MATCHINGS IN GRAPHS

Keywords: graph, matching, cardinality, weight, algorithm

Abstract

A matching in a graph is a set of edges no two of which share a vertex. In this paper there will be discussed theoretical backgrounds and actual algorithmic implementations for finding maximum cardinality and weighted matchings in bipartite graphs, as well as for finding maximum cardinality matchings in dense and general graphs.

Srdjan Krstić, 2005

1 Preface

Graph theory is undoubtedly one of the youngest and most advancing fields of mathematics. The popularity of graph theory lies especially in its great application in computer science. As the matter of fact, the great profusion of the graph theory did not happen until the burst of computers some 50 years ago. The topic of this paper is a rather interesting problem in graph theory. It is finding various matchings in various graphs. Matchings in general have lately been the subject of considerable research, as they often have applications in some real-world problems. The general principle of matching is as follows: given a graph, one should find as many independent edges as possible, such that some global criteria is met, more accurately, such that something is extremed.

2 Some definitions regarding graphs and matchings

A graph G represents a pair (V, E), where V is the set of vertices and E is the set of edges, such that every edge stands for a pair (x, y) of vertices from V. We say that two vertices are *adjacent* if and only if there is an edge connecting them, and that vertex v is *incident* to an edge e if and only if v is one of the vertices connected by e. All the vertices that are adjacent to some vertex x are said to be the *neighbors* of x. The set of all the neighbors of x will be referred to as $\Gamma(x)$. For some subset of vertices $A \subseteq V$, we define $\Gamma(A) = \bigcup_{a\in A} \Gamma(a)$. An edge connecting some vertex x to itself is called a *loop*. A *simple* graph is one that contains no loops and no *multiple edges*, which means that no two vertices are connected by more than one edge. A *path* from vertex x to another vertex y is a sequence of edges that connects these two vertices, where every edge from the graph appears at most once. If some sequence of edges connects some vertex x to itself, than it is called a *circuit*. A *tree* is a graph that contains no circuits. A graph is said to be *connected* if and only if for its any two vertices, there is a path connecting them. If a graph is not connected, it can be partitioned into *connected components*, disjoint with respect to their vertices. A *bipartite* graph $G = (V, E)$ is a graph in which we can distinguish two disjoint sets of vertices, $U \subset V$ and $W \subset V$, such that every edge from E connects one vertex from U with one vertex from W. A *vertex cover* of a graph $G = (V, E)$ is a set $U \subseteq V$, such that every edge from E is incident to at least one vertex from U. A *matching* in a graph $G = (V, E)$ is a set $M \subseteq E$ of independent edges, i.e. edges that do not share any vertices. For every matching M there is a subset $U \subseteq V$ in which every vertex is incident to exactly one edge in M. Then we say that the vertices in U are *matched* while the vertices not incident to any edge in M, and thus not belonging to U, are *unmatched*. We are interesting in finding two types of matchings: a *maximum cardinality matching* in graph $G = (V, E)$ is a set $M \subseteq E$ with maximum cardinality. A matching where $2|M| = |V|$ is called a *perfect* matching. A matching that cannot be improved by simply adding an edge to M is a *maximal* matching. So, every maximum cardinality matching is maximal, but not vice versa. A *maximum weighted matching* is a matching in a graph $G = (V, E)$ where every edge $e \in E$ has its *weight*, and the sum of weights of edges in M is maximized or minimized.

3 Maximum cardinality matching in bipartite graphs

The quest in this problem should be pretty clear by now. So, we have a bipartite graph $G = (V, E)$, with bipartition $\{A, B\}$, and have to find the matching with the maximum number of edges. So how do we find this matching? Consider we have some matching M. Let's consider paths which contain edges from $\{E \setminus M\}$ and $\{M\}$, alternatively, and do not contain any vertex more than once (i.e. do not contain any circuits). Such paths are called *alternating paths.* If an alternating path both starts and ends in unmatched vertices, it is called an *augmenting path.* Clearly, if we take all the edges from this path, and "inverse" them, in a way that we use all edges that have not been in matching for a new matching, and trow out the old ones, we have improved the cardinality of M by 1. Actually, this is the main idea of the algorithm for finding maximum cardinality matchings in bipartite graphs. The basic concept of the algorithm is as follows:

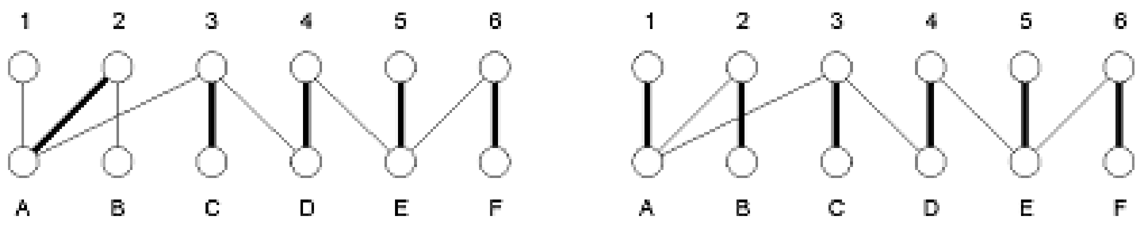
- First, we use a greedy algorithm to find any maximal matching. This step is actually not necessary, but in practice can help improving the running time of the algorithm.
- The main step is that we take all the unmatched vertices in one partition (A) of the graph and apply some graph traversal algorithm (e.g. BFS) with them as the starting points. If we reach an unmatched vertex in B, we have found an augmenting path. If not, than that vertex in B is matched, so we just add the edge by which it is matched to our path, and continue in the same manner. If found an augmenting path, we apply the mentioned "inversion" on the path, update M, and continue with this step as long as there are augmenting paths (i.e. until we have run BFS from every vertex in A).

The algorithm finishes when there are no more augmenting paths to be found. Now we claim that the matching obtained in the process is indeed the maximum matching.

Theorem 1 ***(Berge's theorem)*** *A matching M in G is maximum if and only if there is no augmenting path with respect to it.*

Proof: It is rather obvious that if we have an augmenting path P with respect to M, it would improve M. Conversely, consider that there is not an augmenting path with respect to M, and that M is not maximum. Let M' be the maximum matching. Now we consider the set $M \cup M'$. Clearly, in this set, the maximum degree of any vertex is 2. Moreover, its every connected component is either an alternating path or an alternating circuit with respect to M, and if it is a circuit, it contains an even number of edges, half of them from M and the other half from M'. Given this facts, and the fact that M' has more edges than M, it follows that in at least one of the connected components there will be more edges from M' than from M, and thus this would be an augmenting path. Contradiction.

Note that this theorem applies not only to bipartite, but to general graphs as well. So, by the time the algorithm finishes its work, we definitely have found a maximum cardinality matching. But what is that time? The time complexity of algorithm is $O(|V|(|V| + |E|))$, since the complexity of BFS (breadth-first-search) is $O(|V| + |E|)$, and we apply it for $|V|$ vertices. The well-known speed-up of this algorithm is due to Hopcroft and Karp, which has the running time of $O(\sqrt{|V|}(|V|+|E|))$, and differs from the discussed algorithm only in that it runs breadth-first-search from all the unmatched vertices simultaneously rather than one by one. In this case we have to make sure that the alternating paths used to augment the matching are not modifying one another. The most obvious solution is to restrict them to be vertex disjoint.



Improving the cardinality by finding an augmenting path

Theorem 2 ***(Hall's theorem)*** *A bipartite graph $G = (V, E)$ with bipartition $\{A, B\}$ has a perfect matching for $|A|$ if and only if $|\Gamma(S)| \geq |S|$, for all $S \subseteq A$.*

Proof: It is rather obvious that the first direction of the theorem applies. If G has a perfect matching, then for any subset S, $\Gamma(S)$ will always contain all the matched partners of vertices in S, and thus $|\Gamma(S)| \geq |S|$. Now let's prove the other direction. Let's suppose that a maximum matching M is not perfect. Then there is some vertex $a \in A$ that is unmatched. Let W be a set of vertices connected to a by alternating paths, with respect to M. Since the matching is maximum, none of this paths can be augmenting, and thus a is the only unmatched vertex in W. Now let $S = W \cap A$ and $T = W \cap B$. All the vertices in $S \setminus \{a\}$ are matched with vertices in T, and hence $|T| = |S| - 1$. Furthermore, we have that $\Gamma(S) = T$, since every vertex in $\Gamma(S)$ is connected to a by an alternating path. This implies that $|\Gamma(S)| < |S|$.

4 Minimum weighted matching in bipartite graphs

Again, we have a graph $G = (V, E)$, with bipartition $\{A, B\}$, $|A| = |B| = n$. I will refer to the vertices in A as a_i, and the vertices in B as b_i, $1 \leq i \leq n$. Every edge a_ib_j has its assigned weight w_{a_i,b_j}, $(1 \leq i, j \leq n)$. We try to minimize the sum of all the weights of edges in the matching, $w(M) = \sum\limits_{e \in M} w(e)$. If $|A| \neq |B|$, and/or if the graph is not complete, to be able to run the algorithm, we would simply add the "missing" edges and vertices and assign them the value 0, which will not alter the final result, and then we just ignore this edges that may possibly be contained in the solution. If we are to find the maximum weighted matching, we can simply take the biggest weight, or some even bigger number (inf), change all the weights x to be $inf - x$, and apply the same algorithm. This will work, since our graph is complete. And here is how we find the actual matching:

First, we label each vertex x (i.e. assign a value to it) with a number $l(x)$, such that $\forall a \in A, \forall b \in B, l(a) + l(b) \leq w(a, b)$. Now we make an *equality subgraph* G', a subgraph of G, in a way that it includes all the vertices from V, and those edges (a, b) from E that satisfy the condition $w(a, b) = l(a) + l(b)$. Now we assert:

Lemma 1 ***(Duality lemma)*** *If the equality subgraph G' has a perfect matching M', than M' is the maximum weighted matching in G.*

Proof: So let M' be a perfect matching in G'. Then, by the definition of G', the sum of the costs of all the edges in M' is equal to the sum of the labels of all the vertices in G', and thus equal to the sum of the labels of all the vertices in G. Put into mathematical symbols,

$$\sum_{(a,b)\in M'} w(a,b) = \sum_{x\in M'} l(x) = \sum_{x\in G'} l(x) = \sum_{x\in G} l(x)$$

Now let us consider any perfect matching M with respect to G. By the definition of the vertex labelling, we have

$$\sum_{(a,b)\in M} w(a,b) \geq \sum_{x\in G} l(x) = \sum_{(a,b)\in M'} w(a,b)$$

Therefore, we see that M can never be "better" than M'. Now all we need is a way to implement this idea, algorithm-wise. The following famous algorithm is called the *Hungarian algorithm*:

For the start, we need some feasible vertex labelling. The most obvious feasible labelling is $\forall a \in A, l(a) = \max\limits_{b\in B} w(a,b)$, and $\forall b \in B, l(b) = 0$. Now we create the equality subgraph G' for the given labelling. Then we use the maximum cardinality matching algorithm on the graph G'. If found a perfect matching, the work is done. If not, we need to modify the labelling, and thus G', and search for the perfect matching again. The labelling is modified in such a way that we increase the size of G' by as little as possible. Let Q be a vertex cover of size $|M|$ of the graph G'. Then, let $R = A \cap Q$ and $T = B \cap Q$. Note that the set $A \setminus R$ is actually the set of the vertices from A which are free, or reachable from a free vertex in A via an M-alternating path in G', and that the set T is the set of the vertices from B that are matched to any vertex from $A \setminus R$ (i.e. those passed during the M-alternating paths). Now let $\delta = \min\{w(a,b) - l(a) - l(b);\ a \in A\setminus R, y \in B\setminus T\}$. Clearly, by the definition of the labelling, $\delta > 0$. When found, we use δ to update the labels. For every vertex $a \in A \setminus R$, we increase its label $l(a)$ by δ, and for every vertex $b \in T$, we decrease its label $l(b)$ by δ. Now we just need to prove the correctness of this algorithm. The duality lemma already proves that the perfect matching in an equality subgraph is indeed a maximum weighted matching. But there are two things left to prove regarding the algorithm itself: that subgraphs created by iteration are indeed valid equality subgraphs, and that the algorithm is eventually going to terminate. The first assertion we prove by induction; clearly, the first subgraph we create is valid. Now, suppose that we are at a state having a valid equality subgraph, and ought to iterate to a new one. Let's consider some edge (a,b). There are four distinct cases:

$a \in R$,	$b \in B \setminus T$	$\Rightarrow$	$l(a)$ and $l(b)$ do not change
$a \in R$,	$b \in T$	$\Rightarrow$	$l(a)$ does not change, and $l(b)$ decreases
$a \in A \setminus R$,	$b \in T$	$\Rightarrow$	$l(a)$ increases by δ, and $l(b)$ decreases by δ
$a \in A \setminus R$,	$b \in B \setminus T$	$\Rightarrow$	$l(a) + l(b)$ stays lesser or equal than $w(a,b)$, by the definition of δ

In any case, we obtain a valid equality subgraph. And why does the algorithm terminate? In every iteration, either the cardinality of the maximum matching in G' is directly increased, or the number of vertices that can be reached from U by M-alternating paths grows, where U is the set of unsaturated vertices of M in A. Neither of this two can happen infinitely many times, so the algorithm will eventually terminate.

Let's analyze the complexity of the algorithm. It can be implemented in a straight-forward manner, requiring $O(n)$ iterations, and $O(n^2)$ steps to find the maximum cardinality matching in G' for each iteration, so the overall complexity is $O(n^3)$.

5 Perfect matching in dense graphs

We define a *dense graph* to be a graph $G = (V, E)$, such that the degree of every vertex from V is at least $\lceil \frac{|V|}{2} \rceil$. Here we consider a graph $G = (V, E)$, where $|V| = 2n$, since it is necessary that there is an even number of vertices for a perfect matching to exist. Thus, the degree of every vertex is at least n. By presenting the algorithm for finding a perfect matching in such a graph and proving its correctness, I will also prove that a perfect matching in such a graph always exists. We use induction on the size of the matching (m), and show that while $m < n$, we can either add a new edge to the matching, or replace some edge with two new ones, and thus increment m by 1. The base of the induction is case $m = 1$, which is just an arbitrary edge from E. Now let's consider some matching M with $m < n$ edges. First, we check all the edges not in M, and see if we can add some of them to M. If yes, we are done. Otherwise, M is a maximal matching. Since M is not perfect, there must be at least two distinct vertices v and u, which are nonadjacent, and neither of which belong to M. Obviously, there are at least $2n$ distinct edges that are incident to one of this two vertices. Additionally, all of this edges pair one of the vertices v and u to a vertex that is matched, since M is maximal. Since the number of edges in M is less than n, and there are $2n$ edges coming out of v and u, by the pigeon-hole principle, there must be at least one edge, xy, that is adjacent to at least three edges from v and u. Let those edges be xv, xu and yv (we can assume this without any loss of generality). Now we can simply remove the edge xy from M, and add edges xu and yv, and get a larger matching. QED.

The only thing left is to analyze the complexity of this algorithm. We can obtain a maximal matching in $O(|E|)$. After that, for every of the $O(|E|)$ edges left to complete the matching, we have to find two non-matched vertices, and scan all their neighbors to find two which are matched to eachother. All of this can be done in $O(|V|)$, and thus the total complexity of the algorithm is $O(|E| \cdot |V|)$. Moreover, since $|E| \geq \frac{|V|^2}{4}$, we can say that the complexity of the algorithm is $O(|V|^3)$.

6 Maximum cardinality matching in general graphs

For this problem, we define a new term: an *odd component* of a graph is a connected component with an odd number of vertices. Now let $q(G)$ be the number of odd components of a graph G. Let's prove the following assertion:

Theorem 3 ***(Tutte's theorem)*** *A graph $G = (V, E)$ has a perfect matching if and only if $q(G - S) \leq |S|$ for all $S \subseteq V$.*

Proof: Again, the proof in one direction is trivial. If G has a perfect matching, then for every $S \subseteq V$, every odd component of $G - S$ will have exactly one vertex matched to a vertex in S, and thus $q(G - S) \leq |S|$. Now let's prove the other direction: suppose that $q(G - S) \leq |S|$ for all $S \subseteq V$ indeed, but G does not have a perfect matching. We now find contradiction by finding some *bad* set $S \subseteq G$, i.e. the one that does not satisfy the condition. For the start, we may assume that G has a maximal number of edges (of course, a maximal number of edges such that there is no perfect matching). Indeed, if we make some graph G' by simply adding edges to G, then any odd component of $G' - S$ would be a union of some components of $G - S$, and obviously at least one of them has to be odd, so it follows that if S is bad for G', than it is also bad for G. Considering that G is edge-maximal, it follows that if G has a bad set S, then all the components of $G - S$ are complete and every vertex $s \in S$ is adjacent to all the vertices of $G - s$. Let's mark this assertion as (1). Now, let's suppose that some set $S \subseteq V$ satisfies (1). If $|G|$ is odd, than clearly the empty set is bad. If not, than we can join the odd components of $G - S$ to S disjointly, and pair up the remaining vertices, thus having that S is bad. So proving that G has a subset that satisfies (1) would prove the theorem. Let S be a set of vertices that are adjacent to every other vertex. If S satisfies (1), than we have our set. So let's suppose that it doesn't. Then some component of $G - S$ has non-adjacent vertices a and a'. Now we consider the first three vertices on a shortest path from a to a' in this component. Let them be a, b and c. Clearly, edges ab and bc belong to E, while ac doesn't. Since $b \notin S$, there is a vertex $d \in V$ such that $bd \notin E$. By the maximality of G, there is a matching M_1 of V in $G + ac$, and a matching M_2 of $G + bd$. Now let $P = d \ldots v$ be a maximal path in G that starts in d with an edge from M_1, and contains alternately edges from M_1 and M_2. If the last edge of P is from M_1, than $v = b$, since otherwise we could continue P. Let $C = P + bd$. If the last edge of P lies in M_2, than by the maximality of P the M_1-edge at v must be ac, so $v \in \{a, c\}$. In that case, we define C as the cycle $dPvbd$. In each case, C is an even cycle with every other edge in M_2, and whose only edge not in E is bd. Replacing in M_2 its edges on C with the edges of $C - M_2$, we obtain a matching of V contained in E, a contradiction.

Now let's try to construct an algorithm for finding a maximum matching in a general graph. The greatest contribution to the development of this algorithm was made by Edmonds, and his "blossom shrinking" method. Of course, there have been numerous modifications and improvements of Edmonds' method. The main idea in the algorithm that will be presented is the same as for bipartite graphs: given a matching M, we try to augment it by finding alternating paths.

So suppose we have some matching M and a vertex v that is unmatched in M. We construct an *alternating tree* with v being its root, and the property that all paths in this tree are alternating paths with respect to M. Notice that tree is not necessarily a subgraph of G, since some vertices can show up more than once in the tree. If there is a single vertex u adjacent to v, we add the edge uv to M. Otherwise, every vertex adjacent to v is matched. We construct an alternating tree by induction. First, we put v, as the root, at level 0, and all its neighbors at level 1 of the tree, and join v to all of them. Now since they are all matched, we add their pairs to the tree at level 2, and of course join them to their pairs at level 1. It can happen that some vertices from level 1 are joined to eachother, so they will appear both in level 1 and 2, but placing them on the level 2 as well will have no impact on the outcome of the algorithm, since we can distinguish them by the parity of their level. This is the base of induction. Now suppose that we have constructed the tree through level m, where m is even, and that we have not yet found any augmenting paths. We continue constructing the tree in the following manner: for every vertex x that lies in level m, we examine all the vertices y that are adjacent to x in G. If $xy \in M$, or if $y = v$, we do not make any changes to the tree. So assume that $xy \notin M$ and $y \neq v$. If y is not matched, then we stop. The alternating vx path in the alternating tree, along with the edge xy is an augmenting path with respect to M, so we are done. Suppose now that y is matched to some vertex z. We distinguish three possibilities:

(i) y belongs to some odd level of the alternating tree

(ii) y belongs to some even level of the alternating tree

(iii) y does not belong to the alternating tree

If y belongs to an odd level of the tree, including the level $m+1$ possibly, we do not extend our tree, since in this case the edge yz already belongs to the alternating tree, and we have already explored the possible alternating paths emanating from z, with the edge yz preceding it.

If y belongs to an even level of the tree, we first determine whether or not y lies on the vx path (of length m) in the alternating tree (i.e. whether or not y is the ancestor of x). If it does, we do not make changes to the tree, since we want to keep the property that every path in the alternating tree is also a path in G, and thus when an augmenting path in the tree is found, it is also an augmenting path in G. If y does not appear on the vx path of length m, we add y to level $m+1$, the vertex z to level $m+2$, and of course, join x to y and y to z.

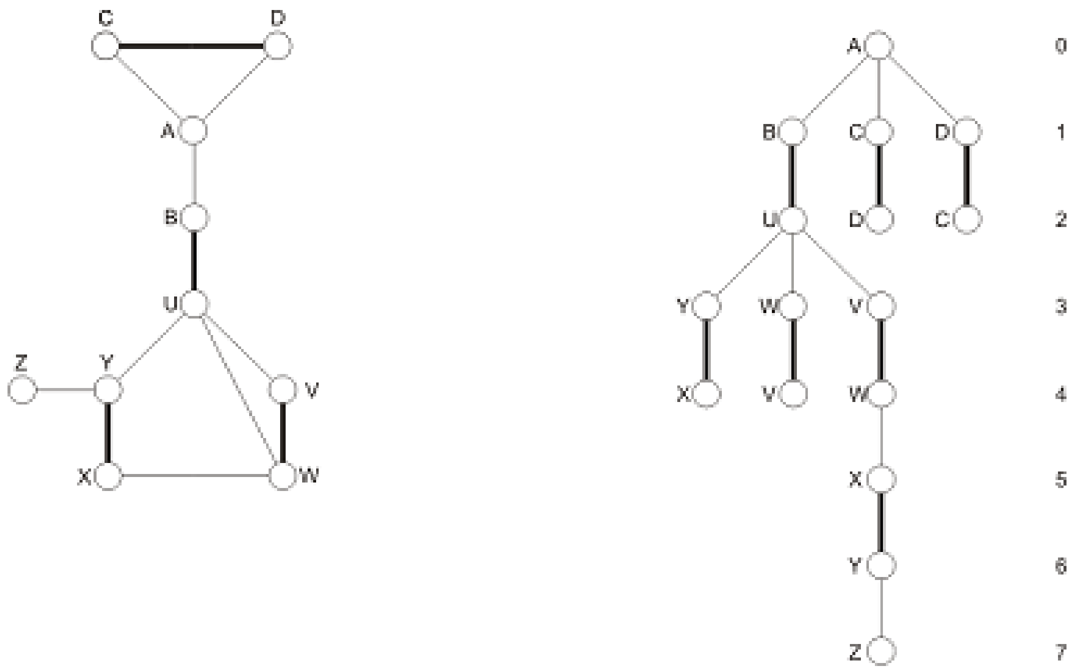
If y does not belong to the tree at all, we add it to level $m+1$, its pair z to level $m+2$, and join x to y and y to z. Note that according to this rules for constructing an alternating tree, every vertex can appear at most twice in in.

We stop constructing the alternating tree if any of the two cases happen:

(i) an augmenting path is found

(ii) if, after the mth level, where m is even, no additional levels can be added.

This alternating tree rooted at v can have at most $|V|-1$ levels. Otherwise, the alternating tree contains a path that emanates from v and has the length of at least $|V|$. This is not possible, since all the vertices on this path have to be distinct. If we finished constructing the tree and found no augmenting paths, then G does not have an augmenting path whose one end-vertex is v.



Constructing an alternating tree

Let's analyze the complexity of this algorithm. We have to construct an alternating tree for $O(|V|)$ vertices. The straightforward implementation of BFS for this stem has the complexity $O(|V||E|)$, because we have to determine whether y appears on the vx path. Hence, the total complexity of the algorithm is $O(|V|^2|E|)$.

7 Sample Problems

1. (CEOI 2002)

Once upon a time, there was a kingdom. It had everything a kingdom needs, namely a king and his castle. The ground-plan of the castle was a rectangle that was divided into $M \times N$ unit squares. Some of the squares are walls, some of them are free. We will call each of the free squares a *room*. The king of our kingdom was extremely paranoid, so one day he decided to make hidden pits (with alligators at the bottom) in some of the rooms. But this was still not enough. One week later, he decided to place as many guards as possible inside his castle. However, this won't be so simple. The guards are trained so that immediately after they see someone, they shoot at him. And so the king has to place the guards carefully, because if two guards would see each other, they would shoot at themselves! Also evidently the king can't place a guard into a room with a pit. Two guards in a room see each other, so each room may contain at most one guard. Two guards in different rooms see each other if and only if the squares corresponding to their rooms are in the same row or in the same column of the plan of the castle and there is no wall between them. (The guard can see only in four directions, much like a rook in chess.) Your task is to find out, how many guards can the king place inside his castle (according to the rules above) and to find one possible assignment of that many guards into the rooms.

Hint: Construct a bipartite graph in which vertices in one partition are continuous parts of a row, surrounded to the left and right by either a wall or the end of the castle. Similarily construct the vertices in other partition from columns. Let the edges exist between to vertices if and only if the according row and column segments intersect and there is no pit at the intersection. Now apply the maximum cardinality matching algorithm.

2. (a problem from acm.timus.ru)

There is certain amount of night guards that are available to protect the local junkyard from possible junk robberies. These guards need to scheduled in pairs, so that each pair guards at different night. The junkyard CEO ordered you to write a program which given the guards characteristics determines the maximum amount of scheduled guards. Only some pairs of guards can work together, because it is possible to find uniforms that suit both of them (The junkyard uses different parts of uniforms for different guards i.e. helmets, pants, jackets. It is impossible to put small helmet on a guard with a big head or big shoes on guard with small feet). Please note that each guard can be scheduled with only one of his colleagues and no guard can work alone.

Hint: This is a straightforward maximum cardinality matching algorithm applied on a general graph.

3. (Serbia and Montenegro qualification round for IOI, 2001)

You are given a $n \times n$ board, with a number written in every cell. You are allowed to put checkers on some cells, but in a way that there is exactly one checker in each row and exactly one in each column. The cost of the layout is the maximum of the numbers in cells which are covered by checkers. Your task is to find the minimum-cost layout.

Hint: Construct a bipartite graph, where rows represent vertices in one partition, and columns in other. You need to find the minimum entry in a some cell, x, such that if you only use edges between vertices in places where the number in a cell where the according row and column intersect is not grater than x. Then run the maximum cardinality matching algorithm to find whether a perfect matching exists. If yes, you are done, and x is the solution. Use binary search to find x, to obtain better complexity.

4.(classical)

You are given a $n \times n$ board, with a number written in every cell. You have to choose n numbers, one from each column and one from each row, such that the sum of the chosen numbers is minimized.

Hint: Construct a bipartite graph, where rows stand for vertices in one partition, and columns in other, and the edge connecting some two vertices has the weight that is the number written in the cell where according row and column intersect. Then apply the minimum weighted matching algorithm.

8 References:

[**1**] Udi Manber, *Introduction to algorithms, a creative approach*, Adison-Wesley Publishing Company Inc., 1989.

[**2**] Reinhard Diestel, *Graph Theory*, SpringerVerlag New York, 2000.

[**3**] Gary Chartrand and Ortrud R. Oellermann, *Applied and algorithmic graph theory*, McGraw-Hill Inc., 1993

[**4**] Cristopher Wolfe, *Weighted Matchings on Bipartite Graphs*, 2003.

[**5**] Harold N. Gabow, *An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs*, University of Colorado, Boulder, Colorado, 1975.

[**6**] John Kececioglu and Justin Pecqueur, *Computing maximum-cardinality matchings in sparse general graphs*, 1998.

[**7**] Samir Khuller, *Matchings*, CMSC 651 Advanced Algorithms, Lecture 2, 2002.

[**8**] Holger Bast, Kurt Mehlhorn, Guido Shäfer and Hisao Tamaki, *Matching Algorithms are Fast in Sparse Random Graphs*

[**9**] J. H. van Lint and R. M. Wilson, *A course in combinatorics*, Cambridge University Press, 2003.

I would like to express special gratitude to Aleksandar Ilić, 2nd year student of the Faculty of Mathematics in Niš.