Backtracking y PD Básica

Julián Braier

FCEN - UBA

1c 2021

Backtracking: refrescando la memoria

Backtracking: refrescando la memoria

- Técnica para recorrer el espacio de soluciones válidas a un problema.

Backtracking: refrescando la memoria

- Técnica para recorrer el espacio de soluciones válidas a un problema.
- Representamos a las soluciones con un vector $\mathbf{a} = (a_1, ..., a_n)$ para representar a las soluciones (habitualmente). Queremos una manera de extender soluciones parciales que nos permita explorar todo el espacio de soluciones.

Backtracking: refrescando la memoria

- Técnica para recorrer el espacio de soluciones válidas a un problema.
- Representamos a las soluciones con un vector $\mathbf{a} = (a_1, ..., a_n)$ para representar a las soluciones (habitualmente). Queremos una manera de extender soluciones parciales que nos permita explorar todo el espacio de soluciones.
- La función de backtracking con $\mathbf{a} = (a_1, ..., a_i)$ como parámetro debería revisar todas las soluciones que tengan a $\mathbf{a}$ como prefijo.

Backtracking: refrescando la memoria

- Técnica para recorrer el espacio de soluciones válidas a un problema.
- Representamos a las soluciones con un vector $\mathbf{a} = (a_1, \ldots, a_n)$ para representar a las soluciones (habitualmente). Queremos una manera de extender soluciones parciales que nos permita explorar todo el espacio de soluciones.
- La función de backtracking con $\mathbf{a} = (a_1, \ldots, a_i)$ como parámetro debería revisar todas las soluciones que tengan a $\mathbf{a}$ como prefijo.
- Ofrece la oportunidad de realizar podas, por factibilidad u optimalidad.

Backtracking: refrescando la memoria

Backtracking: Esquema General - Todas las soluciones

```
BT(a,k)
entrada: a = (a_1,...,a_k) solución parcial
salida: se procesan todas las soluciones válidas

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

Problema del CD

- Te estás por ir con tres amigos a Entre Ríos en auto por un fin de semana. Tenés un CD sobre el cual querés grabar música *súperdivertida* para el viaje.[1]

[1] https://www.youtube.com/watch?v=eDNMBBdfRHY&t=479s&ab_channel=BenitoCanedo

Problema del CD

- Te estás por ir con tres amigos a Entre Ríos en auto por un fin de semana. Tenés un CD sobre el cual querés grabar música *súperdivertida* para el viaje.[1]
- **Problema:** dada K la capacidad del CD (en segundos) y una lista $d = (d_1, ..., d_n)$ con las duraciones de n canciones (en segundos). Cuál es la máxima suma de las duraciones de las canciones que agregamos que podemos lograr (obviamente que sin exceder la capacidad del CD). No vale ni cortar ni repetir canciones.

[1] https://www.youtube.com/watch?v=eDNMBBdfRHY&t=479s&ab_channel=BenitoCanedo

Problema del CD

- Te estás por ir con tres amigos a Entre Ríos en auto por un fin de semana. Tenés un CD sobre el cual querés grabar música *súperdivertida* para el viaje.[1]
- **Problema:** dada K la capacidad del CD (en segundos) y una lista $d = (d_1, \ldots, d_n)$ con las duraciones de n canciones (en segundos). Cuál es la máxima suma de las duraciones de las canciones que agregamos que podemos lograr (obviamente que sin exceder la capacidad del CD). No vale ni cortar ni repetir canciones.
- **Ejemplo:** si $k = 8$ y $d = [2, 4, 5]$ la respuesta es

[1] https://www.youtube.com/watch?v=eDNMBBdfRHY&t=479s&ab_channel=BenitoCanedo

Problema del CD

- Te estás por ir con tres amigos a Entre Ríos en auto por un fin de semana. Tenés un CD sobre el cual querés grabar música *súperdivertida* para el viaje.[1]
- **Problema:** dada K la capacidad del CD (en segundos) y una lista $d = (d_1, ..., d_n)$ con las duraciones de n canciones (en segundos). Cuál es la máxima suma de las duraciones de las canciones que agregamos que podemos lograr (obviamente que sin exceder la capacidad del CD). No vale ni cortar ni repetir canciones.
- **Ejemplo:** si $k = 8$ y $d = [2, 4, 5]$ la respuesta es 7.

[1] https://www.youtube.com/watch?v=eDNMBBdfRHY&t=479s&ab_channel=BenitoCanedo

Problema del CD: preguntas

- Cuál es el espacio de soluciones candidatas?

Problema del CD: preguntas

- Cuál es el espacio de soluciones candidatas?
- Cómo las represento?

Problema del CD: preguntas

- Cuál es el espacio de soluciones candidatas?
- Cómo las represento?
- Cuál es el espacio de soluciones válidas?

Problema del CD: preguntas

- Cuál es el espacio de soluciones candidatas?
- Cómo las represento?
- Cuál es el espacio de soluciones válidas?
- Qué es una solución parcial? Cómo la extiendo?

Problema del CD: solución

Problema del CD: solución

```
int valor_solucion = 0;
vector<int> duracion;
int n;
int k;

void CD(int suma, int i){
    if(i == n) {
        if(suma <= k and suma > valor_solucion) valor_solucion = suma;
    }
    else {
        CD(suma+duracion[i],i+1);
        CD(suma,i+1);
    }
}
//CD(k,0,0) resuelve el problema
```

Problema del CD: solución

Ahora queremos guardar la codificación del subconjunto óptimo además de su duración.

Problema del CD: solución

Ahora queremos guardar la codificación del subconjunto óptimo además de su duración.

```
//y si quiero ademas el subconjunto?
vi sol_parcial;
vi sol_optima;

void CD(int suma, int i){
    if(i == n) {
        if(suma <= k and suma > valor_solucion) {
            valor_solucion = suma;
            sol_optima = sol_parcial;
        }
    }
    else {
        sol_parcial[i] = 1;
        CD(suma+duracion[i],i+1);
        sol_parcial[i] = 0;
        CD(suma,i+1);
    }
}
```

Problema del CD: solución

- Podemos ponerle podas?

Problema del CD: solución

- Podemos ponerle podas?
- Poda por factibilidad: no visitar un nodo que excede la capacidad del CD.

Problema del CD: solución

- Podemos ponerle podas?
- Poda por factibilidad: no visitar un nodo que excede la capacidad del CD.

```
//con poda por factibilidad
void CD(int suma, int i){
    if(suma > k) return;
    if(i == n) {
        if(suma > valor_solucion) {
            valor_solucion = suma;
            sol_optima = sol_parcial;
        }
    }
    else {
        sol_parcial[i] = 1;
        CD(suma+duracion[i],i+1);
        sol_parcial[i] = 0;
        CD(suma,i+1);
    }
}
```

Problema del CD: solución

- Podemos ponerle podas?

Problema del CD: solución

- Podemos ponerle podas?
- Poda por optimalidad: si agregando todas las canciones que quedan no puedo superar el óptimo, retornamos.

Problema del CD: solución

- Podemos ponerle podas?
- Poda por optimalidad: si agregando todas las canciones que quedan no puedo superar el óptimo, retornamos.

```
//con poda por optimalidad
void CD(int suma, int suma_sufijo, int i){
    if(suma > k) return;
    if(suma + suma_sufijo <= valor_solucion) return;
    if(i == n) {
        if(suma > valor_solucion) {
            valor_solucion = suma;
            sol_optima = sol_parcial;
        }
    }
    else {
        suma_sufijo -= duracion[i];
        sol_parcial[i] = 1;
        CD(suma+duracion[i],suma_sufijo,i+1);
        sol_parcial[i] = 0;
        CD(suma,suma_sufijo,i+1);
    }
}
```

Sudoku

SUDOKU

8		6			3		9	
	4			1			6	8
2			8	7				5
1		8			5		2	
	3		1				5	
7		5		3		9		
	2	1			7		4	
6				2		8		
	8	7	6		4			3

ANSWER:

8	7	6	5	4	3	1	9	2
5	4	3	2	1	9	7	6	8
2	1	9	8	7	6	4	3	5
1	9	8	7	6	5	3	2	4
4	3	2	1	9	8	6	5	7
7	6	5	4	3	2	9	8	1
3	2	1	9	8	7	5	4	6
6	5	4	3	2	1	8	7	9
9	8	7	6	5	4	2	1	3

- Dar una función de backtracking que resuelva un sudoku.

Sudoku: primera solución

Sudoku: primera solución

- *sudoku_solver*(i, j): están fijos los números para todas las casillas de las primeras i-1 filas y para las primeras j-1 casillas de la i-ésima fila.

Sudoku: primera solución

```
void sudoku_solver(int i = 0, int j = 0){
    nodos_visitados++;
    if(!solucion.empty()) return;

    if(i == n) {
        solucion = sudoku;
        return;
    }

    int ip = i;
    int jp = j+1;
    if(jp == n) ip++, jp = 0;

    if(sudoku[i][j] == 0){
        vector<int> s = posibles(i,j);
        for(auto x : s) {
            sudoku[i][j] = x;
            sudoku_solver(ip,jp);
            if(!solucion.empty()) return;
        }
        sudoku[i][j] = 0;
    }
    else sudoku_solver(ip,jp);
}
```

Sudoku: Podas

Sudoku: Podas

			5					
				7		1		3
			1		3		7	4
		2		6				9
5					4			
	7	4					1	
	5		8			6		
					9		2	5
	4	1	3	5	6	8	9	7

Sudoku: Podas

			5					
				7		1		3
			1		3		7	4
		2		6				9
5					4			
	7	4					1	
	5		8			6		
					9		2	5
	4	1	3	5	6	8	9	7

- **Idea:** además de chequear que el número no aparece en la fila/columna/sector ver que no estoy dejando otra celda vacía sin ningún candidato.

Sudoku: Resultado de la poda

dificultad	sin poda	con poda
fácil	215	103
medio	6750	1045
difícil	11598	5184
dificilísimo	72097	18567

Table: Nodos visitados en el árbol de Backtracking según dificultad con y sin poda

- Subo un zip con el código y los ejemplos para quien quiera ver.

Coeficientes Binomiales

$$\text{C(n,k)} = \begin{cases} 1 & \text{si } k = 0 \\ 1 & \text{si } k = n \\ C(n-1, k-1) + C(n-1, k) & \text{cc} \end{cases}$$

[2] https://www.youtube.com/watch?v=F54gS38HFrA&ab_channel=BenitoCanedo

Coeficientes Binomiales

$$C(n,k) = \begin{cases} 1 & \text{si } k = 0 \\ 1 & \text{si } k = n \\ C(n-1, k-1) + C(n-1, k) & \text{cc} \end{cases}$$

```
int C(int n, int k){
    if(k == n or k == 0) return 1;
    return C(n-1,k-1) + C(n-1,k);
}
```

[2] https://www.youtube.com/watch?v=F54gS38HFrA&ab_channel=BenitoCanedo

Coeficientes Binomiales

$$C(n,k) = \begin{cases} 1 & \text{si } k = 0 \\ 1 & \text{si } k = n \\ C(n-1, k-1) + C(n-1, k) & \text{cc} \end{cases}$$

```
int C(int n, int k){
    if(k == n or k == 0) return 1;
    return C(n-1,k-1) + C(n-1,k);
}
```

- En esta solución se repite el cómputo de subproblemas cada vez que se vuelve a necesitar su resultado.

[2]

[2] https://www.youtube.com/watch?v=F54gS38HFrA&ab_channel=BenitoCanedo

Coeficientes Binomiales

Coeficientes Binomiales

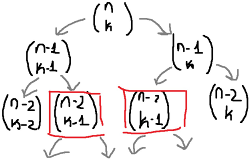
- Y si vamos guardando los resultados?

Coeficientes Binomiales

- Y si vamos guardando los resultados?
- Usamos una matriz $DP[0..n][0..k]$ inicializada en $\perp$.

```
int C(int n, int k){
    if(k == n or k == 0) return 1;
    if(DP[n][k] == BOTTOM)
        DP[n][k] = C(n-1,k-1) + C(n-1,k);
    return DP[n][k];
}
```

Coeficientes Binomiales



```
int C(int n, int k){
    if(k == n or k == 0) return 1;
    return C(n-1,k-1) + C(n-1,k);
}
```

$O\left(\binom{n}{k}\right)$

```
int C(int n, int k){
    if(k == n or k == 0) return 1;
    if(DP[n][k] == BOTTOM)
        DP[n][k] = C(n-1,k-1) + C(n-1,k);
    return DP[n][k];
}
```

$O(n \cdot k)$

Coeficientes Binomiales

Figure: Mentira: no siempre que haya una formulación recursiva vale la pena aplicar **Programación Dinámica**. Tenemos que ver que las invocaciones recursivas sean muchas más que los subproblemas diferentes, y por lo tanto se repiten cómputos (**Propiedad de superposición de subproblemas**).

Programación Dinámica

No es tanto lo que tiene de nuevo esta técnica. Hoy para resolver los ejercicios seguimos sólo tres pasos:

Programación Dinámica

No es tanto lo que tiene de nuevo esta técnica. Hoy para resolver los ejercicios seguimos sólo tres pasos:

1. Dar una función recursiva que resuelva el problema y argumentar su correctitud.

Programación Dinámica

No es tanto lo que tiene de nuevo esta técnica. Hoy para resolver los ejercicios seguimos sólo tres pasos:

1. Dar una función recursiva que resuelva el problema y argumentar su correctitud.
2. Ver que se cumpla la propiedad de superposición de subproblemas (más llamados recursivos que subproblemas diferentes).

Programación Dinámica

No es tanto lo que tiene de nuevo esta técnica. Hoy para resolver los ejercicios seguimos sólo tres pasos:

1. Dar una función recursiva que resuelva el problema y argumentar su correctitud.
2. Ver que se cumpla la propiedad de superposición de subproblemas (más llamados recursivos que subproblemas diferentes).
3. Memoizar.

Problema del CD otra vez

- Demos una formulación recursiva para resolver el problema.

Problema del CD otra vez

- Demos una formulación recursiva para resolver el problema.
- $CD(i, k)$: "máxima duración que puedo lograr eligiendo solamente canciones del prefijo $(d_1, ..., d_i)$ en un CD de capacidad k".

Problema del CD otra vez

- Demos una formulación recursiva para resolver el problema.
- $CD(i, k)$: "máxima duración que puedo lograr eligiendo solamente canciones del prefijo $(d_1, ..., d_i)$ en un CD de capacidad k".

$$CD(i,k) = \begin{cases} 0 & \text{si } i = 0 \\ CD(i-1, k) & \text{si } i \neq 0 \wedge d_i > k \\ max(CD(i-1, k), CD(i-1, k-d_i) + d_i) & \text{cc} \end{cases}$$

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.
- No alcanza con dar un ejemplo de subproblema que se resuelve dos veces.

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.
- No alcanza con dar un ejemplo de subproblema que se resuelve dos veces.
- Qvq la cantidad de subproblemas distintos es mucho mucho más chica que la cantidad de llamados recursivos.

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.
- No alcanza con dar un ejemplo de subproblema que se resuelve dos veces.
- Qvq la cantidad de subproblemas distintos es mucho mucho más chica que la cantidad de llamados recursivos.
- En el llamado a $CD(i, k)$ i puede tomar cualquier valor entre 0 y n. k puede tomar cualquier valor entre 0 y K, la capacidad del CD. La cantidad de subproblemas distintos es $\Theta(nK)$.

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.
- No alcanza con dar un ejemplo de subproblema que se resuelve dos veces.
- Qvq la cantidad de subproblemas distintos es mucho mucho más chica que la cantidad de llamados recursivos.
- En el llamado a $CD(i, k)$ i puede tomar cualquier valor entre 0 y n. k puede tomar cualquier valor entre 0 y K, la capacidad del CD. La cantidad de subproblemas distintos es $\Theta(nK)$.
- $CD(i, k)$ en peor caso puede invocarse a sí misma dos veces decrementando el parámetro i. La cantidad de llamados recursivos es $\Theta(2^n)$.

Problema del CD otra vez

- Ahora a ver la propiedad de superposición de subproblemas.
- No alcanza con dar un ejemplo de subproblema que se resuelve dos veces.
- Qvq la cantidad de subproblemas distintos es mucho mucho más chica que la cantidad de llamados recursivos.
- En el llamado a $CD(i, k)$ i puede tomar cualquier valor entre 0 y n. k puede tomar cualquier valor entre 0 y K, la capacidad del CD. La cantidad de subproblemas distintos es $\Theta(nK)$.
- $CD(i, k)$ en peor caso puede invocarse a sí misma dos veces decrementando el parámetro i. La cantidad de llamados recursivos es $\Theta(2^n)$.
- Si $nK << 2^n$ entonces se cumple la propiedad.

Problema del CD otra vez

- Tenemos que decidir con qué estructura representamos al diccionario en el que guardamos los resultados calculados.

Problema del CD otra vez

- Tenemos que decidir con qué estructura representamos al diccionario en el que guardamos los resultados calculados.
- Usemos matriz $DP[0..n][0..K]$ inicializada en $\perp$.

Problema del CD otra vez

- Tenemos que decidir con qué estructura representamos al diccionario en el que guardamos los resultados calculados.
- Usemos matriz $DP[0..n][0..K]$ inicializada en $\perp$.
- Hay que chequear si ya tenemos guardado en el diccionario el valor que nos piden. Si no lo tenemos lo calculamos y lo guardamos. Después retornamos el valor del diccionario.

Problema del CD otra vez

```
int CD(int i, int k){
    if(i == 0) return 0;
    if(duracion[i] > k) return CD(i-1,k);
    else return max(CD(i-1,k),CD(i-1,k-duracion[i]) + duracion[i]);
}
```

Figure: Implementación recursiva de la función CD.

```
int CD(int i, int k){
    if(i == 0) return 0;
    if(DP[i][k] == BOTTOM){
        if(duracion[i] > k) DP[i][k] = CD(i-1,k);
        else DP[i][k] = max(CD(i-1,k),CD(i-1,k-duracion[i]) + duracion[i]);
    }
    return DP[i][k];
}
```

Figure: Implementación Top-Down.