DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Algoritmos y Estructura de Datos III
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Iván Arcuschin	678/13	iarcuschin@gmail.com
Martín Jedwabny	885/13	martiniedva@gmail.com
José Massigoge	954/12	jmmassigoge@gmail.com
Lucas Puterman	830/13	lucasputerman@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Ejercicio 1 - Demostración

1.1. Definiciones

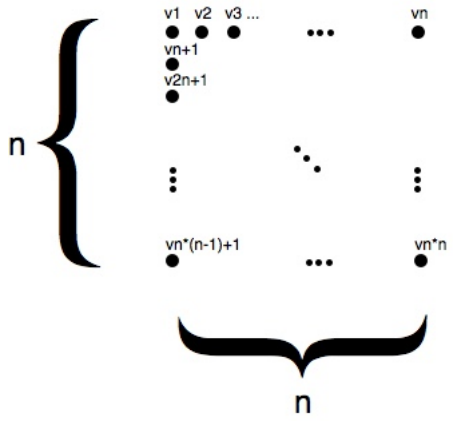
Sea G=(V,E) un grafo simple:

- Definición 1: D $\subseteq$ V es un **conjunto dominante** (CD) $\iff$ $\forall$ v $\in$ V, v $\in$ D ó $\exists$ w $\in$ D tal que (v,w) $\in$ E (tiene un vecino en D).
- Definición 2: D $\subseteq$ V es un **conjunto independiente** (CI) $\iff$ $\forall$ v,w $\in$ D, (v,w) $\notin$ E.
- Definición 3: D $\subseteq$ V es un **conjunto independiente dominante** (CID) $\iff$ D es dominante e independiente.
- Definición 4: D $\subseteq$ V es un **conjunto independiente dominante mínimo** (CIDM) $\iff$ D es el conjunto independiente dominante de V con menos nodos. Es decir que $\forall$ D' $\subseteq$ V tal que D' es independiente dominante, #(D) $\leq$ #(D').
- Definición 5: D $\subseteq$ V es un **conjunto independiente maximal** (CIMax) $\iff$ D es independiente y $\nexists$ D' $\subseteq$ V independiente tal que D $\subset$ D' ($\subset$ estricto).

1.2. Ejercicio A

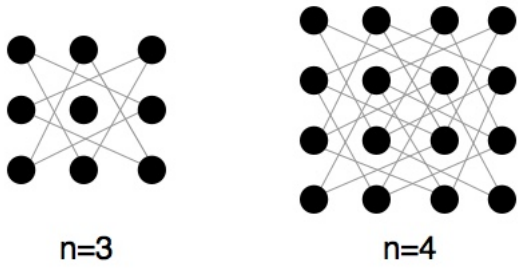
Relacionar el problema de CIDM con el problema 3 del TP 1 (i.e., similitudes y diferencias):

Si definimos a G=(V,E) grafo simple para que represente al tablero de ajedrez que se forma en el ejercicio 3 del tp1, los nodos serían una cuadricula de la pinta (sin considerar los ejes, eso lo hacemos después):



Donde 'n' es el parámetro del ejercicio que indica el tamaño de lado del tablero.

Ahora, como el ejercicio consiste en poner la mínima cantidad de caballos para que todas las casillas tengan un caballo o bien estén amenazadas, podemos representar las 'amenazas' como los ejes del grafo, por ejemplo:



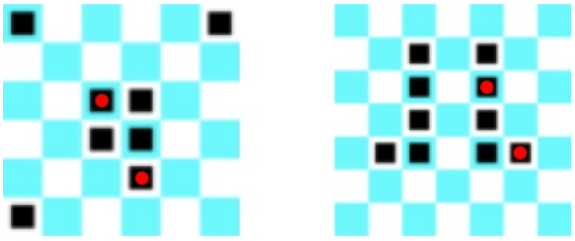
Observación: como cualquier posición x que amenaza a otra posicion 'y' sería amenazada si pongo un caballo en 'y', podemos representar el problema con un grafo simple y no un digrafo.

Entonces los ejercicios son similares ya que lo que estamos buscando en el ejercicio del TP1 es la mínima cantidad de caballos para que todas las posiciones (nodos) tengan un caballo o esten amenazadas (dominadas).

Es decir, si los caballos son un conjunto D de vértices en V, queremos hallar #(D) tal que:

- $\forall$ v $\in$ V, v $\in$ D o $\exists$ w $\in$ D tal que (v,w) $\in$ E $\iff$ D es dominante.
- $\forall$ D' $\subseteq$ V tal que D' es dominante, #(D) $\leq$ #(D') $\iff$ D es mínimo entre los conjuntos dominantes de V.

Por lo tanto el problema consiste en hallar un conjunto dominante mínimo pero, a diferencia del problema en este TP, puede o no ser independiente. Esto se debe a que existen respuestas para el problema del TP1 donde hay caballos en ciertas posiciones que se amenazan entre sí:



Observación: estos graficos fueron extraidos de la página: http://home.earthlink.net/ morgenstern/solution/knsols1.htm. El primero representa una solución óptima del ejercicio del TP1 con un tablero de 6x6 y la segunda imágen es para un tablero de 7x7. En ambas, se marcan con un punto negro los casilleros que tienen un caballo y, como vemos señalado en rojo, hay caballos en posiciones que se amenazan entre sí.

1.3. Ejercicio B

Demostrar que todo conjunto independiente maximal es un conjunto dominante:

Sea G=(V,E) un grafo simple y D $\subseteq$ V un conjunto independiente maximal, quiero ver que D es un conjunto dominante.
Supongamos (por absurdo) que D no es un conjunto dominante:

$\Rightarrow \exists$ v $\in$ V tal que v $\notin$ D $\wedge \forall$ w $\in$ D, (v,w) $\notin$ E (no es vecino de ningún nodo en D).

$\Rightarrow$ D$\cup${v} es independiente ya que como D es independiente sucede que $\forall$ w $\in$ D$\cup${v} $\nexists$ x $\in$ D tal que (x,w) $\in$ E.

$\Rightarrow$ Absurdo! pues D era maximal y $\exists$ D' = D$\cup${v} tal que D' es independiente y D $\subset$ D', asi contradiciendo el hecho de que D es un conjunto independiente maximal.

Este absurdo vino de suponer que D no era dominante.
Por lo tanto, si D es un conjunto independiente maximal $\Rightarrow$ D es dominante.

1.4. Ejercicio C

Describir situaciones de la vida real que puedan modelarse utilizando CIDM:

- **El turista:** tenemos un conjunto de ciudades que forman un grafo simple conexo donde los nodos son las ciudades y dos nodos estan conectados si y solo si las ciudades son vecinas. Suponiendo que queremos conocer tantas culturas distintas como sea posible, decidimos que no queremos visitar ningun par de ciudades vecinas ya que sus culturas son muy similares. Como el problema que proponemos consiste en hallar un conjunto mínimo de ciudades (nodos en el mapa) tal que toda ciudad sea visitada o bien una ciudad vecina sea visitada pero no ambas (el conjunto de nodos sea independiente y dominante), podemos decir que estamos buscando un CIDM.
- **Spamear una red social:** supongamos que tenemos un virus que se ocupa de spamear Facebook de manera que un mensaje sea propagado por toda la red. Asumimos que los usuarios de Facebook son nodos en un grafo simple conexo y que dos nodos estan conectados si y solo si esos dos usuarios son amigos en la red. Ahora, el virus no quiere ser descubierto, asi que lo que debe hacer es infectar la menor cantidad de cuentas que no sean amigas, ya que si spamea demasiadas cuentas o un par de cuentas que son amigas, sería mucho más fácil de descubrirlo. Luego, estamos buscando el mínimo conjunto de nodos en la red de usuarios tal que todos los usuarios vean el mensaje (dominación) y dos amigos no sean infectados a la vez (independencia), el cual es un CIDM.
- **Cámaras de seguridad en un barrio:** tenemos casas en un barrio que representan los nodos de un grafo simple conexo, donde dos nodos estan conectados si y solo si sus casas respectivas con vecinas. Queremos poner cámaras de seguridad en el barrio de manera que toda casa tenga una cámara de seguridad o la tenga una casa vecina, ya que el rango de cobertura de la cámara es amplio y puede filmar la casa donde está y también las vecinas (dominación). No queremos poner cámaras en dos casas vecinas (independencia) para que no sea tan notorio y arruine el paisaje. A la vez, tenemos un presupuesto limitado, por lo cual queremos poner poner tan pocas cámaras como sea posible (minimalidad). Por lo tanto, el conjunto de casas que estamos buscando para ponerles cámaras es un CIDM.

2. Ejercicio 2 - Algoritmo exacto

Diseñar e implementar un algoritmo exacto para CIDM.

2.1. Ejercicio A

Explicar detalladamente el algoritmo implementado. Elaborar podas y estrategias que permitan mejorar los tiempos de resolucion.

2.1.1. Estrategia

El algoritmo implementado se basa en la Técnica Algorítmica de *Backtracking*.

En primer lugar notese que un conjunto dominante trivial es el conjunto de todos los nodos del grafo. Si bien este conjunto en principio no es independiente (a menos que $X(G) = \emptyset$), es el conjunto dominante más grande posible (no hay ningún nodo que quede afuera, es decir: $V(G) \setminus D = \emptyset$).

Luego, la idea principal del algorítmo es empezar con este primer conjunto dominante e ir sacando nodos recursivamente mientras chequeamos dominancia e independencia. Cada vez que encontramos un conjunto dominante e independiente, nos fijamos su cardinal. Si el cardinal de este nuevo conjunto es menor que el mínimo hasta ese momento nos quedamos con el nuevo y lo guardamos como mínimo. En caso de que el cardinal sea mayor, seguimos quitando nodos para encontrar un conjunto más chico.

2.1.2. Podas

Antes de mostrar cuales son las podas que utilizamos en el algoritmo, veamos algunos Lemas.

Lema 2.1. *Sea C un conjunto dominante e independiente de un grafo G. Entonces, cualquier subconjunto de C (distinto de C) es no-dominante respecto a G.*

Demostración. Veamos que vale por absurdo. Supongamos H un subconjunto de C tal que $H \subset C$ y H es dominante de G. Como H está estrictamente incluido en C, entonces $\exists\ v \in C$ tal que $v \notin H$.

Ahora, como C es independiente, no hay nodos de C que sean adyacentes en G, en particular: $\forall\ w \in C,\ (v, w) \notin X(G)$.

Entonces, $v \in V(G)$ pero $v \notin H$ y $\forall\ w \in H \subset C,\ (v, w) \notin X(G)$. Luego, H no es dominante, ya que el nodo v no está en H ni tiene algún vecino que esté en H. Absurdo. □

Lema 2.2. *Sea C un conjunto no-dominante respecto a G. Entonces, cualquier subconjunto de C es no-dominate respecto a G.*

Demostración. Trivial. Sea H un subconjunto de C ($H \subseteq C$).

Como C es no-dominante, $\exists\ v \in V(G)$ tal que $v \notin C$ y $\forall\ w \in C,\ (v, w) \notin X(G)$.

Pero como $H \subseteq C$, $v \notin H$ y $\forall\ w \in H,\ (v, w) \notin X(G)$.

Luego H es no-dominante respecto a G. □

Usando los resultados de estos dos lemas podemos optimizar el algortimo cortando ramas en el árbol de recursión.

1. Por el primer Lema, cada vez que encontramos un conjunto dominante e independiente podemos ahí mismo devolver el que tenga menor cardinal entre ese conjunto y el mínimo encontrado hasta ese momento. Esto es así ya que sabemos que cualquier subconjunto del nuevo conjunto es no-dominante.

2. Por el segundo Lema, al momento de sacar un nodo de un conjunto podemos chequear si el subconjunto es dominante o no. En caso de que no lo sea, ni siquiera lo procesamos, ya que no es dominante y ninguno de sus subconjuntos lo será.

2.1.3. Pseudocódigo

```
funcion resolver:
    Creamos un vector de n elementos, llenandolo con los nodos desde 0 a n-1.
    llamamos a resolver_aux pasandole como parametro la matriz de adyacencia, el vector
        recién creado y la cantidad de nodos en el grafo.

funcion resolver_aux:
    Llamemos dom al conjunto dominante pasado por parametro.
    Llamemos cidm al conjunto dominante e independiente con menor cardinal encontrado
        hasta ahora.
    Si el cidm tiene tamaño 1 hacer
        Devuelvo cidm
    Sino hacer
        // Chequeamos si dom es independiente:
        Para i desde 0 hasta |dom| hacer:
            Para j desde i+1 hasta |dom| hacer:
                Si matriz_adyacencia[dom[i]][dom[j]] == TRUE hacer
                    dom NO es independiente
        Si es independiente hacer
            Devolvemos el conjunto con menor cardinal entre cidm y dom.
    Para i desde 0 hasta |dom| hacer
        Crear un vector nuevo llamado copia con los mismos nodos que dom.
        Borrar el nodo en la posicion i del vector copia.
        // Chequeamos si el conjunto copia es dominante:
        Para i desde 0 hasta n hacer:
            Si i no pertenece a copia hacer
                Bool adyacente = FALSE
                Para j desde 0 hasta |copia| hacer:
                    Si matriz_adyacencia[i][copia[j]] == TRUE hacer
                        adyacente = TRUE
                Si adyacente == FALSE hacer
                    copia NO es dominante
        Si copia es dominante hacer
            Hacer un llamado recursivo a resolver_aux, pasando como parametro la matriz
                de adyacencia, cidm, copia y la cantidad de nodos en el grafo.
            Llamemos nuevo_cidm al conjunto que devuelve el llamado recursivo.
            Devolvemos el conjunto con menor cardinal entre cidm y nuevo_cidm.
```

2.2. Ejercicio B

Calcular el orden de complejidad temporal de peor caso del algoritmo.

2.3. Ejercicio C

Realizar una experimentación que permita observar los tiempos de ejecución del algoritmo en función de los parámetros de la entrada y de las podas y/o estrategias implementadas.

3. Ejercicio 3 - Heurística constructiva golosa

4. Ejercicio 4 - Heurística de búsqueda local

5. Ejercicio 5 - MetaHeurística GRASP

6. Ejercicio 6 - Experimentación final