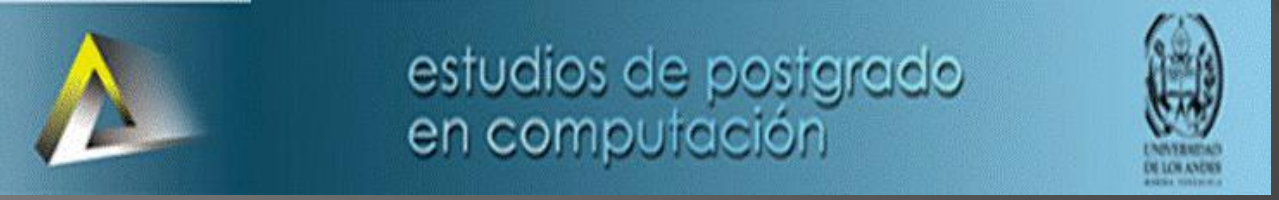


Análisis y Diseño de Algoritmos (AyDA)

Isabel Besembel Carrera

ALGORITMOS PROBABILISTAS
ALGORITMOS MULTIHILOS
PROGRAMACIÓN LINEAL

Algoritmos probabilistas

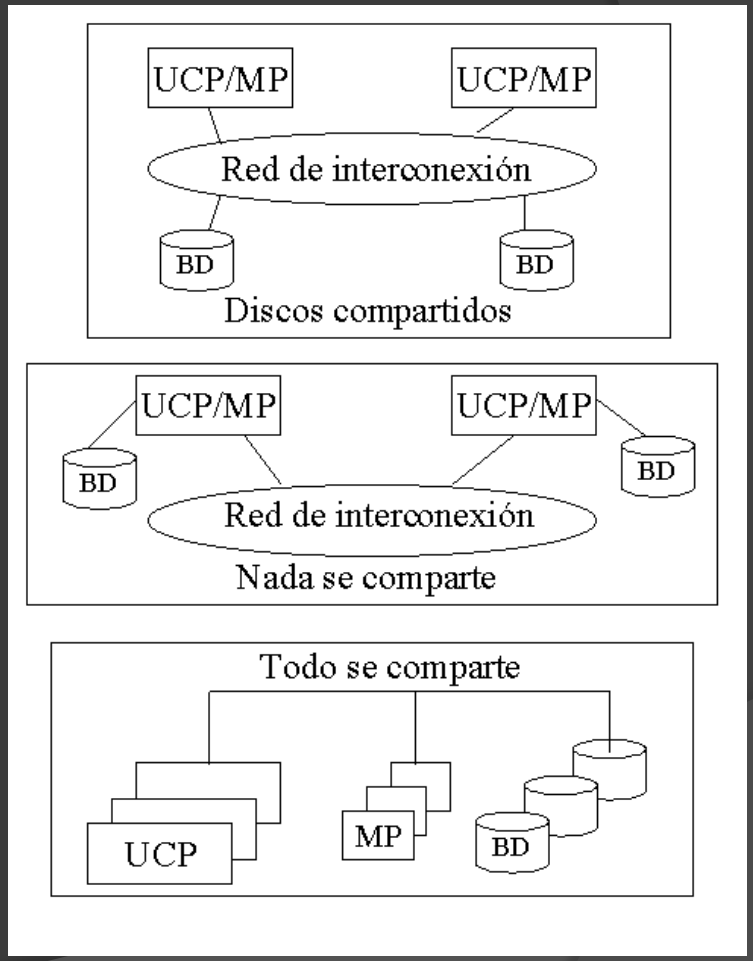
Algoritmos multihilos

Algoritmos seriales vs paralelos

- Algoritmos seriales: para máquinas uniprocesador
- Algoritmos paralelos y multihilos: para máquinas multiprocesadores o paralelas
- Computadoras paralelas: con varias CPU
  - PC o laptop con chip multiprocesador que contiene un multicore, cada “core” es un CPU que puede acceder a la memoria comun (las menos costosas)
  - Clusters construidos con PCs interconectados con red
  - Supercomputadoras: las más costosas pero con el mejor rendimiento

Modelos de arquitecturas

- Memoria compartida: cada procesador accede directamente cualquier localidad de memoria
  - PCs y laptops multicore
- Memoria distribuida: cada procesador tiene su memoria.



Hilos

- Hilado estático: procesadores virtuales
  - Hilo: proceso que accede a la memoria común compartida, manteniendo un contador de programa y ejecutando código independientemente de otros hilos
  - Creación y destrucción de hilos es lento
  - Normalmente el hilo persiste => estático
- Plataformas concurrentes:
  - Proveen una capa de software que coordina, planifica y maneja los recursos paralelos

Multihilado dinámico

- Permite a los programadores especificar aplicaciones paralelas sin preocuparse por los protocolos de comunicación, balance de carga y otros problemas del hilado estático
- Soporta:
  - Paralelismo anidado: permite que una subrutina se ejecute mientras el programa que la invocó continua (ejecución asíncrona)
  - Lazos paralelos: permite que las iteraciones del lazo se ejecuten concurrentemente

Multihilado dinámico

- Es una extensión del modelo serial
- Se incluyen en los algoritmos las palabras claves: paralelo, spawn y sincronizar
  - Si se eliminan dichas palabras del algoritmo queda la versión serial
- Prevee un medio de cuantificar el paralelismo basado en las nociones de “work” y “span”
- Usan la técnica de divide-y-vencerás y su análisis mediante la solución de recurrencias
- Existen varia plataformas que la soportan
  - Clik, Clik++, OpenMP, Task Parallel Library

Multihilado dinámico

- Ejemplo: Generación de los números de Fibonacci
  Fo=0
  F1=1
  Fi=Fi-1 + Fi-2 para i ≥ 2
  Recurrencia $T(n)=T(n-1)+T(n-2)+\Theta(1)= \Theta(Fn) = \Theta(\phi^n)$
  $\phi=(1+\sqrt{5})/2$ “radio dorado”

Junio,04 **fibonacci(Entero: n):Entero** **{ pre: n ≥ 0 }** **{ pos: número de Fibonacci ≥ 0 }**		
1 2 3	**v(0), v(1) = 0, 1** **[v(j) = v(j - 1) + v(j - 2)] j = 2, n** **regrese v(n)**	**-v: Arreglo(0..n)de Entero. Vector auxiliar que almacena los valores intermedios anteriores** **-j: Entero. Subíndice de v**
1 2	n=4, v=(0,1,1,2,3), v(4)=3 n=0, v=(0,1), v(0)=0	Caso exitoso Caso exitoso

Multihilado dinámico

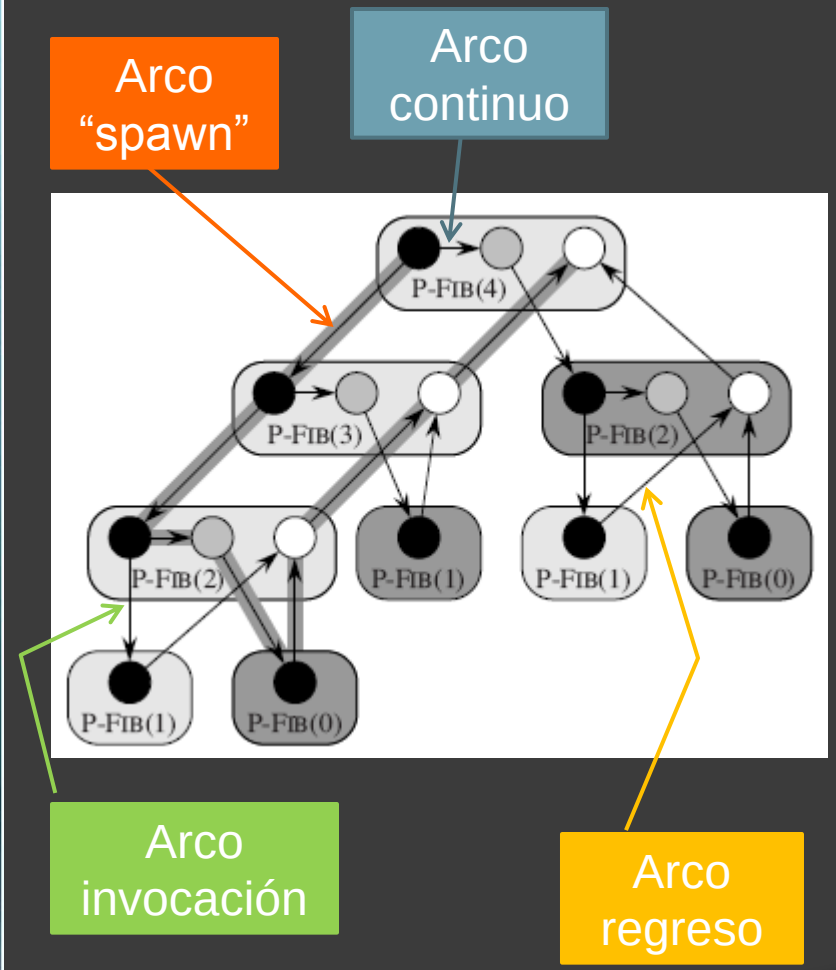
Enero,10		
	fiboRec(Entero: n):Entero	
{ pre: n ≥ 0 }		**{ pos: número de Fibonacci ≥ 0 }**
1	**Si(n <= 1) entonces** **regrese n** **Sino** **x, y = fiboRec(n-1), fiboRec(n-2)** **regrese x+y** **fsi**	**-x, y: Entero. Variables auxiliares**
1 2	n=4, x=2, y=1=> 3 n=0 =>0	Caso exitoso Caso exitoso

Multihilado dinámico

Enero,10		
	fiboRecPara(Entero: n):Entero	
{ pre: n ≥ 0 }		**{ pos: número de Fibonacci ≥ 0 }**
1	**Si(n <= 1) entonces** **regrese n** **Sino** **x, y = spawn fiboRecPara(n-1), fiboRecPara(n-2)** **sinc** **regrese x+y** **fsi**	**-x, y: Entero. Variables auxiliares**
1 2	n=4, x=2, y=1=> 3 n=0 =>0	Caso exitoso Caso exitoso

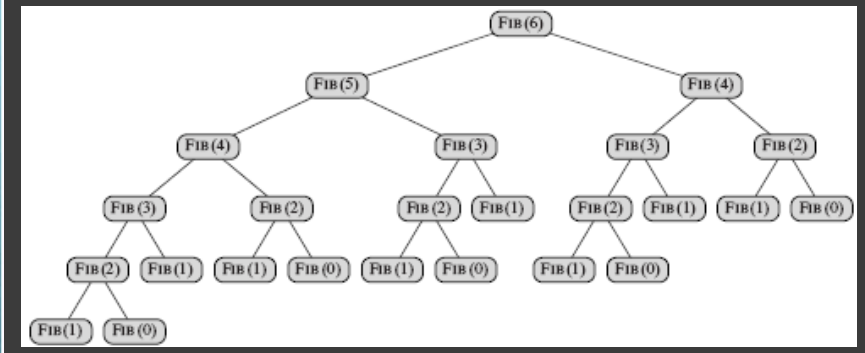
- "spawn" implica que el procedimiento que lo invoca **puede** continuar su ejecución en paralelo con el procedimiento llamado (asíncrono)
- "sinc" implica que el procedimiento debe esperar hasta que todos los procedimientos "spawn" terminen su ejecución (sincronización)
- Paralelismo recursivo

Modelo de ejecución multihilo



- Computación dag: grafo dirigido acíclico de computación multihilo G=(N, A)
  - N conjunto de instrucciones agrupadas "strand"
  - A dependencias entre las instrucciones, (u, v) $\in$ A, la instrucción u se debe ejecutar antes de la instrucción v

Modelo de ejecución multihilo



- Si G tiene un camino de u a v, entonces los 2 “strand” están en serie, de lo contrario están en paralelo

- Instrucciones sin control de paralelismo (“spawn”, “sinc” o “return”) se agrupan en un “strand”
- Si un “strand” tiene 2 sucesores, uno de ellos tiene “spawn”
- Si un “strand” tiene varios antecesores, debe haber un “sinc”antes para unirlos

Computador paralelo ideal

- Conjunto de procesadores
  - En una plataforma concurrente donde las instrucciones se intercalan produciendo los mismos resultados de una ejecución secuencial consistente con el dag
- Memoria compartida secuencial y consistente
  - Memoria que almacena los resultados exactamente como si el algoritmo se hubiera ejecutado en forma secuencial
- Todos los procesadores tienen la misma capacidad de cálculo e ignoran los costos de la planificación (generalmente mínimos)
- Número de procesadores = P

Mediciones de rendimiento

- Métricas:
  - “work”:
    - tiempo total para efectuar el cálculo en un solo procesador
    - Número de vértices en el dag
  - “span”:
    - tiempo más largo para efectuar los “strang” a lo largo de cualquier camino en el dag
    - Número de vértices en el camino más largo en el dag
  - Complejidad en tiempo en P procesadores T_P
  - Ley “work” $T_P \geq T_1/P$
  - Ley “span” $T_P \geq T_\infty$
  - Aceleración: que tan rápido se hace el cálculo en P procesadores en vez de 1 => T_1 / T_P ($T_1 / T_P \leq T_1 / T_\infty < P$)

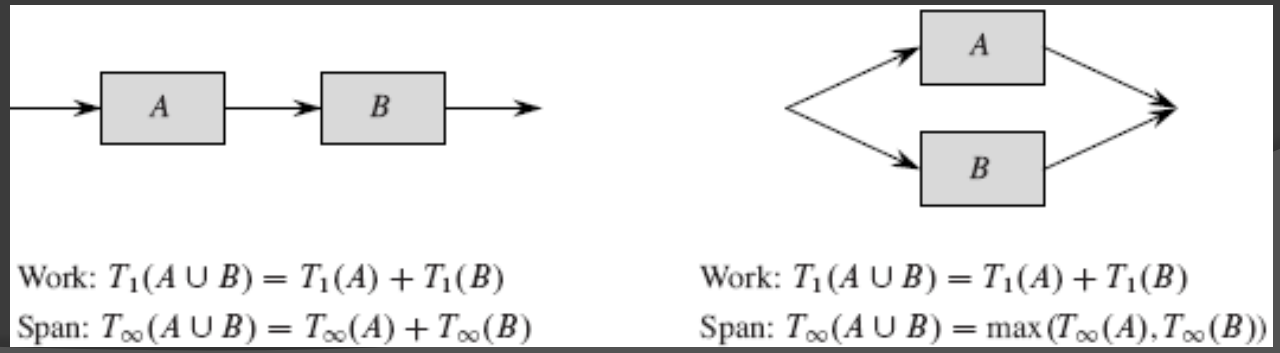
Mediciones de rendimiento

- Aceleración lineal perfecta: $T_1 / T_P = P$
- Paralelismo: T_1 / T_∞
  1. Cantidad promedio de trabajo que puede ser desarrollado en paralelo en cada paso a lo largo del camino crítico
  2. Aceleración máxima posible que puede ser alcanzada con cualquier número de procesadores
  3. Límite a la posibilidad de alcanzar la aceleración lineal perfecta
- Dejadez ("slackness"): $(T_1 / T_\infty) / P = T_1 / (P\ T_\infty)$
  - Dejadez < 1 => no se puede alcanzar la aceleración lineal perfecta
  - Dejadez > 1 => se acerca a la aceleración lineal perfecta

Planificador centralizado en-línea

- Paso completo: si hay P “strand” para ejecutarse en P
- Paso incompleto: no hay P “strand” para ejecutarse
- Teorema 27.1: Sobre un computador paralelo ideal con P procesadores, un planificador voráz ejecuta un cálculo multihilo con T_1 “work” y T_∞ “span” en tiempo $T_P \leq T_1/P + T_\infty$
- Caso Fibonacci

$$\begin{aligned} T_\infty(n) &= \max(T_\infty(n-1), T_\infty(n-2)) + \Theta(1) \\ &= T_\infty(n-1) + \Theta(1)\,, \end{aligned}$$



Lazos paralelos

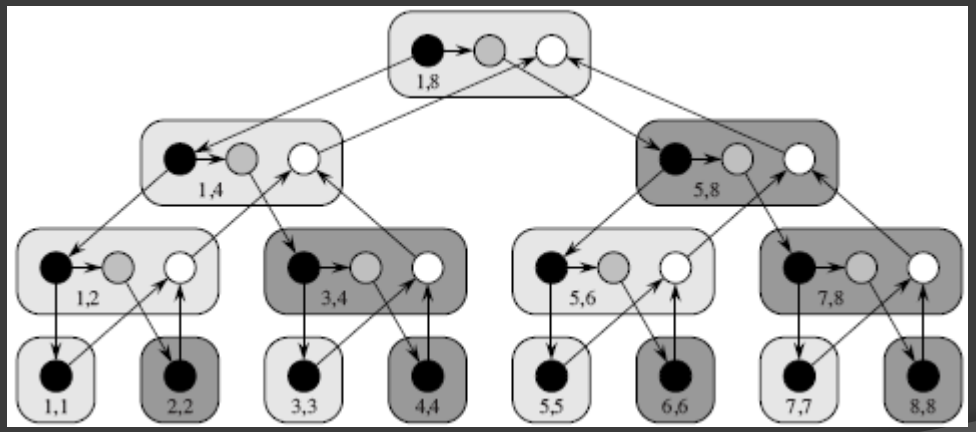
- Ejemplo: multiplicación de una matriz por un vector

$$y_i = \sum_{j=1}^{n} a_{ij} x_j$$

```
MAT-VEC(A, x)
1  n = A.rows
2  let y be a new vector of length n
3  parallel for i = 1 to n
4       y_i = 0
5  parallel for i = 1 to n
6       for j = 1 to n
7            y_i = y_i + a_ij x_j
8  return y
```

"sinc" y "spawn"

Θ(lg n)
Θ(n)




```
MAT-VEC-MAIN-LOOP(A, x, y, n, i, i')
1  if i == i'
2       for j = 1 to n
3            y_i = y_i + a_ij x_j
4  else mid = ⌊(i + i')/2⌋
5       spawn MAT-VEC-MAIN-LOOP(A, x, y, n, i, mid)
6       MAT-VEC-MAIN-LOOP(A, x, y, n, mid + 1, i')
7       sync
```

Multiplicación multihilos de matrices

```
P-SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  parallel for i = 1 to n
4      parallel for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

- "work" $T_1(n)=\Theta(n^3)$
- "span" $T_\infty(n) = \Theta(n)$
- Paralelismo $\Theta(n^3)/ \Theta(n) = \Theta(n^2)$
- Si se aplica Strassen
  - $M_1(n) = 8M_1(n/2) + \Theta(n^2) = \Theta(n^3)$
  - $M_\infty(n) = M_\infty(n/2)+ \Theta(\lg n)$
  - $M_1(n)/ M_\infty(n) = \Theta(n^3/\lg^2 n)$

```
P-MATRIX-MULTIPLY-RECURSIVE(C, A, B)
 1  n = A.rows
 2  if n == 1
 3       c_11 = a_11 b_11
 4  else let T be a new n × n matrix
 5       partition A, B, C, and T into n/2 × n/2 submatrices
              A_11, A_12, A_21, A_22; B_11, B_12, B_21, B_22; C_11, C_12, C_21, C_22;
              and T_11, T_12, T_21, T_22; respectively
 6       spawn P-MATRIX-MULTIPLY-RECURSIVE(C_11, A_11, B_11)
 7       spawn P-MATRIX-MULTIPLY-RECURSIVE(C_12, A_11, B_12)
 8       spawn P-MATRIX-MULTIPLY-RECURSIVE(C_21, A_21, B_11)
 9       spawn P-MATRIX-MULTIPLY-RECURSIVE(C_22, A_21, B_12)
10       spawn P-MATRIX-MULTIPLY-RECURSIVE(T_11, A_12, B_21)
11       spawn P-MATRIX-MULTIPLY-RECURSIVE(T_12, A_12, B_22)
12       spawn P-MATRIX-MULTIPLY-RECURSIVE(T_21, A_22, B_21)
13       P-MATRIX-MULTIPLY-RECURSIVE(T_22, A_22, B_22)
14       sync
15       parallel for i = 1 to n
16            parallel for j = 1 to n
17                 c_ij = c_ij + t_ij
```

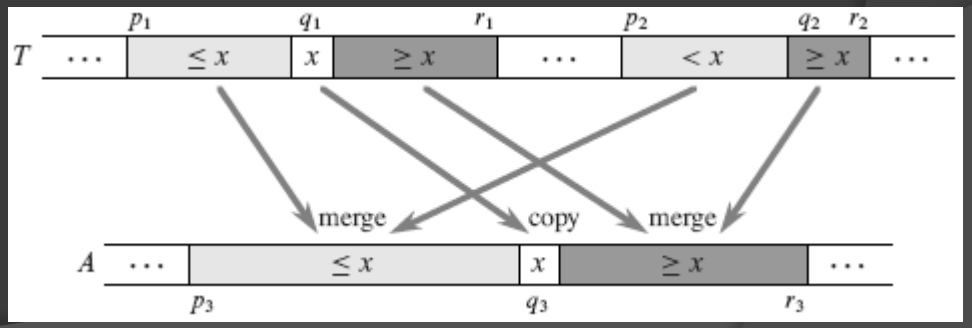
$\Theta(n^{\lg 7}/\lg^2 n)$

Ordenamiento y mezcla multihilos

- $\Theta(n \lg n)$ serial
- $MS'_1(n) = 2\, MS'_1(n/2) + \Theta(n) = \Theta(n \lg n)$
- $MS'_\infty(n) = MS'_\infty(n/2) + \Theta(n) = \Theta(n)$
- Paralelismo

$MS'_1(n) / MS'_\infty(n) = \Theta(\lg n)$

```
MERGE-SORT'(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      spawn MERGE-SORT'(A, p, q)
4      MERGE-SORT'(A, q + 1, r)
5      sync
6      MERGE(A, p, q, r)
```



Ordenamiento y mezcla multihilos


```
P-MERGE-SORT(A, p, r, B, s)
 1  n = r − p + 1
 2  if n == 1
 3       B[s] = A[p]
 4  else let T[1..n] be a new array
 5       q = ⌊(p + r)/2⌋
 6       q' = q − p + 1
 7       spawn P-MERGE-SORT(A, p, q, T, 1)
 8       P-MERGE-SORT(A, q + 1, r, T, q' + 1)
 9       sync
10       P-MERGE(T, 1, q', q' + 1, n, B, s)
```

- $PMS_1(n) = 2\ PMS_1(n/2) + PM_1(n) = 2\ PMS_1(n/2) + \Theta(n) = \Theta(n \lg n)$
- $PMS_\infty(n) = PMS_\infty(n/2) + PM_\infty(n) = PMS_\infty(n/2) + \Theta(\lg^2 n) = \Theta(\lg^3 n)$
- Paralelismo $PMS_1(n) / PMS_\infty(n) = \Theta(n \lg n) / \Theta(\lg^3 n) = \Theta(n/\lg^2 n)$

Algoritmos probabilistas

Programación lineal

Introducción

- Optimizar una función lineal sujeto a un conjunto de desigualdades
- $\{a_1,a_2,\ldots,a_n\} \in \Re$
- $\{x_1,x_2,\ldots,x_n\}$ variables

$F(x_1,x_2,\ldots,x_n) = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = \sum_{j=1}^{n} a_j x_j$

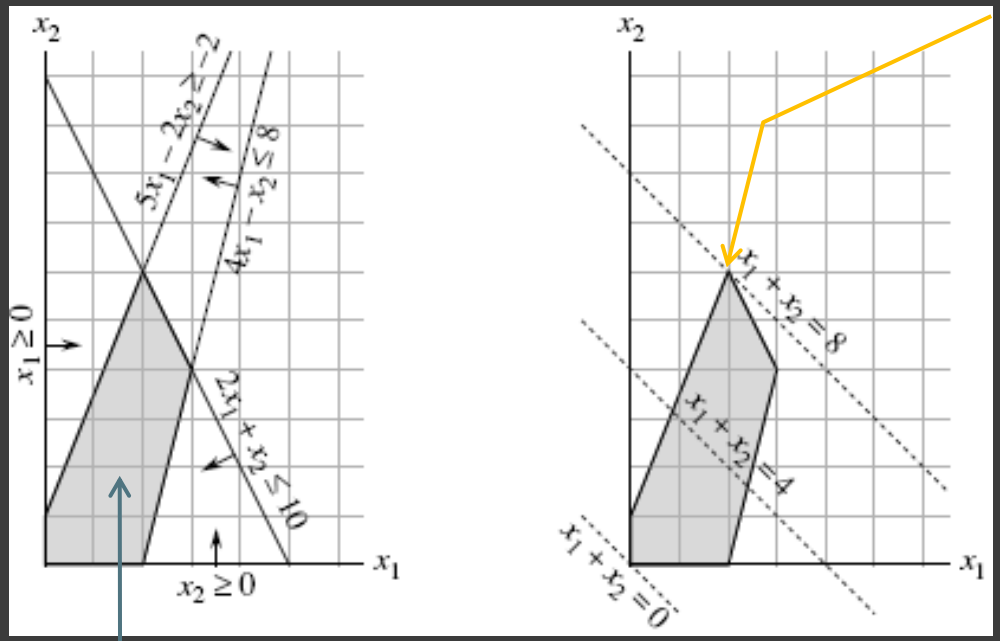
- Restricciones lineales
  - Igualdad lineal $F(x_1,x_2,\ldots,x_n) = b$
  - Desigualdades lineales $F(x_1,x_2,\ldots,x_n) \leq b$ o $F(x_1,x_2,\ldots,x_n) \geq b$

Problema de programación lineal

- Maximizar o minimizar una función lineal objetivo sujeto a un conjunto de restricciones lineales
  - Maximizar la función objetivo de n variables sujeta a un conjunto de m restricciones lineales
- Ejemplo: maximizar $x_1 + x_2$ con

  $4x_1 - x_2 \leq 8$ $2x_1 + x_2 \leq 10$ $5x_1 - 2x_2 \geq -2$ $x_1, x_2 \geq 0$
- Cualquier conjunto de valores x_1, x_2 que satisfaga todas las restricciones es una solución posible del programa lineal
- Valor objetivo: valor de la función objetivo en la región objetivo
  - $x_1 = 2$ y $x_2 = 6$ dan $x_1 + x_2 = 8$

Región posible de solución



Región posible

Máximo
$x_1 + x_2 = z$

- Solución óptima en un punto o línea del borde (los puntos extremos de la línea)
- Simplex: región convexa posible en nD formada por la intersección de los espacios
- Función objetivo es un hiperplano
- Solución óptima es un vértice en el simplex

Algoritmo simplex

- Entrada: programa lineal (función objetivo y las restricciones)
- Salida: solución óptima, óptimo local que a su vez es un óptimo global
- Identificar:
  - Cuando no hay solución
  - Cuando no hay solución óptima finita
  - Cuando el origen no es una solución posible
- Aplicaciones:
  - Planificación de tareas con restricciones (Elecciones, tripulación de vuelos áereos, perforación petrolera, etc.)

Forma estándar

- Forma estándar:
  - Dados: $c = c_1,c_2,\ldots,c_n \in \Re$, $b = b_1,b_2,\ldots,b_m \in \Re$, A $a_{ij} \in \Re$ con $i \in [1,\ldots, n]$ y $j \in [1,\ldots,m]$
  - Se desea encontrar $x = x_1,x_2,\ldots,x_n$ que maximicen $(\sum_{j=1}^{n} c_j x_j)\ c^T x$ sujeto a
    $(\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ con $i \in [1,\ldots, m])\ A x \leq b$
    $(x_j \geq 0$ con $j \in [1,\ldots,n])\ x \geq 0$ restricciones no negativas
- Programa lineal (A, b, c)
- Solución imposible: algún x no puede satisfacer alguna restricción
- Programa imposible de resolver: programa lineal que no tiene solución posible
- Programa ilimitado: aquel que no tiene valor objetivo óptimo finito

Algoritmo de inicio del simplex

INITIALIZE-SIMPLEX(A, b, c)

1 let l be the index of the minimum b_i
2 **if** $b_l \geq 0$ ▷ Is the initial basic solution feasible?
3 **then return** $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
4 form L_{aux} by adding $-x_0$ to the left-hand side of each equation and setting the objective function to $-x_0$
5 let (N, B, A, b, c, v) be the resulting slack form for L_{aux}
6 ▷ L_{aux} has $n+1$ nonbasic variables and m basic variables.
7 $(N, B, A, b, c, v) \leftarrow$ PIVOT$(N, B, A, b, c, v, l, 0)$
8 ▷ The basic solution is now feasible for L_{aux}.
9 iterate the **while** loop of lines 2–11 of SIMPLEX until an optimal solution to L_{aux} is found
10 **if** the basic solution sets $\bar{x}_0 = 0$
11 **then return** the final slack form with x_0 removed and the original objective function restored
12 **else return** "infeasible"

- N={1,2,…,n}
- B={n+1,n+2,…, n+m}
- $\dot{x}_i = b_i \ \forall \ i \in B$
- $\dot{x}_j = 0 \ \forall \ j \in N$
- Solución básica:

  $\dot{x}_i \geq 0 \ \forall \ i \in N \cup B$

Ejemplo: maximizar $2x_1 - x_2$ con

$2x_1 - x_2 \leq 2$ $\quad$ $x_1 - 5x_2 \leq -4$ $\quad$ $x_1, x_2 \geq 0$

Solución inicial imposible, por tanto hay que transformarlo a L_{aux} para obtener la solución posible

Algoritmo de inicio del simplex

- L_{aux} del ejemplo: maximizar $-x_0$ con

  $2x_1 - x_2 - x_0 \leq 2$ $\quad x_1 - 5x_2 - x_0 \leq -4$ $\quad x_1, x_2, x_0 \geq 0$

- Lema 29.11: sea L un programa lineal en forma estándar, y L_{aux} el programa lineal con n+1 variables, entonces L es posible si y solo si el valor objetivo óptimo de L_{aux} es 0
- Para tener la solución se invoca varias veces el procedimiento Pivot

$$
\begin{aligned}
z &= -x_0 \\
x_2 &= \frac{4}{5} - \frac{x_0}{5} + \frac{x_1}{5} + \frac{x_4}{5} \\
x_3 &= \frac{14}{5} + \frac{4x_0}{5} - \frac{9x_1}{5} + \frac{x_4}{5}
\end{aligned}
$$

Solución básica: $x_0 = 0$

Para resolver con el simplex, eliminando x_0

$$\frac{4}{5} + \frac{9x_1}{5} - \frac{x_4}{5},$$

and the slack form

$$
\begin{aligned}
z &= -\frac{4}{5} + \frac{9x_1}{5} - \frac{x_4}{5} \\
x_2 &= \frac{4}{5} + \frac{x_1}{5} + \frac{x_4}{5} \\
x_3 &= \frac{14}{5} - \frac{9x_1}{5} + \frac{x_4}{5}
\end{aligned}
$$

Algoritmo para pivotear

```
PIVOT(N, B, A, b, c, v, l, e)
 1  ▷ Compute the coefficients of the equation for new basic variable x_e.
 2  b̂_e ← b_l / a_le
 3  for each j ∈ N − {e}
 4       do â_ej ← a_lj / a_le
 5  â_el ← 1 / a_le
 6  ▷ Compute the coefficients of the remaining constraints.
 7  for each i ∈ B − {l}
 8       do b̂_i ← b_i − a_ie b̂_e
 9          for each j ∈ N − {e}
10               do â_ij ← a_ij − a_ie â_ej
11          â_il ← −a_ie â_el
12  ▷ Compute the objective function.
13  v̂ ← v + c_e b̂_e
14  for each j ∈ N − {e}
15       do ĉ_j ← c_j − c_e â_ej
16  ĉ_l ← −c_e â_el
17  ▷ Compute new sets of basic and nonbasic variables.
18  N̂ = N − {e} ∪ {l}
19  B̂ = B − {l} ∪ {e}
20  return (N̂, B̂, Â, b̂, ĉ, v̂)
```

Transforma la forma estándar dada en otra forma estándar equivalente, realizando transformaciones sobre la función objetivo y las restricciones

Algoritmo simplex

```
SIMPLEX(A, b, c)
 1  (N, B, A, b, c, v) ← INITIALIZE-SIMPLEX(A, b, c)
 2  while some index j ∈ N has c_j > 0
 3       do choose an index e ∈ N for which c_e > 0
 4          for each index i ∈ B
 5               do if a_ie > 0
 6                    then Δ_i ← b_i / a_ie
 7                    else  Δ_i ← ∞
 8          choose an index l ∈ B that minimizes Δ_i
 9          if Δ_l = ∞
10            then return "unbounded"
11            else  (N, B, A, b, c, v) ← PIVOT(N, B, A, b, c, v, l, e)
12  for i ← 1 to n
13       do if i ∈ B
14            then x̄_i ← b_i
15            else  x̄_i ← 0
16  return (x̄_1, x̄_2, ..., x̄_n)
```

Teorema fundamental de la programación lineal

- Teorema 29.13: cualquier programa lineal L , dado en su forma estándar:
  1. Tiene una solución óptima con un valor objetivo finito
  2. Es imposible de resolver
  3. Es ilimitado

Si L es imposible de resolver, SIMPLEX regresa “infeasible”

Si L es ilimitado, SIMPLEX regresa “unbounded”

De lo contrario SIMPLEX regresa la solución óptima con un valor objetivo finito