Central European Olympiad in Informatics
Brno, July 1 - 7, 2007

NEWSLETTER

A Whole Day Trip To Lednice

The first whole-day trip led the participants to Lednice, a marvelous small town in South Moravia, known for its castle surrounded by a beautiful park with romantic lakes. In 1996 Lednice was inscribed on the UNESCO World Heritage List (together with the twin manor of Valtice) as "an exceptional example of the designed landscape that evolved in the Enlightenment and afterwards under the care of a single family." It contains a palace and the largest park in the country, which covers 200 km^2.

The history of the Lednice castle is closely related to the House of Liechtenstein, utilizing it for representation purposes since 18th century. The palace of Lednice began its life as a Renaissance villa; in the 17th century it became a summer residence of the ruling Princes of Liechtenstein. The estate house - designed and furbished by baroque architects Johann Bernhard Fischer von Erlach, Domenico Martinelli, and Anton Johan Ospel - proclaimed rural luxury on the grandest

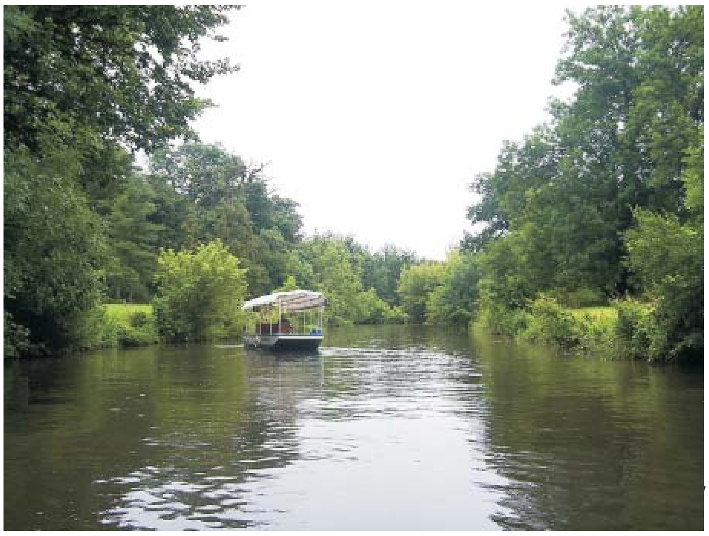
Fig. 2: Boats cruising

scale. The Neo-Gothic reconstruction of 1846 - 1858, designed by Georg Wingelmüller, formed todays outfit of the castle. The castle is also famous because of its glass house built in 1843 - 1845, according to the project by P. H. Desvignes, leaning to the manor on the eastern side.

A French garden and its vast provincial park, one of the largest in the Czech Republic, surround the castle together with an English park, forming a large green area and hosting

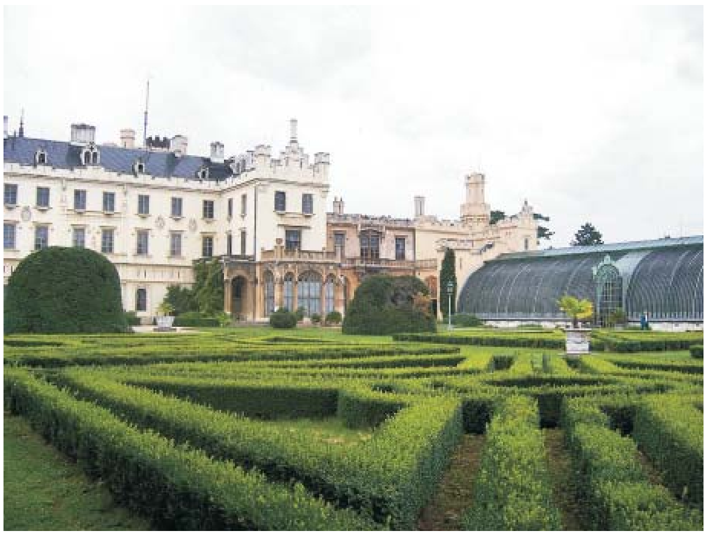
Fig. 1: Lednice Castle

about 600 tree species. The park in Lednice is also known because of noteworthy decorative constructions within it, such as the Temple of the Three Graces, the Temple of Apollon and several other structures.

Not far from the Lednice manor, on the left bank of the Dyje river, a wonderful structure, called Janův Hrad, was founded after 1801. The project was drafted by the architect, Josef Hardmuth. The exterior of this structure, originally conceived as a hunting lodge, imitates a Medieval ruin, in accordance with the then touch for romance.

The trip started in the castle and continued on boats cruising on the water channels in the park to minaret which is an another architectonic monument, constructed based on the design of Josef Hardmuth in 1797 - 1804.

A cut-and-thrust show together with an archery practice closed the cultural program. After the lunch, the participants returned to Brno.

Algorithm Hints for Competition Tasks

Treasury

This task is one of the easy ones. The office of the royal treasurer has a structure similar to the ministry of paperwork reduction from the first competition day; the only difference is that the number of subordinates of a clerk is now unlimited.

Let us also follow the notation of the ministry problem by referring to a clerk and all his subordinates as to a department with the clerk with no supervisors within the department being its head.

Let us first focus on computing the maximum number of pairs that can be formed by the clerks. Using a dynamic programming approach, we will com-pute these numbers for all departments within the office. We will actually compute two numbers: the maximum number N_+ of pairs when the head of the department can be in a pair and the maximum number N_- of pairs that be formed avoiding the head.

The numbers N_+ and N_- will be computed in the direction from the clerks with no subordinates towards the royal treasurer: clerks of single-clerk departments can form no pairs (in either of the two considered ways). In the general case, the value of N_- for the department led by a clerk X is the sum of the numbers N_+ corresponding to the departments led by the direct subordinates of X. The number N_+ for the department led by X is N_-+1 if the numbers N_+ and N_- are the same for one of the departments led by the subordinates of X, and N_+ for X is N_- (for X), otherwise. This approach solves one part of the task.

Let us now focus on the other part which asks for determining the number of ways in which the N_+ pairs of clerks can be formed. Again, we compute for each department the number M_- of ways in which the clerks can form N_- pairs avoiding the head and the number M_+ of ways in which they can form N_+ pairs when the head can be paired. Both the number M_+ and M_- are equal to one for one-clerk departments. The number M_- for the department led by X is the product of the numbers M_+ for the departments led by direct subordinates of X. If $N_+=N_-$, the number M_+ is the sum of M_- and the products of M_+ for the departments led by the subordinates such that in each product one of M_+ is replaced with M_-. Finally, if $N_+=N_-+1$, M_+ for the department led by X is the sum of the products of M_- for the departments led by the subordinates of X such that M_+ corresponding to the department led by a subordinate X' can be replaced with M_- in the product if N_+ and N_- are the same for the department led by X'.

This approach straightforwardly leads to a program that allows to compute the results in time $\mathcal{O}(N^2)$ where N is the number of clerks. A more careful implementation allowing divisions of numbers can improve the running time to $\mathcal{O}(N)$.

Since the numbers M_+ and M_- can be very large, it is necessary to implement long-number arithmetic. Because of the limits on data and time limit for this task, an $\mathcal{O}(N^2)$-solution is sufficient and thus this solution (which needs implementing only additions and multiplications of long numbers) is good enough. Finally, it remains to find an upper bound on the numbers M_+ and M_-: since there are at most 1000 pairs supervisor-subordinate, the numbers do not exceed $2^{1000} \approx 10^{300}$.

Necklaces

The task is to create an implementation of a queue that enables insertion and removal of elements from both ends, and additionally, it can be copied quickly. Of course, the standard implementations using double-linked lists or arrays fail, as copying requires linear time and space.

Another well-known way to implement a queue is using two stacks, F and R (realized by linked lists). Let us show how to perform the operations at the front of the queue, the operations at the rear of the queue are performed analogously, with the role of the stacks F and R swapped: when we insert to the front of the queue, we push the element to F. When we are removing from the front of the queue, and F is not empty, we pop the element off F. Thus we need to ensure that F is never empty (unless the queue is empty). To ensure this, we force the following invariant: the stack R is at most twice as long as F, and vice versa. If this invariant is violated by an operation, say $|F|<2|R|$, we need to move some elements from R to F, preferably in such a way that the lengths of both new stacks are the same (we call this operation **balancing**). Thus, we perform the following actions:

```
len := (|F| + |R|) div 2;
newR := take (R, len);
newF := F + reverse (drop (R, len));
```

Here, `take (L, n)` returns list consisting of first n elements of L and `drop (L, n)` returns list of all elements of

Necklaces (continuation)

L except for the first *n* ones. The function **`reverse (L)`** returns the reversed list *L* – the list is copied, so *L* remains unchanged.

Let us analyze this queue: ignoring the balancing, both insertion and removal of an element takes constant time. Balancing may take linear time, but note that after balancing, we need to perform a linear number of operations in constant time before another balancing is necessary, thus on average balancing takes only constant time per operation as well. And finally, note that we may perform all the operations in such a way that we do not modify the existing lists. Thus, copying the queue can be done in constant time simply by copying the pointers to *F* and *R*.

Although this solution scores *100* points, note that there is a small flaw in the previous analysis. What might happen is that we take an unbalanced queue *Q* and perform a single insertion that causes balancing, taking linear time.Then we copy the same queue *Q* and perform the same operation, again forcing balancing in linear time. We may repeat this process arbitrarily many times, thus causing the time complexity to be quadratic. To fix this problem, instead of computing **`newF`** and **`newR`** immediately during balancing, we can remember only the expressions that define them, and execute the operations gradually (every time we insert or remove an element from a queue, we perform a few steps of the **`newF`** and **`newR`** computations). This way, we can ensure that every operation takes exactly a constant time. However, both the technical details of the implementation and the idea of the solution is fairly nontrivial, so we decided to award full score even to the slightly worse solution described in the previous paragraph.

Airport Show

We examine all pairs of positions *A* and *B* in the sequences of runway reservations and releases. For each such pair, we need to determine the following pieces of information:

- is the pair of positions reachable in some schedule?
- does it cause the performances to block if it is reached?

Observe that the positions *(A, B)* are reachable in the following cases:

- *(A-1,B)* is reachable and the *A*-th operation of the first sequence is a release of a runway, or a reservation of a runway that is not reserved by the second performance
- *(A,B-1)* is reachable and the *B*-th operation of the second sequence is a release of a runway, or a reservation of a runway that is not reserved by the first performance.

The position *(A, B)* will block the airport if the *A+1*-th operation of the first sequence is a reservation of a runway that is reserved by the second performance and the *B+1*-th operation of the second sequence is a reservation of a runway that is reserved by the first performance.

Note also that from the sets of runways reserved by the performances in point *(A-1, B)*, we can get the sets of reserved runways in point *(A, B)* in constant time. Using these observations and the dynamic programming approach, we are able to compute the required information for all points in time $\mathcal{O}(N^2)$.

We also need to write out the sequence of actions that blocks the airport. One needs to be a bit careful when recording this information, so that we do not run out of memory; however, we only need to remember whether the position *(A, B)* is reached from *(A-1, B)* or from *(A, B-1)*, i.e., one bit of information for each pair.

Fig. 3: Competition Day

Fig. 4: Competition Day

CEOI 2007 Results

			Day 1 tasks			Day 2 tasks			
		Name	Ministry	Nasty	Sail	Airport	Necklace	Treasury	Total
Gold medalists									
1.	DEU1	Daniel Grunwald	100	100	0	100	80	95	**475**
2.	POL4	Tomasz Kulczyński	100	100	20	20	100	100	**440**
Silver medalists									
3.	HUN1	András Eisenberger	80	70	20	80	90	70	**410**
4.	DEU2	Ludwig Schmidt	100	90	10	50	80	70	**400**
5.	CZK3	Josef Pihera	50	100	10	70	100	64	**394**
6.	DEU4	Martin Maas	100	50	10	100	50	64	**374**
7.	DEU3	Benito van der Zander	0	100	10	90	70	100	**370**
Bronze medalists									
8.	POL1	Marcin Andrychowicz	100	100	0	100	20	43	**363**
9.	POL3	Jakub Kallas	100	70	30	0	100	58	**358**
10.	ROM3	Victor Rusu	30	100	10	100	10	100	**350**
11.	CRO2	Goran Žužić	100	100	0	20	20	100	**340**
12.	SVK1	Vladimír Boža	100	100	20	30	20	68	**338**
13.	HUN4	Gergely Nagy	50	50	10	100	50	70	**330**
14.	CRO4	Relja Medić	40	50	20	80	70	53	**313**
15.	CRO1	Igor Čanadi	40	100	10	70	20	70	**310**
Other contestants									
16.	POL2	Marcin Kurczych	100	80	0	10	20	73	**283**
17.	SVK4	Jozef Jirásek	40	100	10	0	20	76	**246**
18.	HUN3	Balázs Szalkai	40	40	20	10	60	70	**240**
19.	ROM2	Cosmin Gheorghe	40	50	0	50	0	97	**237**
20.	SVK2	Peter Ondrúška	100	50	0	0	0	70	**220**
21.	CRO3	Domagoj Kusalić	40	60	0	20	50	42	**212**
22.	CZK4	Roman Smrž	60	100	0	30	20	0	**210**
23.	HUN2	Tamás Peregi	20	30	10	70	0	70	**200**
24.	ROM4	Stefan-Alexandru Filip	40	50	10	0	20	67	**187**
25.	ROM1	Andrei Grigorean	50	50	0	10	0	70	**180**
26.	SVK3	Michal Danilák	60	40	0	20	20	37	**177**
	CZK6	Lukáš Lánský	40	90	0	0	0	46	**176**
27.	CZK2	Miroslav Klimoš	60	10	20	30	0	43	**163**
28.	CZK1	Pavel Klavík	0	50	20	0	20	69	**159**
	CZK7	Libor Peltan	0	20	0	20	70	19	**129**
	BRN1	Ondřej Bouda	20	30	0	0	0	40	**90**
	CZK8	Libor Plucnar	20	20	0	0	0	46	**86**
	CZK5	Jakub Kaplan	0	30	0	40	0	2	**72**
	BRN2	Marek Bryša	0	0	0	0	10	35	**45**