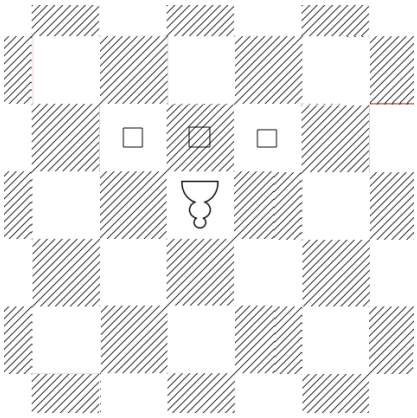
Parcial de Programación 3
30 de noviembre de 2023

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (16 puntos)

Consideramos un juego de mesa que se desarrolla en un tablero de tamaño $n \times n$. Identificamos cada casilla por un par (i, j) $(1 \leq i \leq n, 1 \leq j \leq n)$ donde i y j indican fila y columna, respectivamente. Un *sargento* es una pieza que puede moverse de una casilla (i, j) a cualquier casilla del conjunto

$$\Big\{(i-1, j-1), (i-1, j), (i-1, j+1)\Big\} \bigcap \{1, 2, \ldots, n\}^2 .$$



Para el sargento de la figura, las casillas de ese conjunto se muestran destacadas con un pequeño rectángulo. Notar que la intersección con $\{1, 2, \ldots, n\}^2$ en la ecuación anterior tiene la función de excluir las casillas fuera de los límites del tablero.

Cada casilla tiene asociado un costo $C(i, j)$, que representa la dificultad del terreno (por ejemplo, es más difícil moverse por bosques que por terreno llano).

Salir de la casilla (i, j) tiene costo $C(i, j)$, de modo que el costo de una secuencia de movimientos $(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)$ es $C(i_1, j_1) + \ldots + C(i_{m-1}, j_{m-1})$.

Nuestro objetivo es obtener una secuencia de movimientos de costo mínimo para llevar al sargento desde la casilla inferior izquierda a alguna casilla de la fila superior (es decir, desde $(n, 1)$ a $(1, k)$ para algún k).

(a) Especifique una relación de recurrencia que calcule lo pedido. Justifique explicado el significado de cada término.

Sugerencia: Considere $OPT(i, j)$ como el mínimo costo de moverse hasta la casilla (i, j).

(b) Dé un algoritmo *eficiente iterativo* usando la técnica de programación dinámica, consistente con la parte anterior, para computar el costo mínimo de una secuencia de movimientos que lleva al sargento desde la casilla inferior izquierda a alguna casilla de la fila superior. Tome como dada una rutina C que retorna el costo de cada casilla. Por comodidad, puede suponer $C(i, 0) = \infty$, $C(i, n+1) = \infty$ para toda fila i $(1 \leq i \leq n)$.

(c) Dé un algoritmo *eficiente* para obtener una secuencia de movimientos de costo mínimo que lleva al sargento desde la casilla inferior izquierda a alguna casilla de la fila superior. Puede usar estructuras de datos construidas en la parte anterior.

Solución:

(a) $OPT(n, 1) = 0$
$OPT(n, i) = \infty$ (para $1 < i \leq n$)
$OPT(i, j) = min_{\ell \in \{j-1, j, j+1\} \cap \{1 \ldots n\}} \{OPT(i+1, \ell) + C(i+1, \ell)\}$ (para $1 \leq i < n, 1 \leq j \leq n$)

En la fila n, inicialmente el sargento se ubica en la casilla de más a la izquierda (el costo de llegar es nulo, pues se alcanza con un camino de largo 0). Cualquier otra casilla de esta fila es inalcanzable por lo que marcamos el costo como infinito.

Luego, en una casilla arbitraria (i, j) de cualquier otra fila, el sargento puede venir de las tres (a lo sumo) casillas inferiores (diagonal izquierda, contigua inferior y diagonal derecha). El costo óptimo del camino en cada caso viene dado por el costo de salir de cada casilla, añadido al costo óptimo de alcanzarlas en cada caso.

Notar que una forma de considerar los índices borde es seguir la sugerencia de la parte (b), tomando una fila y columnas extra y considerando siempre las tres casillas inferiores.
Una recurrencia alternativa en ese contexto es, por ejemplo, la siguiente:
$OPT(n,1) = 0$
$OPT(n,i) = \infty$ (para $1 < i \leq n$)
$OPT(i,0) = \infty$ (para $1 \leq i \leq n$)
$OPT(i,n+1) = \infty$ (para $1 \leq i \leq n$)
$OPT(i,j) = min_{\ell \in \{j-1,j,j+1\}}\{OPT(i+1,\ell) + C(i+1,\ell)\}$ (para $1 \leq i < n, 1 \leq j \leq n$)

(b)

```
1  program OPTSargento(n, C[][]){
2    OPT(n, 1) := 0;
3    for j from 2 to n
4      {
5        OPT(n, j) := ∞
6      }
7    for i from 1 to n
8      {
9        OPT(i, 0)   := ∞;
10       OPT(i, n+1) := ∞
11     }
12   for i from n−1 downto 1
13     {
14       for j from 1 to n
15         {
16           OPT(i,j) := min(OPT(i+1, j-1) + C[i+1, j-1],
17                           OPT(i+1, j)   + C[i+1, j],
18                           OPT(i+1, j+1) + C[i+1, j+1]);
19         }
20     }
21   return min(OPT(1,i))_{i∈{1...n}}
22 }
```

(c)

```
program SolSargento(n, C[][]){
  opt      := OPTSargento(n, C);
  sol      := emptyList();
  curr_col := argmin_n (OPT(1, n));
  for i from 1 to n-1
    {
      sol.add((i,curr));
      if opt == OPT (i+1, curr_col-1) + C(i+1, curr_col-1)
        {
          opt  := opt - C(i+1, curr_col-1);
          curr_col := curr_col-1
        }
      else if opt == OPT (i+1, curr_col) + C(i+1, curr_col)
        {
          opt  := opt - C(i+1, curr_col)
        }
        else // if opt == OPT (i+1, curr_col+1)
             //          + C(i+1, curr_col+1)
        {
          opt  := opt - C(i+1, curr_col+1);
          curr_col := curr_col + 1
        }
    }
  sol.add((n,curr)); // (n,1)
  return sol;
}
```

Ejercicio 2 (17 puntos)

Varios directivos de algunas empresas fueron encontrados culpables de ciertos delitos, y como parte de su pena deberán realizar trabajos comunitarios.

Hay un conjunto de h empresas involucradas, $E = \{e_1, \ldots, e_h\}$, y cada empresa $e_i \in E$ tiene una cantidad d_i de directivos culpables.

Hay un conjunto de k trabajos posibles, $T = \{t_1, \ldots, t_k\}$, y cada trabajo $t_i \in T$ tiene un cupo de c_i trabajadores que pueden ser asignados a t_i.

Para disminuir el riesgo de más ilícitos, se quiere evitar que a dos o más directivos de una misma empresa les toque realizar el mismo trabajo. Para esto se quiere asignar a cada empresa $e_i \in E$ un conjunto de d_i trabajos distintos, uno para cada uno de sus directivos.

Una asignación de trabajos a empresas es *válida* si toda empresa tiene una cantidad de trabajos asignados suficiente para todos sus directivos y no se excede el cupo de ningún trabajo.

(a) Proponga un algoritmo que reporte si es o no posible hacer una asignación válida (no se pide que se exhiba una asignación válida, solo que se reporte si existe). El tiempo de ejecución debe ser de orden polinomial en h y k.

(b) Demuestre que si existe una asignación válida, entonces su algoritmo reporta correctamente que sí es posible hacer tal asignación.

(c) Demuestre que el tiempo de ejecución de su algoritmo es de orden polinomial en h y k.

Solución:

Una instancia del problema consiste en un conjunto E de cardinalidad h, donde para cada elemento e de E está definido el entero positivo d_e, y un conjunto T de cardinalidad k, donde para cada elemento t de T está definido el entero positivo c_t.

Sea $n = \sum_{e \in E} d_e$.

Se pretende determinar si es posible obtener una colección R de pares (e, t), con e en E, y t en T, tal que

- cada par (e, t) ocurre a lo sumo una vez,
- para cada e en E la cantidad de pares $(e, \cdot)$ en R es igual a d_e,
- para cada t en T la cantidad de pares $(\cdot, t)$ en R es menor o igual a c_t.

El segundo item implica que la cardinalidad de R es n.

(a)

1. Se construye una red de flujo $G = (V, E)$ con un vértice fuente s, un vértice sumidero z, un vértice e_i, $1 \leq i \leq h$, por cada elemento de E y un vértice t_j, $1 \leq j \leq k$, por cada elemento de T. Hay una arista (s, e_i) con capacidad d_{e_i} para cada vértice que representa a un elemento de E, una arista (t_j, z) con capacidad c_{t_j} para cada vértice que representa a un elemento de T, y una arista con capacidad 1 para cada par de vértices (e_i, t_j) donde e_i representa un elemento de E y t_j representa un elemento de T.
2. Se aplica `Ford Fulkerson` a la red de flujo construída y se obtiene el flujo f asignado.
3. Se calcula el valor de flujo $v(f) = \sum_{1 \leq i \leq h} f(s, e_i)$.
4. Si $v(f) = n$ se reporta que se puede hacer la asignación.

   Si se pidiera una asignación, el conjunto R consistiría en un par por cada una de las aristas (e_i, t_j) para las cuales el flujo asignado es 1.

   En otro caso se reporta que no se puede hacer la asignación.

(b)

Si el algoritmo reporta que se puede hacer una asignación, da como resultado un conjunto R que es una asignacíon válida.

El algoritmo reporta que se puede hacer una asignación solo si en el cuarto paso se evaluó que el valor del flujo obtenido es igual a $n = \sum_{e \in E} d_e$.

El valor del flujo es por definición la cantidad de flujo generado en s. Como por construcción de G hay exactamente una arista saliente de s por cada e de E, que tiene capacidad d_e, y como, por la propiedad de capacidad de los flujos, el flujo en una arista no puede superar la capacidad de la arista, el resultado obtenido implica que cada arista saliente de s tiene flujo igual a su capacidad **(a.1)**.

Por integralidad del resultado de la aplicación de `Ford Fulkerson` (KyT (7.2)), y como las aristas (e_i, t_j) tienen capacidad 1, el flujo asignado a cada una de esas aristas es o bien 1, o bien 0 **(a.2)**.

Otra vez por integralidad del resultado de la aplicación de `Ford Fulkerson` y por la propiedad de capacidad, el flujo asignado a cada arista (t_k, z) es un entero menor o igual a c_{t_j} **(a.3)**.

En R se incluyen los pares que corresponden a aristas (e_i, t_j) que tienen asignado flujo 1. Veamos que R cumple las tres propiedades requeridas.

- Por construcción cada par se incluye a lo sumo una vez.
- Por propiedad de conservación de los flujos, y por **(a.1)** y **(a.2)**, la cantidad de aristas $(e_i, .)$ con flujo 1 es d_{e_i}, y esas son las aristas que se incluyen en R.
- Por propiedad de conservación de los flujos, y por **(a.3)** y **(a.2)**, la cantidad de aristas $(., t_j)$ con flujo 1 es menor o igual a c_{t_j}, y esas son las aristas que se incluyen en R.

Por lo tanto si el algoritmo reporta que hay solución el resultado es una asignación válida.

Si es posible hacer alguna asignación válida, el algoritmo da como resultado una de ellas.

Si hay solución válida, sea R una de ellas. En la red de flujo con flujo inicialmente 0 en cada arista, asignamos una unidad de flujo a cada arista (e_i, t_j) que corresponde a cada par de R. Por cada una de esas aristas sumamos una unidad de flujo a la arista (s, e_i) y una unidad de flujo a la arista (t_j, z), logrando que la asignación de flujo cumpla la propiedad de conservación en cada uno de los vértices.

Veamos que también se cumple la propiedad de capacidad en cada arista. A las aristas (e_i, t_j) se les asigna una unidad una única vez si corresponden a pares de R, y ninguna en otro caso. A cada arista (s, e_i) se le asigna flujo d_{e_i} porque e_i corresponde a un elemento de E que ocurre esa cantidad de veces en R, porque R es una asignación válida. De manera análoga, a cada arista (t_j, z) se le asigna flujo a lo sumo c_{t_j}.

Entonces la asignación es un flujo, y su valor es n.

El algoritmo obtiene el flujo resultado de la aplicación de `Ford Fulkerson`, cuyo valor es máximo (KyT 7.10), por lo que es mayor o igual al valor del que construímos, que es n. Como la capacidad del corte $(\{s\}, V \setminus \{s\})$ es también n, y como el valor de cualquier flujo es menor o igual a la capacidad de cualquier corte (KyT 7.8) se concluye que el valor del flujo obtenido por el algortimo es n, por lo que en el paso 4 reporta que la asignación es posible.

(c)

1. La red de flujo tiene $h + k + 2$ vértices, y $h + k + hk$ aristas, por lo que se construye en tiempo $\Theta(hk)$.
2. El corte $(\{s, e_1, \ldots, e_h\}, \{t_1, \ldots, t_k, z\})$ tiene capacidad hk porque esa es la cantidad de aristas y todas tienen capacidad 1. Por lo tanto hk es una cota superior de la cantidad de iteraciones de la aplicación de `Ford Fulkerson`, cada una de las cuales tiene $O(hk)$ como cota superior, por lo que el segundo paso es $O(h^2k^2)$ (KyT 7.5).
3. El tercer paso consiste en una suma que tiene tiempo $\Theta(h)$.

Si se pidiera, la construcción de R sería $O(hk)$, ya que a lo sumo se agregan todas las aristas (e_i, t_j).

Como tenemos una cantidad constante de pasos el tiempo está determinado por el de mayor orden, que es $O(h^2k^2)$.

Ejercicio 3 (17 puntos)

Usted está a cargo de optimizar la cobertura de una red inalámbrica en un área determinada. En esta área hay un conjunto $D = \{d_1, d_2, \ldots, d_n\}$ de dispositivos inalámbricos a los que se desea dar conectividad a través de k *puntos de accesso*. Hay también un conjunto de lugares $A = \{a_1, a_2, \ldots, a_\ell\}$ con infraestructura donde se podría instalar un punto de acceso. Por ejemplo, en la figura 1 existen tres lugares posibles.

Para cada dispositivo $d \in D$ y cada lugar $a \in A$ conocemos el ancho de banda $B(d, a)$ (un número no negativo) que tendría una comunicación entre d y un punto de acceso ubicado en a. Para un subconjunto $S \subseteq A$ y $d \in D$, decimos que S da *cobertura* a d si $B(d, a) > 0$ para algún $a \in S$.

El problema *ACCESO* es el siguiente: dados D, A, B y un natural k, determinar si es posible dar cobertura a todos los dispositivos de D usando no más de k puntos de acceso en ubicaciones de A. Una instancia del problema se representa mediante tres números naturales, n, ℓ, k y una matriz numérica de dimensiones $n \times \ell$ que representa a B.

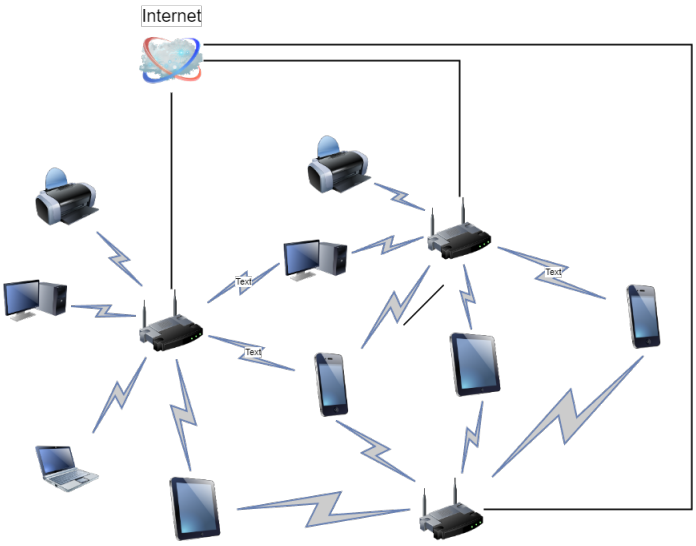


Figura 1: Ejemplo de red inalámbrica de accesso.

Se pide:

(a) Demuestre que *ACCESO* pertenece a $\mathcal{NP}$.

(b) Demuestre que *ACCESO* es $\mathcal{NP}$-Completo.
Sugerencia: Considere el problema *Cobertura de Conjuntos (Set Cover)*.

Solución:

(a) Para demostrar que ACCESS está en NP, necesitamos demostrar que existe un certificador eficiente C para el problema. Es decir que:

- C es un algoritmo de tiempo polinómico que toma dos argumentos de entrada s y t.
- Existe una función polinómica p tal que para cada *string* s, s es una instancia *sí* del problema si y sólo si existe un *string* t tal que $|t| \leq p(|s|)$ y $C(s, t) = s$.

Un a instancia s del problema ACCESS puede representarse con un *string* que codifique la matriz $B(d, a)$ (alcanza con poner sus filas una detrás de la otra en el en *string*) y asumiendo que los dipositivos están identificados por enteros en el rango $1 \ldots n$.

El certificado t lo definimos como un conjunto de lugares donde ubicar los puntos de accesso. Este conjunto lo representamos como vector de k índices $i_1, i_2, \ldots, i_k$, cada indice, en el rango de 1 a l, representa el lugar donde ubicar un punto de acceso, por ejemplo $t = \{a_2, a_5, a_6\}$. Ese vector es fácilmente representable como un *string* de enteros; además, el tamaño de t es siempre menor al de la matriz por lo que se cumple que $|t| \leq p(|s|)$, con p polinomial.

Podemos crear un certificador $C(s, t)$ que compruebe si s proporciona cobertura a cada dispositivo recorriendo las filas de la matriz $B(d, a)$ y verificando que para cada índice a_i en t exista al menos un valor $B(d, a_i) > 0$. Este proceso de verificación lleva un tiempo polinómico, ya que sólo

tenemos que comprobar los valores para cada par de dispositivos y ubicacion en t. Finalmente, podemos ver que si $C(s,t) = TRUE$, entonces $B(d,a_i) > 0$ para al menos un a_i y por lo tanto cada dispositivo esta a alcance de algun punto de acceso y, por lo tanto, s es una instancia *sí* de ACCESO.

(b) Para demostrar que ACCESO es NP-Completo, debemos probar que ACCESO $\in$ NP, cosa que ya hicimos en (a), y probar que para todo Y $\in$ NP, Y $\leq_P$ ACCESO.

Para esto último, primero reduciremos un problema NP-completo conocido, *Set Cover*, a ACCESO, y luego concluimos que, por la propiedad 8.14 del Kleinberg, cualquier problema NP es reducible a ACCESO y por lo tanto ACCESO es NP-completo.

Para la reducción, tomamos una instancia de *Set Cover* dada por un conjunto U, un natural r, y m subconjuntos de U, $S_1, S_2, \ldots, S_m$, y a partir de ella construimos una instancia de *ACCESO* tomando $D = U$, (es decir $n = |U|$), $\ell = m$, $k = r$, y definiendo $B(d,a_i) = 1$ si $d \in S_i$ y $B(d,a_i) = 0$ si $d \notin S_i$. Todas estas construcciones se hacen, evidentemente, en tiempo polinómico.

Vamos a probar que:

(a) Si s_{SC} es una instancia "sí" de *Set Cover*, entonces s_A, obtenida con la reducción mencionada, es una instancia "sí" de ACCESO.

(b) Si s_A es una instancia "sí" de ACCESO, obtenida con la reducción mencionada a partir de la instancia s_{SC} de *Set Cover*, entonces s_{SC} es una instancia "sí" de ese problema.

Demostración de (a): si s_{SC} es una instancia "sí" de *Set Cover*, entonces, dado el conjunto U de n elementos, una colección $S_1, \ldots, S_m$ de subconjuntos de U existe una colección C_r de a lo sumo r de estos conjuntos cuya unión es igual a todo el conjunto U. Por lo tanto, en nuestra reducción, cada dispositivo tiene al menos un valor de $B(d,a_i) > 0$ y por lo tanto tiene cobertura de alguno de los puntos de acceso. Esto hace a nuestra reducción s_A una instancia "sí" de ACCESO.

Demostración de (b): si s_A es una instancia "sí" de ACCESO, entonces existe un conjunto A_k de como mucho k puntos de acceso tales para cualquier $d \in D$, $B(d,a_i) > 0$, con $a_i \in A_k$. Esto quiere decir que enuestra reducción, existe existe una colección C_r de a lo sumo r subconjuntos de U, tales que $d \in S_i$, con $S_i \in C_r$ y por lo tanto la instancia s_{SC} es una instancia "sí" de *Set Cover*.