CC4102 - Diseño y Análisis de Algoritmos
Problemas de Competitivos, Probabilísticos y Aproximados

Prof. Gonzalo Navarro; Aux. Mauricio Quezada

8 de noviembre de 2012

1 Competitivos

1.1 El problema de los k servidores

Considere el escenario donde tiene k puntos (*servidores*) en un *espacio métrico* (donde está definida una función de distancia d: simétrica, no-negativa y que cumple la desigualdad triangular) y una secuencia de puntos (*peticiones*) que debe atender. Cada vez que llega una petición, un servidor debe moverse hacia esa posición.

El problema *online* consiste en minimizar la distancia recorrida por todos los servidores luego de n peticiones, sin saber la secuencia de puntos a atender.

Recuerde que un algoritmo *online* ALG es r-competitivo si existe una constante a tal que para cualquier instancia I y el algoritmo óptimo OPT,

$$\text{cost}_{ALG}(I) \leq r \cdot \text{cost}_{OPT}(I) + a$$

Donde r es el *radio competitivo*.

1. Sea ALG un algoritmo online para el problema de los k servidores bajo un espacio métrico arbitrario con al menos $k+1$ puntos. Pruebe que el radio competitivo de ALG es al menos k.

2. Para el siguiente análisis competitivo, necesitamos usar una conocida herramienta, la función potencial. *Una función Φ es una función de potencial que demuestra un radio competitivo r de un algoritmo ALG si satisface las siguientes condiciones:*

   - Φ es nonegativa
   - Cada respuesta a una petición a OPT incrementa Φ no más de r veces el costo cargado a OPT por esa respuesta.
   - Cada respuesta a una petición a ALG disminuye el potencial por al menos el costo cargado a ALG por esa respuesta.

Por lo que, un algoritmo es r-competitivo, si existe una función de potencial Φ para $r > 0$ que cumple las propiedades anteriores.

Considere el problema de k servidores en una linea (un espacio de dimensión 1) y el siguiente algoritmo:

- Si todos los servidores están a un lado de la petición, entonces envía el servidor más cercano a ella.
- Si una petición se encuentra entre dos servidores, envía los dos servidores a velocidad constante, y se detienen cuando uno de ellos llega a su objetivo.

Finalmente, de una función de potencial Φ que demuestre un radio competitivo de k para este algoritmo.

1.2 P3 C1 2006/1

Usted va a vivir por una temporada a una ciudad extranjera, donde el boleto de metro cuesta \$1. Existe un carnet de descuento que cuesta \$D, y con el cual el boleto de metro se adquiere a \$B< 1. Usted realizará una cantidad n de viajes en metro durante su estadía, pero no tiene idea de qué día volverá y por lo tanto no tiene ninguna estimación confiable de n.

Usted podría comprar el carnet el primer día y pagaría $D+nB$, o no comprarlo nunca y pagaría n, o en general comprarlo luego de i viajes pagando en total $D + i + B(n - i)$.

Diseñe una estrategia online y analice su competitividad con respecto al algoritmo óptimo (que sabe cuántos viajes realizará). Demuestre que no se puede obtener mejor competitividad que la de su método.

1.3 P2 Examen 2010/1

Considere un algoritmo online aleatorizado para el problema de arrendar los esquíes. Si el costo de arrendar por día es a y de comprar es c, este algoritmo decide comprar el día $\lfloor c/(2a) \rfloor + 1$ con probabilidad p, y sino compra el día $\lfloor c/a \rfloor + 1$ como el determinístico.

1. Halle el costo esperado que paga el algoritmo aleatorizado hasta el día t, que es el día (desconocido) en el que se debe volver del resort. Separe en casos $t \leq c/(2a)$, $c/(2a) < t \leq c/a$, y $t > c/a$.
2. Encuentre el valor de p que optimiza la competitividad esperada del algoritmo y muestre que ésta resulta ser menos que 2.
3. Muestre que ningún algoritmo determinístico puede ser mejor que 2-competitivo.

1.4 Cow Search (*-ish*) problem

De repente, usted despierta en medio de una carretera en medio de la lluvia y no sabe dónde está. Además tiene sed y dolor de cabeza, y no recuerda cómo llegó ahí. Lo único que sabe es que su casa está en algún lugar de la carretera, pero no sabe en qué dirección ir. Además, debido a la fuerte lluvia y al dolor de cabeza no puede ver bien y sólo se daría cuenta de que llegó al lugar correcto una vez estando ahí.

Para facilitarle las cosas, suponga que la carretera es una recta infinita y que usted está en $x = 0$ y que su casa está en x^*, con $|x^*| > 1$.

1. Muestre que la estrategia de ir de un lado a otro duplicando la distancia a recorrer es 9-competitiva.
2. De un algoritmo aleatorizado basado en su estrategia anterior, y calcule el radio competitivo.

1.5 Tom y Jerry

Jerry lanza platos desde lo alto de una repisa y Tom intenta recogerlos antes de que se estrellen contra el suelo del pasillo. El pasillo tiene $2k+1$ baldosas y Tom recorre una baldosa por segundo. Cada plato tarda k segundos antes de llegar al suelo desde que Jerry lo lanza. Tom sabe en qué baldosa caerá el plato en el instante en que Jerry lo lanza, de manera que puede recorrer hasta k baldosas para salvarlo. Jerry lanza un plato por segundo, y lo puede lanzar hacia la baldosa que quiera. Nos interesa diseñar una estrategia *competitiva* para Tom, en términos de la cantidad de platos salvados.

1. Muestre que la estrategia de ir a buscar el siguiente plato lanzado en caso de que sea posible alcanzarlo (e ignorar todo el resto hasta recogerlo), y sino seguir en el mismo lugar esperando el próximo plato, no es competitiva.

2. Muestre que la estrategia de ir a buscar siempre el plato más cercano al suelo tampoco es competitiva

3. Diseñe una estrategia $2k$-competitiva para Tom. Demuestre su competitividad y encuentre un caso donde se salve sólo 1 de cada $2k$ platos que podría salvar el algoritmo óptimo.

2 Probabilisticos

2.1 P1 C2 2009/1

Un teorema de Lagrange dice lo siguiente: Sea $\pi(x)$ la cantidad de números primos $\leq x$. Entonces $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln(x)} = 1$.

1. Utilice este teorema y el algoritmo probabilístico visto en clase para *detectar* primos, para diseñar un algoritmo que *genere* un número primo mayor o igual que un n dado.

2. Analice el tiempo promedio de su algoritmo y la probabilidad de error asociada.

3. Discuta si su algoritmo es de tipo Monte Carlo, Las Vegas, u otra cosa.

2.2 Ecuaciones binarias

Suponga que tiene un conjunto de n variables binarias $x_1, \ldots, x_n$ y un conjunto de k ecuaciones, donde la ecuación r-ésima es de la forma

$$(x_i + x_j) \bmod 2 = b_r$$

Para dos variables distintas x_i, x_j y algún valor b_r. Considere el problema de encontrar una asignación de valores que maximice el número de ecuaciones que se cumplen.

1. Sea c^* el máximo número de ecuaciones que se cumplen dada una asignación de valores a las variables. Diseñe un algoritmo que produzca una asignación que satisfaga al menos a la mitad de las ecuaciones.

2. Ahora considere el mismo problema pero para una cantidad arbitraria de variables por ecuación.

2.3 Hashing

1. ¿Cuál es el tiempo esperado (*promedio*) que toma una búsqueda exitosa en una tabla de hash T usando encadenamiento, suponiendo que la función de hash distribuye los elementos de manera uniforme? ¿Cómo analizaría el tiempo en *el peor caso*? Discuta estrategias para mejorar el tiempo esperado usando alguna(s) de las herramientas vistas en el curso hasta el momento.

2. Suponga que tiene una función de hash uniforme, y n llaves distintas a guardar en un arreglo de largo m, ¿cuál es la cantidad esperada de colisiones?

3. Dado el tamaño de página $B = 3$, muestre cómo se construye una tabla de hash según el esquema Extendible Hashing con la secuencia

$$00001, 10000, 10001, 01001, 01111, 00000, 10010, 11111, 10111$$

2.4 Multiplicación de Matrices

Diseñe un algoritmo aleatorizado de tipo Monte Carlo que verifique la identidad $AB = C$, donde A, B y C son matrices de $n \times n$ en tiempo $O(n^2)$. Si el rango de valores de las celdas de las matrices es $[1..N]$, muestre que la probabilidad de error del algoritmo es a lo más $1/N$.

3 Aproximados

3.1 P2 C2 2009/1

Se tienen n tareas de cómputo que requieren tiempos de CPU $t_1, t_2, \ldots, t_n$, y $m \leq n$ procesadores donde distribuirlas. Se busca asignar las tareas a procesadores de modo que el procesamiento de todas ellas en paralelo demore el menor tiempo posible.

Formalmente, sea T_i la suma de los tiempos t_j de las tareas asignadas al procesador i. Se desea que la asignación minimice $T = \max_{1 \leq i \leq m} T_i$. Este problema es NP-completo.

1. Sea T^* el tiempo de la asignación óptima. Demuestre que $T^* \geq \frac{1}{m} \sum_{1 \leq i \leq m} T_i = \frac{1}{m} \sum_{1 \leq j \leq n} t_j$

2. Pruebe que $T^* \geq \max_{1 \leq j \leq n} t_j$.

3. Se propone la siguiente heurística: partir con los $T_1 = \ldots = T_m = 0$, y considerar las tareas t_j de a una. En cada caso, asignar t_j al mínimo actual de los T_i. Pruebe que esta heurística es en realidad una 2-aproximación.

3.2 P3 C3 2011/2

El problema de *bin packing* consiste en empaquetar ítems de tamaños $X = \langle x_1, x_2 \ldots, x_n \rangle$ en la menor cantidad posible de cajas, donde éstas tienen un tamaño fijo b. Este problema es NP-completo.

1. Dé una 2-aproximación para este problema (y muestre que lo es).

2. Muestre que no es posible lograr un factor de aproximación menor a 3/2 en tiempo polinomial si $P \neq NP$. Para ello, recuerde que el problema de partir X en dos conjuntos de igual suma es NP-completo.

3.3 Max Clique

Sea $G = (V, E)$ grafo no dirigido. Para todo $k \geq 1$, se define $G^{(k)}$ como el grafo no dirigido $(V^(k), E^(k))$ tal que $V^(k)$ es el conjunto de todos los vectores de k nodos de V y $E^(k)$ es definido de manera que $(v_1, \ldots, v_k)$ es conectado por una arista con $(w_1, \ldots, w_k)$ si y sólo si para todo $i \in [1..k]$, $(v_i, w_i) \in E$, o $v_i = w_i$.

1. Muestre que si t es el tamaño del clique máximo de G, entonces el tamaño T del clique máximo de $G^{(k)}$ es t^k.

2. Muestre que si existe un algoritmo de aproximación de radio constante para el problema del clique máximo, entonces existe un esquema de aproximación completamente polinomial para este problema.

3.4 Job Scheduling

Dadas n tareas de tiempos $p_1, \ldots, p_n$, respectivamente, y m máquinas, el objetivo de este problema es distribuir las tareas en las máquinas de forma de minimizar el tiempo utilizado hasta que todas las tareas sean procesadas.

Asuma que m es fijo, y con esto diseñe un esquema de aproximación completamente polinomial tal que dado $\varepsilon > 0$, retorne un algoritmo $(1 + \varepsilon)$-aproximado.

3.5 MAX-3SAT

Argumente que el algoritmo de asignar al azar uniformemente 0 o 1 a cada literal de la fórmula en 3CNF (lo que vimos que en promedio satisfacía a 7/8 de las cláusulas) es una $\frac{8}{7}$-aproximación.

4 Miscelaneos

4.1 P1 Examen 2008/1, Examen 2009/1

Para las siguientes afirmaciones, responda V ó F, argumentando en a lo sumo 3 líneas. Respuestas sin la correcta argumentación no valen.

1. Los algoritmos probabilísticos pueden resolver algunos problemas (con cierta probabilidad de error) en un tiempo imposible para un algoritmo determinístico.

2. Todo problema de optimización admite un esquema de aproximación completamente polinomial.

3. Si existe un algoritmo online k-competitivo para un cierto problema, entonces también existe una k-aproximación para el problema.

4. Una k-aproximación es un algoritmo cuyo tiempo de ejecución es a lo sumo k veces el del algoritmo óptimo.

5. Un algoritmo tipo Las Vegas se equivoca con una cierta probabilidad.

4.2 P2 Examen 2009/1

Está usted en el comité de contratación de una prestigiosa universidad, para un concurso al que se han presentado varios candidatos. A éstos se les ha dado un año para que investiguen sobre un cierto problema P, para el cual sólo se conoce una cota inferior de $\Omega(n \log^2 n)$ y un algoritmo de tipo Las Vegas de costo $O(n^3)$. Hoy los candidatos presentan sus resultados. Evalúe, a través del título de sus presentaciones, cuáles deben ser invitados amablemente a irse por no mejorar los resultados existentes, y cuáles deben contratarse inmediatamente por obtener los mejores resultados del grupo (es decir, dentro de los que aportan, los que no son superados en todos los aspectos por algún otro). Argumente.

1. "Una 2-aproximación de tiempo $O(n^3)$ para P".
2. "El problema P es $\Omega(n\sqrt{(}n))$".
3. "Un algoritmo $O(n^3)$ en promedio para P, para entradas uniformemente distribuidas".
4. "Un esquema de aproximación de tiempo $O(\frac{1}{\varepsilon^2} n^{1+2/\varepsilon})$ para P".
5. "El problema P es $\Omega(n^{1/\sqrt{\log n}})$".
6. "Un algoritmo PRAM de tiempo $T(n,p) = O(n \log(n/p))$ para P". (algoritmos paralelos).

5 Algunas soluciones

5.1 P1.1

5.1.1 Parte I

Suponga, sin pérdida de generalidad, que ALG mueve sólo un servidor tras cada petición y que cada uno de los k servidores comienzan en lugares distintos. Escoja un subespacio del espacio métrico con las k posiciones más una arbitraria. Sea d_{ij} la distancia entre los puntos i y j.

Mostraremos cómo un adversario puede escoger una secuencia de peticiones σ_n tal que el costo de ALG sea al menos k veces el costo de un algoritmo óptimo. La elección es simple, el adversario escogerá el siguiente punto como uno no cubierto por ALG. El costo total de ALG es entonces:

$$\text{cost}_{ALG}(\sigma_n) = \sum_{i=1}^{n} d(\sigma(i+1), \sigma(i))$$

Donde $\sigma(i)$ es la petición i-ésima. Como sabemos que en la petición $(i+1)$-ésima el adversario escogerá la ubicación vaciada por ALG en la petición i-ésima, por lo que el movimiento será de $\sigma(i+1)$ a $\sigma(i)$.

Para mostrar que ALG es k-competitivo, mostraremos que existe un algoritmo que puede servir la secuencia por $1/k$ del costo.

Para esto, considere k algoritmos que se comportan de la misma forma y que sólo difieren en dónde comienzan los servidores de cada uno. Existen $\binom{k+1}{k} = k+1$ formas de escoger las k posiciones iniciales. De ellas, k tendrán un servidor en $\sigma(1)$. Esas k posibilidades serán las posiciones iniciales de los k algoritmos.

Si el adversario ya tiene un servidor en la ubicación pedida, entonces no hará nada. Si el request $\sigma(n)$ no está ocupado por alguno de los servidores del adversario, lo moverá desde la ubicación $\sigma(n-1)$. Note que tras cada petición, cada uno de los k algoritmos tendrá distintas *configuraciones* de servidores (el conjunto de las ubicaciones de éstos).

Al comienzo está claro que las configuraciones son distintas. Asuma que es cierto hasta la petición $\sigma(n-1)$ y comparemos dos algoritmos. Antes de la petición $\sigma(n)$ tendrán configuraciones distintas.

- Si ambos tienen un servidor en esa posición, no harán nada y tendrán configuraciones distintas.
- Si ninguno tiene un servidor ahí, ambos moverán el servidor en $\sigma(n-1)$ hacia $\sigma(n)$ y seguirán teniendo configuraciones distintas.
- Si uno de los dos tiene un servidor ahí, pero el otro no, entonces éste moverá su servidor de $\sigma(n-1)$ a $\sigma(n)$. El otro dejará su servidor donde está, por lo que tendrán configuraciones distintas.

Con esto, la idea es ver el costo total de los k algoritmos si los ejecutáramos simultáneamente. No hay costo en la primera petición pues todos ellos tendrán ya un servidor en $\sigma(1)$. Luego de $n-1$ peticiones, cada uno de los k algoritmos tendrá un servidor en $\sigma(n-1)$. Como cada algoritmo tiene sus servidores en distintas configuraciones, y como ningún algoritmo tiene dos servidores en la misma posición al mismo tiempo, tenemso que todos menos un algoritmo tendrá un servidor en $\sigma(n)$. El costo en ese caso será de $d(\sigma(n-1), \sigma(n))$. Por lo tanto, el costo total de correr los k algoritmos sobre la secuencia σ_n es

$$\sum_{i=2}^{n} d(\sigma(i-1), \sigma(i))$$

Como el costo de los k algoritmos no es mayor que el de ALG, entonces al menos uno de ellos no tendrá costo mayor que $\text{cost}_{ALG}(\sigma_n)/k$, por lo que el radio competitivo es k.

5.1.2 Parte II

Sean $a_1, \ldots, a_k$ los servidores del adversario y $s_1, \ldots, s_k$ los de nuestro algoritmo. Usaremos tanto a_j como s_j para denotar la posición de cada servidor en la recta real (por lo que habrá que renombrarlos cada vez que se sobrepasen).

Definamos Φ como

$$\Phi = \Psi + \Theta$$

donde

$$\Psi = k \sum_i |a_i - s_i|$$

y

$$\Theta = \sum_{i<j} |s_i - s_j|$$

Asumiremos, sin pérdida de generalidad, que el adversario no mueve más de un servidor por petición. Consideremos los distintos casos que pueden ocurrir:

- El adversario puede mover un servidor a_i una distancia d incurriendo un costo d. Si se acerca al servidor s_i, entonces Ψ disminuirá en kd. Si se aleja de s_i, Ψ aumentará kd. En cualquier caso, $\Delta\Psi \leq kd$ y por lo tanto $\Delta\Phi \leq kd$.

- Cuando nuestro algoritmo mueve un servidor, tenemos dos casos:

  - Cuando mueve uno solo, puede ser s_1 o s_k. Asumamos que mueve s_1 a la izquierda una distancia d. Como el adversario ya respondió a esa solicitud, tendrá a a_1 a la izquierda de s_1, por lo que Φ disminuye exactamente d, el costo nuestro algoritmo, ya que Ψ disminuirá en kd y Θ aumentará en $(k-1)d$.
  - En el caso en el que mueve dos servidores hacia una petición, supongamos que mueve a s_i y a s_{i+1} y que a_j es un servidor del adversario que ya está en la posición pedida. Si $j \leq i$, entonces s_i que se mueve hacia a_j también se mueve hacia a_i, en este caso, Ψ no aumenta debido a que cada incremento causado por el movimiento de s_{i+1} es cancelado por el movimiento de s_i. Si $j \geq i+1$, entonces los roles de s_i y s_{i+1} se intercambian y tenemos que Ψ no aumenta.

Falta mostrar que Θ disminuirá lo suficiente para pagar el movimiento de los dos servidores. Dado que cada servidor s_j, $j \neq i, i{+}1$ está o a la izquierda de ambos o a la derecha de ambos, tenemos que uno de los servidores se mueve hacia s_j y el otro se aleja. Luego la suma $|s_j - s_i| + |s_j - s_{i+1}|$ se mantiene constante durante la fase. El único término en Θ que no hemos contado es $|s_i - s_{i+1}|$. Este término disminuirá exactamente el costo del movimiento durante esta fase.

5.2 P1.4

5.2.1 Parte I

Considere la estrategia A_m de de buscar la casa en fases. Las fases comienzan con $i = 0$, comenzando en el origen, y caminar una distancia $(-m)^i$ y luego volver al origen, deteniéndose antes si se llega al objetivo. En cada fase en la cual no se encuentre el objetivo, se habrá recorrido una distancia $2m^i$.

Dividamos las posibles ubicaciones en regiones $1, 2, \ldots, k$ donde la región k satisface $m^{k-1} < |x^*| \leq m^k$. Suponga que $|x^*|$ cae en la región k. Entonces no habrá encontrado el punto sino hasta la fase k. Si tenemos suerte, entonces $-(-m)^{k-1} < x^* \leq (-m)^k$ y encuentra la casa en la fase k, por lo que la distancia recorrida es

$$A_m(x^*) = \sum_{i=0}^{k-1} 2m^i + |x^*| = 2\frac{m^k - 1}{m - 1} + |x^*|$$

Si no tenemos suerte, entonces no encontraremos la casa sino hasta la fase $k + 1$, por lo que

$$A_m(x^*) = \sum_{i=0}^{k} 2m^i + |x^*| = 2\frac{m^{k+1} - 1}{m - 1} + |x^*|$$

Como el adversario tratará de maximizar el costo, pondrá la casa en el lado desafortunado, por lo que el radio será

$$r = \frac{2}{|x^*|}\Big(\frac{m^{k+1} - 1}{m - 1}\Big) + 1$$

Como asumimos que $|x^*|$ cae en la región k, el peor radio para la región k ocurre si el adversario pone la casa a distancia $m^k + \epsilon$ del origen, por lo tanto,

$$r < 2\Big(\frac{m^2 - \frac{1}{m^{k-1}}}{m - 1}\Big) + 1$$

Cuando k tiende a infinito, el radio entonces está dado por

$$r = 2\Big(\frac{m^2}{m - 1}\Big) + 1$$

Derivando e igualando a 0,

$$\frac{dr}{dm} = 0 = 2\frac{x(x - 2)}{(x - 1)^2}$$

Tenemos que cuando $m = 2$, $r = 9$, por lo que la estrategia de duplicar la distancia recorrida es 9-competitiva.

5.2.2 Parte II

Como el adversario puede escoger a priori la dirección en la cual estará la casa para aumentar nuestro costo, podemos evitarlo escogiendo al azar al comienzo hacia qué dirección ir.

Al comienzo, lanzamos una moneda equilibrada: si sale cara, en cada fase i caminamos desde 0 hasta $(-m)^i$ y volvemos. Si sale sello, entonces en cada fase i caminamos desde 0 hasta $-(-m)^i$ y volvemos. Usando el análisis anterior y tomando la esperanza, tenemos:

$$\begin{aligned} r &= \frac{1}{2}\Big(\frac{2}{|x^*|}\Big(\frac{m^k-1}{m-1}\Big)+1\Big)+\frac{1}{2}\Big(\frac{2}{|x^*|}\Big(\frac{m^{k+1}-1}{m-1}\Big)+1\Big) \\ &= 1+\frac{1}{|x^*|}\Big(\frac{m^k+1+m^{k-1}-2}{m-1}\Big) \end{aligned}$$

Escogiendo $|x^*| = m^k + \epsilon$ y haciendo tender k a infinito, tenemos

$$r < 1 + \frac{m^2+m}{m-1}$$

Si escogemos $m = 2$, el radio competitivo es entonces $r = 7$. Si derivamos e igualamos a 0, tenemos que cuando $m = 1+\sqrt{2} \approx 2.414$, el radio competitivo es $r = 4+2\sqrt{2} \approx 6.8284$.

5.3 P1.5

5.3.1 Parte I

Tirar un plato en una esquina, esperar a que llegue, y luego tirar $n-1$ en la otra, donde no irá a buscarlos (o infinitos platos en la otra esquina); el óptimo ignoraría el primero y salvaría todo el resto.

5.3.2 Parte II

Tirar uno en -1, y los demás en $1, 2, 3, \ldots, k, k-1, k-2, \ldots$, etc. Los pierde todos. El óptimo ignoraría el primero y salvaría el resto.

5.3.3 Parte III

Pararse al medio e ir a buscar el próximo plato que salga (siempre llegará), y volver al medio. En el peor caso tarda $2k$ y salva 1 plato. El óptimo puede salvar $2k$ platos, por lo que la estrategia es $2k$-competitiva. Se llega a esta tasa si, por ejemplo, tiran un plato en $+k$ y $2k-1$ platos en $-k$.

5.4 P2.2

5.4.1 Parte I

Usando la misma estrategia para MAX-3SAT, asignando un 1 o 0 con la misma probabiliadad a cada variable. Sea $E[X_r]$ el valor esperado de la variable que toma el valor 1 si la ecuación r-ésima es satisfecha. De las cuatro posibles valuaciones de la ecuación i, hay dos que causan que se evalúe a 0 ($x_i = x_j = 0$, y $x_i = x_j = 1$), y dos que hacen que evalúe a 1. Por lo tanto, $E[X_r] = 2/4 = 1/2$.

Por linealidad de la esperanza, $E[X] = \sum_r E[X_r] = k/2$. Como el máximo de ecuaciones satisfacibles c^* es a lo más k, se satisfacen $c^*/2$ ecuaciones en promedio.

5.4.2 Parte II

Queremos hacer, igual que para la parte anterior, que $E[X_r] = 1/2$, incluso si hay una cantidad arbiraria de variables en la ecuación r-ésima; en otras palabras, la probabilidad de que la ecuación tome el valor correcto módulo 2 sea exactamente $1/2$.

Hay una forma fácil de hacerlo: se asignan valores arbitrarios a todas las variables excepto a la última. Como una asignación lleva a la solución esperada y la otra no, al asignar un valor al azar, la probabilidad de satisfacer a la ecuación r es $1/2$.

Otra forma es contar las valuaciones para las cuales una ecuación toma el valor 1, y el valor 0, y verificar que son iguales (por lo que la probabilidad es la misma, $1/2$).

Suponga que la ecuación r tiene t términos, por lo que hay 2^t asignaciones posibles. Queremos decir que 2^{t-1} producen una suma 1, y 2^{t-1} producen una suma 0, lo cuál mostrará que $E[X_r] = 1/2$. Probaremos esto por inducción en t.

- $t = 1$; hay dos asignaciones, para $t = 2$ ya lo vimos en la parte anterior.
- Supongamos que la proposición se cumple para $t - 1$, entonces hay exactamente 2^{t-1} formas de tener una suma 0 con t variables, como sigue:
  - 2^{t-2} formas de tener una suma par (0) en las primeras $t - 1$ variables (por inducción), seguido de una asignación 0 al término t-ésimo,
  - 2^{t-2} formas de tener una suma impar (1) en las primeras $t - 1$ variables (por inducción), seguido de una asignación 1 al término t-ésimo.
- Para obtener una suma impar (1), el análisis es análogo al anterior, por lo que la mitad de todas las posibles asignaciónes de valores a x_i producen una suma par, y la mitad una suma impar.

Una vez que tenemos que $E[X_r] = 1/2$, concluimos igual que en la primera parte.

5.5 P2.3

1. Usando encadenamiento, sabemos que para buscar tenemos que mirar todas las posiciones de la lista enlazada correspondiente hasta encontrar el elemento. Calculemos el largo esperado $l(x)$ de la lista en la celda $T[h(x)]$.

   En esta lista estarán todos los valores que colisionaron. Definamos la indicatriz $C_{xy} = I\{h(x) = h(y)\}$, luego $l(x) = \sum_{y \in T} C_{xy}$.

   Notemos además que $E[C_{xy}] = \Pr\{h(x) = h(y)\} = 1/m$, donde m es el largo de la tabla, ya que la función distribuye las llaves uniformemente. Por último, $E[l(x)] = \sum_y E[C_{xy}] = \sum_y 1/m = n/m$

   Definimos $\alpha = n/m$ el *factor de carga*. Finalmente, buscar un elemento toma en promedio $O(1 + \alpha)$. Consideremos que si n es proporcional a m, entonces el tiempo es constante. Podemos mejorar este tiempo usando otras estructuras en vez de listas enlazadas (árbol binario, más hash tables, ... árbol de tablas).

   Para analizar el peor caso, debemos considerar el largo máximo de la lista, $L = \max_x l(x)$, por lo que el tiempo será $O(1 + L)$. A menos que el factor de carga sea *muy menor* a 1, este largo

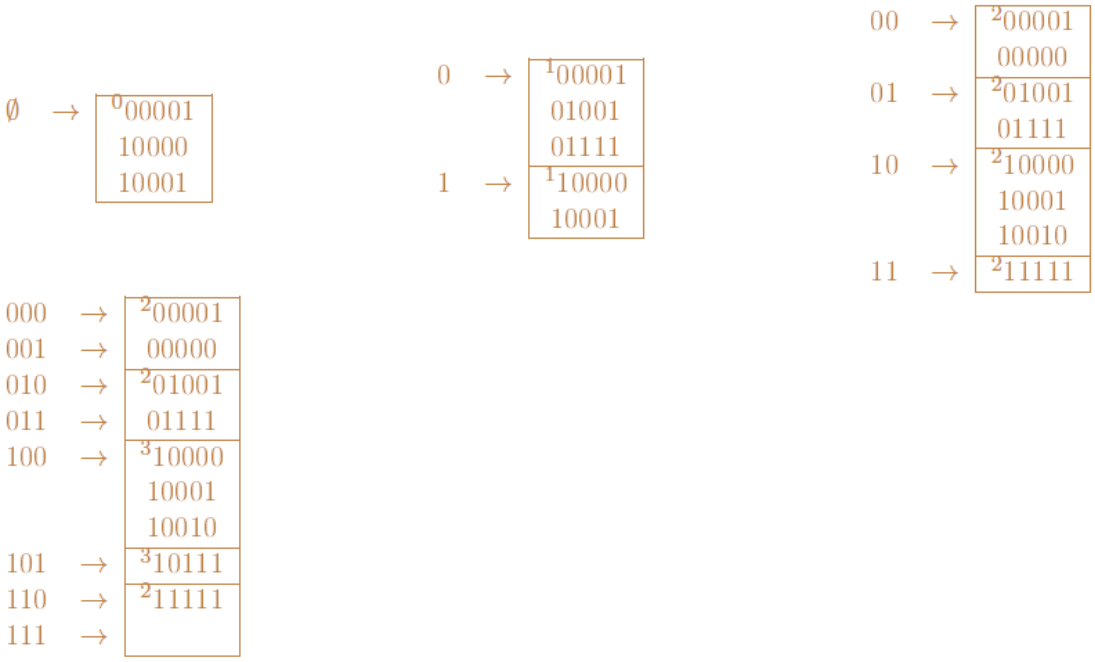
puede ser tan largo como la cantidad de elementos en la lista. Como dato, si $\alpha = 1$, el largo máximo (esperado) es $\Theta(\log n / \log \log n)$ (lo cual no es mucho mejor que un árbol binario, por ejemplo).

2. Para cada par de llaves k, l donde $k \neq l$, definamos la indicatriz $X_{kl} = I\{h(k) = h(l)\}$. Como asumimos que la función de hash distribuye las llaves uniformemente, $\Pr\{X_{kl} = 1\} = \Pr\{h(k) = h(l)\} = 1/m$, y por tanto, $E[X_{kl}] = 1/m$.

   Definamos la variable aleatoria Y como el número total de colisiones, de forma que $Y = \sum_{k \neq l} X_{kl}$. Entonces, el número de colisiones es $E[Y] = \sum_{k \neq l} E[X_{kl}] = \binom{n}{2} \frac{1}{m} = \frac{n(n-1)}{2m}$.

   Una forma más intuitiva de verlo es considerar, como ejemplo, el ejercicio de distribuir de manera uniforme n bolas en m urnas. ¿De cuántas formas pueden haber 2 bolas (..distintas) en una misma urna? ¿Y considerando la probabilidad de que hayan 2 bolas en una urna?

3. Extendible Hashing



5.6 P3.3

5.6.1 Parte I

Por inducción en k

- Caso base, trivial, dado que $G^{(1)} = G$.
- Caso inductivo: Asumiendo que la propiedad se tiene para k, se mostrará que para $G^{(k+1)}$ se cumple con $T' = t^{k+1}$. Dado $G^{(k)} = (V^{(k)}, E^{(k)})$, asumimos que en la definición de $V^{(k)}$ se consideran los vectores con vértices repetidos y permutados, esto es, si $V = \{1, 2\}$, entonces $V^{(2)} = \{(1,1), (1,2), (2,1), (2,2)\}$.

Luego, si V tiene n vértices, el tamaño de $V^{(k)}$ es por lo tanto n^k, y el tamaño de $V^{(k+1)}$ es n veces más grande.

En $V^{(k)}$, al aumentar los vectores con las nuevas combinaciones al pasar a $V^{(k+1)}$, como sabemos que cada vértice es "aumentado" n veces, en particular será aumentado t veces con cada vértice del clique, y que, por lo tanto, el tamaño del clique máximo de $G^{(k+1)}$ es t veces más grande que el anterior, esto es, $T' = t^k t = t^{k+1}$. Para ilustrar este resultado, por ejemplo, si el clique máximo de G lo componen $(1, 2, 3)$ y G tiene 4 vértices, entonces entre los vértices de $G^{(2)}$ podemos considerar a $v_1 = (1, 1)$, $v_2 = (1, 2)$, $v_3 = (1, 3)$ (entre otros). Al pasar a $G^{(3)}$, por cada vértice en $G^{(2)}$ añadimos todos los vértices de G, es decir, para v_1 tenemos $v_1^1 = (1, 1, 1)$, $v_1^2 = (1, 1, 2)$, $v_1^3 = (1, 1, 3)$, y $v_1^4 = (1, 1, 4)$, $v_2^1 = (1, 2, 1)$, $v_2^2 = (1, 2, 2)$ y $v_2^3 = (1, 2, 3)$, y $v_2^4 = (1, 2, 4)$, y así sucesivamente. Es claro que cada vértice de G que no pertenece al clique tampoco pertenecerá al clique de $G^{(3)}$, por lo que la cantidad de vértices que pertenecen al clique que se añaden en $G^{(3)}$ es 3, y su tamaño es $T = t^k = 27$.

5.6.2 Parte II

Suppose that there is a constant $\alpha > 1$ and an algorithm A such that, given a graph G over n nodes and of maximal clique size k, A finds a clique of size at least k/α in time $p(n)$, polynomial in n.

Given $\varepsilon > 0$, can we use a polynomial (in n and in $1/\varepsilon$) number of calls to A in order to find a clique of size $k/(1 + \varepsilon)$?

- Consider a graph G, of maxclique size t.
- Choose $k = 1/\lg_\alpha(1 + \varepsilon)$ such that $\alpha^{1/k} = 1 + \varepsilon$.
- Build $G^{(k)}$ of maxclique size t^k in time polynomial in n^k.
- Run A on $G^{(k)}$ finding a clique of size t^k/α in time $p(n^k)$.
- Deduce a clique for G, of size $t/\alpha^{1/k} = t/(1 + \varepsilon)$.

5.7 $\star$ Set Cover

El problema del Set Cover consiste en lo siguiente: Dados un conjunto $X = \{x_1, \ldots, x_n\}$, $S = \{S_1, \ldots, S_m\}$, donde $S_i \subset X$, y $k \leq m$, asumiendo que $\bigcup_{S_i \in S} S_i = X$, determinar si existe un *set cover* de X de tamaño k. Un set cover es un subconjunto $S' \subseteq S$ tal que

$$\bigcup_{S_j \in S'} S_j = X$$

Para este problema, nos enfocaremos en determinar un set cover de tamaño mínimo (mínima cantidad de subconjuntos S_i).

El problema del set cover puede ser también descrito como el problema del *Hitting Set*. Para este problema, tenemos un *hipergrafo* $H = (V, E)$, donde $V = \{v_1, \ldots, v_n\}$ y $E = \{e_1, \ldots, e_m\}$ y $e_i \subseteq V$ para $i \in [1..m]$. Es decir, cada arista puede incidir en más de un vértice (¿recuerda el problema del coloreo de conjuntos? pues bien, busque sobre coloreo de hipergrafos). En este problema, el objetivo es escoger la menor cantidad de vértices que "cubren" todos los hiperarcos en H.

Para mostrar que este problema es equivalente al del set cover, podemos transformar una instancia de este problema al del set cover: $X = E$, y $S = \{S_1, \ldots, S_n\}$, donde $e \in S_i$ si y sólo si $e \in E$ toca el vértice v_i. Para cada vértice v_i en V, existe el correspondiente S_i en S. El inverso de esta transformación es similar.

El problema del Hitting Set puede ser visto como una generalización del problema del Vertex Cover en hipergrafos. Los dos problemas son idénticos salvo que el Vertex Cover impone la restricción que cada arista toca exactamente dos vértices. El Vertex Cover puede ser visto como especialización del Set Cover al ver que cada elemento de X aparece exactamente en dos conjuntos de S.

En lo siguiente se mostrará una ρ-aproximación para el Set Cover, donde $\rho = H(|S_{max}|)$, y S_{max} es el conjunto más grande de S. H es la función armónica tal que

$$H(t) = \sum_{i=1}^{t} \frac{1}{i}$$

Una estrategia greedy es tomar avaramente conjuntos de S. En cada paso tomamos $S_j \in S$ que contenga la mayor cantidad de elementos no cubiertos anteriormente y repetir hasta cubrirlos todos:

- $U = X$
- $S' = \emptyset$
- mientras $U \neq \emptyset$
  - tomar $S_j \in S$ tal que $|S_j \cap U|$ sea máximo
  - $S' = S' \cup S_j$
  - $U = U - S_j$
- retornar S'

Para mostrar que el radio de aproximación es $H(|S_{max}|)$, usaremos el método de contabilidad de costos: el costo de escoger un conjunto será \$1. Si escogemos un conjunto con k elementos no cubiertos, a cada uno de estos elementos le vamos a cobrar $\frac{1}{k}$. Como cada elemento es cubierto una sola vez, se le cobrará una vez. El costo C del algoritmo será lo que se le cobra a todos los elementos:

$$C = \sum_{x_i \in X} c(x_i) \leq \sum_{S_i \in OPT} C(S_i) \leq |OPT| H(|S_{max}|)$$

donde $c(x_i)$ es el costo asignado al elemento x_i. S_i es un conjunto en el set cover óptimo. $C(S_i)$ es el costo del conjunto S_i sumando los costos de todos sus elementos, esto es:

$$C(S_i) = \sum_{x_i \in S_i} c(x_i)$$

La desigualdad $\sum_{x_i \in X} c(x_i) \leq \sum_{S_i \in OPT} C(S_i)$ dice que el costo del set cover óptimo es al menos la suma de los costos de sus elementos. Podemos verlo más claro reescribiendo este costo como

$$\sum_{S_i \in OPT} C(S_i) = \sum_{x_i \in X} c(x_i) \cdot n(x_i)$$

Donde $n(x_i)$ es el número de veces que x_i aparece en el set cover óptimo. Como cada elemento aparece al menos una vez, se deduce la desigualdad.

Para mostrar la segunda desigualdad, $\sum_{S_i \in OPT} C(S_i) \leq |OPT| H(|S_{max}|)$, basta mostrar que $C(S_i) \leq H(|S_i|)$, y con esto tendríamos

$$\sum_{S_i \in OPT} C(S_i) \leq \sum_{S_i \in OPT} H(|S_i|) \leq \sum_{S_i \in OPT} H(|S_{max}|) = |OPT| \cdot H(|S_{max}|)$$

Para mostrar que $C(S_i) \leq H(|S_i|)$, considere un conjunto arbitrario $S_i \in S$. Como los conjuntos son escogidos por el algoritmo greedy, nos interesa ver qué pasa con los elementos de S_i. Sea U_0 el conjunto de elementos no cubiertos en S_i cuando comienza el algoritmo. Luego de j iteraciones del algoritmo, U_j representa los elementos no cubiertos de S_i. Si el algoritmo escoge un conjunto que cubre elementos de U_j, escribimos el nuevo conjunto de elementos no cubiertos como U_{j+1}. Note que U_{j+1} es un subconjunto propio de U_j. Luego, $U_0, U_1, \ldots, U_k, U_{k+1}$ es el "historial" de elementos no cubiertos de S_i, donde $U_{k+1} = \emptyset$.

Al avanzar de U_j a U_{j+1}, el número de elementos al que le cobramos su costo es $|U_j - U_{j+1}|$. ¿Cuánto se les cobra a todos esos elementos?

Suponga que estamos en el punto del algoritmo donde U_j representa el conjunto de elementos no cubiertos de S_i, y el algoritmo greedy está a punto de escoger un conjunto. Asuma que escoge el conjunto donde al menos un elemento no cubierto de S_i es cubierto (de otra manera, el algoritmo continuará hasta que esto suceda). Queremos mostrar que a los elementos que son cubiertos no se les cobra más que $1/|U_j|$. Si escoge S_i, entonces a cada elemento de U_j se le cobra $1/|U_j|$. Asumamos que no escoge a S_i, sino a T. La única razón por la cual escogería a T es que tiene al menos $|U_j|$ elementos no cubiertos. En este caso, a cada uno de los elementos no cubiertos de T se le cobra a lo más $1/|T| \leq 1/|U_j|$.

Por la suposición, dijimos que algunos elementos de T tienen una intersección no vacía con elementos en S_i. Específicamente, $T \cap U_j$ fue cubierto cuando se escogió a T. El costo total adicional a S_i al escoger T es a lo más

$$|T \cap U_j| \cdot \frac{1}{|U_j|} = |U_j - U_{j+1}| \cdot \frac{1}{|U_j|}$$

Por lo tanto, el costo total de S_i es

$$C(S_i) \leq \frac{|U_0 - U_1|}{|U_0|} + \frac{|U_1 - U_2|}{|U_1|} + \ldots + \frac{|U_k - U_{k+1}|}{|U_k|}$$

El último término es 1, ya que $U_{k+1} = \emptyset$.

Cada uno de los términos de la suma tienen la forma $(x-y)/x$ para $x > y$. Mostraremos que cada uno es menor que $H(x) - H(y)$.

$$\begin{aligned} H(x) - H(y) &= (1 + \frac{1}{2} + \ldots + \frac{1}{x}) - (1 + \frac{1}{2} + \ldots + \frac{1}{y}) \\ &= \frac{1}{y+1} + \frac{1}{y+2} + \ldots + \frac{1}{x} \\ &\geq (x-y) \cdot \frac{1}{x} \end{aligned}$$

La última desigualdad viene del hecho que la igualdad anterior tiene $x - y$ términos no mayores a $1/x$.

Con esto, el costo de S_i lo podemos reescribir como

$$\begin{aligned} C(S_i) &\leq (H(|U_0|) - H(|U_1|)) + (H(|U_1|) - H(|U_2|)) + \ldots + (H(|U_k|) - H(|U_{k+1}|)) \\ &= H(|U_0|) - H(|U_{k+1}|) \\ &= H(|U_0|) \end{aligned}$$

Y como $U_0 = S_i$, entonces $C(S_i) \leq H(|S_i|)$.

¿Qué tan ajustada es la desigualdad de $C(S_i)$? Suponga que S_i tiene 5 elementos. En un paso 3 elementos son cubiertos y un paso subsecuente 2 elementos son cubiertos. El costo del conjunto es a lo más $\frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{1}{2} + \frac{1}{2}$ lo cual es menor que $H(5)$. ¿Es posible que $C(S_i)$ sea igual a $H(|S_i|)$? Sí, pero sólo si a lo más un elemento es cubierto en S_i cada vez que el algoritmo escoja un conjunto a ser agregado al set cover.

Hay una demostración un poco distinta para el Set Cover aproximado en Introduction to Algorithms, CLRS, Sección 35.3 (y en general, todo el capítulo 35 para algoritmos aproximados).