|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

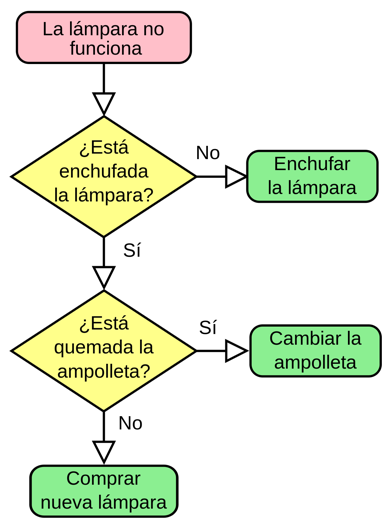
salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Tista Gacía Edgar M.I. |
| *Asignatura:* | Estructuras de datos y algoritmos I |
| *Grupo:* | 1 |
| *No de Práctica(s):* | 11 |
| *Integrante(s):* | Garrido Sánchez Samuel Arturo <418046193> |
|  | Hernández Beltrán David Roni <315082061> |
|  |  |
| *Semestre:* | 2018-2 |
| *Fecha de entrega:* | 7 de Mayo del 2018 |
| *Observaciones:* |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**Estrategias para la construcción de algoritmos 1**

**Introducción**

La construcción de programas que sean útiles para resolver problemas ha existido prácticamente antes que se inventara la computación. En la antigüedad el matemático persa Al-Khowarizmi por primera vez define la palabra algoritmo como una serie de pasos que deben seguirse para obtener un resultado. En 1936 con el cálculo lambda Alonzo Church define formalmente el término algoritmo. La máquina de Turing igual fue partícipe de esto.

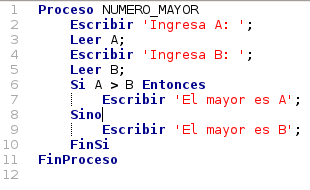
En la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas. Algunos ejemplos son los manuales de usuario, que muestran algoritmos para usar un aparato, o las instrucciones que recibe un trabajador de su patrón. Algunos ejemplos en matemática son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema de ecuaciones lineales.

En términos de programación, un algoritmo es una secuencia de pasos lógicos que permiten solucionar un problema.

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un cálculo o un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida). Sin embargo, cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular.

**Un algoritmo computacional debe cumplir con los siguientes requisitos:**

* Debe ser una descripción exacta de las actividades a realizar que cualquiera pueda tener éxito al realizarlas incluso si no tiene idea del objetivo del algoritmo.
* Debe ser absolutamente libre de ambigüedades excluyendo interpretaciones diferentes.
* No importa quien ejecute las instrucciones y en qué momento del tiempo se realicen, el resultado debe ser exactamente el mismo cada vez que se aplique.

En los algoritmos principalmente libre de ambigüedades podemos hacer una notoria diferencia entre cada uno dependiendo de su paradigma.

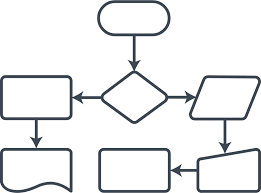
**Formas de representar un algoritmo**

* Diagrama de flujo
* Pseudocódigo

**Objetivo de la práctica:**

* Aplicar los algoritmos básicos para la solución de problemas.

**Actividades:**

* Revisar el concepto y un ejemplo de la estrategia por fuerza bruta.
* Revisar el concepto y un ejemplo de la estrategia ávida.
* Aplicar las estrategias Bottom-Up y Top-Down en un ejemplo.
* Escribir en archivos de texto plano.
* Escribir y leer de archivos binarios.

**Teoría básica:**

**Algoritmo:**

Un algoritmo es un conjunto de pasos lógicos y estructurados que nos permiten dar solución a un problema.

La importancia de un algoritmo radica en desarrollar un razonamiento lógico matemático a través de la comprensión y aplicación de metodologías para la resolución de problemáticas, estas problemáticas bien pueden ser de la propia asignatura o de otras disciplinas como matemáticas, química y física que implican el seguimiento de algoritmos, apoyando así́ al razonamiento critico deductivo e inductivo.

Dentro de los tipos de algoritmos podemos ver diferentes tipos de paradigmas, de los cuales solo se hablará los primeros. Tienen sus características, pero todos pueden guiarnos a una solución, queda determinar cuál es la óptima porque somos programadores y la optimización garantiza programas de calidad al momento de desarrollarlos.

**Ejercicio 1: Ejemplos**

**Ejemplo 1: Fuerza bruta**

**Características**

Es el algoritmo más simple posible.

Consiste en probar todas las posibles posiciones del patrón en el texto

Compara de izquierda a derecha.

**Lógica**

Se sitúa el patrón en la primera posición, y se compara carácter a carácter hasta encontrar un fallo o llegar al final del patrón.

Se pasa a la siguiente posición y se repite el proceso.

El proceso finaliza al alcanzar el final del texto

**Ventajas**

Es el algoritmo más simple posible.

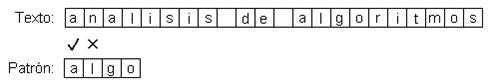
Su razonamiento es sencillo y compresible para mayoría de público

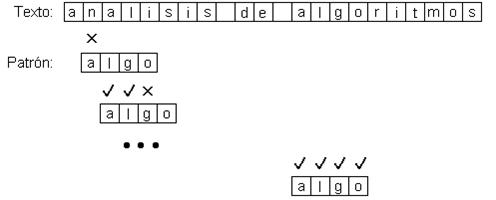
**Desventajas**

Gasto de tiempo

Gasto de recursos del ordenador innecesariamente

**Ejemplo:** Se alinea la primera posición del patrón con la primera posición del texto, y se comparan los caracteres uno a uno hasta que se acabe el patrón, esto es, se encontró una ocurrencia del patrón en el texto, o hasta que se encuentre una discrepancia.





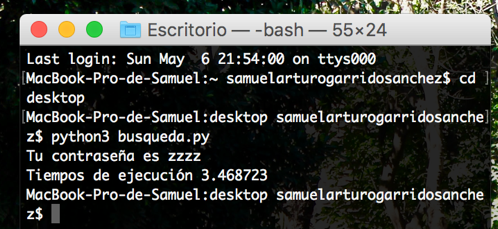
Si se detiene la búsqueda por una discrepancia, se desliza el patrón en una posición hacia la derecha y se intenta calzar el patrón nuevamente.

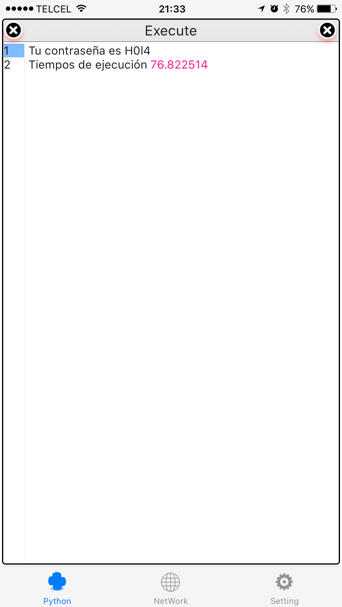
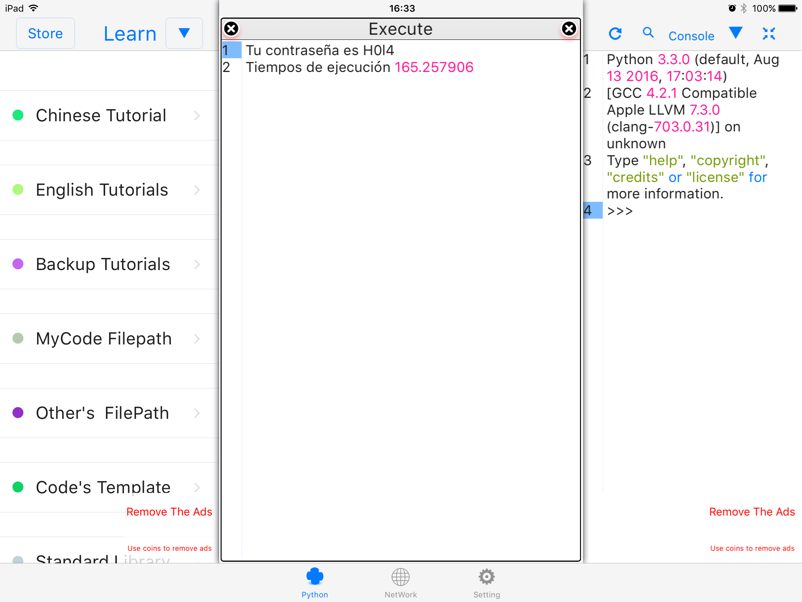
Fuerza bruta es como es llamado al algoritmo más sencillo de comprender y escribir, sin embargo, casi siempre es el menos óptimo para realizar diversas tareas. Un ejemplo muy común es el de tratar de adivinar una contraseña a través de un código. Si nos podemos a pensar la manera más sencilla para nosotros humanos, lo primero que se nos vendría en mente es intentar con todas la letras y números disponible hasta llegar al resultado. Esto suena muy integrante que un humano lo realice, ya que más larga sea a contraseña las combinaciones posibles se aumentan. En particular si queremos adivinar cuántas combinaciones posible hay teniendo 26 letras en nuestro alfabeto, doblando la cantidad anterior por las mayúsculas, tendríamos entonces una cantidad de 2\*26^(longitud de la contraseña), si consideramos una contraseña de 4 caracteres, obtendremos 913,952 combinaciones posibles SOLO de 4 espacios de longitud, si agregáramos letras y un intervalo que el usuario podría almacenar (digamos que pueda ingresar una contraseña entre 4 a 6 caracteres), tendríamos que aumentar esa cantidad a otra descomunal.

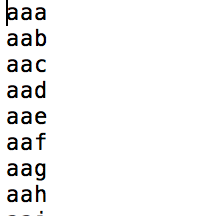
*Como podemos ver la fuerza bruta no es la mejor opción para diseñar algoritmos, ya que el análisis que se realiza es muy vago pero la talacha en este es mucha, por lo que la principal desventaja de este algoritmo es el tiempo gastado hasta obtener el resultado además de gastar recursos.*

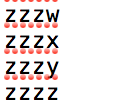
En el ejemplo mostrado en el repositorio pudimos poner a prueba este tipo de algoritmo de búsqueda exhaustiva de una contraseña. Para que pudiéramos darnos cuenta de la desventaja de este algoritmo tenemos unos comandos especiales dentro de la función buscador que básicamente nos colocarán todas las opciones que haya checado el ordenador hasta encontrar la correcta. Por lo que, mediante la gestión de archivos (creación de archivos de texto mediante un código) podremos ver las combinaciones posibles y el tiempo que se tarda en encontrar la contraseña.

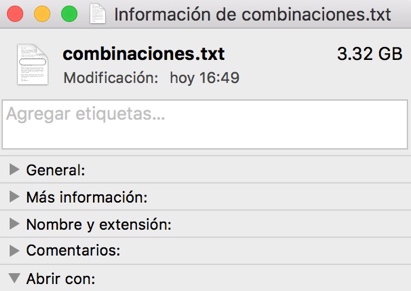
**Esto es muy interesante ya que podremos usar este algoritmo para medir la capacidad de procesamiento de un ordenador o incluso se probó el algoritmo con dispositivos celulares y tabletas a manera de GEEKBENCH para que se notara más la potencia de procesamiento entre cada uno:**

****

 **Computadora de escritorio. Laptop**

  **Tableta electrónica Celular**

Como podemos ver estos números varían dependiendo de la capacidad de procesamiento que tenga nuestro ordenador, así que veremos que el algoritmo de tipo Fuerza Bruta no es nada apto para ordenadores poco ponentes y aun así tarda algo de tiempo. Además de esto fue creado un archivo .txt con todas las combinaciones posibles que el ordenador haría:

* Se experimentó cambiar la longitud de la contraseña a 8 caracteres, checando desde longitudes de 3, pasando por las de 4 etc. El resultado fue **desastroso** ya que son demasiadas combinaciones por lo que a los 5 minutos fue detenida la ejecución y el TXT (sí, un simple archivo de texto) pesaba 3.32 GB!!

Podemos decir que los algoritmos de fuerza bruta no son los más amigables con el tiempo de ejecución y en el caso que nos genere archivos en memoria pues menos, ya que lo demás no nos sirve, solo la contraseña correcta.

**Ejemplo 2: Voraces o ávidos (Greedy)**

**Características**

Trata de tomar las mejores opciones a corto o a la ejecución en curso

**Lógica**

Estando en una situación elige al que más sea conveniente dependiendo de las condicione con que sea programada, sin embargo, no es necesariamente el camino óptimo para todo el algoritmo

**Ventajas**

Su lógica es simple, solo eligiendo al mejor en el paso en curso y sucesivamente.

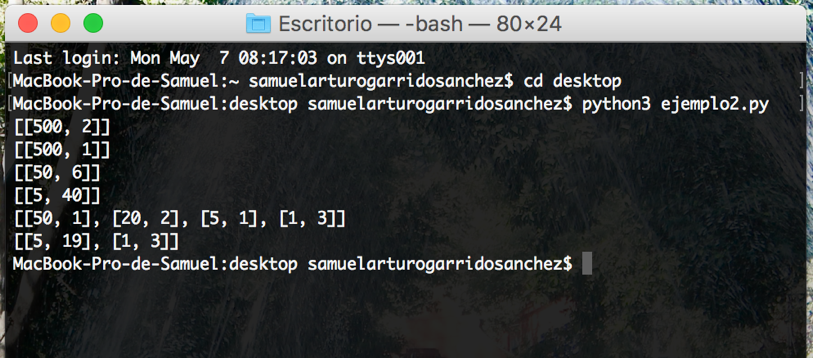
Su análisis es práctico, aunque no necesariamente óptimo.

**Desventajas**

Muchas ocasiones la solución ideal para cada caso no nos guía por una solución óptima universal. Algunas ocasiones es incluso el peor camino de resultados posibles del algoritmo

Los algoritmos voraces se caracterizan por no pensar a futuro sino elegir la opción “más beneficiosa” a corto plazo que es la acción inmediata. Esto es contraproducente ya que no siempre eligiendo “lo más conveniente” en el instante siguiente nos garantiza que al final del algoritmo hayamos obtenido la solución óptima. Un ejemplo claro puede darse en un recorrido de automóvil, si voy desviando por el recorrido más corto llego más rápido al destino, pero hay semáforos de por medio. Si nos ponemos a pensar mejor podría ser que no necesariamente la ruta más corta sea la más rápida. Esto se aplica en este tipo de algoritmos que se quisiera elegir lo mejor en cada decisión que se tome.

En algoritmo que se presenta de ejemplo corresponde a un cambio de divisas creado en Python. El problema consiste en regresar el cambio de monedas, de cierta denominación, usando el menor número de éstas. Este problema se resuelve escogiendo sucesivamente las monedas de mayor valor hasta que ya no se pueda seguir usándolas y cuando esto pasa, se utiliza la siguiente de mayor valor.

Nótese que en particular este problema no podremos garantizar que se utilice la menor cantidad de monedas posibles ya que el algoritmo solo piensa a corto plazo.

Este problema se resuelve escogiendo sucesivamente las monedas de mayor valor hasta que ya no se pueda seguir usándolas y cuando esto pasa, se utiliza la siguiente de mayor valor. La desventaja en esta solución es que, si no se da la denominación de monedas en orden de mayor a menor, se resuelve el problema, pero no de una manera óptima.

**Ejemplo 3: Bottom-Up**

**Características**

Es equivalente a un pensamiento inductivo en la investigación, va de lo específico a lo general, por lo que la solución del algoritmo se ve realizada de los primeros pasos para general todo el conjunto de soluciones.

**Lógica**

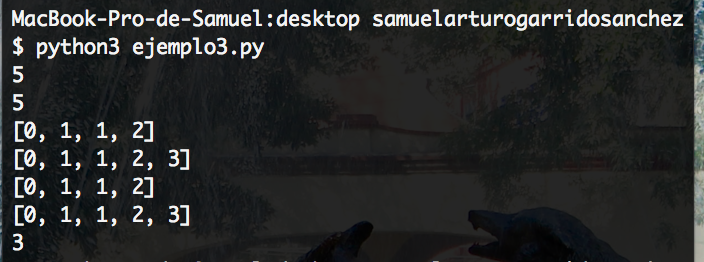
Determina las soluciones de un problema a partir de subproblemas que ya han sido resueltos.

**Ventajas**

Su lógica es muy utilizada en muchos programas de hoy en día, partiendo de los primeros elementos.

**Desventajas**

Puede ser contraproducente trabajar con ello ya que algunas ocasiones los casos específicos no nos guían a una solución o formula general.

****También conocida como de arriba-abajo consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema.

Se nos plantea un algoritmo que genere la serie de Fibonacci a través de los primeros elementos de la serie siendo 0 y 1 (la serie comienza con 1 y 1 pero por el uso de una variable auxiliar tmp empezamos en 0), esto vemos que básicamente va construyendo a la serie a través de estas 2 iniciales.

Su funcionamiento es utilizar la variable auxiliar para primero cambiar a f1 y f2 (recorrerlos), luego de esto tmp toma el valor de la suma de estos 2 y finalmente a f2 le asignamos el valor de la suma determinada anteriormente. Con esto podemos decir que vamos recorriendo la serie de Fibonacci

**Ejemplo 4: Top-down**

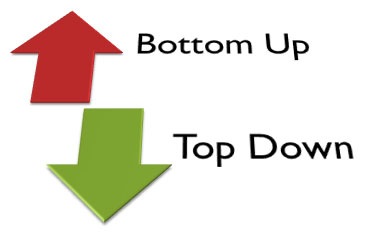
**Características**

Es equivalente a un pensamiento deductivo en la metodología de la investigación, solo que en esta ocasión partimos de una generalidad hacia casos específicos, descomponiendo el problema en pedazos pequeños.

**Lógica**

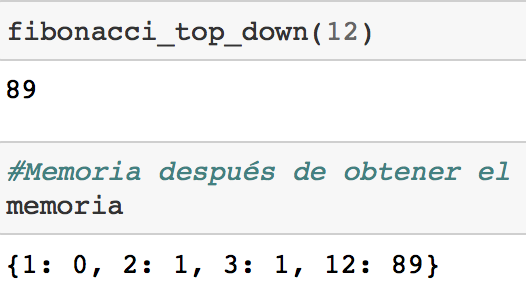
Se determina la solución de un problema partiendo de una estructura y separándola en conjuntos menores para realizar en cada una.

**Ventajas**

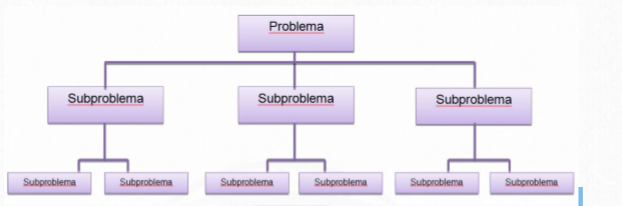
Su lógica es muy útil para la solución de varios problemas como es la operación de una calculadora insertando varios comandos a la vez

**Desventajas**

Puede ser incorrecta su utilización ya que descomponer le problema en subproblemas debe ser de una manera que garanticemos que juntando las soluciones se llega a una solución general.

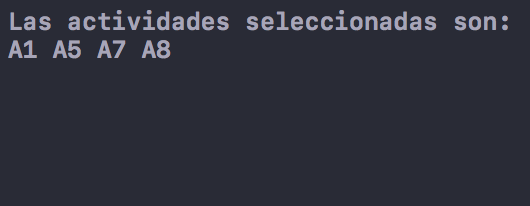
A diferencia de **Bottom-Up**, aquí se empiezan a hacer los cálculos de n hacia abajo

Si se aplica un enfoque y una apreciación empleando un modelo top-down, **las decisiones relacionadas con la inversión son evaluadas desde una mirada de información global, hasta ir abordando los valores y las variables más detalladas y específicas**.



Además, se aplica una técnica llamada memorización la cual consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones

Para aplicar la estrategia **Top-Down**, se utiliza un diccionario (memoria) el cual va a almacenar valores previamente calculados. Una vez que se realice el cálculo de algún elemento de la sucesión de Fibonacci, este se va a almacenar.

**Ejercicio 2: Horarios Greedy**

*El ejercicio es un problema de optimización relativo a la selección de actividades “no conflictivas” a realizar denro de un marco de tiempo determinado. Dado un conjunto de actividades cada una marcada por un tiempo de inicio y un tiempo de finalización. Máximo de actividades que pueden ser realizadas por una persona, suponiendo que solo puede trabajar en una sola actividad a la vez.*

**Tipo de algoritmo: GREEDY (voraz)**

Recuerde que un algoritmo Greedy o voraz busca la solución más cercana, sin analizar las demás posibles soluciones. Estas tomas de decisiones a corto plazo en algunas ocasiones nos derivan en soluciones generales no óptimas. Ya se había puesto un ejemplo en los ejemplos que no necesariamente nos derivan en respuestas generales más adecuadas.

El ejercicio 2 es un claro ejemplo del algoritmo GREEDY ya que genera una solución, pero no la más optima, aparte que una vez tomada una decisión, esta no vuelve a remplazarse en el futuro. Ejemplo de un funcionamiento no óptimo del código 2 es el siguiente:

**Si los valores de los horarios fueran los siguientes:**

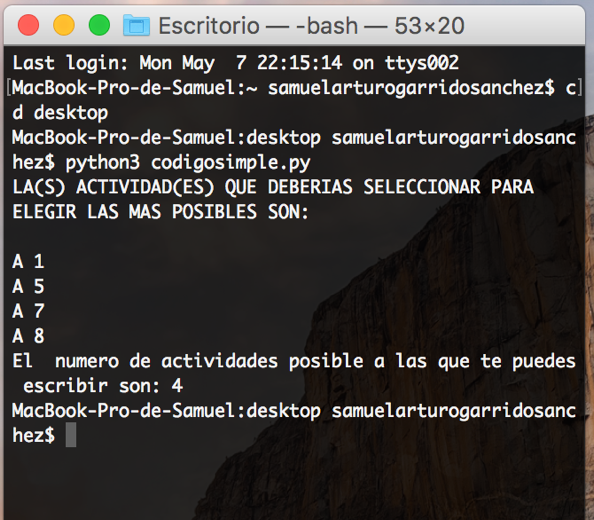
**s[] = {20,13,15,17,19}**

**f[] = {22,14,16,18,20}**

El código te arrojará como resultado que selecciones las actividades de: A1(20-22 hrs) sin embargo esa no es la respuesta óptima, la respuesta óptima sería que se seleccione el conjunto:

**A2(13-14 hrs) A3(14-15 hrs) A4(15-16 hrs) A5(17-18 hrs) A6(19-20 hrs) A7(20-21hrs)**

Ninguna se interfiere. Y además será la mejor opción la que no incluya a A1 que nuestro algoritmo GREEDY sin dudarlo lo hubiera elegido por ser la solución a corto plazo más convincente.

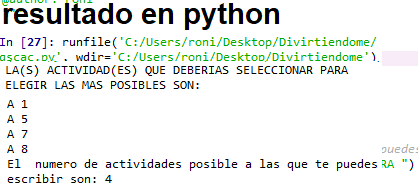
****Como se puede observar, la diferencia entre la solución arrojada por el código del ejercicio número 2 , con la solución óptima del programa es bastante significativa.

Se realizó el algoritmo en el lenguaje de programación Python, con los mismos valores que se asignaron a la actividad en el ejemplo colocado en la hoja de actividades donde:

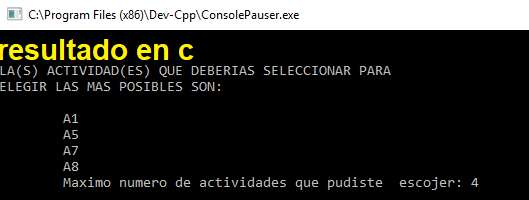
**inicio=[1,2,3,2,4,5,6,8,7]**

**final=[4,5,6,8,6,7,7,12,9]**

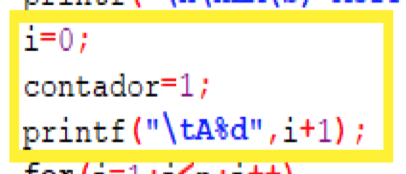
Corremos el algoritmo y nos determina que los mejores horarios serían los mostrados en la captura.

**Análisis**

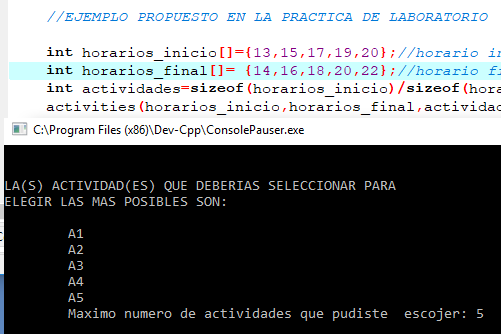
Antes que nada aquí dejo la captura del correcto funcionamiento del programa del ejercicio 2, en los 2 diferentes programas, python , y c.

**Como observación podemos ver claramente que arrojan los mismos resultados.**

El código del ejercicio 2 es muy sencillo, para empezar siempre en todos los casos ese código considerada como primera opción la actividad numero 1 (A1). En la imagen siguiente se pude ver claramente , en el cuadro subrayado de amarillo lo mencionado anteriormente.

****

Esto se afirma ya que antes de entrar el ciclo for , la función ya esta imprimiendo como resultado la actividad numero 1 (A1) , en pocas palabras el programa considera esa opción por default automáticamente , sin interrogarla ni cuestionar su posible ineficiencia, sin hacerle ningún filtro.

Después lo que realizara por medio del ciclo for es, si la hora de termino de la actividad numero 1 es mayor a la hora de inicio de la actividad 2, la actividad 2 no podrá incluirse a la lista , y asi sucesivamente hasta que haya una x actividad que inicie despues o al mismo tiempo que acaba la actividad numero 1 , apartir de ese momento x actividad se agenda en a lista de actividades , y ahora esta es la que buscara sucesivamente actividades hasta que enceuntre una que inicie cuando esta acaba o despues, y asi sucesivamente hasta que se terminan las actividades , etc.

Y bueno ese fue un simple problema, con una solución bastante fácil, y el programa no pudo ejecutarlo, y **¿Por qué?** Simplemente como ya habíamos mencionado el algoritmo del programa toma como primera solución forzada sin pensarla , la actividad número 1 , y la actividad número uno era la que más tarde empezaba y las siguientes empezaban más temprano. Ese es una deficiencia de bastantes más que el algoritmo no está preparado para resolver, una posible solución para este problema en particular, sería que el algoritmo ordene de menor a mayor sus horas de actividades para que no ocurra esto, si esto hubiese ocurrido el resultado hubiese sido distinto:

**Conclusiones**

**Conclusión de Samuel Arturo Garrido Sánchez**

Dentro de esta práctica se aprendió a diferenciar todas las variantes de algoritmos que se plantearon. Se vio que no es siempre conveniente utilizar una u otra sino dependiendo de la situación es como tendremos que analizar cuál es el tipo de algoritmo que nos ofrece una optimización de nuestro programa. Esto es crucial en la programación para poder determinar mejores soluciones e imprentarlas en programas optimizados que garanticen una experiencia de usuario maravillosa y un uso de recursos mínimos.

**Conclusión de David Roni Hernández Beltrán**

La programación es simplemente muy laboriosa pero muy beneficiosa, y digo esto porque problemas cotidianos que parecieran tan simples y sobre todo tan comunes de la vida real, son realmente complicados.

Creía que en el mundo de la programación no existía tanta metodología, u mecanismos para la solución de **GRANDES PROBLEMAS.** A medida que avanzan las practicas, siento que mi nivel a como llegue al principio de semestre ha mejorado bastante, al programar ya no veo el compilador como antes lo hacía, si no que creo que veo más allá de códigos y errores por sintaxis, puedo ver una grandiosa forma de comunicarte con un dispositivo tan inteligentemente lógico con una potencia inimaginable.

*El objetivo de la práctica:*

* **Aplicar los algoritmos básicos para la solución de problemas.**

Se concluyó con mucho éxito al entender muy bien los diferentes tipos de metodologías que siguen estos tipos de algoritmos, puede que alguno nos ayude más en otras situaciones que en otras debido a que, así como pueden ser diversas las soluciones, pueden ser diversos los problemas por lo que se debe practicar más a tener esa visión para determinar con qué tipo de algoritmo o algoritmos somos capaces de dar solución a nuestros problemas a través de programas pues somos programadores.

**Crítica constructiva:**

La práctica se nos hizo muy entretenida, aunque algo sencilla esta ocasión. Se realizó además otro código donde el usuario ingresaría el arreglo de los horarios, no estarían inicializados y veríamos más recalcado ese paradigma en el ejercicio. Es importante que como programadores aprendamos muy bien a utilizar prudentemente estas herramientas porque puede que todas nos guien a una solución pero no la mejor y nuestro op¡bjetivo es obtener la mejor posible, minimizando costos y posibles retrasos, mejorando la experiencia para los futuros usuarios de nuestras plataformas

**Bibliografía**

https://github.com/eegkno/FI\_UNAM/blob/master/02\_Estructuras\_de\_datos\_y\_algoritmos\_1/P11/EDyA11\_I.ipynb

https://docs.python.org/3/library/itertools.html#itertools.product

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edition.