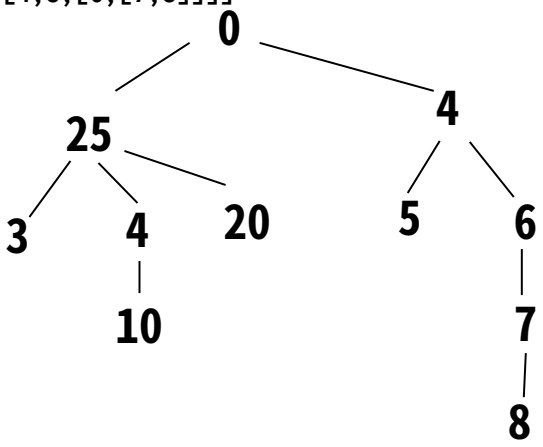
Curso de Programación en C++.
TPL2-2021. Trabajo Práctico de Laboratorio. [2021-07-07]

Ejercicios

[Ej. 1] **maxdepth**: Usando JSON podemos escribir árboles de la siguiente forma `T=[root,T0,T1,T2,...]` donde `root` es la raíz del árbol y `Tj` es el hijo `j` de la raíz. Como es un árbol, los `Tj` pueden ser subárboles, y así recursivamente. Por ejemplo, el árbol de la figura lo podemos escribir como `T=[0,[25,3,[4,10],20],[4,5,[6,[7,8]]]]`



La **máxima profundidad** o **altura** de un nodo es la máxima distancia de ese nodo a cualquiera de sus descendientes a la raíz. Por ejemplo, en este caso la altura de la raíz (nodo 0) es 4 (correspondiente al nodo 8, que es el que está a máxima profundiad), y la altura del nodo 25 es 2.

Consigna: Escribir una función `int maxdepth(json &tree);` que calcula la máxima profundidad del nodo representado por la estructura JSON `tree`.

Ayuda: Puede verse que la altura de un nodo es `h(root)= 1+max(h(Tj))` es decir es la máxima altura de sus hijos más uno. En el ejemplo la altura de los hijos de la raíz 0 son `h(25)=2` y `h(4)=3`, y se comprueba la regla ya que `4=max(2,3)+1`.

Por lo tanto basta con hacer la función recursiva, entonces

- Si el nodo es una hoja debe retornar 0.
- Si no es una hoja debe calcular el máximo de las alturas de los hijos y agregar una unidad.

Como saber si el nodo es una hoja? Notar que si el nodo es una hoja entonces no tendrá hijos y por lo tanto va a ser directamente un entero. Esto se puede derminar con el método `j.is_number()` que retorba verdadero si el JSON `j` es un número. Por el contrario, para los nodos que no son hojas, serán **listas** lo cual se puede determinar con el método `j.is_array()`. (De todas formas hará falta uno sólo, ya que estos JSON serán sólo números o arrays).

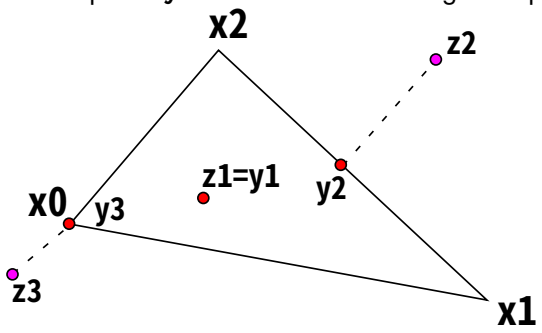
Nota: Notar que para si el nodo tiene `n` hijos entonces el JSON correspondiente tiene `n+1` elementos (el nodo y sus hijos). Por lo tanto para recorrer los hijos hay que recorrer los elementos correspondientes desde la posición `1` a la `n`.

[Ej. 2] **maxleaf**: Con la misma forma de representar árboles que en el ejemplo anterior, escribir una función `int maxleaf(json &tree);`

que retorna el valor máximo de los nodos que son **hojas** (es decir nodos que no tienen hijos). En el caso del ejemplo debe retornar 20, ya que las hojas son **`3,10,20,5,8`** y el máximo es 20. Notar que hay un nodo mayor (el 25) pero no es hoja.

Ayuda: Conviene resolverlo también en forma recursiva. Si el nodo es una hoja (usar el predicado **`is_number()`**) retornar el valor. Si no retornar el máximo de los **`maxleaf`** de los hijos (aplicando recursivamente la función).

[Ej. 3] **xcloser**: Dado un triángulo cuyos vértices son **`xj`** (con **`j=0,1,2`**) y un punto **`z`** que puede estar afuera o dentro del triángulo, determinar el punto **`y`** más cercano del triángulo al punto **`z`**.



Consigna: escribir una función
`void xcloser(MatrixXd &vert,VectorXd &x,VectorXd &y);` que realiza la tarea indicada. Notar que,

- Si el punto **`z`** está dentro del triángulo entonces **`y=z`** (como **`y1,z1`** en la figura).
- Para puntos fuera del triángulo el punto cercano puede caer en algún punto intermedio de un lado (caso **`x2,y2`** o en un vértice **`x3,y3`**).

Ayuda: El problema se puede plantear como un problema de optimización con restricciones, es decir el punto y como está dentro del triángulo (o en su frontera) puede ponerse como

$$\begin{aligned} \alpha &= \underset{\alpha_j}{\operatorname{argmin}} \|z - \sum_{j=0}^{2} \alpha_j x_j\|^2, \\ y &= \sum_{j=0}^{2} \alpha_j x_j. \end{aligned} \tag{1}$$

Con las siguientes rerstricciones

$$\begin{aligned} \sum_j \alpha_j &= 1, \\ 0 \leq \alpha_j &\leq 1 \end{aligned} \tag{2}$$

Notar que la primera restricción es **bilateral** (es decir por igualdad) mientra que las otras restricciones ($0 \leq \alpha_j \leq 1$) son **unilaterales** (es decir por desigualdad). Estas últimas restricciones sólo se **activan** cuando el **z** está fuera del triángulo y por lo tanto **y** está en el contorno.

Veamos primero las ecuaciones que se obtienen suponiendo que el punto está dentro del triángulo, por lo tanto no es necesario tener en cuenta las restricciones unilaterales.

Planteando las ecuaciones para la minimización de ese funcional con restricciones vía multiplicadores de Lagrange, se obtiene el siguiente sistema matricial

$$\begin{bmatrix} X^T X & q_1 \\ q_1' & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \lambda \end{bmatrix} = \begin{bmatrix} X^T z \\ 1 \end{bmatrix} \tag{3}$$

Donde

- X es una matriz de $n_d \times (n_d + 1)$ (en 2D de 2x3) donde las columnas son las coordenadas de los vértices x_0, x_1, x_2.
- q_1 indica un vector de 1's de dimensión $(n_d + 1) \times 1$.
- q_1' indica un vector de 1's de dimensión $1 \times (n_d + 1)$.
- α es un vector de longitud $n_d + 1$ con los parámetros α_j.

El sistema es de tamaño $(n_d + 2) \times (n_d + 2)$ Resolviendo este sistema se puede obtener los parámetros α_j. SI todos los α_j están en el rango $0 \leq \alpha_j \leq 1$ entonces el punto está dentro del triángulo y basta con devolver $y = z$.

Si uno o más de los α son negativos, entonces el punto está fuera del triángulo, por lo tanto hay que activar las restricciones $\alpha_j \geq 0$. Notar que no hace falta chequear las condiciones $\alpha_j \leq 1$ ya que si los $\alpha_j \geq 0$ y la suma es 1 ninguno puede ser $\alpha > 1$.

Para imponer estas restricciones hay que agregar multiplicadores de Lagrange adicionales, y las ecuaciones finales son

$$\begin{bmatrix} X^T X & q_1 & F \\ q_1' & 0 & 0 \\ F & 0 & I - F \end{bmatrix} \begin{bmatrix} \alpha \\ \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} X^T z \\ 1 \\ 0 \end{bmatrix} \tag{4}$$

Donde

- El sistema es de $(2n_d + 3) \times (2n_d + 3)$.
- μ son los multiplicadores de Lagrange correspondiente a las restricciones unilaterales (de tamaño $n_d + 1$).
- F es una matriz de $(n_d + 1) \times (n_d + 1)$ diagonal, cuyos elementos diagonales son $F_{jj} = 1$ si el $\alpha_j < 0$ y si no $F_{jj} = 0$.
- Notar que $I - F$ es el complemento es decir si $F_{jj} =$ para algún j entonces $(I - F)_{jj} = 0$ y viceversa.

Algoritmo:

- Se deben realizar una serie de iteraciones en las cuales se van **activando** las restricciones unilaterales.
- Inicialmente las restricciones (elementos diagonales de F) se ponen todas en 0 (ninguna activa). Es decir que inicialmente $F = 0$.
- (1) Armar el sistema y resolver.
- Extraer los α del vector solución. Si alguno dio negativo poner en 1 el valor correspondiente en la diagonal de F.

- Volver al punto (1). Iterar hasta que todos los α_j den no negativos.
- Calcular el punto $y = X\alpha$

Nota: Las restricciones no se **desactivan** nunca. Una vez que dan negativa en una iteración esa restricción sigue activada para siempre, por más que al activarla el α_j correspondiente va a ser nulo en la siguiente iteración.