Tecnológico de Monterrey, Campus Monterrey
Departamento de Ciencias Computacionales
ANALISIS Y DISEÑO DE ALGORITMOS - Ing. Román Martínez M.
SEGUNDO EXAMEN DE PROGRAMACIÓN - 19 de Abril 2016

Nombre: _____Matrícula: _____

Instrucciones:

- Este es un examen que podrá ser resuelto desde la comodidad de tu casa, accesando el material del curso y cualquier otra referencia que consideres valiosa para que TÚ generes las respuestas a los problemas planteados.
- El examen deberá resolverse en forma INDIVIDUAL. Evita caer en la tentación de conversar con compañeros del grupo y/o ajenos al grupo. Este examen evalúa TUS conocimientos y habilidades y cualquier acción que realices para entregar un conocimiento o habilidad que no es tuyo, es una acción deshonesta que será penalizada fuertemente de acuerdo a nuestro reglamento. También recuerda que "tanto peca el que mata a la vaca como el que le estira la pata", así que evita hacerte cómplice de algún compañero en afán de ayudarlo o apoyarlo.
- Para resolver este examen puedes utilizar el compilador de C++ de tu preferencia para programar tus respuestas.
- Cada problema del examen deberá programarse en un archivo propio e independiente, respetando los nombres de archivo que se indican en la redacción del problema.
- Los archivos con los códigos programados se subirán en la página del curso en el espacio indicado en la sección de exámenes.
- Adicionalmente, se te pedirá un video que deberás generar como respuesta a este examen y lo deberás entregar en la página de Facebook del curso a través de un mensaje al *inbox* del profesor. Trata de que este video no exceda de 5 minutos de duración.
- Una vez accesado este examen en la página del curso en Bb, deberás dedicar el tiempo en horas continuas para trabajarlo y entregarlo (se registrará el tiempo de entrega). El video puede realizarse y entregarse posteriormente a este tiempo de acceso. Considera que tanto los archivos de respuesta como el video, deben estar entregados electrónicamente SIN EXCEPCIÓN antes de la sesión del viernes 22 de abril a las 9 am. Adicionalmente, en la sesión entregarás estas hojas impresas junto con la impresión de tus códigos.
- Para cualquier duda o aclaración del examen, envía un mensaje por *inbox* al profesor y se te dará respuesta lo más pronto posible.

PROBLEMA 1. *(30 puntos).*

El algoritmo de Dijkstra y el algoritmo de Prim, aunque resuelven problemas distintos, tienen en común que toman decisiones sobre los arcos de los nodos seleccionados hasta el momento en el proceso de solución. Esto significa que la implementación de ambos algoritmos es muy similar.

En el anexo 1 encontrarás el programa que contiene la implementación del algoritmo de Dijkstra para encontrar el camino más corto del primer nodo vértice hacia todos los demás vértices del grafo. Esta es la implementación que se revisó en clase. Copia, ejecuta el programa y analiza el código con la prueba que contiene.

Utilizando la estructura del algoritmo de Dijkstra ya implementado, modifica el código de este programa para convertirlo en el algoritmo de Prim, de tal manera que el programa sirva para encontrar el árbol de extensión mínima de un grafo no dirigido.

Comprueba el funcionamiento del nuevo programa probándolo con el mismo arreglo de datos que se usa en el programa de Dijkstra, pero adaptándolo para que sea un grafo no dirigido (modifica los valores que desees). Una vez que estés seguro que tu programa funciona, pruébalo con la matriz que se te enviará por mail y en base a tus resultados, responde las siguientes preguntas:

a) ¿Cuál es el costo mínimo para conectar todos los nodos del grafo que se te envió? _____
b) ¿Cuáles son los arcos que salen del tercer nodo y que forman parte del árbol de extensión mínima en el grafo que se te envió? _____

Como resultado de este problema deberás entregar lo siguiente:

1. El código con la solución implementada y que deberá llamarse P1EX2EM16.cpp. Documenta el código fuente con los comentarios que consideres convenientes para facilitar la revisión del examen, así como con tus datos personales (matrícula, nombre), y el comentario inicial de si tu programa funciona o no. Incluye también en esta documentación la lista de referencias en Internet que consultaste (si las hubo). Sube este archivo en la página del curso.

2. En el video deberás mencionar lo siguiente:
    a. Cuál es el cambio principal que hiciste al algoritmo de Dikjstra para convertirlo en el algoritmo de Prim.
    b. Indica si tu programa funciona correctamente y cuál es la solución que da el programa al grafo que se envió para la prueba. Para esto, muestra la pantalla de tu programa en ejecución.
    c. Cuánto tiempo tardaste en solucionar el caso y cualquier otro comentario que consideres importante considerar para la revisión.

PROBLEMA 2. *(30 puntos).*

El proceso de "rankear" un conjunto de datos es muy común en muchas aplicaciones. Cuando hemos usado *Kahoot*, se muestra el "top 5" de los participantes. Esta funcionalidad existe en muchos juegos en los que hay que ganar puntos y los jugadores se ordenan de mayor a menor, mostrándose sólo el "top X". Cuando haces una búsqueda en internet, también los resultados son "rankeados", mostrándose principalmente los sitios por mayor popularidad.

¿Cómo se puede obtener el "top X de un ranking" de un grupo de datos?

Seguramente pensarás en propuestas como estas:

1. Ordenar todos los datos de mayor a menor y de ahí obtener a los primeros "X".
2. Buscar el mayor entre todos los datos, el cual estaría en el "top 1", sacarlo de la lista, y repetir el proceso "X" veces para obtener todos los datos del "top X".
3. Meter los datos en un HEAP y realizar "X" eliminaciones de datos del HEAP que corresponderían a los datos del "top X".

Realiza un programa que implemente la opción 3. El HEAP deberá manejarse a través de la cola priorizada que provee STL. Busca en internet referencias que explican cómo funciona la cola priorizada de STL para que comprendas su funcionamiento. El programa deberá preguntarle al usuario el valor de "X" para mostrarle el "top X" de un conjunto de datos que estará declarado en el mismo programa como un arreglo constante. Se te enviará por correo electrónico el conjunto de datos de prueba.

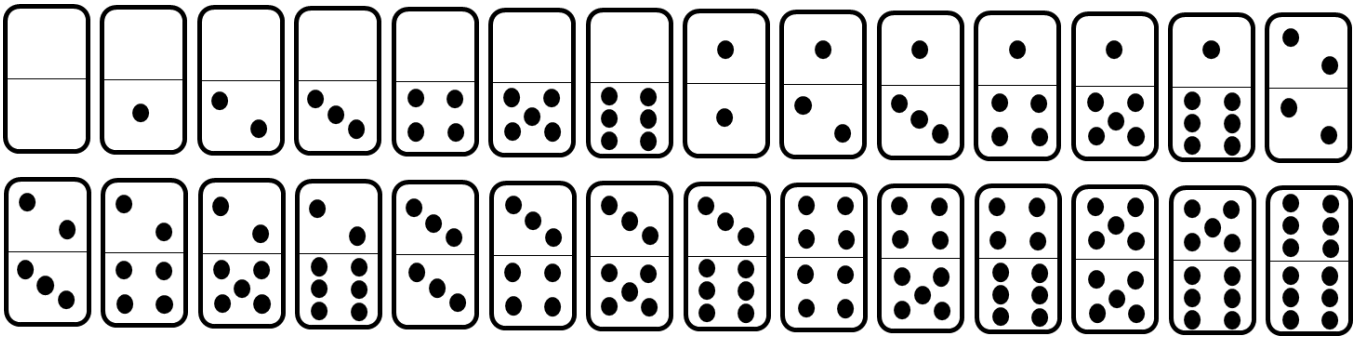
Adicionalmente, deberás reflexionar sobre la eficiencia de esta propuesta de solución comparada con las otras 2 opciones que se mencionaron. ¿Cuál es el orden de complejidad de tiempo de cada una de las propuestas? ¿cuál es la más recomendable utilizar cuando se tienen miles o millones de datos a "rankear"?

Como resultado de este problema deberás entregar lo siguiente:

1. El código con la solución implementada y que deberá llamarse P2EX2EM16.cpp. Documenta el código fuente con los comentarios que consideres convenientes para facilitar la revisión del examen, así como con tus datos personales (matrícula, nombre), y el comentario inicial de si tu programa funciona o no. Incluye en la documentación el orden de complejidad de los segmentos de código que consideres relevantes para la reflexión de cuál es la mejor opción de solución al problema. Incluye también en esta documentación la lista de referencias en Internet que consultaste (si las hubo). Sube este archivo en la página del curso.
2. En el video deberás mencionar lo siguiente:
    a. Cómo fue tu proceso de solución al caso y cómo fue tu proceso de entendimiento de la cola priorizada de STL.
    b. Indica si tu programa funciona correctamente y muestra la pantalla de tu programa en ejecución con los datos de prueba que se te enviaron para al menos 3 valores de "X" diferentes.
    c. Comenta brevemente cuál consideras la mejor opción para solucionar este caso y justifica tu respuesta.
    d. Cuánto tiempo tardaste en solucionar el caso y cualquier otro comentario que consideres importante considerar para la revisión.

PROBLEMA 3. *(40 puntos).*

El juego del domino seguramente te es conocido y consta de las siguientes 28 fichas:



Implementa un programa que sirva para verificar si las 28 fichas del domino pueden ser acomodadas en un tablero de 7 X 8 casillas, cada una de las cuales contiene un número entre 0 y 6. Una ficha se acomoda en el tablero al hacer coincidir sus números con los números de las 2 casillas contiguas seleccionadas.

Por ejemplo, un tablero en donde se pueden acomodar todas las fichas sin problema es:

0	3	0	2	2	0	2	3
1	5	6	5	5	1	2	2
3	4	1	4	5	4	4	4
6	6	1	0	5	2	3	0
4	0	3	2	4	1	6	0
1	4	1	5	6	6	3	0
1	2	6	5	5	6	3	3

El programa tendrá el arreglo del tablero como valor constante y te serán enviados algunos tableros de prueba. El programa dará como resultado la cantidad de combinaciones válidas en que las fichas se pueden acomodar en el tablero.

Como resultado de este problema deberás entregar lo siguiente:

1. El código con la solución implementada y que deberá llamarse P3EX2EM16.cpp. Documenta el código fuente con los comentarios que consideres convenientes para facilitar la revisión del examen, así como con tus datos personales (matrícula, nombre), y el comentario inicial de si tu programa funciona o no. Sube este archivo en la página del curso.
2. En el video deberás mencionar lo siguiente:
    a. Cómo fue tu proceso de solución al caso, explicando el diseño del árbol de búsqueda de soluciones que utilizaste (qué significan los niveles y qué significan los hijos).
    b. Si tu programa funciona correctamente y cuál es la solución que da el programa a las tres pruebas que se te enviaron. Para esto, muestra la pantalla de tu programa en ejecución, y los segmentos de código que consideres importantes explicar.
    c. Cuánto tiempo tardaste en solucionar el caso y cualquier otro comentario que consideres importante considerar para la revisión.

Para cualquier situación no aclarada en este documento, no dudes en preguntarle al profesor.

No olvides que en la sesión del viernes 22 de abril, deberás entregar estas hojas impresas con tu nombre y engrapadas con la impresión de los códigos que implementaste.

ANEXO 1 – Implementación del algoritmo de Dijkstra

```
#define tam 5
int datos[tam][tam] = {{0, 999, 4, 5, 999},
                  {3, 0, 5, 999, 999},
                        {999, 999, 0, 7, 1},
                        {999, 6, 999, 0, 9},
                              {6, 2, 999, 999, 0}};

void camino (int T[], int v)
{  if (T[v] != 1)
   {  camino(T, T[v]);
      cout << "-" << T[v];  }
}

void dijkstra (int W[tam][tam])
{    int L[tam+1], T[tam+1], R[tam+1], min, vmin;

     for (int i = 2; i<=tam; i++)
     {  L[i] = W[0][i-1];
        T[i] = 1;     }
     for (int x = 1; x < tam; x++)
     {      min = 999;
            for (int i = 2; i<= tam; i++)
          if ( 0 <= L[i] && L[i] <= min)
            {  min = L[i];  vmin = i; }
            for (int i=2; i<= tam; i++)
         if (L[vmin]+W[vmin-1][i-1] < L[i])
             {    L[i] = L[vmin]+W[vmin-1][i-1];
            T[i] = vmin; }
        R[vmin] = L[vmin];
        L[vmin] = -1;
     }
     cout << "\nCAMINOS MAS CORTOS\n";
     for (int i = 2; i <= tam; i++)
     { cout << "de 1 a " << i << ": 1";
       camino(T, i);
       cout << "-" << i << " cuesta " << R[i] << " unidades\n";
     }
}

int main()
{
    dijkstra(datos);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```