Índice

1. Estructuras

1.1. SegmentTree

```java
class SegmentTree {
  private int[] st, A;
  private int n;
  private int left (int p) { return p << 1; }
  private int right(int p) { return (p << 1) + 1; }
  private void build(int p, int L, int R) {
    if (L == R)
      st[p] = L;
    else {
      build(left(p) , L, (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R);
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  } }

  private int rmq(int p, int L, int R, int i, int j) {
    if (i >  R || j <  L) return -1;
    if (L >= i && R <= j) return st[p];
    int p1 = rmq(left(p) , L, (L+R) / 2, i, j);
    int p2 = rmq(right(p), (L+R) / 2 + 1, R, i, j);
    if (p1 == -1) return p2;
    if (p2 == -1) return p1;
    return (A[p1] <= A[p2]) ? p1 : p2; }

  private int update_point(int p, int L, int R, int idx, int new_value) {
    int i = idx, j = idx;
    if (i > R || j < L)
      return st[p];
    if (L == i && R == j) {
      A[i] = new_value;
      return st[p] = L;
    }
    int p1, p2;
    p1 = update_point(left(p) , L, (L + R) / 2, idx, new_value);
    p2 = update_point(right(p), (L + R) / 2 + 1, R, idx, new_value);
    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  }
  public SegmentTree(int[] _A) {
    A = _A; n = A.length;
    st = new int[4 * n];
    for (int i = 0; i < 4 * n; i++) st[i] = 0;
    build(1, 0, n - 1);
  }
  public int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }
  public int update_point(int idx, int new_value) {
    return update_point(1, 0, n - 1, idx, new_value); }
}

class RMQ {
```

```java
50    public static void main(String[] args) {
51      int[] A = new int[] { 18, 17, 13, 19, 15, 11, 20 };
52      SegmentTree st = new SegmentTree(A);
53      st.rmq(1, 3); // answer 2
54      st.rmq(0, 6); // answer =  5
55      st.update_point(5, 100); // A[5] from 11 to 100
56      st.rmq(1, 3); // 2
57      st.rmq(0, 6); // 5->2
58    }
59  }
```

1.2.   SegmentTree - lazy

```java
1   class SegmentTree {
2     private int[] st, A;
3     private int n;
4     private int left (int p) { return p << 1; }
5     private int right(int p) { return (p << 1) + 1; }
6     private void build(int p, int L, int R) {
7       if (L == R)
8         st[p] = L;
9       else {
10        build(left(p) , L, (L + R) / 2);
11        build(right(p), (L + R) / 2 + 1, R);
12        int p1 = st[left(p)], p2 = st[right(p)];
13        st[p] = (A[p1] <= A[p2]) ? p1 : p2;
14    } }
15
16    private int rmq(int p, int L, int R, int i, int j) {
17      if (i >  R || j <  L) return -1;
18      if (L >= i && R <= j) return st[p];
19      int p1 = rmq(left(p) , L, (L+R) / 2, i, j);
20      int p2 = rmq(right(p), (L+R) / 2 + 1, R, i, j);
21      if (p1 == -1) return p2;
22      if (p2 == -1) return p1;
23      return (A[p1] <= A[p2]) ? p1 : p2; }
24
25    private int update_point(int p, int L, int R, int idx, int new_value) {
26      int i = idx, j = idx;
27      if (i > R || j < L)
28        return st[p];
29      if (L == i && R == j) {
30        A[i] = new_value;
31        return st[p] = L;
32      }
33      int p1, p2;
34      p1 = update_point(left(p) , L, (L + R) / 2, idx, new_value);
35      p2 = update_point(right(p), (L + R) / 2 + 1, R, idx, new_value);
36      return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
37    }
38    public SegmentTree(int[] _A) {
39      A = _A; n = A.length;
40      st = new int[4 * n];
41      for (int i = 0; i < 4 * n; i++) st[i] = 0;
```

```java
42      build(1, 0, n - 1);
43    }
44    public int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }
45    public int update_point(int idx, int new_value) {
46      return update_point(1, 0, n - 1, idx, new_value); }
47  }
48
49  class RMQ {
50    public static void main(String[] args) {
51      int[] A = new int[] { 18, 17, 13, 19, 15, 11, 20 };
52      SegmentTree st = new SegmentTree(A);
53      st.rmq(1, 3); // answer 2
54      st.rmq(0, 6); // answer =  5
55      st.update_point(5, 100); // A[5] from 11 to 100
56      st.rmq(1, 3); // 2
57      st.rmq(0, 6); // 5->2
58    }
59  }
```

1.3.   FenwickTree - consultas por rango, update puntual

```java
1   class FenwickTree {
2     private Vector<Integer> ft;
3     private int LSOne(int S) { return (S & (-S)); }
4     public FenwickTree() {}
5     // initialization: n + 1 zeroes, ignore index 0
6     public FenwickTree(int n) {
7       ft = new Vector<Integer>();
8       for (int i = 0; i <= n; i++) ft.add(0);
9     }
10    public int rsq(int b) { // returns RSQ(1, b)
11      int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
12      return sum; }
13    public int rsq(int a, int b) { // returns RSQ(a, b)
14      return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }
15    // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
16    void adjust(int k, int v) {// note: n = ft.size() - 1
17      for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v); }
18  };
19
20  class Test {
21    public static void main(String[] args) {
22      // idx    0 1 2 3 4 5 6 7  8 9 10, no index 0!
23      FenwickTree ft = new FenwickTree(10); // ft = {-,0,0,0,0,0,0,0, 0,0,0}
24      ft.adjust(2, 1);// ft = {-,0,1,0,1,0,0,0, 1,0,0}, idx 2,4,8 => +1
25      ft.adjust(4, 1);// ft = {-,0,1,0,2,0,0,0, 2,0,0}, idx 4,8 => +1
26      ft.adjust(5, 2);// ft = {-,0,1,0,2,2,2,0, 4,0,0}, idx 5,6,8 => +2
27      ft.adjust(6, 3);// ft = {-,0,1,0,2,2,5,0, 7,0,0}, idx 6,8 => +3
28      ft.adjust(7, 2);// ft = {-,0,1,0,2,2,5,2, 9,0,0}, idx 7,8 => +2
29      ft.adjust(8, 1);// ft = {-,0,1,0,2,2,5,2,10,0,0}, idx 8 => +1
30      ft.adjust(9, 1);// ft = {-,0,1,0,2,2,5,2,10,1,1}, idx 9,10 => +1
31      System.out.printf("%d\n", ft.rsq(1, 1));  // 0 => ft[1] = 0
32      System.out.printf("%d\n", ft.rsq(1, 2));  // 1 => ft[2] = 1
33      System.out.printf("%d\n", ft.rsq(1, 6));  // 7 => ft[6] + ft[4] = 5 + 2 = 7
```

```java
34      System.out.printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11
35      System.out.printf("%d\n", ft.rsq(3, 6));  // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1
36      ft.adjust(5, 2); // update demo
37      System.out.printf("%d\n", ft.rsq(1, 10)); // now 13
38    }
39  }
```

1.4. FenwickTree, consultas puntuales, update por rango

```java
1   class FenwickTree2 {
2     private Vector<Integer> ft;
3     private int LSOne(int S) { return (S & (-S)); }
4     public FenwickTree() {}
5     // initialization: n + 1 zeroes, ignore index 0
6     public FenwickTree(int n) {
7       ft = new Vector<Integer>();
8       for (int i = 0; i <= n; i++) ft.add(0);
9     }
10    public int rsq(int b) { // returns RSQ(1, b)
11      int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
12      return sum; }
13    public int rsq(int a, int b) {                    // returns RSQ(a, b)
14      return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }
15    // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
16    void adjust(int k, int v) {  // note: n = ft.size() - 1
17      for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v); }
18    void range_adj(int i, int j, int v){
19      adjust(i, v);
20      adjust(j+1, -v);
21    }
22  }
```

2. Algoritmos

2.1. Básico

```java
1   class team implements Comparable<team> {
2     private int id, solved, penalty;
3
4     public team(int id, int solved, int penalty) {
5       this.id = id;
6       this.solved = solved;
7       this.penalty = penalty;
8     }
9
10    public int compareTo(team o) {
11      if (solved != o.solved) // can use this primary field to decide sorted order
12        return o.solved - solved;   // ICPC rule: sort by number of problem solved
13      else if (penalty != o.penalty)// solved == o.solved, but we can use
14        // secondary field to decide sorted order
15        return penalty - o.penalty;  // ICPC rule: sort by descending penalty
16      else // solved == o.solved AND penalty == o.penalty
```

```java
17        return id - o.id;  // sort based on increasing team ID
18    }
19
20    public String toString() {
21      return "id: " + id + ", solved: " + solved + ", penalty: " + penalty;
22    }
23  }
24
25  class Collection {
26    public static void main(String[] args) {
27      Vector<Integer> v = new Vector<Integer>();
28
29      v.add(10);
30      v.add(7);
31      v.add(2);
32      v.add(15);
33      v.add(4);
34
35      // sort descending with vector
36      Collections.sort(v);
37      // if we want to modify comparison function, use the overloaded method: Collections.
           sort(List list, Comparator c);
38      Collections.reverse(v);
39
40      System.out.println(v);
41      System.out.printf("================\n");
42
43      // shuffle the content again
44      Collections.shuffle(v);
45      System.out.println(v);
46      System.out.printf("================\n");
47
48      // sort ascending
49      Collections.sort(v);
50      System.out.println(v);
51      System.out.printf("================\n");
52
53      Vector<team> nus = new Vector<team>();
54      nus.add(new team(1, 1, 10));
55      nus.add(new team(2, 3, 60));
56      nus.add(new team(3, 1, 20));
57      nus.add(new team(4, 3, 60));
58
59      // without sorting, they will be ranked like this:
60      for (int i = 0; i < 4; i++)
61        System.out.println(nus.get(i));
62
63      Collections.sort(nus);              // sort using a comparison function
64      System.out.printf("================\n");
65      // after sorting using ICPC rule, they will be ranked like this:
66      for (int i = 0; i < 4; i++)
67        System.out.println(nus.get(i));
68      System.out.printf("================\n");
69
```

```
70      int pos = Collections.binarySearch(v, 7);
71      System.out.println("Trying to search for 7 in v, found at index = " + pos);
72
73      pos = Collections.binarySearch(v, 77);
74      System.out.println("Trying to search for 77 in v, found at index = " + pos); //
            output is -5 (explanation below)
75
76      /*
77      binarySearch will returns:
78        index of the search key, if it is contained in the list;
79        otherwise, (-(insertion point) - 1).
80        The insertion point is defined as the point at which the key would be inserted into
            the list:
81        the index of the first element greater than the key,
82        or list.size(), if all elements in the list are less than the specified key.
83        Note that this guarantees that the return value will be >= 0 if and only if the key
            is found.
84      */
85
86      // sometimes these two useful simple macros are used
87      System.out.printf("min(10, 7) = %d\n", Math.min(10, 7));
88      System.out.printf("max(10, 7) = %d\n", Math.max(10, 7));
89    }
90 }
```

3. Strings

3.1. KMP

```
1  class KMP{
2    char[] T, P; // T = text, P = pattern
3    int n, m; // n = length of T, m = length of P
4    int [] b; // b = back table
5
6    void naiveMatching() {
7      for (int i = 0; i < n; i++) { // try all potential starting indices
8        Boolean found = true;
9        for (int j = 0; j < m && found; j++) // use boolean flag 'found'
10         if (i + j >= n || P[j] != T[i + j]) // if mismatch found
11           found = false; // abort this, shift starting index i by +1
12       if (found) // if P[0 .. m - 1] == T[i .. i + m - 1]
13         System.out.printf("P is found at index %d in T\n", i);
14   } }
15
16   void kmpPreprocess() { // call this before calling kmpSearch()
17     int i = 0, j = -1; b[0] = -1; // starting values
18     while (i < m) { // pre-process the pattern string P
19       while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
20       i++; j++; // if same, advance both pointers
21       b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
22   } }              // in the example of P = "SEVENTY SEVEN" above
23
24   void kmpSearch() { // this is similar as kmpPreprocess(), but on string T
```

```
25     int i = 0, j = 0; // starting values
26     while (i < n) { // search through string T
27       while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
28       i++; j++; // if same, advance both pointers
29       if (j == m) { // a match found when j == m
30         System.out.printf("P is found at index %d in T\n", i - j);
31         j = b[j]; // prepare j for the next possible match
32   } } }
33
34   void run() {
35     String Tstr = "I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN";
36     String Pstr = "SEVENTY SEVEN";
37     T = new String(Tstr).toCharArray();
38     P = new String(Pstr).toCharArray();
39     n = T.length;
40     m = P.length;
41
42     System.out.println(T);
43     System.out.println(P);
44     System.out.println();
45
46     System.out.printf("Naive Mathing\n");
47     naiveMatching();
48     System.out.println();
49
50     System.out.printf("KMP\n");
51     b = new int[100010];
52     kmpPreprocess();
53     kmpSearch();
54     System.out.println();
55
56     System.out.printf("String Library\n");
57     int pos = Tstr.indexOf(Pstr);
58     while (pos != -1) {
59       System.out.printf("P is found at index %d in T\n", pos);
60       pos = Tstr.indexOf(Pstr, pos + 1);
61     }
62     System.out.println();
63   }
64
65   public static void main(String[] args){
66     new ch6_02_kmp().run();
67   }
68 }
```

3.2. Suffix Array

```
1  import java.util.*;
2
3  class SA{
4    Scanner scan;
5    char T[];                          // the input string, up to 100K characters
6    int n;                                           // the length of input string
7
```

```
 8     int[] RA, tempRA;              // rank array and temporary rank array
 9     Integer[] SA, tempSA;          // suffix array and temporary suffix array
10     int[] c;                                 // for counting/radix sort
11
12     char P[];      // the pattern string (for string matching)
13     int m;              // the length of pattern string
14
15     int[] Phi;     // for computing longest common prefix
16     int[] PLCP;
17     int[] LCP;     // LCP[i] stores the LCP between previous suffix "T + SA[i-1]" and
              current suffix "T + SA[i]"
18
19     void countingSort(int k) {
20       int i, sum, maxi = Math.max(300, n);    // up to 255 ASCII chars or length of n
21       for (i = 0; i < 100010; i++) c[i] = 0;              // clear frequency table
22       for (i = 0; i < n; i++)                 // count the frequency of each rank
23         c[i + k < n ? RA[i + k] : 0]++;
24       for (i = sum = 0; i < maxi; i++) {
25         int t = c[i]; c[i] = sum; sum += t;
26       }
27       for (i = 0; i < n; i++)                 // shuffle the suffix array if necessary
28         tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
29       for (i = 0; i < n; i++)                 // update the suffix array SA
30         SA[i] = tempSA[i];
31     }
32
33     void constructSA() {          // this version can go up to 100000 characters
34       int i, k, r;
35       for (i = 0; i < n; i++) RA[i] = T[i];                // initial rankings
36       for (i = 0; i < n; i++) SA[i] = i;         // initial SA: {0, 1, 2, ..., n-1}
37       for (k = 1; k < n; k <<= 1) {         // repeat sorting process log n times
38         countingSort(k);      // actually radix sort: sort based on the second item
39         countingSort(0);              // then (stable) sort based on the first item
40         tempRA[SA[0]] = r = 0;              // re-ranking; start from rank r = 0
41         for (i = 1; i < n; i++)                 // compare adjacent suffices
42           tempRA[SA[i]] =        // if same pair => same rank r; otherwise, increase r
43             (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
44         for (i = 0; i < n; i++)                 // update the rank array RA
45           RA[i] = tempRA[i];
46     } }
47
48     void computeLCP() {
49       int i, L;
50       Phi[SA[0]] = -1;                                    // default value
51       for (i = 1; i < n; i++)                         // compute Phi in O(n)
52         Phi[SA[i]] = SA[i-1];       // remember which suffix is behind this suffix
53       for (i = L = 0; i < n; i++) {                 // compute Permuted LCP in O(n)
54         if (Phi[i] == -1) { PLCP[i] = 0; continue; }         // special case
55         while (i + L < T.length && Phi[i] + L < T.length && T[i + L] == T[Phi[i] + L]) L++;
              // L will be increased max n times
56         PLCP[i] = L;
57         L = Math.max(L-1, 0);                         // L will be decreased max n times
58       }
59       for (i = 1; i < n; i++)                         // compute LCP in O(n)
60         LCP[i] = PLCP[SA[i]];    // put the permuted LCP back to the correct position
61     }
62
63     int strncmp(char[] a, int i, char[] b, int j, int n){
64       for (int k=0; i+k < a.length && j+k < b.length; k++){
65         if (a[i+k] != b[j+k]) return a[i+k] - b[j+k];
66       }
67       return 0;
68     }
69
70     int[] stringMatching() {                               // string matching in O(m log n)
71       int lo = 0, hi = n-1, mid = lo;                 // valid matching = [0 .. n-1]
72       while (lo < hi) {                                 // find lower bound
73         mid = (lo + hi) / 2;                                 // this is round down
74         int res = strncmp(T, SA[mid], P, 0, m);         // try to find P in suffix 'mid'
75         if (res >= 0) hi = mid;              // prune upper half (notice the >= sign)
76         else          lo = mid + 1;                 // prune lower half including mid
77       }                                 // observe '=' in "res >= 0" above
78       if (strncmp(T,SA[lo], P,0, m) != 0) return new int[]{-1, -1};         // if not found
79       int[] ans = new int[]{ lo, 0} ;
80
81       lo = 0; hi = n - 1; mid = lo;
82       while (lo < hi) {                 // if lower bound is found, find upper bound
83         mid = (lo + hi) / 2;
84         int res = strncmp(T, SA[mid], P,0, m);
85         if (res > 0) hi = mid;                                 // prune upper half
86         else          lo = mid + 1;                 // prune lower half including mid
87       }                                 // (notice the selected branch when res == 0)
88       if (strncmp(T, SA[hi], P,0, m) != 0) hi--;                 // special case
89       ans[1] = hi;
90       return ans;
91     } // return lower/upper bound as the first/second item of the pair, respectively
92
93     void LRS() {                               // print out the length and the actual LRS
94       int i, idx = 0, maxLCP = 0;
95
96       for (i = 1; i < n; i++)                                 // O(n)
97         if (LCP[i] > maxLCP) {
98           maxLCP = LCP[i];
99           idx = i;
100        }
101
102      System.out.printf("\nThe LRS is '%s' with length = %d\n\n",
103        new String(T).substring(SA[idx], maxLCP), maxLCP);
104    }
105
106    int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }
107
108    void LCS() {                               // print out the length and the actual LCS
109      int i, j, maxLCP = 0, idx = 0;
110      // not used in Java version
111      // char ans[MAX_N];
112      // strcpy(ans, "");
113
```

```
114    //System.out.printf("\nRemember, T = '%s'\nNow, enter another string P:\n", new
           String(T));
115    // T already has '.' at the back
116    P = new String("CATA").toCharArray();
117    m = P.length;
118    T = (new String(T) + new String(P) + "#").toCharArray();   // append P and '#'
119    n = T.length;                                              // update n
120    constructSA();                                             // O(n log n)
121    computeLCP();                                              // O(n)
122    System.out.printf("\nThe LCP information of 'T+P' = '%s':\n", new String(T));
123    System.out.printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
124    for (i = 0; i < n; i++)
125      System.out.printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(SA[i]),
126         new String(T, SA[i], T.length - SA[i]));
127
128    for (i = 1, maxLCP = -1; i < n; i++)
129      if (LCP[i] > maxLCP && owner(SA[i]) != owner(SA[i-1])) {  // different owner
130        maxLCP = LCP[i];
131        idx = i;
132        // not used in Java version
133        // strncpy(ans, T + SA[i], maxLCP);
134        // ans[maxLCP] = 0;
135      }
136
137    System.out.printf("\nThe LCS is '%s' with length = %d\n",
138       new String(T).substring(SA[idx], SA[idx] + maxLCP),
139       maxLCP);
140  }
141
142  void run() {
143    int MAX_N = 100010;
144    c = new int[MAX_N];
145    RA = new int[MAX_N];
146    tempRA = new int[MAX_N];
147    SA = new Integer[MAX_N];
148    tempSA = new Integer[MAX_N];
149    Phi = new int[MAX_N];
150    PLCP = new int[MAX_N];
151    LCP = new int[MAX_N];
152
153    //System.out.printf("Enter a string T below, we will compute its Suffix Array:\n");
154    T = new String("GATAGACA$").toCharArray();
155    n = T.length;
156
157    constructSA();                                             // O(n log n)
158    System.out.printf("The Suffix Array of string T = '%s' is shown below (O(n log n)
           version):\n", new String(T));
159    System.out.printf("i\tSA[i]\tSuffix\n");
160    for (int i = 0; i < n; i++)
161      System.out.printf("%2d\t%2d\t%s\n", i, SA[i], new String(T, SA[i], T.length - SA[i])
           );
162
163    computeLCP();                                             // O(n)
164
```

```
165    // LRS demo
166    LRS();          // find the longest repeated substring of the first input string
167
168    // stringMatching demo
169    //System.out.printf("\nNow, enter a string P below, we will try to find P in T:\n");
170    P = new String("A").toCharArray();
171    int[] pos = stringMatching();
172    if (pos[0] != -1 && pos[1] != -1) {
173      System.out.printf("%s is found SA [%d .. %d] of %s\n",
174         new String(P), pos[0], pos[1], new String(T));
175      System.out.printf("They are:\n");
176      for (int i = pos[0]; i <= pos[1]; i++)
177        System.out.printf("  %s\n", new String(T, SA[i], T.length - SA[i]));
178    } else System.out.printf("%s is not found in %s\n", new String(P), new String(T));
179
180    // LCS demo
181    LCS();                    // find the longest common substring between T and P
182
183    // note that the LRS and LCS demo are slightly different in Java version
184  }
185  public static void main(String[] args){
186    new ch6_04_sa().run();
187  }
188 }
```

3.3. Trie example

```
1  import java.io.*;
2  import java.util.LinkedList;
3  import java.util.Queue;
4  import java.util.StringTokenizer;
5
6  public class AutoCorrect_Jimmy {
7
8      public static class Trie {
9          private Trie[] letter = new Trie[28]; // 26 = autoComplete, 27 = parent
10         private int distance = -1; // shortest distance to this node from the root
11
12         public Trie(Trie parent) {
13             this.letter[27] = parent;
14         }
15     }
16
17     public static void main(String[] args) throws FileNotFoundException {
18         Kattio io = new Kattio(System.in, System.out);
19         int n = io.getInt(), m = io.getInt();
20         Trie root = new Trie(null);
21         for (int i = 0; i < n; i++) {
22             String word = io.getWord();
23             Trie current = root;
24             for (int j = 0; j < word.length(); j++) {
25                 int c = word.charAt(j) - 'a';
26                 if (current.letter[c] == null) {
27                     current.letter[c] = new Trie(current);
```

```
28                  }
29                  current = current.letter[c];
30              }
31              Trie end = current;
32              current = root;
33              for (int j = 0; j < word.length(); j++) {
34                  if (current.letter[26] == null) {
35                      current.letter[26] = end;
36                  }
37                  current = current.letter[word.charAt(j) - 'a'];
38              }
39              if (current.letter[26] == null) {
40                  current.letter[26] = end; // TODO: Include test case when forgetting this
                                              fails
41              }
42          }
43          Queue<Trie> q = new LinkedList<>();
44          q.add(root);
45          root.distance = 0;
46          root.letter[26] = null; // can't autocomplete if no letter typed
47          while (q.size() > 0) {
48              Trie current = q.poll();
49              for (int i = 0; i < 28; i++) {
50                  if (current.letter[i] != null && current.letter[i].distance == -1) {
51                      current.letter[i].distance = current.distance + 1;
52                      q.add(current.letter[i]);
53                  }
54              }
55          }
56
57          for (int i = 0; i < m; i++) {
58              String word = io.getWord();
59              Trie current = root;
60              int t = 0;
61              while (t < word.length() && current.letter[word.charAt(t) - 'a'] != null) {
62                  current = current.letter[word.charAt(t++) - 'a'];
63              }
64              io.println(current.distance + word.length() - t);
65          }
66          io.close();
67      }
68 }
```

4. Grafos

4.1. LCA

```
1  class LCA{
2      int MAX = 20;
3      int[][] P = new int[N][MAX]; //P[i][0] parent de i, P[root][0] = root
4      int[] level = new int[N]; //level[root] = 0
5
6      public int dist(int A, int B){
7          int C = lca(A,B);
8          lastlca = C;
9          return level[A] + level[B] - 2*level[C];
10     }
11     public int anc_dist(int A, int dist) {
12         for (int i = 0; i < MAX; i++) {
13             if (((1 << i) & dist) != 0) {
14                 A = P[A][i];
15             }
16         }
17         return A;
18     }
19     public int lca(int A, int B) {
20         if (level[A] < level[B]){
21             int aux = A;
22             A = B;
23             B = aux;
24         }
25         int dif = level[A] - level[B];
26         A = anc_dist(A, dif);
27         if (A == B){
28             return A;
29         }
30         for (int k = MAX-1; k >= 0; --k) {
31             if (P[A][k] != P[B][k]) {
32                 A = P[A][k];
33                 B = P[B][k];
34             }
35         }
36         return P[A][0];
37     }
38     public void init_lca() {
39         for (int k = 1; k < MAX; ++k)
40             for (int i = 0; i < N; ++i)
41                 P[i][k] = P[P[i][k-1]][k-1];
42     }
43     public int anc_level(int A, int l) {
44         for (int k = MAX-1; k >= 0; --k) {
45             if (level[P[A][k]] >= l)
46                 A = P[A][k];
47         }
48         return A;
49     }
50 }
```

```
1
2  class BitManipulation {
3      private static int setBit(int S, int j) { return S | (1 << j); }
4
5      private static int isOn(int S, int j) { return S & (1 << j); }
6
7      private static int clearBit(int S, int j) { return S & ~(1 << j); }
8
9      private static int toggleBit(int S, int j) { return S ^ (1 << j); }
```

```java
   private static int lowBit(int S) { return S & (-S); }

   private static int setAll(int n) { return (1 << n) - 1; }

   private static int modulo(int S, int N) { return ((S) & (N - 1)); } // returns S % N,
        where N is a power of 2

   private static int isPowerOfTwo(int S) { return (S & (S - 1)) == 0 ? 1 : 0; }

   private static int nearestPowerOfTwo(int S) { return ((int)Math.pow(2.0, (int)((Math.
        log((double)S) / Math.log(2.0)) + 0.5))); }

   private static int turnOffLastBit(int S) { return ((S) & (S - 1)); }

   private static int turnOnLastZero(int S) { return ((S) | (S + 1)); }

   private static int turnOffLastConsecutiveBits(int S) { return ((S) & (S + 1)); }

   private static int turnOnLastConsecutiveZeroes(int S) { return ((S) | (S - 1)); }

   private static void printSet(int vS) {  // in binary representation
     System.out.printf("S = %2d = ", vS);
     Stack<Integer> st = new Stack<Integer>();
     while (vS > 0) {
       st.push(vS % 2);
       vS /= 2;
     }
     while (!st.empty()) { // to reverse the print order
       System.out.printf("%d", st.peek());
       st.pop();
     }
     System.out.printf("\n");
   }

   public static void main(String[] args) {
     int S, T;

     System.out.printf("1. Representation (all indexing are 0-based and counted from right
        )\n");
     S = 34; printSet(S);
     System.out.printf("\n");

     System.out.printf("2. Multiply S by 2, then divide S by 4 (2x2), then by 2\n");
     S = 34; printSet(S);
     S = S << 1; printSet(S);
     S = S >> 2; printSet(S);
     S = S >> 1; printSet(S);
     System.out.printf("\n");

     System.out.printf("3. Set/turn on the 3-th item of the set\n");
     S = 34; printSet(S);
     S = setBit(S, 3); printSet(S);
     System.out.printf("\n");

     System.out.printf("4. Check if the 3-th and then 2-nd item of the set is on?\n");
     S = 42; printSet(S);
     T = isOn(S, 3); System.out.printf("T = %d, %s\n", T, T != 0 ? "ON" : "OFF");
     T = isOn(S, 2); System.out.printf("T = %d, %s\n", T, T != 0 ? "ON" : "OFF");
     System.out.printf("\n");

     System.out.printf("5. Clear/turn off the 1-st item of the set\n");
     S = 42; printSet(S);
     S = clearBit(S, 1); printSet(S);
     System.out.printf("\n");

     System.out.printf("6. Toggle the 2-nd item and then 3-rd item of the set\n");
     S = 40; printSet(S);
     S = toggleBit(S, 2); printSet(S);
     S = toggleBit(S, 3); printSet(S);
     System.out.printf("\n");

     System.out.printf("7. Check the first bit from right that is on\n");
     S = 40; printSet(S);
     T = lowBit(S); System.out.printf("T = %d (this is always a power of 2)\n", T);
     S = 52; printSet(S);
     T = lowBit(S); System.out.printf("T = %d (this is always a power of 2)\n", T);
     System.out.printf("\n");

     System.out.printf("8. Turn on all bits in a set of size n = 6\n");
     S = setAll(6); printSet(S);
     System.out.printf("\n");

     System.out.printf("9. Other tricks (not shown in the book)\n");
     System.out.printf("8 %c 4 = %d\n", '%', modulo(8, 4));
     System.out.printf("7 %c 4 = %d\n", '%', modulo(7, 4));
     System.out.printf("6 %c 4 = %d\n", '%', modulo(6, 4));
     System.out.printf("5 %c 4 = %d\n", '%', modulo(5, 4));
     System.out.printf("is %d power of two? %d\n", 9, isPowerOfTwo(9));
     System.out.printf("is %d power of two? %d\n", 8, isPowerOfTwo(8));
     System.out.printf("is %d power of two? %d\n", 7, isPowerOfTwo(7));
     for (int i = 0; i <= 16; i++)
       System.out.printf("Nearest power of two of %d is %d\n", i, nearestPowerOfTwo(i));
     System.out.printf("S = %d, turn off last bit in S, S = %d\n", 40, turnOffLastBit(40))
        ;
     System.out.printf("S = %d, turn on last zero in S, S = %d\n", 41, turnOnLastZero(41))
        ;
     System.out.printf("S = %d, turn off last consectuve bits in S, S = %d\n", 39,
        turnOffLastConsecutiveBits(39));
     System.out.printf("S = %d, turn on last consecutive zeroes in S, S = %d\n", 36,
        turnOnLastConsecutiveZeroes(36));
   }
}
```