UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria **Informàtica** etsinf

Description of the solutions for the problems used at SWERC 2012

Judges and Local Problem Setters

Departament de Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

November 27, 2012

Contents

External problem setters:

- Shahriar Manzoor, Southeast University, Bangladesh
- Rujia Liu - Eryiju, China
- Derek Kisman - Translattice, USA
- Sohel Hafiz - University of Texas at San Antonio, USA
- Jane Alam Jan - Google, USA
- Mohammad Mahmudur Rahman - Mukto Software Ltd, Bangladesh
- Md. Mahbubul Hasan - BUET, Bangladesh
- Md. Towhidul Islam Talukder - SDSL, Bangladesh

Local problem setters, also judges:

- Paco Álvaro, UPV, Spain
- Carlos Martínez, UPV, Spain
- Joan Pastor, UPV, Spain
- Ximo Planells, UPV, Spain
- Mario Rodriguez, UPV, Spain
- Emilio Vivancos, UPV, Spain
- Jon Ander Gómez, UPV, Spain

Solutions accepted until freeze

- Problem A: 1 team. ENS Ulm 1
- Problem B: 36 teams. NULL Team
- Problem C: 9 teams. UPC-2
- Problem D: 1 team. UTC
- Problem E: -
- Problem F: 5 teams. I'm stuck in the ACM database.
- Problem G: 8 teams. UPC-2
- Problem H: -
- Problem I: -
- Problem J: 5 teams. Enter

Beehives

- Problem: Calculate the shortest cycle of a un-directed graph.
- Solution: Start a BFS from each node stopping when you find an already-visited node.
- See also: http://en.wikipedia.org/wiki/Girth_(graph_theory)

Bits

		Target	
		0	**1**
Source	**0**	A	B
	1	C	A
	?	D	E

- Swap B 0's and C 1's.
- Convert B 0's into 1's
- Check if it is possible generate enough 0's. (E < C).
- Change C 1's into ?'s and then into 0's.
- Convert E, F ?'s into 0's and 1's respectively.

LCMP Sum Pack

$$\sum_{\substack{1 \leq p \leq q \leq N \\ lcm(p,q)=N}} p + q$$

- We can use prime factorization of the number to determine the sum.

- Let us define:
  - $SOD(X)$ to be the sum of divisors of X
  - $NOD(X)$ to be the number of divisors of X.

- These functions can easily be calculated from the prime factorization of N.

LCMP Sum Pack

- Now we can fix some of prime factors to be in their highest power and calculate the sum of all pairs associated with all numbers having exactly those factors with their power maximized.
- Let these factors are Q_i and the other factors are R_i.
- So, for these pairs one number will have exactly the factors Q_is to have their power maximized while the other number of the pair will have at least R_is to have their power maximized.
- Of course, $P = Q \cup R$, P is the set of divisors of N.
- We can derive the following expression to calculate this:

$$\sum_{i=0}^{M} Q_i \times NOD(Q_i) \times SOD(R_{i \oplus M})$$

where M is $2^{|F|} - 1$, F is the set of prime factors.

LCMP Sum Pack

$$12 = 2^2 * 3^1$$

i	i_{binary}	$NOD(Q_i)$	$SOD(Q_i)$	Q_i
0	0000	1	1	1
1	0001	3	3	4
2	0010	2	1	3
3	0011	6	3	12

$$\sum_{i=0}^{M} NOD(Q_i) \times SOD(R_{i \oplus M}) \times Q_i$$

where M is $2^{|F|} - 1$, F is the set of prime factors.

$$\sum_{\substack{1 \le p \le q \le 12 \\ lcm(p,q)=12}} p + q = 3 + 12 + 18 + 72 = 117$$

RNA

- This problem can be solved by using Dynamic Programming with a 3 dimension table.
- Performs initial alignment for extreme blocks and erases the complementaries.
- Employs a recursive process for calculating pairings in a substring $rna_s \ldots rna_e$:
  - When rna_s and rna_e makes an AU pair, calculate recursively for (rna_{s+1}, rna_{e-1}).
  - In other case:
    - From $i = rna_{s+1}$ till $i = rna_e$, if rna_s pairs rna_i, calculate recursively for (rna_{s+1},rna_{i-1}) and for (rna_{i+1},rna_e), applying CG pairing restriction when necessary
    - Calculate recursively for (rna_{s+1},rna_e) (start base unmatched)
    - Return the highest result

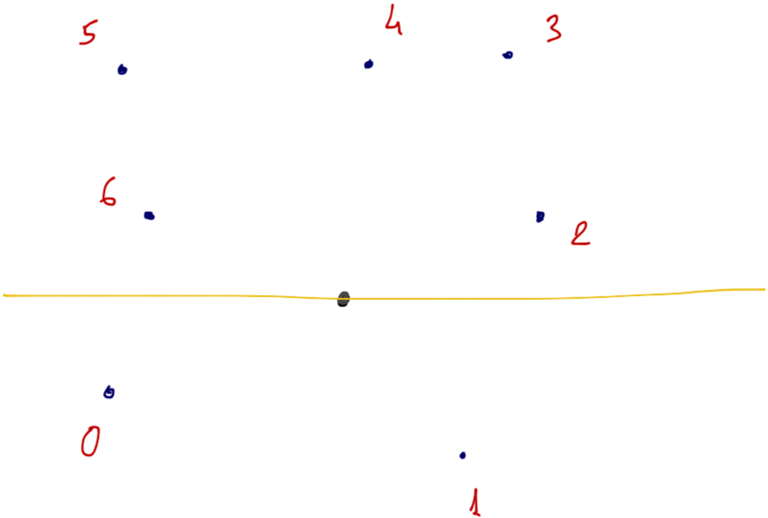
Old School Days

The trivial solution to this problem is:

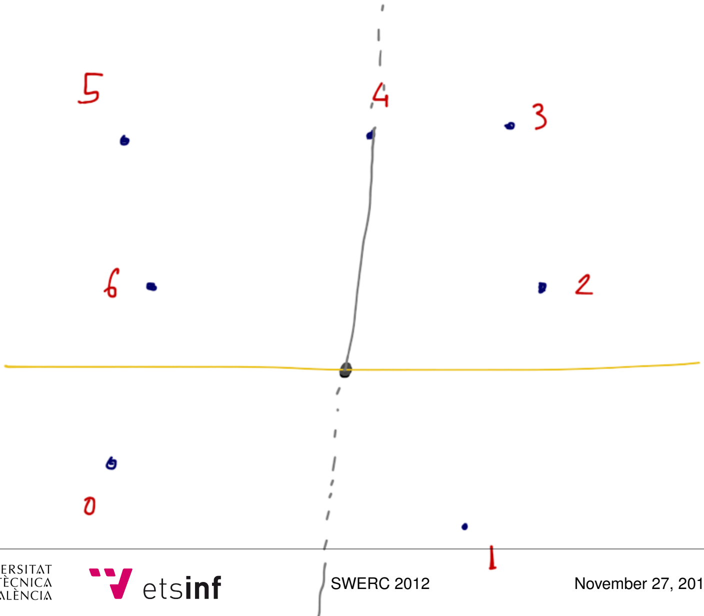
```
SUM=0;
for( int i=0; i < N; i++ )
    for( int j=i+1; j < N; j++ )
        for( int k=j+1; k < N; k++ )
            for( int l=k+1; l < N; l++ )
                if ( convex ) {
                    SUM += area( i, j, k, l );
                } else {
                    SUM += area( i, j, k, l );
                    SUM += area( i, k, j, l );
                    SUM += area( i, j, l, k );
                }
```

$$T(n) \in O(n^4)$$

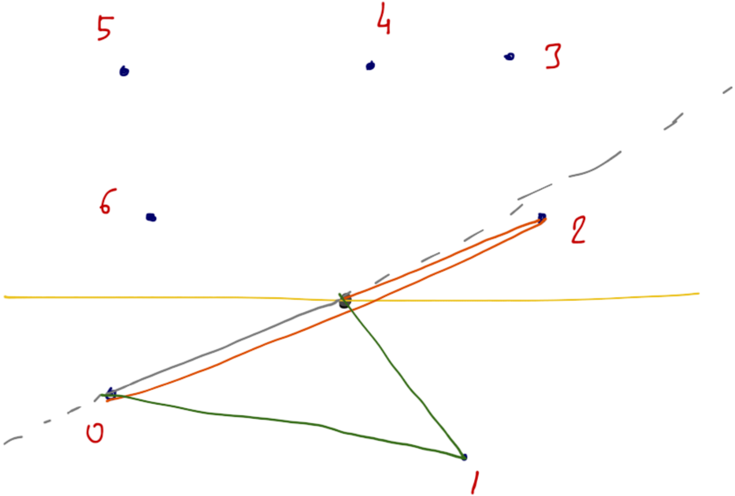
Old School Days



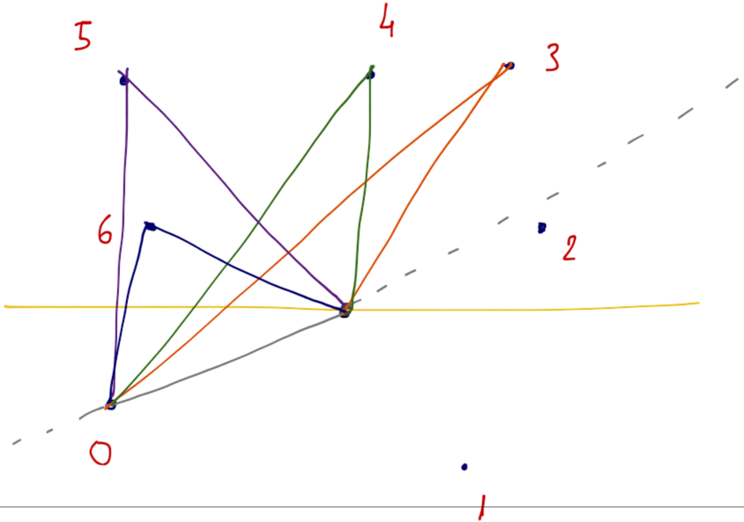
Old School Days



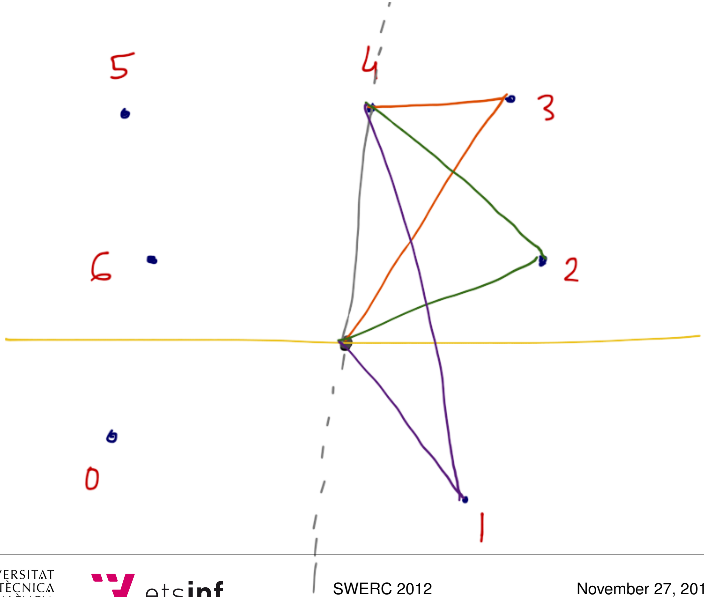
Old School Days



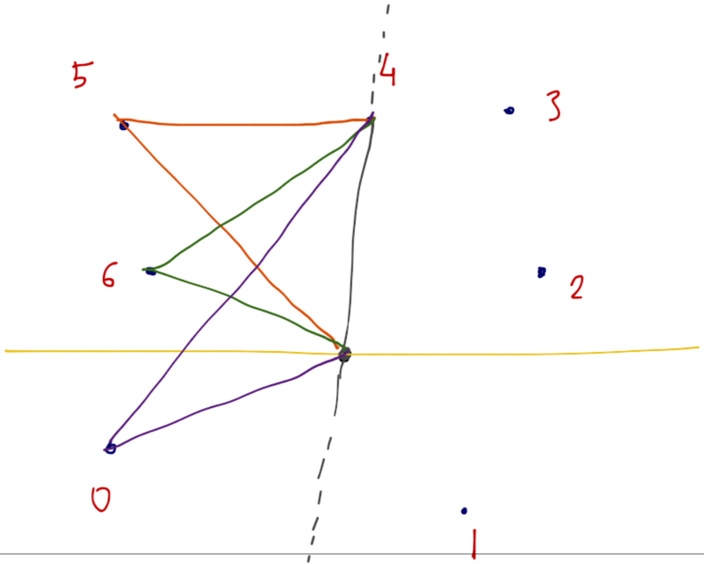
Old School Days



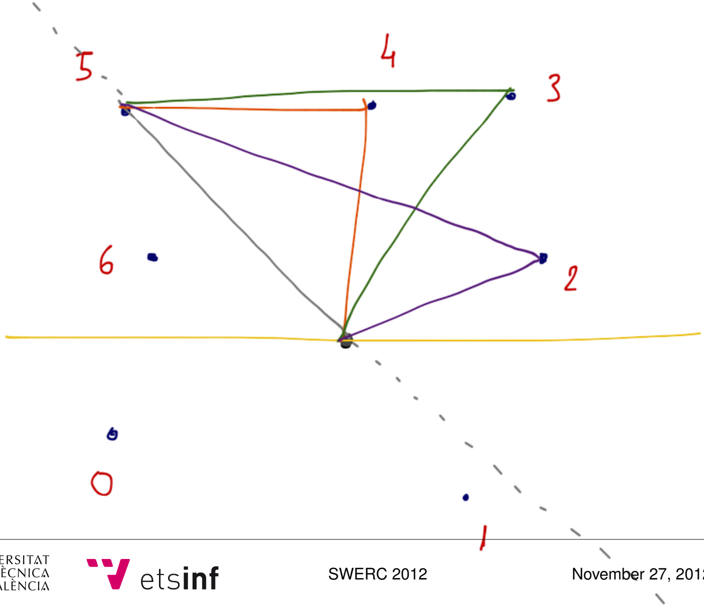
Old School Days



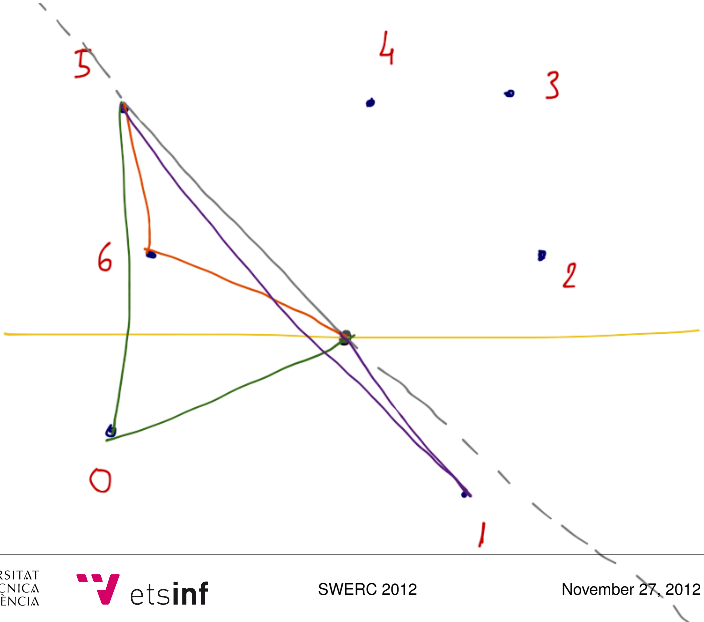
Old School Days



Old School Days



Old School Days



Old School Days

n:
- $n \cdot \log n$
- n

$$T(n) \in O(n^2 \cdot \log n)$$

Old School Days

Area of convex poligons is accumulated 4 times.

Area of concave poligons is accumulated 2 times.

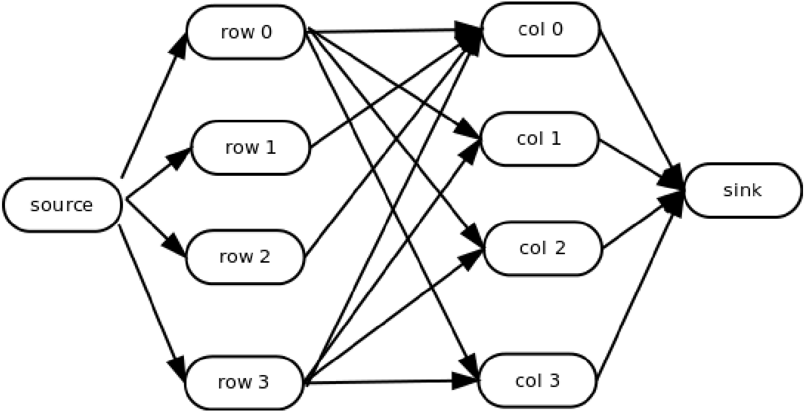
Sentry Robots

- Goal: Cover all the * using vertical or horizontal lines taking into account obstacles #
- Solution without obstacles: Maximum matching using a flow network.

* * * .	(0,0), (0,1), (0,2),
* . . .	(1,0),
* . . .	(2,0),
* * * *	(3,0), (3,1), (3,2), (3,3)

Sentry Robots

Maximum matching between rows and columns.



Sentry Robots

Solution with obstacles: Transform the input to a new grid where obstacle don't matter.

```
. * * . .
. * # * .
. # * . .
. . * . .
```

```
. * . * . . . .
. * . . . . . .
. . . . . # . .
. . . . . . . *
. . # . . . . .
. . . . . . * .
. . . . . . * .
```

Solve this new grid using maximum matching.

Sentry Robots

Equivalence between grids:

```
. . * . # . . *          . . * . . . . .
                         . . . . # . . .
                         . . . . . . . *
```

```
*                        * . .
#                        . # .
*                        . . *
```

Transform the grid first by rows and then by columns.

Sentry Robots

Probabilistic solution. I'm sorry didn't work.

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char** argv) {
    int C;
    cin >> C;
    while( C-- )
        cout << rand() % 100 << endl;

    return 0;
}


```

Water Spiders

- D is the distance in metres from calm waters to waterfall.

  P is the spider's jumping power.

- If a sequence doesn't match a recurrence relation of second order, then the sequence is complete in the input.

  $D + 1$ numbers will appear. In this case the problem is trivial.

- If a sequence matches a recurrence relation of second order, $S_n = a \cdot S_{n-1} + b \cdot S_{n-2}$, then the sequence can be represented by a minimum of four numbers. S_0, S_1, S_2, S_3 , then obtaining a and b is also trivial.

$$S_3 = a \cdot S_2 + b \cdot S_1$$
$$S_2 = a \cdot S_1 + b \cdot S_0$$

- The solution is $D - i$, where $S_i \leq P$ and $S_{i+1} > P$

Shares

- The greedy solution is not optimal.
  - Sort and take the packs according to the ratio benefit/cost
- The complete solution is so time consuming for the given limit time (5 secs).
- This problem is solved efficiently when it is considered as the discrete Knapsack problem.
- Considerations to solve this problem:
  - Calculate the profit obtained by each pack $p \in P$.
  - Delete those packs $p' \in P$ that have a negative profit or a cost greater than the available capital C. They cannot be part of the optimal solution.

Shares

- How to solve this problem:
  - Dynamic Programming:
    - Computational cost: $O(C \times P)$
    - Efficient use of memory $\rightarrow$ Memory cost: $O(2 \times C)$
  - Greatest Common Divisor (gcd):
    - Not all the prices between 0 and C must be checked.
    - Only those prices $c \in [0, C]$ that are multiple of the gcd of the cost of packs.
    - Reduce both computational and memory cost by the $gcd(\text{cost of packs})$.
    - Computational cost: $O\left(\frac{C}{gcd(\text{cost of packs})} \times P\right)$
    - Memory cost: $O\left(2 \times \frac{C}{gcd(\text{cost of packs})}\right)$

The Moon of Valencia

- This problem can be efficiently solved by using the A^* algorithm.
- The heuristic for sorting the hypotheses in the priority queue is

  $$|S^* - f()|$$

  where S^* represents the goal grade of satisfaction on arrival,

  $f() = g() - h()$ is the heuristic to be optimized,

  $g()$ is the grade of satisfaction of the current hypothesis, which can include the satisfaction of current node or not.

  $h()$ is the time required to reach the goal from the current node using Floyd-Warshall.

Count Down

- This problem can be solved by using a Breadth or depth first search.
- For each available set of numbers, all possible combinations of addition, substraction with non-negative result, multiplication, and exact integer division are tested.
- Each new operation generates a new node in the queue with the contents:
  - Available numbers.
  - Last operation performed.
  - Previous node (that with the previous operation).
- When result is achieved, nodes are retrieved from current node and operation sequence is obtained.
- The same process applies for non-exact results, but after exhaustive search the node with the closest approximation is selected.