Strings

Pablo Blanc
(con diapos robadas a Leopoldo Taravilse)

Buen Kilo de Pan Flauta

Training Camp 2017

Contenidos

Contenidos

Qué es String Matching?

Definición del problema

El problema de String Matching consiste en, dados dos strigns S y T, con $|S| < |T|$, decidir si S es un substring de T, es decir, si existe un índice i tal que

$$S[0] = T[i], S[1] = T[i+1], \ldots, S[|S|-1] = T[i+|S|-1]$$

Solución Trivial

- Existe una solución $O(|S||T|)$ que consiste en evaluar cada substring de T de longitud $|S|$ y compararlo con S caracter por caracter.

Solución Trivial

- Existe una solución $O(|S||T|)$ que consiste en evaluar cada substring de T de longitud $|S|$ y compararlo con S caracter por caracter.
- Esta solución no reutiliza ningún tipo de información sobre S o sobre T.

Solución Trivial

- Existe una solución $O(|S||T|)$ que consiste en evaluar cada substring de T de longitud $|S|$ y compararlo con S caracter por caracter.
- Esta solución no reutiliza ningún tipo de información sobre S o sobre T.
- Existen soluciones que reutilizan información y así nos evitan tener que hacer $O(|S||T|)$ comparaciones.

Contenidos

Bordes de un String

Definición de borde

Un borde de un string S es un string B ($|B| < |S|$) que es a su vez prefijo y sufijo de S.

Bordes de un String

Definición de borde

Un borde de un string S es un string B ($|B| < |S|$) que es a su vez prefijo y sufijo de S.

Por ejemplo, *a* y *abra* son bordes de *abracadabra*.

Detección de bordes

- Un problema muy común es querer encontrar el borde más largo de un string.

Detección de bordes

- Un problema muy común es querer encontrar el borde más largo de un string.
- Nuevamente podríamos comparar cada prefijo con el sufijo correspondiente, lo que nos llevaría a una solución cuadrática.

Detección de bordes

- Un problema muy común es querer encontrar el borde más largo de un string.
- Nuevamente podríamos comparar cada prefijo con el sufijo correspondiente, lo que nos llevaría a una solución cuadrática.
- Existe una solución lineal para el cálculo del máximo borde de un string.

Detección de bordes

- Un problema muy común es querer encontrar el borde más largo de un string.
- Nuevamente podríamos comparar cada prefijo con el sufijo correspondiente, lo que nos llevaría a una solución cuadrática.
- Existe una solución lineal para el cálculo del máximo borde de un string.
- Esta solución se basa en encontrar el mayor borde de todos los prefijos del string uno por uno.

Detección de bordes

Lema 1

Si S' es borde de S y S'' es borde de S' entonces S'' es borde de S.

Al ser S'' prefijo de S' y S' prefijo de S, entonces S'' es prefijo de S, y análogamente es sufijo de S.

Detección de bordes

Lema 1

Si S' es borde de S y S'' es borde de S' entonces S'' es borde de S.

Al ser S'' prefijo de S' y S' prefijo de S, entonces S'' es prefijo de S, y análogamente es sufijo de S.

Lema 2

Si S' y S'' son bordes de S y $|S''| < |S'|$, entonces S'' es borde de S'.

Como S'' es prefijo de S y S' también, entonces S'' es prefijo de S'. Análogamente S'' es sufijo de S'.

Detección de bordes

Lema 1

Si S' es borde de S y S'' es borde de S' entonces S'' es borde de S.

Al ser S'' prefijo de S' y S' prefijo de S, entonces S'' es prefijo de S, y análogamente es sufijo de S.

Lema 2

Si S' y S'' son bordes de S y $|S''| < |S'|$, entonces S'' es borde de S'.

Como S'' es prefijo de S y S' también, entonces S'' es prefijo de S'. Análogamente S'' es sufijo de S'.

Lema 3

Si S' y S'' son bordes de S y el mayor borde de S' es S'', entonces S'' es el mayor borde de S de longitud menor a $|S'|$.

Solución lineal al problema de detección de bordes

- Empezamos con el prefijo de longitud 1. Su mayor borde tiene longitud 0. (Recordemos que no consideramos al string entero como su propio borde).

Solución lineal al problema de detección de bordes

- Empezamos con el prefijo de longitud 1. Su mayor borde tiene longitud 0. (Recordemos que no consideramos al string entero como su propio borde).
- A partir del prefijo de longitud 1, si al borde más largo del prefijo de longitud i le sacamos el último caracter, nos queda un borde del prefijo de longitud $i - 1$.

Solución lineal al problema de detección de bordes

- Empezamos con el prefijo de longitud 1. Su mayor borde tiene longitud 0. (Recordemos que no consideramos al string entero como su propio borde).
- A partir del prefijo de longitud 1, si al borde más largo del prefijo de longitud i le sacamos el último caracter, nos queda un borde del prefijo de longitud $i - 1$.
- Luego probamos con todos los bordes del prefijo de longitud $i - 1$ de mayor a menor, hasta que uno de esos bordes se pueda extender a un borde del prefijo de longitud i. Si ninguno se puede extender a un borde del prefijo de longitud i (ni siquiera el borde vacío), entonces el borde de dicho prefijo es vacío.

Algoritmo de detección de bordes

```
1  int i=1, j=0;
2  bordes[0] = 0;
3  while(i<n)
4  {
5      while(j>0 && st[i] != st[j])
6          j = bordes[j-1];
7      if(st[i] == st[j])
8          j++;
9      bordes[i++] = j;
10 }
```

Este es el código del algoritmo de detección de bordes siendo *st* el string y *n* su longitud.

Algoritmo de detección de bordes

```
1  int i=1, j=0;
2  bordes[0] = 0;
3  while(i<n)
4  {
5      while(j>0 && st[i] != st[j])
6          j = bordes[j-1];
7      if(st[i] == st[j])
8          j++;
9      bordes[i++] = j;
10 }
```

Este es el código del algoritmo de detección de bordes siendo *st* el string y *n* su longitud.
En bordes[*i*] queda guardada la longitud del máximo borde del prefijo de *st* de longitud *i*. Luego en bordes[$n-1$] queda guardada la longitud del máximo borde de *st*.

Correctitud del Algoritmo

```
1 | while(j>0 && st[i] != st[j])
2 |       j = bordes[j-1];
```

En estas dos líneas comparamos el mayor borde del prefijo de longitud i con el mayor borde del prefijo de longitud $i-1$. Si dicho borde no se puede extender, entonces probamos con el mayor borde de ese borde, y así sucesivamente.

```
1 | if(st[i] == st[j])
2 |       j++;
3 | bordes[i++] = j;
```

En estas líneas comparamos a ver si el borde efectivamente se puede extender (o si es el borde vacío y no se puede extender) y guardamos el borde en el arreglo bordes.

Complejidad del Algoritmo

El único lugar donde decrementamos *j* es en la línea

```
1 | j = bordes[j-1];
```

Además, *j*, que empieza inicializado en 0, se incrementa a lo sumo *n* veces, y nunca es menor que 0, por lo que decrementamos *j* a lo sumo *n* veces, luego la complejidad del algoritmo es lineal en el tamaño del string.

String matching con bordes

¿Podremos resolver el problema de string matching con el algoritmo de bordes?

String matching con bordes

¿Podremos resolver el problema de string matching con el algoritmo de bordes? ¡Si!
Una manera de resolver el problema de string matching en tiempo lineal es concatenando los strings S+'$'+$T$ y calculando los bordes (aquí $ es un caracter que no esta en S ni T). Siempre que el borde en alguna posición correspondiente a T sea de longitud $|S|$, quiere decir que hay un substring en T que concide con S.

Contenidos

String Matching

- Vimos que existen soluciones más eficientes que $O(|S||T|)$ para el problema de String Matching.

String Matching

- Vimos que existen soluciones más eficientes que $O(|S||T|)$ para el problema de String Matching.
- Knuth-Morris-Pratt (también conocido como KMP) es una de ellas y su complejidad es $O(|T|)$

String Matching

- Vimos que existen soluciones más eficientes que $O(|S||T|)$ para el problema de String Matching.
- Knuth-Morris-Pratt (también conocido como KMP) es una de ellas y su complejidad es $O(|T|)$
- KMP se basa en la tabla de bordes. La idea es que si el string viene matcheando y de repente no matchea, no empezamos de cero sino que empezamos del borde. Por ejemplo, si matcheó hasta *abracadabra* y luego no matchea, podemos ver qué pasa matcheando con el borde *abra*.

Código de KMP

```
1  string s, t;
2  void fill_table()
3  {
4      int pos = 2, cnd = 0;
5      kmp_table[0] = -1;
6      kmp_table[1] = 0;
7      while(pos<s.size())
8      {
9          if(s[pos-1] == s[cnd])
10             kmp_table[pos++] = ++cnd;
11         else if(cnd>0)
12             cnd = kmp_table[cnd];
13         else
14             kmp_table[pos++] = 0;
15     }
16 }
```

Así llenamos la tabla de KMP.

Código de KMP

```
1  int kmp(){
2      fill_table();
3      int m=0, i=0;
4      while(m+i<t.size()){
5          if(s[i] == t[m+i]){
6              if(i==s.size()-1)
7                  return m;
8              i++;
9          }
10         else{
11             m = m+i-kmp_table[i];
12             if(kmp_table[i]>-1)
13                 i = kmp_table[i];
14             else
15                 i = 0;
16         }
17     }
18     return -1;
19 }
```

Contenidos

Qué es un Trie?

Definición de Trie

Los tries sirven para representar diccionarios de palabras. Un trie es un árbol de caracteres en el que cada camino de la raiz a un nodo final (no necesariamente una hoja) es una palabra de dicho diccionario.

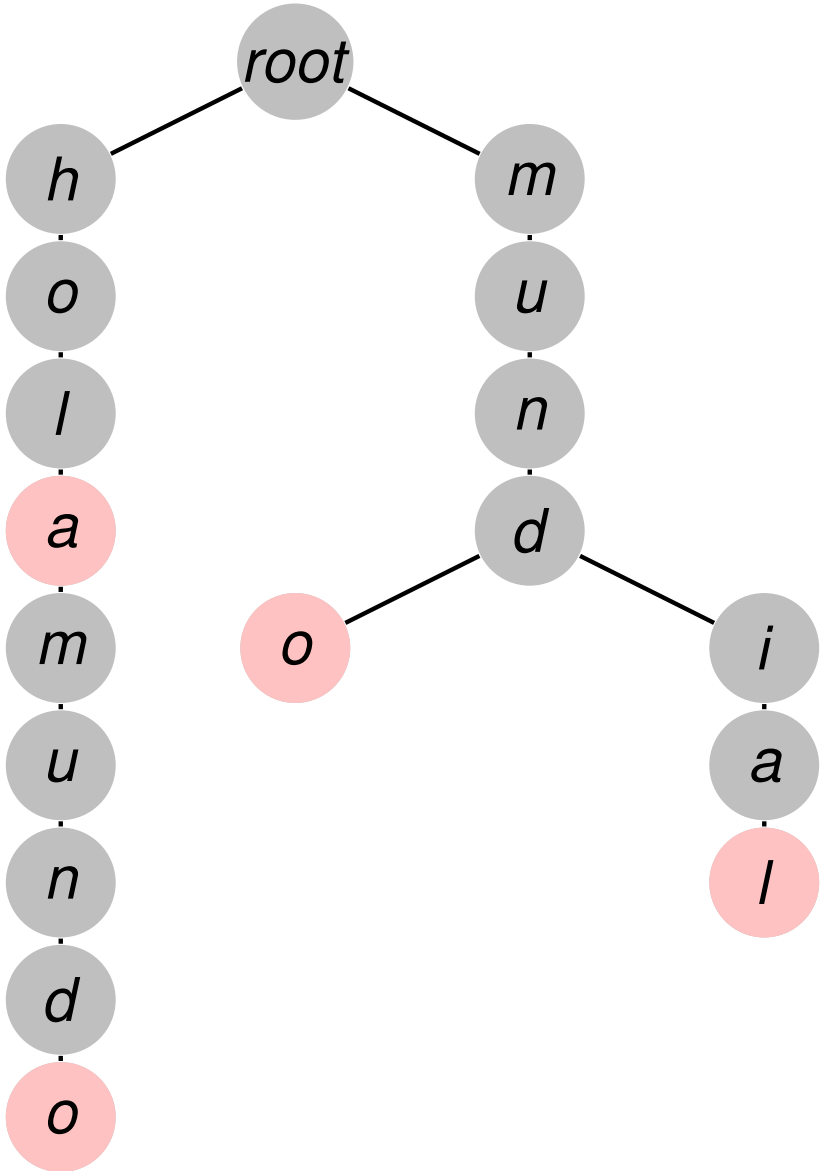
Qué es un Trie?

Definición de Trie

Los tries sirven para representar diccionarios de palabras. Un trie es un árbol de caracteres en el que cada camino de la raiz a un nodo final (no necesariamente una hoja) es una palabra de dicho diccionario.

Veamos un ejemplo de un Trie con las palabras *hola*, *holamundo*, *mundo* y *mundial*.

Ejemplo de un Trie



Código del Trie

```
1  struct trie{
2      map<char,int> sig;
3      bool final;
4  //puede ser map<int,int> si el alfabeto son enteros
5  };
6  trie t[MAXN];
7  int n;
8  void init()
9  {
10     n = 1;
11 t[0].sig.clear();
12 t[0].final = false;
13 }
```

MAXN en este caso es una constante que determina el máximo tamaño del trie.

Código del Trie

```
1  void insertar(string st){
2      int pos = 0;
3      for(int i=0;i<st.size();i++){
4          if(trie[pos].sig.find(st[i])==trie[pos].sig.end()){
5              trie[pos].sig[st[i]] = n;
6              trie[n].sig.clear();
7              trie[n].final = false;
8              n++;
9          }
10         pos = trie[pos].sig[st[i]];
11     }
12     trie[pos].final = true;
13 }
```

Ejemplo

Diseño de Camisetas

Dados dos equipos de rugby con n jugadores cada uno, quieren compartir las camisetas (un jugador de cada equipo por cada camiseta) de modo tal que cada camiseta tenga un prefijo común entre los apellidos de los dos jugadores que la usan (es válido el prefijo vacío), y entre todas las camisetas usen la mayor cantidad de letras posibles.

Problema D - TAP 2012. Link a la prueba: http://goo.gl/ypdYS

Diseño de Camisetas

La solución al problema consiste en:

- Probar que los dos jugadores (de distintos equipos) con el prefijo común más largo usan la misma camiseta. (Esta parte queda como ejercicio).

Diseño de Camisetas

La solución al problema consiste en:

- Probar que los dos jugadores (de distintos equipos) con el prefijo común más largo usan la misma camiseta. (Esta parte queda como ejercicio).
- Insertar todos los apellidos en un trie.

Diseño de Camisetas

La solución al problema consiste en:

- Probar que los dos jugadores (de distintos equipos) con el prefijo común más largo usan la misma camiseta. (Esta parte queda como ejercicio).
- Insertar todos los apellidos en un trie.
- Cada nodo del trie que es parte de a apellidos del equipo 1 y b apellidos del equipo 2 aporta $min(a, b)$ letras a las camisetas.

Problemas

- http://goo.gl/gQOSG
- http://goo.gl/KTVKd

Contenidos

Empezamos con los problemas

Dado S encontrar el mayor n tal que existe T con $T^n = S$.
http://www.spoj.com/problems/FINDSR/
https://icpcarchive.ecs.baylor.edu/index.php?option=onlinejudge&page=show_problem&problem=1027

Problema

Cuantas veces puedo meter S en un texto de longitud l?
http://www.spoj.com/problems/FILRTEST/

Problema

Extend to Palindrome.
http://www.spoj.com/problems/EPALIN/

Problema

Given a list of n ($2 \leq n \leq 100000$) phone numbers, determine if it is consistent in the sense that no number is the prefix of another.
http://www.spoj.com/problems/PHONELST/

Problema

Portuñol.
https://icpcarchive.ecs.baylor.edu/index.php?option=onlinejudge&page=show_problem&problem=3803

Ultimo Problema

Given an array of n ($2 \leq n \leq 100000$) numbers, we wish to choose a contiguous sub-sequence of the array, so that the bitwise XOR of all chosen numbers is maximum.
https://icpcarchive.ecs.baylor.edu/index.php?option=onlinejudge&page=show_problem&problem=2683
Y una parecido mas dificil
http://www.spoj.com/problems/SUBXOR/