Proyecto de Análisis y Diseño de Algoritmos

Loja Zumaeta, Alex Jose; Neira Riveros, Jorge Luis

I. Pregunta 1: Heurística Voraz

Analice, diseñe e implemente una heurística voraz para el problema MIN-TRIE. Su algoritmo no deberá encontrar necesariamente la respuesta óptima.

Entrada del algoritmo: Una secuencia S de n cadenas de longitud m sobre el alfabeto Σ.

Salida del algoritmo: Un S-ptrie y su número de aristas.

Tiempo de ejecución del algoritmo: $O(nm + mlgm)$.

Espacio utilizado por el algoritmo: $O(nm|\Sigma|)$.

Elección voraz: Insertar primero la posición que tiene menos caracteres distintos

Entrada: Un S-ptrie vacío, una secuencia de cadenas $S = (s_1, s_2, \ldots, s_n)$, y el conjunto de posiciones $P = 0, 1, \ldots, m-1$ a insertar.

Salida: Un S-ptrie con una cantidad mínima de aristas.

Pseudocódigo

```
function MIN-EDGES-SPTRIE(S-PTRIE, S, P)
    if P = ∅ then
        return
    end if
    P_{i*} = P_i : A(P_i) = min{A(P_j) : P_j ∈ P}
    for j = 1 → n do
        S-ptrie.insert(s_j, P_{i*})    ▷ Inserta el caracter P_{i*} de s_j
    end for
    P' = P\{P_{i*}}
    MIN-EDGES-SPTRIE(S-ptrie, S, P')
end function
```

Nota 1: El algoritmo debe ser llamado con un S-ptrie vacío, el conjunto de cadenas S y el conjunto de posiciones P.

Nota 2: La función $A(P_i)$ cuenta los caracteres distintos en la posición P_i de las cadenas. Este procesamiento debe hacerse antes del algoritmo.

Lema 1: (Elección voraz). Existe una solución óptima para el problema que inicia la inserción de caracteres de las cadenas de S en la posición P_i en cada iteración de la recursión, tal que $A(P_i) = min\{A(P_j) : P_j \in P\}$.

Prueba: Sea X el S-ptrie óptimo del problema. Este fue generado siguiendo una permutación ordenada de las posiciones en P. Sea esta permutación $p = (p_1 p_2 \ldots)$, entonces p corresponde con la secuencia creciente $A = (A_{p_1}, A_{p_2}, \ldots) : p_i \in p \wedge A_{p_i} = A(P_i)$. Demostraremos que la solución óptima se da cuando A es creciente, es decir, no tiene inversiones.

Suponga por contradicción que A tiene por lo menos una inversión. Tome la inversión (i, j) en A tal que $j - i$ es mínimo (i y j son índices p_i de A).

Mostraremos que $j - i = 1$. Suponga por contradicción que $j - i > 1$. Entonces existe un índice k tal que $i < k < j$. Si $A_i > A_k$ entonces (i, k) es una inversión de tamaño menor a la escogida (i, j), lo cual es una contradicción a la elección mínima. Si $A_i < A_k$ entonces, como $A_i > A_j$, tenemos $A_j < A_k$ y (k, j) sería una inversión de tamaño menor a la escogida (i, j), lo cual contradice a la elección mínima.

Del párrafo anterior, entonces sea (i, j) una inversión de tamaño mínimo en A. Sea A' la secuencia que resulta de intercambiar A_i y A_j. Esta nueva secuencia A' genera una nueva solución X'.

Demostraremos que X' es solución al problema. A' genera una permutación de posiciones, la cual si se ejecuta en el algoritmo, generamos un nuevo S-ptrie, no necesariamente con cantidad de aristas mínima. Como X' genera un S-ptrie, entonces es una solución al problema.

Mostraremos que la cantidad de aristas en X' es menor o igual a la cantidad de aristas en X.

Como $j = i + 1$, entonces el intercambio se realiza entre 2 índices consecutivos de A. Entonces las aristas en el nivel p_i de X estarán un nivel arriba a las de p_j, como muestra la siguiente imagen.

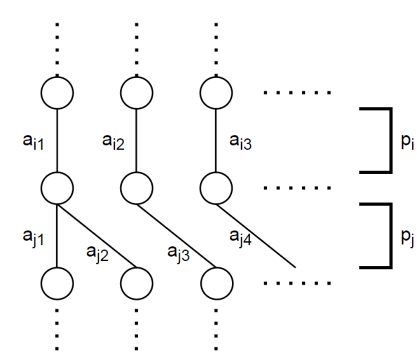

Fig 1.1 Posiciones p_i y p_j de la solución X

Sean a_{i1}, a_{i2}, ... los caracteres únicos en p_i, y a_{j1}, a_{j2}, ... los de p_j. Como $A_i > A_j$, entonces $|p_i| > |p_j|$. Además, se puede representar la secuencia $p_i p_j$ del S-ptrie como un grafo bipartito.

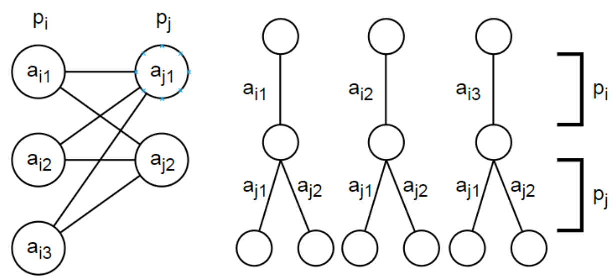

Fig 1.2 Posiciones p_i y p_j de la solución X_1 en un grafo bipartito

La figura 1.2 muestra una relación total entre los caracteres de p_i y p_j, es decir, existen cadenas en S que contienen a cada par de caracteres $a_i a_j$ en las posiciones $p_i p_j$. La figura también muestra al S-ptrie X_1 que generaría el grafo bipartito. Note que p_i tendrá una arista por caracter, y p_j tendrá A_j aristas por cada arista en p_i. La cantidad total de aristas sería $A_i + A_i \times A_j$.

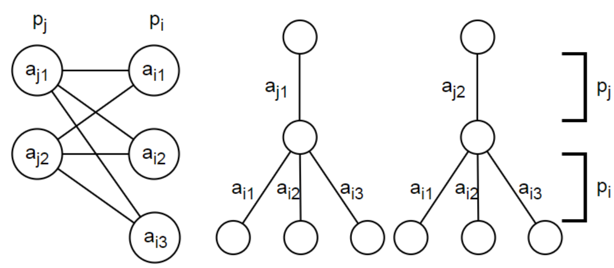

Fig 1.3 Posiciones p_j y p_i de la solución X_2 en un grafo bipartito

Cuando intercambiamos la inversión, obtenemos el grafo bipartito de la figura 1.3, y su correspondiente S-ptrie X_2 que lo genera. Note que la cantidad total de aristas será $A_j + A_i \times A_j$. Como $A_i > A_j$, entonces observamos que la cantidad de aristas disminuye al quitar la inversión, por lo que, X_2 tendrá menos aristas que X_1.

Si el S-ptrie no genera un grafo bipartito completo, note que su grafo puede formarse a partir de la eliminación de aristas de su grafo bipartito completo correspondiente, lo que le quitaría solo 1 arista a ambos S-ptrie X_1 y X_2. Note también que no pueden quedar nodos aislados.

A partir de esta prueba, concluimos que la solución X', la cual se generó por intercambiar las posiciones de una inversión en A de X, es más óptima dado que X es óptimo. Por este motivo, la solución más óptima se dará cuando A no presente inversiones, es decir, sea creciente.

Lema 2: (Subestructura óptima) Sea X la solución óptima para (S, P), entonces $x' = X \backslash \{P_{i*}\}$ es una solución óptima para (S, P').

Prueba: Suponga por contradicción que X' no es óptima. Entonces, existe Y' tal que $aristas(Y') < aristas(X')$. Pero, en este caso $Y = Y' \cup \{P_{i*}\}$ y $aristas(Y) = aristas(Y') + A(P_{i*}) < aristas(X') + A(P_{i*}) = aristas(X)$, lo cual es una contradicción a la optimalidad de X.

Implementación del algoritmo

```
vector<string> cad(n);
vector<unordered_set<int>> charByPosition(m);
for(ll i = 0; i < n; i++){
    cin >> cad[i];
    for(ll j = 0; j < m; j++)
        charByPosition[j].insert(cad[i][j]-'a');
}
```

En primer lugar, creamos un vector de cadenas(cad) en el que almacenaremos las cadenas ingresadas lo cual nos tomará un tiempo de n, ya que esta es la cantidad de inputs que tendremos. Además, un vector de unordered_set, que nos permitirá obtener cuántos caracteres distintos hay en cada posición de las cadenas, y esto a su vez toma un tiempo de $O(1)$ por cada posición. Para poder hacer estas operaciones necesitamos recorrer cada palabra ingresada, esto nos tomará un tiempo de m ya que esta es la longitud de cada input. **Tiempo total:** $O(n * m)$.

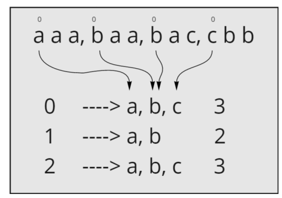

Fig 1.4 Cantidad de caracteres únicos por índice

```
bool sortP(pair<ll, ll> p1, pair<ll, ll> p2){
    if(p1.second == p2.second)  return p1.first < p2.first;
    return p1.second < p2.second;
}
...
vector<pair<ll, ll>> permut;
for(ll i = 0; i < m; i++)
    permut.push_back(make_pair(i, charByPosition[i].size()));
sort(permut.begin(), permut.end(), sortP);
```

Creamos un vector permutación en el que guardaremos la posición y la cantidad de caracteres distintos en él, esto tomará un tiempo de $O(m)$. Después, ordenamos el vector de acuerdo al número de caracteres distintos, lo que tomará $O(mlg(m))$. **Tiempo total:** $O(m + mlg(m))$.

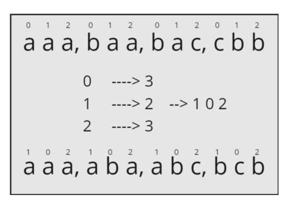

Fig 1.5 Orden óptimo de acuerdo a los índices

```
unordered_set<int> characters;
vector<int> alf;
for(string &s : cad){
    string news = "";
    for(ll i = 0; i < m; i++){
        news += s[permutation[i].first];
        if(!characters.count(s[permutation[i].first] - 'a')){
            characters.insert(s[permutation[i].first] - 'a');
            alf.push_back(s[permutation[i].first] - 'a');
        }
    }
    s = news;
}
```

Luego, reordenaremos los caracteres de las cadenas de acuerdo a la permutación obtenida. En el vector alf guardaremos el orden de caracteres, y usamos un unordered_set (characters) para saber si ya fue insertado en el vector. Como recorreremos las n palabras y por cada palabra sus m caracteres nos tomará un tiempo de n * m. **Tiempo total:** O($n * m$)

```
vector<vector<ll>> sptrie(m * n + 1);
for(ll i = 0; i < m * n + 1; i++)
    sptrie[i].resize(alf.size(), 0);
```

Creamos nuestra matriz S-ptrie. Tendrá como máximo (si todos los caracteres de todas las cadenas son distintas) $m*n+1$ filas (un nodo por fila) y $|\Sigma|$ columnas. **Tiempo total:** O($m * n$).

```
ll parent = 0, node = 0;
for(string s : cad){
    parent = 0;
    for(char c : s){
        if(sptrie[parent][c-'a'] == 0)
            //No existe, la creamos
            sptrie[parent][c-'a'] = ++node;
        parent = sptrie[parent][c-'a'];
    }
}
```

i	a	b	c
0 →	1	7	0
1 →	2	4	0
2 →	3	0	0
3 →	0	0	0
4 →	5	0	6
5 →	0	0	0
6 →	0	0	0
7 →	0	0	8
8 →	0	9	0

Fig 1.6 Matriz del S-ptrie

Finalmente, para cada cadena guardada en cad, la insertamos en el S-ptrie. Como recorremos el vector de palabras(n), y para cada palabra sus caracteres(m) nos tomará n*m **Tiempo total:** O(n * m)

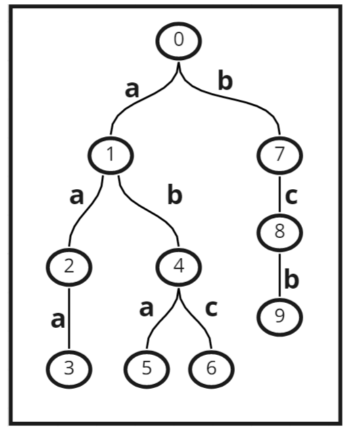

Fig 1.7 S-ptrie con el número mínimo de aristas.

Como salida, mostramos el S-ptrie formado, la permutación óptima y la cantidad mínima de aristas.

II. PREGUNTA 2: RECURRENCIA

Plantee una recurrencia para $OPT(i,j)$.

Definimos $OPT(i,j)$ como el número de aristas de una solución óptima para el subproblema que contiene a las cadenas $s_i, ..., s_j$. Consideremos para la notación: $aristas(X) = |X|$.

Sea X el S-ptrie óptimo para las cadenas $s_i, s_{i+1}, ..., s_j$, entonces $OPT(i,j) = |X|$. Note que existe una cadena s_k ubicada entre s_i y s_j tal que $i \leq k < j$. Entonces X'_1 es el S-ptrie óptimo para las cadenas $s_i, s_{i+1}, ..., s_k$ y X'_2 el S-ptrie óptimo para las cadenas $s_{k+1}, ..., s_j$.

Prueba: Suponga por contradicción que X'_1 y X'_2 no son óptimos. Entonces existen los S-ptries Y'_1 y Y'_2 tal que $|Y'_1 \cup Y'_2| < |X'_1 \cup X'_2|$. Note que al unir Y'_1 y Y'_2 tendríamos $|Y| = |Y'_1 \cup Y'_2| < |X'_1 \cup X'_2| = |X|$, lo cual es una contradicción a la optimalidad de X.

Lema:

$$OPT(i,j) = min\{OPT(i,k) + OPT(k+1,j) : i \leq k < j\}$$

Prueba: Sea X el S-ptrie óptimo para las cadenas $s_i, s_{i+1}, ..., s_j$. Sea X'_1 el S-ptrie óptimo para las cadenas $s_i, s_{i+1}, ..., s_k$ y X'_2 el S-ptrie óptimo para las cadenas $s_{k+1}, ..., s_j$, tal que $i \leq k < j$. Como $|X'_1|$ y $|X'_2|$ son soluciones óptimas para sus correspondientes subproblemas, por propiedad de subestructura óptima demostrada en la prueba anterior, entonces

$$|X'_1 \cup X'_2| \in \{|X'_{ik} \cup X'_{(k+1)j}| : i \leq k < j\}$$

donde X'_{ik} es una solución para las cadenas $s_i, ..., s_k$ y $X'_{(k+1)j}$ es una solución para las cadenas $s_{k+1}, ..., s_j$. Suponga por contradicción que existe k^* en dicho rango tal que $|X'_{ik^*} \cup X'_{(k^*+1)j}| < |X'_1 \cup X'_2|$. En este caso, $X'_{ik^*} \cup X'_{(k^*+1)j}$ tendría una cantidad de aristas menor a X, lo cual es una contradicción a la optimalidad de X. Esto demuestra que

$$|X'_1 \cup X'_2| = min\{|X'_{ik} \cup X'_{(k+1)j}| : i \leq k < j\}$$

Con este lema en mano, diseñamos un algoritmo recursivo para el problema.