Programació i Algorítmica Avançades

Guia de Laboratori

Curs 2023-24, quadrimestre de primavera

José Luis Balcázar, Jordi Delgado

Departament de Ciències de la Computació, UPC

12/03/2024:

Máquinas de Turing; Introducción a la búsqueda combinatoria

Hoy:

- Programación de máquinas de Turing
- Búsqueda combinatoria exhaustiva
- Problemas para tu trabajo personal posterior

Programación de máquinas de Turing de una cinta

Se aconseja ponerse de acuerdo con los vecinos de mesa para que unos trabajen sobre la página web `http://morphett.info/turing/turing.html` y otros sobre la página alternativa `https://turingmachine.io` de manera que podáis comparar las soluciones. Empezad por revisar dos o tres de los ejemplos que trae la web elegida, entendiendo la sintaxis que se usa en cada una para describir los estados y las transiciones.

En cada uno de los ejercicios que siguen, pensad primero cómo creéis que ha de ser la solución (se autoriza e incluso se anima a trabajar conjuntamente en equipo con los vecinos de mesa) y comprobadla después por separado, cada uno en la web que le corresponda.

1. Construye una máquina de Turing que no recibe entrada y escribe en la cinta el saludo “Hello”.
2. Construye una máquina de Turing que recibe dos palabras separadas por un espacio en blanco y comprueba si son iguales. Empezad pensándolo con un alfabeto de sólo dos letras y ampliadlo después.

3. Construye una máquina de Turing que recibe una palabra y la modifica de manera que quede un espacio en blanco entre cada dos caracteres. Empezad con un alfabeto reducido.

Programación de máquinas de Turing de varias cintas

La web `https://turingmachinesimulator.com` contiene ejemplos de programación de máquinas de Turing de varias cintas. En seguida veréis que eso hace la vida del programador de máquinas de Turing más fácil (pero complica la del programador que ha de fabricar su máquina universal).

1. Entiende algunos de los ejemplos que aparecen preprogramados.
2. Repite en este simulador alguno de los ejercicios hechos en el modelo de una sola cinta.

Búsqueda combinatoria exhaustiva

1. Construye en Python tu propia versión de una función que retorne el *powerset* de un conjunto dado. Pruébala sobre `https://jutge.org/problems/P18957_ca` (de la lista Recursion Revisited) hasta lograr luz verde.
2. Usando esa función, resuelve `https://jutge.org/problems/X94664_en` (de la lista Combinatorial Search Schemes I) hasta lograr luz verde. (En la semana siguiente, empezamos por aquí.)

Problemas para tu trabajo personal posterior

Máquinas de Turing de una cinta: usa los simuladores indicados anteriormente y

1. Construye una máquina de Turing que recibe una palabra y comprueba si se puede partir por la mitad en dos palabras iguales.
2. Construye una máquina de Turing que recibe en su cinta un número natural en binario y termina con el triple de ese número natural escrito en su cinta.

Máquinas de Turing de varias cintas: usa el simulador indicado anteriormente y

1. Repite en este simulador los ejercicios que hayas hecho en clase sobre los otros simuladores.
2. Construye una máquina de Turing para la función *diferencia modificada* entre números naturales, la cual, dados naturales `x` e `y`, calcula `max(x-y, 0)`.

Búsqueda exhaustiva: si en clase no te dio tiempo a completar hasta luz verde los ejercicios de Jutge indicados, hazlo ahora.

- `https://jutge.org/problems/P18957_ca`,
- `https://jutge.org/problems/X94664_en`.

Generadores (opcional): aprende a programar generadores en Python (salvo que ya sepas) y emplea un generador para recorrer el *powerset* en alguno de los ejercicios anteriores. Si tu generador se basa en un recorrido iterativo, busca también un generador recursivo.

19/03/2024:

Búsqueda combinatoria: *set-based backtracking.*

Hoy:

- Búsqueda combinatoria exhaustiva: el ejemplo de la mochila
- *Backtracking:* el ejemplo de la mochila
- Variantes enfocadas a encontrar solamente la primera solución
- La variante de optimización
- El ejemplo de la suma del subconjunto
- Problemas para tu trabajo personal posterior
- Material adicional

Búsqueda combinatoria exhaustiva: el ejemplo de la mochila

Baja a través de la página de PAA (enlace en el Racó) la carpeta cuyo nombre corresponde a la fecha de hoy y descomprímela. En el fichero `inputs_knapsack.txt` encontrarás algunos ejemplos de entradas para el problema decisional de la mochila. Los ficheros `knapsack_dec_all_exh_pws_lab.py` y `knapsack_dec_all_exh_tr_lab.py` contienen programas que resuelven el problema de obtener todas las soluciones mediante búsqueda exhaustiva, tal como los hemos visto en la clase de teoría, siguiendo el enunciado de la transparencia 44 (`Mochila, II`) y siguientes.

1. Entiende el contenido del programa `knapsack_dec_all_exh_pws_lab.py` y pruébalo con algunas llamadas para entender bien su funcionamiento.
2. Entiende el contenido del programa `knapsack_dec_all_exh_tr_lab.py` y pruébalo con algunas llamadas para entender bien su funcionamiento.
3. Completa ambos con un programa principal que realice la lectura de datos y la escritura de resultados de manera que obtengas luz verde en X94664 (todas las referencias al Jutge hoy están en la lista `Combinatorial Search I`).

Backtracking: el ejemplo de la mochila

Modifica el contenido del programa `knapsack_dec_all_exh_tr_lab.py` (en la versión que acabas de completar para obtener luz verde, por supuesto) siguiendo las indicaciones que vimos en clase para evitar exploraciones innecesarias (transparencia 56, `Mochila, VII`, y las anteriores y siguientes). Comprueba que lo haces correctamente mediante una nueva luz verde en el mismo problema X94664.

Variantes enfocadas a encontrar solamente la primera solución

1. Modifica tu extensión al programa `knapsack_dec_all_exh_pws_lab.py` para que retorne únicamente una solución, la primera que encuentre.
2. Modifica tu extensión al programa `knapsack_dec_all_exh_tr_lab.py` para que retorne únicamente una solución, la primera que encuentre.
3. Modifica tu programa de *backtracking* del apartado anterior para que retorne únicamente una solución, la primera que encuentre.

La variante de optimización

No hay particular dificultad en usar los casos dados en el fichero `inputs_knapsack.txt` para la variante de optimización: basta omitir el primero de los valores, que corresponde al valor mínimo deseado, dando solamente el peso máximo, el número de objetos, y los pares de valor y peso por objeto. Estos ejemplos están tomados de la web `https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01`, la cual contiene unos cuantos más, adicionalmente, por si los precisas (pero están en un formato un poquito diferente).

1. Completa el programa `knapsack_dec_opt_exh_pws_lab.py`, que optimiza por búsqueda exhaustiva (transparencia 65, `Mochila, XIV`, salvo que han cambiado levemente algunos identificadores), obteniendo luz verde en X59240.
2. Resuelve el mismo poblema por *backtracking* (transparencia 66, `Mochila, XV`).

El ejemplo de la suma del subconjunto

Subset sum es un problema muy similar; de hecho, ocasionalmente, se encuentran referencias a *Subset sum* bajo el nombre *Knapsack*. Al no tener que diferenciar los pesos de los valores, es algo más simple. Resuelve el problema P40685 hasta obtener luz verde. *Ten mucha precaución*: el enunciado permite valores negativos, así que tendrás que inventar algo para poder podar alguna parte de la exploración.

Problemas para tu trabajo personal posterior

1. Completa las variantes del problema de la mochila que transmiten como parámetro la solución parcial candidata en curso, tanto en la versión que hace copias en

las llamadas como en la que mantiene una única solución parcial candidata que se va modificando con `append()` y `pop()`, a partir de los programas que se han explicado en la clase de teoría (transparencias 59 y 60).

Material adicional

```
def knapsack(weights, values, current_item, max_w):
    if current_item == -1:
        return ([],0,0)
    else:
        "current_item >= 0"
        best0, bestw0, bestv0 = knapsack(weights, values, current_item - 1, max_w)
        if weights[current_item] <= max_w:
            best1, bestw1, bestv1 = knapsack(
                weights, values, current_item - 1, max_w - weights[current_item])
            if bestv1 + values[current_item] > bestv0:
                best1.append(current_item)
                return (best1, bestw1 + weights[current_item],
                               bestv1 + values[current_item])
    return best0, bestw0, bestv0
```

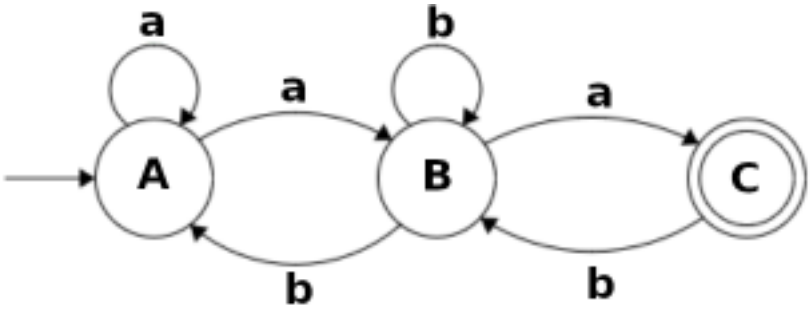
02/04/2024:

Preparación del examen parcial

Entra en examen:

- lenguajes regulares:
  - autómatas finitos,
  - expresiones regulares,
  - gramáticas regulares...
- lenguajes incontextuales:
  - gramáticas incontextuales,
  - autómatas a pila...
- búsqueda combinatoria: *backtracking*,
- búsqueda combinatoria: *greedy*.

Problema del examen parcial del curso pasado Aplica el lema de Arden para encontrar una expresión regular que describa el lenguaje formal aceptado por el autómata finito siguiente:



Algorítmica

1. Si aún no lo tienes resuelto, resuelve hasta obtener luz verde el problema X33098 (problema del examen parcial del curso pasado).
2. Si ya tienes ese problema resuelto, pero no has intentado aún P81009, inténtalo ahora hasta lograr: (a) AC en los juegos de prueba públicos y (b) o bien AC o bien "time limit exceeded" (más probablemente será lo segundo) en los privados. En general, para obtener luz verde, probablemente sea necesario aplicar un esquema algorítmico que estudiaremos después del examen parcial.
3. Si ya estás en ese punto con ambos problemas, resuelve hasta obtener luz verde el problema P81629.
4. El problema P92008 es muy similar; sin embargo, es más estricto en términos de eficiencia y existen soluciones que pasan P81629 y, sin embargo, en éste dan "time limit exceeded". Prueba a basarte en tu solución a P81629 y, de ser necesario, encuentra cómo acelerarlo hasta obtener luz verde en P92008.
5. Si hay enunciados indicados en los días anteriores que aún no has resuelto, resuélvelos.
6. Completa, si aún no lo has hecho, las listas Combinatorial Search I y Combinatorial Search II, excepto X52116.

Material adicional

El código indicado a continuación obtiene luz verde en X33098. Coincide con el desarrollado en clase el martes 2 de abril en el grupo de las 12h, y es muy similar al indicado en la clase de las 8h del mismo día, pero algo más simple y claro de entender.

```
def opt_b(clayleft, prices, currsize):
    if currsize == 0 or clayleft == 0:
        return list(), 0.0
    else:
        "try not making bowl of currsize, then making it"
        s0, b0 = opt_b(clayleft, prices, currsize - 1)
        if currsize <= clayleft:
            s1, b1 = opt_b(clayleft - currsize, prices, currsize)
            if b1 + prices[currsize] > b0:
                s1.append(currsize)
                s0, b0 = s1, b1 + prices[currsize]
        return s0, b0

from pytokr import pytokr
item = pytokr()

clay = int(item())
prices = [0.0] # placeholder for length 0 so that lengths and indexes coincide
for _ in range(clay):
    prices.append(float(item()))

s, b = opt_b(clay, prices, clay)
print(f"{b:.2f}") # o bien: print(f"{b:.2f}", s)
```

16/04/2024:

Programación Dinámica, I

"The gold ones are Galleons", he [Hagrid] explained. "Seventeen silver Sickles to a Galleon and twenty-nine Knuts to a Sickle, it's easy enough."

J. K. Rowling: *Harry Potter and the Philosopher's Stone*

- *Giving change*
- Aún otra vez "La Mochila"
- Problemas adicionales para clase o para tu trabajo personal posterior

"Giving change"

a/ Repasa y entiende los programas vistos en clase. Si mantenemos (por ejemplo, en un `dict`) la tabla bidimensional, facilitará la comprensión el recorrer las denominaciones de monedas por sus índices:

```
def minchange(goal, denoms):
  denoms = [0] + list(denoms) # placeholder
  dptable = {}
  for quantity in range(goal + 1):
    "init table for no coins"
    dptable[0, quantity] = float("inf")
  for coin in range(len(denoms)):
    dptable[coin, 0] = 0
  for quantity in range(1, goal + 1):
    '''
    dptable[coin, quantity]:
    how many coins up to that one needed to add up to quantity
    '''
    for coin in range(1, len(denoms)):
      "try using it"
      if (denoms[coin] <= quantity and
        1 + dptable[coin, quantity - denoms[coin]]
          < dptable[coin - 1, quantity]):
        dptable[coin, quantity] = 1 + dptable[coin, quantity - denoms[coin]]
      else:
        dptable[coin, quantity] = dptable[coin - 1, quantity]
  return ...
```

b/ Complétalo siguiendo las indicaciones como para poderlo probar en P81009, pero no lo intentes aún: de momento, asegúrate de cómo funciona en tu propia máquina.

c/ Cuando trabajemos con la tabla unidimensional, no es preciso usar los índices de las denominaciones y podemos iterar directamente sobre ellas:

```
def minchange(goal, denoms):
    dptable = [ float("inf") ] * (goal + 1)
    dptable[0] = 0
    for quant in range(1, goal+1):
        '''
        dptable[quant]: how many coins to add up to it
        '''
        for coin in denoms:
            "try using it"
            if coin <= quant:
                dptable[quant] = min(dptable[quant], 1 + dptable[quant - coin])

    return ...
```

Complétalo y compara con la versión anterior.

d/ Cuando creas que los entiendes suficientemente, modifícalos para poder obtener luz verde en P81009; sin embargo, has de tener en cuenta que, en ese problema, **no** tienes garantía de que esté la denominación 1 y, por tanto, es posible que, en algunos casos, no haya solución. Es muy poco probable que logres luz verde con la variante que mantiene la tabla bidimensional, que probablemente dará "time limit exceeded" en los juegos de prueba privados. Con la alternativa de tabla unidimensional es posible que lo logres.

e/ Añade (ahora ya al margen del Jutge) una función que permita reconstruir la solución que usa el mínimo de monedas, a base de comparar los contenidos de las celdas, tal como hemos visto en clase:

```
from collections import Counter

def trace(gctab, denoms, q):
    "returns solution in dict r, uses non-generalizable method"
    r = Counter()
    d = ... # use it to traverse denominations from large to small
    while q:
        "hack - we have just two cases and, in one, t[d,q] == t[d-1,q]"
        if ...:
            "low end, only smallest coins remain to be used"
            r[denoms[d]] += q
            break
        elif gctab[d, q] == gctab[d-1, q]:
            "coins of denoms[d] units were not employed"
            d -= 1
        else:
            "at least one coin of denoms[d] units was employed"
            r[denoms[d]] += 1
            q -= denoms[d]
    return r
```

Completa el programa y asegúrate de que funciona adecuadamente.

f/ La manera *generalizable* de poder reconstruir la solución en una aplicación de programación dinámica es mediante el mantenimiento de una segunda tabla. En este caso, queremos poder reconstruir las soluciones así:

```
def trace(best, goal):
    coins = []
    while goal:
        use = best[goal]
        coins.append(use)
        goal -= use
    return coins
```

Has de añadir al programa principal la gestión de la tabla secundaria (que aquí llamamos `best`): cada vez que cambia la tabla principal, en la segunda tabla anotamos el por qué. La solución óptima viene, entonces, en forma de lista con repeticiones en lugar de un Counter. Completa el programa, asegúrate de que funciona adecuadamente, por ejemplo, comparando con el anterior y, finalmente, intenta reemplazar la lista por un Counter.

Aún otra vez "La Mochila"

Inspirándote en las propuestas vistas para *Giving Change*, resuelve la variante 0/1 de Knapsack por Programación Dinámica: cada objeto puede elegirse para ponerlo en la mochila o bien dejarse fuera. Puedes usar X59240 (que ya conoces) para probar tus soluciones.

Problemas adicionales para clase o para tu trabajo personal posterior

- *Echando una mano a la alfarera*, versión DP, X52116.
- *Cortes de la varilla* ("Rod cutting"), X71353, que posiblemente ya has resuelto por *backtracking* pero ahora toca hacerlo por programación dinámica.
- *Canonical coin systems (2)*, X88410 (la versión X24976 es la del examen y supone que se buscan soluciones óptimas por *backtracking*; en cambio, X88410 requerirá hacerlo por Programación Dinámica y muy probablemente de error por *time limit exceeded* si se sube una solución basada en *backtracking*.

23/04/2024:

Programación Dinámica, II

- Supersequences
- Aún otra vez caminos mínimos
- Ejemplos adicionales

Supersequences

Completa el ejemplo de encontrar la supersecuencia más corta de dos *string* dados. Para ello, has de reflexionar con sumo cuidado para identificar qué parte de la tabla bidimensional hay que inicializar antes de lanzar el doble bucle que la llena y, sobre todo, entender cuál es la inicialización adecuada. Puedes probar tus soluciones en X40399 de la lista Combinatorial Search III.

Por supuesto, has de partir de la ecuación de Bellman y el esbozo de algoritmo vistos en clase y documentados en las transparencias ("Supersecuencias, III" y "IV").

All-pairs shortest paths

Dado un grafo (implementado de alguna de las maneras que tengas de Programació i Algoritmes 2), construye y programa el algoritmo de Floyd a partir de la correspondiente ecuación de Bellman vista en clase (Transparencia "Caminos mínimos, IV").

Ejemplos adicionales

Ya listábamos la semana pasada estos ejemplos a resolver; atácalos si aún no lo has hecho:

- *Echando una mano a la alfarera*, versión DP, X52116 en la lista Combinatorial Search III.
- *Cortes de la varilla* ("Rod cutting"), X71353 en la lista Combinatorial Search I. Posiblemente ya lo has resuelto por *backtracking* (y si no lo has hecho, hazlo pronto) pero ahora toca resolverlo por programación dinámica.

Para evitar *spoilers*, no se incluyen aquí aún soluciones, pero aparecerán en un futuro próximo.