Gestión de memoria

- La memoria la gestiona el sistema operativo
- Sistema operativo es la capa de software que está justo arriba del hardware
    - Dos funciones:
        - Provee un API para abstraer los recursos, brinda funcionalidades de recursos que existen.
        - Maneja todos los recursos que ofrece la computadora.
    - Gestiona:
        - Se encarga de gestionar todos los procesos.
        - Gestiona la memoria
        - Gestiona archivos y el disco duro
        - Gestiona Inputs y outputs
- ¿Qué es gestión de la memoria?
- ¿Qué involucra gestión de la memoria?
- ¿Qué hace el sistema operativo para gestionarla?

Formas de gestionar la memoria

- Es la multiprogramación la que ha ido cambiando la manera en la que se gestiona la memoria.
- Van desde no gestionar la memoria hasta **memoria virtual**
    - Antes la memoria no necesitaba ser gestionada porque las computadoras solo realizaban una tarea a la vez.
- La parte del sistema operativo que gestiona la memoria es el gestor de memoria.

Sin separación de la memoria física

- Problemas:
    - Un problema de este enfoque es el hecho de que un programa le puede caer encima al sistema operativo(Se soluciona con *Static relocation*)
        - Static relocation:
            - **Cuando se carga** un programa en memoria a todas las direcciones donde está el programa se suma la dirección de inicio del programa.
              Es poco eficiente
                - **Accesos al disco duro**
                - **Recalcular todas las direcciones**
    - Solo puede haber un programa a la vez(Se soluciona con *swapping*)
        - Swapping:
            - Cuando se ocupa tener otro programa en memoria, se lleva al disco el actual y se carga el nuevo.

Bloques de memoria

- Registros base y límite
    - Base: donde inicia el programa
    - Límite: cantidad de instrucciones
- Dynamic relocation
    - Suma el registro base **instrucción por instrucción**, bajo demanda, más eficiente
- Inicio de multiprogramación

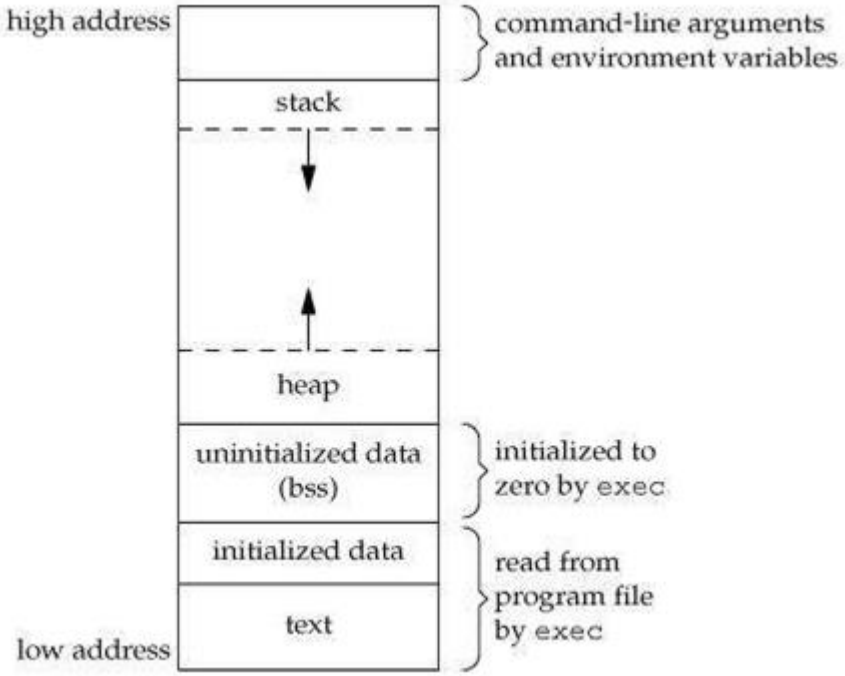
Memoria virtual

- Sobrecarga de memoria: Cuando quiero correr un programa que es más grande que mi memoria física.
- En este tipo de abstracción cada programa está dividido en páginas de tamaño fijo, no importa el programa cualquier página es de tamaño fijo.
- Cada **página** es un rango pequeño de direcciones de memoria
- El MMu convierte direcciones virtuales en físicas (Quiz)
- Si la página no está cargada se da un *page fault*
    - Page fault:
        - La página solicitada no está cargada
        - Se aplica un algoritmo de reemplazo

Manejo de memoria desde el nivel del programa

- Un programa en ejecución tiene un *memory layout*
    - *Memory layout:*

- Cómo está diseñada la memoria cuando estoy diseñando un programa.



- Solo hay una copia de todos los programas en ejecución
    - Esto significa que si hay dos instancias del mismo programa igualmente solo existe una sola copia del mismo en memoria.
- Dynamic binding: En tiempo de ejecución se asigna a cada instancia(se refiere a diferentes ventanas) del programa la parte del código que se necesita.
- Stack y heap crecen en sentido contrario, si se cruzan, segmentation fault

Stack

- Sección en el memory layout
- Trabaja como LIFO
- Compuesto por un conjunto de stack frames. cada frame es una llamada a función
- Cada vez que una función es llamada un stack frame es insertado en la pila, cada vez que se da un return se libera el stack frame asociado a la llamada.
- Cada stack frame está compuesto por:
    - Espacio para variables automáticas para la nueva función
    - Línea de la función a la que se está llamando
    - Argumentos o parámetros de las funciones
- Maneja la memoria, transparente al programador
- Se interactúa mediante llamadas a funciones, indirectamente
- Los posibles scope de una variable son:
    - Local: en una sola función, clase, etc...
    - Global: todo el programa

Heap

- Es la sección de un proceso en memory layout
- No se gestiona automáticamente
- Se interactúa con él, asignando memoria, liberando y cambiándola
- Se puede evitar el uso del heap con ciertos lenguajes

Cuándo usar cuál

- Stack:
    - Para variables pequeñas
    - Enfoques pequeños
- Heap:
    - Cuando se necesite asignar bloques de memoria largos
    - Crecimiento dinámico de una estructura
    - Variables globales o que se vayan a utilizar mucho

Ventajas y desventajas del stack

- Ventajas
    - Eficiente en tiempo y espacio
    - Las copias locales evitan problemas colaterales
    - Conveniente ya que temporalmente es independiente de la memoria
- Desventajas:
    - Poco tiempo de vida:
        - Solo existen en cierto enfoque
    - Comunicación restringida

Ventajas y desventajas del heap

- Ventajas:
    - El programador controla cuando la memoria es desocupada y ocupada
    - Es posible construir estructuras de datos y retornarlas al llamador
    - La memoria en la cual está alojada puede ser controlada por nosotros
- Desventajas:
    - Más trabajo: responsabilidad del programador manejar la memoria
    - Más errores

Punteros

- Almacenan direcciones de memoria
- Siempre tienen el mismo tamaño
- El espacio que toma un puntero es un integer
- “Apuntan” hacia otra dirección de memoria

- *:
    - Obtiene el valor **apuntado** por el puntero
- **:
    - Es un puntero de otro puntero
- Por default normalmente es inicializada por un bad value, o un random address o nullPointer.
- El puntero siempre debe ser inicializado, desreferenciar un puntero tirando a nulo tira un error en tiempo de ejecución
- Varias maneras para darle una variable al puntero:
    - & antes de la variable devuelve la dirección en memoria donde está almacenada **la variable**
    - Operador * va a la izquierda del puntero variable y da el valor del lugar a donde está apuntando el puntero, a **esto se le llama desreferenciar un puntero.**
    - El puntero sin ningún operador devuelve la dirección de memoria que tiene **el mismo puntero.**

Sharing

- Se puede tener más de un puntero referenciando una misma dirección de memoria
- Shallow copy:
    - Copia las referencias, no los datos
    - Dos Punteros referencian el mismo dato
- Deep copy:
    - Copiar la data, no la referencia
    - Se necesitan:
        - memcpy, strcpy, o similar, por tanto se paga un costo de tiempo de ejecución

Arreglos y aritmética

- Declarar un array es alojar bloques de memoria contiguos

Referencias

- Casi lo mismo que un pointer, es más general que un puntero
- Son usadas en el contexto de paso de parámetros
- C++ Soporta referencias

Pasar parámetros por valor

- El método llamado, no ve los cambios que hace al que estoy llamando.
- Al que estoy llamando recibe una copia independiente del parámetro.
- Posiciones de memoria distintos
- Se puede decir que la copia tiene un scope local

Pasar parámetros por referencia

- El método que está haciendo la llamada pasa la referencia
- El método que está recibiendo la llamada recibe una referencia y no un valor
- Por tanto si cambia algo dentro del método que recibe la referencia también se cambia fuera del mismo

Heap en C y C++

- En C:
    - malloc y free
- En C++:
    - New() y delete()

Manejo de memoria en Java

- Existen punteros pero se manejan de manera indirecta
- Características principales:
    - Propenso a menos errores, java gestiona la memoria por nosotros
    - El programador se puede dedicar a cosas más productivas.
    - Es más lento, el lenguaje hace todo el trabajo de manejo de memoria.
- Garbage collector:
    - Hace la liberación del espacio por el programador
    - Libera los espacios de memoria no referenciados
    - Mantiene una lista de todas las referencias al heap, al no estar referenciada una celda del heap, se libera.
    - Dos fases:
        - Marcado
            - Recorre toda la memoria identificando si hay celdas que están referenciadas
        - Reclamación:
            - Las celdas no marcadas son retornadas al pool de memoria disponible y se ejecuta el proceso de compactación
                - Compactación:
                    - Juntar celdas libres.

Análisis de algoritmos

- Algoritmo:
    - Serie de pasos finitos bien definidos para resolver un problema
    - Sin ambigüedad, efectivos, finitos y precisos.
    - Características:
        - Finito: Se termina de ejecutar después de una cierta cantidad de pasos
        - Definiteness: Tiene que ser bien definidos
        - Existe un input, un output y es efectivo
        - La eficiencia es un deseable, pero no es algo necesario, porque depende de qué parámetro le estoy evaluando
        - Determinístico o único: mismo input = mismo resultado. Sin embargo, algunas veces puede que no se cumpla.
- Maneras de representar un algoritmo:
    - Pseudocódigo
    - Lenguaje natural, muy ambiguo
    - Diagramas
    - Lenguaje formal, 0% ambiguo pero complejo
- ¿Qué es análisis de algoritmos?
    - Metodología para estimar el consumo de recursos
    - Se analizan los algoritmos para evaluar si se puede aplicar en otros contextos
        - También para clasificar problemas de acuerdo a su dificultad, compararlos y refinarlos (optimizarlos)
- Factores:
    - Developer
    - Size of input
    - Compilador
    - Sistema Operativo
    - Lenguaje
    - Sistema (CPU, Memoria, DIsco)
    - Los programas corriendo en un computador
    - Todo alv
- Características de interés (para evaluar algoritmo):
    - Espacio
    - Tiempo
    - Cualquier otro recurso que utilice del computador (external devices, etc)
- Tres modelos para medir algoritmos:
    - Empírico
        - Benchmarks
        - Problema:
            - Muy easy
    - Simulacional
        - Datos simulador
        - Casos de prueba
        - Problema:

                - No se puede simular totalmente todo, no hay tiempo
        - Analítico
            - Modelos matemáticos
                - Es el más exacto pero el más complejo.
- Complejidad computacional
    - Clasifica los problemas de acuerdo a su dificultad
    - Dos tipos:
        - De tiempo y espacio
    - Complejidad del tiempo (enfoque del curso)
        - Cantidad de tiempo que toma en ejecutarse un algoritmo
    - Complejidad del espacio

Complejidad de tiempo

- Se estima mediante el conteo de instrucciones elementales
- Hay dos tipos:
    - Tiempo real o de pared y de reloj (CPU Speed)
        - De reloj: Frecuencia a la que corre el procesador (Hz o GHz)
- Instrucciones elementales:
    - Instrucciones que siempre toman el mismo tiempo en la misma computadora
    - Le llamamos T al tiempo requerido que toma ejecutar una instrucción elemental.
    - Operaciones:
        - Aritméticas (+, -, /, %...)
        - Corrimientos (sobre bits <<, >>)
        - Operadores lógicos (==, !=, or, and, ~...)
        - Jumps (return values, method calls…)
        - Asignaciones y acceso a estructuras indexadas.
        - Declaraciones
    - En los ifs y switch se toman siempre los peores casos.
- Hay que analizar siempre el caso intermedio, el mejor y el peor.

Algoritmos de búsqueda

Búsqueda secuencial:

- Complejidad O(n) -> en el peor de los escenarios se van a hacer el número de comparaciones igual al tamaño del elemento
- Usado para buscar un elemento en una lista o vector
- La más simple

Búsqueda binaria:

- Parte el array en dos, compara con la mitad del array, si es mayor busca en la parte derecha, si es menor, en la parte izquierda.
- Ejecuta el código varias veces

Interpolación:

- Modificación de la búsqueda binaria
- Hace un cierto cálculo para elegir un mejor medio (pivote)
- O(n)
- middle = (low + ((number-array[low])*(high-low)))/(array[high] - array[low])

Hash:

- Función matemática que convierte un conjunto más grande en otro más pequeño
- Se utilizan llaves
  - Convierte llaves en índices
- Idealmente usar una función inyectiva o biyectiva
- Restas sucesivas:
  - ay :c
- Aritmética modular
  - Usar un número primo
  - Índice es el residuo de la división entre ay :c
- Midsquare
  - Eleva el valor de la llave y toma los r dígitos al medio del resultado
- Truncade
  - Ignorar parte de la llave y usar el resto como el índice del arreglo
- Folding
  - Divide en parte y combina usando operadores

Colisiones

- Una colisión es cuando dos o más llaves tienen el mismo índice
- Tabla hash pequeña y muchas llaves causan más colisiones
- Lidiar con ellos:
  - Al encontrar un buque lleno moverse hacia abajo
- Investigar algoritmos para manejar colisiones

Pathfinding

- Basado en Dijkstra
- Resuelve laberintos
- Explora los nodos adyacentes y selecciona los más cercanos

- Trabaja sobre una matriz, cada celda de la matriz es un nodo
- Este tipo de algoritmos se utiliza en juegos de estrategia
- Formas de implementarlo:
    - Sin obstáculos
        - Muevo el eje x cuanto sea necesario hasta que llegue al mismo x que el objetivo y lo mismo con el eje y
    - Random Bouncing (prohibido utilizar en proyectos)
        - Fuerza bruta básicamente
    - Object tracing
        - Rodea obstáculos
    - Breadth-First Search
        - Onda expansiva para detectar celdas alrededor
        - Ineficiente
        - Va probando con los nodos adyacentes y guardando las posiciones de los obstáculos hasta llegar a su objetivo
    - Distance-First Search
        - Le asigna prioridad según distancia horizontal/vertical (evita el uso de diagonales pero no lo prohíbe)
    - Better heuristic pathfinding
        - Busca tomar la mejor decisión, o sea, que tarde menos celdas, menos tiempo en cálculos y obtención de la ruta más corta
        - No va a encontrar la ruta más óptima, pero sí alguna de las más cortas
    - A*
        - Preciso
        - Rápido
        - Confiable
        - Usado en juegos
        - Se divide el área en celdas:
            - Red wall
            - Blue goal
            - Green start
            - id: caminado / no caminado
        - Pasos:
            - Agregar el punto de inicio a una lista abierta de elementos
            - Chequear los cuadrados en los que se puede caminar, agregar los cuadrados a la lista abierta y guardar el punto de inicio como padre de esos cuadrados
            - Quitar el punto de inicio de la lista abierta y meterlo a la lista cerrada
            - Escoger un cuadrado adyacente de la lista abierta y repetir
        - Path Scoring:
            - Close list: Nodos ya escogidos
            - Open list: Nodos por escoger
            - F = G + H, se elige el nodo (cuadrado) con el menor Fua (F)

- G: Costo estimado para moverse del punto inicial a un cuadrado dado en la cuadrícula
    - Movimiento horizontal o vertical vale 10 y el diagonal 14
    - No “importa” si hay obstáculos (no es contemplado)
    - Se redefinen los padres de los nodos
    - Va variando conforme se avanza y se le va sumando el H del nodo actual al valor F anterior (F=G+H+...+Hn)
    -
- H: Costo estimado para moverse de un cuadrado dado hasta su destino final (heurístico - Manhattan)
    - Le importan los obstáculos

UML, Patterns y calidad de software

- Buscar estándares para realizar trabajos.

Paradigma orientado a objetos

- Objeto: empaquetamiento de datos y operaciones
    - Métodos y atributos
- Entre objetos se comunican con mensajes
- Todo el estado interno de un objeto está encapsulado, no se pueden acceder los atributos directamente
- Nombre de método, parámetros y tipo de retorno es la firma de un método
- Conjunto de firmas públicas es la interfaz de un objeto
- Las clases definen la implementación del objeto
- Los objetos son creados instanciando una clase
- Cuando un objeto es instanciado se guarda memoria para un objeto y se asocian los métodos a esta memoria
- Herencia:
    - Los atributos definidos del padre no se pueden heredar
    - Los datos y operaciones privadas no se pueden heredar
- Clases abstractas
    - Una clase que define interfaces comunes entre sus subclases
    - No puede ser instanciada
    - Los métodos que no son implementados se llaman métodos abstractos
- Overwriting:
    - Sobrecargar el método de la clase padre
- Overloading:
    - Sobrecargar mis propios métodos
- Polimorfismo:
    - Tratar diferentes objetos como si fueran los mismos.
    - Dynamic binding:

        - Hay una sola copia del código en memoria, cada vez que una clase hija quiere usar ese código se va a la única copia del método y se ejecuta.
- Composición de objetos:
    - Se pueden reusar los objetos mediante:
        - Herencia
        - Composición
    - En composición se logra que varios objetos trabajen en conjunto, se crea una nueva funcionalidad a partir de la funcionalidad de dos objetos
- Cohesión y acoplamiento:
    - Se necesita alta cohesión (alta relación entre las clases de un mismo módulo) y bajo acoplamiento (baja dependencia entre módulos).)
        - Cohesión:
            - Que las partes de un módulo trabajen bien
        - Acoplamiento:
            - Que los módulos entre sí se relacionen lo mínimo

UML

- **SDLC:**
    - **Software development life cycle**
- Entre los años 1989 & 1994: existían 50 lenguajes de modelado orientados a objetos
- Lo crearon Grady Booch, Ivar Jacobson y James Rumbaugh
- Usamos UML para:
    - Comunicar qué comportamiento quiero que tenga mi sistema
    - Visualizar y controlar la arquitectura de mi sistema
    - Entender el sistema que construimos
    - Manejar riesgos
- Un modelo es una simplificación de la realidad (una abstracción)
- UML está compuesto por:
    - Modelo funcional
    - Modelo de objetos
    - Modelo dinámico
    - Cada modelo está compuesto por conjuntos de diagramas

Modelo funcional

- Diagrama de casos de uso:
    - Comportamiento de sistema, subsistema, clase o componente que particiona el sistema en transacciones entre actores
    - Un actor es una entidad que interactúa con el sistema (no necesariamente una persona)

Modelo de objetos

- Diagrama de clases:

    - Se conoce como vista estática
    - Una relación reflexiva se aplica cuando una clase se relaciona consigo misma
    - Agregación vs composición:
        - Agregación es débil, composición es fuerte.
- Diagrama de componentes:
    - Se definen los módulos que forman parte del sistema y sus relaciones
- Diagrama de despliegue:
    - Diagrama que indica las bibliotecas o paquetes a ser instalados

Modelo dinámico

- Diagrama de secuencias: muestra la forma en que un grupo de objetos se comunican (interactúan) entre sí para realizar procesos/transacciones que se deben cumplir secuencialmente para completar un procedimiento.

Patrón de diseño

- Regularidad discernible en el mundo en un diseño hecho por el hombre
- Solución general reusable (reutiliza comportamientos y relaciones entre objetos) de un problema **en un contexto de software específico**.
- Se dividen en:
    - Propósito:
        - Creacional
            - Abstraen el proceso de instanciación
            - Enfoques:
                - Encapsular el conocimiento de cómo el sistema utiliza las clases concretas
            - Ejemplos:
                - Builder
        - Estructural
            - Jejeps
        - Comportamiento
            - x2
    - Alcance (scope):
        - Clase y subclases
        - Objetos
            - **Se utilizan en tiempo de ejecución**

Patrones de Diseño vistos en clase:

- Facade: Se utiliza para proporcionar una interfaz, por medio de la cual permita utilizar distintos subsistemas.
- Adapter: Facilita interacción entre clases
- Observer: Existe un emisor y varios receptores, cuando algo le pasa al emisor se le comunica a todos los receptores y cada uno decide si quiere más detalle del suceso.

Algorithm design:

- Técnicas de diseño de algoritmos:
    - Divide-and-conquer
    - Backtracking:
        - Trying out various sequences of decisions, until you find one that works.
        - Se utiliza mucho en lenguajes como Prolog.
        - Las soluciones se pueden ver como un árbol.
        - Naturalmente backtracking es un algoritmo recursivo.
        - Se tienen nodos/posiciones/estados de posibles nodos y acciones, si se encuentra un fallo en la "ruta" se devuelve a al nodo padre e intenta otras alternativas.
        - Algoritmo de las 8 reinas.
    - Dynamic programming
        -
    - Greedy (voraces) algorithms
    - Probabilistic algorithms

Genetic algorithms

- Son algoritmos de búsqueda heurísticos basados en las ideas evolucionarias de selección natural y genética.
- Representa una explotación inteligente de la búsqueda aleatoria para resolver problemas.
- No es totalmente aleatoria, utiliza información histórica para dirigir la búsqueda.
- Se busca una representación genética del dominio de la solución.
- Se busca una fitness function para evaluar el dominio de la solución.
- Definiciones:
    - Cell: contiene cromosomas (cadena ADN).
    - Chromosome: conjunto de genes (bloques de ADN).
    - Genotype: colección de genes responsables de un rasgo.
    - Reproducción: combinación de genes padre.
    - Mutación: errores durante la reproducción (alteraciones).
    - Fitness: cuánto puede reproducirse antes de que muera.

- La evolución inicia con una población random, que se llama primera generación.
- En cada generación, el fitness de cada uno es calculado.
- Algunos individuos son seleccionados con base en su fitness.
- Los individuos seleccionados son combinados para obtener nuevos individuos.
- Algunos individuos son descartados (fitness más bajo).
- Se crea una nueva generación y se repite el proceso.
- Termina cuando se alcanza el número máximo de generaciones.
- No hay cambio en el material genético.
- Se alcanza una "solución"/características deseadas.
- Simple point crossover: combina la mitad de los bits de los padres (mitad y mitad)
- Inversión: selecciona una cadena de bits y aplica complemento.
- Bit vectors:
    - Tipo especializado para trabajar con arrays booleanas
- Función de fitness:
    - Se define sobre la representación genética.
    - Mide la calidad de una solución determinada.
- Población inicial:
    - Completamente random
- Selección:
    - Se selecciona, normalmente, el que tenga mejor fitness.
- Reproducción
    - Cruce
        - Parte de uno, parte de otro
    - Mutación
        - Baja probabilidad
        - Se selecciona un bit random, se invierte y se elimina el overflow
    - Inversión
        - Mucha menos probabilidad
        - Se selecciona una cadena de bits random y se le aplica inversión.
    - Por cada nueva solución se selecciona un par de padres.

Divide y vencerás

- Divida el problema en subproblemas
- Conquiste los subproblemas resolviendolos de forma recursiva.
- Combine las soluciones
- Una vez que haya un problema tan pequeño que no se puede dividir se alcanzan los casos base y son esos los que se trabajan
- Para aplicar esta técnica se requiere de la capacidad de poder utilizar recursividad
- Es altamente eficiente

- Se suelen buscar múltiples soluciones de forma paralela

Programación dinámica

- Método para resolver problemas dividiéndolos en problemas más simples, para luego sumarlos y solucionar problemas más complejos.
- Características presentes:
    - Subestructura óptima:
        - Cada solución de un subproblema es óptima
    - Overlapping de subproblemas:
        - El espacio de los subproblemas debe ser pequeño, y se tiene que resolver el mismo subproblema una y otra vez.
    - La diferencia contra divide y vencerás es que se reutilizan soluciones
    - Cada vez que soluciono un problema guardo el resultado en una tabla.
    - Cuando voy a resolver un problema nuevo reviso la tabla y si no está la solución la guardo
- Problema de la mochila
    - Tarea

Algoritmos voraces

- Consume menos recursos que dynamic
- Toma decisiones con la información que tiene en el momento
- En algunos casos no provee la mejor solución, pero sí se acerca
- Es rápido.
- La solución óptima depende de la función heurística que se defina
- Dijkstra
- Partes:
    - Conjunto de candidatos
    - Una función de selección: seleccion el mejor candidato con la información que tengo
    - Determinar si un candidato contribuye o no para la solución
    - Un objetivo (función objetivo que asigna un valor a una solución o a una solución parcial)
    - Una función que permita determinar si ya encontré una solución

Algoritmos probabilísticos

- A veces es mejor tomar una decisión random
- Mucho más rápido, menos recursos
- Menos cercano a la respuesta óptima
- Mismos inputs, resultados diferentes
- El tiempo de ejecución y los resultados varían

- Comparando con los algoritmos determinísticos, podría fallar o nunca terminar de ejecutarse (no deja de correrse).
- Categorías:
    - Numéricos:
        - Solución aproximada
        - 90% de dar una solución accurate
        - Si se ejecuta más veces se incrementa la probabilidad de que salga la correcta
    - Las Vegas:
        - Toma decisiones random
    - Monte Carlo:
        - Calcula la mejor solución con un alto grado de probabilidad
        - Si se ejecuta más veces se incrementa la probabilidad de que salga la correcta

Estructuras de datos en almacenamiento externo

- Diferencia entre memoria primaria y secundaria: la primaria posee mayor rapidez de acceso y la secundaria una mayor capacidad de almacenamiento.

Hard disk drive

- No son volátiles.
- Almacena datos en platos rígidos, que poseen superficies magnéticas, que rotan rápidamente
- HDD es una unidad sellada que contiene un número de platos en una pila.
- Posee cabezales de lectura y escritura electromagnética que se coloca sobre los platos.
- Son dispositivos de almacenamiento de acceso directo (toma casi el mismo tiempo acceder a un dato en cualquier parte del disco).
- Los platos están hechos de aluminio o substrato de “vidrio”.
- Cada plato está dividido en miles de círculos concéntricos llamados pistas, donde se almacena toda la información.
- La pista más externa (más hacia afuera) es la pista 0.
- Hay pistas en cada cara del plato.
- Cada pista está formada por miles de sectores; los sectores son la unidad básica de almacenamiento en un HDD.
- Un grupo de sectores se llama cluster (es como sacar la misma tajada de cada plato).
- Los gaps se utilizan para delimitar los sectores.
- Los cabezales crean la interfaz entre el campo electromagnético donde los datos están almacenados y los componentes electrónicos en el disco duro.
- Convierte bits en pulsos magnéticos cuando se almacenan en un plato.

- Un cilindro es un grupo que conforman las pistas de mismo número de cada plato.
- Cylinder method and sector method.
    - Dirección de registro es la dirección de sector
- En cabezas moviles Access Time = Positioning Time+ head activation time+rotational delay+transfer time
- En cabezas estáticas:
    - Una cabeza por pista
    - Access Time= Head activation+rotational time+ transfer time

Tipos de discos duros

- ATA -> PATA Cuando entró SATA
- IDE es obsoleto
- SCSI:
    - Small computer system interface
    - Estándar para conectar una computadora con un dispositivo

Arreglos de discos

- Data striping
    - Dividir lógicamente la data secuencial.
- Mirroring
    - Replicación de discos lógicos en discos físicos separados en tiempo real para asegurar la disponibilidad de la información.
    - Trabajan como si estuvieran en un mismo lugar.
- Bit de paridad o chequeo
    - Bit agregado al final de un string en un código binario que indica si el número de unos en el string es par o impar
    - La forma más simple de detección de errores.
    - Bit de paridad par:
        - 1 si el número de unos en la cadena es impar
    - Bit de paridad impar
        - 1 si el número de unos en la cadena es par

RAID

- Arreglo redundante de discos independientes
- 0:
    - Striping a nivel de bloques
    - No hay paridad
    - No hay mirroring
    - No hay redundancia
    - Performance improvement
    - Additional Storage

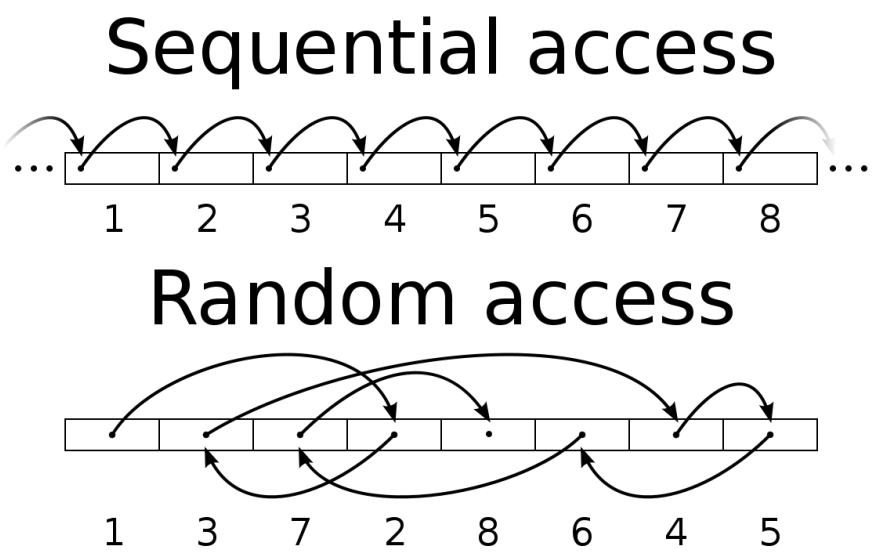
    - No hay tolerancia a fallos
        - Falla un disco y se muere el arreglo
    - Acceso en paralelo
    - Mínimo 2 discos
    - Mirroring vs redundancy:
        - Redundancy:
            - Capacidad de un sistema de reponerse a fallos
        - Mirroring:
            - Capacidad de un sistema de replicar (copia) datos en otro disco
        - Mirroring implica redundancia, al revés no.
- 1:
    - Hay mirroring
    - No hay paridad
    - No stripping
    - Los datos están escritos idénticamente en dos discos
    - Hay redundancia
- 2:
    - No es usado normalmente
    - Stripping a nivel de bit
    - Los discos rotan sincronizadamente
    - Almacenado en al menos una unidad de paridad
- 3:
    - Stripping a nivel de byte
    - Paridad dedicada
        - Un solo disco para el cálculo de paridad
    - Los discos rotan sincronizadamente
- 4:
    - Raid 5, pero toda la paridad está en un solo disco
    - Los archivos distribuidos entre los dispositivos (discos?)
    - Trabajan de forma independiente
    - I/O En paralelo
- 5:
    - Stripping a nivel de bloque
    - Paridad distribuida
    - Provee una tolerancia a fallo en al menos dos discos
    - Distribuye la paridad y los datos entre todos los dispositivos (discos?)
    - Arreglo no se destruye por el fallo de un solo disco.
    - Lectura rápida.
    - Escritura lenta, porque se recalcula la paridad
    - Se utilizan los 4 discos de forma simultánea
    - El total de espacio disponible es el número de discos menos uno por el tamaño del disco más pequeño
        - Se trabaja por filas, por eso el tamaño del array es así
        - Misma cantidad de filas y misma cantidad de columnas

    - Dentro de más discos se utilice el espacio de almacenamiento es más eficiente sin perder redundancia
    - Puede sobrevivir sin un dispositivo, pero se vuelve lento
    - Utiliza XOR para reconstruir los datos y crear la cadena de paridad
- 6..
- 10:
    - Raid 1 + 0
    - Mirroring stripping
    - Striping
    - Mínimo 4 discos
    - Excelente redundancia
    - Excelente rendimiento
    - Mejor opción para cosas críticas

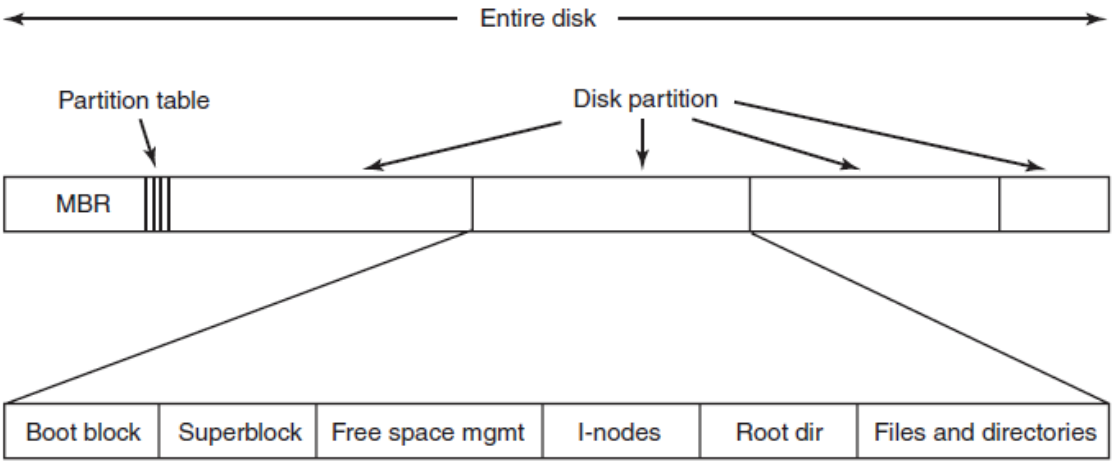
File systems

- El sistema operativo da una abstracción del hardware
- Archivos: Representación abstracta de la información en el disco
    - Unidades lógicas de información creadas por procesos
    - Información en los archivos debe ser persistente (no debe desaparecer de no sea porque el dueño de ese archivo lo desee), no ser afectado por ningún proceso de creación o terminación.
    - La parte del sistema operativo que gestiona archivos es el file system.
    - Los programadores utilizan un API para interactuar con el file system.
    - Existen varios file systems:
        - Ext2, Ext3 (Linux)
        - NTFS (Windows)
        - FAT, FAT32 (DOS, Windows)
        - etc.
    - Requisitos de un file system:
        - Persistencia
        - Velocidad
        - Tamaño (que sea óptimo)
        - Protección (personalizable)
        - Que permita compartirlo con otros usuarios
        - Fácil de usar
        - Acceso Random
    - Terminología:
        - Disco: permanent storage
        - Bloque o sector: la unidad más básica de un disco o un FS (file system).
        - Partición: subconjunto de bloques
        - Volumen: colección de bloques
        - Superbloque: donde el file system guarda la información crítica.
    - Un archivo es un mecanismo de abstracción

        - Provee un camino para guardar información
        - Oculta al usuario los detalles de como y donde la información es almacenada (que son detalles de implementación)
    - Existen tres formas de para un file system:
        - Secuencia de bytes: Unix y Windows
            - El sistema operativo no ayuda pero tampoco estorba
            - Almacena como una cadena de bytes, provee flexibilidad
            - Normalmente llamados archivos planos
        - Récord de tamaño fijo
            - Se trabaja un récord por completo, siempre. Se lee por completo aunque no sea necesario leer todas sus partes
            - Se utiliza en mainframe y midframe
        - Árbol de récord
            - Cada record tiene una llave
            - El árbol se ordena por un key field para permitir una busqueda mas rapida por keys
            - La operación básica es obtener un récord en específico
    - Tipos de archivos
        - Archivos regulares: contienen información del usuario.
        - Directorios: system files para mantener la estructura del file system.
        - Character special files: están relacionados con I/O y usado para algunos dispositivos.
        - Archivos binarios: con programas ejecutables
- Tipos de acceso a archivos:
    - Acceso secuencial
        - Implementado en sistemas operativos primitivos
        - Un proceso puede leer únicamente todos los bytes que están contenidos
    - Random Access File
        - Puedo leer los bytes o records de un archivo en cualquier orden
        - Acceso a los récords por llaves



- File metadata
  - Datos que describen a otros datos
- Directorios
  - Archivos especiales usados para llevar track de los archivos
  - La forma más simple de un sistema de directorios es teniendo un directorio que contiene todos los archivos
  - Un directorio de sistema jerárquico nos permite tener muchos directorios almacenados.
- Layout



  - Sector 0 es llamado Master Boot Record (MBR)
  - https://gcallah.github.io/OperatingSystems/FileImplementation.html (información adicional sobre la imagen)
  - En el superblock están los parámetros llave del file system
  - En los i nodos está el arreglo de la estructura de datos, ahí se guarda la metadata
  - Asignación contigua:
    - Todos los bloques del archivo están uno a la par del otro.
    - Se asignan los archivos de forma secuencial
    - Desfragmentar memoria ram tiene su costo, hacerlo en discos duros es mucho peor.
  - Asignación por lista enlazada:
    - Cada archivo hace una lista enlazada de bloques.
    - El acceso random es más costoso
    - Punteros en disco duro
      - Perdemos espacio
  - I nodos, tarea moral…
    - Investigar qué son y cómo se utilizan para implementar un sistema de archivos

-

Algoritmos de compresión

- Tipos:
    - Con pérdida
        - Identifica información innecesaria y se remueve
    - Sin pérdida
        - Se repite el patrón
- Proceso de reducir el tamaño de un archivo

Con pérdida

- La secuencia general no puede ser regenerada de la comprimida
- Usa aproximaciones inexactas para representar un contenido
- Ocupo métodos que me permitan tratar de aproximar los métodos que tenía la información
- Es comúnmente usada para comprimir multimedia, especialmente en streaming

Sin pérdida

- No hay pérdida de información