Ayudantía 13

Comparación de técnicas

Definición

Técnicas para construir algoritmos

Los 4 jinetes del apocalipsis

1. Dividir para conquistar
2. Backtracking
3. Greedy
4. Programación dinámica

La idea es que sean capaces de decidir, **dependiendo de la situación** qué técnica usar para resolver un problema.

Definición

Mini resumen

Dividir para conquistar:
Serie de problemas **disjuntos** para los cuales hay una forma de **solución común**.

Backtracking:
Una serie de **variables** que se deben **asignar**, existencia de **restricciones** y **dominios** claros.

Definición

Mini resumen

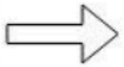
Programación dinámica:
Serie de problemas que se **solapan** y permiten optimizar próximos resultados con información ya sabida.

Greedy:
Serie de decisiones necesarias para tomar y una **heurística** que busca tomar siempre la mejor decisión local, buscando el óptimo global.

Ejercicio 1

Un tablero de kakuro square es una grilla de NxM donde cada posición tiene una celda que puede ser rellenada por un número natural distinto de 0. Además, cada fila y columna tiene un número natural que indica el valor que debe tener la suma de los números correspondientes a la fila o columna. Por ejemplo:

	23	15	12
23			
17			
10			



	23	15	12
23	9	8	6
17	8	4	5
10	6	3	1

Pregunta 1 - I2-2018-2

a) Identifica variables del problemas y sus respectivos dominios.
b) Explica con tus palabras cómo se podría revisar, en O(1), si la asignación de una variable rompe una restricción.
c) Explica con tus palabras una poda o heurística que se puede aplicar para resolver el problema de manera más eficiente.

Pregunta 1 - I2-2018-2



Pregunta 1 - I2-2018-2

a) Para resolver con backtracking este problema se debe asignar a cada celda un número, por lo que las variables son las celdas [1pto].

En teoría cada celda puede tener cualquier número natural, pero ya que no puede haber un dominio infinito para resolverlo con backtracking es necesario acotar los dominios. Un domino acotado correcto puede ser los números del 1 al MAX(restriction). Ya que se sabe que nunca un número puede ser mayor al valor de su restricción [1pto].

b) Se puede mantener un contador por cada restricción del problema que cuente la suma actual de cada fila o columna. De esta manera se inmediatamente si rompo la restricción cuando el contador supera la restricción [2pts].

c) [2pts]
Podas: Ver que los contadores descritos en b) no superen el máximo antes de tener completa la fila o columna. Ver que la suma de una fila o columna actual más el número de casillas vacías de la fila o

Ejercicio 2

Supongamos que quieres hacer una caminata por un parque nacional, que te va a tomar varios días. Tu estrategia es caminar lo más posible de día, pero acampar cuando oscurece. El mapa proporcionado por la oficina de turismo muestra muchos buenos lugares para acampar, y tú has decidido que cada vez que llegues a uno de estos lugares vas a calcular (correctamente) si alcanzas a llegar al próximo antes de que oscurezca; en caso afirmativo, sigues; de lo contrario, te detienes y acampas.

Argumenta rigurosamente que esta estrategia minimiza el número de detenciones para acampar que vas a tener que hacer.

Pregunta 3-Ex-2018-2

Pregunta 3-Ex-2018-2

Por contradicción.

Primero, algunos supuestos y definiciones. Sean L la longitud total del camino, d la distancia que puedes caminar en un día, y $x_1, x_2, \ldots x_n$ los puntos de detención que muestra el mapa (distancias desde la entrada del camino). Decimos que un conjunto de puntos de detención es *válido* si
- la distancia entre cualquier par de puntos adyacentes (en el conjunto) es a lo más d,
- la distancia desde la entrada al primer punto es a lo más d, y
- la distancia desde el último punto a la salida del camino es a lo más d.

Sean $R = \{x_{p1}, x_{p2}, \ldots, x_{pk}\}$ el conjunto (válido) de puntos de detención elegidos por tu estrategia codiciosa, y $S = \{x_{q1}, x_{q2}, \ldots, x_{qm}\}$, con $m < k$, un conjunto válido de menos puntos que R.

Primero, los puntos de R están más lejos que los puntos de S; es decir, para cada $j = 1, 2, \ldots, m$, tenemos que $x_{pj} \geq x_{qj}$, lo que demostramos por inducción sobre j.

> Para $j = 1$, sale de la definición de la estrategia codiciosa: tú viajas lo más posible el primer día antes de detenerte.
>
> Para $j > 1$ y suponiendo que la afirmación es válida para todo $i < j$: $x_{qj} - x_{qj-1} \leq d$, y también $x_{qj} - x_{qj-1} \geq x_{qj} - x_{pj-1}$, lo que implica que $x_{qj} - x_{pj-1} \leq d$. Es decir, una vez que tú sales de x_{pj-1}, podrías caminar hasta x_{qj} en un día, por lo que x_{pj}, que es donde finalmente vuelves a detenerte, solo puede estar más lejos que x_{qj}.

Segundo, como $m < k$, entonces $x_{pm} < L - d$; de lo contrario no tendría sentido que te detengas en x_{pm+1}. Como además $x_{qm} \leq x_{pm}$, por la demostración anterior, resulta que $x_{qm} < L - d$, lo que contradice que S sea un conjunto válido de puntos de detención.

Ejercicio 3

Suponga que en una sala se han programado n charlas $c_1, \ldots, c_n$, cada una de las cuales dura exactamente una hora, y tal que cada charla c_{i+1} comienza apenas termina la anterior charla c_i. La asistencia a una charla c_i $(1 \leq i \leq n)$ entrega un puntaje p_i (representado por un entero positivo) pero produce tanto cansancio que es imposible asistir a las siguientes d_i charlas, teniendo que descansar. Alice quiere asistir a las charlas y sumar la máxima cantidad de puntaje posible, respetando las reglas de descanso indicadas. En este ejercicio usted tendrá que ayudarla a calcular el puntaje máximo.

(a) (3 puntos) Dados los valores $p_1, \ldots, p_n$ y $d_1, \ldots, d_n$ para las n charlas, escriba una ecuación de recurrencia para la función $S(i)$, la que calcula (de forma exhaustiva) el puntaje máximo que se puede obtener con las charlas $c_i, \ldots, c_n$, para $1 \leq i \leq n$. Además de la definición recurrente, indique claramente los casos base necesarios para la correcta definición de la función. Si lo prefiere, puede usar pseudocódigo para representar la ecuación de recurrencia.

Pregunta 4 - I2-2024-01

Pregunta 4 - I2-2024-01

$$S(i) = \begin{cases} 0, & i > n \\ \text{máx}\,\{S(i+1), S(i+d_i+1)+p_i\}, & i \leq n. \end{cases}$$

Claramente, la solución al problema original se obtiene con $S(1)$. La versión pseudocódigo es:

Algoritmo 1: $S(i)$

```
if i > n then
    return 0
else
    return máx {S(i+1), S(i+d_i+1)+p_i}
```

(b) (3 puntos) Implemente el algoritmo del punto anterior usando programación dinámica, tal que evite recalcular subproblemas antes calculados. Su solución puede ser iterativa o recursiva, según su preferencia.

Pregunta 4 - I2-2024-01

Pregunta 4 - I2-2024-01

Solución: En este caso se mantiene un arreglo $M[1..n]$ con todas sus entradas inicializadas con $\emptyset$. La versión de programación dinámica recursiva es:

Algoritmo 2: $SPD(i)$

```
if i > n then
    return 0
else
    if M[i] = ∅ then
        M[i] ← máx {SPD(i + 1), SPD(i + d_i + 1) + p_i}
    return M[i]
```

También es válida la versión iterativa.

Feedback