Acta et Commentationes, Sciences of Education, nr. 1(35), 2024 ISSN 1857-0623
p. 97-102 E-ISSN 2587-3636

CZU: 004.4:519.1=111 DOI: 10.36120/2587-3636.v35i1.97-102

A COMPARATIVE STUDY OF MAXIMUM MATCHING ALGORITHMS

Mihai ESANU, Orizont Lyceum
https://orcid.org/0009-0004-4314-2209
Sergiu CORLAT, Moldova State University
https://orcid.org/0000-0002-5471-2957

Abstract. One of the central problems of discrete optimization is the problem of determining the maximal matching, which can be solved in various ways, depending on the structure of the initial data. In this article, a comparative study of the efficiency of maximal flow algorithms (Ford-Fulkerson) and the direct algorithm for determining maximal matching (Even & Kariv) on unweighted bipartite graphs is conducted. The analysis is based on a set of high-difficulty computing competition problems. Therefore, the results will be useful not only to those interested in the maximal matching problem but also to all those who are preparing to participate in various programming competitions.
Keywords: graph, bipartite graph, maximum flow, matching algorithm, Ford-Fulkerson algorithm, Even & Kariv algorithm.

STUDIU COMPARATIV AL ALGORITMILOR DE POTRIVIRE MAXIMĂ

Abstract. Una dintre problemele centrale ale optimizării discrete este problema determinării potrivirii maxime, care poate fi rezolvată în diverse moduri, în funcție de structura datelor inițiale. În acest articol, se efectuează un studiu comparativ al eficienței algoritmilor de flux maxim (Ford-Fulkerson) și a algoritmului direct pentru determinarea potrivirii maxime (Even & Kariv) pe grafuri bipartite neponderate. Analiza se bazează pe un set de probleme competitive de calcul cu dificultate ridicată. Prin urmare, rezultatele vor fi utile nu numai celor interesați de problema potrivirii maxime, ci și tuturor celor care se pregătesc să participe la diferite competiții de programare.
Cuvinte cheie: graf, graf bipartit, flux maxim, algoritm de potrivire, algoritmul Ford-Fulkerson, algoritmul Even & Kariv.

For a comparative study of the efficiency of maximal flow algorithms (Ford-Fulkerson) and the direct algorithm for determining maximal matching (Even & Kariv) on unweighted bipartite graphs, the following problem is considered:

Problem: In an undirected graph $G = (V, E)$, the set of edges M is called a matching if any pair of edges in M don't have any common vertex. A maximum matching is a matching that contains the largest number of edges possible.

The maximum matching problem can be solved differently on graphs with special properties; in particular, the maximum flow algorithms can be used in bipartite graphs.

In the following, the used algorithms for solving this problem will be described.

1. ***Even & Kariv***

The algorithm is based on an implementation described in this paper, which solves the maximum matching problem for a general graph. It consists of several phases, each being $\mathcal{O}(n^2)$, that find a maximal set of vertices disjoint minimum length augmenting

paths. Because J. E. Hopcroft and R. M. Karp showed that it is possible only to use $\mathcal{O}(\sqrt{n})$ phases, the final complexity is $\mathcal{O}(n^{2.5})$. Each phase consists of four stages. The first stage involves running a simultaneous BFS(Breadth-First-Search) from each unmatched vertex. At the end this stage once we find an alternating path ending in an unmatched vertex of minimum length. This will provide us with the levels of each vertex that can lie on a minimum length path in this phase. In the second stage, we use the levels previously defined to construct a new graph where all the edges not lying on a minimum length alternating path are removed, and odd cycles are shrunk down to a base vertex. In the third stage, we use the HLFS(Highest-Level-First-Search) to construct the maximal set of alternating paths. In the fourth stage, we reconstruct the paths in the reduced graph, then in the original graph, and augment them to increase the matching.

2. *Ford-Fulkerson*

For this, we transform the maximum matching on bipartite graphs problem into a maximum flow problem by connecting each vertex from the right side to a sink t and each vertex from the left side to a source s and making all the edges directed with a flow capacity of 1 and going from left to right. Then, we run the Ford-Fulkerson maximum flow algorithm, which works by iteratively finding a path from s to t, increasing the flow until it finds no such path. The complexity of Ford-Fulkerson is given as $\mathcal{O}(mf)$ where f is the maximum flow. In our case, the maximum flow is the same as the maximum matching, so it has an upper bound of $\frac{n}{2}$, and the number of edges m has an upper bound of $\frac{n^2}{4}$; this gives the final complexity $\mathcal{O}(n^3)$.

Comparative Analysis

In order to find the more efficient algorithm, we solved some specific hard, competitive programming problems formulated for international programming contests.

Problem 1. Double Sort [1]

Statement: Given two permutations a and b, each of size n.

An operation is as follows: Select an integer i such that $1 \leq i \leq n$.

Identify x where $a_x = i$. Then, execute a swap between a_i and a_x.

Similarly, find y where $b_y = i$. Subsequently, swap b_i and b_y.

The problem is to find a minimum cardinality set of moves that rearrange both permutations into ascending order, satisfying the conditions $a_1 < a_2 < \cdots < a_n$ and $b_1 < b_2 < \cdots < b_n$.

Solution

For this problem, we use the cyclic decomposition on permutations a and b. A sorted permutation of length has n disjoint cycles. If we do an operation with index i and it is in a self-cycle, the operation does nothing.

Otherwise, let's suppose that x is the element before i in the cycle ($p_x = i$), and y is the element after i in the cycle ($p_i = y$). After we apply an operation on i, $p_i = i$, and $p_x = y$. So, i leaves the cycle and forms its separate cycle and y becomes the next vertex in the cycle after x. To separate a cycle of k elements into k disjoint cycles, it takes $k - 1$ operations. By picking all vertices in a cycle except one, we can separate it. Now we have to figure out which vertex not to pick in each cycle, and we try to find the maximum amount of such vertices to separate all cycles in both permutations. To do this, we build a graph where each vertex represents a cycle in the cyclic decomposition in the permutations a and b. We draw an edge between a vertex coming from permutation a and a cycle coming from permutation b if the cycles contain some vertex i. We solve the maximum matching problem on this bipartite graph and choose for the operations every number except the ones that created the matched vertices. This will sort the permutations using the minimum number of operations.

Implementations

Both algorithms were tested by evaluation tool of https://codeforces.com. Because of code size, we will present only working times in milliseconds for each solution (on first 25 tests) (table 1).

Table 1. Working time comparation for problem: Double Sort

code/ test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Even-Kariv	15	0	15	0	15	15	15	15	15	15	15	0	0	15	15	15	15	30	31	31	15	46	15	30	30
Max Flow	15	0	15	15	15	15	0	0	30	15	0	0	15	826	858	889	873	811	857	826	795	858	452	420	404

The full implementation of each algorithm can be analyzed, as follows:

1. Solution, using maximal flow algorithm [2].
2. Solution, using Even and Kariv algorithm [3].

Problem 2. Euclid Guess [4]

Statement: The problem involves analyzing Euclid's algorithm for computing the greatest common divisor (GCD).

$GCD(a, b)$:
$if\ a < b,\ swap\ a\ and\ b$
$if\ b = 0,\ return\ a$
$r = a \bmod b$
$if\ r > 0,\ append\ r\ to\ t$
$return\ GCD(b, r)$

Given an empty list t and an array p containing pairs of positive integers, each not exceeding a value m, the modified Euclid algorithm is applied to each pair in p. After processing all pairs in p, list t is shuffled. The task is to determine whether it is possible to reconstruct the original array p from the shuffled list t and, if so, to find at least one such array p that corresponds to the given list t. If no such array exists, this must be identified.

Solution

If $t_i \leq \frac{m}{3}$, it can be generated by a pair $(3t_i, 2t_i)$.

If $t_i > \frac{m}{3}$, then $b > t_i$, this means that $a = b + t_i$ because $2b + t_i > m$. We can see that $b = t_i + t_j$ such that $t_j \leq \frac{m}{3}$ and is a divisor of t_i. Now for each $t_i > \frac{m}{3}$ we have to find a $t_j \leq \frac{m}{3}$ such that $t_j \mid t_i$.

We construct a bipartite graph with the left side containing all elements greater than $\frac{m}{3}$ and the left side everything else. We draw an edge between an element from the left x to an element on the right y if $y \mid x$. Running the maximum matching algorithm, we check if every vertex on the left is matched. If not, say it is impossible to construct p; otherwise, build the p according to the algorithm.

Implementations

Like in previous case both algorithms were tested by evaluation tool of https://codeforces.com. Because of code size, we will present only working times in milliseconds for each solution (on first 25 tests) (table 2).

Table 2. Working time comparation for problem: Euclid Guess

code /test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Even-Kariv	0	15	0	0	46	31	0	0	0	15	0	0	0	0	15	0	0	0	0	0	15	0	0	0	15
Max Flow	0	0	0	0	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103

The Maximal Flow algorithm runs out of time starting with 5-th test.

The full implementation of each algorithm can be analyzed, as follows:

1. Solution, using maximal flow algorithm [5].
2. Solution, using Even and Kariv algorithm [6].

Problem 3. Alice and Recoloring [7]

Statement: Consider a grid A of size $n \times m$ of black and white cells. Initially, all cells are colored white. The following operations are possible:

- Select any subrectangle that includes cell $(1,1)$ and flip the colors of all its cells. The cost of this operation is 1 coin.
- Select any subrectangle that includes cell $(n,1)$ and flip the colors of all its cells. The cost of this operation is 3 coins.
- Select any subrectangle that includes cell $(1,m)$ and flip the colors of all its cells. The cost of this operation is 4 coins.
- Select any subrectangle that includes cell (n,m) and flip the colors of all its cells. The cost of this operation is 2 coins.

The objective is to determine the minimum number of coins required to transform the grid into any desired color configuration.

Solution

To solve this problem, we transform it so that initially, we have the desired grid and want to achieve an all-white matrix. This is equivalent to the original problem. Let's also denote white cells with 0 and black cells with 1. The first observation we use is that operations of types 2 and 3 aren't needed since we can construct the same outcome with operations of type 1 using fewer coins. Next let's think about a grid B where $B_{ij} = (A_{ij} + A_{ij+1} + A_{i+1j} + A_{i+1j+1}) \bmod 2$ where cells outside of bounds have a value of 0. It is clear that A will be all white if and only if B is all 0. When we do an operation on A of type 1 with a corner in (x,y) the value of B_{xy} flips parity. When we do an operation on A of type 2 with a corner in (x,y) it flips the parity of cells $B_{x-1y-1}, B_{x-1m}, B_{ny-1}, B_{nm}$. Easy to observe that it is only optimal to do an operation of type 2 if we can find a pair (x,y) such that B_{xy}, B_{xm}, B_{ny} are all 1. We create a bipartite graph of $m-1$ vertices on the left side and $n-1$ vertices on the right side and draw edges between the i-th element on the left and the $j-th$ elements on the right if B_{ij}, B_{im}, B_{nj} are all 1. Now, by finding the maximum matching, we can calculate the upper bound k of operations of type 2 we are going to do. The final answer will be $min_{1 \le i \le k}(ones\ left\ in\ B + 2k)$.

Implementations

Both algorithms were tested by evaluation tool of https://codeforces.com. Because of code size, we will present only working times in milliseconds for each solution (on first 25 tests) (table 3).

Table 3. Time limits by algorithm for problem: Euclid Guess

code/ test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Even-Kariv	15	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	15	15	0	0	0
Max Flow	0	15	15	15	15	15	15	15	15	15	15	0	15	15	15	15	15	15	15	15	15	0	15	15	15

The full implementation of each algorithm can be analyzed, as follows:

1. Solution, using maximal flow algorithm [8].
2. Solution, using Even and Kariv algorithm [9].

Conclusions

Both algorithms were implemented in C++, and the implementation of the Even & Kariv algorithm is longer, being ~ 500 lines, as opposed to the implementation of Ford-Fulkerson (based on Maximal Flow algorithm) with ~ 170 lines. However, in all cases, the Even & Kariv algorithm performed better. Note that the Ford-Fulkerson algorithms exceeded the 1s time limit for Euclid Guess, the memory data is from that test.

In conclusion, the efficiency of Even and Kariv algorithm is better than the adaptation of Ford Fulkerson Max Flow algorithm. Out of competition develop solutions based on it.

During the competitions try to implement the Ford Fulkerson Maximal Flow algorithm. In a reasonable time (less then 1 hour) the algorithm can be described in C++ and tested.

Bibliography

1. Codeforces Educational Round 141, F. *Double sort* II. Accesibil online: https://codeforces.com/contest/1783/problem/F
2. https://codeforces.com/contest/1783/submission/238408198
3. https://codeforces.com/contest/1783/submission/235288302
4. Codeforces Round 792, *G. Euclid Guess*. Accesibil online: https://codeforces.com/contest/1684/problem/G
5. https://codeforces.com/contest/1684/submission/238411597
6. https://codeforces.com/contest/1684/submission/233672186
7. Codeforces Round 746, *F2. Alice and Recoloring 2*. Accesibil online: https://codeforces.com/contest/1592/problem/F2
8. https://codeforces.com/contest/1592/submission/238408400
9. https://codeforces.com/contest/1592/submission/234579098