IIC2283 — Diseño y Análisis de Algoritmos
Ayudantía 6

Problema 1

En sistemas como LaTeX, se necesita dividir cada párrafo en líneas, de forma que estas no excedan el ancho de la página, y al mismo tiempo optimicen la distribución de las palabras de acuerdo a algún criterio.

Parte 1

Proponga un algoritmo *greedy* que dado un párrafo y un ancho de línea, genere una asignación de saltos de línea razonable, y analice su tiempo de ejecución.

Idea de solución Generar las líneas de forma secuencial. Por cada palabra, se revisa si agregarla a la última línea excedería el ancho de la página. Si no lo excede, entonces se agrega en esa línea, y en caso contrario se agrega al comienzo de una línea nueva.

El algoritmo anterior funciona en tiempo lineal en la cantidad de palabras, asumiendo que estas se entregan como un arreglo con los largos.

Parte 2

Para los siguientes objetivos, demuestre que el algoritmo anterior es correcto, o de un contraejemplo donde no lo sea. Cuando la solución no sea óptima proponga y analice un algoritmo que la genere.

Como notación se utiliza X para el ancho máximo, y ℓ_i para la suma de los largos de las palabras asignadas a la línea i. Las palabras del párrafo se denotan por w_i.

1. Minimizar el número de líneas,

   Idea de solución Es óptimo. Se puede demostrar por inducción en la cantidad de líneas generadas. Si genera una línea, entonces es óptimo trivialmente.

   Si la solución del algoritmo genera $t+1$ líneas, donde la primera está formada por las palabras $w_1, \ldots, w_i$, entonces se considera el párrafo formado por las palabras desde w_{i+1}. Para esa entrada, el algoritmo genera la solución óptima por hipótesis de inducción, y por lo tanto se necesitan al menos t líneas para cubrir esas palabras. Luego, se considera la ubicación del primer salto de línea. Por el funcionamiento del algoritmo, este debe estar a lo más en w_i, por que agregar w_{i+1} a la primera línea excedería el ancho máximo. Por ese motivo, para cualquier asignación correcta, las palabras desde w_{i+1} están desde la segunda línea, y además estas necesitan al menos t líneas, por lo tanto toda solución correcta tiene al menos $t+1$ líneas.

2. Minimizar $\sum_i (X - \ell_i)^3$, sin considerar la última línea,

Idea de solución En este caso no es óptimo. Se puede considerar el contraejemplo: $X = 3$, palabras: a b c d eee. Ahí, la solución óptima tiene costo 2, y el algoritmo genera una solución de costo 8. La solución se puede obtener por programación dinámica, en base a calcular C:

$$C(i) = \begin{cases} (X - |w_1|)^3 & i = 1 \\ \text{mín}\{C(j) + (X - \sum_{t=j+1}^{i} |w_t|)^3 | \sum_{t=j+1}^{i} |w_t| \leq X\} & i > 1 \end{cases}$$

y luego considerar las opciones para la última línea: $\text{mín}\{C(j) | \sum_{t=j+1}^{i} |w_t| \leq X\}$.

3. Minimizar $\text{máx}_i(X - \ell_i)$, sin considerar la última línea,

   Idea de solución Tampoco es óptimo. Se puede usar el mismo contraejemplo del caso anterior. Esto también se puede obtener con programación dinámica.

4. Minimizar $\sum_i (X - \ell_i)$, sin considerar la última línea.

   Idea de solución Es óptimo. Se puede demostrar considerando que la suma de la definición se minimiza al minimizar el número de líneas.

Problema 2

Se consideran dos operaciones S y M, tales que $S(n) = n + 1$ y $M(n) = 2n$. Desarrolle y analice un algoritmo que dado un número como entrada, encuentre una secuencia de operaciones de largo mínimo, tal que al aplicarla desde 1 genere el número dado.

Idea de solución Se considera el problema inverso, donde $S'(n) = n - 1$, y $M'(n) = n/2$. Luego, dado un número n, se busca una secuencia de operaciones que comenzando en n termine en 1. Con una solución a este problema, se pueden invertir los pasos para generar una solución al problema original.

La solución se obtiene con un algoritmo greedy, que elige M' cuando el número es par y S' cuando es impar.

El algoritmo anterior es óptimo. Esto se puede demostrar usando una función auxiliar H, definida sobre la representación binaria del número como:

$$H(n) = \mathit{Largo}(n_b) + \#1(n_b) - 2,$$

donde $\mathit{Largo}(n_b)$ es el largo de la representación binaria de n, y $\#1(n_b)$ es la cantidad de unos de esa representación. Por ejemplo, $H(5) = 5$, por que $\mathit{Largo}(101) = 3$, y $\#1(101) = 2$.

Luego, se considera como cambia H al ejecutar las operaciones:

- $H(S'(n)) = H(n) - 1$, cuando n es impar,
- $H(M'(n)) = H(n) - 1$, cuando n es par,
- $H(S'(n)) \geq H(n)$, cuando $n > 2$ es par.

Ahí, se ve que dado un input n, $H(n)$ es el número de pasos ejecutados por el algoritmo greedy. Por otra parte, $H(n) = 0$ exactamente cuando $n = 1$, y el valor de la función solo se reduce al seguir las elecciones del algoritmo: usar S' con un número par no reduce el valor de la función, y por lo tanto en ningún caso permitiría obtener una solución más corta.

Problema 3

Parte 1 Demuestre la siguiente propiedad: Dado un ciclo en un grafo, donde e es el arco de mayor peso del ciclo, entonces existe un árbol de cobertura de costo mínimo que no contiene e.

Solución Dado un MST que contiene e, al eliminar e se obtienen dos componentes conexas. Por otra parte, como e es parte de un ciclo, necesariamente hay otro arco del ciclo, que conecta ambas componentes, cuyo costo es menor o igual al de e. Agregar ese arco, genera otro árbol de cobertura, que no utiliza e, y cuyo costo es menor o igual al del árbol original.

Parte 2 Demuestre que el siguiente algoritmo es correcto:

```
function MST(G = (V, E))
    Ordenar los arcos por peso, en orden decreciente
    for cada arco e ∈ E do
        if e es parte de un ciclo then
            Remover e del grafo
        end if
    end for
    return G
end function
```

Idea de solución Por inducción en la cantidad de ciclos. Si no hay ciclos, entonces es correcto trivialmente. Si hay $n+1$ ciclos, entonces encontrará un primer arco que pertenece a algún ciclo. Al eliminarlo, el número de ciclos se reduce, y por lo tanto en ese grafo el algoritmo es óptimo por hipótesis de inducción, y además la solución en ese grafo es un árbol de costo mínimo sobre el grafo original, por la propiedad de la parte 1.

Parte 3 En cada iteración el algoritmo debe revisar si existe un ciclo que contenga un arco específico. Describa un algoritmo de tiempo lineal para este problema, y justifique que es correcto.

Idea de solución Una solución es eliminar el arco, y revisar si un extremo es alcanzable desde el otro, con algún algoritmo estándar para esto.

Parte 4 Analice el tiempo de ejecución del algoritmo de la parte 2.

Idea de solución Ordenar los arcos toma tiempo $O(|E| \log |E|)$. Luego, por cada arco, se necesita resolver si pertenece a un arco. Con el algoritmo de la parte 3, eso funciona en tiempo $O(|E| + |V|)$. Luego, en total el algoritmo funciona en tiempo $O(|E| \log |E|) + O(|E| \cdot (|E| + |V|)) \in O(|E|(|E| + |V|))$.