Markov::Model< char >
- std::map< char, Node < char > * > nodes
- Node< char > * starterNode
- std::vector< Edge< char > * > edges
+ Model()
+ char * RandomWalk(Markov ::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)
+ void AdjustEdge(const char *payload, long int occurrence)
+ bool Import(std::ifstream *)
+ bool Import(const char *filename)
+ bool Export(std::ofstream *)
+ bool Export(const char *filename)
+ Node< char > * StarterNode()
+ std::vector< Edge< char > * > * Edges()
+ std::map< char, Node < char > * > * Nodes()
+ void OptimizeEdgeOrder()
Markov::API::MarkovPasswords
- std::ifstream * datasetFile
- std::ofstream * modelSavefile
- std::ofstream * outputFile
+ MarkovPasswords()
+ MarkovPasswords(const char *filename)
+ std::ifstream * OpenDataset File(const char *filename)
+ void Train(const char *datasetFileName, char delimiter, int threads)
+ std::ofstream * Save (const char *filename)
+ void Generate(unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
+ void Buff(const char *str, double multiplier, bool bDontAdjustSelfLoops =true, bool bDontAdjustExtendedLoops=false)
- void TrainThread(Markov ::API::Concurrency::ThreadShared ListHandler *listhandler, char delimiter)
- void GenerateThread (std::mutex *outputLock, unsigned long int n, std ::ofstream *wordlist, int minLen, int maxLen)
Markov::API::ModelMatrix
char ** edgeMatrix
long int ** valueMatrix
int matrixSize
char * matrixIndex
long int * totalEdgeWeights
bool ready
+ ModelMatrix()
+ bool ConstructMatrix()
+ void DumpJSON()
+ int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen =12, int threads=20, bool bFileIO=true)
+ void Import(const char *filename)
+ void Train(const char *datasetFileName, char delimiter, int threads)
# int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen =12, int threads=20, bool bFileIO=true)
# void FastRandomWalkPartition (std::mutex *mlock, std:: ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)
# void FastRandomWalkThread (std::mutex *mlock, std ::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)
# bool DeallocateMatrix()
Markov::API::CUDA:: CUDADeviceController
+ static __host__ void ListCudaDevices()
# static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true)
# static __host__ cudaError _t CudaMalloc2DToFlat(T **dst, int row, int col)
# static __host__ cudaError t CudaMemcpy2DToFlat(T *dst, T **src, int row, int col)
# static __host__ cudaError _t CudaMigrate2DFlat(T **dst, T **src, int row, int col)
Markov::API::CUDA:: CUDAModelMatrix
- char * device_edgeMatrix
- long int * device_valueMatrix
- char * device_matrixIndex
- long int * device_totalEdge Weights
- char ** device_outputBuffer
- char ** outputBuffer
- char * flatEdgeMatrix
- long int * flatValueMatrix
- int cudaBlocks
- int cudaThreads
- int iterationsPerKernelThread
- long int totalOutputPerSync
- long int totalOutputPerKernel
- int numberOfPartitions
- int cudaGridSize
- int cudaMemPerGrid
- long int cudaPerKernelAllocationSize
- int alternatingKernels
- unsigned long ** device _seeds
- cudaStream_t * cudastreams
+ __host__ void MigrateMatrix()
+ __host__ void FlattenMatrix()
+ __host__ void FastRandom Walk(unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)
# __host__ char * AllocVRAMOutput Buffer(long int n, long int singleGenMaxLen, long int CUDAKernel GridSize, long int sizePerGrid)
# __host__ void LaunchAsync Kernel(int kernelID, int minLen, int maxLen)
# __host__ void prepKernel MemoryChannel(int numberOfStreams)
# __host__ void GatherAsync KernelOutput(int kernelID, bool bFileIO, std::ofstream &wordlist)
Python.CudaMarkopy.CudaModel MatrixCLI
+ model
+ bInfinite
+ def __init__(self)
+ def add_arguments(self)
+ def init_post_arguments (self)
- def _generate(self, str wordlist)
Python.CudaMarkopy.CudaMarkopyCLI
+ args
+ cli
+ None __init__(self)
+ def help(self)
+ def parse(self)
+ def parse_fail(self)