Índice

1. Estructuras 1
1.1. SegmentTree 1
1.2. SegmentTree - lazy 2
1.3. FenwickTree - consultas por rango, update puntual 2
1.4. FenwickTree, consultas puntuales, update por rango 3

2. Algoritmos 3
2.1. Básico 3

3. Strings 4
3.1. KMP 4

4. Geometría 5

5. Matemática 5

6. Grafos 5
6.1. DFS 5
6.2. BFS 5
6.3. LCA 5

7. Flow 5

8. Otros 5

1. Estructuras

1.1. SegmentTree

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

1.2. SegmentTree - lazy

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

1.3. FenwickTree - consultas por rango, update puntual

```java
class FenwickTree {
  private Vector<Integer> ft;
  private int LSOne(int S) { return (S & (-S)); }
  public FenwickTree() {}
  // initialization: n + 1 zeroes, ignore index 0
  public FenwickTree(int n) {
    ft = new Vector<Integer>();
    for (int i = 0; i <= n; i++) ft.add(0);
  }
  public int rsq(int b) { // returns RSQ(1, b)
    int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
    return sum; }
  public int rsq(int a, int b) { // returns RSQ(a, b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }
  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {// note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v); }
};

class Test {
  public static void main(String[] args) {
    // idx   0 1 2 3 4 5 6 7  8 9 10, no index 0!
    FenwickTree ft = new FenwickTree(10); // ft = {-,0,0,0,0,0,0,0, 0,0,0}
    ft.adjust(2, 1);// ft = {-,0,1,0,1,0,0,0, 1,0,0}, idx 2,4,8 => +1
    ft.adjust(4, 1);// ft = {-,0,1,0,2,0,0,0, 2,0,0}, idx 4,8 => +1
    ft.adjust(5, 2);// ft = {-,0,1,0,2,2,2,0, 4,0,0}, idx 5,6,8 => +2
    ft.adjust(6, 3);// ft = {-,0,1,0,2,2,5,0, 7,0,0}, idx 6,8 => +3
    ft.adjust(7, 2);// ft = {-,0,1,0,2,2,5,2, 9,0,0}, idx 7,8 => +2
    ft.adjust(8, 1);// ft = {-,0,1,0,2,2,5,2,10,0,0}, idx 8 => +1
    ft.adjust(9, 1);// ft = {-,0,1,0,2,2,5,2,10,1,1}, idx 9,10 => +1
    System.out.printf("%d\n", ft.rsq(1, 1));  // 0 => ft[1] = 0
    System.out.printf("%d\n", ft.rsq(1, 2));  // 1 => ft[2] = 1
    System.out.printf("%d\n", ft.rsq(1, 6));  // 7 => ft[6] + ft[4] = 5 + 2 = 7
    System.out.printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11
    System.out.printf("%d\n", ft.rsq(3, 6));  // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1
    ft.adjust(5, 2); // update demo
    System.out.printf("%d\n", ft.rsq(1, 10)); // now 13
  }
}
```

1.4. FenwickTree, consultas puntuales, update por rango

```
class FenwickTree2 {
  private Vector<Integer> ft;
  private int LSOne(int S) { return (S & (-S)); }
  public FenwickTree() {}
  // initialization: n + 1 zeroes, ignore index 0
  public FenwickTree(int n) {
    ft = new Vector<Integer>();
    for (int i = 0; i <= n; i++) ft.add(0);
  }
  public int rsq(int b) { // returns RSQ(1, b)
    int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
    return sum; }
  public int rsq(int a, int b) {                       // returns RSQ(a, b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }
  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {  // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v); }
  void range_adj(int i, int j, int v){
    adjust(i, v);
    adjust(j+1, -v);
  }
}
```

2. Algoritmos

2.1. Básico

```
class team implements Comparable<team> {
  private int id, solved, penalty;

  public team(int id, int solved, int penalty) {
    this.id = id;
    this.solved = solved;
    this.penalty = penalty;
  }

  public int compareTo(team o) {
    if (solved != o.solved) // can use this primary field to decide sorted order
      return o.solved - solved;   // ICPC rule: sort by number of problem solved
    else if (penalty != o.penalty)// solved == o.solved, but we can use
      // secondary field to decide sorted order
      return penalty - o.penalty;  // ICPC rule: sort by descending penalty
    else // solved == o.solved AND penalty == o.penalty
      return id - o.id;  // sort based on increasing team ID
  }

  public String toString() {
    return "id: " + id + ", solved: " + solved + ", penalty: " + penalty;
  }
}

class Collection {
  public static void main(String[] args) {
    Vector<Integer> v = new Vector<Integer>();

    v.add(10);
    v.add(7);
    v.add(2);
    v.add(15);
    v.add(4);

    // sort descending with vector
    Collections.sort(v);
    // if we want to modify comparison function, use the overloaded method: Collections.sort(List list, Comparator c);
    Collections.reverse(v);

    System.out.println(v);
    System.out.printf("==================\n");

    // shuffle the content again
    Collections.shuffle(v);
    System.out.println(v);
    System.out.printf("==================\n");

    // sort ascending
    Collections.sort(v);
    System.out.println(v);
    System.out.printf("==================\n");

    Vector<team> nus = new Vector<team>();
    nus.add(new team(1, 1, 10));
    nus.add(new team(2, 3, 60));
    nus.add(new team(3, 1, 20));
    nus.add(new team(4, 3, 60));

    // without sorting, they will be ranked like this:
    for (int i = 0; i < 4; i++)
      System.out.println(nus.get(i));

    Collections.sort(nus);               // sort using a comparison function
    System.out.printf("==================\n");
    // after sorting using ICPC rule, they will be ranked like this:
    for (int i = 0; i < 4; i++)
      System.out.println(nus.get(i));
    System.out.printf("==================\n");

```

```java
    int pos = Collections.binarySearch(v, 7);
    System.out.println("Trying to search for 7 in v, found at index = " + pos);

    pos = Collections.binarySearch(v, 77);
    System.out.println("Trying to search for 77 in v, found at index = " + pos); // output is -5 (explanation below)

    /*
    binarySearch will returns:
      index of the search key, if it is contained in the list;
      otherwise, (-(insertion point) - 1).
      The insertion point is defined as the point at which the key would be inserted into the list:
      the index of the first element greater than the key,
      or list.size(), if all elements in the list are less than the specified key.
      Note that this guarantees that the return value will be >= 0 if and only if the key is found.
    */

    // sometimes these two useful simple macros are used
    System.out.printf("min(10, 7) = %d\n", Math.min(10, 7));
    System.out.printf("max(10, 7) = %d\n", Math.max(10, 7));
  }
}
```

3. Strings

3.1. KMP

```java
class KMP{
  char[] T, P; // T = text, P = pattern
  int n, m; // n = length of T, m = length of P
  int [] b; // b = back table

  void naiveMatching() {
    for (int i = 0; i < n; i++) { // try all potential starting indices
      Boolean found = true;
      for (int j = 0; j < m && found; j++) // use boolean flag 'found'
        if (i + j >= n || P[j] != T[i + j]) // if mismatch found
          found = false; // abort this, shift starting index i by +1
      if (found) // if P[0 .. m - 1] == T[i .. i + m - 1]
        System.out.printf("P is found at index %d in T\n", i);
  } }

  void kmpPreprocess() { // call this before calling kmpSearch()
    int i = 0, j = -1; b[0] = -1; // starting values
    while (i < m) { // pre-process the pattern string P
      while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
      i++; j++; // if same, advance both pointers
      b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
  } }           // in the example of P = "SEVENTY SEVEN" above

  void kmpSearch() { // this is similar as kmpPreprocess(), but on string T
    int i = 0, j = 0; // starting values
    while (i < n) { // search through string T
      while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
      i++; j++; // if same, advance both pointers
      if (j == m) { // a match found when j == m
        System.out.printf("P is found at index %d in T\n", i - j);
        j = b[j]; // prepare j for the next possible match
  } } }

  void run() {
    String Tstr = "I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN";
    String Pstr = "SEVENTY SEVEN";
    T = new String(Tstr).toCharArray();
    P = new String(Pstr).toCharArray();
    n = T.length;
    m = P.length;

    System.out.println(T);
    System.out.println(P);
    System.out.println();

    System.out.printf("Naive Mathing\n");
    naiveMatching();
    System.out.println();

    System.out.printf("KMP\n");
    b = new int[100010];
    kmpPreprocess();
    kmpSearch();
    System.out.println();

    System.out.printf("String Library\n");
    int pos = Tstr.indexOf(Pstr);
    while (pos != -1) {
      System.out.printf("P is found at index %d in T\n", pos);
      pos = Tstr.indexOf(Pstr, pos + 1);
    }
    System.out.println();
  }

  public static void main(String[] args){
    new ch6_02_kmp().run();
  }
}
```

3.2. Suffix Array

```java
import java.util.*;

class SA{
  Scanner scan;
  char T[];                           // the input string, up to 100K characters
  int n;                                                // the length of input string

```

```
int[] RA, tempRA;             // rank array and temporary rank array
Integer[] SA, tempSA;         // suffix array and temporary suffix array
int[] c;                                        // for counting/radix sort

char P[];     // the pattern string (for string matching)
int m;             // the length of pattern string

int[] Phi;    // for computing longest common prefix
int[] PLCP;
int[] LCP;    // LCP[i] stores the LCP between previous suffix "T + SA[i-1]" and
     current suffix "T + SA[i]"

void countingSort(int k) {
  int i, sum, maxi = Math.max(300, n);   // up to 255 ASCII chars or length of n
  for (i = 0; i < 100010; i++) c[i] = 0;                 // clear frequency table
  for (i = 0; i < n; i++)                     // count the frequency of each rank
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++)                  // shuffle the suffix array if necessary
    tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)                             // update the suffix array SA
    SA[i] = tempSA[i];
}

void constructSA() {               // this version can go up to 100000 characters
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];                       // initial rankings
  for (i = 0; i < n; i++) SA[i] = i;          // initial SA: {0, 1, 2, ..., n-1}
  for (k = 1; k < n; k <<= 1) {            // repeat sorting process log n times
    countingSort(k);       // actually radix sort: sort based on the second item
    countingSort(0);               // then (stable) sort based on the first item
    tempRA[SA[0]] = r = 0;                  // re-ranking; start from rank r = 0
    for (i = 1; i < n; i++)                         // compare adjacent suffixes
      tempRA[SA[i]] =      // if same pair => same rank r; otherwise, increase r
        (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
    for (i = 0; i < n; i++)                          // update the rank array RA
      RA[i] = tempRA[i];
} }

void computeLCP() {
  int i, L;
  Phi[SA[0]] = -1;                                               // default value
  for (i = 1; i < n; i++)                                 // compute Phi in O(n)
    Phi[SA[i]] = SA[i-1];          // remember which suffix is behind this suffix
  for (i = L = 0; i < n; i++) {                // compute Permuted LCP in O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }                 // special case
    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
    PLCP[i] = L;
    L = Math.max(L-1, 0);                        // L will be decreased max n times
  }
  for (i = 1; i < n; i++)                                 // compute LCP in O(n)
    LCP[i] = PLCP[SA[i]];   // put the permuted LCP back to the correct position
}

int strncmp(char[] a, int i, char[] b, int j, int n){
  for (int k=0; i+k < a.length && j+k < b.length; k++){
    if (a[i+k] != b[j+k]) return a[i+k] - b[j+k];
  }
  return 0;
}

int[] stringMatching() {                           // string matching in O(m log n)
  int lo = 0, hi = n-1, mid = lo;                   // valid matching = [0 .. n-1]
  while (lo < hi) {                                            // find lower bound
    mid = (lo + hi) / 2;                                     // this is round down
    int res = strncmp(T, SA[mid], P, 0, m);       // try to find P in suffix 'mid'
    if (res >= 0) hi = mid;             // prune upper half (notice the >= sign)
    else          lo = mid + 1;                // prune lower half including mid
  }                                           // observe '=' in "res >= 0" above
  if (strncmp(T,SA[lo], P,0, m) != 0) return new int[]{-1, -1};        // if not found
  int[] ans = new int[]{ lo, 0} ;

  lo = 0; hi = n - 1; mid = lo;
  while (lo < hi) {                  // if lower bound is found, find upper bound
    mid = (lo + hi) / 2;
    int res = strncmp(T, SA[mid], P,0, m);
    if (res > 0) hi = mid;                                     // prune upper half
    else         lo = mid + 1;                 // prune lower half including mid
  }                              // (notice the selected branch when res == 0)
  if (strncmp(T, SA[hi], P,0, m) != 0) hi--;                      // special case
  ans[1] = hi;
  return ans;
} // return lower/upper bound as the first/second item of the pair, respectively

void LRS() {                             // print out the length and the actual LRS
  int i, idx = 0, maxLCP = 0;

  for (i = 1; i < n; i++)                                                 // O(n)
    if (LCP[i] > maxLCP) {
      maxLCP = LCP[i];
      idx = i;
    }

  System.out.printf("\nThe LRS is '%s' with length = %d\n\n",
    new String(T).substring(SA[idx], maxLCP), maxLCP);
}

int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

void LCS() {                             // print out the length and the actual LCS
  int i, j, maxLCP = 0, idx = 0;
  // not used in Java version
  // char ans[MAX_N];
  // strcpy(ans, "");
```

```
    //System.out.printf("\nRemember, T = '%s'\nNow, enter another string P:\n", new String(T));
    // T already has '.' at the back
    P = new String("CATA").toCharArray();
    m = P.length;
    T = (new String(T) + new String(P) + "#").toCharArray();   // append P and '#'
    n = T.length;                                                      // update n
    constructSA();                                                  // O(n log n)
    computeLCP();                                                         // O(n)
    System.out.printf("\nThe LCP information of 'T+P' = '%s':\n", new String(T));
    System.out.printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
    for (i = 0; i < n; i++)
      System.out.printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(SA[i]),
        new String(T, SA[i], T.length - SA[i]));

    for (i = 1, maxLCP = -1; i < n; i++)
      if (LCP[i] > maxLCP && owner(SA[i]) != owner(SA[i-1])) {  // different owner
        maxLCP = LCP[i];
        idx = i;
        // not used in Java version
        // strncpy(ans, T + SA[i], maxLCP);
        // ans[maxLCP] = 0;
      }

    System.out.printf("\nThe LCS is '%s' with length = %d\n",
      new String(T).substring(SA[idx], SA[idx] + maxLCP),
      maxLCP);
  }

  void run() {
    int MAX_N = 100010;
    c = new int[MAX_N];
    RA = new int[MAX_N];
    tempRA = new int[MAX_N];
    SA = new Integer[MAX_N];
    tempSA = new Integer[MAX_N];
    Phi = new int[MAX_N];
    PLCP = new int[MAX_N];
    LCP = new int[MAX_N];

    //System.out.printf("Enter a string T below, we will compute its Suffix Array:\n");
    T = new String("GATAGACA$").toCharArray();
    n = T.length;

    constructSA();                                                  // O(n log n)
    System.out.printf("The Suffix Array of string T = '%s' is shown below (O(n log n) version):\n", new String(T));
    System.out.printf("i\tSA[i]\tSuffix\n");
    for (int i = 0; i < n; i++)
     System.out.printf("%2d\t%2d\t%s\n", i, SA[i], new String(T, SA[i], T.length - SA[i]));

    computeLCP();                                                         // O(n)

    // LRS demo
    LRS();        // find the longest repeated substring of the first input string

    // stringMatching demo
    //System.out.printf("\nNow, enter a string P below, we will try to find P in T:\n");
    P = new String("A").toCharArray();
    int[] pos = stringMatching();
    if (pos[0] != -1 && pos[1] != -1) {
      System.out.printf("%s is found SA [%d .. %d] of %s\n",
        new String(P), pos[0], pos[1], new String(T));
      System.out.printf("They are:\n");
      for (int i = pos[0]; i <= pos[1]; i++)
        System.out.printf("  %s\n", new String(T, SA[i], T.length - SA[i]));
    } else System.out.printf("%s is not found in %s\n", new String(P), new String(T));

    // LCS demo
    LCS();                    // find the longest common substring between T and P

    // note that the LRS and LCS demo are slightly different in Java version
  }
  public static void main(String[] args){
    new ch6_04_sa().run();
  }
}
```

4. Geometría

5. Matemática

6. Grafos

6.1. DFS

6.2. BFS

6.3. LCA

```
class LCA{
  int MAX = 20;
  int[][] P = new int[N][MAX]; //P[i][0] parent de i, P[root][0] = root
  int[] level = new int[N]; //level[root] = 0

  public int dist(int A, int B){
    int C = lca(A,B);
    lastlca = C;
    return level[A] + level[B] - 2*level[C];
  }
  public int anc_dist(int A, int dist) {
    for (int i = 0; i < MAX; i++) {
      if (((1 << i) & dist) != 0) {
        A = P[A][i];
      }
    }
    return A;
  }
  public int lca(int A, int B) {
    if (level[A] < level[B]){
      int aux = A;
      A = B;
      B = aux;
    }
    int dif = level[A] - level[B];
    A = anc_dist(A, dif);
    if (A == B){
      return A;
    }
    for (int k = MAX-1; k >= 0; --k) {
      if (P[A][k] != P[B][k]) {
        A = P[A][k];
        B = P[B][k];
      }
    }
    return P[A][0];
  }
  public void init_lca() {
    for (int k = 1; k < MAX; ++k)
      for (int i = 0; i < N; ++i)
        P[i][k] = P[P[i][k-1]][k-1];
  }
  public int anc_level(int A, int l) {
    for (int k = MAX-1; k >= 0; --k) {
      if (level[P[A][k]] >= l)
        A = P[A][k];
    }
    return A;
  }
}
```

7. Flow

8. Otros

```

class BitManipulation {
  private static int setBit(int S, int j) { return S | (1 << j); }

  private static int isOn(int S, int j) { return S & (1 << j); }

  private static int clearBit(int S, int j) { return S & ~(1 << j); }

  private static int toggleBit(int S, int j) { return S ^ (1 << j); }

  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));
  }
}
```