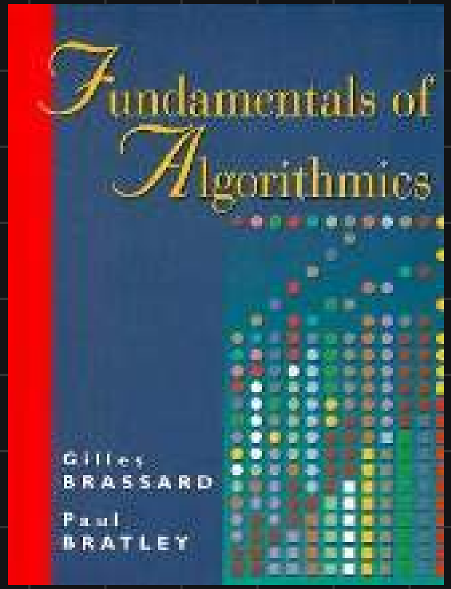
Backtracking

Técnicas de Diseño de Algoritmos

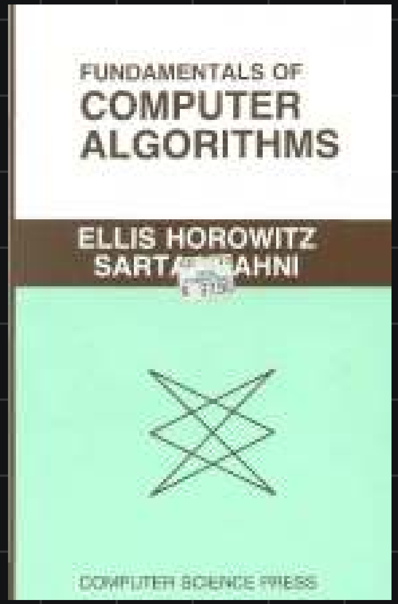
1er Cuatrimestre 2024

Bibliografía

El CLRS y el Kleinberg | Tardos no tienen secciones dedicadas a backtracking, se puede consultar:



Brassard, Bratley, Fundamentals of Algorithmics



Horowitz, Sahni, Fundamentals of computer Algorithms

Repaso de la teoría

Dado un problema Π, Π se puede resolver haciendo una busqueda exhaustiva entre sus soluciones candidatas.

La técnica de Backtracking hace una exploración ordenada de este espacio de soluciones por medio de la extensión de soluciones parciales.

Repaso de la teoría

Cada posible extensión de una solución parcial es explorada haciendo recursión sobre la nueva solución extendida.

Podemos representar visualmente esta exploración como el recorrido en profundidad de un árbol.

Frecuentemente es posible aplicar podas al árbol para reducir el espacio de búsqueda.

Repaso de la teoría

Podemos pensar una solución candidata como una tupla $(x_1, \ldots, x_k)$ donde x_i es una decisión de entre un conjunto S_i de alternativas

Pensamos la recursión como el problema donde algunas de estas decisiones ya vienen tomadas

Tres Problemas:

- CD
- Sudoku
- Dobra

Problema del CD

(Basado en https://vjudge.net/problem/UVA-624)

Se tiene un CD que soporta hasta P minutos de música, y dado un conjunto con N canciones de duración p_i $(0 \leq i \leq N-1,\ p_i \in \mathbb{N})$ queremos encontrar la mayor cantidad de minutos de música que podemos poner en el CD usando temas del conjunto sin repetir.

(Este problema es un caso particular de Knapsack, visto en la teórica. Es un problema de <u>optimización</u>)

CD - Ejemplo

$P = 5$ $N = 3$ $p_0 = 1$ $p_1 = 4$ $p_2 = 2$

Solución: 5 $(1+4)$

CD - Soluciones

¿Cuáles son las soluciones candidatas?

Todas las maneras de seleccionar entre las N canciones

¿Cuántas son?

Tantas como subconjuntos de un conjunto de N elementos

¿Cómo puedo construírlas a partir de soluciones parciales?

Eligiendo agregar o no agregar cada canción

CD - Formulación recursiva del conjunto de partes

$$\mathcal{P}(\phi) = \{\phi\}$$

$$\mathcal{P}(c:C) = c \times \mathcal{P}(C) \cup \mathcal{P}(C)$$

C con c agregado

agregar a todos los elementos

El conjunto se particiona entre los subconjuntos que contienen a c y los que no.

CD - Formulación Recursiva

Idea de la semántica: "¿Cuál es la máxima cantidad de minutos si mi capacidad es K y uso los temas del i en adelante?"

CD - Formulación Recursiva

- La cantidad máxima de minutos usando los temas desde el i teniendo capacidad K se puede computar considerando que se saltea el tema i o si se agrega el tema i

tomo el máximo de estas dos alternativas

→ Si se saltea, la respuesta es la cantidad de minutos suponiendo capacidad K empezando del tema i+1.

→ Si se agrega, la respuesta es la cantidad de minutos suponiendo capacidad (K-Pi) empezando del tema i+1, y sumando Pi a ese valor (pues el tema estará en el disco)

alternativa: se podría definir que el caso base es cuando queda un tema

- Si ya no quedan temas por decidir (i=N), entonces si el CD aún tiene espacio ($K \geq 0$), la respuesta es 0 (no lo puedo llenar más). Si en cambio ($K<0$), el CD ya se pasó de capacidad, luego quiero que mi respuesta sea el mínimo valor posible, para que cuando se la compare con max con otra, siempre se elija la otra. El valor que cumple esto es $-\infty$.

CD - Formulación Recursiva

$$CD(i,k) = \begin{cases} -\infty & \text{si } i = N \text{ y } K < 0 \\ 0 & \text{si } i = N \text{ y } K \geq 0 \\ \max\left(CD(i+1, K),\ CD(i+1, K-P_i) + P_i\right) & \text{c.c.} \end{cases}$$

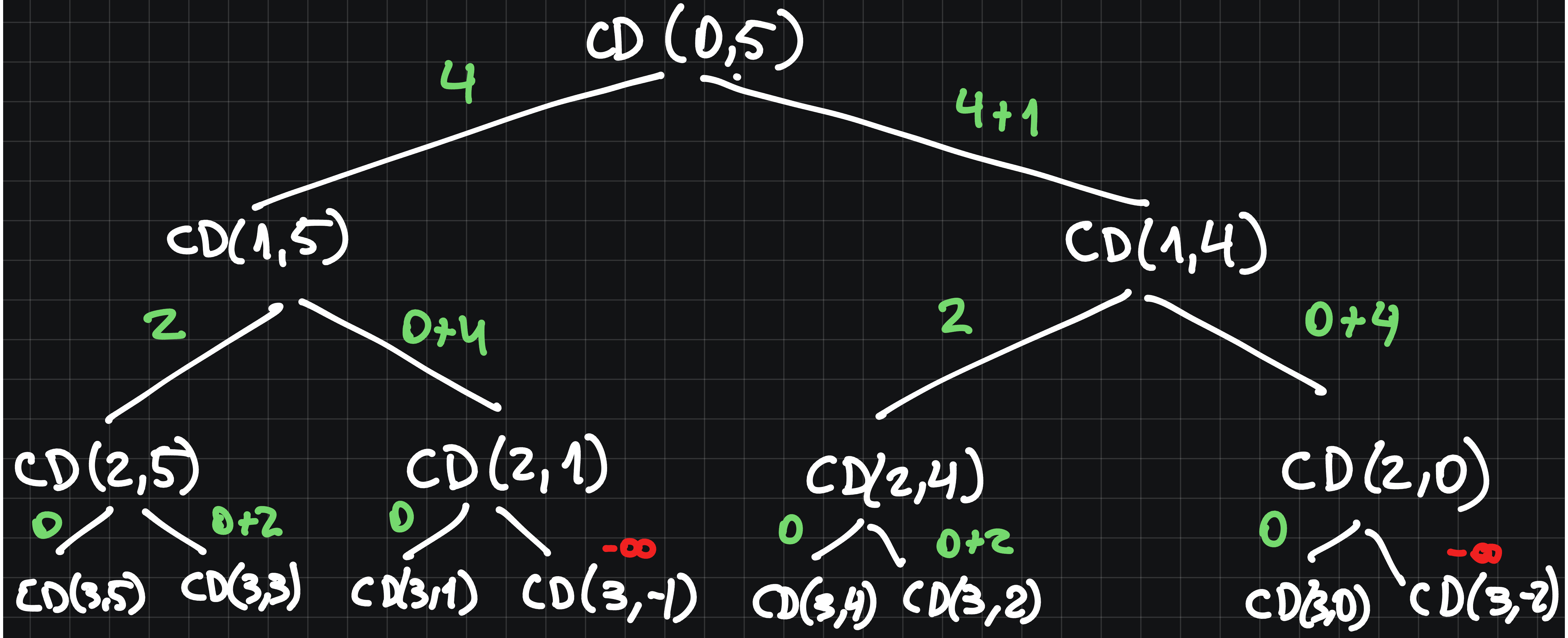
Para resolver el Problema invoco $CD(0,P)$

CD - Pseudo Código

```
CD(i, k) {
    if (i==N) {
        if (k<0) {
            return -INF;
        } else {
            return 0;
        }
    } else {
        return max(CD(i+1,k), CD(i+1,k-p[i])+p[i])
    }
}
```

CD - Seguimiento del ejemplo

$N=3 \quad P=5 \quad P_0=1 \quad P_1=4 \quad P_2=2$



CD - Podas

Hay ramas del árbol que podemos determinar que no vamos a seguir explorando antes de llegar a las hojas.

¿Ejemplos?

CD - Podas

Hay ramas del árbol que podemos determinar que no vamos a seguir explorando antes de llegar a las hojas.

¿Ejemplos?

Poda por factibilidad - Si ya veo que $K<0$ antes que $i=N$, corto.
(evito explorar nodos no factibles)

Poda por optimalidad - Si $\sum_{j>i} p_j \leq K$, agrego todos los temas y corto.
(evito explorar nodos subóptimos)

CD - Formulación con Podas

$$CD(i,k) = \begin{cases} -\infty & \text{si } k < 0 \\ \sum_{i \le j < N} p_j & \text{si } \sum_{i \le j < N} p_j \le k \\ \max(CD(i+1,k), CD(i+1,k-p_i)+p_i) & \text{c.c.} \end{cases}$$

(Notar que cuando $i=N$, la suma da 0)

Pseudo código

```
CD(i, k) {
    if (k<0)
        return -INF;
    if (k >= suma_restante(i)) {
        return suma_restante(i);
    } else {
        return max(CD(i+1,k), CD(i+1,k-p[i])+p[i]);
    }
}
```

ojo! esto puede aumentar la complejidad!

en vez de calcularlo cada vez, se puede preprocesar en $O(n)$

CD - Podas

¿Y si quiero podar cuando la suma de todas las duraciones que quedan no superan a la máxima encontrada?

Se puede, pero requiere una ligera reformulación - notar que el máximo se calcula en la raiz y se desconoce hasta el final del cálculo, así que no puedo conocer el "máximo hasta ahora".

CD - Pseudocódigo con evaluación del máximo en las hojas

```
CD(i, k) {
   if (k<0) {
     return -INF;
   }
   if (i==N) {
     return 0;
   } else {
       return max(CD(i+1,k), CD(i+1,k-p[i])+p[i])
   }
}
```

Formulación con máximo en el caso recursivo

El llamado inicial es $CD(0,P)$

```
CD(i, k, max_found) {
   if (k<0) {
       return max_found;
   }
   if (i==N) {
       return max(P-k,max_found);
   } else {
       new_max = CD(i+1,k,max_found);
       return CD(i+1,k-p[i],new_max);
   }
}
```

Usualmente no es un parámetro explícito sino una variable global

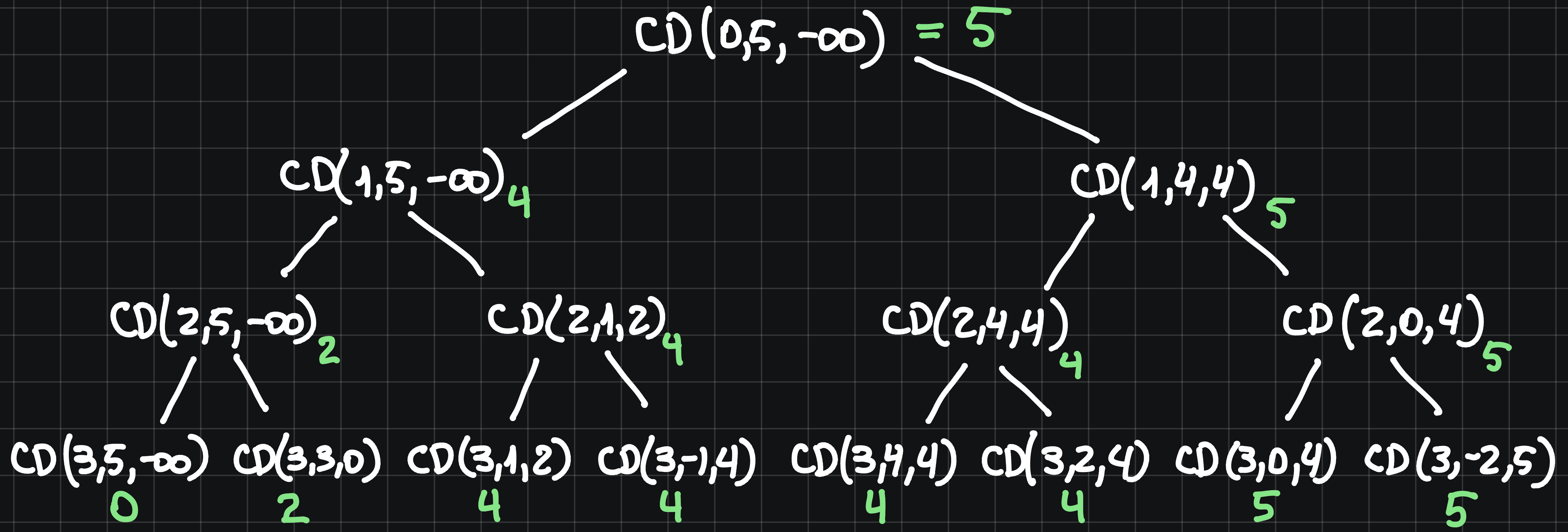
Formulación con máximo en el caso base (conoce el máximo hasta ahora)

El llamado inicial es $CD(0,P,-\infty)$

Este es solamente el cambio de formulación, en ambos casos la única poda es que se corta si el CD se pasó de capacidad

(1)- Seguimiento del ejemplo en la formulación alternativa

$N=3 \quad P=5 \quad P_0=1 \quad P_1=4 \quad P_2=2$



En este esquema, el parámetro max_found contiene el mejor valor encontrado hasta ahora. Funcionalmente hace lo mismo, la ventaja es que puedo agregar la poda

(1)- Poda acotando por el máximo ya encontrado

```
CD(i, k, max_found) {
    if (k<0 || suma_restante(i)<max_found) {
        return max_found;
    }
    if (i==N) {
        return max(P-k,max_found);
    } else {
        new_max = CD(i+1,k,max_found);
        return CD(i+1,k-p[i],new_max);
    }
}
```

Si incluso agregando todos los items que quedan no llego al máximo encontrado, corto

En problemas de optimización, los algoritmos que exploran podando por este tipo de cotas, entre otras estrategias, se conocen como branch & bound

CD - Complejidad.

¿Cuántos nodos tiene el árbol?

Es aproximadamente un árbol binario de altura N, $O(2^N)$

¿Qué costo tienen las operaciones en cada nodo?

Son $O(1)$

CD - Solución con conjunto

¿Cómo hago si además de la duración máxima quiero el conjunto de temas?

Al igual que el máximo actual, voy a necesitar pasármelo por parámetro o mantener una variable global.

Pasarlo por referencia para mantener la complejidad

Cuando encuentro un nuevo máximo, me hago una copia
(esto aumenta la complejidad!)

Problema : Sudoku

(Ejemplo: https://vjudge.net/problem/Gym-102697-040)

Dado um tablero de $N \times N$ (siendo N un cuadrado perfecto) donde algumos lugares están llenos, se quiere saber si es posible completar el tablero con números de 1 a N sin repetir números en la misma fila, misma columna, ni mismo recuadro de $\sqrt{N} \times \sqrt{N}$.

(Este es um problema de <u>decisión</u>)

Sudoku

¿Cuáles son las soluciones candidatas?

Las maneras de llenar el tablero (incluyendo ilegales)

¿Qué deben verificar?

Que cumpla las reglas de Sudoku

¿Cómo puedo construirlas a partir de soluciones parciales?

Llenando de a uno cada casillero libre

Sudoku – Formulación

Si no está en el input es 0.

verifica solución

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

T al poner k en la posición i,j

sig devuelve las coordenadas siguientes yendo de arriba a abajo e izquierda a derecha:

$$\text{sig}(i,j) = \begin{cases} (i,j+1) & \text{si } j < N \\ (i+1,0) & \text{si } j = N \end{cases}$$

¿Llamado para solución? Sudoku$(T,0,0)$

Implementación: No queremos pasar el tablero por copia

Sudoku - Complejidad

¿Cuántos nodos tiene el árbol?

Es un árbol de altura N^2 con N hijos por nodo, $O(N^{(N^2)})$

¿Cuánto cuesta pasar por cada uno?

Verificar las hojas, que son $O(N^{(N^2)})$, cuesta $O(N^2)$

La complejidad queda en $O(N^{(N^2)} N^2)$

Sudoku – Podas

Dada una celda, podemos obtener en $O(N)$ una lista de los números que se pueden poner en esa celda de manera legal, basta con mirar las N posiciones de su fila, de su columna y de su recuadro.

$\Rightarrow$ Puedo evitar probar números inválidos

Sudoku - Podas

Más aún, puedo filtrar más la lista quitando además los números que hacen que otra celda se quede sin candidatos.

(Para cada celda de la misma fila, columna y recuadro, me fijo si alguna tiene un único candidato, y si ese único candidato está en mi lista, no lo puedo usar)

El costo de la poda es $O(N^2)$

Sudoku - Podas

Puedo hacer algo aún más sofisticado:
En vez de recorrer el tablero de arriba a abajo e izquierda a derecha, siempre que hayan celdas libres busco la más restringida, ie., la que tiene menos candidatos posibles.

Defino una función mas_cond que dado un tablero devuelve la celda más restringida, y se indefine si no quedan celdas vacías

Sudoku - Podas

Formulación:

celda más condicionada (se indefine si no quedan)

$$\text{Sudoku}(T) = \begin{cases} \text{esValido}(T) & \text{si } \text{mas\_cond}(T) = \perp \\ \bigvee\limits_{k \in \text{cand}(\text{mas\_cond}(T))} \text{sudoku}(T[\text{mas\_cond}(T)] \leftarrow k) & \text{c.c.} \end{cases}$$

resuelve con Sudoku(T)

valores candidatos

"bottom"; ie. se indefine

(La complejidad asintótica aumenta pero la performance mejora drásticamente)

Problema - Dobra

(Basado en https://vjudge.net/problem/DMOJ-coci09c1p3)

(Nota: Este ejercicio no se llegó a dar en la clase)

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

- No queremos 3 vocales ni 3 consonantes seguidas
- Tiene que haber por lo menos una "L" en la palabra

(Este es un problema de <u>conteo</u>)

¿Cuáles son las soluciones candidatas?

Toda cadena con alguna letra en el lugar de cada comodín

¿Cuáles son las soluciones parciales?

Las cadenas que completan algunos de los comodines
(por ej. todos los que están a la izquierda de una posición i)

Dobra - Formulación

$\rightarrow$ Verificar : Palabra $\rightarrow \{0,1\}$.

$$\text{dobra}(i, I) = \begin{cases} \text{Verificar}(I) & \text{si } i = n \\ \text{dobra}(i+1, I) & \text{si } I[i] \neq \_ \\ \sum_{c} \text{dobra}(i+1, I[i] \leftarrow c) & \text{c.c.} \end{cases}$$

c letra mayusc.

¿Cuál es el llamado para resolver? Dobra $(0, I)$

¿Cuántos nodos tiene el árbol? $O(26^N)$
(Probando todas las letras sin la Ñ)

Dobra - Podas

¿Por qué contar cada letra como distinta?

Podemos separar en los casos consonante/vocal

Aún más, no es necesario llegar al final de la cadena para ver su validez

Hay que prestar atención a si ya agregué una L

Dobra - Formulación distinguiendo consonantes/vocales

formulamos $dobra(i, I, tiene\_L)$ de manera que:

- Si $i = N$, devolvemos 1 si $tiene\_L$, 0 si no.
- Si $I[i] \neq \_$, verificamos que cumple, y de hacerlo vamos a $dobra(i+1, I, tiene\_L \vee I[i] = L)$
- Si $I[i] = \_$ no admite consonante pero si vocal, hacemos $5 * dobra(i+1, I[i] \leftarrow A, tiene\_L)$
- Si $I[i] = \_$ no admite vocal pero si consonante hacemos $20 * dobra(i+1, I[i] \leftarrow B), tiene\_L) +$ $dobra(i+1, I[i] \leftarrow L, true)$
- Si $I[i] = \_$ y admite tanto consonante como vocal, sumamos los dos casos anteriores
- En caso contrario devolvemos 0.

Dobra - Complejidad

Haciendo la separación Consonante - Vocal - L, podemos acotar la cantidad de nodos del àrbol com $O(3^n)$.

Aún mejor, analizando con um poco màs de detalle podemos ver que es $O(n2^n)$

Dobra - $O(n2^n)$

Notar que una vez que tiene_L es verdadero, se puede omitir la separación en casos L/no L y el subárbol queda binario.

En el nivel i hay 2^i nodos correspondientes a cadenas sin L y 2^i raíces de árboles binarios de altura $n-i$.

Entonces hay $O(2^n)$ nodos para cadenas sin L y

$$\sum_{i=1}^{n} 2^i 2^{n-i} = \sum_{i=1}^{n} 2^n = n2^n$$

nodos correspondientes a cadenas que tienen una L.