ACM-ICPC South Western European Regional SWERC 2008

FAU Contest Team

icpc@i2.informatik.uni-erlangen.de
Friedrich-Alexander Universität Erlangen-Nürnberg

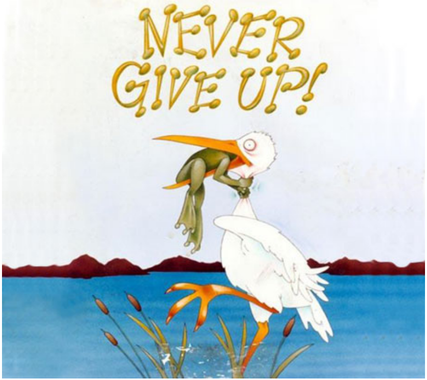
November, 23 2008

Übersicht

Problem	min. LOC	max. LOC
Bring Your Own Horse	54	180
First Knight	82	123
Postal Charges	44	103
Randomly-priced Tickets	71	138
The Game	80	248
The Merchant Guild	47	117
Toll Road	48	134
Top Secret	49	115
Transcribed Books	36	82
Wizards	66	262
$\sum$	577	1502

Bring Your Own Horse

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



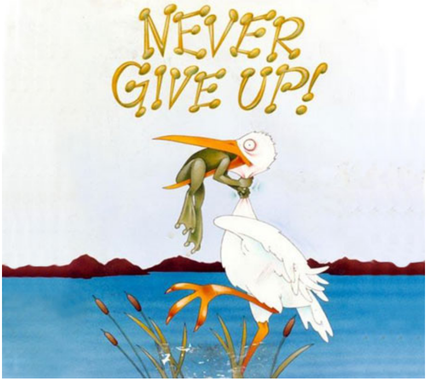
Bring Your Own Horse

For each test case:

- places and roads define an undirected graph
- find a minimum spanning tree (Kruskal or Prim)
- for each query, do a BFS or DFS in this tree
- return the longest edge found

First Knight

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



First Knight

- independent moves $\rightsquigarrow$ Markov chain
- $E_{i,j}$ = expected number of moves from (i, j) to (m, n)
- system of $m \cdot n$ linear equations with $m \cdot n$ variables $E_{i,j}$

$$
\begin{aligned}
E_{i,j} &= 1 + p_{i,j}^{(1)} E_{i+1,j} + p_{i,j}^{(2)} E_{i,j+1} + p_{i,j}^{(3)} E_{i-1,j} + p_{i,j}^{(4)} E_{i,j-1} \\
E_{m,n} &= 0
\end{aligned}
$$

First Knight

- independent moves $\rightsquigarrow$ Markov chain
- $E_{i,j}$ = expected number of moves from (i, j) to (m, n)
- system of $m \cdot n$ linear equations with $m \cdot n$ variables $E_{i,j}$

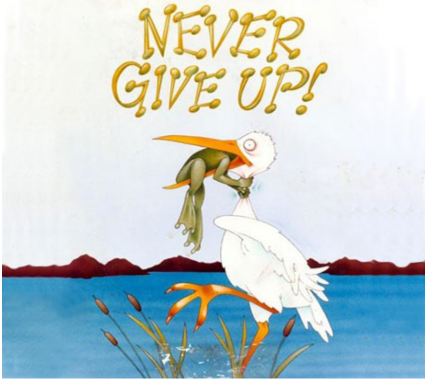
$$\begin{aligned} E_{i,j} &= 1 + p_{i,j}^{(1)} E_{i+1,j} + p_{i,j}^{(2)} E_{i,j+1} + p_{i,j}^{(3)} E_{i-1,j} + p_{i,j}^{(4)} E_{i,j-1} \\ E_{m,n} &= 0 \end{aligned}$$

- Gaussian elimination
- $p_{i,j}^{(1)} \neq 0$ or $p_{i,j}^{(2)} \neq 0 \rightsquigarrow$ no pivoting required
- block tridiagonal matrix, bandwidth $2n+1 \rightsquigarrow$ time complexity $O(mn^3)$
- iterative solutions converge too slowly

$$
\left[\begin{array}{cccc|cccc|cccc}
1 & -p_{1,1}^{(2)} & & & -p_{1,1}^{(1)} & & & & & & & \\
-p_{1,2}^{(4)} & 1 & -p_{1,2}^{(2)} & & & -p_{1,2}^{(1)} & & & & & & \\
& -p_{1,3}^{(4)} & 1 & -p_{1,3}^{(2)} & & & -p_{1,3}^{(1)} & & & & & \\
& & -p_{1,4}^{(4)} & 1 & & & & -p_{1,4}^{(1)} & & & & \\
\hline
-p_{2,1}^{(3)} & & & & 1 & -p_{2,1}^{(2)} & & & -p_{2,1}^{(1)} & & & \\
& -p_{2,2}^{(3)} & & & -p_{2,2}^{(4)} & 1 & -p_{2,2}^{(2)} & & & -p_{2,2}^{(1)} & & \\
& & -p_{2,3}^{(3)} & & & -p_{2,3}^{(4)} & 1 & -p_{2,3}^{(2)} & & & -p_{2,3}^{(1)} & \\
& & & -p_{2,4}^{(3)} & & & -p_{2,4}^{(4)} & 1 & & & & -p_{2,4}^{(1)} \\
\hline
& & & & -p_{3,1}^{(3)} & & & & 1 & -p_{3,1}^{(2)} & & \\
& & & & & -p_{3,2}^{(3)} & & & -p_{3,2}^{(4)} & 1 & -p_{3,2}^{(2)} & \\
& & & & & & -p_{3,3}^{(3)} & & & -p_{3,3}^{(4)} & 1 & -p_{3,3}^{(2)} \\
& & & & & & & 0 & & & 0 & 1
\end{array}\right]
\cdot
\left[\begin{array}{c}
E_{1,1} \\ E_{1,2} \\ E_{1,3} \\ E_{1,4} \\ \hline
E_{2,1} \\ E_{2,2} \\ E_{2,3} \\ E_{2,4} \\ \hline
E_{3,1} \\ E_{3,2} \\ E_{3,3} \\ E_{3,4}
\end{array}\right]
=
\left[\begin{array}{c}
1 \\ 1 \\ 1 \\ 1 \\ \hline
1 \\ 1 \\ 1 \\ 1 \\ \hline
1 \\ 1 \\ 1 \\ 0
\end{array}\right]
$$

Postal Charge

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –

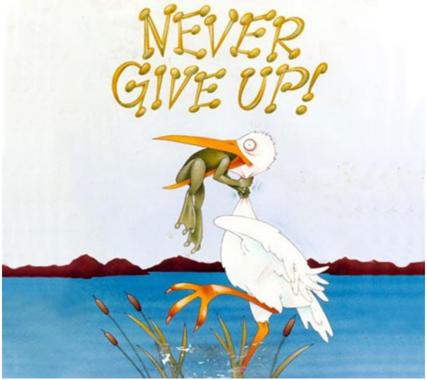


Postal Charges

- Manhattan distance!
- there are 100 rectangles, count points in each one
- store the sum of the distances of each point in a rectangle to its lower left / upper right corners
- loop over all pairs of rectangles and do some simple arithmetic

Randomly-priced Tickets

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



Randomly-priced Tickets

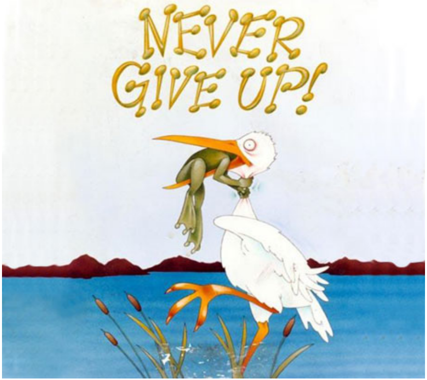
- route with least expected price $\rightsquigarrow$ shortest path in an undirected graph
- use Floyd-Warshall for all-pairs shortest paths
- probability depends only on length L of the path

Randomly-priced Tickets

- consider drawing of L numbers between 1 and R
- probability $=$ $\#$ of outcomes with sum $\leq$ budget divided by total $\#$ of possibilities
- problem equivalent to: How many ways to throw L dice such that their sum is $\leq$ budget?
- solved by dynamic programming

The Game

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –

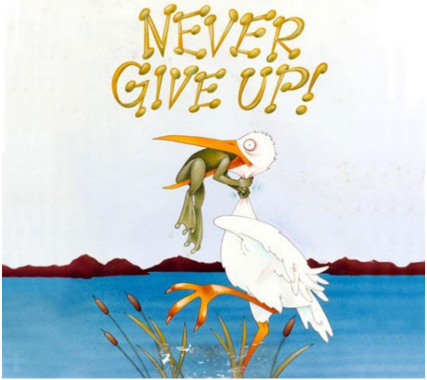


The Game

- 2-player game
- variant of well-known Kalaha
- problem description must be read carefully
- both players play optimally $\Rightarrow$
  current player moves so that the best score of the opponent after this move is minimized
- use Alpha-beta pruning to narrow the search space

The Merchant Guild

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



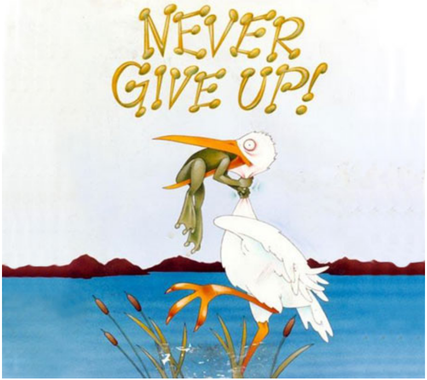
- assignment is valid iff
  ≤ 1 trader is assigned a position $\geq n$,
  ≤ 2 traders are assigned positions $\geq n-1$,
  ≤ 3 traders are assigned positions $\geq n-2$, ...
- DP with $a_{k,m} = \#$ of valid assignments to positions $k, \ldots, n$ with m available slots
- runs in $O(n^3)$

The Merchant Guild

$$
\begin{aligned}
s_k &= \#\text{ of local traders with position} \leq k \\
r_{k,m} &= k - s_{k-1} + m \\
b_{k,m} &= m + 1 - s_{k-1} + s_{k-2} \\
a_{k,m} &= \sum_{i=0}^{b_{k,m}} \binom{r_{k,m}}{b_{k,m} - i} a_{k-1,i}
\end{aligned}
$$

Toll Road

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –

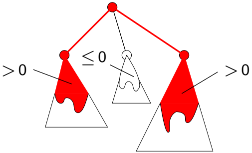


Toll Road

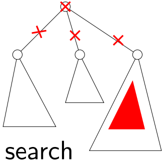
- map of towns and roads is a tree
- edges have weights
- find subtree S with maximal sum of edge weights
- generalisation of ‘maximal subsegment sum’

Toll Road

- map of towns and roads is a tree
- edges have weights
- find subtree S with maximal sum of edge weights
- generalisation of 'maximal subsegment sum'



- choose arbitrary root to 'hang up' tree
- root in S $\rightsquigarrow$ collect positive upper parts from all child trees
  or not $\rightsquigarrow$ take best solution among all child trees
- linear time complexity by divide-and-conquer
- arrange nodes in bottom-up order by breadth- or depth-first search



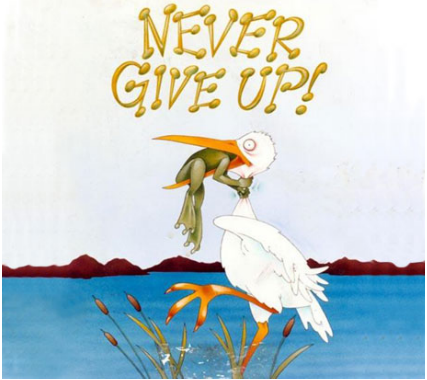
Toll Road

```
#include <iostream>
#include <list>
using namespace std;

typedef pair<int,int> pi;
list<pi> adj[1<<18];
int mss;

int dfs(int x, int y) {
  int p = 0;
  for ( ; !adj[x].empty() ; adj[x].pop_back())
    if (adj[x].back().first != y)
      p += max(0,adj[x].back().second +
                 dfs(adj[x].back().first, x));
  mss = max(mss,p);
  return p;
}
```

```
int main() {
  int n, a, b, p;
  while (cin >> n && n) {
    while (n--) {
      cin >> a >> b >> p;
      adj[a].push_back(pi(b,p));
      adj[b].push_back(pi(a,p));
    }
    mss = 0;
    dfs(b, b);
    cout << mss << endl;
  }
  return 0;
}
```

Top Secret

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



Top Secret

- decrypted numbers on a ring
- encryption is simple: for each number:
  add L times the number on the left and R times the number on the right (repeat this step S times)

Top Secret

- decrypted numbers on a ring
- encryption is simple: for each number:
  add L times the number on the left and R times the number on the right (repeat this step S times)
- can be accelerated by using matrix representation and repeated squaring

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 1 & R & 0 & L \\ L & 1 & R & 0 \\ 0 & L & 1 & R \\ R & 0 & L & 1 \end{pmatrix}^S \cdot \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

Top Secret

- matrix multiplication is still too slow for $N = 1000$
- matrix is circulant,
  product of circulant matrices is circulant
- $\leadsto$ speed up matrix multiplication:

$$\begin{pmatrix} 1 & R & 0 & L \\ L & 1 & R & 0 \\ 0 & L & 1 & R \\ R & 0 & L & 1 \end{pmatrix} \leadsto \begin{pmatrix} 1 & R & 0 & L \end{pmatrix}$$

- $O(N^2)$ space $\leadsto$ $O(N)$ space
  $O(N^3)$ time $\leadsto$ $O(N^2)$ time

Transcribed Books

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –

Transcribed Books

- notice: $N | \sum_{i=1}^{9} a_i - a_{10}$ for each serial number

Transcribed Books

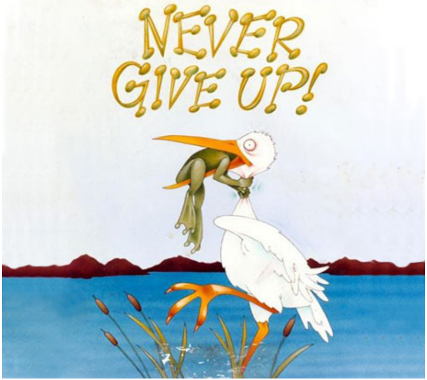
- notice: $N | \sum_{i=1}^{9} a_i - a_{10}$ for each serial number
- calculate the gcd of all $\sum_{i=1}^{9} a_i - a_{10}$

Transcribed Books

- notice: $N | \sum_{i=1}^{9} a_i - a_{10}$ for each serial number
- calculate the gcd of all $\sum_{i=1}^{9} a_i - a_{10}$
- if the gcd is 0, or the gcd is 1, or any a_{10} is larger than the calculated gcd, output "`impossible`"
- else output the gcd as N

Wizards

- # Submissions: 0 :(
- # Accepted: 0:(
- First Team: –
- Time: –



Wizards

- find out, if the given polynomial has roots with multiplicity greater than 1

Wizards

- find out, if the given polynomial has roots with multiplicity greater than 1
- consider the factorisation of the polynomial $P = (x - x_1)(x - x_2)\dots(x - x_n)$

Wizards

- find out, if the given polynomial has roots with multiplicity greater than 1
- consider the factorisation of the polynomial $P = (x - x_1)(x - x_2)\dots(x - x_n)$
- a root with multiplicity > 1 appears multiple times in this factorisation

Wizards

- find out, if the given polynomial has roots with multiplicity greater than 1
- consider the factorisation of the polynomial $P = (x - x_1)(x - x_2)\ldots(x - x_n)$
- a root with multiplicity > 1 appears multiple times in this factorisation
- polynomials cannot be factored analytically, but...

Wizards

- look at the derivative of P:

$$\begin{aligned} P' &= (x - x_1)' \cdot (x - x_2) \cdots (x - x_n) \\ &+ (x - x_1) \cdot (x - x_2)' \cdots (x - x_n) \\ &+ \cdots \\ &+ (x - x_1) \cdot (x - x_2) \cdots (x - x_n)' \end{aligned}$$

- a root with multiplicity > 1 appears in every term of this sum, hence is a root of P',

Wizards

- look at the derivative of P:

$$\begin{aligned} P' &= (x - x_1)' \cdot (x - x_2) \cdots (x - x_n) \\ &+ (x - x_1) \cdot (x - x_2)' \cdots (x - x_n) \\ &+ \cdots \\ &+ (x - x_1) \cdot (x - x_2) \cdots (x - x_n)' \end{aligned}$$

- a root with multiplicity > 1 appears in every term of this sum, hence is a root of P',
- hence it is a root of the polynomial $\gcd(P,P')$
- $\gcd(P,P')$ is constant if P has only simple roots

Wizards

- get polynomial gcd by Euclidean algorithm in the ring of polynomials, using polynomial long division
- watch out that the coefficients don't get too big – divide them by their gcd
- use rational coefficients (reduce the fractions!), or integer coefficients (multiply the polynomial by the least common multiple of the denominators)

Wizards

- get polynomial gcd by Euclidean algorithm in the ring of polynomials, using polynomial long division
- watch out that the coefficients don't get too big – divide them by their gcd
- use rational coefficients (reduce the fractions!), or integer coefficients (multiply the polynomial by the least common multiple of the denominators)
- or: solve numerically using Bairstow's method
- or: determinant of the Sylvester matrix $\neq 0$

Abendgestaltung

Kanapee?