Estructuras de Datos y Algoritmos - IIC2133
Pauta Control 6

1. Dado un string $S = a_1 \cdots a_n$ se define k como el mínimo valor tal que $S = P_1 \cdots P_k$, donde cada P_i es un palíndromo.

a) Escribe la función de recurrencia $K(s, i, j)$ que dado un subtring $s[i{:}j]$ determine el valor del k antes descrito, justificando su correctitud.

Solución:

$$K(s,i,j) = \begin{cases} 1 & \text{si } s[i:j] \text{ es un palíndromo} \\ \min\limits_{x=i}^{j-1}(K(s,i,x) + K(s,x+1,j)) & \text{en otro caso} \end{cases}$$

[0.5pt] por el caso base de la recurrencia

[1pt] por el caso recursivo

[0.5pts] por que los límites del caso recursivo sean correctos. Se perdona hasta 1 error en los límites.

Por ejemplo

Min de x=i … j, o K(s,i,x) + K(s,x,j) son errores en los límites del caso recursivo.

[1pt] Por la justificación de correctitud.

Justificación:

La correctitud del caso base es trivial. **[0pts]**

Lema: existe **alguna** manera de cortar S en dos strings S_1 y S_2 tal que $S = S_1 S_2$ y $k(S) = k(S_1) + k(S_2)$. **[0.25pts]** (Esta propiedad es la que hace que la recurrencia funcione)

No tenemos cómo saber a priori cual es la manera óptima de cortar S, pero por el lema, **alguno** de los cortes entregará el k mínimo. Así que para encontrarlo, probamos **todos** los posibles cortes de S y elegimos el que nos entregue el menor k **[0.25pts]**

[Demostración del lema] **[0.5pts]**

(Para demostrar esto basta con probar que existe al menos un corte que cumple la propiedad del lema)

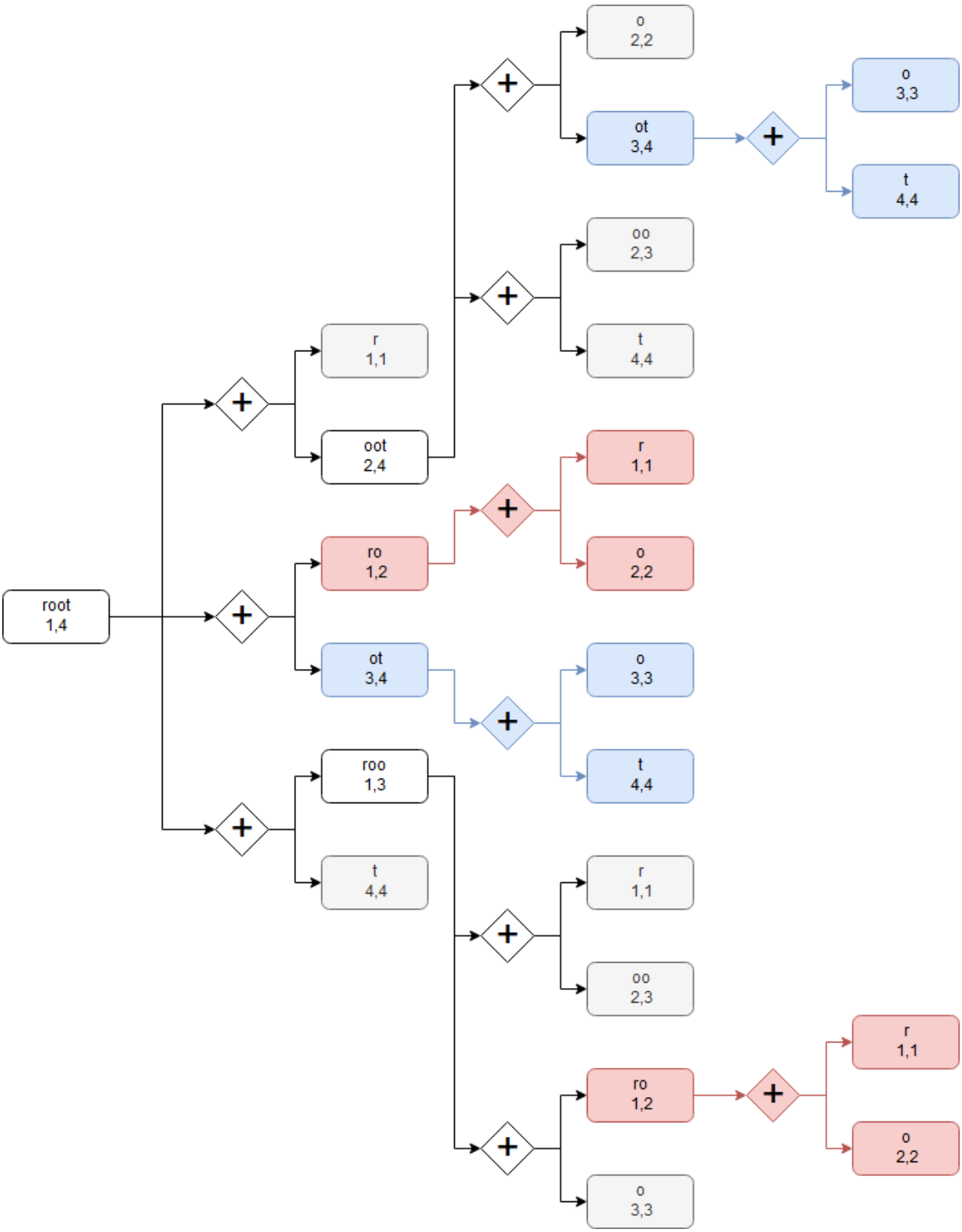
Posible demostración:

Para un string S dado siempre existe una forma de expresarlo como $S = P_1 \cdots P_k$, aunque sea con $k = n$. Si elegimos un y cualquiera entre 1 y $k - 1$, y lo usamos para cortar S en 2 de la siguiente manera: $S_1 = P_1 \cdots P_y$, $S_2 = P_{y+1} \cdots P_k$ tenemos que el k de S_1 es y, y el k de S_2 es $k - y$. Con esto tenemos al menos una forma de cortarlo.

[No es necesario que la justificación sea tan formal, mientras mencionen y justifiquen correctamente los puntos descritos.]

b) Explica y justifica el uso de programación dinámica para este problema. Haz un diagrama de la recursión para respaldar tus argumentos.

Usemos como ejemplo la palabra *root*. (pueden usar la palabra que quieran)


root
1,4
r
1,1
oot
2,4
o
2,2
ot
3,4
o
3,3
t
4,4
oo
2,3
t
4,4
ro
1,2
r
1,1
o
2,2
ot
3,4
o
3,3
t
4,4
roo
1,3
t
4,4
r
1,1
oo
2,3
ro
1,2
r
1,1
o
2,2
o
3,3

[1pt] por mostrar que la recurrencia puede generar un mismo subproblema más de una vez:

Ejemplo: como se ve en el diagrama, las llamadas K(s,1,2) y K(s,3,4) se repiten, y generan un árbol entero repetido. [No es necesario dibujar el diagrama **completo** de la recursión mientras quede claro que se generan subárboles repetidos y los indiquen correctamente]

Sólo si está justificado correctamente, [1pt] por explicar cómo aplicar programación dinámica:

Ejemplo: Podemos calcular el valor de cada llamada una sola vez, guardar su resultado, y las siguientes veces que se haga esa llamada simplemente retornar el valor previamente calculado, así ahorramos generar árboles de llamadas repetidos.

c) **[1pt]** por complejidad con PD:

Con programación dinámica se calculará a lo más una vez cada llamada distinta por lo que la complejidad es el costo de cada llamada una sola vez.

Sabemos que se tiene que verificar si la palabra es un palíndromo por lo que cada llamada toma O(largo sub palabra).

Por lo tanto el tiempo total es la suma de los largos de todas las sub palabras distintas:

$$T(n) = 1 \cdot n + 2 \cdot (n-1) + 3 \cdot (n-2) + \ldots + n \cdot 1$$

$$T(n) \in O(n^3)$$

[0.5pts de bonus] por la complejidad sin programación dinámica:

El peor caso es cuando $k = n$.

Planteamos T(n) como el tiempo de procesar una palabra de largo n

$$T(n) = n + \sum_{x=1}^{n-1} T(x) + T(n-x)$$

Por simetría, eso es equivalente a

$$T(n) = n + 2\sum_{x=1}^{n-1} T(x)$$

Esta función es estrictamente mayor a

$$f(n) = \sum_{x=1}^{n-1} f(x)$$

Desarrollamos:

$$f(n) = f(n-1) + f(n-2) + f(n-3) + \ldots + f(1)$$

$$f(n) = f(n-1) + f(n-1)$$

$$f(n) = 2 \cdot f(n-1)$$

Y esto es $f(n) = 2^{n-1}$

Por lo tanto $T(n) \in \Omega(2^n)$. En otras palabras es al menos exponencial.