International Olympiad in Informatics 2011
2011年7月22-29日, タイ, パタヤ
Day 2 競技課題
日本語版

PARROTS

バージョン: 1.3

オウム (Parrots)

Yanee は鳥マニアである. Yanee は「鳥を用いた IP 通信 (IP over Avian Carriers: IPoAC)」を知った. そこで賢いオウムを訓練し, 何羽かのオウムを使って遠く離れた場所までメッセージを送ることにした.

Yanee の夢は, オウムを使って遠い遠い島にメッセージ **M** を送ることである. メッセージ **M** は, **N** 個の 0 以上 255 以下の整数からなる数列である (ただし, 同じ整数が含まれる可能性がある). Yanee はよく訓練された **K** 羽のオウムを飼育している. それぞれのオウムは 0 以上 **R** 以下の整数を 1 つだけ覚えることができる. ただし, オウム同士の区別はできないので, 個体を区別して整数を覚えさせることはできない.

最初, Yanee はメッセージを送る安直な方法として, 「カゴからオウムを 1 羽ずつ取り出し, それぞれのオウムにメッセージを表す数列の整数を順番に覚えさせて放つ」 という方法を考えた. しかし, 残念ながらこの方法のままではうまく送ることはできない. なぜなら, 最終的にすべてのオウムは目的地に到着するが, 放った順番に到着するとは限らないからである. この方法では, Yanee はメッセージに含まれるすべての整数を復元できるが, それらの整数を正しく並び替えることはできない.

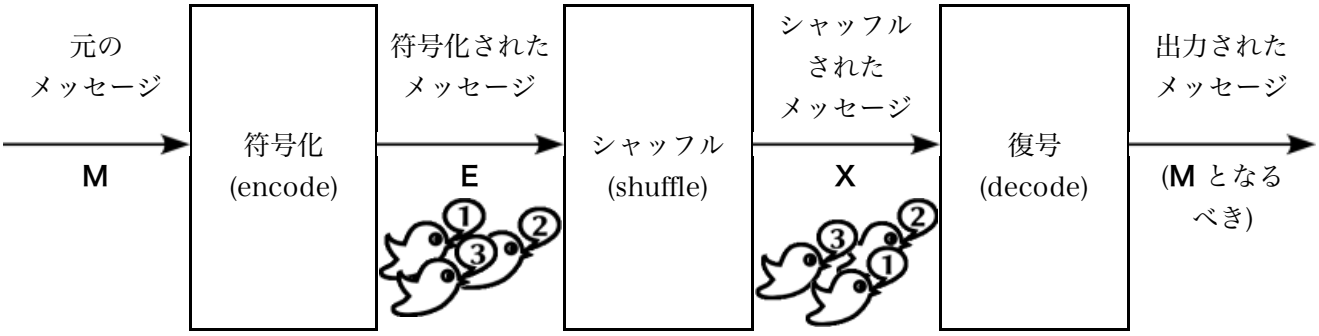
Yanee の夢を叶えるため, より良い方法が必要であり, Yanee にはあなたの助けが必要である. メッセージ **M** が与えられた時, Yanee は, 先の方法と同様にオウムを 1 羽ずつ放つことを考えている. あなたは以下に示される 2 つの操作をするプログラムをそれぞれ実装する:

1. プログラムはメッセージ **M** を読み, オウムに覚えさせる **K** 個以下の 0 以上 **R** 以下の整数からなる数列に符号化する.
2. 到着したオウムが覚えていた 0 以上 **R** 以下の整数からなる数列を読み, 元のメッセージ **M** を復元する.

すべてのオウムは必ず目的地に到着し, それぞれのオウムは覚えさせられた整数を, 到着したときも正しく覚えていると仮定してよい. ただし, オウムが到着する順序は変化する可能性があることに注意せよ. また, Yanee はオウムを **K** 羽しか飼育していないため, あなたは **K** 個以下の 0 以上 **R** 以下の整数からなる数列に符号化しなければならない.

課題 (Your task)

全体の流れは次の図に示される通りである.



以下の 2 つのプロシージャーを実装せよ. ただし, 1 つは送信用 (encode) として用いられ, もう 1 つは受信

用 (decode) として用いられる.

- 次のパラメータを持つプロシージャー **encode(N, M)** を実装せよ:
  - **N** — メッセージの長さ.
  - **M** — メッセージの内容を表す **N** 個の整数からなる 1 次元配列. **0 ≦ i < N** なる各 **i** について, **0 ≦ M[i] ≦ 255** である.

  このプロシージャーはメッセージ **M** をオウムに覚えさせるため, **0** 以上 **R** 以下の整数からなる数列に符号化しなければならない. このプロシージャーはオウムに覚えさせる各整数 **a** に対して, **send(a)** を呼び出す必要がある.
- 次のパラメータを持つプロシージャー **decode(N, L, X)** を実装せよ:
  - **N** — 元のメッセージの長さ.
  - **L** — 受け取ったメッセージの長さ (放たれたオウムの数).
  - **X** — **L** 個の整数からなる 1 次元配列. ただし, **X** はプロシージャー **encode** で生成された整数の組み合わせと一致するが, 順序が異なる場合がある.

  このプロシージャーは, 元のメッセージを復元しなければならない. このプロシージャーは, 元のメッセージの各整数 **b** に対して, **output(b)** を正しい順序で呼び出す必要がある.

  注意: **R** と **K** は入力のパラメータとして与えられない. 後述する小課題の説明を参照せよ.

  小課題を正しく解くためには, プロシージャーは以下の条件を満たす必要がある:

  - プロシージャー **encode** から送られるすべての整数は, 小課題で指定された範囲に収まらなければならない.
  - プロシージャー **encode** が呼び出すプロシージャー **send** は, 小課題で指定される **K** の回数を超えて呼び出されてはならない. ただし, **K** はメッセージの長さ **N** に依存する.
  - プロシージャー **decode** は元のメッセージ **M** を正しく復元し, **M[0], M[1], ..., M[N−1]** とそれぞれ一致する **b** に対してプロシージャー **output(b)** を, この順番にちょうど **N** 回呼び出さなければならない.

  最後の小課題では, あなたの点数は符号化されたメッセージと元のメッセージの長さ比によって決まる.

例 (Example)

N = 3 で,

M = 10 30 20

の場合を考える.

プロシージャー **encode(N,M)** が, ある変わった方法でメッセージを整数の数列 **(7, 3, 2, 70, 15, 20, 3)** に符号化したとする. この数列を報告するためには, プロシージャー **send** を次のように呼び出さなければならない.

send(7)
send(3)
send(2)
send(70)

send(15)
send(20)
send(3)

すべてのオウムが目的地に到着して，次の数列が得られたと仮定せよ: (3, 20, 70, 15, 2, 3, 7). この場合は， N = 3, L = 7,

3
20
70
X = 15
2
3
7

としてプロシージャー decode が呼び出される.

プロシージャー decode は元のメッセージ (10, 30, 20) を生成しなければならない. プロシージャー output を次のように呼び出すことで結果を報告せよ.

output(10)
output(30)
output(20)

小課題 (Subtasks)

小課題 1 (17 点)

- N = 8 で，配列 M 内の整数はすべて 0 または 1 である.
- R = 65535. つまり, 符号化された整数は 0 以上 65535 以下である.
- K = 10 × N. つまり, プロシージャー send を 10 × N 回まで呼び出すことができる.

小課題 2 (17 点)

- 1 ≦ N ≦ 16
- R = 65535. つまり, 符号化された整数は 0 以上 65535 以下である.
- K = 10 × N. つまり, プロシージャー send を 10 × N 回まで呼び出すことができる.

小課題 3 (18 点)

- 1 ≦ N ≦ 16
- R = 255. つまり, 符号化された整数は 0 以上 255 以下である.
- K = 10 × N. つまり, プロシージャー send を 10 × N 回まで呼び出すことができる.

小課題 4 (29 点)

- 1 ≦ N ≦ 32
- R = 255. つまり, 符号化された整数は 0 以上 255 以下である.

- **K = 10 × N**. つまり, プロシージャー **send** を **10 × N** 回まで呼び出すことができる.

小課題 5 (最大で 19 点)

- **16 ≦ N ≦ 64**
- **R = 255**. つまり, 符号化された整数は **0** 以上 **255** 以下である.
- **K = 15 × N**. つまり, プロシージャー **send** を **15 × N** 回まで呼び出すことができる.
- **重要:** この小課題の得点は符号化されたメッセージと元のメッセージの長さの比によって決まる.

  この小課題のテストケース **t** において, 符号化されたメッセージの長さ L_t と元のメッセージの長さ N_t の比を $P_t = L_t/N_t$ とおく. すべての P_t の最大値を **P** とおく. この小課題のあなたの得点は次の規則によって決定される.

  - **P ≦ 5** のとき, あなたには満点である 19 点が与えられる.
  - **5 < P ≦ 6** のとき, あなたには 18 点が与えられる.
  - **6 < P ≦ 7** のとき, あなたには 17 点が与えられる.
  - **7 < P ≦ 15** のとき, あなたの得点は **1 + 2 × (15 − P)** の小数点以下を切り捨てた値である.
  - **P > 15** または 1 つでも出力が間違っている場合は, あなたの得点は 0 点である.

- **重要:** 小課題 1 から 4 までで得点が得られる解答は, それ以前の小課題でも得点が得られる. しかし, 小課題 5 では **K** の上限が大きいため, 小課題 5 で得点が得られる解答であっても, 小課題 1 から 4 までで得点が得られるとは限らない. なお, すべての小課題に対し同一の解答で満点を得ることはできる.

実装の詳細 (Implementation details)

制限 (Limits)

- 採点環境 (Grading Environment): 実際の採点環境では, あなたの提出した解答は別々に実行される 2 つのプログラム **e** と **d** にコンパイルされる. 各実行ファイルにはあなたの符号化プログラム (encoder) と復号プログラム (decoder) がリンクされるが, **e** は **encode** のみを呼び出し, **d** は **decode** のみを呼び出す.
- CPU 時間制限 (CPU time limit): プログラム **e** はプロシージャー **encode** を 50 回呼び出し, 2 秒以内に終了しなければならない. また, プログラム **d** はプロシージャー **decode** を 50 回呼び出し, 2 秒以内に終了しなければならない.
- メモリ制限 (Memory limit): 256 MB
  注意: スタックのサイズには決められた制限はない. スタックとして使用されたメモリは, メモリ総使用量に含まれる.

インターフェース (API) (Interface (API))

- 実装フォルダ (Implementation folder): `parrots/`
- 選手が実装するファイル:
  - `encoder.c` または `encoder.cpp` または `encoder.pas`
  - `decoder.c` または `decoder.cpp` または `decoder.pas`

  C/C++ プログラマへの注意: 採点プログラムのサンプルおよび実際の採点プログラムの両方において, `encoder.c[pp]` と `decoder.c[pp]` はまとめてリンクされる. したがって, 各ファイル内のすべてのグローバル変数を **static** で宣言して, 他のファイルの変数との干渉を避ける必要がある.

- 提出ファイルのインターフェース (Contestant interface):
  - `encoder.h` または `encoder.pas`
  - `decoder.h` または `decoder.pas`
- 採点プログラムのインターフェース (Grader interface):
  - `encoderlib.h` または `encoderlib.pas`
  - `decoderlib.h` または `decoderlib.pas`
- 採点プログラムのサンプル (Sample grader): `grader.c` または `grader.cpp` または `grader.pas`
  採点プログラムのサンプルは, 2 つの別々のラウンドで実行される. 各ラウンドでは, まず与えられた入力データを入力として **encode** が呼び出され, 次にあなたのプロシージャー **encode** の生成した出力を入力として **decode** が呼び出される. 第 1 ラウンドでは, 採点プログラムは符号化されたメッセージの中の整数の順序を変更しない. 第 2 ラウンドでは, 採点プログラムのサンプルは奇数番目の整数と偶数番目の整数を入れ替える. 実際の採点プログラムは符号化されたメッセージに対して様々な種類の置換を行う. プロシージャー **shuffle** (C/C++ の場合) または **Shuffle** (Pascal の場合) を変更することで, 採点プログラムのサンプルがデータを並び替える方法を変更できる.
  採点プログラムのサンプルは符号化されたデータの範囲と長さを確認する. デフォルトの設定では, 符号化されたデータが **0** 以上 **65535** 以下の範囲に入っているかどうかを確認し, 長さが **10 × N** 以下であることを確認する. この設定は, 定数 **channel_range** を (例えば 65535 から 255 に) 調整したり, **max_expansion** を (例えば 10 から 15 または 7 に) 調整することで変更できる.
- 採点プログラムの入力のサンプル (Sample grader input): `grader.in.1`, `grader.in.2`, ...
  注意: 採点プログラムのサンプルは次の書式の入力を読み込む.
  - 1 行目: **N**
  - 2 行目: **N** 個の整数からなるリスト: **M[0]**, **M[1]**, ..., **M[N−1]**.
- 採点プログラムの入力のサンプルに対して, 期待される出力: `grader.expect.1`, `grader.expect.2`, ...
  この課題において, これらのファイルはいずれも文字列 **“Correct.”** のみを含むファイルである.