¡Bienvenidos al Labo de Algo 3!

1er cuatrimestre 2020

¿Qué vamos a ver hoy?

- Presentación de la materia.
- ¿Qué se espera del labo?
- Técnicas algorítmicas - Backtracking.
- Presentación del TP1.
- Consultas.

El equipo del laboratorio

1. Gabriel Budiño (AY2)
2. Mariano Rean (AY1)
3. Agustín Garassino (AY1)
4. Gonzalo Lera Romero (JTP)

Modalidad a distancia

- Cada viernes se sube una clase en alguno de tres modos.
  - Grabada.
  - Slides.
  - Escrita.
- La elección del modo depende del contenido de la clase y el tiempo disponible para su preparación.
- Se utiliza el foro del campus para hacer preguntas a otros alumnos y docentes.
- Cuando se acerque la fecha de entrega de cada TP habrá al menos una instancia de consulta virtual por medio de alguna plataforma (zoom, hangouts, etc).

Los TPs

Para aprobar el laboratorio hay que aprobar **2 trabajos prácticos**.
En general, (a menos que se especifique lo contrario):

- El primer TP es de a dos, el resto se realiza de a grupos de a 4.
- Los TPs tienen 2 notas: Reentregar y Aprobado.
- Hay una fecha de entrega y una fecha de reentrega.
- Toda reentrega debe estar acompañada por un informe de modificaciones detallando brevemente las diferencias entre las dos entregas (archivo .txt).
- Reentregar un TP significa corregir **todo** el TP.
- Quienes tengan los TPs aprobados del cuatrimestre pasado pueden elegir no cursarlos este cuatrimestre, pero deben avisarlo.

Lenguajes de comunicación

Si bien toda la materia se cursa y se aprueba en español, es muy recomendable el manejo de inglés técnico básico.

- Cursos de inglés en la facu.
- Becas o descuentos por ser alumnos de la UBA (en el CUI, o FILO).
- Alguna otra opción más artesanal como Duolingo o armar un grupo de estudio con bibliografía en inglés.

Lenguajes de programación

- El lenguaje de programación es C++.
  - Esto solo aplica a los códigos específicos de los algoritmos necesarios para el TP.
- Toda la experimentación, la generación de gráficos, etc. es posible (y recomendable!) hacerlo en otros lenguajes.
- No hay que reinventar la rueda, pero sí hay que hacer sus propios algoritmos.
- Se puede usar todo lo que está en la STL y tomar http://www.cplusplus.com/reference/stl/ como referencia para el uso y complejidad.
- El procesador de texto a utilizar también es a elección del grupo, aunque (sinceramente y por su bien) se sugiere el uso de LaTeX.

Comunicación científica

Una clásica comunicación científica sobre una resolución experimental de un problema en Ciencias de la Computación puede tener el siguiente formato:

- Introducción: Se introduce el problema, se muestra un review de literatura y otros aspectos introductorios como posibles aplicaciones.
- Desarrollo: Posiblemente dividido en subsecciones donde se muestran los modelos considerados, los algoritmos propuestos y los resultados teóricos obtenidos, entre otras cosas.
- Experimentación: Se definen instancias de fundamentado interés y se experimenta con las propuestas mencionadas en el desarrollo.
- Discusión y conclusiones: Se discuten los resultados observados en la sección anterior y se da un cierre general.

¿Qué se espera del TP?

Para cada ejercicio de los TPs se esperan los siguientes puntos:

1. Describir detalladamente el problema a resolver dando ejemplos del mismo y sus soluciones.
2. Explicar de forma clara, sencilla, estructurada y **concisa**, las ideas desarrolladas para la resolución del problema. Para esto se pide utilizar pseudocódigo y lenguaje coloquial combinando adecuadamente ambas herramientas (sin usar código fuente!). Se debe justificar por qué el procedimiento desarrollado resuelve efectivamente el problema.

¿Qué se espera del TP?

3. Deducir una **cota de complejidad temporal** del algoritmo propuesto, en función de los parámetros que se consideren correctos y **justificar** por qué el algoritmo desarrollado para la resolución del problema cumple con la cota dada.

4. Dar un código fuente claro que implemente la solución propuesta. El mismo no sólo debe ser correcto sino que además debe seguir las ***buenas prácticas de la programación*** (comentarios pertinentes, nombres de variables apropiados, estilo de indentación coherente, modularización adecuada, etc.)

¿Qué se espera del TP?

5. Realizar una experimentación computacional para medir la performance del programa implementado. Para ello se debe preparar un conjunto de casos de test que permitan observar los tiempos de ejecución en función de los parámetros de entrada.
Deberán desarrollarse tanto experimentos con instancias aleatorias (detallando cómo fueron generadas) como experimentos con instancias particulares (de peor/mejor caso en tiempo de ejecución, por ejemplo). Presentar **adecuadamente** en forma gráfica una comparación entre los tiempos medidos y la complejidad teórica calculada y extraer conclusiones de la experimentación.

Caso de estudio: Subset Sum

Aprendamos un poco de backtracking mientras vemos los puntos que hay que hacer en el TP.

El problema de la suma de subconjuntos consiste en, dado un conjunto $S = \{a_1, \ldots, a_n\}$ de números naturales y otro número natural w, determinar si existe un subconjunto S' de S tal que la suma de los elementos de S' sea igual a w. Escribir un algoritmo para resolver este problema que consista en revisar todos los subconjuntos de S para determinar si alguno suma w. Este tipo de algoritmo se llama algoritmo de "fuerza bruta".

1. Ejemplos y aplicaciones

Cuando comenzamos un informe lo primero que debemos hacer es explicar el problema, dar ejemplos y sus soluciones y enumerar posibles aplicaciones.

Ejemplos

1. S = { 1, 2, 3, 4, 5 }, W = 5, *existen varias soluciones* (S' = { 2, 3 } o { 1, 4 } o { 5 }), pero la de mínimo cardinal es S' = { 5 }.
2. S = { 1, 2, 3, 4, 5 }, W = 2, *existe una única solución* S' = { 2 } y por lo tanto es mínima.
3. S = { 2, 4 }, W = 3, *no existe solución* porque S contiene sólo números pares y w es impar.

Una posible aplicación nace naturalmente en *scheduling* de procesos, donde ante la presencia de dos procesadores, se desea saber si un conjunto de tareas puede ser dividido en dos partes tal que cada conjunto tarde lo mismo. Ese caso equivale a buscar si algún subconjunto suma la mitad del tiempo total.

2. Explicar ideas desarrolladas

Ahora vamos a resolver este problema utilizando fuerza bruta y backtracking. Luego vamos a comunicar los algoritmos con un pseudocódigo y justificar su correctitud y complejidad.

Para eso primero vamos a repasar las técnicas y ver cómo aplicarlas a este problema.

Comparando técnicas

Fuerza bruta

- Definir conjunto de posibles soluciones.
- Explorarlo completo y quedarse con alguna solución **factible** (decisión), o la **mejor** (optimización).

Backtracking

- Ir construyendo soluciones parciales explorando todas las opciones.
- Cuando una solución parcial no puede progresar con seguridad, se elimina esa opción y se vuelve sobre los pasos anteriores (backtrack).
- La exploración de todas las decisiones da lugar a un árbol de backtracking que se puede podar para ayudar a la búsqueda.

Fuerza bruta

Decimos que una solución es ***factible*** si cumple con todas las restricciones del enunciado. En este caso, si la suma de sus elementos es igual a W.

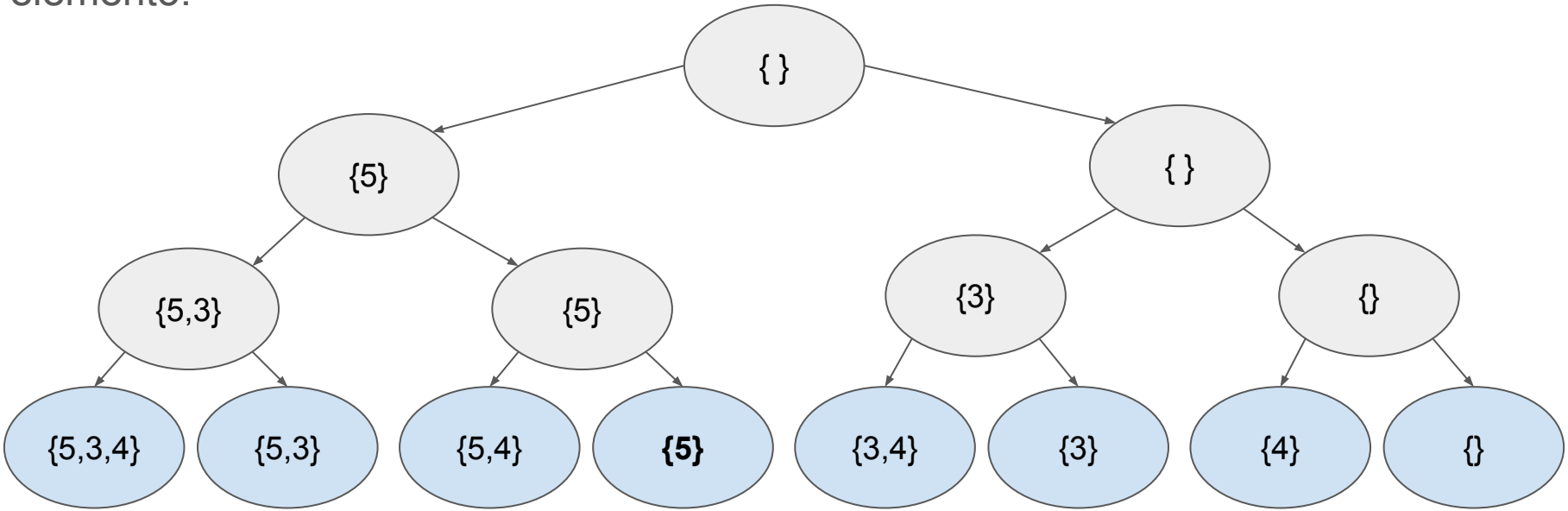
Por ejemplo, si S = { 5, 3, 4 } y W = 5, la solución S' = {3, 4} no es **factible**, mientras que S' = { 5 } sí es **factible**.

Definimos entonces el conjunto de soluciones $\mathcal{S}$ como el conjunto de partes de S que nuestro algoritmo va a explorar en busca de aquella **factible** de menor cardinal.

¿Cómo hacemos un código que lo explore?

Fuerza bruta

Proponemos un algoritmo recursivo para explorar S. La idea es ir construyendo parcialmente subconjuntos, decidiendo en cada paso si agregar o no el i-ésimo elemento.



Fuerza bruta

Después de haber explicado el algoritmo de fuerza bruta y mostrado un ejemplo de su ejecución, mostramos su pseudocódigo.

Algorithm 1 Algoritmo de fuerza bruta para Subset Sum.

```
1: function SubsetSum(S, W, i, w, k)
2:     //i: próximo elemento a procesar, w: suma parcial del nodo, k: # elementos seleccionados.
3:     if i = |S| + 1 then
4:         if w = W then
5:             return k
6:         else
7:             return ∞
8:     return min { SubsetSum(S, W, i + 1, w + S_i, k + 1), SubsetSum(S, W, i + 1, w, k) }
```

Fuerza bruta

Acá vemos un ejemplo de código en C++. Es importante que este código no figure en el informe, porque es más difícil de leer que un pseudocódigo y no agrega información extra.

```
int subsetSum(vector<int>& S, int W, int i, int w, int k) {
    if (i == S.size()) return w == W ? k : S.size()+1;
    return min(subsetSum(S, W, i+1, w+S[i], k+1), subsetSum(S, W, i+1, w, k));
}
```

Fuerza bruta

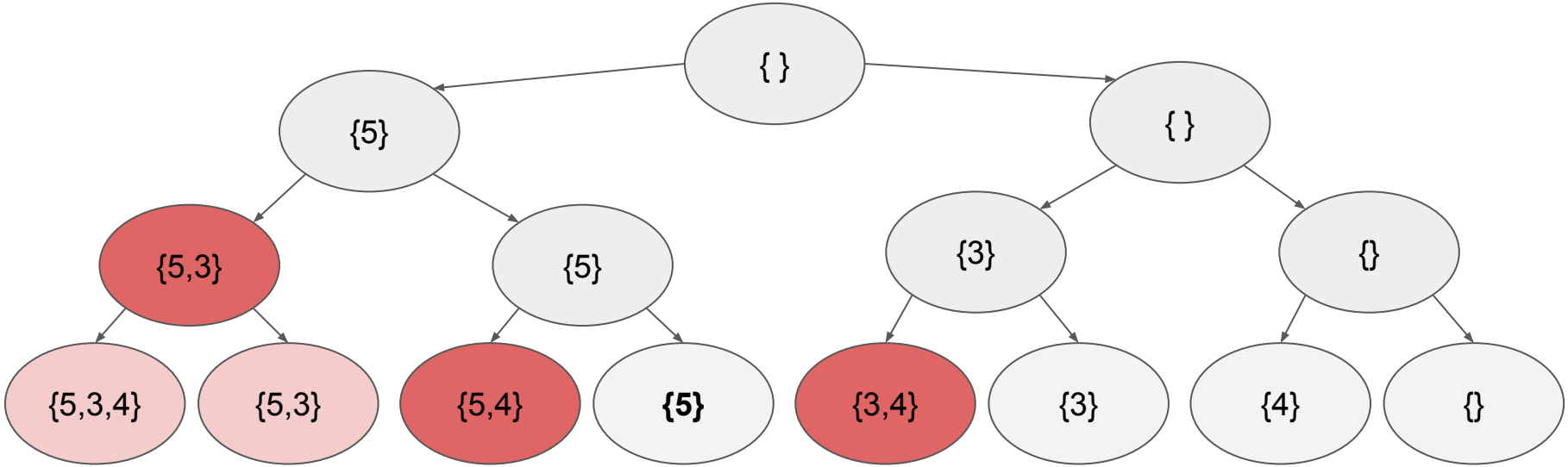
A continuación justificamos la correctitud y complejidad del algoritmo.

El algoritmo **es correcto porque** explora todo el conjunto de soluciones S porque en cada paso de la recursión explora ambas opciones (elegir o no el elemento i-ésimo), y consecuentemente acumula su valor. Cada hoja del árbol es un subconjunto y en ella se decide si es **factible** o no.

La complejidad del algoritmo es $O(2^n)$. Esto vale porque el árbol de recursión es binario y tiene una hoja por cada subconjunto, que son $O(2^n)$. Además, procesar cada nodo es constante porque solo se utilizan operaciones con ese orden de complejidad.

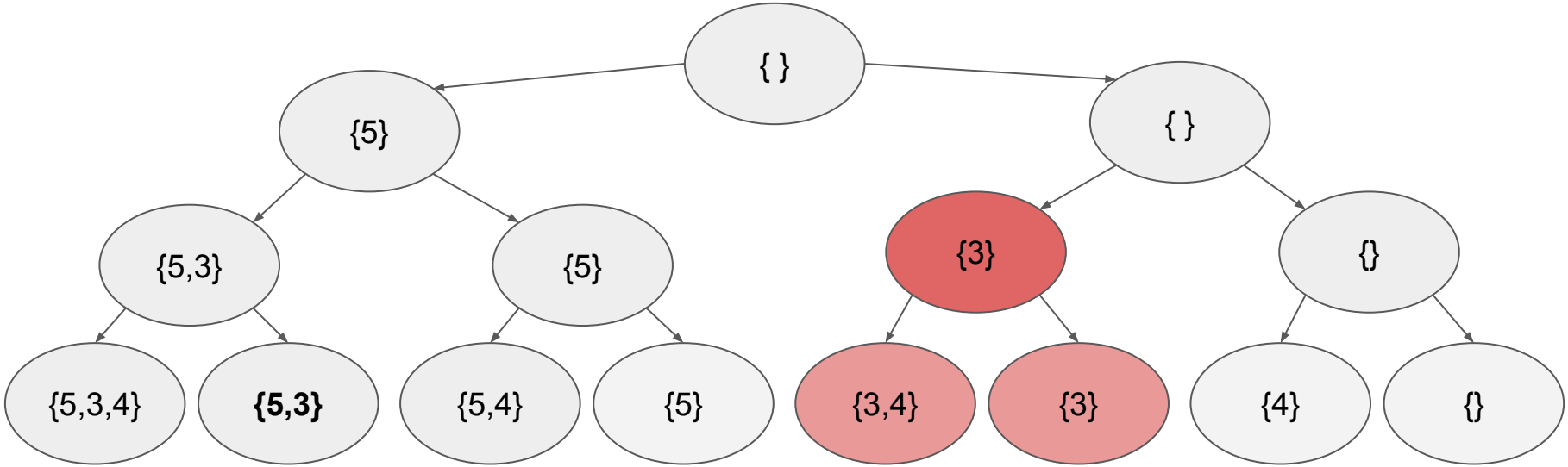
Backtracking (Poda por factibilidad)

Notemos que hay ciertos nodos del árbol, que ya tienen una suma superior a W = 5 (en rojo). Ni bien se entra en uno de estos nodos, ya sabemos que no vamos a encontrar una solución **factible** allí ni en ningún subárbol. Entonces, podemos evitar seguir recorriendo. Esto es una poda por **factibilidad**.



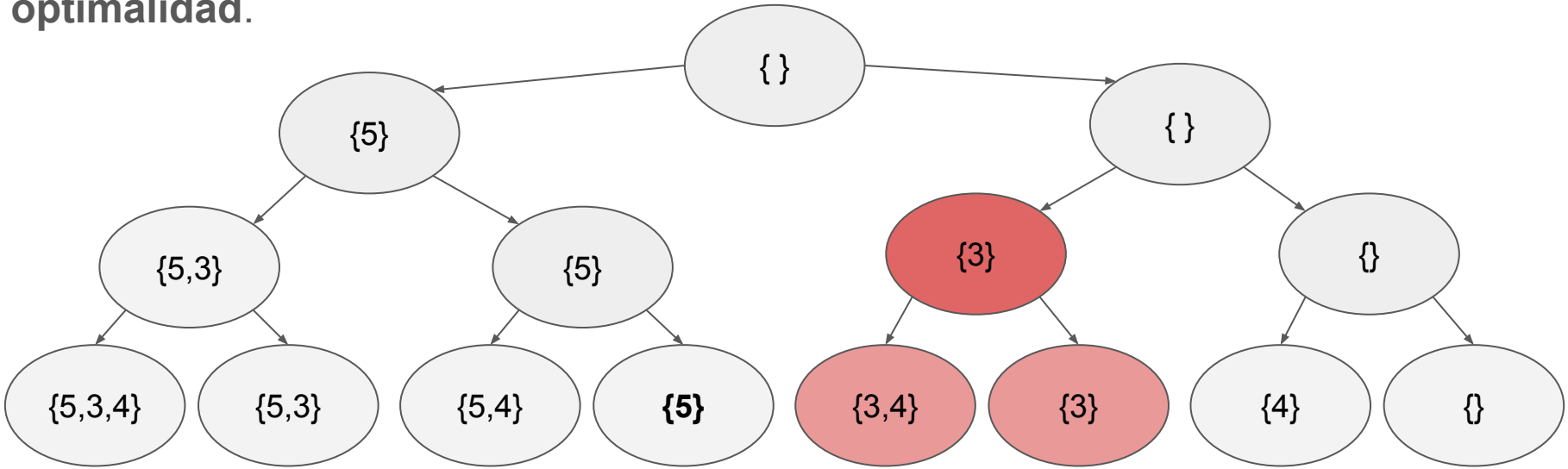
Backtracking (Poda por factibilidad 2)

Otra poda por factibilidad es considerar los elementos restantes. Supongamos que W = 8. En este caso, en el nodo {3}, solamente se puede sumar a lo sumo 7. Por lo tanto ninguna solución factible va a existir en su subárbol. Por lo tanto se puede podar.



Backtracking (Poda por optimalidad)

Ahora supongamos que ya tenemos una solución con cardinal 1. Y estamos en el nodo que representa a {3}. En este caso, sabemos que cualquier solución representada por su subárbol va a ser al menos tan buena como la que ya conocemos. Por lo tanto, podemos evitar explorar. Esto es una poda por **optimalidad**.



Repaso

- ¿Cómo se empieza un informe?
- ¿Qué es fuerza bruta?
- ¿Qué es backtracking?
- ¿Cómo se implementan?
- ¿Qué es una solución factible?
- ¿Qué es el espacio de soluciones? ¿Es único?
- ¿Qué es una poda por factibilidad?
- ¿Qué es una poda por optimalidad?