PROGRESS PEMBELAJARAN TUGAS AKHIR

Raditya Yusuf Annaafi' (5025231033)

Sebagai hasil dari beberapa referensi yang telah dibaca, berikut akan dilakukan percobaan pembuatan solusi dari permasalahan ADACROW dalam bentuk kode CPP.

1. Pseudocode

Pseudocode yang digunakan berasal dari sebuah tesis dengan judul "*Choosing the Efficient Algorithm for Vertex Cover Problem*" (Kumar, 2009).

```
   min-cover (G, k, uncov, opt)
1  begin
2  if k = 0 then {leaf of tree reached?}
3  begin
4  if uncov < opt then {new minimum found?}
5  begin
6  opt := uncov;
7  clear set of stored configurations;
8  end;
9  store configuration;
10 end;
11 if bound condition is true (see text) then
12 return;
13 let i∈ V a vertex marked as free of maximal current degree;
14 mark i as covered;
15 k := k − 1;
16 adjust degrees of all neighbours j of i : d(j) := d(j) − 1;
17 min-cover (G, k, uncov − d(i), opt) {branch into 'left' subtree};
18 mark i as uncovered;
19 k := k + 1;
20 (re)adjust degrees of all neighbours j of i: d(j):= d(j) + 1;
21 min-cover (G, k, uncov, opt) {branch into 'right' subtree};
22 mark i as free;
23 end
```

2. Pembahasan Kode Solusi (CPP)

Alur jalannya kode solusi identik dengan pseudocode. Hanya saja, dalam proses translasi dari pseudocode ke bentuk cpp terdapat beberapa penyesuaian.

2.1.Baris 1-10

Untuk baris 1-10 (mengacu pada pseudocode) bentuknya dalam kode CPP tidak terlalu berbeda.

```
if (k == 0) {
    if (uncov < best) {
        best = uncov;
        res.clear();
        res = cover_set;
```

```
    }

    return;
}
```

2.2.Baris 11-12

Pada baris 11-12 adalah proses pemeriksaan terpenuhi atau tidaknya kondisi *lower bound*. Seperti yang terdapat pada laporan *progress* pertama, pemeriksaan dilakukan dengan membandingkan nilai M dengan jumlah *edge* yang tidak ter-*cover* pada konfigurasi terbaik sejauh ini. Nilai M dapat diperoleh dengan persamaan di bawah:

$$M = \max(0, E - \max(d(v_1) + d(v_2) + \cdots d(v_k)))$$

```
int max_coverable = 0;
vector<int> degrees;
for (int i = 0; i < G.size(); i++) if(status[i]==0) degrees.push_back(G[i].size());
sort(degrees.rbegin(), degrees.rend());
for(int i = 0; i < min(k, (int)degrees.size()); i++) max_coverable += degrees[i];
if(uncov - max_coverable >= best) return;
```

Dilakukan *looping* pada *vertices* yang masih *free* untuk mengambil jumlah *edge* yang terhubung, dan belum ter-*cover*, pada *vertices* tersebut. Kemudian, diurutkan secara menurun agar saat mengambil sebanyak k *vertices* akan diperoleh *vertices* dengan derajat terbesar. Lalu, jika M, dalam kode CPP "uncov – max_coverable", lebih besar atau sama dengan jumlah *edge* yang tidak ter-*cover* pada konfigurasi terbaik sejauh ini, cabang tidak akan ditelusuri lebih lanjut.

2.3.Baris 13

Baris ini bertujuan untuk mencari *vertex* dengan derajat terbesar, yaitu yang paling banyak bersinggungan *edges*. Jika sudah tidak ditemukan adanya *vertex* lagi, penelusuran tidak akan dilanjutkan.

```
int vertex = -1, max_deg = -1;
for (int i = 0; i < G.size(); i++) {
    if (status[i] != 0) continue;

    int deg = G[i].size();
    if (deg > max_deg) {
        max_deg = deg;
        vertex = i;
    }
}

if (vertex == -1) return;
```

2.4.Baris 14-17

Baris ini adalah percabangan pertama dari suatu *node*/*state* dalam *Branch&Bound search tree*.

```
    vector<pair<int,int>> newly_covered;
    for (int neighbor : G[vertex]) {
        if (!edge_covered[vertex][neighbor]) {
            edge_covered[vertex][neighbor] = edge_covered[neighbor][vertex] = true;
            newly_covered.push_back({vertex, neighbor});
        }
    }
    cover_set.push_back(vertex);
    status[vertex] = 1;
    bb(G, k - 1, uncov - newly_covered.size(), best);
    cover_set.pop_back();
    status[vertex] = 0;
```

Penyesuaian derajat pada tetangga-tetangga *vertex* yang telah dipilih pada tahap sebelumnya, sebut saja *vertex* "i", dilakukan dengan menyimpan *state edges* tersebut dalam vector "edge_covered" dan terdapat vector "newly_covered" yang berperan sebagai representasi derajat *vertex* "i".

Terdapat vector "status" yang berfungsi untuk menyimpan kondisi suatu *vertex*. Nilai 0 berarti *vertex* masih *free*, 1 berarti *vertex* ditandai sebagai *cover*, dan 2 berarti *vertex* ditandai sebagai *uncovered*.

Vertex "i" diberi nilai 1 dan dimasukkan ke dalam himpunan *vertex cover*, kemudian dilakukan percabangan seperti yang terdapat pada pseudocode. Yang menjadi sedikit perbedaan adalah k dikurangi dalam pemanggilan fungsi saat membuat percabangan dari suatu *node*, jadi nilai k pada node tersebut tidak berubah.

Terakhir, vertex "i" dikeluarkan dari himpunan *vertex cover* dan nilainya diubah menjadi 0 kembali (*free*).

2.5.Baris 18-21

Baris ini adalah percabangan ke-dua dari suatu *node*/*state* dalam *Branch&Bound search tree*.

```
    for (auto e : newly_covered) edge_covered[e.first][e.second] = edge_covered[e.second][e.first] = false;

    status[vertex] = 2;
    bb(G, k, uncov, best);
    status[vertex] = 0;
```

Penyesuaian kembali derajat pada tetangga-tetangga *vertex* "i" dengan mengembalikan *state edges* yang sebelumnya telah diubah. Pengembalian *state* dilakukan dengan memanfaatkan vector "newly_covered".

Vertex "i" diberi nilai 2, yang artinya *uncovered*, lalu pemanggilan fungsi dilakukan untuk membuat cabang ke-dua, di mana *vertex* "i" tidak ditandai sebagai *vertex cover*.

Terakhir, vertex “i” dikeluarkan dari himpunan *vertex cover* dan nilainya diubah menjadi 0 kembali (*free*).

2.6.Baris 22-23

Pengubahan kembali vertex “i” menjadi *free* telah dilakukan setelah kedua percabangan.

2.7.Keseluruhan Kode

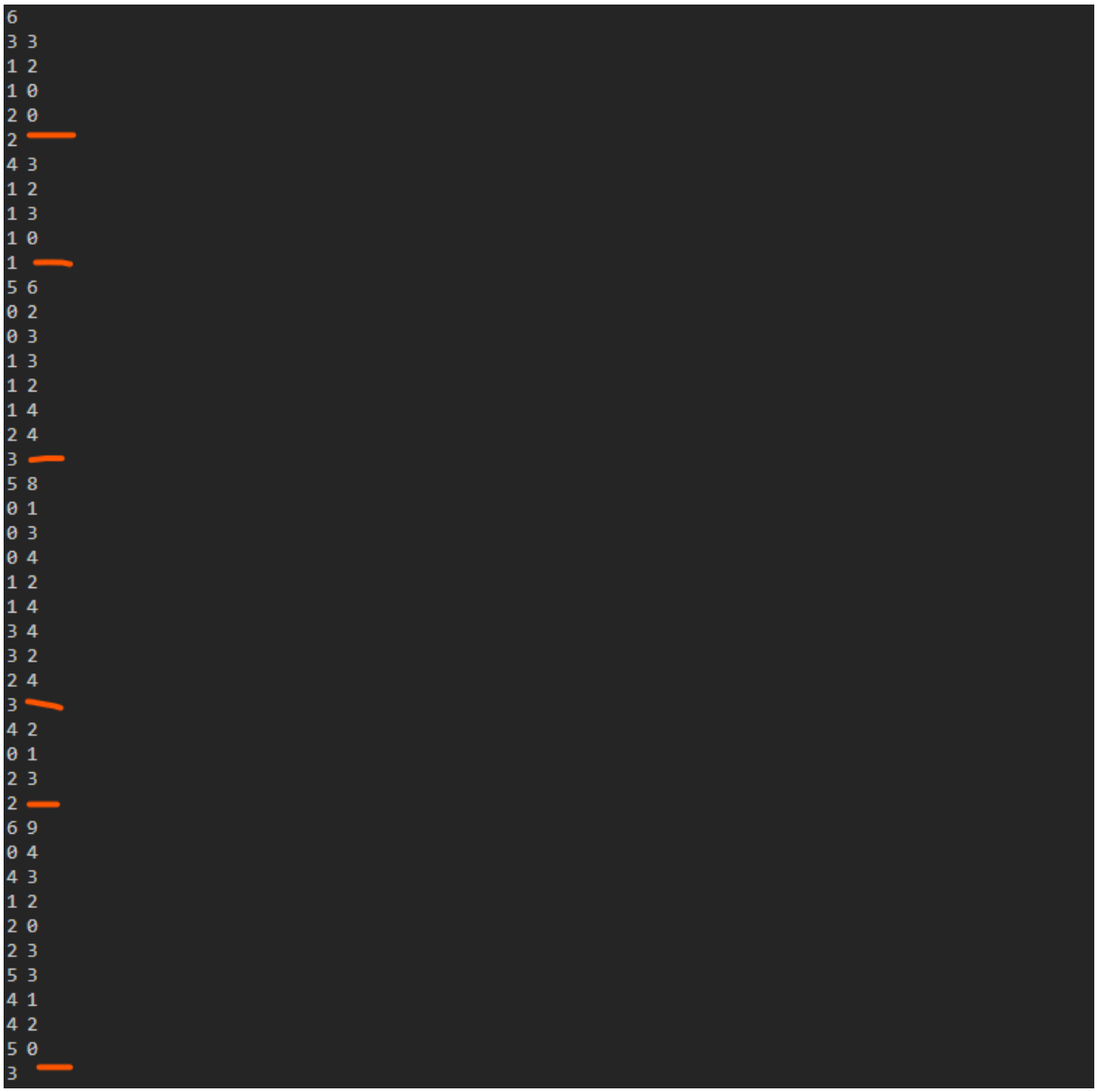
```
#include <stdio.h>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> res;
vector<int> cover_set;
vector<int> status;
bool edge_covered[151][151];

void bb(vector<vector<int>> G, int k, int uncov, int& best) {
    if (k == 0) {
        if (uncov < best) {
            best = uncov;
            res.clear();
            res = cover_set;
        }

        return;
    }
    int max_coverable = 0;
    vector<int> degrees;
    for (int i = 0; i < G.size(); i++) if(status[i]==0)
degrees.push_back(G[i].size());
    sort(degrees.rbegin(), degrees.rend());
    for(int i = 0; i < min(k, (int)degrees.size()); i++) max_coverable
+= degrees[i];
    if(uncov - max_coverable >= best) return;

    int vertex = -1, max_deg = -1;
    for (int i = 0; i < G.size(); i++) {
        if (status[i] != 0) continue;

        int deg = G[i].size();
        if (deg > max_deg) {
            max_deg = deg;
            vertex = i;
        }
    }
```

```
    if (vertex == -1) return;

    vector<pair<int,int>> newly_covered;
    for (int neighbor : G[vertex]) {
        if (!edge_covered[vertex][neighbor]) {
            edge_covered[vertex][neighbor] =
edge_covered[neighbor][vertex] = true;
            newly_covered.push_back({vertex, neighbor});
        }
    }
    cover_set.push_back(vertex);
    status[vertex] = 1;
    bb(G, k - 1, uncov - newly_covered.size(), best);
    cover_set.pop_back();
    status[vertex] = 0;
    for (auto e : newly_covered) edge_covered[e.first][e.second] =
edge_covered[e.second][e.first] = false;

    status[vertex] = 2;
    bb(G, k, uncov, best);
    status[vertex] = 0;

    return;
}

int main() {
    int T;
    scanf("%d", &T);

    while (T--) {
        int M, N;
        scanf("%d %d", &M, &N);
        vector<vector<int>> G(M);
        for (int i = 0; i < N; i++) {
            int u, v;
            scanf("%d %d", &u, &v);
            G[u].push_back(v);
            G[v].push_back(u);
        }
        int best = 1e9;

        status.assign(M, 0);
        for (int k = 1; k <= M; k++) bb(G, k, N, best);

        printf("%d\n", res.size());
        res.clear();
    }
```

```
    return 0;
}
```

3. Hasil

Sesuai dengan target saya saat memulai percobaan pertama ini. Solusi telah berhasil menjawab dua contoh *input* yang terdapat pada laman ADACROW dengan benar



Example Output

```
2
1
3
3
2
3
```

Example Output 2

```
4
9
```

Yang masih menjadi masalah adalah saat di-*submit* pada SPOJ masih mendapatkan hasil "*Time Limit Exceeded*". Hal tersebut menunjukkan bahwa solusi terkini masih belum cukup efisien.

35033620	2025-09-24 03:35:06	033_Raditya_PAA_B	time limit exceeded edit ideone it	-	5.2M	CPP14-CLANG

4. Langkah-Langkah Berikutnya

- Mempelajari teknik *coding* yang efisien.
- Melakukan beberapa penyesuaian ulang pada kode karena setelah ditinjau ulang, masih terdapat langkah-langkah yang redundan.

5. Daftar Pustaka

Kumar, K. V. R. (2009). *Choosing the Efficient Algorithm for Vertex Cover Problem*. https://gdeepak.com/thesisme/thesis-

Choosing%20the%20Efficient%20Algorithm%20for%20Vertex%20Cover%20problem.pdf