HKOI Mini-Competition (I) Editorial

5 March 2016

Problem Origins

- A: Alex Tung
- B: Alex Tung (modified from POJ 1042 Gone Fishing)
- C: Steven
- D: Alex Tung (idea from (the definition of) topology)

Test case setters

- A: RB
- B: LMH
- C: Steven
- D: Anson

A: Alice and Geometry Problem

- “Stick” a square and a rectangle to form a polygon
- Condition: perimeter <= p
- Maximize area

- Topics: Math, DP

Calculating the Perimeter

- Assume we pick square i and rectangle j

- If $s_i <= b_j$, min. perimeter = $2 * (s_i + a_j + b_j)$
- If $s_i > b_j$, min. perimeter = $2 * (2 * s_i + a_j)$

Subtask 1

- For each query, try all NM combinations
- Time complexity: O(NMQ)

Subtask 2

- Observation: For a fixed rectangle, increasing square length gives increasing perimeter and area (of the resultant polygon)

- First, sort the squares by length
- For each query, consider each rectangle
- Binary search for the “best” square

- Time complexity: $O(N^2 + MQ \log N)$, if $O(N^2)$ sort is used

Subtask 3

- We have the perimeter formulas → Solve for s_i in each case!

- If $s_i <= b_j$, need $2 * (s_i + a_j + b_j) <= p$
- $s_i <= (p – 2 * a_j – 2 * b_j) / 2$

- If $s_i > b_j$, need $2 * (2 * s_i + a_j) <= p$
- $s_i <= (p – 2 * a_j) / 4$

Subtask 3

- Full solution:
  - Precompute B[x], the largest square length <= x
  - For each query, consider each rectangle
  - Get inequalities in the form "s_i <= L"
  - Choose square with length B[L], update answer

- Time complexity: O(N + MQ + max_length)

B: Buying Candies

- Bob has M dollars, wants to buy candies at $1 each
- There are N types of candies
- For the i-th type, the j-th candy that Bob buys gives $(v_i – d_i * (j – 1))$ units of happiness
  - e.g. $v_i = 9$, $d_i = 2$ → 9, 7, 5, 3, 1, -1, …
- Find the maximal units of happiness achievable

- Topics: greedy, binary search

A “Greedy” Observation

- If Bob buys the candies one by one, Bob should always select the candy which gives the highest happiness

- Buy until:
  - Bob has M candies, or
  - Buying the “best” candy does not give an increase in happiness

Subtask 1

```
ans = 0
for i from 1 to M
      best = 1
      for j from 2 to N
            if v[j] > v[best] then best = j
      if v[best] <= 0, break
      ans = ans + v[best]
      v[best] = v[best] – d[best]
```

- Time complexity: O(MN)

Subtask 2

- We need to perform “find max” faster!
- Use a more efficient data structure: heap

- Time complexity: O(M log N)

Subtask 3

- Buy the candies as described below and update answer
- Check if there is enough money left at each step

- v[] = {1, 5, 5, 8, 9}
- → {1, 5, 5, 8, **8**}
- → {1, 5, 5, **5, 5**}
- → {1, 5, **5, 5, 5**}
- → {1, **1, 1, 1, 1**}
- → {**0, 0, 0, 0, 0**}

Subtask 3

- Example 1: M = 12

- v[] = {1, 5, 5, 8, 9}, M = 12
- → {1, 5, 5, 8, **8**}, M = 11
- → {1, 5, 5, **5, 5**}, M = 5
- → {1, 5, **5, 5, 5**}, M = 5
- → {1, **3, 4, 4, 4**}, M = 0

Subtask 3

- Example 2: M = 55

- v[] = {1, 5, 5, 8, 9}, M = 55
- → {1, 5, 5, 8, **8**}, M = 54
- → {1, 5, 5, **5, 5**}, M = 48
- → {1, 5, **5, 5, 5**}, M = 48
- → {1, **1, 1, 1, 1**}, M = 32
- → {**0, 0, 0, 0, 0**}, M = 27

Subtask 3

- Time complexity: O(N log N) (requires O(N log N) sorting)

Subtask 4

- Key idea: binary search

- Binary search for what?
  - The lowest happiness value of the candies Bob buys

Subtask 4

```
L = 0, R = infinity //i.e. a big enough number
while R – L > 1
    int mid = (L + R) / 2;
    if at least M candies give happiness >= mid
        L = mid
    else
        R = mid
```

Subtask 4

- Now we have L as the lowest happiness value of the candies Bob buys
- Can find total happiness value using L

- Beware: $d_i = 0$
  - division by 0 when calculating the answer ☹

- Time complexity: O(N log(max_v))

C: Fair Patrol

- Given N integers $a_1, a_2, \ldots, a_N$
- let $f(l, r) = a_l + a_{l+1} + \ldots + a_{r-1} + a_r$
- Find the minimum of
  $\max(|f(1,x-1)-f(x,y-1)|, |f(x,y-1)-f(y,N)|, |f(1,x-1)-f(y,N)|)$
  for all $1 < x < y <= N$

Subtask 1

- N = 3
- Output max(|a_1 - a_2|, |a_2 - a_3|, |a_1 - a_3|)
- O(1)

Subtask 2

- N <= 100
- for each x and y where 1 < x < y <= N,
  calculate f(1,x-1), f(x,y-1) and f(y,N) in O(N)
  ans = min(ans, max(|f(1,x-1) - f(x,y-1)|,
  |f(x,y-1) - f(y,N)|,
  |f(1,x-1) - f(y,N)|))
- $O(N^3)$

Subtask 3

- $N <= 2000$
- Pre-compute prefix sums $s_i = a_1 + a_2 + \ldots + a_i$ for $1 <= i <= N$ in $O(N)$
- for each x and y where $1 < x < y <= N$,

  $f(1,x-1) = s_{x-1}$, $f(x,y-1) = s_{y-1} - s_{x-1}$, $f(y,N) = s_N - s_{y-1}$

  $ans = \min(ans, \max(|f(1,x-1) - f(x,y-1)|,$
  $|f(x,y-1) - f(y,N)|,$
  $|f(1,x-1) - f(y,N)|))$

- $O(N^2)$

Subtask 4

- N <= 100000
- Observation:
  Suppose we have fixed y, we don't need to try for all 1 < x < y. We only need to find the x yielding minimal |f(1,x-1) - f(x,y-1)|. The x can be found by binary search in O(logN)

Subtask 4

- Pre-compute prefix sums $s_i = a_1 + a_2 + \ldots + a_i$ for $1 <= i <= N$ in O(N)
- for $3 <= y <= N$,

  binary search for $1 < x < y$ where $|f(1,x-1) - f(x,y-1)|$ is minimum

  $f(1,x-1) = s_{x-1}$, $f(x,y-1) = s_{y-1} - s_{x-1}$, $f(y,N) = s_N - s_{y-1}$

  ans = min(ans, max($|f(1,x-1) - f(x,y-1)|$,
  $|f(x,y-1) - f(y,N)|$,
  $|f(1,x-1) - f(y,N)|$))
- O(NlogN)

Subtask 4

- That gives you 100 points, but we can further improve
- Let p_y = x where |f(1,x-1) - f(x,y-1)| is minimum for 1 < x < y <= N
- Observe that $p_3 <= p_4 <= \ldots <= p_N$

Subtask 4

- Pre-compute prefix sums $s_i = a_1 + a_2 + \ldots + a_i$ for $1 <= i <= N$ in O(N)
- set x to 2
- for $3 <= y <= N$,

  increment x until $|f(1,x-1) - f(x,y-1)|$ is minimum

  $f(1,x-1) = s_{x-1}$, $f(x,y-1) = s_{y-1} - s_{x-1}$, $f(y,N) = s_N - s_{y-1}$

  ans = min(ans, max($|f(1,x-1) - f(x,y-1)|$,
  $|f(x,y-1) - f(y,N)|$,
  $|f(1,x-1) - f(y,N)|$))
- O(N)

D: Stable Set

- A set contains N integers in [0, 65535]
- Check if it is closed under operations AND and OR

- Easy to check condition (1). In the following presentation, assume condition (1) has already been checked

- Topic: Math

Subtask 1

- Exhaust all possible (nonempty) subsets
- For each subset,
  - check their AND is in the set
  - check their OR is in the set
- Use boolean array (size 65536) to mark which number is in the set

- Time complexity: $O(N * 2^N)$

Subtask 2

- Let P(k) be “for each size-k subset, their AND and OR are in the set”
- Stable means P(1), P(2), …, P(N) are all true
- P(1) is obviously true
- If the set is stable, P(2) is true
- In fact, if P(2) is true, then the set is stable!

Subtask 2

- Why P(2) being true is sufficient for set stability?

- If P(1) and P(2) are true, P(3), …, P(N) can be proved to be true by mathematical induction

Subtask 2

- For each pair of elements, check if their AND and OR are in the set
- Time complexity: $O(N^2)$

Subtask 3

- Didn’t expect trainees to come up with a “correct” solution
- Therefore, only 10 points ☺

Subtask 3

- Correct solution 1
  - Somewhat mysterious…
  - Given a set A with numbers in [0, 65535]
  - For each bit 2^b (b = 0 … 15), calculate u[b], which is the AND of all numbers in A containing bit 2^b
  - Now you have u[0], u[1], …, u[15]
  - Check if numbers produced by OR of any sub-collection of u[i] are in A

- (WARNING: difficult math!)
- Proof of correctness (see proposition 1): http://www.maths.ed.ac.uk/~aar/papers/stong2.pdf

Subtask 3

- Correct solution 2 (Credits: RB)

```
Initialise b[i] = a[i]
for i from 1 to N
     if b[i] > 0
          for j from i + 1 to N
               check (a[i] AND a[j]) is in the set
                check (a[i] OR a[j]) is in the set
                b[i] = b[i] & (65535 – a[i]) //bit flip
```

Subtask 3

- Why is solution 2 correct?
- b[i] > 0 means that it cannot be written as “OR” of previous “selected” elements
- Selecting a[i] must add one more bit to the OR of all “selected” elements
- Hence, we “select” at most 16 elements → check at most 16N pairs