Penggunaan Algoritma Edmonds-Karp dalam Mencari Debit Air Maksimum dari Sistem Kanal Bawah Tanah

Naufal Prima Yoriko - 13518146
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518146@std.stei.itb.ac.id

***Abstraksi*—Sistem kanal bawah tanah dapat dimodelkan menjadi salah satu bentuk graf khusus, yakni flow network. Permodelan sistem kanal tersebut dapat membantu kita menganalisis berbagai komponen dalam sistem kanal tersebut, salah satu yang paling penting adalah mencari debit maksimum air dari sistem kanal tersebut. Salah satu metode mencari debit yang dapat digunakan adalah algoritma Edmonds-Karp, yakni mencari maxflow dari suatu flow network. Dengan mengetahuinya kita dapat mengetahui dan memcahkan berbagai masalah yang dapat timbul.**

***Kata Kunci*— Algorima Edmonds-Karp, Flow Network, Kanal, Maxflow.**

I. PENGENALAN

Sistem kanal bawah tanah adalah salah satu komponen penting dalam kehidupan masyrarakat, khususnya daerah perkotaan. Daerah perkotaan memiliki wilayah tanah kosong yang semakin sedikit, sehingga umum digunakan sistem pembuangan air berupa kanal bawah tanah guna menghemat tanah yang terpakai.

Fungsi utama dari sistem kanal ini adalah guna mentransportasikan air, umumnya untuk pembuangan ke daerah lain. Hal ini dilakukan karena daerah serapan air yang semakin sedikit, akan menimbulkan masalah jika terjadi penumpukan air di suatu wilayah akibat suatu hal, semisal hujan yang sangat lebat, agar tidak menimbulkan maslah besar, semisal banjir.

Namun, sistem kanal ini tentu saja memiliki debit maksimum air yang dapat dipindahkan. Maka, apabila sudah terlampaui, tentu saja tidak akan mampu mencegah permaslahan yang timbul. Oleh karena itu, penting untuk mengetahui, terutama bagi perancang tata kota, untuk dapat mengetahui debit maksimum dari sistem kanal yang dibuat, sehingga dapat mengindari maslah yang timbul.

Salah satu metode mencarinya adalah dengan memodelkan sistem kanal menjadi salah satu bentuk graf khusus, yang disebut dengan flow network. Menggunakan permodelan ini, debit maksimum (maxflow) dari suatu sistem kanal dapat digunakan dengan algoritma tertentu, seperti Edmonds-Karp.

II. STRUKTUR GRAF

A. Pengertian dan Karakteristik

Dalam dunia komputasi dan algoritma struktur data, graf adalah salah satu struktur data non-linear dimana secara garis besar terdiri dari dua komponen utama:

1. Titik, simpul atau node (atau dapat disebut vertex)
   Titik umumnya disimbolkan dengan satu karakter alfabet, seperti A, B , dan lainnya.
2. Sisi atau edge
   Sisi utamanya berfungsi sebagai pengubung antar node, sehingga biasanya dapat disimbolkan sebagai tuple (a, b) dimana a dan b adalah dua node yang dihungkan oleh edge itu.

Graf biasa dinyatakan sebagai G = (V, E) dimana G sebagai simbol graf, sedangkan V adalah set/himpunan dari kumpulan node pada graf dan E adalah set dari edge graf tersebut.

B. Istilah dan Terminologi

Pada graf ada beberapa istilah umum yang sering digunakan, diantaranya:

1. Bertetangga/Adjacent
   Dua node i dan j dikatakan bertetangga apabila ada terdapat sebuah edge yang menghubungkan keduanya langsung tanpa perantara edge lain, dengan kata lain terdapat edge E = (i, j).
2. Bersisian/Incidency
   Node V dan edge E dikatakan bersisian jika salah satu ujung E ada di V, atau E = (V, X) dimana X adalah node sembarang.
3. Derajat/Degree
   Derajat suatu node V adalah banyaknya sisi yang bersisian dengan V.
4. Lintasan/Path
   Lintasan adalah jalur yang dilalui untuk mencapai suatu node ke node lain. Lintasan dapat disimbolkan sebagai sekuens node ataupun edge. Untuk tiap node yang berurutan e_i, e_{i+1} mesti terdapat edge yang menghubungkan. Panjang dari lintasan adalah banyaknya edge yang dilaluinya.
5. Siklus/Cycle

Siklus pada graf adalah lintasan dimana node awal dan akhirnya sama, dengan syarat tidak ada dua node yang sama yang dilalui selain pada titik awal dan akhir.

6. Terhubung/Connected

   Dua node atau subgraf dikatakan terhubung, jika setidaknya ada sebuat edge yang menghubungkan mereka, baik secara langsung maupun tidak.

C. Macam Graf

Struktur data graf pada penggunaannya memiliki beberapa variasi. Jika dilihat dari macam edge-nya dibagi menjadi:

1. Graf tak berarah / Undirected Graph

   Ini merupakan graf sederhana dimana edge hanya menyatakan keterhubungan dua node tanpa perlu memerhatikan arah, sehingga edge E = (i, j) = (j, i).

2. Graf berarah / Directed Graf

   Graf ini selain menyatakan keterhubungan, juga memiliki arah. Pada graf berarah, jika edge E1 = (i, j) dan E2 = (j, i), maka E1 ≠ E2.

Struktur data graf pada penggunaannya memiliki beberapa variasi. Jika dilihat dari macam bobot/value komponen dibagi menjadi:

1. Graf tak berbobot / Unweighted Graph

   Graf ini hanya menggambarkan wujud dari graf tanpa ada bobot khusus pada edge maupun nodenya, atau dengan kata lain bobot tiap edge atau node setara. Sehingga, minimum penggunaan dari graf ini adalah penggunaan jumlah minimum dari edge atau node.

2. Graf berbobot / Weighted Graph

   Pada graf ini tiap komponen dapat diberikan bobot yang berbeda-beda, bahkan dapat negatif. Umumnya yang diberikan bobot pada graf ini adalah edgenya, walau dalam bebapa kasus dapat saja nodenya atau pun keduanya. Graf ini lebih realistik dalam memodelkan suatu kasus, sehingga lebih umum digunakan dalam implementasi suatu persoalan, seperti salah satu yang paling pol=puler adalah travelling salesman problem.

D. Representasi Graf

Dalam pembuatan dan pengolahan graf oleh komputer, karena strukturnya yang tidak linear, kita memerlukan data yang dapat merepresantasikan wujud dari graf itu sendiri ke dalam struktur linear yang umum digunakan, dan tentu saja bisa direpresentasikan oleh komputer. Terdapat beberapa macam representasi untuk menyatakan graf:

1. Matriks keterhubungan / Adjacency Matrix

   Metode ini membuat sebuah matriks M dengan ukuran NxN, dimana N adalah banyak dari node di sebuah graf dan isi dari M[i][j] adalah 0 (jika tak terhubung) dan 1(jika terhubung). Untuk graf berbobot dapat diganti menjadi bobotnya jika terhubung, dan infinit jika tak terhubung.

   Metode ini memiliki keunggulan yakni struktur datanya yang simpel, mudah diimplementasi, memiliki waktu accesing yang cepat. Namun memiliki kekurangan yang cukup mendasar, yakni boros penggunaan secara memory maupun waktu(saat melakukan iterasi seperti DFS atau BFS).

2. Daftar keterhubungan / Adjacency List

   Metode ini membuat sebuah array A dimana indeksnya merepresentasikan node dari graf, sedangkan komponennya adalah list dari node-node yang terhubung dengan node tersebut (list kosong jika tidak ada yang terhubung), dengan kata lain membuat array of list. Untuk graf berbobot, komponennya dalah list of pair (N, val) dimana N adalah node dengan val sebagai bobot dari edge yang menghubungkannya.

   Graf ini sedikit rumit dalam pengimplementasian karena menggunakan dua atau tiga tipe struktur data secara simultan, namun dari sisi penggunaan memory dan waktu secara garis besar lebih efisien dari adjacency matrix.

III. Flow Network

A. Pengertian dan Karakteristik

Flow Network adalah salah satu variasi dari graf yang sederhananya menggambar yang memiliki karakteristik khas:

1. Setiap edge memiliki kapasitas (yang disimbolkan C) yang mana menyatakan nilai maksimum aliran/flow (yang disimbolkan F) yang dapat lewat. Sehingga

   $$0 \leq F \leq C$$

2. Kapasitas dan flow memiliki arah, yang artinya ‘mengalirkan’ sesuatu dari suatu node ke node lain. Arah dari flow tentu saja harus sama dengan arah dari kapasitas.
3. Node ‘biasa’ direpresentasikan sepagai percabangan (mungkin pula tidak bercabang) antar edge dan tidak memiliki bobot. Pada node berlaku hubungan

4. $$\Sigma \text{Fout} = \Sigma \text{Fin}$$

   Hubungan ini disebut aturan konservasi aliran.

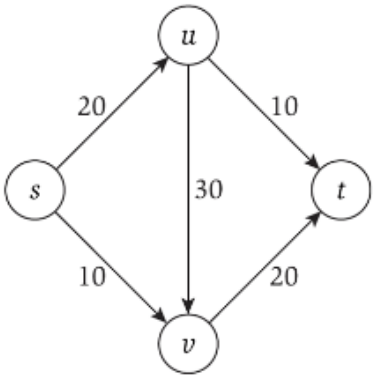
5. Terdapat dua node ‘tak biasa’, yakni source dan sink. Source berperan sebagai titik permulaan aliran/flow, sedang sink merupakan ujungnya. Pada flow network juga berlaku

   $$\Sigma \text{Fsource, out} = \Sigma \text{Fsink, in}$$

   $$\Sigma \text{Fsource, in} = 0$$

   $$\Sigma \text{Fsink, out} = 0$$

Pada kasus sederhana, flow network hanya memiliki satu source dan satu sink, gambar dibawah ini merupakan salah satu ilustarasi dari flow network



Gambar 3.1. Contoh sederhana dari Flow Network (sumber: http://courses.cs.vt.edu/~cs4104/murali/Fall09/lectures/lecture-20-network-flow.pdf)

Pada flow network ini, angka pada edge merepresentasikan kapasitas dari edge, sedangkan arah pada edge juga merepresentasikan arah aliran yang dapat dilakukan edge tersebut. Disini, s adalah source, t adalah sink, u dan v adalah node biasa.

Flow dapat dinyatakan sebagai G = (V, E, s, t, C), dimana G adalah flow network, V adalah set node non-source dan non-sink, E set edge, s adalah source, t adalah sink, dan C adalah set dari kapasitas edge (ukuran dan urutannya sama dengan set E).

B. Graf Residu / Residual Graph

Dalam flow network, dikenal suatu graf pembantu dalam menganalisis flow network yang dikenal sebagai graf residu. Pada graf residu dikenal karakteristik berikut:

1. Node, sink, dan source dasarnya sama dengan flow network asal.
2. Pada edge E di flow network, dengan kapasitas C dan seumpama dialiri flow F, maka pada graf residu akan terdapat

   $$Eres = C - F$$

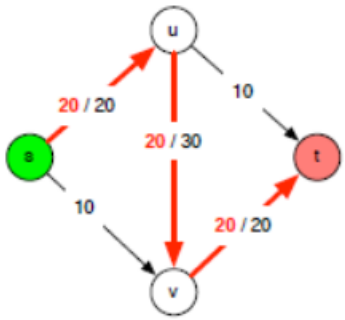
   E_{res} ini menyatakan kapasitas sisa yang dapat digunakan.

   $$Ecounter = F$$

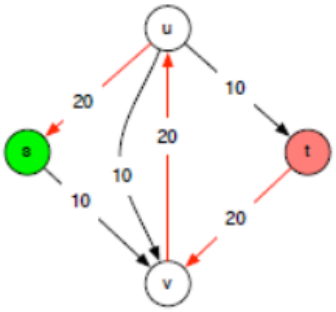
   $E_{counter}$ menyatakan kapasitas yang sudah terpakai.
3. E_{res} maupun $E_{counter}$ akan menghubungkan node yang sama, namun E_{res} memiliki arah yang sama dengan kapasitas sedangkan $E_{counter}$ sebaliknya.
4. E_{res} maupun $E_{counter}$ pada implementasinya nanti akan digunakan untuk tujuan yang sama, yakni kapasitas residu.

Graf residu ini dibuat dengan tujuan agar dapat meng-'undo' langkah iterasi (dengan digunakannya kapasitas residu, bukan kapasitas asli) dalam pengakumulasian total flow suatu network, yang akan dijelaskan lebih detail nanti.

Berikut ini adalah ilustrasi yang dapat dibuat dari flow network contoh pada gambar 3.1



Gambar 3.2.1. Contoh path terpilih s-u-v-t dengan flow 20 (sumber : https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/netflow.pdf)



Gambar 3.2.2. Graf residu setelah implementasi path (sumber : https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/netflow.pdf)

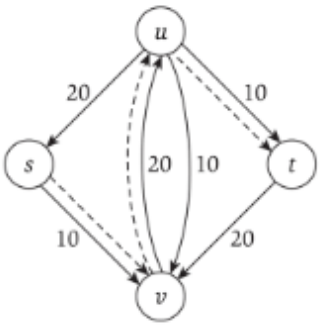
Disini dapat kita lihat bahwa $E_{counter}$ adalah bagian yang ditandai berwarna merah, dan sesuai karakteristik, arahnya berlawanan flow, namun besarnya sama. Sedangkan E_{res} adalah edge yang berwarna hitam, dan E_{res} yang besarnya nol tidak kita gambarkan disini (karena graf residu awal juga memiliki $E_{counter}$ yang besarnya nol, namun tidak dituliskan).

C. Jalur Teraugmentasi / Augmented Path dan Bottleneck capacity

Salah satu istilah penting lainnya yang perlu kita gunakan dalam menganalisis flow network adalah jalur teraugmentasi. Jalur teraugmentasi sendiri adalah jalur/path yang tersedia yang dapat menghubungkan source dengan sink. Definisi 'tersedia' disini berarti kapasitas residu dari edge (i, j), dengan i, j adalah setiap pasangan node yang berurutan dalam path, haruslah lebih besar dari 0, dengan kata lain masih mungkin dialiri suatu flow tambahan.

Hal penting lainnya adalah bottleneck capacity atau kapasitas ambang(karena aliran air disuatu botol yang menuangkan air terhambat oleh tutup botolnya yang kecil), yakni kapasitas residu minimum pada suatu jalur teraugmentasi. Kapasitas bottleneck ini digunakan untuk memastikan bahwa setiap jalur teraugmentasi mengalirkan flow maksimum sebesar bottleneck tersebut.

Untuk contoh ilustrasi dari jalur teraugmentasi, semisal kondisi tercipta setelah graf residu seperti pada gambar 3.2.2, maka dapat dipilih jalur teraugmentasi seperti berikut



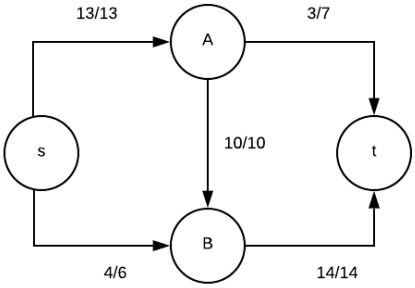
Gambar 3.3. Jalur teraugmentasi s-v-u-t yang ditandai dengan dashed line
(sumber : http://courses.cs.vt.edu/~cs4104/murali/Fall09/lectures/lecture-20-network-flow.pdf)

Dari jalur teraugmentasi s-v-u-t dapat dilihat ada 3 edge yang dilalui, yang mana masing-masing memiliki kapasitas residu 10, 20, dan 10. Maka berdasar definisi kapasitas bottlenecknya sebesar 10.

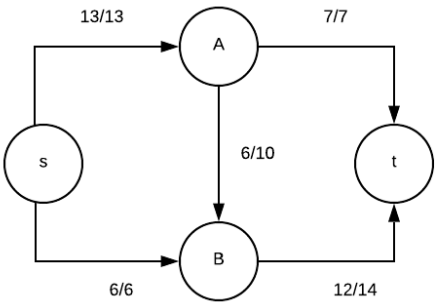
D. Max Flow

Salah satu problem utama yang dapat dicoba pemecahannya adalah mencari aliran/flow maksimum yang dapat masuk ke sink, atau lebih dikenal dengan istilah maxflow. Perlu diketahui, pada flow network, saat tidak ada jalur teraugmentasi yang berbeda hasilnya tidak selalu sama, dapat berbeda, tergantung cara kita memulai pengisian flow network itu sendiri. Dengan kata lain, flow akhir yang sampai ke sink dapat berbeda-beda.

Berikut ini akan ditampilkan dua flow network yang sama, dan keduanya sudah tak bisa ditambahkan flow lagi



Gambar 3.4.1. Flow network dengan flow akhir sebesar 17
(sumber : Dokumen Pribadi)



Gambar 3.4.2. Flow network yang sama dengan flow akhir sebesar 19
(sumber : Dokumen Pribadi)

Dua flow network yang sama, namun di gambar 3.4.2 bisa menghasilkan maxflow ynag sesungguhnya, sedangkan gambar 3.4.1 tidak tepat. Akibat hasil yang sensitif terhadap cara pengisian inilah, muncul masalah maxflow ini. Oleh karena itu, perlu disusun suatu metode atau algoritma, supaya kita selalu mendapat nilai maxflow dari suatu flow network.

IV. Pencarian Max Flow dari Flow Network

A. Pendekatan dengan Solusi Greedy

Untuk penyelasaian masalah yang ingin kita cari solusinya, sebagai awalan kita tentu memikirkan sebuah pendekatan yang intuitif. Kemungkinan terbesar, cara yang intuitif yang terpikirkan tersebut adalah dengan terus mencari jalur dari source menuju sink (yang mana sudah didefinisikan dengan istilah jalur teraugmentasi), terus mengiterasinya hingga tak ada lagi jalur teraugmentasi yang tersedia, setelah itu menjumlahkan semua yang masuk ke sink untuk memeroleh nilai max flow dari flow network tersebut. Inilah yang disebut pemecahan greedy, yakni metode yang mengupayakan untuk selalu mencari optimasi/keuntungan dari setiap langkah kecil (dalam hal ini setiap iterasi) dalam pemecahan masalah.

Ternyata, ada masalah yang timbul disebabkan pendekatan ini. Pendekatan ini hanya mencoba mencari keuntungan dari setiap iterasi, tanpa bisa memerhatikan secara global (satu algoritma keseluruhan) bagaimana cara mendapat keuntungan maksimum di akhir.

Sebagai contoh kesalahan yang mungkin timbul adalah flow akhir dari gambar 3.4.1. Kesalahan ini timbul akibat kesalahan dalam melakukan pengisian, sebagaimana yang telah disebutkan sebelumnya. Maka, dapat disimpulkan pendekatan secara greedy seperti ini saja tidak dapat diterapkan.

B. Algoritma Ford-Fulkerson

Sampai saat ini, belum ditemukan metode untuk meninjau secara global pencarian maxflow, maka sebagai gantinya perlu dicari cara agar bisa meng-‘undo’ kesalahan langkah dalam suatu iterasi, dan mengulangi, atau dengan kata lain menyesuaikan flow (yang normalnya dialokasikan berlebihan) agar dapat digunakan secara optimum. Sehingga, dicetuskan algoritma Ford-Fulkerson ini guna menyempurnakan pendekatan greedy yang kita coba lakukan sebelumnya.

Algoritma Ford-Fulkerson menggunakan pembuatan graf residu sebagaimana yang telah dijelaskan di bagian sebelumnya. Hal ini berguna untuk melakukan reversing/undo bilamana terjadi kelebihan alokasi flow di suatu edge, untuk kemudian dapat dikurangi. Hal ini dimungkinkan sebab adanya $E_{counter}$ pada graf residu sehingga segala kesalahan dapat direalokasi kembali.

Berikut ini adalah pseudocode dari algoritma Ford-Fulkerson

```
Max-Flow(Graf G)
  Inisialisasi flow[edge] = capacity[edge] untuk
    semua edge di G
  Inisialisasi maxflow ke 0
  While(ada jalur teraugmentasi s ke t di graf residu)
        Misal P adalah jalur teraugmentasi tersebut
        Misal B adalah kapasitas bottleneck dari P
        flow[edge] yang searah dengan jalur
            teraugmentasi dikurangi B
        flow[edge]  yang berlawanan arah dengan
            jalur teraugmentasi ditambah B
        maxflow ditambah dengan B
  return B
```

Dengan menggunakan algoritma ini, yang mana sudah dilengkapi sistem perbaikan, maka dipastikan akan mendapat maxflow, walau sebenarnya ada satu kelemahan mendasar pada algortima ini. Algoritma ini sensitif terhadap nilai dari kapasitas edge yang diberikan, dan akan terus mencari solusi hingga tidak ada augment path tersisa. Pada kasus kapasitas dari edge adalah bilangan real positif (bukan bulat positif), ada kemungkinan terjadi infinite loop akibat sulit atau bahkan tidak mungkin menghabiskan jalur teraugmentasi. Sehingga algoritma ini hanya berkerja dengan baik pada kapasitas bilangan bulat positif.

C. Algoritma Edmonds-Karp

Jika kita perhatikan, algortima Ford-Fulkerson ini sudah jelas langkah logikanya dalam penyelesaian masalah maxflow ini. Namun, jika diperhatikan lagi, untuk bisa diimplementasikan di komputer algoritma ini masih sedikit ‘mentah’, masih perlu pernyempurnaan lebih lanjut.

Algoritma Ford-Fulkerson belum memberikan metode pencarian jalur terugmentasi secara rinci, maka algoritma itu disempurnakan oleh algoritma Edmonds-Karp. Algoritma ini membuat metode BFS(*breadth-search-first*) untuk mendapatkan jalur teraugmentasi. Maka, pada makalah ini kita akan menggunakan algoritma Edmonds-Karp ini untuk menyelesaikan persoalan mengenai kanal ini.

D. Impelementasi Algoritma Edmonds-Karp di Bahasa C++

Penggunaan pseudocode saja belum mampu untuk menunjang penyelesaian masalah secara langsung, karena algoritma tersebut belum dapat dijalankan, Oleh karena itu, saya membuat algoritma ini dalam bahasa C++ untuk mencoba menjalankan dan menguji algoritma ini, atau dapat dengan mudah diimplementasikan oleh pembaca kedepannya.

Pada program ini saya mencoba meminta input banyak node/vertex, indeks source dan sink, jumlah edge, beserta list edge dan kapasitasnya. Data ini graf ini adalah ditranslasi ke representasi graf adjacency matriks, dan beberapa array dan queue pembantu. Output dari program ini adalah nilai dari maxflow beserta edge list dengan nilai flow di tiap edgenya.

```cpp
const int INF = 1e9+7;
const int sizeG = 1e3+3;
int Graf[sizeG][sizeG];
// graf dengan representasi adjacency matrix
int ResGraf[sizeG][sizeG];
// graf residu dengan representasi adjacency matrix pula
int parent[sizeG];
// Mengecek asal dari suatu node pada jalur teraugmentasi
bool visited[sizeG];
// untuk utility BFS mengecek apakah telah dikunjungi/belum
int E, V, s, t;
// secara berurutan menyatakan
// jumlah edge, vertex/node, serta posisi source, dan sink
```

Gambar 4.4.1. Kamus global algoritma Edmonds-Karp di C++
(sumber: Dokumen Pribadi)

```cpp
bool augmentedPath(){
  /*Untuk mengecek apakah masih ada jalur terugmentasi atau tidak
  sekaligus menyimpan pathnya di array parent*/
    memset(visited, 0, V);
    memset(parent, -1, sizeof(parent));
    queue <int> Q;

    Q.push(s);
    visited[s] = true;

    while(!Q.empty()){
        int i = Q.front(); Q.pop();
        for(int j=0 ; j<V; j++){
            if(!visited[j] && ResGraf[i][j]>0){
                Q.push(j);
                parent[j] = i;
                visited[j] = true;
            }
        }
    }
    return visited[t];
}
```

Gambar 4.4.2. Fungsi mencari jalur teraugmentasi pada algoritma Edmonds-Karp di C++
(sumber: Dokumen Pribadi)

```cpp
int EdmondsKarp(){
/* Algoritma edmonds-karp/ford-fulkerson yang melakukan iterasi
  dan mengakumulasi nilai dari maxflow pada graf residu
*/
    int totalFlow = 0;
    while(augmentedPath()){
        int bottleneck = INF;
        for (int i = t; i!=s; i = parent[i]){
            int j = parent[i];
            bottleneck = min(bottleneck, ResGraf[j][i]);
        }
        totalFlow += bottleneck;
        for(int i = t; i!=s; i = parent[i]){
            int j = parent[i];
            ResGraf[i][j] += bottleneck;
            ResGraf[j][i] -= bottleneck;
        }
    }
    return totalFlow;
}
```

Gambar 4.4.3. Fungsi utama algoritma Edmonds-Karp di C++

(sumber: Dokumen Pribadi)

```
void printEdgeListRes(){
  // fungsi untuk mencetak ke layar flow dari tiap edge graf hasil
  cout<<"Edge List hasilnya adalah:"<<endl;
  cout<<"Formatnya <node asal> <node tujuan> <flow digunakan>"<<endl;
  for(int i=0;i<V;i++){
    for(int j=0;j<V;j++){
      if(Graf[i][j]!=0){
        cout<<i<<' '<<j<<' '<<ResGraf[j][i]<<endl;
      }
    }
  }
}
```

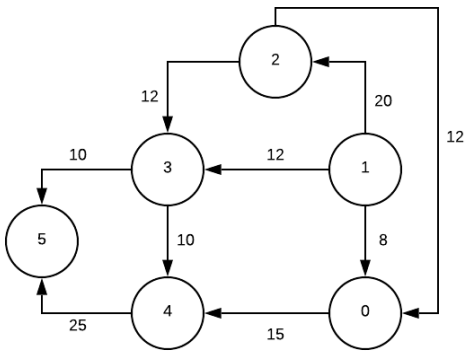
Gambar 4.4.4. Fungsi mencetak Edge List hasil di C++
(sumber: Dokumen Pribadi)

```
int main(){
    cout<<"Masukkan jumlah vertex: "; cin>>V;
    cout<<"Berhasil! node anda diberi indeks 0 sampai "<<V-1<<endl;
    cout<<"Masukkan indeks source: "; cin>>s;
    cout<<"Masukkan indeks sink: "; cin>>t;
    cout<<"Masukkan jumlah edge: "; cin>>E;
    memset(Graf, 0, sizeof(Graf));
    memset(ResGraf, 0, sizeof(ResGraf));
    cout<<"Masukkan pair vertex yang terhubung edge sekaligus kapasitasnya sebanyak "<<E<<endl;
    cout<<"Formatnya <node asal> <node tujuan> <kapasitas>"<<endl;
    for(int i=0;i<E;i++){
        int a, b, val;
        cin>>a>>b>>val;
        Graf[a][b] = val;
        ResGraf[a][b] = val;
    }
    cout<<"Jadi flow maksimumnya adalah "<<EdmondsKarp()<<endl;
    printEdgeListRes();
}
```

Gambar 4.4.5. Main program algoritma Edmonds-Karp di C++
(sumber: Dokumen Pribadi)

E. Pencarian Maxflow Sistem Kanal dengan Algoritma Edmund-Karp

Sekarang kita akan mengimplementasikan algoritma ini ke salah satu pemasalahan real yang dapat dipecahkan dengan adanya algoritma ini. Bayangkan ada kota imajiner dengan sistem kanal bawah tanah yang dimodelkan menjadi flow network berikut



Gambar 4.5.1. Permodelan sistem kanal bawah tanah
(sumber: Dokumen Pribadi)

Pada sistem kanal yang berupa flow network tersebut, node 1 adalah source, yang diasumsikan sebagai sumber pembuangan air (baik dari hujan, sisa rumah tangga, dll) utama yang berada di pusat kota (terlihat dari posisinya yang cukup tengah), dan akan dialirkan ke ujung pembuangan, yakni node sink, yang disimbolkan dengan angka 5. Asumsikan pula edge dari flow network tersebut sebagai suatu kanal dengan kapasitas adalah debit maksimumnya. Jika graf flow network tersebut dibuat menjadi representasi edge list, akan diperoleh representasi seperti ini

```
0 4 15
1 0 8
1 3 12
1 2 10
2 0 12
2 3 12
3 4 10
3 5 10
4 5 25
```

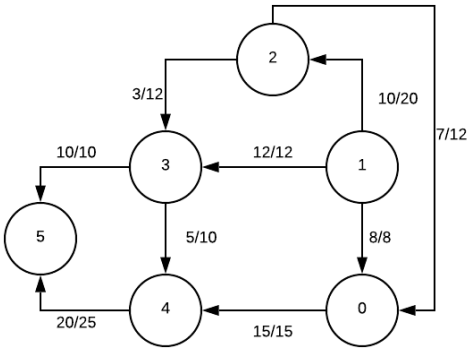
Dimana tiga buah integer dalam satu baris berturut menyatakan <node asal> <node tujuan> <kapasitas>. Setelah dijalankan oleh program didapatkan tampilan

```
Masukkan jumlah vertex: 6
Berhasil! node anda diberi indeks 0 sampai 5
Masukkan indeks source: 1
Masukkan indeks sink: 5
Masukkan jumlah edge: 9
Masukkan pair vertex yang terhubung edge sekaligus kapasitasnya sebanyak 9
Formatnya <node asal> <node tujuan> <kapasitas>
0 4 15
1 0 8
1 3 12
1 2 10
2 0 12
2 3 12
3 4 10
3 5 10
4 5 25
Jadi flow maksimumnya adalah 30
Edge List hasilnya adalah:
Formatnya <node asal> <node tujuan> <flow digunakan>
0 4 15
1 0 8
1 2 10
1 3 12
2 0 7
2 3 3
3 4 5
3 5 10
4 5 20
```

Gambar 4.5.2. Tampilan antarmuka program dan hasilnya
(sumber: Dokumen Pribadi)

Jadi, hasil maxflow atau debit air maksimum yang dapat disalurkan sistem kanal jika dikomputasi menggunakan program sebesar 30.

Edge list dari hasil pemrosesan program saat ditranslasi ke dalam bentuk flow network kembali akan diperoleh hasil sebagai berikut



Gambar 4.5.3. Flow network sistem kanal yang dilengkapi flow hasil pemrosesan algoritma Edmonds-Karp
(sumber: Dokumen Pribadi)

V. Kompleksitas Algoritma

Algoritma Ford-Fulkerson akan mengiterasi terus-menerus flow network hingga tidak ada jalur teraugmentasi tersisa, maka kompeksitas pada worst casenya adalah O(max_flow *BFS_time), dengan asumsi setiap iterasi menambah satu flow pada maxflow dan tiap iterasi memakan waktu satu kali BFS guna mencari jalur teraugmentasi, dan waktu BFS bergantung dari representasi data yang digunakan.

Pada implementasi Ford-Fulkerson menjadi Edmonds-Karp dengan representasi adjacency matriks, diperoleh kompleksitas sebesar $O(EV^3)$ karena iterasi BFS dengan adjacency matriks menghabiskan waktu V^2, sedangkan iterasi dari Edmond-Karp diperkirakan berkisar O(EV), sehingga diperoleh kompleksitas akhir yang masih cukup besar. Kompleksitas ini dapat digunakan dengan penggunaan representasi data yang lain seperti adjacency list, atau menggunakan algoritma maxflow lain, seperti algortima Dinic.

VI. Implementasi dan Manfaat Lain Algoritma Edmonds-Karp

Flow network merupakan salah satu topik yang paling penting dari pembahasan graf, karena implementasinya yang sangat luas, tak terkecuali maxflow dan algoritma Edmonds-Karp. Beberapa implementasi lain yang penulis ketahui diantaranya:

1. Penjadwalan transportasi publik
2. Bipartite matching dalam teori graf yang lain
3. Design survei
4. Teknik segmentasi gambar
5. Pemilihan tim secara optimum
6. Problem eliminasi pertandingan baseball

Dan masih banyak kemungkinan lain yang dapat dieksplor lebih jauh, sehingga dapat menyederhanakan atau bahkan menyelesaikan masalah-masalah lain yang kompleks.

VII. Ucapan Terima Kasih

Penulis mengucap syukur kepada Allah swt. Karena atas limpahan rahmat-Nya penulis diberikan kesempatan untuk menyelesaikan makalah ini. Selain itu penulis juga mengucapkan terima kasih kepada Bapak Rinaldi Munir dan Ibu Harlili sebagai dosen pengajar dan pembimbign mata kuliah IF2120 Matematika Diskrit yang telah memberikan ilmu yang bermanfaat ini selama satu semester ini kepada saya.

References

[1] Sedgewick, Robert, Kevin Wayne, *Algorithms Fourth Edition*. USA, 2011.
[2] Halim, Steven, Felix Halim, *Competitive Programming 3*. Singapore, 2013.
[3] https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/netflow.pdf, diakses pada Rabu - Jumat, 4-6 Desember 2019.
[4] https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI.pdf, diakses pada Rabu - Jumat, 4-6 Desember 2019.
[5] https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/tutorial/, diakses pada Rabu - Jumat, 4-6 Desember 2019.
[6] Kleinberg, Jon, Eva Tardos, *Algorithm Design*. USA, 2009.
[7] Cormen, H. Thomas, dkk., *Introduction to Algorithms*. USA, 2009.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 6 Desember 2019

Naufal Prima Yoriko - 13518146