Analysis: 2006B
Remember the A La Mode!

HISTORY:
- v. 1.2: 2007.02.09, AI - drobne poprawki w wygladzie
- v. 1.1: 2007.02.07, JP - dostosowanie opracowania do wymagan systemu SINOL
- v. 1.0: XXXX.XX.XX, TB,MD - powstanie opracowania

dokument systemu SINOL 1.6

1 Introduction

Let us define the problem in the graph language. Let V be the set of all cake slices, and W a set of all ice cream scoops. We will build a bipartite graph with nodes $V \cup W$. Nodes $v \in V$ and $w \in W$ are connected, if given cake-ice cream combination is possible. Weight of each edge is equal to the price of the combination.

A *matching* in a graph is a subset of edges, in which no pair of edges has a common node. A *perfect matching* is a matching, which covers all the nodes. A perfect matching in our bipartite graph is related to a solution, in which all cake slices and ice cream scoops are assigned. We know that at least one such solution exists.

Weight of matching M (denoted as $\sigma(M)$) is a sum of weights of edges belonging to the matching.

2 Optimal solution

In our problem, we have to find both smallest and greatest weight of perfect matching in a bipartite graph. This is well know problem, called *the assignment problem*. There are many algorithms to solve it. The fastest known algorithms work in $O(n^3)$ time, for example the *Hungarian algorithm* or the *successive shortest path algorithm (SSP)*. Our solution is based on the latter one. For simplicity, we will implement it in $O(n^4)$ time.

Definition 1 *Let $G = (V, W, E, M)$ be a bipartite graph with node sets V and W, edges E and a matching M.*

- **an augmenting path** *in graph G is a path starting in unmatched node $v \in V$, ending in unmatched node $w \in W$, whose first edge does not belong to M, the second belongs, the third does not and so on,*
- **the weight of augmenting path**, $\sigma(\mathcal{P})$ *is a sum of weights of its edges belonging to M minus a sum of weights of edges not belonging to M.*

An augmenting path can be used to augment the matching M in a following way:

- each edge belonging to M is removed from M,
- each edge not belonging to M is added to M.

Using the path in this way we obtain new matching M', which has one more edge than M. The weight of M', $\sigma(M')$ is equal to $\sigma(M) + \sigma(\mathcal{P})$.

The SSP algorithm is based on following theorem:

Theorem 1 *Let M be a matching in a bipartite graph $G = (V, W, E, M)$, which has the smallest weight among all matchings with k edges. Let $\mathcal{P}$ be the shortest augmenting path among all possible augmenting paths. New matching M', created by augmenting M by path $\mathcal{P}$ has the smallest weight among all matchings with $k + 1$ edges.*

The SSP algorithm:

```
Start with empty matching $M$
Repeat:
   Find the shortest augmenting path
   If a path is found, use it to augment $M$
   If no path found:  STOP
```

In order to find shortest augmenting paths, we will slightly modify the graph:

- non matched edges are directed from V to W,
- matched edges are directed from W to V and they have opposite weight,
- we add a start node s and edges from s to each unmatched node $v \in V$.

In this graph we will search for the shortest path from s to any unmatched node $w \in W$. The graph has no negative cycles. If they were, we would use them like an augmenting path and we would have a matching with the same number of edges but with lower weight. This is not possible. So we can use Bellmann-Ford algorithm to find the shortest path.

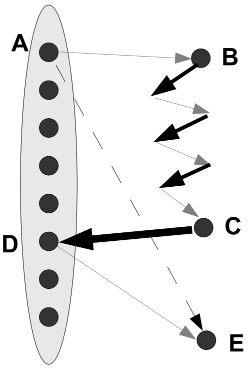
The SSP algorithm has the following feature, which will be used in construction of the algorithm.

Proposition 1 *Successive paths in SSP algorithm have non decreasing weight.*

In our problem, the set of nodes can have up to 10000 elements. It is too many to use $O(n^3)$ algorithm or even $O(n^4)$. To create fastest algorithm we will use the fact that there are no more than 50 pie types and no more than 50 ice cream types.

Let *group* be a set of all nodes of the same type. Is it possible, that the shortest augmenting path goes more than once through a group?

On the picture below we can see how a path goes twice through a group. The path enters the group in node A for the first time and then comes back in node D. Edges belonging to the matching are thicker. Let $|XY|$ denote the weight of path from X to Y. Nodes A and D belongs to the same group, so $|AB| = |DB|$ and $|AE| = |DE|$. If the path $A \to B \to \ldots \to C \to D \to E$ was shorter than $A \to E$, the path $A \to B \to \ldots \to C \to D$ would have negative weight. The weight of this path is equal to the weight of cycle $D \to B \to \ldots \to C \to D$. We know that this graph has no negative cycles, so the path $A \to B \to \ldots \to C \to D$ has non-negative weight. So the shortest path can go from A directly E.



Thus we can consider only paths, which goes no more than once through each group. So we can search for the shortest path in a *compressed* graph, in which nodes from each group are collapsed into one.

We will use following data structures:

- $matched$ – two dimensional array, $matched[i, j]$ means how many pie slices from group i are matched with ice cream scoops from group j,
- mV, mW – vectors, $mV[i]$ means how many pie slices from group i are matched, similarly mW.

When a shortest path is found in compressed graph, we can use it to augment the matching by one edge in non-compressed graph. However, we know that the next shortest path will not have lower weight, so we can use the path more than once if it is possible. In the algorithm, when the path is found, we first go through it to figure out how many times it can be used. Then we go through it for the second time and we augment the matching.

The algorithm counts the minimum profit. It can be easily used to count the maximum profit in a following way:

- change all prices to opposite and add 2000 to remove negative values,
- use the SSP algorithm to count minimum profit P,
- subtract $N * 2000$ from P, where N is number of all desserts,
- return -P.

3 Tests

To verify solutions, 11 test cases have been prepared:

- `2006b0.in` test from problem statement, 2 problem instances,
- `2006b1.in` 3 instances, total number of desserts equal to 10 in each instance,
- `2006b2.in` 5 instances, total number of desserts equal to 100 in each instance,
- `2006b3.in` 50 instances, total number of desserts equal to 100 in each instance,
- `2006b4.in` 30 instances, total number of desserts equal to 1 000 in each instance,
- `2006b5.in` 100 instances, total number of desserts equal to 3 000, about 50% possible edges does not exist,
- `2006b6.in` 10 instances 50×50 types, numbers in each type 10..100, prices 0.01..10.00, 10% possible edges does not exist,
- `2006b7.in` 10 instances 50×50 types, 100 elements in each type, prices 0.01..10.0, full bipartite graph,
- `2006b8.in` 10 instances 50×50 types, numbers in each type 90..100, prices 0.01..10.00, 50% possible edges does not exist,
- `2006b9.in` 10 instances 50×50 types, numbers in each type 50..100, prices 0.01 or 0.02, 10% possible edges does not exist,
- `2006b10.in` 10 instances 50×50 types, numbers in each type 1..50, prices 5.00..6.00, full bipartite graph.