD/AW6BT79hn+FidX4ngn44fC/j/l4u/8A0GOvXaKAPIvieCfjh8L+P+Xi7/8AQY67X4nf8ko8Vf8AYIuf/RTV1FFAHh13Dez/ALHMEen3kVnKdKjLvNIIw8e8F03HoWXKj1zjvXO2lt4Lli03Wbv4G+Inkmt0k/0K3ea3PHXYrBT/AMCUZ64r1T4z+B9S+IHw9fSdEuVhvI7mO5RJJCiT7QQUYgHj5sjtlV6darwePPHi6cEn+Ft8L8KRsj1K38kt2+fOQPwOPegDofA3jbQ/GujPP4fEsAsnFvPZTw+VJauB9xl6DHTjI4I7Gumrz74X+CNU8O3PiDX/ABM1uNa8RXf2ieC2OY7dAWKIDxkjecn6cnqfQaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/AMnX+Bf+uNp/6UzV9EV87/FX/k6/wL/1xtP/AEpmr6IoAKKKKACvnf8Aan/5Cvgb/rtdf+hW9fRFfO/7U/8AyFfA3/Xa6/8AQregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/ydf4F/642n/pTNX0RXzv8AFX/k6/wL/wBcbT/0pmr6IoAKKKKACvnf9qf/AJCvgb/rtdf+hW9fRFfO/wC1P/yFfA3/AF2uv/QregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/AMnX+Bf+uNp/6UzV9EV87/FX/k6/wL/1xtP/AEpmr6IoAKKKKACvnf8Aan/5Cvgb/rtdf+hW9fRFfO/7U/8AyFfA3/Xa6/8AQregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA+d/ir/ydf4F/642n/pTNX0RXzv8AFX/k6/wL/wBcbT/0pmr6IoAKKKKACvnf9qf/AJCvgb/rtdf+hW9fRFfO/wC1P/yFfA3/AF2uv/QregD6IooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACisrXvC2heJ4Ei8Q6RZaksYYRG5gWRot2N2wkZXOB0x0FfBngrT7XV/H/AIf03UIvOtLzU7aCePcV3o8qqwyCCMgnkHNAH6EUV5Nrf7PnhEaRdP4Phu9B1YRN9nura+mPzdQrBmOVJAzjBryz4D/GLXIPGNn4Z8S6nLqGnai3kwSXcm+SCY/cAdjkhj8u055IxjnIB9WUUUUAFFFFABRRRQAUUUUAFFFFABRRWL4u8U6d4M8L3muavJtgtUyEBAaV/wCFFz1JPH69BQBtUVleFdWl17wdo2sXEaRS6hYQXTomdqtJGrEDPbJrVoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD53+Kv/ACdf4F/642n/AKUzV9EV87/FX/k6/wAC/wDXG0/9KZq+iKACiiigAr53/an/AOQr4G/67XX/AKFb19EV87/tT/8AIV8Df9drr/0K3oA+iKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAr8/PANzBZ/Enw1dXk0cFvBq1rJLLK4VI1EyksxPAAAySa/QOvz88A20F58SfDVreQxz28+rWscsUqBkkUzKCrA8EEHBBoA+x/E/xk8E+H9Bur2LxHpuoXEcZMNtY3STvK+PlXCk457ngV89/Ab4W6xrvjTTfEeo2M1to2nSrdJNMpT7RIvzRhOm4btpJ6YGO9Xv2iPhTD4X1CPxP4cs44NIu2EVxbwR7UtpccEAcBW/IEe4r0b9nX4nnxR4f/wCEY1mdn1bS4wYZZGBNxBnA+rJwD7bTyc0Aeo+MPFmm+CfC13russwtrZRhEGXlcnCoo9SePQdTgA15n4d1z4nfErwlJ4q0LWdN8OwGSQ2OmixFwbkIcbZJXPygsCMqueM8VhftaXtxHoPhuwRCbae5mmkfsHRVCj8RI/5U74N+GvHOqfCnSbnw/wDEcaNYbp1Sw/sSC48kiZ8/vGOTk5bnpux2oA6z4LfGQfEqC60/VraK01qzQSMsJ+S4jzguoJyCDgEc9V55wF+K/wAXJ/COsaf4V8K28N34k1J41T7QD5VuHbapbBGWJ6DPHU9gc/4a/AGb4deNIdei8Ui+RYpIZbY6f5XmKw/veacYYKeh6Y9x4prupTT/ALVf2zUEMRh8TwId3H7uKdEU89iiA/jQB6b48+IPxE+D3iDTW1zWrLxRY6mjyGFrBbURFCNyIyZP8a4LZ7cevr2ueO9I8PfD8eLtR81LFraOeOIoRK5kAKIFPRjkDnpznABrbv8AS9P1WJItUsba9jjcSIlzCsgVgchgGBwQe9eCftZ3s8WgeG7COM/ZprmaZ2A4VkVVUfiJH/KgDe8O658TviV4Sk8VaFrOm+HYDJIbHTRYi4NyEONskrn5QWBGVXPGeK0fgt8ZB8SoLrT9WtorTWrNBIywn5LiPOC6gnIIOARz1XnnA5P4N+GvHOqfCnSbnw/8Rxo1hunVLD+xILjySJnz+8Y5OTluem7Hat34a/AGb4deNIdei8Ui+RYpIZbY6f5XmKw/veacYYKeh6Y9wAcj4/8Aj7458CeOtR8OPb+Hr42Rj/0gWU8e8PGrj5fPOPvY6npXZeOdX+Ls2lS+IfCMem6XpFrGbhbORfNvbiJQSS6shVcjnYpDDpnPFeDftB/8l28Q/wDbt/6TRV9sSRrLE0bjKuCpHsaAPLvgv8YV+JVjc2WqQwWmt2Sh5I4Sds8XA8xQeRgkAjJxkc84Hh/7ReqeL5PGlrpviyeyjtxbC6tLLTpnkiiVmZMszIpZ/kPOMYPGMmqf7OF1dW/xr06O2QtFcW9xHcEKTtjEZcH2+dUGT6471s/tV/8AJU9N/wCwLF/6PnoA9k+FP/Cxf+Ee8L/2p/wi/wDwjP8AZkG37P8AaPtvleQPKzu+TdnZu7fex2qL4sfF648Iazp/hbwtbQXfiLUWjCm4yYrcO21cgYyxOcDPHU54B7P4cf8AJLPCn/YFs/8A0QlfKusXU+oftYo99EyOPFNvCFbIyiToiHnsVVT9DQB7R4w1X4m/DDw83ifUPEmneJrNJ4xd2EmmrbeQjNj926Nk8kLlgexwec+keCvF+neOvCdpr2klhDOCHif70Ug4ZD7g9+4we9cp+0H/AMkJ8Q/9u3/pTFXBfsl3d0+h+JbN0ItIriCWJ9p5kdXDjPQ4CR8e/uKAKXxA+PvjnwJ461Hw41v4evjZGP8A0gWU8e8PGrj5fPOPvY6npW18YPin490DUrA+D9JeDRp5AsOoNAs/258j5QBnapzgZAZskqe9ePftB/8AJdvEP/bt/wCk0Vfa8MSQQJFGMJGoVR6ADAoAraPc3l5oVhc6na/Y72a2jkuLbdu8mQqCyZ74JI/CrlFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAfO/xV/5Ov8C/9cbT/wBKZq+iK+d/ir/ydf4F/wCuNp/6UzV9EUAFFFFABXzv+1P/AMhXwN/12uv/AEK3r6Ir53/an/5Cvgb/AK7XX/oVvQB9EUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAGVr3iG28PwI9xaaldvIGMcen6dPdElccHy0YL1GNxGecdDXxd4P8ABfjDRfHGhare+DvEQtrHUbe5lK6VOTsSRWbA2+gNfc1FAGFKuk+PPC17Y3dneGwula3mivbKa1fkA5CyqrcZBDAYyODkV8j3Pw7+IHw1+JH2jw5omp6g+m3HmWl7aWMk0UyHpkqMcqSGXPcj3r7VooA8e8daNP8AGz4WiK10vUtG13T3S6S01KyltwZNrK0Yd1AYEZ5UnBC5xmvLfhV8S9W+EFxc+GfGHh/VGtLmctBCkOJo5eFYIrYDhuOh68jOa+s6KAOQ8LeJNb8X3seopo11oWhJGwEeqRBLu7kPAIQE+Wi9cnJYkYwBz4f+0P8ACXVpPE0vi/w7ZTX1teKv22GBC7wyABd+0clSAOg4IOetfT9FAHinw9+Ot74l0m20xvCWs6hr8a+RLLbRKLYyAfekkZh5eepyDjPGeK6D4sfDvUviD8MorF54W12ycXURQbI5JMENHznCkMcH1Vc969LooA+TPhV8S9W+EFxc+GfGHh/VGtLmctBCkOJo5eFYIrYDhuOh68jOa+hPC3iTW/F97HqKaNdaFoSRsBHqkQS7u5DwCEBPlovXJyWJGMAc9fRQB8S/tB/8l28Q/wDbt/6TRV9IeJvjFpvh3RLmDVdP1Cy8Q7Gig017SRlnnxhRHMqlGUnBBBzg8gHIr5u/aBZW+OviEqQRm3GQe4toga+2gQygqQQeQR3oA+fv2b/hbqvh+6uvFfiO1msZpoPs9nazLtfYxVmkYdV6AAHB+9kdKyv2pfBur3euad4psbSW6sI7H7JcNFGW+zlHdwzY5CkOecYG3k8ivpeigDzD4M+MLrxH4P0SwtdCvrOy0zTYbe4vr1AqTukaoBDg/MDjcSQMDjvXDfGP4b6rpXxL074jeG9Pn1KCK7t7i/tLZS8qPEy/OqjkqQoBx0IJPB4+iKKAPBfil8RtK+JHgK48KeA7bUdY1e9nhSa2jsZUNqFkDHzCygDlMdcDk5wK774PfD5vh14Di067ZH1G5kNzeMhyocgAKD6KAB7nJ713lFAHxt8YPDXiXxX8V9Z1rQvCviG5sLkwiKU6PcJu2wxoThkBHKnqK+vtM1GLVtNhvYIrmFJQSI7u2eCVcEghkcBhyO456jIINW6KACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPnf4q/8nX+Bf+uNp/6UzV9EV87/ABV/5Ov8C/8AXG0/9KZq+iKACiiigAr53/an/wCQr4G/67XX/oVvX0RXzt+1QQuqeBySABNdEk/71vQB9E0Vm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGb/wAK48D/APQm+H//AAVwf/E1t6fptjpFhHY6VZW9jaRZ8u3toljjTJJOFUADJJP1Jqt/wkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAGlRWb/wkmh/9BnT/APwKT/Gj/hJND/6DOn/+BSf40AaVFZv/AAkmh/8AQZ0//wACk/xo/wCEk0P/AKDOn/8AgUn+NAGlRWb/AMJJof8A0GdP/wDApP8AGj/hJND/AOgzp/8A4FJ/jQBpUVm/8JJof/QZ0/8A8Ck/xo/4STQ/+gzp/wD4FJ/jQBpUVm/8JJof/QZ0/wD8Ck/xo/4STQ/+gzp//gUn+NAGlRWb/wAJJof/AEGdP/8AApP8aP8AhJND/wCgzp//AIFJ/jQBpUVm/wDCSaH/ANBnT/8AwKT/ABo/4STQ/wDoM6f/AOBSf40AaVFZv/CSaH/0GdP/APApP8aP+Ek0P/oM6f8A+BSf40AaVFZv/CSaH/0GdP8A/ApP8aP+Ek0P/oM6f/4FJ/jQBpUVm/8ACSaH/wBBnT//AAKT/Gj/AISTQ/8AoM6f/wCBSf40AaVFZv8Awkmh/wDQZ0//AMCk/wAaP+Ek0P8A6DOn/wDgUn+NAGlRWb/wkmh/9BnT/wDwKT/Gj/hJND/6DOn/APgUn+NAHhPxV/5Ov8C/9cbT/wBKZq+iK+cPiVe2t/8AtVeBpbG5huYxFaKXhkDgH7TLxkd+RX0fQAUUUUAFcJ8S/hRpXxP/ALM/te/vLT+zvN8v7Lt+bzNmc7gemwfma7uigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8J/4ZR8K/9B3WPzi/+Io/4ZR8K/8AQd1j84v/AIivdqKAPCf+GUfCv/Qd1j84v/iKP+GUfCv/AEHdY/OL/wCIr3aigDwn/hlHwr/0HdY/OL/4ij/hlHwr/wBB3WPzi/8AiK92ooA8b8Ofs2eHPDXiXT9atNZ1SWawuEnSOQx7WKnIBwucV7JRRQAUUUUAf//Z)

Figure 18 protein inhibition system architecture

**4.3 Logo:**

Icon

Description automatically generated

Figure 19 st-inpro logo

**4.4 System Design:**

**4.4.1 Home:**

A screenshot of a computer

Description automatically generated

Figure 20: home page

* + 1. **Enter primary sequence:**

**Graphical user interface, website

Description automatically generated**

Figure 21: insert amino acid sequence

* + 1. **How it works:**

**Graphical user interface, website

Description automatically generated**

Figure 22: how tool works

* + 1. **About us:**

Graphical user interface, website

Description automatically generated

Figure 23: About us

**Chapter 6:**

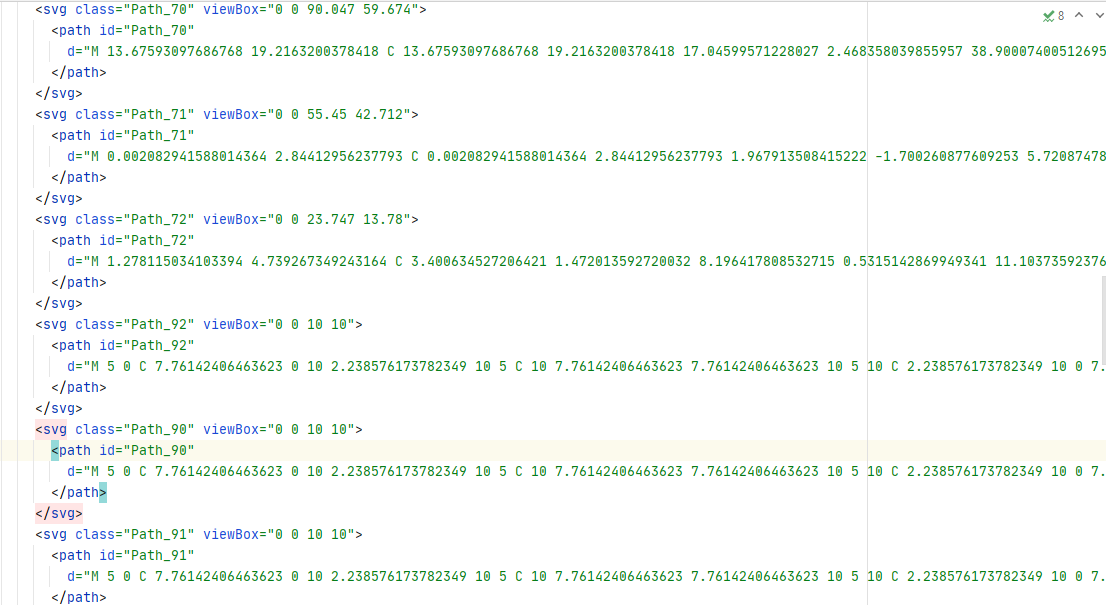
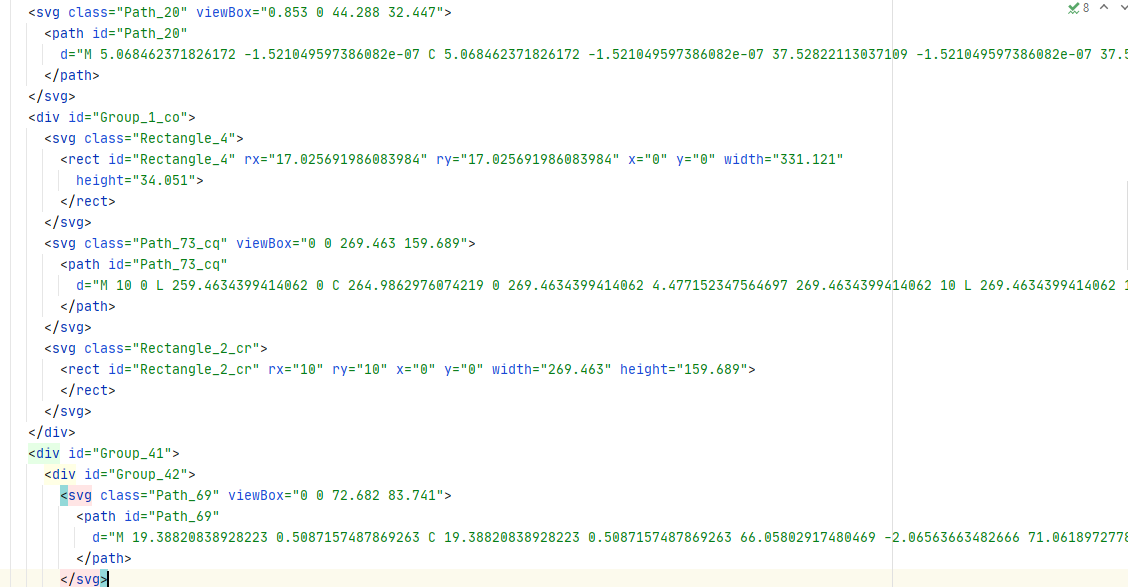
**Website Implementation**

**Home:**

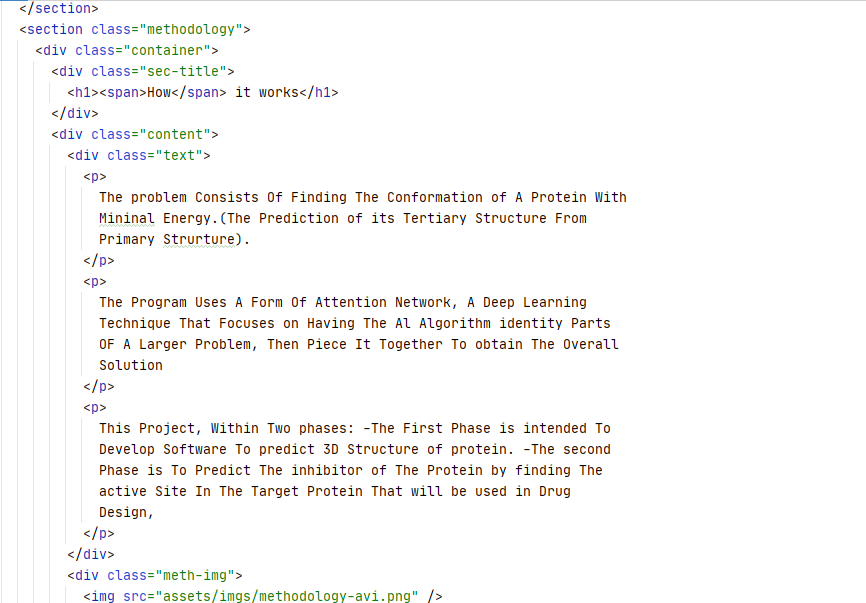
****

**Enter primary sequence:**

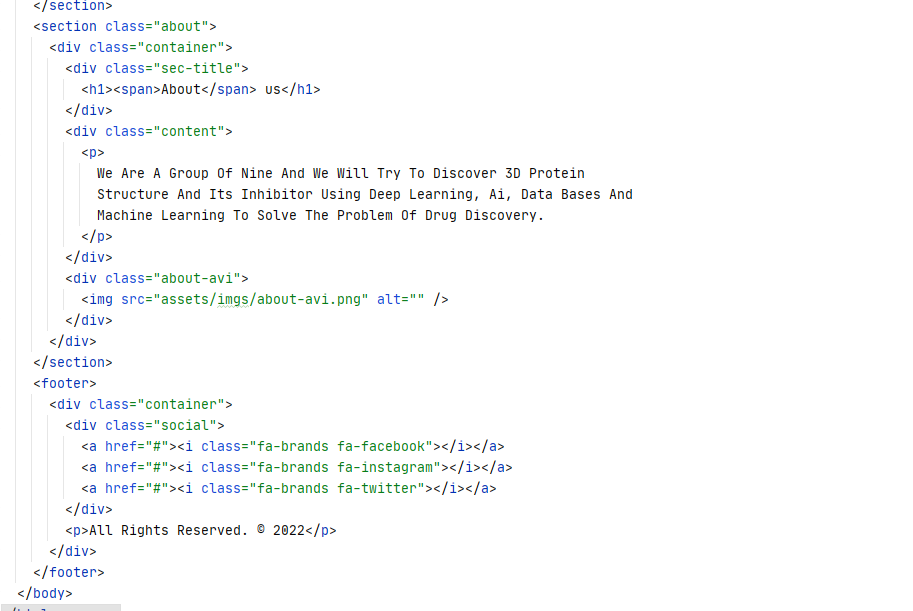
****

****

**How it works:**

****

**About us:**

****

**Chapter 5:**

**System implementation**

**Implementation steps:**

**In this chapter, we will clarify the methods that have been used to predict protein structure:**

1. Templet selection
2. Target templet alignment
3. Model construction
4. Model assessment

**At first taking Input from user:**

Input can be in **different formats:**

* Fasta
* Stockholm
* A3m

1-identify some sequence constants:

# constants

GAP = "-"

MATCH\_GAP = GAP

INSERT\_GAP = "."

ALPHABET\_PROTEIN\_NOGAP = "ACDEFGHIKLMNPQRSTVWY"

ALPHABET\_PROTEIN = GAP + ALPHABET\_PROTEIN\_NOGAP

# amino acid alphabet ordered by amino acid properties

ALPHABET\_PROTEIN\_NOGAP\_ORDERED = "KRHEDNQTSCGAVLIMPYFW"

# keep in line with convention that first character is gap

ALPHABET\_PROTEIN\_ORDERED = GAP + ALPHABET\_PROTEIN\_NOGAP\_ORDERED

ALPHABET\_DNA\_NOGAP = "ACGT"

ALPHABET\_DNA = GAP + ALPHABET\_DNA\_NOGAP

ALPHABET\_RNA\_NOGAP = "ACGU"

ALPHABET\_RNA = GAP + ALPHABET\_RNA\_NOGAP

HMMER\_PREFIX\_WARNING = "# WARNING: seq names have been made unique by adding a prefix of"

* **Identify the formats:**
  1. **FASTA format:**

def read\_fasta(fileobj):

    current\_sequence = ""

    current\_id = None

    for line in fileobj:

        # Start reading new entry. If we already have

        # seen an entry before, return it first.

        if line.startswith(">"):

            if current\_id is not None:

                yield current\_id, current\_sequence

            current\_id = line.rstrip()[1:]

            current\_sequence = ""

        elif not line.startswith(";"):

            current\_sequence += line.rstrip()

    # Also do not forget last entry in file

    yield current\_id, current\_sequence

def write\_fasta(sequences, fileobj, width=80):

    for (seq\_id, seq) in sequences:

        fileobj.write(">{}\n".format(seq\_id))

        fileobj.write(wrap(seq, width=width) + "\n")

-then write **ALN format file**

Currently, the file will not contain headers but simply

a block matrix of the alignment

  for (seq\_id, seq) in sequences:

        fileobj.write(seq + "\n")

* 1. **Stockholm format:**

Stockholm format is a multiple sequence alignment format

used by Pfam and Rfam to disseminate protein and RNA sequence alignments.

The alignment editors Ralee Belvu and Jalview support Stockholm format

as do the probabilistic database search tools, Infernal and HMMER,

and the phylogenetic analysis tool Xrate.

#=GF <feature> <Generic per-File annotation, free text>

#=GC <feature> <Generic per-Column annotation, exactly 1 char per column>

#=GS <seqname> <feature> <Generic per-Sequence annotation, free text>

#=GR <seqname> <feature> <Generic per-Residue annotation, exactly 1 char per residue>

StockholmAlignment = namedtuple(

    "StockholmAlignment",

    ["seqs", "gf", "gc", "gs", "gr"]

)

def read\_stockholm(fileobj, read\_annotation=False, raise\_hmmer\_prefixes=True):

    seqs = DefaultOrderedDict(str)

    gf = DefaultOrderedDict(list)

    gc = DefaultOrderedDict(str)

    gs = DefaultOrderedDict(lambda: DefaultOrderedDict(list))

    gr = DefaultOrderedDict(lambda: DefaultOrderedDict(str))

    # line counter within current alignment (can be more than one per file)

    i = 0

    # read alignment

    for line in fileobj:

        if i == 0 and not line.startswith("# STOCKHOLM 1.0"):

            raise ValueError(

                "Not a valid Stockholm alignment: "

                "Header missing. {}".format(line.rstrip())

            )

        if raise\_hmmer\_prefixes and line.startswith(HMMER\_PREFIX\_WARNING):

            raise ValueError(

                "HMMER added identifier prefixes to alignment because of non-unique "

                "sequence identifiers. Either some sequence identifier is present "

                "twice in the sequence database, or your target sequence identifier is "

                "the same as an identifier in the database. In the first case, please fix "

                "your sequence database. In the second case, please choose a different "

                "sequence identifier for your target sequence that does not overlap with "

                "the sequence database."

            )

        # annotation lines

        if line.startswith("#"):

            if read\_annotation:

                if line.startswith("#=GF"):

                    # can have multiple lines for the same feature

                    \_, feat, val = line.rstrip().split(maxsplit=2)

                    gf[feat].append(val)

                elif line.startswith("#=GC"):

                    # only one line with the same GC label

                    \_, feat, seq = line.rstrip().split(maxsplit=2)

                    gc[feat] += seq

                elif line.startswith("#=GS"):

                    # can have multiple lines for the same feature

                    \_, seq\_id, feat, val = line.rstrip().split(maxsplit=3)

                    gs[seq\_id][feat] = val

                elif line.startswith("#=GR"):

                    # per sequence, only one line with a certain GR feature

                    \_, seq\_id, feat, seq = line.rstrip().split()

                    gr[seq\_id][feat] += seq

            i += 1

        # terminator line for current alignment;

        # only yield once we see // to avoid reading

        # truncated alignments

        elif line.startswith("//"):

            yield StockholmAlignment(seqs, gf, gc, gs, gr)

            # reset counter to check for valid start of alignment

            # once we read the next line

            i = 0

        # actual alignment lines

        else:

            splitted = line.rstrip().split(maxsplit=2)

            # there might be empty lines, so check for valid split

            if len(splitted) == 2:

                seq\_id, seq = splitted

                seqs[seq\_id] += seq

            i += 1

    # Do NOT yield at the end without // to avoid returning truncated alignments

* 1. **A3m format:**

The A3M format consists of aligned fasta, in which alignments are shown with inserts as lower case characters, matches as upper case characters, deletions as ' - ', and gaps aligned to inserts as ' . '

def read\_a3m(fileobj, inserts="first"):

seqs = OrderedDict()

    for i, (seq\_id, seq) in enumerate(read\_fasta(fileobj)):

        # remove any insert gaps that may still be in alignment

        # (just to be sure)

        seq = seq.replace(".", "")

        if inserts == "first":

            # define "spacing" of uppercase columns in

            # final alignment based on target sequence;

            # remaining columns will be filled with insert

            # gaps in the other sequences

            if i == 0:

                uppercase\_cols = [

                    j for (j, c) in enumerate(seq)

                    if (c == c.upper() or c == "-")

                ]

                gap\_template = np.array(["."] \* len(seq))

                filled\_seq = seq

            else:

                uppercase\_chars = [

                    c for c in seq if c == c.upper() or c == "-"

                ]

                filled = np.copy(gap\_template)

                filled[uppercase\_cols] = uppercase\_chars

                filled\_seq = "".join(filled)

        elif inserts == "delete":

            # remove all lowercase letters and insert gaps .;

            # since each sequence must have same number of

            # uppercase letters or match gaps -, this gives

            # the final sequence in alignment

            seq = "".join([c for c in seq if c == c.upper() and c != "."])

        else:

            raise ValueError(

                "Invalid option for inserts: {}".format(inserts)

            )

        seqs[seq\_id] = filled\_seq

    return seqs

def write\_a3m(sequences, fileobj, insert\_gap=INSERT\_GAP, width=80):

    for (seq\_id, seq) in sequences:

        fileobj.write(">{}\n".format(seq\_id))

        fileobj.write(seq.replace(insert\_gap, "") + "\n")

* **detect format:**

Detect if an alignment file is in FASTA or Stockholm format. (a3m is an aligned fasta)

def detect\_format(fileobj):

    for i, line in enumerate(fileobj):

        # must be first line of Stockholm file by definition

        if i == 0 and line.startswith("# STOCKHOLM 1.0"):

            return "stockholm"

        # This indicates a FASTA file

        if line.startswith(">"):

            return "fasta"

        # Skip comment lines and empty lines for FASTA detection

        if line.startswith(";") or line.rstrip() == "":

            continue

        # Arriving here means we could not detect format

        return None

* **parse header:**

Extract ID of the (overall) sequence and the sequence range format a sequence header of the form sequence ID/start-end.

If the header contains any additional information

after the first whitespace character (e.g. sequence annotation),

it will be discarded before parsing. If there is no sequence range, only the id (part of string before whitespace) will be returned but no range.

def parse\_header(header):

    header = header.split()[0]

    # try to extract region from sequence header

    m = re.search("(.+)/(\d+)-(\d+)", header)

    if m:

        id\_, start\_str, end\_str = m.groups()

        region\_start, region\_end = int(start\_str), int(end\_str)

        return id\_, region\_start, region\_end

    else:

        # cannot find region, so just give back sequence iD

        return header, None, None

* **sequences\_to\_matrix:**

Transforms a list of sequences into a numpy array.

def sequences\_to\_matrix(sequences):

if len(sequences) == 0:

        raise ValueError("Need at least one sequence")

    N = len(sequences)

    L = len(next(iter(sequences)))

    matrix = np.empty((N, L), dtype=np.str)

    for i, seq in enumerate(sequences):

        if len(seq) != L:

            raise ValueError(

                "Sequences have differing lengths: i={} L\_0={} L\_i={}".format(

                    i, L, len(seq)

                )

            )

        matrix[i] = np.array(list(seq))

    return matrix

* **map from alphabet:**

Creates a mapping dictionary from a given alphabet.

def map\_from\_alphabet(alphabet=ALPHABET\_PROTEIN, default=GAP):

map\_ = {

        c: i for i, c in enumerate(alphabet)

    }

    try:

        default = map\_[default]

    except KeyError:

        raise ValueError(

            "Default {} is not in alphabet {}".format(default, alphabet)

        )

    return defaultdict(lambda: default, map\_)

* **map matrix:**

Map elements in a numpy array using alphabet

 def map\_matrix(matrix, map\_):

return np.vectorize(map\_.\_\_getitem\_\_)(matrix)

* **Alignment:**

Container to store and manipulate multiple sequence alignments.

Create new alignment object from ready-made components.

Use factory method Alignment from file to create alignment from file,

or Alignment from dictionary of sequences.

class Alignment:

    def \_\_init\_\_(self, sequence\_matrix, sequence\_ids=None, annotation=None,

                 alphabet=ALPHABET\_PROTEIN):

        self.matrix = np.array(sequence\_matrix)

        self.N, self.L = self.matrix.shape

        # characters coding for gaps in match-state and insert

        # columns of the alignment

        self.\_match\_gap = MATCH\_GAP

        self.\_insert\_gap = INSERT\_GAP

        # defined alphabet of alignment

        self.alphabet = alphabet

        self.alphabet\_default = self.\_match\_gap

        self.alphabet\_map = map\_from\_alphabet(

            self.alphabet, default=self.alphabet\_default

        )

        self.num\_symbols = len(self.alphabet\_map)

        # Alignment matrix remapped into in integers

        # Will only be calculated if necessary for downstream

        # calculations

        self.matrix\_mapped = None

        self.num\_cluster\_members = None

        self.weights = None

        self.\_frequencies = None

        self.\_pair\_frequencies = None

        if sequence\_ids is None:

            # default to numbering sequences if not given

            self.ids = [str(i) for i in range(self.N)]

        else:

            if len(sequence\_ids) != self.N:

                raise ValueError(

                    "Number of sequence IDs and length of "

                    "alignment do not match".format(

                        len(sequence\_ids), self.L

                    )

                )

            # make sure we get rid of iterators etc.

            self.ids = np.array(list(sequence\_ids))

        self.id\_to\_index = {

            id\_: i for i, id\_ in enumerate(self.ids)

        }

        if annotation is not None:

            self.annotation = annotation

        else:

            self.annotation = {}

 Construct an alignment object from a dictionary with sequence IDs as keys and aligned sequences as values.

    @classmethod

    def from\_dict(cls, sequences, \*\*kwargs):

        matrix = sequences\_to\_matrix(sequences.values())

        return cls(

            matrix, sequences.keys(), \*\*kwargs

        )

Construct an alignment object by reading in an alignment file.

  @classmethod

    def from\_file(cls, fileobj, format="fasta",

                  a3m\_inserts="first", raise\_hmmer\_prefixes=True,

                  \*\*kwargs):

        annotation = {}

        # read in sequence alignment from file

        if format == "fasta":

            seqs = OrderedDict()

            for seq\_id, seq in read\_fasta(fileobj):

                seqs[seq\_id] = seq

        elif format == "stockholm":

            # only reads first Stockholm alignment contained in file

            ali = next(

                read\_stockholm(

                    fileobj, read\_annotation=True,

                    raise\_hmmer\_prefixes=raise\_hmmer\_prefixes

                )

            )

            seqs = ali.seqs

            annotation["GF"] = ali.gf

            annotation["GC"] = ali.gc

            annotation["GS"] = ali.gs

            annotation["GR"] = ali.gr

            kwargs["annotation"] = annotation

        elif format == "a3m":

            seqs = read\_a3m(fileobj, inserts=a3m\_inserts)

        else:

            raise ValueError("Invalid alignment format: {}".format(format))

        return cls.from\_dict(seqs, \*\*kwargs)

allow fancy indexing and offer the functionality of select ().

 def \_\_getitem\_\_(self, index):

        if index in self.id\_to\_index:

            return self.matrix[self.id\_to\_index[index], :]

        elif index in range(self.N):

            return self.matrix[index, :]

        else:

            raise KeyError(

                "Not a valid index for sequence alignment: {}".format(index)

            )

Count occurrences of a character in the sequence alignment.

The counts are raw counts not adjusted for sequence redundancy.

    def count(self, char, axis="pos", normalize=True):

        if axis == "pos":

            naxis = 0

        elif axis == "seq":

            naxis = 1

        else:

            raise ValueError("Invalid axis: {}".format(axis))

        c = np.sum(self.matrix == char, axis=naxis)

        if normalize:

            c = c / self.matrix.shape[naxis]

        return c

Create a sub-alignment that contains a subset of sequences and/or columns.

def select(self, columns=None, sequences=None):

        if columns is None and sequences is None:

            return self

        sel\_matrix = self.matrix

        ids = self.ids

        if columns is not None:

            sel\_matrix = sel\_matrix[:, columns]

        if sequences is not None:

            sel\_matrix = sel\_matrix[sequences, :]

            ids = ids[sequences]

        # do not copy annotation since it may become

        # inconsistent

        return Alignment(

            np.copy(sel\_matrix), np.copy(ids),

            alphabet=self.alphabet

        )

Apply a function along columns and/or rows of alignment matrix, or to entire matrix.

* columns : np.array(bool) or np.array(int), optional Vector containing True for each column that should be retained, False otherwise; or the indices of columns that should be selected
* sequences : np.array(bool) or np.array(int), optional Vector containing True for each sequence that should be retained, False otherwise; or the indices of sequences that should be selected
* func : callable Vectorized numpy function that will be applied to the selected subset of the alignment matrix

Alignment with modified columns and sequences (this alignment maintains annotation)

   def apply(self, columns=None, sequences=None, func=np.char.lower):

        mod\_matrix = np.copy(self.matrix)

        if columns is None and sequences is None:

            return self

        else:

            if columns is not None:

                mod\_matrix[:, columns] = func(mod\_matrix[:, columns])

            if sequences is not None:

                mod\_matrix[sequences, :] = func(mod\_matrix[sequences, :])

        return Alignment(

            mod\_matrix, np.copy(self.ids), deepcopy(self.annotation),

            alphabet=self.alphabet

        )

Replace character with another in full matrix or subset of columns/sequences.

   def replace(self, original, replacement, columns=None, sequences=None):

        return self.apply(

            columns, sequences,

            func=lambda x: np.char.replace(

                x, original, replacement

            )

        )

Change a subset of columns to lowercase character and replace "-" gaps with "." gaps, e.g. to exclude them from EC calculations

  def lowercase\_columns(self, columns):

        return self.apply(

            columns=columns, func=np.char.lower

        ).replace(

            self.\_match\_gap, self.\_insert\_gap, columns=columns

        )

Ensure self-matrix mapped exists and Calculate weights for sequences in alignment in alignment by clustering all sequences with sequence identity greater or equal to the given threshold..

    def \_\_ensure\_mapped\_matrix(self):

        if self.matrix\_mapped is None:

            self.matrix\_mapped = map\_matrix(

                self.matrix, self.alphabet\_map

            )

    def set\_weights(self, identity\_threshold=0.8):

        self.\_\_ensure\_mapped\_matrix()

        self.num\_cluster\_members = num\_cluster\_members(

            self.matrix\_mapped, identity\_threshold

        )

        self.weights = 1.0 / self.num\_cluster\_members

        # reset frequencies, since these were based on

        # different weights before or had no weights at all

        self.\_frequencies = None

        self.\_pair\_frequencies = None

calculates single-site frequencies of symbols in alignment. Also sets self. frequencies member variable for later reuse.

Previously calculated sequence weights using self-set weights ()

will be used to adjust frequency counts; otherwise, each sequence will contribute with equal weight.

    @property

    def frequencies(self):

        if self.\_frequencies is None:

            self.\_\_ensure\_mapped\_matrix()

            # use precalculated sequence weights, but only

            # if we have explicitly calculated them before

            # (expensive calculation)

            if self.weights is None:

                weights = np.ones((self.N))

            else:

                weights = self.weights

            self.\_frequencies = frequencies(

                self.matrix\_mapped, weights, self.num\_symbols

            )

        return self.\_frequencies

calculates pairwise frequencies of symbols in alignment. Also sets self.\_pair frequencies member variable for later reuse.

Previously calculated sequence weights using self.set weights() will be used to adjust frequency counts; otherwise, each sequence will contribute with equal weight.

 @property

    def pair\_frequencies(self):

        if self.\_pair\_frequencies is None:

            self.\_\_ensure\_mapped\_matrix()

            if self.weights is None:

                weights = np.ones((self.N))

            else:

                weights = self.weights

            self.\_pair\_frequencies = pair\_frequencies(

                self.matrix\_mapped, weights,

                self.num\_symbols, self.frequencies

            )

        return self.\_pair\_frequencies

Calculate sequence identity between sequence and all sequences in the alignment.

  def identities\_to(self, seq, normalize=True):

        self.\_\_ensure\_mapped\_matrix()

        # make sure this doesnt break with strings

        seq = np.array(list(seq))

        seq\_mapped = map\_matrix(seq, self.alphabet\_map)

        ids = identities\_to\_seq(seq\_mapped, self.matrix\_mapped)

        if normalize:

            return ids / self.L

        else:

            return ids

Calculate per-column conservation of sequence alignment based on entropy of single-column frequency distribution.

  def conservation(self, normalize=True):

        return np.apply\_along\_axis(

            lambda x: entropy(x, normalize=normalize),

            axis=1, arr=self.frequencies

        )

**Write an alignment to a file.**

  def write(self, fileobj, format="fasta", width=80):

        seqs = (

            (id\_, "".join(self.matrix[i]))

            for (i, id\_) in enumerate(self.ids)

        )

        if format == "fasta":

            write\_fasta(seqs, fileobj, width)

        elif format == "a3m":

            write\_a3m(seqs, fileobj, self.\_insert\_gap, width)

        elif format == "aln":

            write\_aln(seqs, fileobj, width)

        else:

            raise ValueError(

                "Invalid alignment format: {}".format(format)

            )

Calculate single-site frequencies of symbols in alignment

@jit(nopython=True)

def frequencies(matrix, seq\_weights, num\_symbols):

    N, L = matrix.shape

    fi = np.zeros((L, num\_symbols))

    for s in range(N):

        for i in range(L):

            fi[i, matrix[s, i]] += seq\_weights[s]

    return fi / seq\_weights.sum()

Calculate pairwise frequencies of symbols in alignment.

@jit(nopython=True)

def pair\_frequencies(matrix, seq\_weights, num\_symbols, fi):

    N, L = matrix.shape

    fij = np.zeros((L, L, num\_symbols, num\_symbols))

    for s in range(N):

        for i in range(L):

            for j in range(i + 1, L):

                fij[i, j, matrix[s, i], matrix[s, j]] += seq\_weights[s]

                fij[j, i, matrix[s, j], matrix[s, i]] = fij[i, j, matrix[s, i], matrix[s, j]]

    # normalize frequencies by the number

    # of effective sequences

    fij /= seq\_weights.sum()

    # set the frequency of a pair (alpha, alpha)

    # in position i to the respective single-site

    # frequency of alpha in position i

    for i in range(L):

        for alpha in range(num\_symbols):

            fij[i, i, alpha, alpha] = fi[i, alpha]

    return fij

Calculate number of identities to given target sequence for all sequences in the matrix.

@jit(nopython=True)

def identities\_to\_seq(seq, matrix):

    N, L = matrix.shape

    identities = np.zeros((N, ))

    for i in range(N):

        id\_i = 0

        for j in range(L):

            if matrix[i, j] == seq[j]:

                id\_i += 1

        identities[i] = id\_i

    return identities

Calculate number of sequences in alignment within given identity threshold of each other.

@jit(nopython=True)

def num\_cluster\_members(matrix, identity\_threshold):

    N, L = matrix.shape

    L = 1.0 \* L

    # minimal cluster size is 1 (self)

    num\_neighbors = np.ones((N))

    # compare all pairs of sequences

    for i in range(N - 1):

        for j in range(i + 1, N):

            pair\_id = 0

            for k in range(L):

                if matrix[i, k] == matrix[j, k]:

                    pair\_id += 1

            if pair\_id / L >= identity\_threshold:

                num\_neighbors[i] += 1

                num\_neighbors[j] += 1

    return num\_neighbors

* **Hmmer:**

Its general usage is to identify homologous protein or nucleotide sequences, and to perform sequence alignments. It detects homology by comparing a profile-HMM (a Hidden Markov model constructed explicitly for a particular search) to either a single sequence or a database of sequences.

hmmbuild - construct profile HMMs from multiple sequence alignments

HmmbuildResult = namedtuple(

    "HmmbuildResult",

    ["prefix", "hmmfile", "output"]

)

def run\_hmmbuild(alignment\_file, prefix, cpu=None,

                 stdout\_redirect=None, symfrac=None,

                 binary="hmmbuild"):

    verify\_resources(

        "Input file does not exist or is empty",

        alignment\_file

    )

    create\_prefix\_folders(prefix)

    # store filenames of all individual results;

    # these will be returned as result of the

    # function.

    result = HmmbuildResult(

        prefix,

        prefix + ".hmm",

        prefix + ".output" if stdout\_redirect is None else stdout\_redirect,

    )

    cmd = [

        binary,

        "-o", result.output,

    ]

    # number of CPUs

    if cpu is not None:

        cmd += ["--cpu", str(cpu)]

    if symfrac is not None:

        cmd += ["--symfrac", str(symfrac)]

    cmd += [result.hmmfile, alignment\_file]

    return\_code, stdout, stderr = run(cmd)

    # also check we actually created some sort of alignment

    verify\_resources(

        "hmmbuild returned empty HMM profile: "

        "stdout={} stderr={} file={}".format(

            stdout, stderr, result.hmmfile

        ),

        result.hmmfile

    )

    return result

# output fields for storing results of a hmmsearch run

# (returned by run\_hmmsearch)

HmmsearchResult = namedtuple(

    "HmmsearchResult",

    ["prefix", "alignment", "output", "tblout", "domtblout"]

)

hmmsearch - search profile HMMs against a sequence database

def run\_hmmsearch(hmmfile, database, prefix,

                  use\_bitscores, domain\_threshold, seq\_threshold,

                  nobias=False, cpu=None,

                  stdout\_redirect=None, binary="hmmsearch"):

    verify\_resources(

        "Input file does not exist or is empty",

        hmmfile, database

    )

    create\_prefix\_folders(prefix)

    # store filenames of all individual results;

    # these will be returned as result of the

    # function.

    result = HmmsearchResult(

        prefix,

        prefix + ".sto",

        prefix + ".output" if stdout\_redirect is None else stdout\_redirect,

        prefix + ".tblout",

        prefix + ".domtblout"

    )

    cmd = [

        binary,

        "-o", result.output,

        "-A", result.alignment,

        "--tblout", result.tblout,

        "--domtblout", result.domtblout,

        "--noali",

        "--notextw"

    ]

    # reporting thresholds are set accordingly to

    # inclusion threshold to reduce memory footprint

    if use\_bitscores:

        cmd += [

            "-T", str(seq\_threshold),

            "--domT", str(domain\_threshold),

            "--incT", str(seq\_threshold),

            "--incdomT", str(domain\_threshold)

        ]

    else:

        cmd += [

            "-E", str(seq\_threshold),

            "--domE", str(domain\_threshold),

            "--incE", str(seq\_threshold),

            "--incdomE", str(domain\_threshold)

        ]

    # number of CPUs

    if cpu is not None:

        cmd += ["--cpu", str(cpu)]

    # bias correction filter

    if nobias:

        cmd += ["--nobias"]

    cmd += [hmmfile, database]

    return\_code, stdout, stderr = run(cmd)

    return result

jackhmmer - iteratively search sequences against a protein database

# output fields for storing results of a jackhmmer run

# (returned by run\_jackhmmer)

JackhmmerResult = namedtuple(

    "JackhmmerResult",

    ["prefix", "alignment", "output", "tblout", "domtblout"]

)

def run\_jackhmmer(query, database, prefix,

                  use\_bitscores, domain\_threshold, seq\_threshold,

                  iterations=5, nobias=False, cpu=None,

                  stdout\_redirect=None, checkpoints\_hmm=False,

                  checkpoints\_ali=False, binary="jackhmmer"):

    verify\_resources(

        "Input file does not exist or is empty",

        query, database

    )

    create\_prefix\_folders(prefix)

    # store filenames of all individual results;

    # these will be returned as result of the

    # function.

    result = JackhmmerResult(

        prefix,

        prefix + ".sto",

        prefix + ".output" if stdout\_redirect is None else stdout\_redirect,

        prefix + ".tblout",

        prefix + ".domtblout"

    )

    cmd = [

        binary,

        "-N", str(iterations),

        "-o", result.output,

        "-A", result.alignment,

        "--tblout", result.tblout,

        "--domtblout", result.domtblout,

        "--noali",

        "--notextw"

    ]

    # reporting thresholds are set accordingly to

    # inclusion threshold to reduce memory footprit

    if use\_bitscores:

        cmd += [

            "-T", str(seq\_threshold),

            "--domT", str(domain\_threshold),

            "--incT", str(seq\_threshold),

            "--incdomT", str(domain\_threshold)

        ]

    else:

        cmd += [

            "-E", str(seq\_threshold),

            "--domE", str(domain\_threshold),

            "--incE", str(seq\_threshold),

            "--incdomE", str(domain\_threshold)

        ]

    # number of CPUs

    if cpu is not None:

        cmd += ["--cpu", str(cpu)]

    # bias correction filter

    if nobias:

        cmd += ["--nobias"]

    # save checkpoints for alignments and HMMs?

    if checkpoints\_ali:

        cmd += ["--chkali", prefix]

    if checkpoints\_hmm:

        cmd += ["--chkhmm", prefix]

    cmd += [query, database]

    return\_code, stdout, stderr = run(cmd)

    # also check we actually created some sort of alignment

    verify\_resources(

        "jackhmmer returned empty alignment: "

        "stdout={} stderr={} file={}".format(

            stdout, stderr, result.alignment

        ),

        result.alignment

    )

    return result

hmmscan - search protein sequences against a profile HMM database

HmmscanResult = namedtuple(

    "HmmscanResult",

    ["prefix", "output", "tblout", "domtblout", "pfamtblout"]

)

def run\_hmmscan(query, database, prefix,

                use\_model\_threshold=True, threshold\_type="cut\_ga",

                use\_bitscores=True, domain\_threshold=None, seq\_threshold=None,

                nobias=False, cpu=None, stdout\_redirect=None, binary="hmmscan"):

    verify\_resources(

        "Input file does not exist or is empty",

        query, database

    )

    create\_prefix\_folders(prefix)

    result = HmmscanResult(

        prefix,

        prefix + ".output" if stdout\_redirect is None else stdout\_redirect,

        prefix + ".tblout",

        prefix + ".domtblout",

        prefix + ".pfamtblout"

    )

    cmd = [

        binary,

        "-o", result.output,

        "--tblout", result.tblout,

        "--domtblout", result.domtblout,

        "--pfamtblout", result.pfamtblout,

        "--notextw",

        "--acc",

    ]

    # number of CPUs

    if cpu is not None:

        cmd += ["--cpu", str(cpu)]

    # bias correction filter

    if nobias:

        cmd += ["--nobias"]

    # either use model-specific threshold, or custom

    # bitscore/E-value thresholds

    if use\_model\_threshold:

        THRESHOLD\_CHOICES = ["cut\_ga", "cut\_nc", "cut\_tc"]

        if threshold\_type not in THRESHOLD\_CHOICES:

            raise ValueError(

                "Invalid model threshold, valid choices are: " +

                ", ".join(THRESHOLD\_CHOICES)

            )

        cmd += ["--" + threshold\_type]

    else:

        if seq\_threshold is None or domain\_threshold is None:

            raise ValueError(

                "Must define sequence- and domain-level reporting"

                "thresholds, or use gathering threshold instead."

            )

        if use\_bitscores:

            cmd += [

                "-T", str(seq\_threshold),

                "--domT", str(domain\_threshold),

            ]

        else:

            cmd += [

                "-E", str(seq\_threshold),

                "--domE", str(domain\_threshold),

            ]

    cmd += [database, query]

    return\_code, stdout, stderr = run(cmd)

    # also check we actually created a table with hits

    verify\_resources(

        "hmmscan did not return results: "

        "stdout={} stderr={} file={}".format(

            stdout, stderr, result.domtblout

        ),

        result.domtblout

    )

    return result

Parse a HMMER file in (dom)tbl format into a pandas DataFrame.

def \_read\_hmmer\_table(filename, column\_names):

    res = []

    num\_splits = len(column\_names) - 1

    with open(filename) as f:

        for line in f:

            if line.startswith("#"):

                continue

            fields = line.rstrip().split(maxsplit=num\_splits)

            res.append(fields)

    # at the moment, all fields in dataframe are strings, even

    # if numeric. To convert to numbers, cheap trick is to store

    # to csv file and let pandas guess the types, rather than

    # going through convert\_objects (deprecated) or to\_numeric

    # (more effort)

    tempfile = temp()

    pd.DataFrame(

        res, columns=column\_names

    ).to\_csv(tempfile, index=False)

    return pd.read\_csv(tempfile)

Read a HMMER tbl file into DataFrame.

def read\_hmmer\_tbl(filename):

    column\_names = [

        "target\_name", "target\_accession",

        "query\_name", "query\_accession",

        "full\_Evalue", "full\_score", "full\_bias",

        "best\_domain\_Evalue", "best\_domain\_score",

        "best\_domain\_bias",

        "domain\_exp", "domain\_reg", "domain\_clu",

        "domain\_ov", "domain\_env", "domain\_dom",

        "domain\_rep", "domain\_inc",

        "description"

    ]

    return \_read\_hmmer\_table(filename, column\_names)

Read a HMMER domtbl file into DataFrame.

def read\_hmmer\_domtbl(filename):

    column\_names = [

        "target\_name", "target\_accession", "target\_len",

        "query\_name", "query\_accession", "query\_len",

        "full\_Evalue", "full\_score", "full\_bias",

        "hit\_number", "total\_hit\_number",

        "domain\_c\_Evalue", "domain\_i\_Evalue",

        "domain\_score", "domain\_bias",

        "hmm\_from", "hmm\_to",

        "ali\_from", "ali\_to",

        "env\_from", "env\_to",

        "acc", "description"

    ]

    return \_read\_hmmer\_table(filename, column\_names)

hhfilter Filter: an MSA by maximum sequence identity, coverage, and other criteria. reduce a sequence alignment using hhfilter from the HHsuite alignment suite.

def run\_hhfilter(input\_file, output\_file, threshold=95,

                 columns="a2m", binary="hhfilter"):

    if columns not in ["first", "a2m"]:

        raise ValueError(

            "Invalid column selection: {}".format(columns)

        )

    verify\_resources(

        "Alignment file does not exist or is empty",

        input\_file

    )

    create\_prefix\_folders(output\_file)

    cmd = [

        binary,

        "-i", input\_file,

        "-o", output\_file,

        "-id", str(threshold),

        "-M", columns,

        "-v", str(2)

    ]

    return\_code, stdout, stderr = run(cmd)

    verify\_resources(

        "hhfilter returned empty alignment: "

        "stdout={} stderr={} file={}".format(

            stdout, stderr, output\_file

        ),

        output\_file

    )

    return output\_file

**Pfam** is a database of protein families that includes their annotations and multiple sequence alignments generated using hidden Markov models. The general purpose of the Pfam database is to provide a complete and accurate classification of protein families and domains.

Parse family size table from Pfam flat file

def create\_family\_size\_table(full\_pfam\_file, outfile=None):

    data = []

    with gzip.open(full\_pfam\_file, "rt", encoding='latin-1') as gz\_ref:

        pfam\_id = None

        num\_seqs = None

        for line in gz\_ref:

            # identifier at the beginning of the family entry

            if line.startswith("#=GF AC"):

                pfam\_id = line[10:17]

            # the number of sequences in the family, follows after the identifier

            elif line.startswith("#=GF SQ"):

                num\_seqs = int(line[10:])

            # stores the result at the end of an entry

            elif (line.startswith("//") and

                    pfam\_id is not None and num\_seqs is not None):

                data.append({"pfam\_id": pfam\_id, "num\_seqs": num\_seqs})

                pfam\_id = None

                num\_seqs = None

    df = pd.DataFrame(data, columns=["pfam\_id", "num\_seqs"])

    if outfile is not None:

        df.to\_csv(outfile, index=False)

    return df

Remove overlapping Pfam hits from same Pfam clan (equivalent of PfamScan.pl). Currently only allows to remove overlaps by domain bitscore.

def remove\_clan\_overlaps(pfam\_table):

    # could make this a parameter, if switching to E-values

    # we would have to changing sorting order of DataFrame

    # and sign of comparison further below.

    score = "domain\_score"

    # group by sequence ID and clan to resolve overlaps

    grouped = pfam\_table.sort\_values(

        by=score, ascending=False

    ).groupby(

        by=["query\_name", "clan\_id"], as\_index=False, sort=False

    )

    # store index value of all entries to discard

    remove\_hits = []

    for (uniprot\_ac, clan\_name), grp in grouped:

        # safety check here that we are not grouping hits that are

        # not in the same clan (missing value) if pandas ever changed

        # the behaviour of groupby to not iterate through groups

        # with missing values. Otherwise, we would have to skip grouop.

        assert clan\_name.startswith("CL")

        # go through all pairwise combinations of hits

        for idx1, hit1 in grp.iterrows():

            for idx2, hit2 in grp.iterrows():

                if idx1 < idx2:

                    if range\_overlap(

                        (int(hit1["ali\_from"]), int(hit1["ali\_to"]) + 1),

                        (int(hit2["ali\_from"]), int(hit2["ali\_to"]) + 1),

                    ) > 0:

                        if float(hit1[score]) >= float(hit2[score]):

                            remove\_hits.append(idx2)

                        else:

                            remove\_hits.append(idx1)

    return pfam\_table.loc[~pfam\_table.index.isin(remove\_hits)]

Identify hits of Pfam HMMs in a set of sequences.

def pfam\_hits(query\_file, hmm\_database, prefix,

              clan\_table\_file, size\_table\_file,

              resolve\_overlaps=True,

              \*\*kwargs):

    # find HMM hits with hmmscan

    scan\_res = run\_hmmscan(

        query\_file, hmm\_database, prefix,

        \*\*kwargs

    )

    hits = read\_hmmer\_domtbl(scan\_res.domtblout)

    # remove version information from family name

    hits.loc[:, "pfam\_id"] = hits.target\_accession.map(

        lambda x: x.split(".")[0]

    )

    # add information about Pfam clan for each family,

    # this is necessary to resolve overlapping hits

    # clan file is Pfam-A.clans.tsv from Pfam FTP site

    if clan\_table\_file is not None:

        clans = pd.read\_csv(

            clan\_table\_file, sep="\t",

            names=[

                "pfam\_id", "clan\_id", "clan\_name",

                "family\_name", "family\_text"

            ]

        )

        hits = hits.merge(clans, on="pfam\_id", how="left")

    # add number of sequences in each Pfam family,

    # this file has to be created using create\_family\_sie\_table()

    # from Pfam-A.full.gz flatfile

    if size\_table\_file is not None:

        sizes = pd.read\_csv(size\_table\_file)

        hits = hits.merge(sizes, on="pfam\_id", how="left")

        hits.loc[:, "num\_seqs\_over\_len"] = (

            hits.loc[:, "num\_seqs"] /

            pd.to\_numeric(hits.loc[:, "target\_len"], errors="raise")

        )

    # multiple members of the same clan might hit overlapping regions

    # in these cases, we may only want to keep the top-scoring hit

    if resolve\_overlaps:

        if clan\_table\_file is None:

            raise ValueError(

                "Need to specify clan\_table\_file to resolve "

                "overlapping hits from same clan."

            )

        hits = remove\_clan\_overlaps(hits)

    return hits

* store parameters of undirected graphical model of sequences and perform calculations using the model (statistical energies, coupling scores)

Calculates the Hamiltonian of the global probability distribution P(A\_1, ..., A\_L)for a given sequence A\_1,...,A\_L from J\_ij and h\_i parameters

@jit(nopython=True)

def \_hamiltonians(sequences, J\_ij, h\_i):

    # iterate over sequences

    N, L = sequences.shape

    H = np.zeros((N, NUM\_COMPONENTS))

    for s in range(N):

        A = sequences[s]

        hi\_sum = 0.0

        Jij\_sum = 0.0

        for i in range(L):

            hi\_sum += h\_i[i, A[i]]

            for j in range(i + 1, L):

                Jij\_sum += J\_ij[i, j, A[i], A[j]]

        H[s] = [Jij\_sum + hi\_sum, Jij\_sum, hi\_sum]

    return H

Calculate matrix of all possible single-site substitutions

@jit(nopython=True)

def \_single\_mutant\_hamiltonians(target\_seq, J\_ij, h\_i):

    L, num\_symbols = h\_i.shape

    H = np.empty((L, num\_symbols, NUM\_COMPONENTS))

    # iterate over all positions

    for i in range(L):

        # iterate over all substitutions

        for A\_i in range(num\_symbols):

            # iterate over couplings to all other sites

            delta\_hi = h\_i[i, A\_i] - h\_i[i, target\_seq[i]]

            delta\_Jij = 0.0

            for j in range(L):

                if i != j:

                    delta\_Jij += (

                        J\_ij[i, j, A\_i, target\_seq[j]] -

                        J\_ij[i, j, target\_seq[i], target\_seq[j]]

                    )

            H[i, A\_i] = [delta\_Jij + delta\_hi, delta\_Jij, delta\_hi]

    return H

calculate Vector of length 3, where elements correspond to delta of

1) total Hamiltonian and the 2) J\_ij and 3) h\_i sub-sums

@jit(nopython=True)

def \_delta\_hamiltonian(pos, subs, target\_seq, J\_ij, h\_i):

    L, num\_symbols = h\_i.shape

    M = pos.shape[0]

    delta\_hi = 0.0

    delta\_Jij = 0.0

    # iterate over all changed positions

    for m in range(M):

        i = pos[m]

        A\_i = subs[m]

        # change in fields

        delta\_hi += h\_i[i, A\_i] - h\_i[i, target\_seq[i]]

        # couplings to all other positions in sequence

        for j in range(L):

            if i != j:

                delta\_Jij += (

                    J\_ij[i, j, A\_i, target\_seq[j]] -

                    J\_ij[i, j, target\_seq[i], target\_seq[j]]

                )

        # correct couplings between substituted positions:

        # 1) do not count coupling twice (remove forward

        #     and backward coupling)

        # 2) adjust background to new sequence

        for n in range(m + 1, M):

            j = pos[n]

            A\_j = subs[n]

            # remove forward and backward coupling delta

            delta\_Jij -= J\_ij[i, j, A\_i, target\_seq[j]]

            delta\_Jij -= J\_ij[i, j, target\_seq[i], A\_j]

            delta\_Jij += J\_ij[i, j, target\_seq[i], target\_seq[j]]

            # the following line cancels out with line further down:

            # delta\_Jij += J\_ij[i, j, target\_seq[i], target\_seq[j]]

            # now add coupling delta once in correct background

            delta\_Jij += J\_ij[i, j, A\_i, A\_j]

            # following line cancels out with line above:

            # delta\_Jij -= J\_ij[i, j, target\_seq[i], target\_seq[j]]

    return np.array([delta\_Jij + delta\_hi, delta\_Jij, delta\_hi])

Transform coupling matrix into zero-sum gauge

def \_zero\_sum\_gauge(J\_ij, inplace=False):

    L, L2, num\_symbols, num\_symbols2 = J\_ij.shape

    assert L == L2 and num\_symbols == num\_symbols2

    if inplace:

        J\_ij\_0 = J\_ij

    else:

        J\_ij\_0 = np.zeros((L, L, num\_symbols, num\_symbols))

    # go through all pairs of positions

    for i in range(L - 1):

        for j in range(i + 1, L):

            ij\_mat = J\_ij[i, j]

            # calculate matrix, row and column averages

            avg\_ab = np.mean(ij\_mat)

            # can't use axis argument of np.mean in numba,

            # so have to calculate rows/cols manually

            avg\_a = np.zeros(num\_symbols)

            avg\_b = np.zeros(num\_symbols)

            ij\_mat\_T = ij\_mat.T

            for k in range(num\_symbols):

                avg\_a[k] = np.mean(ij\_mat[k])

                avg\_b[k] = np.mean(ij\_mat\_T[k])

            # subtract correction terms from each entry

            for a in range(num\_symbols):

                for b in range(num\_symbols):

                    J\_ij\_0[i, j, a, b] = (

                        ij\_mat[a, b] - avg\_a[a] - avg\_b[b] + avg\_ab

                    )

                    J\_ij\_0[j, i, b, a] = J\_ij\_0[i, j, a, b]

    return J\_ij\_0

* **Couplings Model**: store parameters of pairwise undirected graphical model of sequences and compute evolutionary couplings, sequence statistical energies, etc
* def \_\_init\_\_(self, model\_file, precision="float32", file\_format="plmc\_v2", \*\*kwargs):
* is\_file\_obj = hasattr(model\_file, "read")
* if file\_format == "plmc\_v2":
* if is\_file\_obj:
* self.\_\_read\_plmc\_v2(model\_file, precision)
* else:
* with open(model\_file, "rb") as f:
* self.\_\_read\_plmc\_v2(f, precision)
* elif file\_format == "plmc\_v1":
* if is\_file\_obj:
* self.\_\_read\_plmc\_v1(
* model\_file, precision, kwargs.get("alphabet", None)
* )
* else:
* with open(model\_file, "rb") as f:
* self.\_\_read\_plmc\_v1(
* f, precision, kwargs.get("alphabet", None)
* )
* else:
* raise ValueError(
* "Illegal file format {}, valid options are:"
* "plmc\_v2, plmc\_v1".format(
* file\_format
* )
* )
* self.alphabet\_map = {s: i for i, s in enumerate(self.alphabet)}
* # in non-gap mode, focus sequence is still coded with a gap character,
* # but gap is not part of model alphabet anymore; so if mapping crashes
* # that means there is a non-alphabet character in sequence array
* # and therefore there is no focus sequence.
* try:
* self.target\_seq\_mapped = np.array([self.alphabet\_map[x] for x in self.target\_seq])
* self.has\_target\_seq = (np.sum(self.target\_seq\_mapped) > 0)
* except KeyError:
* self.target\_seq\_mapped = np.zeros((self.L), dtype=np.int32)
* self.has\_target\_seq = False
* self.\_reset\_precomputed()

**Chapter 7:**

**Conclusion and Future work**

**7.1 Project Conclusion:**

PTD has several advantages over them:

* silico approaches used in protein structure prediction and in drug discovery research
* Computational methods used in the search for inhibitors play an essential role in the process of discovering new drugs.
* The application of protein modelling methods has contributed significantly in cases where the structure of the target protein has not been solved, allowing the SBVS process be completed.
* Good results obtained by virtual screening depend on the quality of structures, databases to be scanned, the search algorithms, and scoring functions.

Therefore, there must be a good interaction and exchange of information between in silico and experimental methods.

**7.2 Future Work:**

Since the project’s aim is to solve the protein structure prediction and its inhibitor process problems, so we will keep on developing it and continue in improving its services to provide all the possible features with the maximum performance to make this process as easy as possible.

Since we are using **RAD phased methodology** we divided our project into **4 versions:**

1. Homology modelling
2. Threading modelling
3. Templet free modelling
4. Protein inhibitor

The first 3 versions are already done so:

* We will implement the fourth version.
* Improving algorithms and techniques that are used to get a prediction closer to the optimal one.

**References**

[1] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lip‐

man, D. (1997). Gapped BLAST and PSI-BLAST: A New Generation of Protein Data‐

base Search Programs. Nucleic Acids Research, 25(17), September, 3389-3402,

1362-4962.

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic Local

Alignment Search Tool. Journal of Molecular Biology, 215(3), October, 403-410,

0022-2836.

[3] Arnold, K., Bordoli, L., Kopp, J., & Schwede, T. (2006). The SWISS-MODEL Work‐

space: a Web-Based Environment for Protein Structure Homology Modelling. Bioin‐

formatics, 22(2), January 2005, 195-201, 1460-2059.

[4] Bowie, J. U., Lüthy, R., & Eisemberg, D. (1991). A Method to Identify Protein Sequen‐

ces that Fold into a Known Three-Dimensional Structure. Science, 253(5016), July,

164-170, 0036-8075.

[5] Brady, G., & Stouten, P. (2000). Fast Prediction and Visualization of Protein Binding

Pockets with PASS. Journal of Computer-Aided Molecular Design, 14(4), May, 383-401,

1573-4951.

[6] Colovos, C., & Yeates, T. O. (1993). Verification of Protein Structures: Patterns of

Nonbonded Atomic Interactions. Protein Science, 12(9), September, 1511-1519,

0036-8075.

[7] Deane, C. M., & Blundell, T. L. (2001). CODA: A Combined Algorithm for Predicting

the Structurally Variable Regions of Protein Models. Protein Science, 10(3), March,

599-612, 0146-9896 X.

[8] Ebalunode, J. O., Zheng, W., & Tropsha, A. (2011). Application of QSAR and Shape

Pharmacophore Modeling Approaches for Targeted Chemical Library Design. Meth‐

ods in Molecular Biology, 685, 111-133, 1064-3745.

[9] Eswar, N., Marti-Renom, M. A., Webb, B., Madhusudhan, M. S., Eramian, D., Shen,

M., Pieper, U., & Sali, A. (2007). Comparative Protein Structure Modelling With

MODELLER. Current Protocols in Bioinformatics, 50, (November), unit 2.9.1-2.9.31,

1934-340X.

[10] Friesner, R. A., Banks, J. L., Murphy, R. B., Halgren, T. A., Klicic, J. J., Mainz, D. T.,

Repasky, M. P., Knoll, E. H., Shaw, D. E., Shelley, M., Perry, J. K., Francis, P., & Shen‐

kin, P. S. (2004). Glide: A New Approach for Rapid, Accurate Docking and Scoring. 1.

Method and Assessment of Docking Accuracy. Journal of Medical Chemistry, 47(7),

March, 1739-1749, 1520-4804.

[11] Greene, N., Judson, P., Langowski, J., & Marchant, C. A. (1999). Knowledge-based ex‐

pert Systems for Toxicity and Metabolism Prediction: DEREK, StAR and METEOR.

SAR QSAR Environmental Research, 10(2-3), 299-313, 0013-9351.

Computer-Based Methods of Inhibitor Prediction

http://dx.doi.org/10.5772/ 52334

[12] Huang, B., & Schroeder, M. (2006). LIGSITEcsc: Predictiong Ligand Binding Sites us‐

ing the Connolly Surface and Degree of Conservation. BMC Structural Biology, 6, Sep‐

tember, 19, 1472-6807.

[13] Huang, B. (2009). MetaPocket: A Meta Approach to Improve Protein Ligand Binding

Site Prediction. OMICS: A Journal of Integrative Biology, 13(4), August, 325-330,

1557-8100.

[14] Jiang, X., Chena, L., Xua, J., & Yanga, Q. (2010). Molecular Mechanism Analysis of

Gloydius Shedaoensis Venom Gloshedobin. International Journal of Biological Macro‐

molecules, 48(1), January, 129-133, 0141-8130.

[15] Jones, D. T., Taylor, W. R., & Thornton, J. M. (1992). A New approach to Protein Fold

Recognition. Nature, July, 358, 86-96, 0028-0836.

[16] Jones, G., Willett, P., Glen, R. C., Leach, A. R., & Taylor, R. (1997). Development and

Validation of a Genetic Algorithm for Flexible Docking. Journal of Molecular Biology,

267(6381), July, 727-748, 0022-2836.

[17] Kastenholz, M. A., Pastor, M., Cruciani, G., Haaksma, E. E. J., & Fox, T. (2000). GRID/

CPCA: A New Computational Tool to Design Selective Ligands. Journal of Medical

Chemistry, 43(16), August, 3033-3044, 1520-4804.

[18] Kelley, L. A., & Stemberg, J. E. (2009). Protein Structure Predicition on the Web: a

Case Study using the Phyre Server. Nature Protocols, 4(3), February, 363-371,

1754-2189.

[19] Klepeis, J. L., & Floudas, C. A. (2003). ASTRO-FOLD: A Combinatorial and Global

Optimization Framework for Ab Initio prediction of Three-Dimensional Structures of

Proteins from the Amino Acid Sequence. Biophysical Journal, 85(4), October,

2119-2146, 0006-3495.

[20] Krivov, G. G., Shapovalov, M. V., & Dunbrack, R. L. (2009). Improved Prediction of

Protein Side-Chain Conformations with SCWRL4. Proteins, 77(4), December, 778-795,

1097-0134.

[21] Larkin, M. A., Blackshields, G., Brown, N. P., Chenna, R., Mc Gettigan, P. A., Mc Wil‐

liam, H., Valentin, F., Wallace, I. M., Wilm, A., Lopez, R., Thompson, J. D., Gibson, T.

J., & Higgins, D. G. (2007). Clustal W and Clustal X Version 2.0. Bioinformatics, 23(21),

November, 2947-2948, 1460-2059.

[22] Laskowiski, R. (1995). SURFNET: a Program for Visualizing Molecular Surfaces, Cav‐

ities and Intermolecular Interactions. Journal of Molecular Graphics, 13(5), October,

323-330, 0263-7855.

[23] Laskowski, R. A., Macarthur, M. W., Moss, D. S., & Thornton, J. M. (1993). PRO‐

CHECK: a Program to Check the Stereochemical Quality of Protein Structures. Jour‐

nal of Applied Crystallography, 26(2), April, 283-291, 1600-5767.

[24] Laurie, A., & Jackson, R. (2005). Q-SiteFinder: an Energy-based Method for the Pre‐

diction of Protein-Ligand Binding Sites. Bioinformatics, 21(9), May, 1908-1916,

1046-2059.

[25] Lipinski, C. A., Lombardo, F., Dominy, B. W., & Feeney, P. J. (2001). Experimental

and Computational Approaches to Estimate Solubility and Permeability in Drug Dis‐

covery and Development Settings. Advanced Drug Delivery Reviews, 46(1-3), March,

3-26, 0016-9409 X.

[26] Lüthy, R., Bowie, J. U., & Eisemberg, D. (1992). Assessment of Protein Models with

Three-Dimensional Profiles. Nature, 356(6364), March, 83-85, 0028-0836.

[27] Mc Guffin, L. J. (2008). The ModFOLD Server for the Quality Assessment of Protein

Structural Models. Bioinformatics, 24, 586-587, 1460-2059.

[28] Milleer, R. T., Jones, D. T., & Thornton, J. M. (1996). Protein Fold Recognition by Se‐

quence Threading: Tools and Assessment Techniques. The FASEB Journal, 10(1), Janu‐

ary, 171-178, 1530-6860.

[29] Morris, G. M., Huey, R., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S., &

Olson, A. J. (2004). AutoDock4 and AutoDockTools4: Automated Docking with Se‐

lective Receptor Flexibility. Journal of Computational Chemistry, 30(16), December,2009,

2785-2791, 0109-6987 X.

[30] Pontius, J., Richelle, J., & Wodak, S. J. (1996). Deviations from Standard Atomic Vol‐

umes as a Quality Measure of Protein Crystal Structures. Journal of Molecular Biology,

264(1), November, 121-126, 0022-2836.

[31] Ramachandran, G. N., Ramakrishnan, C., & Sasisekharan, V. (1963). Stereochemistry

of Polypeptide Chain Configurations. Journal of Molecular Biology, 7, July, 95-99,

0022-2836.

[32] Rohl, C. A., Strauss, C. E., Misura, K. M. S., & Baker, D. (2004). Protein Sructure Pre‐

diction using Rosetta. Methods Enzymol, 383, 66-93, 0076-6879.

[33] Sali, A. E., & Blundell, T. L. (1993). Comparative Protein Modelling by Satisfaction of

Spatial Restraints. Journal of Molecular Biology, 234, 779-815, 0022-2836.

[34] Söding, J., Biegert, A., & Lupas, A. N. (2005). The HHpred Interactive Server for Pro‐

tein Homology Detection and Structure Prediction. Nucleic Acids Research, 33(3), De‐

cember, W244-W248, 1362-4962.

[35] Subramani, A., Wei, Y., & Floudas, C. A. (2012). ASTRO-FOLD 2.0: An Enhanced

Framework for Protein Structure Prediction. American Institute of Chemical Engineers

Journal, 58(5), May, 1619-1637, 1547-5905.

[36] Sunitha, K., Hemshekhar, M., Gaonkar, S. L., Santhosh, M. S., Kumar, M. S., Basappa

Priya, B. S., Kemparaju, K., Rangappa, K. S., Swamy, S. N., & Girish, K. S. (2011).

Neutralization of Hanemorrhagic Activity of Viper Venoms by 1-(3-Dimethylamino‐

propyl)-1-(4-Fluorophenyl)-3-Oxo-1, 3-Dihydroisobenzofuran-5-Carbonitrile. Basic &

Clinical Pharmacology & Toxicology, 109(4), October, 292-299, 1742-7843.

[37] Sussman, J. L., Lin, D., Jiang, J., Manning, N. O., Prilusky, J., Ritter, O., & Abola, E. E.

(1998). Protein data bank (PDB): a Database of 3D Structural Information of Biologi‐

cal Macromolecules. Acta Crystal, D54, 1078-1084, 1600-5759.

[38] Utsintong, M., Talley, T. T., Taylor, P. W., Olson, A. J., & Vajragupta, O. (2009). Virtu‐

al Screening Against α-Cobratoxin. Journal of Biomolecular Screening, 14(9), October,

1109-1118, 1087-0571.

[39] Vriend, G. (1990). WHAT IF: A Molecular Modelling and Drug Design Program.

Journal of Molecular Graphics, 8(1), March, 52-56, 0263-7855.

[40] Yang, U. S. (2010). Pharmacophore Modeling and Applications in Drug Discovery:

Challenges and Recent Advances. Drug Discovery Today, 15(11-12), June, 446-450,

1359-6446.

[41] Peng , J., & Xu, J. (2010). Low-homology protein threading. Bioinformatics, 26, i294-

i300, 10-1093.

[42] Peng , J., & Xu, J. (2011). RaptorX: Exploiting Structure Information for protein align‐

ment by statistical inferenc. Proteins: Structure, Functon, and Bioinformatics, 79(S10),

167-171, 10-1002

[43] Anfnsen CB (1973) Science 181(4096):223

[44] Martìnez L (2014) J Chem Educ 91(11):1918. https://doi.org/10.1021/ed300302h.

[45] Levinthal C (1969) Mossbauer spectroscopy in biological systems 67:22

[46] Hooft RW, Sander C, Vriend G (1997) Bioinformatics 13(4):425

[47] Hollingsworth SA, Karplus PA (2010) Biomol Concepts 1(3–4):271

[48] https://www.uniprot.org/statistics/TrEMBL/, Accessed: 2021–02–03

[49] https://www.ebi.ac.uk/uniprot/TrEMBLstats/, Accessed: 2021–02–03

[50] https://www.rcsb.org/stats/summary/, Accessed: 2021–03–31

[51] https://predictioncenter.org/, Accessed : 2020–12–12

[52] Kryshtafovych A, Schwede T, Topf M, Fidelis K, Moult J (2019) Proteins: Structure. Function, and Bioinformatics 87(12):1011

[53] Senior AW, Evans R, Jumper J, Kirkpatrick J, Sifre L, Green T, Qin C, Žìdek A, Nelson AW, Bridgland A et al (2020) Nature 577(7792):706

[54]. Senior AW, Evans R, Jumper J, Kirkpatrick J, Sifre L, Green T, Qin C, Žìdek A, Nelson AW, Bridgland A et al (2019) Proteins: Structure. Function, and Bioinformatics 87(12):1141

[55] R.E.e. John Jumper, in In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book), 30 November - 4 December 2020 (2020)

[56] AlQuraishi M (2019) Cell Syst 8(4):292

[57] J. Ingraham, A.J. Riesselman, C. Sander, D.S. Marks, in ICLR (2019)

[58] Anfnsen CB (1971) Les Prix Nobel en 1972:103–119

[59] Wei GW (2019) Nature Machine Intelligence 1(8):336

[60] A. Fiser, in Computational biology (Springer, 2010), pp. 73–94

[61] Lam SD, Das S, Sillitoe I, Orengo C (2017) Acta Crystallographica Section D: Structural Biology73(8):628