Bienvenidos a Técnicas de Diseño de Algoritmos (TDA)

- Facultad de Ciencias Exactas y Naturales, UBA.

Bienvenidos a Técnicas de Diseño de Algoritmos (TDA)

- Facultad de Ciencias Exactas y Naturales, UBA.
- Cuatrimestre 1ro 2024 - Turno Tarde.

Bienvenidos a Técnicas de Diseño de Algoritmos (TDA)

- Facultad de Ciencias Exactas y Naturales, UBA.
- Cuatrimestre 1ro 2024 - Turno Tarde.
- Licenciaturas:

Bienvenidos a Técnicas de Diseño de Algoritmos (TDA)

- Facultad de Ciencias Exactas y Naturales, UBA.
- Cuatrimestre 1ro 2024 - Turno Tarde.
- Licenciaturas:
  - Ciencias de la Computación.

Bienvenidos a Técnicas de Diseño de Algoritmos (TDA)

- Facultad de Ciencias Exactas y Naturales, UBA.
- Cuatrimestre 1ro 2024 - Turno Tarde.
- Licenciaturas:
  - Ciencias de la Computación.
  - Ciencias de Datos.

Programa del Curso - Febrero 2024

- Un curso intensivo en Técnicas de Diseño de Algoritmos.

Programa del Curso - Febrero 2024

- Un curso intensivo en Técnicas de Diseño de Algoritmos.
- Adaptado para estudiantes de Ciencias de la Computación y Ciencias de Datos.

Programa del Curso - Febrero 2024

- Un curso intensivo en Técnicas de Diseño de Algoritmos.
- Adaptado para estudiantes de Ciencias de la Computación y Ciencias de Datos.
- Objetivo: Desarrollar habilidades críticas en diseño y análisis de algoritmos.

Equipo Docente

- Profesores: Bonomo Flavia, Lin Min Chih, Platzer Emilio, Soulignac Francisco Juan.

Equipo Docente

- Profesores: Bonomo Flavia, Lin Min Chih, Platzer Emilio, Soulignac Francisco Juan.
- JTP: Iglesias Matias, Terlisky Pablo Ezequiel.

Equipo Docente

- Profesores: Bonomo Flavia, Lin Min Chih, Platzer Emilio, Soulignac Francisco Juan.
- JTP: Iglesias Matias, Terlisky Pablo Ezequiel.
- Ayudantes: Braier Julián (*), Brandwein Eric, Amster Martín (*), Companeetz Ezequiel, Dinkel Ayelen, Frassia Fernando, Laks Joaquín (*), Nores Manuel, Pages Julieta Belen (*), Raffo Leandro Javier, Umfurer Alfredo. (* = TM)

Detalles del Curso

- Lunes Aula Magna I, Miercoles Aula 5 Pab 2.

Detalles del Curso

- Lunes Aula Magna I, Miercoles Aula 5 Pab 2.
- Horarios de Clase:

Detalles del Curso

- Lunes Aula Magna I, Miercoles Aula 5 Pab 2.
- Horarios de Clase:
  - Mañana: Lunes y Miércoles de 9:00 a 13:00.

Detalles del Curso

- Lunes Aula Magna I, Miercoles Aula 5 Pab 2.
- Horarios de Clase:
  - Mañana: Lunes y Miércoles de 9:00 a 13:00.
  - Tarde: Lunes de 18:00 a 22:00, Miércoles de 17:00 a 21:00.

Detalles del Curso

- Lunes Aula Magna I, Miercoles Aula 5 Pab 2.
- Horarios de Clase:
  - Mañana: Lunes y Miércoles de 9:00 a 13:00.
  - Tarde: Lunes de 18:00 a 22:00, Miércoles de 17:00 a 21:00.
- Consultas: Miércoles en clase.

Correlatividades y Requerimientos

- Pre-requisitos: CBC, Introducción a la Programación, Álgebra, Algoritmos y Estructuras de Datos.

Correlatividades y Requerimientos

- Pre-requisitos: CBC, Introducción a la Programación, Álgebra, Algoritmos y Estructuras de Datos.
- Post-requisitos: Complejidad Computacional, Redes de Comunicaciones y Cómputo.

Correlatividades y Requerimientos

- Pre-requisitos: CBC, Introducción a la Programación, Álgebra, Algoritmos y Estructuras de Datos.
- Post-requisitos: Complejidad Computacional, Redes de Comunicaciones y Cómputo.
- Participación activa y dedicación a prácticas de programación recomendadas.

Evaluación y Exámenes

- Dos evaluaciones parciales tipo multiple choice en el sistema del Campus.

Evaluación y Exámenes

- Dos evaluaciones parciales tipo multiple choice en el sistema del Campus.
- Consideración de asistencia, participación en clase y realización de TPs para la evaluación.

Temario del Curso

- Basado en "Algorithm Design" por Jon Kleinberg y Éva Tardos.

Temario del Curso

- Basado en "Algorithm Design" por Jon Kleinberg y Éva Tardos.
- Contenidos incluirán: Análisis de algoritmos, Grafos, Divide & Conquer, Greedy, Programación Dinámica, y Flujo en Redes.

Trabajos Prácticos (TPs)

- Cuatro TPs individuales centrados en técnicas algorítmicas específicas.

Trabajos Prácticos (TPs)

- Cuatro TPs individuales centrados en técnicas algorítmicas específicas.
- Evaluación utilizando un sistema de juez automático.

Trabajos Prácticos (TPs)

- Cuatro TPs individuales centrados en técnicas algorítmicas específicas.
- Evaluación utilizando un sistema de juez automático.
- Libertad en la elección del lenguaje de programación, sujeto a factibilidad de evaluación.

Integración de IA en el Proceso Educativo de Consulta

- Objetivo: Mejorar la resolución de consultas en Técnicas de Diseño de Algoritmos (TDA).

Integración de IA en el Proceso Educativo de Consulta

- Objetivo: Mejorar la resolución de consultas en Técnicas de Diseño de Algoritmos (TDA).
- Respuesta a la demanda excepcional que desafía la calidad educativa.

Integración de IA en el Proceso Educativo de Consulta

- Objetivo: Mejorar la resolución de consultas en Técnicas de Diseño de Algoritmos (TDA).
- Respuesta a la demanda excepcional que desafía la calidad educativa.
- Implementación de la Plataforma de Consultas para TDA y Protocolo de Consultas.

La Importancia del Proceso de Consulta

- El diálogo en consultas fortalece el aprendizaje.

La Importancia del Proceso de Consulta

- El diálogo en consultas fortalece el aprendizaje.
- Desafíos: Escalabilidad, recursos limitados, adaptabilidad.

La Importancia del Proceso de Consulta

- El diálogo en consultas fortalece el aprendizaje.
- Desafíos: Escalabilidad, recursos limitados, adaptabilidad.
- El reto: 400 estudiantes para 10 docentes.

La Necesidad de Acción

- Potencial insatisfacción y fatiga entre estudiantes y docentes.

La Necesidad de Acción

- Potencial insatisfacción y fatiga entre estudiantes y docentes.
- Urgencia en medidas que asistan a docentes en la gestión educativa.

La Necesidad de Acción

- Potencial insatisfacción y fatiga entre estudiantes y docentes.
- Urgencia en medidas que asistan a docentes en la gestión educativa.
- Introducción de la Plataforma de Consultas como solución.

Plataforma de Consultas: Una Solución Innovadora

- Canalización digital de consultas para optimizar el proceso educativo.

Plataforma de Consultas: Una Solución Innovadora

- Canalización digital de consultas para optimizar el proceso educativo.
- Integración de un modelo de IA para tutoría instantánea.

Plataforma de Consultas: Una Solución Innovadora

- Canalización digital de consultas para optimizar el proceso educativo.
- Integración de un modelo de IA para tutoría instantánea.
- Etapa de desarrollo y pruebas por docentes, con planes de pruebas piloto.

Alcance y Limitaciones

- Testeo en TDA durante el 1er cuatrimestre de 2024.

Alcance y Limitaciones

- Testeo en TDA durante el 1er cuatrimestre de 2024.
- Limitaciones inherentes a los modelos de lenguaje grande (LLM).

Alcance y Limitaciones

- Testeo en TDA durante el 1er cuatrimestre de 2024.
- Limitaciones inherentes a los modelos de lenguaje grande (LLM).
- Importancia del análisis crítico de las respuestas de la IA.

Beneficios del Proyecto

- Para estudiantes: Disponibilidad 24/7, feedback instantáneo, experiencia personalizada.

Beneficios del Proyecto

- Para estudiantes: Disponibilidad 24/7, feedback instantáneo, experiencia personalizada.
- Para docentes: Reducción de horas de trabajo, datos para mejorar estrategias docentes.

Beneficios del Proyecto

- Para estudiantes: Disponibilidad 24/7, feedback instantáneo, experiencia personalizada.
- Para docentes: Reducción de horas de trabajo, datos para mejorar estrategias docentes.
- Para la institución: Innovación educativa, mejor reputación, decisiones basadas en datos.

Marco de Evaluación

- Evaluación centrada en mejoras basadas en feedback.

Marco de Evaluación

- Evaluación centrada en mejoras basadas en feedback.
- Planes para medir el impacto en aprendizaje tras mejoras iniciales.

Marco de Evaluación

- Evaluación centrada en mejoras basadas en feedback.
- Planes para medir el impacto en aprendizaje tras mejoras iniciales.
- Criterios: Rendimiento estudiantil, compromiso, satisfacción.

Inquietudes y Estrategias

- Calidad de retroalimentación de la IA: revisión periódica por educadores.

Inquietudes y Estrategias

- Calidad de retroalimentación de la IA: revisión periódica por educadores.
- Reemplazo de interacción humana: IA como complemento, no sustituto.

Inquietudes y Estrategias

- Calidad de retroalimentación de la IA: revisión periódica por educadores.
- Reemplazo de interacción humana: IA como complemento, no sustituto.
- Privacidad y seguridad de datos: cumplimiento de leyes de protección de datos.

Inquietudes y Estrategias

- Calidad de retroalimentación de la IA: revisión periódica por educadores.
- Reemplazo de interacción humana: IA como complemento, no sustituto.
- Privacidad y seguridad de datos: cumplimiento de leyes de protección de datos.
- Dependencia tecnológica y fallos técnicos: soporte técnico y planes de contingencia.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.
- Análisis de respuesta y preparación para la hora de consulta.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.
- Análisis de respuesta y preparación para la hora de consulta.
- Retroalimentación al sistema y decisiones sobre consultas adicionales.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.
- Análisis de respuesta y preparación para la hora de consulta.
- Retroalimentación al sistema y decisiones sobre consultas adicionales.
- Registro del código de consulta y ayudante asignado.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.
- Análisis de respuesta y preparación para la hora de consulta.
- Retroalimentación al sistema y decisiones sobre consultas adicionales.
- Registro del código de consulta y ayudante asignado.
- Las consultas se registran hasta las 16:00 del día de la práctica.

Protocolo de Consultas para Estudiantes

- Acceso a la Plataforma y selección de ejercicio.
- Registro y envío de dudas.
- Análisis de respuesta y preparación para la hora de consulta.
- Retroalimentación al sistema y decisiones sobre consultas adicionales.
- Registro del código de consulta y ayudante asignado.
- Las consultas se registran hasta las 16:00 del día de la práctica.
- Uso exclusivo de la Plataforma de Consultas para el turno tarde.

Consideraciones Finales y Novedades

- El sistema distribuye eficientemente el tiempo con docentes para más de 250 inscriptos.

Consideraciones Finales y Novedades

- El sistema distribuye eficientemente el tiempo con docentes para más de 250 inscriptos.
- Preparación para la Hora de Consulta: Resumen de dudas y respuestas recibidas.

Consideraciones Finales y Novedades

- El sistema distribuye eficientemente el tiempo con docentes para más de 250 inscriptos.
- Preparación para la Hora de Consulta: Resumen de dudas y respuestas recibidas.
- **El contenido oficial de la materia es el que está en la bibliografía.**

Consideraciones Finales y Novedades

- El sistema distribuye eficientemente el tiempo con docentes para más de 250 inscriptos.
- Preparación para la Hora de Consulta: Resumen de dudas y respuestas recibidas.
- **El contenido oficial de la materia es el que está en la bibliografía.**
- Novedades: Ya se puede testear la plataforma (ver link proporcionado).

Consideraciones Finales y Novedades

- El sistema distribuye eficientemente el tiempo con docentes para más de 250 inscriptos.
- Preparación para la Hora de Consulta: Resumen de dudas y respuestas recibidas.
- **El contenido oficial de la materia es el que está en la bibliografía.**
- Novedades: Ya se puede testear la plataforma (ver link proporcionado).
- En caso de problemas, escribir a matuteiglesias@gmail.com.

Detalles de Implementación y Contacto

- Desarrollo voluntario y prototipo funcional ya en uso.

Detalles de Implementación y Contacto

- Desarrollo voluntario y prototipo funcional ya en uso.
- Protocolo de consultas diseñado para optimizar el aprendizaje.

Detalles de Implementación y Contacto

- Desarrollo voluntario y prototipo funcional ya en uso.
- Protocolo de consultas diseñado para optimizar el aprendizaje.
- Pruebas piloto planificadas para el primer cuatrimestre de 2024.

Detalles de Implementación y Contacto

- Desarrollo voluntario y prototipo funcional ya en uso.
- Protocolo de consultas diseñado para optimizar el aprendizaje.
- Pruebas piloto planificadas para el primer cuatrimestre de 2024.
- Contacto para soporte y sugerencias: matuteiglesias@gmail.com.

Backtracking

Algoritmos y Estructuras de Datos III

Santiago Cifuentes

Departamento de computación
FCEN – UBA

Agosto 2023

Técnicas algorítmicas

- Fuerza Bruta / Búsqueda exhaustiva
- *Backtracking*
- *Divide&Conquer*
- Algoritmos Golosos
- Programación Dinámica
- Heurísticas y algoritmos aproximados.

Técnicas algorítmicas

- Fuerza Bruta / Búsqueda exhaustiva
- *Backtracking*
- *Divide&Conquer*
- Algoritmos Golosos
- Programación Dinámica
- Heurísticas y algoritmos aproximados.

Técnicas algorítmicas

- Fuerza Bruta / Búsqueda exhaustiva
- *Backtracking*
- *Divide&Conquer*
- Algoritmos Golosos
- Programación Dinámica
- Heurísticas y algoritmos aproximados.

Fuerza Bruta / Búsqueda exhaustiva

- Para problemas de búsqueda en un conjunto S.

Fuerza Bruta / Búsqueda exhaustiva

- Para problemas de búsqueda en un conjunto S.
- Queremos hacer algo con los elementos que cumpla una cierta propiedad P.

Fuerza Bruta / Búsqueda exhaustiva

- Para problemas de búsqueda en un conjunto S.
- Queremos hacer algo con los elementos que cumpla una cierta propiedad P.
- La idea más simple: recorremos todo S evaluando P en cada elemento.

Fuerza Bruta / Búsqueda exhaustiva

- Para problemas de búsqueda en un conjunto S.
- Queremos hacer algo con los elementos que cumpla una cierta propiedad P.
- La idea más simple: recorremos todo S evaluando P en cada elemento.
- La complejidad en general será $\Omega(|S|)$.

Esquema de Fuerza Bruta

for x $\in$ S **do**:
    if P(x):
        procesar x

- Hay que definir quiénes son S, P y **procesar**.

Esquema de Fuerza Bruta

for x $\in$ S **do**:
    if P(x):
        procesar x

- Hay que definir quiénes son S, P y **procesar**.
- Ejemplo: S es el conjunto de tableros de ajedrez con 8 reinas, P verifica que no se ataquen entre ellas, y **procesar** lleva la cuenta de la cantidad de tableros.

Esquema de Fuerza Bruta

for x $\in$ S **do**:
    if P(x):
        procesar x

- Hay que definir quiénes son S, P y **procesar**.
- Ejemplo: S es el conjunto de tableros de ajedrez con 8 reinas, P verifica que no se ataquen entre ellas, y **procesar** lleva la cuenta de la cantidad de tableros.
- Subproblema: ¿Cómo se genera S?

Backtracking

- Es una técnica para generar espacios de búsqueda "recursivos". En particular, lo hace mediante la extensión de **soluciones parciales**.

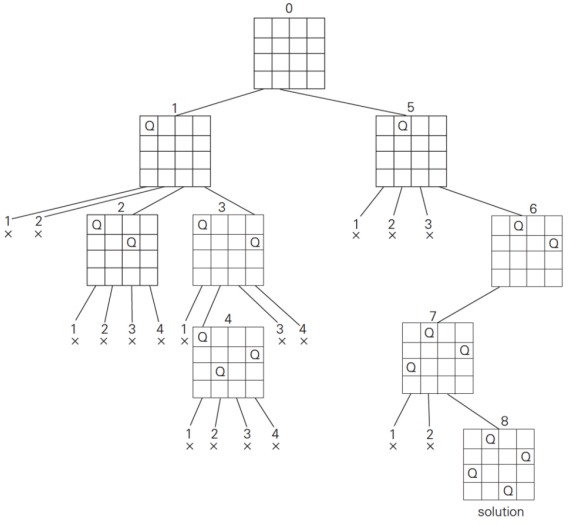
Backtracking

- Es una técnica para generar espacios de búsqueda "recursivos". En particular, lo hace mediante la extensión de **soluciones parciales**.
- La idea es definir este método de extensión, y mediante recursión generar *de forma ordenada* el espacio de soluciones S.

Backtracking

- Es una técnica para generar espacios de búsqueda "recursivos". En particular, lo hace mediante la extensión de **soluciones parciales**.
- La idea es definir este método de extensión, y mediante recursión generar *de forma ordenada* el espacio de soluciones S.
- La extensión muchas veces es una operación "local", y es fácil de definir e implementar.

Backtracking



Backtracking

```
algoritmo BT(a, k)
    si a es solución entonces
        procesar(a)
        retornar
    sino
        para cada a' ∈ Sucesores(a, k)
            BT(a', k + 1)
        fin para
    fin si
    retornar
```

- La generación de S se redujo a implementar *Sucesores*.

Backtracking

```
algoritmo BT(a, k)
    si a es solución entonces
        procesar(a)
        retornar
    sino
        para cada a' ∈ Sucesores(a, k)
            BT(a', k + 1)
        fin para
    fin si
    retornar
```

- La generación de S se redujo a implementar *Sucesores*.
- ¿Cómo es *Sucesores* para el caso del problema de las reinas?

Ejercicio: Suma de Subconjuntos

Enunciado

Dado un conjunto $S = \{n_1, n_2, ..., n_N\}$ y un valor objetivo T, determinar si existe un subconjunto de S cuya suma sea exactamente T.

Ejercicio: Suma de Subconjuntos

Enunciado

Dado un conjunto $S = \{n_1, n_2, ..., n_N\}$ y un valor objetivo T, determinar si existe un subconjunto de S cuya suma sea exactamente T.

Ejemplo: Dado $T = 12$ y un conjunto $S = \{6, 12, 6\}$, una solución válida es el subconjunto $\{6, 6\}$ porque $6 + 6 = 12$.

Análisis del Espacio de Búsqueda

- Espacio de búsqueda: Todos los subconjuntos de S. ¿Cuántos hay? 2^N.

Análisis del Espacio de Búsqueda

- Espacio de búsqueda: Todos los subconjuntos de S. ¿Cuántos hay? 2^N.
- Cada subconjunto se verifica contra el valor objetivo T.

Generación Recursiva de Subconjuntos

- Método recursivo para explorar el espacio de soluciones usando backtracking.

Generación Recursiva de Subconjuntos

- Método recursivo para explorar el espacio de soluciones usando backtracking.

Generación Recursiva de Subconjuntos

- Método recursivo para explorar el espacio de soluciones usando backtracking.

Función recursiva básica (pseudocódigo simplificado):

[Inserte el pseudocódigo de la función recursiva aquí]

Árbol de Decisión para el Problema de Suma de Subconjuntos

Visualización del árbol de decisiones para incluir o no incluir cada elemento.

Árbol de Decisión para el Problema de Suma de Subconjuntos

Visualización del árbol de decisiones para incluir o no incluir cada elemento.

[Inserte visualización del árbol aquí]

Pseudocódigo: Backtracking para Suma de Subconjuntos

Estrategia de backtracking aplicada al problema de suma de subconjuntos.

Algorithm Backtracking para Suma de Subconjuntos

1: [Inserte el pseudocódigo detallado aquí]

Análisis de Complejidad

- Número total de nodos en el árbol de decisión: $O(2^N)$.

Análisis de Complejidad

- Número total de nodos en el árbol de decisión: $O(2^N)$.
- Operaciones por nodo: una cantidad constante para verificar la suma y decidir la poda.

Análisis de Complejidad

- Número total de nodos en el árbol de decisión: $O(2^N)$.
- Operaciones por nodo: una cantidad constante para verificar la suma y decidir la poda.
- Complejidad temporal: $O(2^N)$.

Análisis de Complejidad

- Número total de nodos en el árbol de decisión: $O(2^N)$.
- Operaciones por nodo: una cantidad constante para verificar la suma y decidir la poda.
- Complejidad temporal: $O(2^N)$.
- Complejidad espacial: $O(N)$ por la profundidad del árbol de recursión.

Consideraciones Finales y Podas

Discusión sobre cómo las podas pueden mejorar significativamente la eficiencia de la búsqueda:

- Factibilidad: detener la exploración si la suma parcial supera T.

Consideraciones Finales y Podas

Discusión sobre cómo las podas pueden mejorar significativamente la eficiencia de la búsqueda:

- Factibilidad: detener la exploración si la suma parcial supera T.
- Optimalidad: si alcanzamos T antes de considerar todos los elementos, detenemos esa rama.

Ejercicio: *Prime ring*

Prime Ring

Dados N números naturales $p_0, \ldots, p_{N-1}$, con $1 < p_i < 10N$, queremos saber cuántas permutaciones j de ellos hay que cumplan que $p_{j_i} + p_{(j_{i+1} \mod n)}$ sea primo para todo $0 \leq i \leq n-1$

Ejercicio: *Prime ring*

Prime Ring

Dados N números naturales $p_0, \ldots, p_{N-1}$, con $1 < p_i < 10N$, queremos saber cuántas permutaciones j de ellos hay que cumplan que $p_{j_i} + p_{(j_{i+1} \mod n)}$ sea primo para todo $0 \leq i \leq n-1$

- Lo vamos a resolver con backtracking.

Ejercicio: *Prime ring*

Prime Ring

Dados N números naturales $p_0, \ldots, p_{N-1}$, con $1 < p_i < 10N$, queremos saber cuántas permutaciones j de ellos hay que cumplan que $p_{j_i} + p_{(j_{i+1} \mod n)}$ sea primo para todo $0 \leq i \leq n-1$

- Lo vamos a resolver con backtracking.
- ¿Cuál es el espacio de búsqueda?

Ejercicio: *Prime ring*

Prime Ring

Dados N números naturales $p_0, \ldots, p_{N-1}$, con $1 < p_i < 10N$, queremos saber cuántas permutaciones j de ellos hay que cumplan que $p_{j_i} + p_{(j_{i+1} \mod n)}$ sea primo para todo $0 \leq i \leq n-1$

- Lo vamos a resolver con backtracking.
- ¿Cuál es el espacio de búsqueda?
- ¿Cuáles son las soluciones parciales?

Ejercicio: *Prime ring*

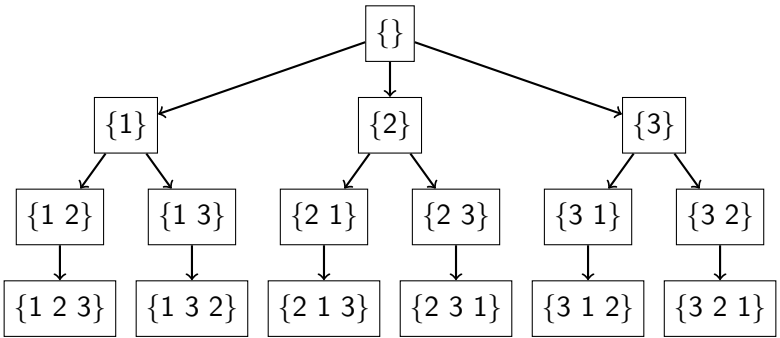
Prime Ring

Dados N números naturales $p_0, \ldots, p_{N-1}$, con $1 < p_i < 10N$, queremos saber cuántas permutaciones j de ellos hay que cumplan que $p_{j_i} + p_{(j_{i+1} \mod n)}$ sea primo para todo $0 \leq i \leq n-1$

- Lo vamos a resolver con backtracking.
- ¿Cuál es el espacio de búsqueda?
- ¿Cuáles son las soluciones parciales?
- ¿Cuál es la operación de extensión?

Árbol de *Prime ring*

- Árbol para $n = 3$, si $p = [1, 2, 3]$.



Árbol de *Prime ring*

- Cada nodo interno del piso l es una permutación de un subconjunto de i números.

Árbol de *Prime ring*

- Cada nodo interno del piso I es una permutación de un subconjunto de i números.
- En las hojas verificamos que se cumpla la condición de primalidad.

Prime ring

La función que hay que implementar entonces es:

$$primeRing(I) = \begin{cases} esValida(I) & \text{si } |I| = N \\ \sum_{p_i \notin I} primeRing(I \oplus p_i) & \text{cc} \end{cases}$$

'La cantidad de permutaciones que extienden a I, usan todos los elementos de p y generan un anillo de primos'

Prime ring

La función que hay que implementar entonces es:

$$primeRing(I) = \begin{cases} esValida(I) & \text{si } |I| = N \\ \sum_{p_i \notin I} primeRing(I \oplus p_i) & \text{cc} \end{cases}$$

'La cantidad de permutaciones que extienden a I, usan todos los elementos de p y generan un anillo de primos'

La solución al problema es $primeRing(\{\})$

Podas

- ¿Podas?

Podas

- ¿Podas?
- Podríamos verificar la condición de primalidad durante la selección de sucesores.

Podas

- ¿Podas?
- Podríamos verificar la condición de primalidad durante la selección de sucesores.
- El árbol cambia: ahora las soluciones parciales son las permutaciones de los subconjuntos que cumplen la condición de primalidad.

Podas

- ¿Podas?
- Podríamos verificar la condición de primalidad durante la selección de sucesores.
- El árbol cambia: ahora las soluciones parciales son las permutaciones de los subconjuntos que cumplen la condición de primalidad.
- ¿Hay que verificar algo en las hojas?
- En las hojas solo tenemos que verificar que cierre bien el anillo.

Podas

La función queda entonces como

$$primeRing(l) = \begin{cases} esPrimo(ultimo(l) + primero(l)) & \text{si } |l| = N \\ \sum_{\substack{p_i \notin l \\ esPrimo(p_i + ultimo(l))}} primeRing(l \oplus p_i) & \text{cc} \end{cases}$$

Podas

La función queda entonces como

$$primeRing(l) = \begin{cases} esPrimo(ultimo(l) + primero(l)) & \text{si } |l| = N \\ \sum_{\substack{p_i \notin l \\ esPrimo(p_i + ultimo(l))}} primeRing(l \oplus p_i) & \text{cc} \end{cases}$$

Más podas

- Hay algunas podas interesantes que se pueden usar debido a que este es un problema de conteo.

Más podas

- Hay algunas podas interesantes que se pueden usar debido a que este es un problema de conteo.
- Podemos explotar simetrías: dada una permutación válida, se pueden obtener otras moviendo los elementos a la derecha x unidades.

Más podas

- Hay algunas podas interesantes que se pueden usar debido a que este es un problema de conteo.
- Podemos explotar simetrías: dada una permutación válida, se pueden obtener otras moviendo los elementos a la derecha x unidades.
- Podemos suponer fijo el primer elemento, y multiplicar por N la cantidad de permutaciones con ese elemento primero.

Más podas

La función queda entonces como

$$primeRing(l) = \begin{cases} esPrimo(ultimo(l) + primero(l)) & \text{si } |l| = N \\ \sum\limits_{\substack{P_i \notin l \\ esPrimo(P_i + ultimo(l))}} primeRing(l \oplus i) & \text{cc} \end{cases}$$

La función no cambia, pero ahora sabemos que la solución se puede escribir como $N * primeRing([p_0])$

Complejidad de la solución de *Prime Ring*

- ¿Cuántos nodos tiene el árbol de recursión?

Complejidad de la solución de *Prime Ring*

- ¿Cuántos nodos tiene el árbol de recursión?
- El árbol tiene $O(n-1!)$ nodos (en la práctica tienen que demostrar un caso similar).

Complejidad de la solución de *Prime Ring*

- ¿Cuántos nodos tiene el árbol de recursión?
- El árbol tiene $O(n-1!)$ nodos (en la práctica tienen que demostrar un caso similar).
- En cada nodo hacemos $O(n)$ operaciones, y en particular $O(n)$ llamados a *esPrimo*.

Complejidad de la solución de *Prime Ring*

- ¿Cuántos nodos tiene el árbol de recursión?
- El árbol tiene $O(n-1!)$ nodos (en la práctica tienen que demostrar un caso similar).
- En cada nodo hacemos $O(n)$ operaciones, y en particular $O(n)$ llamados a *esPrimo*.
- Si *esPrimo* es $O(1)$ (podemos precalcular la criba de Heratóstenes hasta $20n$ en $O(n \log\log n)$[1]), la complejidad final es $O(n \log\log n + (n-1)!\ n) = O(n!)$.

[1] Esto se puede hacer en $O(n)$, ver `https://cp-algorithms.com/algebra/prime-sieve-linear.html`

Detalles adicionales

- En realidad la complejidad es menor, ya que en cada paso hay a lo sumo $\frac{n}{2}$ opciones por la paridad.

Detalles adicionales

- En realidad la complejidad es menor, ya que en cada paso hay a lo sumo $\frac{n}{2}$ opciones por la paridad.
- Aparte, queda por explotar la simetría que surge de invertir las soluciones.

Sudoku

Enunciado

Dado un tablero de Sudoku de $N \times N$ con algunas casillas ocupadas hay que decidir si se puede completar de forma que sea el resultado final sea un tablero válido.

Sudoku

Enunciado

Dado un tablero de Sudoku de $N \times N$ con algunas casillas ocupadas hay que decidir si se puede completar de forma que sea el resultado final sea un tablero válido.

- ¿Cuáles son las soluciones parciales?

Sudoku

Enunciado

Dado un tablero de Sudoku de $N \times N$ con algunas casillas ocupadas hay que decidir si se puede completar de forma que sea el resultado final sea un tablero válido.

- ¿Cuáles son las soluciones parciales?
- ¿Cuál es la función de extensión?

Sudoku

Enunciado

Dado un tablero de Sudoku de $N \times N$ con algunas casillas ocupadas hay que decidir si se puede completar de forma que sea el resultado final sea un tablero válido.

- ¿Cuáles son las soluciones parciales?
- ¿Cuál es la función de extensión?
- ¿Qué verificamos en las hojas?

Sudoku

$$sudoku(T,(i,j)) = \begin{cases} esValido(T) & \text{si } i = N \\ sudoku(T, sig(i,j)) & \text{si } T[i][j] \neq 0 \\ \bigvee_{1 \leq k \leq N} sudoku(T \oplus ((i,j) \to k), sig(i,j)) & \text{cc} \end{cases}$$

Sudoku

$$sudoku(T,(i,j)) = \begin{cases} esValido(T) & \text{si } i = N \\ sudoku(T, sig(i,j)) & \text{si } T[i][j] \neq 0 \\ \bigvee_{1 \leq k \leq N} sudoku(T \oplus ((i,j) \to k), sig(i,j)) & \text{cc} \end{cases}$$

La solución es $sudoku(T, 0, 0)$

Sudoku

- ¿Cuántos nodos tiene el árbol?

Sudoku

- ¿Cuántos nodos tiene el árbol?
- ¿Cuántas operaciones hacemos en los nodos internos?

Sudoku

- ¿Cuántos nodos tiene el árbol?
- ¿Cuántas operaciones hacemos en los nodos internos?
- ¿Y en las hojas?

Sudoku

- ¿Cuántos nodos tiene el árbol?
- ¿Cuántas operaciones hacemos en los nodos internos?
- ¿Y en las hojas?
- La complejidad final se puede acotar por $O(n^{n^2} n^2)$

Podas

- ¿Qué podas podemos implementar?

Podas

- ¿Qué podas podemos implementar?
- No pongamos números que ya están prohibidos. Para eso revisamos las filas, columnas y subcuadrados en cada paso.

Podas

- ¿Qué podas podemos implementar?
- No pongamos números que ya están prohibidos. Para eso revisamos las filas, columnas y subcuadrados en cada paso.
- ¿Hace falta ir en orden?

Podas

- ¿Qué podas podemos implementar?
- No pongamos números que ya están prohibidos. Para eso revisamos las filas, columnas y subcuadrados en cada paso.
- ¿Hace falta ir en orden?
- No, usemos siempre la posición mas condicionada.

Sudoku

$$sudoku(T) = \begin{cases} esValido(T) & \text{si } mas\_cond(T) = \bot \\ \bigvee_{k \in cand(mas\_cond(T))} sudoku(T \oplus (max\_cond(T) \to k)) & \text{cc} \end{cases}$$

Sudoku

$$sudoku(T) = \begin{cases} esValido(T) & \text{si } mas\_cond(T) = \bot \\ \bigvee_{k \in cand(mas\_cond(T))} sudoku(T \oplus (max\_cond(T) \to k)) & \text{cc} \end{cases}$$

- La solución es $sudoku(T)$

Sudoku

$$sudoku(T) = \begin{cases} esValido(T) & \text{si } mas\_cond(T) = \bot \\ \bigvee_{k \in cand(mas\_cond(T))} sudoku(T \oplus (max\_cond(T) \to k)) & \text{cc} \end{cases}$$

- La solución es $sudoku(T)$

Sudoku

$$sudoku(T) = \begin{cases} esValido(T) & \text{si } mas\_cond(T) = \bot \\ \bigvee_{k \in cand(mas\_cond(T))} sudoku(T \oplus (max\_cond(T) \to k)) & \text{cc} \end{cases}$$

- La solución es $sudoku(T)$
- ¿La complejidad cambia?

Sudoku

Pruning Condition		Puzzle Complexity		
`next_square`	`possible_values`	Easy	Medium	Hard
arbitrary	local count	1,904,832	863,305	never finished
arbitrary	look ahead	127	142	12,507,212
most constrained	local count	48	84	1,243,838
most constrained	look ahead	48	65	10,374

The algorithm design manual, Skiena

Dobra

Enunciado

Tenemos una cadena I de N caracteres que son letras en mayúscula o bien comodines _. Queremos ver cuántas cadenas podemos formar si reemplazamos los comodines por mayúsculas, teniendo en cuenta que:

- No queremos que haya 3 vocales ni 3 consonantes seguidas.
- Tiene que haber una L en la palabra.

Dobra

Enunciado

Tenemos una cadena I de N caracteres que son letras en mayúscula o bien comodines _. Queremos ver cuántas cadenas podemos formar si reemplazamos los comodines por mayúsculas, teniendo en cuenta que:

- No queremos que haya 3 vocales ni 3 consonantes seguidas.
- Tiene que haber una L en la palabra.

- ¿Un posible espacio de búsqueda? ¿Soluciones parciales? ¿Extensión?

Dobra

Enunciado

Tenemos una cadena I de N caracteres que son letras en mayúscula o bien comodines _. Queremos ver cuántas cadenas podemos formar si reemplazamos los comodines por mayúsculas, teniendo en cuenta que:

- No queremos que haya 3 vocales ni 3 consonantes seguidas.
- Tiene que haber una L en la palabra.

- ¿Un posible espacio de búsqueda? ¿Soluciones parciales? ¿Extensión?
- ¿Qué verificamos en las hojas?

Dobra

Enunciado

Tenemos una cadena I de N caracteres que son letras en mayúscula o bien comodines _. Queremos ver cuántas cadenas podemos formar si reemplazamos los comodines por mayúsculas, teniendo en cuenta que:

- No queremos que haya 3 vocales ni 3 consonantes seguidas.
- Tiene que haber una L en la palabra.

- ¿Un posible espacio de búsqueda? ¿Soluciones parciales? ¿Extensión?
- ¿Qué verificamos en las hojas?
- ¿Cuántas opciones tenemos en cada _?

Dobra

$$dobra(i, l) = \begin{cases} verificar(l) & \text{si } i = n \\ dobra(i+1, l) & \text{si } l[i] \neq \_ \\ \sum_{c \in MAYUS} dobra(i+1, l \oplus (i \to c)) & cc \end{cases}$$

Dobra

$$dobra(i, l) = \begin{cases} verificar(l) & \text{si } i = n \\ dobra(i+1, l) & \text{si } l[i] \neq \_ \\ \sum_{c \in MAYUS} dobra(i+1, l \oplus (i \to c)) & cc \end{cases}$$

- ¿Complejidad?

Dobra

$$dobra(i, l) = \begin{cases} verificar(l) & \text{si } i = n \\ dobra(i+1, l) & \text{si } l[i] \neq \_ \\ \sum_{c \in MAYUS} dobra(i+1, l \oplus (i \to c)) & cc \end{cases}$$

- ¿Complejidad?
- El árbol tiene una cantidad de nodos acotable por $O(26^N)$. En las hojas hacemos $O(N)$ operaciones.

Dobra

$$dobra(i, l) = \begin{cases} verificar(l) & \text{si } i = n \\ dobra(i+1, l) & \text{si } l[i] \neq \_ \\ \sum_{c \in MAYUS} dobra(i+1, l \oplus (i \to c)) & cc \end{cases}$$

- ¿Complejidad?
- El árbol tiene una cantidad de nodos acotable por $O(26^N)$. En las hojas hacemos $O(N)$ operaciones.

¿Qué llamado resuelve el problem?

Dobra

$$dobra(i, l) = \begin{cases} verificar(l) & \text{si } i = n \\ dobra(i+1, l) & \text{si } l[i] \neq \_ \\ \sum_{c \in MAYUS} dobra(i+1, l \oplus (i \to c)) & cc \end{cases}$$

- ¿Complejidad?
- El árbol tiene una cantidad de nodos acotable por $O(26^N)$. En las hojas hacemos $O(N)$ operaciones.

¿Qué llamado resuelve el problem? $dobra(0, l)$

Podas

- ¿Qué podas podemos hacer?

Podas

- ¿Qué podas podemos hacer?
- Vamos verificando si los reemplazos que hacemos de los comodines son válidos _.

Podas

- ¿Qué podas podemos hacer?
- Vamos verificando si los reemplazos que hacemos de los comodines son válidos _.
- Por otro lado, ¿Importa *cuál vocal / consonante* usamos?

Podas

- ¿Qué podas podemos hacer?
- Vamos verificando si los reemplazos que hacemos de los comodines son válidos _.
- Por otro lado, ¿Importa *cuál vocal / consonante* usamos?
- Qué vocal se usa es irrelevante. Solo importa el hecho de que usamos una vocal, y entonces podemos usar una vocal cualquiera y multiplicar por 5.

Podas

- ¿Qué podas podemos hacer?
- Vamos verificando si los reemplazos que hacemos de los comodines son válidos _.
- Por otro lado, ¿Importa *cuál vocal / consonante* usamos?
- Qué vocal se usa es irrelevante. Solo importa el hecho de que usamos una vocal, y entonces podemos usar una vocal cualquiera y multiplicar por 5.
- ¿Podemos hacer lo mismo para las consonantes?

Podas

- ¿Qué podas podemos hacer?
- Vamos verificando si los reemplazos que hacemos de los comodines son válidos _.
- Por otro lado, ¿Importa *cuál vocal / consonante* usamos?
- Qué vocal se usa es irrelevante. Solo importa el hecho de que usamos una vocal, y entonces podemos usar una vocal cualquiera y multiplicar por 5.
- ¿Podemos hacer lo mismo para las consonantes?
- Hay que controlar si usamos o no una *L*.

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:verificamos que esté bien, y en caso afirmativo seguimos con $dobra(i + 1, l, tiene\_L \vee l[i] == L)$.
- Si $l[i] = \_$, no puede ir una consonante, pero si una vocal:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:verificamos que esté bien, y en caso afirmativo seguimos con $dobra(i + 1, l, tiene\_L \vee l[i] == L)$.
- Si $l[i] = \_$, no puede ir una consonante, pero si una vocal: hacemos recursión con $5 * dobra(i + 1, l \oplus (i \to A), tiene\_L)$
- Si $l[i] = \_$, no puede ir una vocal, pero si una consonante:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:verificamos que esté bien, y en caso afirmativo seguimos con $dobra(i + 1, l, tiene\_L \vee l[i] == L)$.
- Si $l[i] = \_$, no puede ir una consonante, pero si una vocal: hacemos recursión con $5 * dobra(i + 1, l \oplus (i \to A), tiene\_L)$
- Si $l[i] = \_$, no puede ir una vocal, pero si una consonante: hacemos dos recursiones, devolviendo $20 * dobra(i + 1, l \oplus (i \to B), tiene\_L) + dobra(i + 1, l \oplus (i \to L), true)$.
- Si $l[i] = \_$ y podemos tanto vocal como consonante:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:verificamos que esté bien, y en caso afirmativo seguimos con $dobra(i + 1, l, tiene\_L \vee l[i] == L)$.
- Si $l[i] = \_$, no puede ir una consonante, pero si una vocal: hacemos recursión con $5 * dobra(i + 1, l \oplus (i \to A), tiene\_L)$
- Si $l[i] = \_$, no puede ir una vocal, pero si una consonante: hacemos dos recursiones, devolviendo $20 * dobra(i + 1, l \oplus (i \to B), tiene\_L) + dobra(i + 1, l \oplus (i \to L), true)$.
- Si $l[i] = \_$ y podemos tanto vocal como consonante: sumamos los casos anteriores.
- Caso contrario:

Dobra

Los casos de la función recursiva $dobra(i, l, tiene\_L)$ quedan en

- Si $i == N$: Devolvemos $tiene\_L$.
- Si $l[i] \neq \_$:verificamos que esté bien, y en caso afirmativo seguimos con $dobra(i + 1, l, tiene\_L \vee l[i] == L)$.
- Si $l[i] = \_$, no puede ir una consonante, pero si una vocal: hacemos recursión con $5 * dobra(i + 1, l \oplus (i \to A), tiene\_L)$
- Si $l[i] = \_$, no puede ir una vocal, pero si una consonante: hacemos dos recursiones, devolviendo $20 * dobra(i + 1, l \oplus (i \to B), tiene\_L) + dobra(i + 1, l \oplus (i \to L), true)$.
- Si $l[i] = \_$ y podemos tanto vocal como consonante: sumamos los casos anteriores.
- Caso contrario: devolvemos 0;

Dobra

- ¿Cuál es la complejidad de esta nueva solución?

Dobra

- ¿Cuál es la complejidad de esta nueva solución?
- Hay a lo sumo 3^n nodos, y hacemos $O(1)$ operaciones en cada paso.

Dobra

- ¿Cuál es la complejidad de esta nueva solución?
- Hay a lo sumo 3^n nodos, y hacemos $O(1)$ operaciones en cada paso.
- Complejidad final: $O(3^n)$.

Contando bien

- ¿El árbol siempre se abre en 3?

Contando bien

- ¿El árbol siempre se abre en 3?
- Se puede probar que este árbol tiene $\Theta(n2^n)$ nodos en el peor caso, y por lo tanto la complejidad es un poco mejor.

Contando bien

- ¿El árbol siempre se abre en 3?
- Se puede probar que este árbol tiene $\Theta(n2^n)$ nodos en el peor caso, y por lo tanto la complejidad es un poco mejor.
- El árbol de recursión funciona como una herramienta para acotar la complejidad del algoritmo.