Answers2

Answer 1

```
class Solution:
    def contains_duplicate(self, nums: List[int]) -> bool:
        seen = set()
        for num in nums:
            if num in seen:
                return True
            seen.add(num)
        return False
```

Answer 2

Script `list_ec2_instances.go`

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/ec2"
    "github.com/aws/aws-sdk-go-v2/service/ec2/types"
)

func main() {
    // Load the AWS SDK configuration
    cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-
east-1"))
    if err != nil {
        log.Fatalf("Unable to load SDK config, %v", err)
    }

    // Create an EC2 client
    client := ec2.NewFromConfig(cfg)

    // Create input parameters
    input := &ec2.DescribeInstancesInput{
        Filters: []types.Filter{
            {
                Name:   new(string),
                Values: []string{"running"},
            },
        },
    }
    *input.Filters[0].Name = "instance-state-name"

    // Call DescribeInstances
    result, err := client.DescribeInstances(context.TODO(), input)
```

```
    if err != nil {
        log.Fatalf("Unable to describe instances, %v", err)
    }

    // Process the results
    for _, reservation := range result.Reservations {
        for _, instance := range reservation.Instances {
            fmt.Printf("Instance ID: %s\n", *instance.InstanceId)
            fmt.Printf("Instance Type: %s\n", string(instance.InstanceType))
            if instance.PublicIpAddress != nil {
                fmt.Printf("Public IP: %s\n", *instance.PublicIpAddress)
            } else {
                fmt.Println("Public IP: N/A")
            }
            fmt.Println("------------------------")
        }
    }
}
```

Run with `go run list_ec2_instances.go`

Answer 3

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "time"
)

const (
    baseURL = "https://api.cosmos.network"
)

// BlockInfo represents the structure of a block's information
type BlockInfo struct {
    Header struct {
        Height    string    `json:"height"`
        Time      time.Time `json:"time"`
        LastBlock struct {
            Hash string `json:"hash"`
        } `json:"last_block_id"`
    } `json:"header"`
    Data struct {
        Txs []string `json:"txs"`
    } `json:"data"`
}

// Transaction represents a simple transaction structure
type Transaction struct {
    From   string `json:"from"`
    To     string `json:"to"`
    Amount string `json:"amount"`
```

```
    Memo   string `json:"memo"`
}

// TxResponse represents the response after sending a transaction
type TxResponse struct {
    TxHash string `json:"txhash"`
    Code   int    `json:"code"`
    RawLog string `json:"raw_log"`
}

func getBlockInfo(height int) (BlockInfo, error) {
    var blockInfo BlockInfo
    url := fmt.Sprintf("%s/blocks/%d", baseURL, height)

    resp, err := http.Get(url)
    if err != nil {
        return blockInfo, fmt.Errorf("error making GET request: %v", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return blockInfo, fmt.Errorf("unexpected status code: %d",
resp.StatusCode)
    }

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return blockInfo, fmt.Errorf("error reading response body: %v", err)
    }

    err = json.Unmarshal(body, &blockInfo)
    if err != nil {
        return blockInfo, fmt.Errorf("error unmarshaling JSON: %v", err)
    }

    return blockInfo, nil
}

func sendTransaction(tx Transaction) (TxResponse, error) {
    var txResponse TxResponse
    url := fmt.Sprintf("%s/txs", baseURL)

    jsonData, err := json.Marshal(tx)
    if err != nil {
        return txResponse, fmt.Errorf("error marshaling JSON: %v", err)
    }

    resp, err := http.Post(url, "application/json", bytes.NewBuffer(jsonData))
    if err != nil {
        return txResponse, fmt.Errorf("error making POST request: %v", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return txResponse, fmt.Errorf("unexpected status code: %d",
resp.StatusCode)
    }
```

```
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return txResponse, fmt.Errorf("error reading response body: %v", err)
    }

    err = json.Unmarshal(body, &txResponse)
    if err != nil {
        return txResponse, fmt.Errorf("error unmarshaling JSON: %v", err)
    }

    return txResponse, nil
}

func main() {
    // Example usage of getBlockInfo
    blockHeight := 1000000
    blockInfo, err := getBlockInfo(blockHeight)
    if err != nil {
        fmt.Printf("Error getting block info: %v\n", err)
    } else {
        fmt.Printf("Block Height: %s\n", blockInfo.Header.Height)
        fmt.Printf("Block Time: %s\n", blockInfo.Header.Time)
        fmt.Printf("Number of Transactions: %d\n", len(blockInfo.Data.Txs))
    }

    // Example usage of sendTransaction
    tx := Transaction{
        From:   "cosmos10ju7r4vcsgtc6tprra3s2qekj543a7r8ry64wm",
        To:     "cosmos1ehd68nseclwwv0zd4vrpuxf6vu9d0cjsxeae0j",
        Amount: "10000stake" # or something like "10atom",
        Memo:   "Test transaction",
    }

    txResponse, err := sendTransaction(tx)
    if err != nil {
        fmt.Printf("Error sending transaction: %v\n", err)
    } else {
        fmt.Printf("Transaction Hash: %s\n", txResponse.TxHash)
        fmt.Printf("Transaction Code: %d\n", txResponse.Code)
        fmt.Printf("Transaction Log: %s\n", txResponse.RawLog)
    }
}
```

This solution addresses the main requirements of the interview question:

1. It implements two main functions: `getBlockInfo` and `sendTransaction`.
2. The `getBlockInfo` function performs a GET request to retrieve information about a specific block.
3. The `sendTransaction` function performs a POST request to send a simple transaction.
4. Error handling is implemented throughout the code.
5. The code follows Go conventions and best practices.

Key points about the implementation:

1. Structs are defined to represent the block info, transaction, and transaction response.
2. The `http` package is used to make GET and POST requests.
3. JSON marshaling and unmarshaling are used to handle API request and response data.

4. The `main` function demonstrates the usage of both `getBlockInfo` and `sendTransaction`.
5. Error messages are descriptive to aid in debugging.

Areas for improvement or expansion:

1. Implement a command-line interface for user input.
2. Add unit tests for the functions.
3. Implement rate limiting to prevent API abuse.
4. Use a more robust HTTP client with timeouts and retries.
5. Implement proper logging instead of just printing to stdout.