Documentació projecte ESIN

Victor Gómez
Pere Montpeó

Justificació de la representació escollida

diccionari.rep

```
struct node {
  char valor;
  node *esq;
  node *dret;
  node *cent;
};
node *_arrel;
nat _n_paraules;
...
```

L'estructura escollida per la representació de la classe `diccionari` és un arbre ternari de cerca (TST). Aquesta estructura, semblant a un BST, facilita la cerca incremental d'*strings* per prefixos, que és l'objectiu que cercàvem. Els arbres ternaris de cerca faciliten la descomposició de paraules.

Cada node conté tres punters: dos apunten als fills esquerre i dret d'un BST i un altre central que forma un sub BST i que conté els símbols de la posició següent de totes les claus que tenen el mateix prefix.

Tenim també un punter al primer node `node *_arrel` per facilitar l'accés i un natural `nat _n_paraules` per reduir els cost de calcular el nombre de paraules a constant.

De mitjana, el cost de les operacions d'inserir, eliminar i consultar un node és de $O(l \cdot \log(s))$, sent s el nombre de símbols i l la longitud mitjana dels símbols de les claus. Té uns costos millors que un trie de tipus primer fill següent germà que seria $O(l \cdot s)$ però no millors que un vector punter que tindria $O(l)$.

Els arbres ternaris de cerca combinen l'eficiència en l'accés dels subarbres amb l'estalvi de memòria, ja que com s'ha dit és semblant al BST. L'eficiència dels TST varia significativament depenent de les entrades. Van millor quan s'afegeixen strings similars, especialment quan aquests comparteixen prefix. Alternativament, els TST són efectius també quan emmagatzemen un gran nombre d'*strings* curts com en el cas d'un diccionari. Els costos són molt similars als del BST; normalment, el temps mitjà és logarítmic, però pot ser lineal en el pitjor dels casos.

Si comparem el tries amb les taules de dispersió, les taules de dispersió freqüentment usen més memòria que els tries i són més lentes quan es tracta de comparar un *string* que no es troba dintre de l'estructura, perquè ha de comparar tot l'*string* sencer en comptes de només uns caràcters com fa el tries. Així que no ho vam considerar una bona opció.

Per tant, encara que no sigui la implementació més ràpida de tries (la de vector punter és més ràpida que aquesta), els TST són la millor opció per emmagatzemar grans quantitats d'*strings* gràcies a la seva

eficiència emmagatzemant informació i sent perfecte per guardar *strings* similars. Per tant, considerem que és la millor estructura per a un diccionari, ja que estalvia més memòria i, quan parlem de grans quantitats de paraules similars com és el cas d'un diccionari, és molt important no consumir en excés tots els recursos.

anagrames.rep

```
struct node_hash {
  string k;
  list<string> v;
  node_hash* seg;
  ...
};
node_hash **_taula;
nat _quants;
nat _M;
...
```

Per a la representació de la classe `anagrames` s'ha escollit una estructura de taula de dispersió per la velocitat d'accés a les dades quan tenim un conjunt de dades grans. El cost de les operacions d'inserir, buscar (`mateix_anagrama_canonic`) és constant `O(1)` un cop tenim l'índex, i podria ser `O(n)` en el pitjor dels casos, en cas de col·lisions. Per tal d'evitar les col·lisions, es crea sempre una taula amb un nombre primer, ja que la possibilitat de col·lisions és menor, i quan s'arriba al 90% de càrrega practiquem la redispersió.

Hem escollit la taula d'encadenats indirectes en què cada entrada apunta a la llista encadenada de sinònims. La taula no guarda els elements en si, sinó un punter al primer element de la llista.

La taula de dispersió conté un `string k` que serà la clau que identifica l'anagrama canònic i que relacionarem amb la posició de la taula, una llista de sinònims `list<string> v`, que contindrà els sinònims relacionats amb la clau, i un punter `node_hash* seg`, que marcarà un punter a l'element següent de la llista en cas de col·lisió.

També tenim una variable amb la taula de dispersió per poder accedir-hi `node_hash **_taula`, un natural amb el nombre d'elements inserits `nat _quants` i, finalment, un natural amb la mida de la taula `nat _M`. Amb aquestes dues últimes variables podrem calcular el factor de càrrega i veure si necessitem aplicar una redispersió.

En aquest cas, hem escollit taules de dispersió perquè permeten l'accés a l'instant quan l'element es troba dintre de l'estructura. En el plantejament ens demanen una llista amb els elements que comparteixen el mateix canònic en `mateix_anagrama_canonic`, per tant aquests compartiran la mateixa clau `string k`. Com hem comentat en l'estructura anterior, les taules de dispersió són una molt bona opció quan la clau que busquem ja es troba inserida a l'estructura, com és en aquest cas sent llavors una elecció més eficient que el tries.

Un cop trobada la clau a l'estructura, s'introduirà l'element en la llista de sinònims per tal que pugui ser accessible en la pròxima cerca. En cas que hi hagi col·lisió en el punter `seg` es crearà o s'actualitzarà un altre punter amb la informació de la col·lisió. Si la funció de dispersió és correcta, com és en aquest cas, les col·lisions seran mínimes.

Hem escollit les taules de dispersió d'encadenats indirectes. Aquest tipus de taules faciliten l'ordenació per clau, cosa que ens era important en aquesta part del projecte. L'accés al primer element no és immediat, sinó que es fa mitjançant una taula. Les taules d'encadenats indirectes ens permeten no haver de comprovar si una posició està lliure o no i treballar sense necessitat de llistes auxiliars, per tant hem considerat que era una estructura més eficient que les taules d'encadenats directes o d'encaminament obert.

iter_subset.rep

```
nat _n;
nat _k;
subset _actual;
bool _final;
...
```

L'estructura de `iter_subset` en si mateixa és un renombrament d'un vector de naturals, per tant, la definició de la classe es basa en aquest i ja ens ve donada. Els costos per les operacions d'incrementar o construir seran costos lineals en funció de la variable `k`.

L'estructura conté un natural `nat _n` que ens dona la mida màxima del *string* donat i un natural `nat _k` que ens dona la mida del nostre vector. L'estructura també té un subset `subset _actual`, que ens diu el subset que tenim en un moment donat, i un booleà `bool _final`, que ens marca si ens trobem en l'últim element del subconjunt.

Pensant en els temps de càlcul, vam decidir que en comptes de calcular tots els subsets era molt millor calcular el subset on ens trobàvem en un moment donat (`subset _actual`) i que, en cas que volguéssim avançar l'estructura, calculés el següent. Estalviàvem molt més temps perquè si calculéssim tots els subsets d'un *string* molt gran podríem acabar trigant molt i podria ser que només en necessitéssim un o dos.

L'estructura amb el booleà `bool _final` era per poder accedir rapidament a saber si estàvem en l'última posició, ja que només havíem de preguntar si era vertader. Això és essencial en un vector, ja que quan el recorres vols saber si estàs en l'última posició per acabar de recorre'l rapidament i afegint un booleà amb la resposta fèiem que el temps de calcular-ho fos constant.

Per tant considerem que aquesta és la millor estructura quant a temps i eficiència, ja que combina una càrrega petita de treball amb uns accessos ràpids.

Estil i convencions

Per les variables hem adoptat el següent conveni:

- Constants: majúscules i paraules separades per guions baixos
- Variables privades: guió baix a l'inici

```
_quants // variable privada
CAPACITAT // constant
```

La identació és de 2 espais. Al voltant dels operadors va un espai en blanc.

Si un dels blocs de `if else` té un claudàtor, l'altra també encara que sigui una sola línia. L'`else` va a la mateixa línia que el claudàtor que tanca. Hi ha d'haver un espai entre la entre `if` i el parèntesi obert, igual que abans del claudàtor que obre.

```
if (condicio) { // Espai després d'IF i abans de {
  ...  // 2 espais d'identació
} else {  // L'else va a la mateixa línia que el claudàtor que tanca
  ...
}
```

Hem pres com a referència de la guia d'estil de Google en C++ només obviant el nom de les variables privades (no recomana els guions precedents) i la llargada de línies de 80 caràcters com a màxim, ja que les pantalles actuals permeten visualitzar més de 80 caràcters.