Example of Ferry Loading Algorithm

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 October 15, 2010 About Programming / Algorithms / ANSI/POSIX C

Example of Ferry Loading algorithm.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

ferry_loading.c:

```
/**
 * Ferry Loading Algorithm
 * @author: Alejandro G. Carlstein R. M.
 */
#include <stdio.h>

/* Macros Functions */
#define max(a,b) \
           ({ typeof (a) _a = (a); \
              typeof (b) _b = (b); \
            _a > _b ? _a : _b; })

/* Debugging flags */
#define DBG_LV1 0
#define DBG_LV2 1
#define DBG_LV3 0

/* Constants */
#define MAX_NUM_CAR 100
#define MAX_FERRY_LEN 100

enum{ UNKNOWN = -1, LEFT = 0, RIGHT = 1,
};

/* Our dynamic programming table element.  We have the value
 * that we can store, and also a way to remember what we selected.
 */
typedef struct{
  int length;
  int prev_row;
  int prev_col;
} Table;

int cars[MAX_NUM_CAR];
int side[MAX_NUM_CAR];
int i, j;
```

```
int ferry_len;
int total_cars;

// [ROW] [COLUMN]
Table ferry[MAX_FERRY_LEN][MAX_FERRY_LEN];

/*
 *
 */
int main(int argc, char *argv[]){

  /* Read the length of the ferry */
  scanf('%d', &ferry_len);

  // Read length of each car and set side to UNKNOWN since
  // we haven't decided yet where they will go
  cars[0] = 0;
  side[0] = UNKNOWN;
  for (i = 1; scanf('%d', &cars[i]) == 1; ++i){
    side[i] = UNKNOWN;
  };
  total_cars = i;

  // Initialize the first row and the first column
  for (i = 0; i <= ferry_len; ++i)
    ferry[0][i].length = ferry[i][0].length = 0;
    ferry[0][i].prev_row = ferry[i][0].prev_col = 0;

  // Print info
  if (DBG_LV2){
    printf('\nLength of Ferry: %d \n', ferry_len);
    printf('Number of Cars: %d \n\n', total_cars);

    for (i = 0; i < total_cars; ++i)
      printf('Car[%d]: %d length\n',
             i, cars[i]);
    printf('************************************\n');
  }//end if

  // Fill table
  int result = recursive_ferry(0,0,0, ferry_len);

  if (DBG_LV2){
    printf('Result: %d length value\n',  result);
  }

  if (DBG_LV2)
    print_tables();

  // Track back foot steps
  if (DBG_LV2)  printf('tracking(%d, %d, %d)\n\n', ferry_len, ferry_len, total_cars);
  tracking(ferry_len, ferry_len, total_cars);
```

```
  // Print results
  if (DBG_LV2) printf('\n SIDE: \n');

  for (i = 1; i < total_cars; ++i)
    switch(side[i]){
      case UNKNOWN:
        if (DBG_LV2) printf('UNKNOWN side[%d]: %d\n', i, side[i]);
        break;

      case RIGHT:
        if (DBG_LV2) printf('RIGHT side[%d]: %d\n', i, side[i]);
        printf('right\n');
        break;

      case LEFT:
        if (DBG_LV2) printf('LEFT side[%d]: %d\n', i, side[i]);
        printf('left\n');
        break;

    };

  return 0;
}

/*
 *
 */
int tracking(int left, int right, int idx_car){

  if (1) printf('\ntracking(%d,%d,%d)\n', left, right, idx_car);

  if (idx_car < 0)
    return;

  if (DBG_LV2) printf('.1 ');

  if (ferry[left][right].length == 0)
    return;

  if (DBG_LV2) printf('.2 ');

  --idx_car;

  int prev_len = (left + right) - cars[idx_car];
  int prev_left = left - cars[idx_car];
  int prev_right = right - cars[idx_car];

  if (DBG_LV2) printf('prev_len: %d = (%d + %d) - %d (car[%d])\n', prev_len, left,
right, cars[idx_car], idx_car);

  if (ferry[prev_left][right].length == prev_len && prev_left >= 0){

    if (DBG_LV2) printf('ferry[prev_LEFT: %d][right: %d] MATCH [from RIGHT]\n',
```

```
prev_left, right);

    side[idx_car] = RIGHT;

    tracking (prev_left, right, idx_car);

  }else  if (ferry[left][prev_right].length == prev_len && prev_right >= 0){

    if (DBG_LV2) printf('ferry[left: %d][prev_RIGHT: %d] MATCH [from LEFT]\n', left,
prev_right);

    side[idx_car] = LEFT;

    tracking (left, prev_right, idx_car);

  }else{

    if (DBG_LV2) printf('ferry[prev_left][right] and ferry[left][prev_right] DO NOT
MATCH\n');
    tracking (left, right, idx_car);

  }//end if

  return;
}

/*
 *
 */
int recursive_ferry(int left, int right, int idx_car, int ferry_len){

  if (DBG_LV1) printf('r_f(left: %d, right: %d, idx_car: %d (car_len: %d) \n',
                      left, right, idx_car, cars[idx_car]);

  if (left > ferry_len || right > ferry_len || idx_car > total_cars)
    return 0;

  if (ferry[left][right].length > 0){
    if (DBG_LV3) printf('++Return length: %d \n', ferry[left][right].length);
    return ferry[left][right].length;
  }// end if

  // Store value in ferry
  ferry[left][right].length += (left + right);

  // Record foot steps
  int nxt_row = ferry[left][right].prev_row = left;
  int nxt_col = ferry[left][right].prev_col = right;

  //
  int rtn_left = recursive_ferry(left + cars[idx_car], right, idx_car + 1, ferry_len);
  int rtn_right = recursive_ferry(left, right + cars[idx_car], idx_car + 1, ferry_len);
```

```
  if (DBG_LV1 && (rtn_left == 0))
    printf('L>');

  if (DBG_LV1)
    printf('[%d,%d](%d | %d)', nxt_row, nxt_col, rtn_left, rtn_right);

  if (DBG_LV1 && (rtn_right == 0))
    printf('<R');

  if (DBG_LV2) printf('\n');
  // Obtain the result of the best route
  return max(rtn_left, rtn_right);
}

/*
 *
 */
int print_tables(void){

    printf('\n\nTABLE\n ');
    for (i = 0; i <= ferry_len; ++i)
      printf(' %d ',i);
    printf('\n');

    for (i = 0; i <= ferry_len; ++i){
      for (j = 0; j <= ferry_len; ++j){
        if (j == 0) printf('%d', i);
        printf (' %d ', ferry[i][j].length);
      }// end for
      printf('\n');
    }//end for
    printf('\n');

    printf('\n TABLE XY\n ');
    for (i = 0; i <= ferry_len; ++i)
      printf(' [ %d ]',i);
    printf('\n');

    for (i = 0; i <= ferry_len; ++i){
      for (j = 0; j <= ferry_len; ++j){
        if (j == 0) printf('%d ', i);
        printf('[%d,%d] ', ferry[i][j].prev_row, ferry[i][j].prev_col);
      }//end for
      printf('\n');
   }//end for
   printf('\n');

  return 0;
}
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.