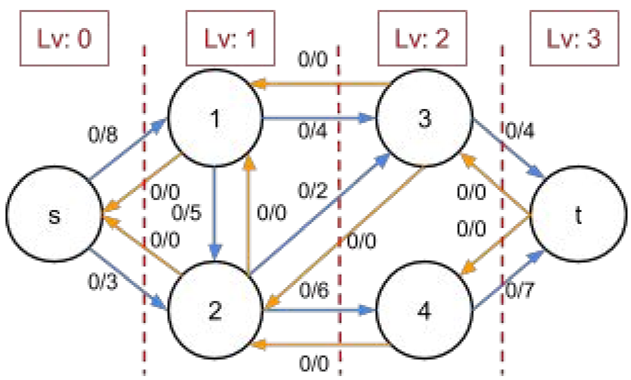
Maximum Flow Algorithms and Applications II

Alexander Shieh
INFOR, Taipei Chien Kuo High School

In my previous report, I wrote about a basic maximum flow algorithm, the Ford-Fulkerson algorithm, but in reality, this algorithm is not fast enough. Therefore in this report I will present another faster and more widely used algorithm, the Dinic's algorithm, followed by some classic types of problems I met and the method to solve it with maximum flow models.



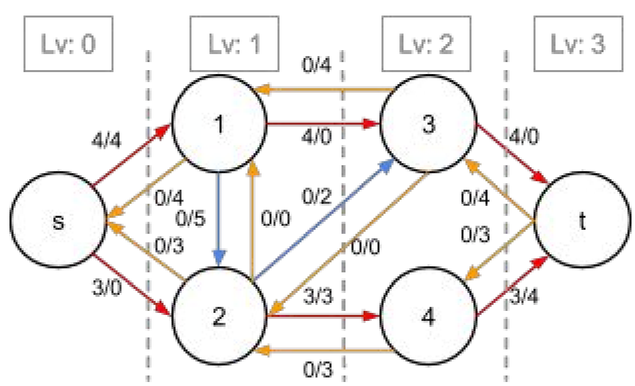
The Dinic algorithm used the following concepts to optimize its performance:

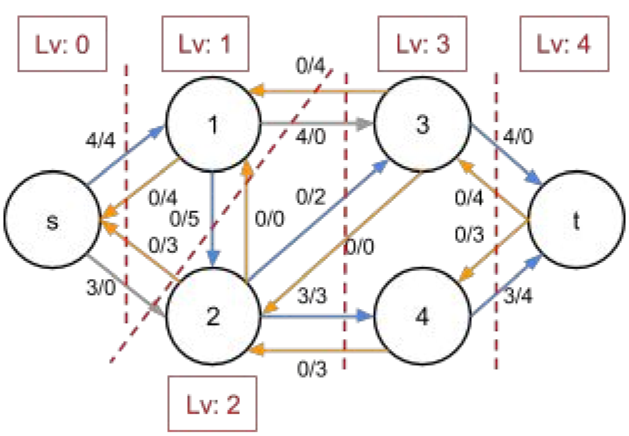
Level Graph: Every vertex v is given a level value which is defined as the shortest path from source s to v (suppose distance of each edge is 1).

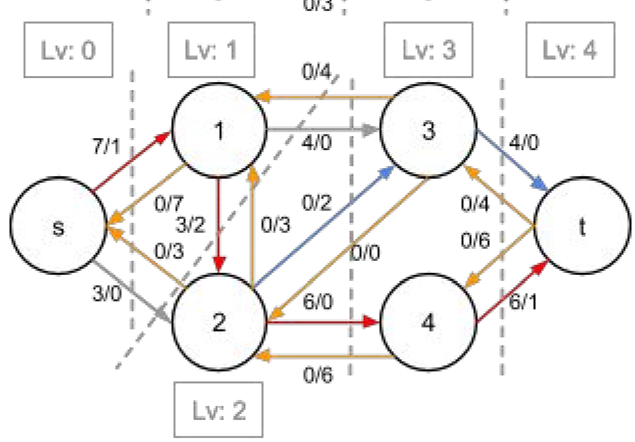
Augmenting Path: Unlike the augmenting path in Ford-Fulkerson algorithm, the augmenting path here is defined as a flow from s to t in the residual network which for every edge (u, v), $level(v) = level(u) + 1$.

Blocking Flow: The blocking flow is the maximum flow(without any augmenting paths) from s to t in the current level graph.

Top: Example of a level graph. Right: Example of the blocking flow of the current level graph.







Left: The recalculated level graph, there exists no edge $(s, 2)$ which has a residual flow so vertex 2 became level 2. Right: A new augmenting path in the new level graph.

The previous examples depicted the core idea if Dinic's algorithm: Keep finding augmenting paths on current level graph and achieve the blocking flow, then revise the level graph of the residual network.

Dinic's Algorithm

While exists a path $s \to t$ **(** *level*(t) *is not null* **)**
  Do revise level for all vertex using Breadth-First Search
  While exists an augmenting path on G_f(Terminate after finding the blocking flow)
    Do find augment path f' in G_f and augment $f \leftarrow f + f'$using Depth-First Search

Now we analyze the time complexity of Dinic's algorithm briefly: Because the distance of a shortest path on the unit weighted graph would not exceed $|V| - 1$, since it won't contain a cycle, we do BFS for at most $|V| - 1$ times. The BFS costs $O(|E|)$ and finding the blocking flow costs $O(|V||E|)$. The total time complexity of Dinic algorithm is $O(|E||V|^2)$ which is significantly faster than the Ford-Fulkerson algorithm and Edmonds-Karp algorithm in dense graphs ($|E| \approx |V|^2$).

Dinic's Algorithm Using C++

```
#include <vector>
#include <queue>
#include <algorithm>
#include <cstring>
using namespace std;
const int INF = 2147483647;
int n, m, level[1002], iter[1002], cnt, s = 0, t = 1001;
struct edge{int to, cap, rev;};
vector<edge> G[1002];
void add_edge(int u, int v, int cap){
    G[u].push_back((edge){v, cap, G[v].size()});
    G[v].push_back((edge){u, 0, G[u].size()-1});
}
void bfs(){ //Revising the level graph
    memset(level, -1, sizeof(level));
    queue<int> Q;
    Q.push(s);
    level[s] = 0;
    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        for(int i = 0; i < G[u].size(); ++i){
            edge &e = G[u][i];
```

```
            if(e.cap > 0 && level[e.to] < 0){
                level[e.to] = level[u] + 1;
                Q.push(e.to);
            }
        }
    }
}
int dfs(int u, int f){
    if(u == t) return f;
    for(int &i = iter[u]; i < G[u].size(); ++i){
        edge &e = G[u][i]; //Avoid checking used edges
        if(e.cap > 0 && level[e.to] > level[u]){
            int d = dfs(e.to, min(f, e.cap));
            if(d > 0){
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}
int max_flow(){
    int flow = 0, f;
    while(true){
        bfs();
        if(level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        while((f = dfs(s, INF)) > 0) flow += f;
    }
}
```

Other than Dinic's algorithm, there's a few push-relabel algorithm that runs $O(|E||V|^2)$, which, after optimization, can achieve $O(|V|^3)$ (push-relabel to front algorithm). These algorithms used a completely different intuition: preflow and height. There's a complete discussion of these algorithm in *Introduction to Algorithms.*

References:

1. Yefim Dinitz. *Dinitz' Algorithm the Original Version and Evens Version*.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*.