Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI-5651 Diseño de algoritmos I
enero-abril 2010, Profesor: Oscar Meza

Proyecto 1

Se quiere que implemente el algoritmo Greedy de Brélaz llamado DSATUR (artículo en la página web: Brelaz.pdf) para colorear los vértices de un grafo, incorporándole el refinamiento dado en el documento “interchangeProcedure.pdf “ que se encuentra junto a los artículos. Este algoritmo lo llamaremos “Brelaz+interchangeProcedure”.

De igual forma se quiere que implemente un algoritmo de “enumeración implícita” para hallar una coloración mínima de un grafo. El algoritmo procederá como sigue:

1) Se obtiene la coloración dada por la heurística BreLaz+interchangeProcedure, para obtener una cota superior (COTA_SUP) del número cromático.
2) Se obtendrá una cota inferior (COTA_INF) del número cromático obteniendo la máxima clique que se obtiene de aplicar BreLaz+interchangeProcedure N veces (donde N es el orden del grafo) partiendo de cada uno de los vértices del grafo, en lugar del vértice de mayor grado (ver sección 1 de Brelaz.pdf).
3) Se procederá a enumerar todas las posibles permutaciones de los N vértices, y para cada permutación se utilizará la heurística de coloración siguiente: partiendo de una permutación de los vértices (que determina un ordenamiento de los vértices) se procede a colorear los vértices en ese orden, con la estrategia “el color menor posible” (los colores son 1, 2, etc.). Note que si al colorear el siguiente vértice obtenemos igual número de colores que COTA_SUP, no tiene sentido seguir coloreando los vértices. Si al terminar de colorear todos los vértices el número de colores es COTA_INF, paramos el programa, pues hemos conseguido una coloración mínima. Note también que si hasta el vértice que hemos coloreado el grafo inducido por los vértices coloreados es una clique de mayor tamaño que COTA_INF, puede actualizar COTA_INF con este valor. Además, puede considerar el hecho que no será necesario considerar las permutaciones que comiencen con cualquier permutación de los vértices de esa clique pues dará la misma coloración.

Si el algoritmo de “enumeración implícita” tarda más de 5 minutos en una instancia dada, se aborta el programa. Puede agregar las mejoras que considere necesarias respetando el esquema de enumeración de todas las permutaciones.

Entre todos los grupos generarán 95 grafos pseudoaleatorios, con el formato de las instancias en http://mat.gsia.cmu.edu/COLOR/instances.html. El número de vértices n de los grafos será de 10 a 20 (de 2 en 2), y la densidad d será .1 a .9 (de .2 en .2), ver sección 3.1 de Brelaz.pdf. Para cada n y d generarán 3 grafos.

RESULTADOS:

En el informe del proyecto deberá incluir, aparte de lo señalado en las normas de presentación del informe:

- Un ejemplo de un grafo donde Brelaz+interchangeProcedure no da la solución óptima.
- Demostrar formalmente que en efecto el algoritmo implícito dado determina una coloración mínima.
- Los resultados de los dos algoritmos (Brelaz+interchangeProcedure y enumeración implícita) los darán como en las tablas del artículo: "Sequential and Backtracking algorithms for Graph coloring 2002.pdf", pero disgregados como en las tablas del artículo "generalized implicit graph enumeration algorithm for graph coloring 1982.pdf". Si procede, es decir, si el algoritmo de enumeración implícita termina, se deberá indicar por cada grupo (n,d) para cuantos grafos en el grupo el algoritmo de Brelaz+interchange da la solución óptima.
- Las normas de presentación del informe y las reglas de documentación del código están en http://www.ldc.usb.ve/~meza/ci-5651/e-a2010/ . Es obligatorio documentar bien el código.
- Debe programar en C en ambiente UNIX.

Comentarios finales:

- En la página http://mat.tepper.cmu.edu/COLOR/color.html puede ver varias aplicaciones del problema de coloración de grafos.
- En la página http://webdocs.cs.ualberta.ca/~joe/Coloring/Colorsrc/index.html puede encontrar varios programas en C de coloración de grafos. Podría comparar sus resultados, por ejemplo con el algoritmo "Backtrack DSATUR"
- Para que tomen el tiempo con la mayor precisión posible, está una función "gettimeofday " de C que obtiene el tiempo, aquí está un ejemplo de su uso:

```
#include <sys/time.h>
struct timeval t_p;
…..
  if (!gettimeofday(&t_p,NULL))
     TIEMPOINICIAL = (double) t_p.tv_sec + ((double) t_p.tv_usec)/1000000.0;
  else printf("\n mal tiempo \n");

.....llamada al programa.....

  if (!gettimeofday(&t_p,NULL))
     TIEMPOFINAL = (double) t_p.tv_sec + ((double) t_p.tv_usec)/1000000.0;
  else printf("\n mal tiempo \n");
printf("tiempo EN SEGUNDOS de ejecución del programa: #1.4f",
  TIEMPOINICIAL – TIEMPOFINAL);
```

- **IMPORTANTE:** EL INFORME DEL PROYECTO DEBE SER ENTREGADO EN UN ARCHIVO .PDF ESCRITO CON UN PROCESADOR DE TEXTO (ES DECIR, NO A MANO). ME DEBEN ENTREGAR EL INFORME EN PAPEL Y ENVIARME POR E-MAIL EL ARCHIVO PDF EL DÍA DE LA ENTREGA, AL IGUAL QUE UN ARCHIVO CON EL CODIGO FUENTE, EL MAKEFILE Y LAS INSTANCIAS DE GRAFOS DE MANERA DE YO COMPILARLO Y EJECUTARLO.

- FECHA DE ENTREGA: 17 DE FEBRERO