El formato BCD o *Binary Coded Decimal* se usa para representar números enteros positivos en base 10. Cada cifra decimal del número se codifica en binario en 4 bits. Por ejemplo el número 60219 se codifica como 0b0110 0000 0010 0001 1001, en donde el 0110 codifica el 6, el 0000 el 0, el 0010 el 2, el 0001 el 1 y el 1001 el 9. La ventaja es que al anotar la secuencia de bits en hexadecimal se lee exactamente el mismo número representado en BCD, es decir 0x60219. Observe que las secuencias 0b1010 … 0b1111 o 0xa … 0xf no pueden ocurrir y que 0x60219 representa en BCD el número 60219 mientras que el mismo 0x60219 representa en binario el número 393753 ($9 + 1*16+2*16^2+6*16^4$).

Programe la función *sumaBcd* que retorna la suma de 2 números en formato BCD, almacenados en enteros sin signo de 64 bits, es decir *unsigned long long*. El número más grande representable puede tener hasta 16 cifras decimales. Si el resultado de la suma tiene 17 cifras decimales se produce desborde y Ud. debe retornar 0xffffffffffffffff. El encabezado de la función *sumaBcd* es:

```
typedef unsigned long long Bcd;
Bcd sumaBcd(Bcd x, Bcd y);
```

Ejemplo de uso:

```
Bcd a= sumaBcd(0x60219, 0x1); // a es 0x60220
Bcd b= sumaBcd(0x199305, 0x9781); // b es 0x209086
Bcd c= sumaBcd(0x9999999999999999, 0x1); // c es 0xfff...ffff
```

Observe que no sirve sumar directamente los números en BCD con el operador + de C porque 0x60219 + 0x1 es 0x6021A, que es una secuencia de bits inválida en BCD. Por lo tanto Ud. necesita separar las cifras decimales de a 4 bits y sumar exactamente como aprendió a hacerlo en enseñanza básica.

Restricciones:

- Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use los operadores de bits eficientemente.
- No se permite convertir los números a binario, sumarlos con + y convertir el resultado a BCD.
- Se descontará medio punto por no usar el estilo de indentación de Kernighan como se explica en esta sección de los apuntes.
- El estándar de C no especifica el resultado para desplazamientos mayores o iguales al tamaño del operando. Sanitize rechaza el desplazamiento *x<<nbits* cuando *nbits* es mayor o superior a la cantidad de bits de *x*.

Instrucciones

Baje *t1.zip* de U-cursos y descomprímalo. El directorio *T1* contiene los archivos (a) *test-suma.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *suma.h* que incluye el encabezado de la función pedida, y (d) *Makefile* que le servirá para compilar y ejecutar su tarea. Ud. debe programar la función *sumaBcd* en el archivo *suma.c*.

Pruebe su tarea bajo Debian 11 de 64 bits nativo o virtualizado con VirtualBox, Vmware, QEmu o WSL 2. **Ejecute el comando *make* sin parámetros**. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo desplazamientos indefinidos.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *suma.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.

Parte a.- Programe la función:

```
void eliminar(char *str, char *pat);
```

Esta función elimina del string *str* todas las apariciones del patrón *pat*. Ejemplos:

```
char s1[]= "las palas van"; // 13 caracteres
eliminar(s1, "las"); // s1 es " pa van" (7 caracteres)
char s2[]= "11101101";
eliminar(s2, "10"); // s2 es "11101"
```

Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Use ++ -- *p+i* o *p-i*. Por razones de eficiencia, Ud. no puede usar *malloc* o declarar un arreglo para pedir memoria adicional. Use múltiples punteros para direccionar distintas partes del string.

Parte b.- Programe la función:

```
char *eliminados(char *str, char *pat);
```

Esta función entrega un nuevo string en donde se han eliminado del string *str* todas las apariciones del patrón *pat*. Ejemplos:

```
char *s1= eliminados("las palas van", "las");
          // s1 es " pa van" (7 caracteres)
char *s2= eliminados("11101101", "10");
          // s2 es "11101"
```

Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Use ++ o *p+i*. Para recorrer el string use aritmética de punteros. Use *malloc* para pedir memoria para el string resultante. Debe pedir exactamente la cantidad de bytes que necesita el resultado, no más. Para el primer ejemplo debe pedir 8 bytes, para el segundo 6 bytes. Si pide más memoria que la que necesita, el test de uso de memoria podría agotar la memoria de su computador haciendo que se ponga muy lento antes de que el programa falle.

Instrucciones

Baje *t2.zip* de U-cursos y descomprímalo. El directorio *T2* contiene los archivos (a) *test-elim.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *elim.h* que incluye los encabezados de las funcion pedidas, y (d) *Makefile* que le servirá para compilar y ejecutar su tarea. **Ejecute en un terminal el comando *make*** para recibir instrucciones adicionales. Estos son los requerimientos para aprobar su tarea.

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo goteras de memoria.

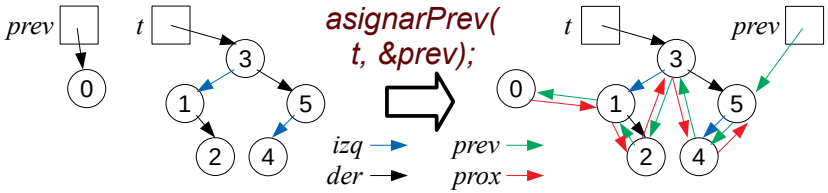
Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *elim.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.

En un *recorrido en orden* de un árbol binario, se visita recursivamente primero el subárbol izquierdo, luego se visita la raíz y finalmente se visita recursivamente el subárbol derecho. Considere que se está visitando un nodo T al recorrer un árbol binario en orden. Se define como *previo* a T el nodo que se visitó anteriormente, y como *próximo* el nodo que se visitará a continuación. Estudie el lado derecho de la figura de ejemplo. Programe en el archivo *prev.c* la función *asignarPrev* que asigna los campos *prev* y *prox* agregados a la estructura de los nodos de un árbol *t*. El encabezado de la función se muestra a la derecha. El parámetro **pprev* es de entrada y salida. El nodo previo del primer nodo visitado (el nodo 1 en el ejemplo) debe ser el nodo apuntado inicialmente por **pprev* (nodo 0) y el nodo próximo del último nodo en ser visitado (nodo 5) debe ser NULL. En **pprev* debe quedar finalmente la dirección del último nodo visitado (nodo 5). En el siguiente ejemplo de uso las variables *t* y *prev* son de tipo Nodo *.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
  struct nodo *prev, *prox;
} Nodo;
void asignarPrev(Nodo t,
                 Nodo **pprev);
```



Restricción: Su solución debe tomar tiempo linealmente proporcional al número de nodos en el árbol *t*.

Ayuda: Cuando visite el nodo T, su nodo previo es **pprev*. Asigne NULL a su nodo próximo por ahora. Si el nodo previo a T no es NULL, T es el nodo próximo del nodo previo a T. Antes de continuar el recorrido, asigne T a **pprev*.

Instrucciones

Descargue *t3.zip* de U-cursos y descomprímalo. El directorio *T3* contiene los archivos (a) *test-prev.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *prev.h* que incluye los encabezados de las funcion pedidas, (d) *Makefile* que le servirá para compilar y ejecutar su tarea, y (e) *prev.cbf* para que pueda probar su tarea con *codeblocks*. **Ejecute en un terminal el comando *make*** para recibir instrucciones adicionales. Estos son los requerimientos para aprobar su tarea.

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo *heap-buffer-overflow*.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *prev.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.

El archivo de texto *dicc.txt* que viene a continuación almacena un diccionario representado como una tabla de hashing cerrado.

```
celular:aparato portatil de un sistema de telefonia celular

casa:edificacion construida para ser habitada

posada:establecimiento economico de hospedaje para viajeros
alimento:sustancia ingerida por un ser vivo
```

Todas las líneas son de 80 caracteres más el fin de línea (caracter '\n'). Se usa el caracter ':' para separar la llave de su valor asociado. Una línea que contiene solo espacios en blanco no contiene ninguna definición.

Escriba en el archivo *definir.c* un programa que agregue una llave al diccionario. El siguiente es un ejemplo de uso:

```
$ make definir.bin
$ ./definir.bin dicc.txt bolsillo "bolsa pequena"
```

En donde el primer parámetro es el nombre del archivo que contiene el diccionario, el segundo es la llave definida y el tercero es la definición. En el archivo *definir.c*, Ud. deberá programar la función:

*int main(int argc, char *argv[ ]);*

Requerimientos

- Abra el archivo del diccionario en modo "r+" para poder leerlo y escribirlo. Es decir: *fopen(..., "r+")*.
- Calcule el tamaño y el número de líneas en el archivo usando *fseek y ftell*.
- **Debe ser eficiente.** Use *hash_string(llave)* módulo número de líneas del archivo para calcular el número de línea en donde debe ir la llave definida. Luego use fseek y fread para leer esa línea. Si esa línea está desocupada reescriba esa misma línea para agregar la definición en esa misma línea. Si no, lea las líneas que vienen a continuación hasta encontrar una línea en blanco y reescríbala para agregar la defición en esa línea. El archivo es circular: si llega al final, continúe desde el comienzo.
- Está prohibido leer el archivo completo, salvo si el archivo está lleno.

Debe diagnosticar las situaciones de error en la salida estándar de errores. En caso de error termine el programa de inmediato. Estos son los errores que debe diagnosticar:

- No se puede abrir el archivo. Debe diagnosticar este error con *perror*.
- La llave ya estaba definida en el diccionario.

El test de prueba incluido verifica que Ud. diagnostique estos errores con exactamente el mismo mensaje que entrega la solución de referencia incluida en formato binario. Puede probar la solución de referencia con un diccionario y llave específica. Por ejemplo:

```
$ ./prof.ref-$(arch) dicc.txt bolsillo "bolsa pequena"
```

Vea el diccionario resultante con el comando: `less dicc.txt`

Cada ejecución de *definir* modifica el diccionario. Para restaurarlo ejecute el comando: `bash mk-dicc.sh`

Instrucciones

Descargue *t4.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros en el directorio *T4* para recibir instrucciones acerca del archivo en donde debe programar su solución (*T4/definir.c*), cómo compilar y probar su solución, los requisitos que debe cumplir para aprobar la tarea (*make run-san*, *make run-g* y *make run*) y cómo entregar su tarea por U-cursos (*make zip*).

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *definir.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.

Una palabra es una secuencia contigua de caracteres que no contiene ningún espacio en blanco. Por ejemplo el string "estaesunasolapalabra," contiene una palabra y el string " estas son 4 palabras. " contiene 4 palabras.

La función *sort* está programada en C en el archivo *sort-c.c* y en assembler Risc-V en *sort-rv.s*. Esta función ordena lexicográficamente un arreglo de strings usando un algoritmo ridículamente ineficiente, porque toma tiempo $O(n^3)$. En *sort-rv.s*, el código equivalente en C está comentado, mostrando la ubicación de las variables en los registros.

El encabezado de la función es: *void sort(char *a[ ], int n);*

En el arreglo de strings *a* vienen *n* frases como *"quien quiere queque" o "antes yo no era asi y ahora sigo igual"*.

El archivo *sort-rv-wc.s* es una copia de *sort-rv.s* y *sort-c-wc.c* es una copia de *sort-c.c*. Modifique las funciones *sort* en *sort-c-wc.c* y en *sort-rv-wc.s* de modo que se ordene el arreglo de strings ascendentemente por el número de palabras en el string. El siguiente ejemplo muestra a la izquierda un arreglo ordenado lexicográficamente y a la derecha ordenado por número de palabras.

ordenado lexicográficamente	*ordenado por número de palabras*
Defina universo y de 2 ejemplos El futuro ya no es lo que solia ser Soy inseguro, ¿o no?	Soy inseguro, ¿o no? Defina universo y de 2 ejemplos El futuro ya no es lo que solia ser

Instrucciones

Baje *t5.zip* de U-cursos y descomprímalo. Contiene el *Makefile* y los archivos que necesita para hacer esta tarea. Ejecute el comando *make* sin parámetros para recibir instrucciones sobre la ubicación de los archivos que debe modificar y cómo compilar, ejecutar y depurar. En particular lea los tips para la depuración y la solución de problemas.

Restricciones

No hay restricciones para la programación de *sort-c-wc.c*. Para la programación de *sort-rv-wc.s* **Ud. solo puede modificar el código que compara los elementos consecutivos**. Está claramente delimitado en el archivo original. No modifique nada más. Sin esta restricción la tarea sería trivial. **Además para hacer la comparación *no puede* invocar otras funciones**. Una vez hecha la comparación, la ejecución debe continuar en la etiqueta *.decision* y el resultado de la comparación debe quedar en el registro *t1*. Si *t1>0*, los números en *p[0]* y *p[1]* están desordenados y por lo tanto se intercambiarán. Si *t1≤0* no se intercambiarán.

Ayuda

- Programe en *sort-c-wc.c* una función auxiliar que entregue el número de palabras en un string. Haga un esfuerzo en llegar a la función más pequeña, porque así será menor la cantidad de líneas en assembler que deberá programar y depurar. Pruebe *sort-c-wc.c* ejecutando el comando: *make sort-c-wc.run* (*make sort-c-wc.ddd* para depurar)
- Una vez que pase exitosamente la prueba de *sort-c-wc.c*, ejecute el comando *make sort-c-wc.s* y luego estudie en el archivo *sort-c-wc.s* la traducción de la función que calcula el número de palabras a assembler Risc-V. Use un código similar para calcular el número de palabras en la zona delimitada en *sort-rv-wc.s* y así completar su tarea. Recuerde: sin invocar otras funciones. Pruebe su tarea con: *make sort-rv-wc.run* (*make sort-rv-wc.ddd* para depurar)
- Revise si el toolchain para compilar y ejecutar programas para RiscV está instalado en */opt/riscv*. Si no encuentra ese directorio, descargue el archivo *riscv.tgz* (*riscv-arm.tgz* para Arm) de esta carpeta de google drive e instale el *toolchain* para RiscV con estos comandos:

```
cd ... directorio en donde descargó riscv.tgz ...
sudo bash
cat riscv.tgz | ( cd / ; tar zxvf - )
```

- En la clase auxiliar del viernes 6 de octubre se estudió la solución de una tarea similar de un semestre pasado.

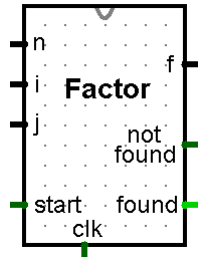
Entrega

Entregue por medio de U-cursos el archivo ***wc.zip*** generado con el comando *make zip*. Este comando verifica que su tarea funcione correctamente y luego genera *wc.zip* con los archivos *sort-c-wc.c*, *sort-rv-wc.s* y *resultados.txt* con la ejecución de la tarea. **Recuerde descargar de u-cursos lo que entregó, descargar nuevamente los archivos adjuntos** y vuelva a probar la tarea tal cual como la entregó. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificacion. Su tarea debe ordenar correctamente, si no será rechazada. Se descontará medio punto por día de atraso (excluyendo sábados, domingos, festivos o vacaciones).

CC3301 Programación de Software de Sistemas – Tarea 6 – Primavera 2023 – Profesor: Luis Mateu

Parte a. (3 puntos) El circuito *Factor* debe buscar un factor impar de un número entero *n* en el intervalo [*i*, *j*] usando el siguiente algoritmo:

```
typedef unsigned int uint;
uint factor(int n, int i, int j) {
  for (uint k= i; k<=j; k+=1) {
    if (n % k == 0) { // calcula el resto de n/k
      return k; // k es un factor de n
  }
  return 0; // no se encontró ningún factor
}
```



La búsqueda comienza cuando se detecta que *start* es 1 justo en el momento en que la entrada *clk* pasa de 1 a 0. En ese instante las salidas *found* y *not fount* deben ir a 0 y permanecer en 0 mientras se realiza el cálculo. Si se encuentra un factor se entrega un 1 en la salida *found* y el factor se entrega en la salida *f*. Si no hay un factor impar en [*i*,*j*] se entrega un 1 en la salida *not-found* y *f* es irrelevante. Las entradas *n*, *i* y *j* permanecerán constantes hasta que se solicite una nueva búsqueda llevando *start* a 1. Debe mantener las salidas *f, found* y *not fount* constantes hasta que *start* vuelva a 1.

Implemente el circuito *Factor* en el módulo *factor* del circuito *factor.circ* incluido en los archivos adjuntos de esta tarea. Para entender cómo resolver esta parte y cómo verificar que funciona correctamente, vea este video en donde explico la solución de un problema similar de un control de arquitectura de computadores. La solución del problema que sale en el video está en el circuito *max1s.circ* de los archivos adjuntos. También le será de mucha utilidad ver los videos de estas 2 cátedras y la clase auxiliar, todos sobre circuitos. **Pruebe que su solución funciona** correctamente seleccionando el módulo *test-factor* y simulando el circuito con *control-r* y *control-k*. Solo obtendrá el puntaje de esta parte si se enciende la luz verde.

Como resultado de esta pregunta Ud. debe entregar el circuito *factor.circ* en donde completó la implementación del módulo *factor*. Puede regular la velocidad de la simulación en *Simular → Seleccionar Frecuencia del reloj*.

***Ayuda*:** El circuito *factor.circ* adjunto ya incluye las componentes que se necesitan para resolver el problema, pero puede no usarlas y usar otras componentes si lo estima necesario. Use el registro *RegK* para representar la variable *k* de la solución en C. Observe que en los comentarios del módulo *factor* está la solución del problema en texto. Solo debe traducir ese texto a un circuito de logisim.

Parte b. (1,5 puntos) La figura muestra un extracto del estado actual de un *caché* de 4 KB (2^{12} bytes) de 1 grado de asociatividad con 256 líneas de 16 bytes. Por ejemplo en la línea 2a del caché (en hexadecimal) se almacena la línea de memoria que tiene como etiqueta 92a (es decir, la línea que va de la dirección 92a0 a la dirección 92af).

línea cache	*etiqueta*	*contenido*
e2	4e2	
45	c45	
2a	92a	

Un programa accede a las siguientes direcciones de memoria (en hexadecimal): c450, 92ac, 5e24, 5e20, 92a8, 2450, 4e20, 92a4. Indique qué accesos a la memoria son aciertos en el caché, cuáles son desaciertos y rehaga la figura mostrando el estado final del cache. Por ejemplo el acceso c450 es un acierto.

Parte c. (1,5 puntos) La tabla de la derecha muestra las instrucciones Risc-V ejecutadas por un programa. Haga un diagrama que muestre el ciclo en que se ejecuta cada etapa de las instrucciones, considerando una arquitectura (i) en pipeline con etapas *fetch*, *decode* y *execute*, y (ii) superescalar, con 2 pipelines con las mismas etapas de (i). Suponga que el salto en F ocurre (y no hay ningún tipo de predicción de saltos). Base su diagrama en los ejemplos que aparecen en estas cátedras.

```
A sub  a3,s5,t2
B add  a5,t2,s4
C andi a3,a3,255
D addi a3,a3,1
E ori  a5,a5,15
F bgt  a3,s1,R
G add  ...
H sub  ...
I xor  ...
J andi ...
...
R sub a3,a3,a5
S ori a3,a3,255
```

Instrucciones

Baje *t6.zip* de U-cursos y descomprímalo. Contiene el circuito *factor.circ*, que Ud. debe modificar, y el circuito *max1s.circ* con la solución del ejemplo del video.

Entrega

Entregue por medio de U-cursos un archivo *.zip* con el circuito *factor.circ* modificado con su solución de la parte *a*, y las soluciones de las partes *b* y *c* en el formato de su elección (por ejemplo foto legible de su solución en papel). La parte *a* es binaria, se otorga 0 o todo el puntaje, pero en las partes *b* y *c* se otorga puntaje de acuerdo a lo logrado. Se descontará medio punto por día de atraso (excluyendo sábados, domingos, festivos o recesos).

El problema del viajante o vendedor viajero responde a la siguiente pregunta: dadas *n+1* ciudades (enumeradas de 0 a *n*) y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que inicia en la ciudad 0, visita cada ciudad una vez y al finalizar regresa a la ciudad 0?

La solución óptima se puede obtener en tiempo $O(n!)$. Un algoritmo más eficiente puede hacerlo casi en tiempo $O(2^n)$. En la práctica es demasiado lento buscar la solución óptima si *n* es grande. La función *viajante* de más abajo es una heurística simple para encontrar una solución aproximada. Recibe como parámetros el número *n* de ciudades (además de la ciudad 0), la matriz *m* de distancias entre ciudades (*m[i][j]* es la distancia entre las ciudades *i* y *j*), un arreglo de salida *z* de tamaño *n+1*, en donde se almacenará la ruta más corta, y un número *nperm*. Esta función genera *nperm* permutaciones aleatorias de las ciudades 1 a *n*. Cada permutación corresponde a una ruta aleatoria partiendo de la ciudad 0, pasando por todas las otras ciudades y llegando a la ciudad 0 nuevamente. La función calcula la distancia recorrida para cada ruta, selecciona la ruta más corta (la que recorre la menor distancia), entregando en *z* cuál es esa ruta y retornando la distancia recorrida por *z*. No es la ruta óptima, pero mientras más grande es *nperm*, más se acercará al óptimo.

```
double viajante(int z[], int n, double **m, int nperm) {
  double min= DBL_MAX; // la menor distancia hasta el momento
  for (int i= 1; i<=nperm; i++) {
    int x[n+1];           // almacenará una ruta aleatoria
    gen_ruta_alea(x, n); // genera ruta x[0]=0, x[1], x[2], ..., x[n], x[0]=0
    // calcula la distancia al recorrer 0, x[1], ..., x[n], 0
    double d= dist(x, n, m);
    if (d<min) {     // si distancia es menor que la que se tenía hasta el momento
      min= d;        // d es la nueva menor distancia
      for (int j= 0; j<=n; j++)
        z[j]= x[j]; // guarda ruta que recorre la menor distancia en parámetro z
  } }
  return min;
}
```

Las funciones *viajante*, *gen_ruta_alea* y *dist* son dadas. Por ejemplo si *n* es 4, después de la llamada a *gen_ruta_alea(x, n)* el arreglo *x* podría ser 0, 4, 1, 3, 2. También podría ser 0, 3, 1, 4, 2, etc. Hay *n!* permutaciones posibles.

Programe la función *viajante_par* con la misma heurística pero generando las *nperm* rutas aleatorias en paralelo en *p* procesos pesados (creados con *fork*). Recibe los mismos parámetros que *viajante*, más el parámetro *p*. Note que **debe entregar la mejor ruta encontrada en el arreglo z**.

Metodología obligatoria: Lance *p* nuevos procesos pesados (hijos) invocando *p* veces *fork*. Cada hijo evalúa *nperm/p* rutas aleatorias invocando *viajante(..., nperm/p)*. Cuidado: recuerde que los procesos pesados no comparten la memoria. Cada hijo debe enviar su mejor ruta al padre por medio de un *pipe*. Use el proceso padre solo para crear a los hijos y para elegir la mejor solución entre las recibidas a través de los *p pipes* que conectan al padre con sus hijos. Si la mejor solución fue por ejemplo la del hijo 3, *viajante_par* debe retornar el valor *min* que calculó el hijo 3 y llevar una copia del arreglo *z* calculado por el hijo 3 al parámetro *z* de *viajante_par*. La forma de crear los procesos hijos y pipes es muy similar a la manera en que se hizo para encontrar un factor de un entero en la cátedra del jueves 16 de noviembre. La forma de enviar el arreglo *z* por el pipe es similar a la manera en que se envía el arreglo ordenado en el quicksort paralelo de la cátedra del martes 14 de noviembre.

Ud. encontrará el siguiente problema: todos los procesos hijo generarán exactamente los mismos subconjuntos llegando todos a la misma solución porque la función *gen_ruta_alea* entregará la misma ruta aleatoria en los 8 procesos hijos. Para lograr que cada proceso genere rutas distintas cambie la semilla para la función *random* en cada proceso hijo, antes de generar los subconjuntos aleatorios con:

*srandom(getUSecsOfDay()*getpid());*

Para evitar el warning de encabezado indefinido para *srandom* agregue esta primera línea en *viajante.c* (como está hecho en *test-viajante.c*):

#define _XOPEN_SOURCE 500

Se requiere que el incremento de velocidad (*speed up*) sea al menos un factor 1.5x. Cuando pruebe su tarea en su notebook asegúrese que posea al menos 2 cores y que esté configurado en modo máximo rendimiento. Si está configurado para ahorro de batería podría no lograr el *speed up* solicitado.

Instrucciones

Descargue *t7.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros en el directorio *T7* para recibir instrucciones acerca del archivo en donde debe programar su solución (*viajante.c*), cómo compilar y probar su solución y los requisitos que debe cumplir para aprobar la tarea.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *viajante.zip* generado por el comando *make zip*. Contiene los archivos *viajante.c* y *resultados.txt*. A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábado, domingo o festivos.