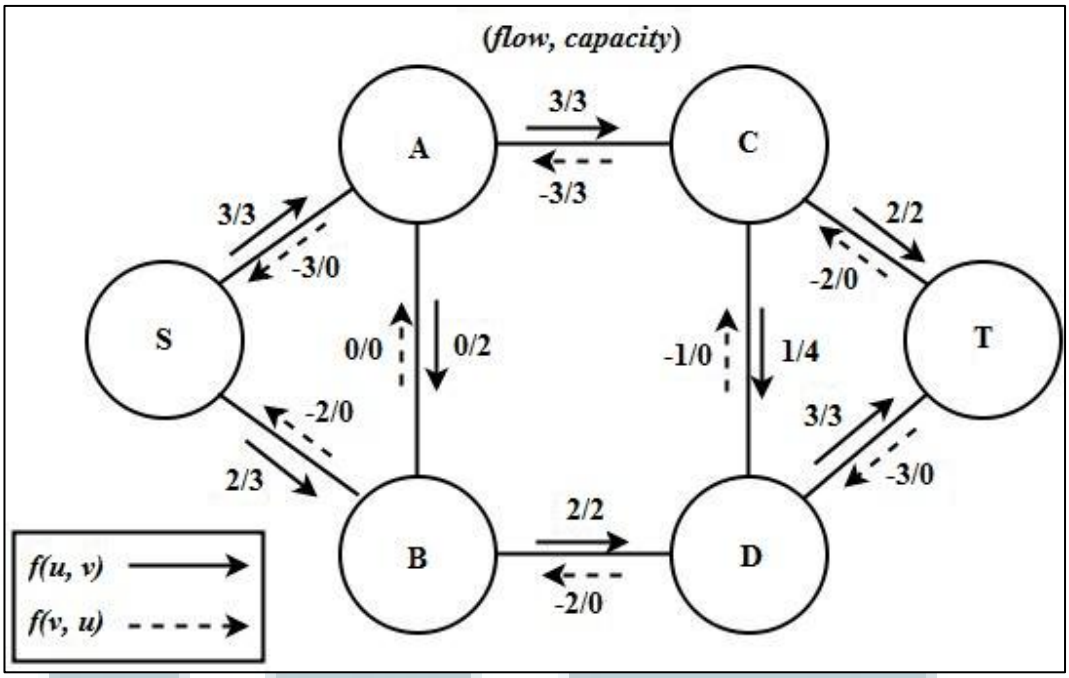
BAB II

LANDASAN TEORI

2.1 Flow Network

Flow network adalah sebuah *directed graph* (*graph* yang setiap *edge*-nya memiliki arah, dari suatu *vertex* ke *vertex* lain), di mana setiap *edge*-nya memiliki kapasitas (*capacity*) tertentu, dan menerima suatu aliran (*flow*). Setiap *vertex* memiliki jumlah aliran masuk dan aliran keluar yang sama, kecuali pada *node* asal (*source*) yang hanya memiliki aliran keluar dan *node* tujuan (*sink*) yang hanya memiliki aliran masuk.

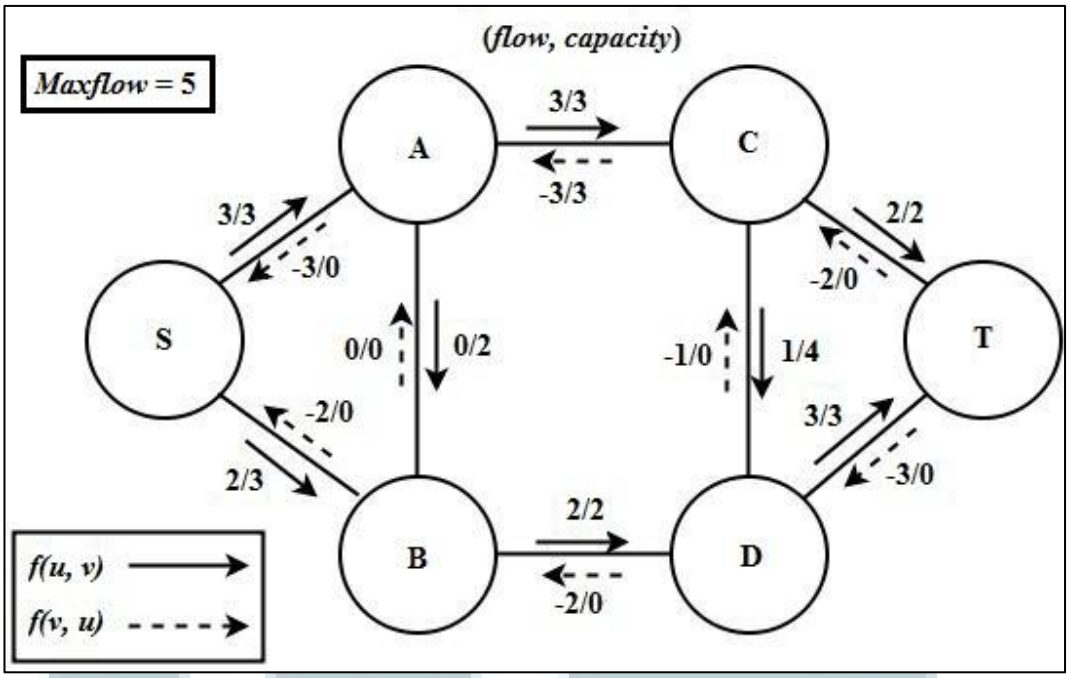
Definisi matematis dari *flow network* adalah sebagai berikut. Diketahui bahwa $G = (V, E)$ adalah suatu *directed graph* dengan sejumlah *vertex* V dan sejumlah *edge* E; di mana setiap *edge* $(u, v) \in E$ memiliki kapasitas $c(u, v)$ yang bernilai non-negatif; aliran $f(u, v)$ di mana $f(u, v) \leq c(u, v)$ dan memiliki dua *vertex* yang unik, yaitu *source* s yang menghasilkan aliran dan *sink* t yang menerima aliran, serta *skew symmetry*, di mana $f(u, v) = -f(v, u)$ yaitu aliran dari u ke v pasti merupakan kebalikan dari aliran dari *node* v ke u. Gambar 2.1 menunjukkan contoh *flow network* yang memiliki *flow* dan *capacity*.



Gambar 2.1 Contoh *Flow Network*

2.2 Maximum Flow

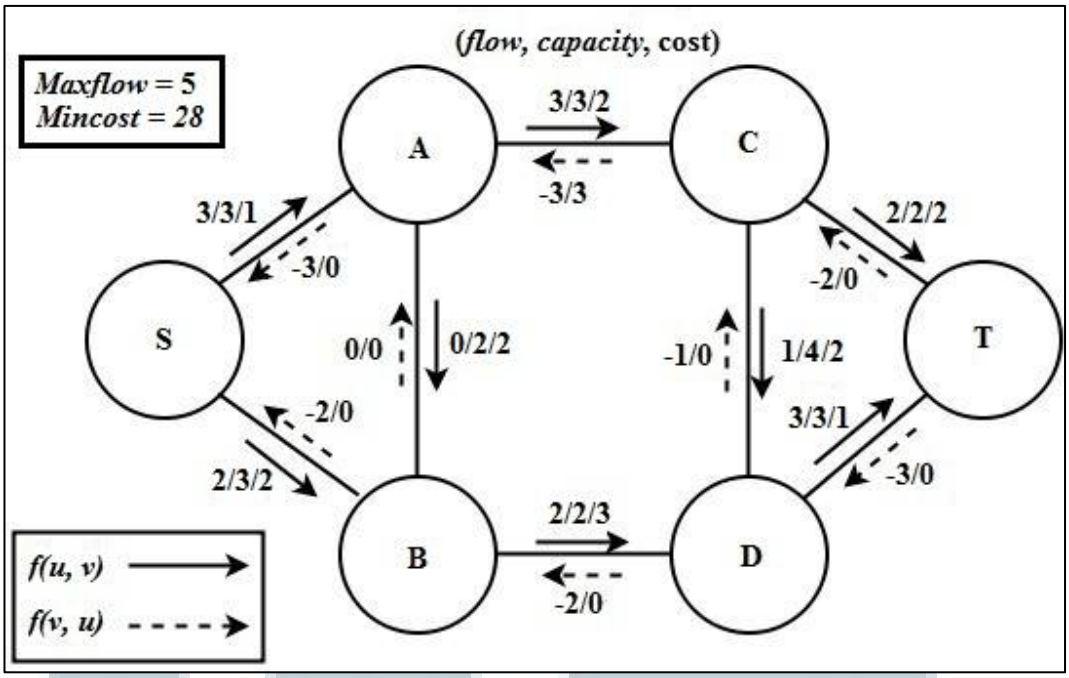
Menurut Ford dan Fulkerson (1957), suatu *flow network* dikatakan sebagai *maximum flow*, jika tidak memiliki *augmenting path*, yaitu suatu *path* dari *sink* ke *source* di mana sisa kapasitas (*residual capacity*) $c_f(u,v) = c(u,v) - f(u,v)$ minimalnya lebih besar dari 0. Namun, istilah *maximum flow* lebih dielaborasi menjadi *maximum flow problem*, yaitu suatu metode untuk menemukan jumlah aliran maksimal yang dapat diterima oleh *node* tujuan (*sink*). Gambar 2.2 menunjukkan contoh *maximum flow network*.



Gambar 2.2 Contoh *Maximum Flow Network*

2.3 Min-Cost Max-Flow

Suatu pengembangan dari *maximum flow problem*, yaitu mencari *maximum flow* dari suatu *flow network* yang memiliki *minimum cost*. Setiap *edge* yang ada pada *graph* tidak hanya memiliki kapasitas, namun juga memiliki biaya tertentu. *Min-cost max-flow* umumnya digunakan untuk menemukan *minimum cost maximum matching* (Hopcroft dan Karp, 1973) atau pencocokan maksimal dengan biaya terendah, seperti jaringan telekomunikasi, jasa penerbangan, *maximum bipartite matching*, *client assignment problem*, dan *job assignment problem*. Gambar 2.3 menunjukkan contoh *minimum cost maximum flow network* yang memiliki *flow*, *capacity*, dan *cost*.



Gambar 2.3 Contoh *Min-Cost Max-Flow Network*

2.4 Edmonds-Karp dan Dijkstra

Dalam implementasinya, algoritma *min-cost max-flow* merupakan kombinasi dari algoritma *max-flow* dan *shortest path*. Salah satunya yaitu kombinasi antara algortima Edmonds-Karp (1972) dan Dijkstra (1959). Gambar 2.4 menunjukkan *pseudocode* algoritma Dijkstra untuk mencari *shortest path* dari suatu *node* ke *node* lain pada suatu *graph*.

```
function Dijkstra(Graph, source):
  for each vertex v in Graph:           //  Inisialisasi
    dist[v] := infinity                 //  jarak awal dari source ke vertex v
    previous[v] := undefined            //  node sebelumnya dari path yang
                                            optimal
 dist[source] := 0                      //  jarak dari source ke source
  Q := the set of all nodes in Graph     // semua node pada Graph masuk ke
                                            dalam suatu queue (antrian) Q
  while Q is not empty:
    u := node in Q with smallest dist[]
    remove u from Q
    for each neighbor v of u:           //  dimana v masih ada di dalam antrian
                                            Q
      alt := dist[u] + dist_between(u, v)
      if alt < dist[v]
        dist[v] := alt
        previous[v]:= u
  return previous[]
```

Gambar 2.4 *Pseudocode* untuk Algoritma Dijkstra
(Sumber: *http://www.gitta.info/*, dengan perubahan)

Gambar 2.5 menunjukkan *pseudocode* algoritma Edmonds-Karp untuk mencari *maximum flow* dari suatu *graph* yang berbentuk *flow network* (*graph* yang setiap *edge*-nya memiliki *flow* dan *capacity*.

```
algorithm EdmondsKarp
  input:
    graph                                   (Graph dengan Adjacency List
                                             dengan node yang memiliki
                                             capacity, flow, reverse dan
                                             destination)
    s                                       (Source)
    t                                       (Sink)
  output:
    flow                                    (Nilai dari maximum flow)
    flow := 0                               (Aliran awal bernilai 0)
    q := array(1..n)                        (Inisialisasi q sebesar
                                             ukuran Graph)
  while true
    qt := 0                                 (Variabel untuk melakukan
                                             iterasi)
    q[qt++] := s                            (Inisialisasi array Source)
    pred := array(q.length)                 (Inisialisasi array
                                             Predecessor)
    for qh=0;qh < qt && pred[t] == null
      cur := q[qh]
      for (graph[cur])                      (Iterasi seluruh edge)
        Edge[] e :=  graph[cur]             (Setiap edge harus memiliki
                                             capacity)
        if pred[e.t] == null && e.cap > e.f
          pred[e.t] := e
          q[qt++] : = e.t
        if pred[t] == null
          break
      int df := MAX_VALUE                   (Inisialisasi ke nilai
                                             maksimal)
      for u = t; u != s; u = pred[u].s
          df := min(df, pred[u].cap - pred[u].f)
      for u = t; u != s; u = pred[u].s
        pred[u].f := pred[u].f + df
        pEdge := array(PredEdge)
        pEdge := graph[pred[u].t]
        pEdge[pred[u].rev].f := pEdge[pred[u].rev].f - df;
      flow := flow + df
  return flow
```

Gambar 2.5 *Pseudocode* untuk Algoritma Edmonds-Karp
(Sumber: *http://www.cse.unt.edu/*, dengan perubahan)