Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – IF184802

DESAIN DAN ANALISIS ALGORITMA INVERSE MEDIAN FILTERING DENGAN PENDEKATAN DYNAMIC PROGRAMMING PADA STUDI KASUS: SPOJ 3003 MEDIAN FILTER

YOSHIMA SYACH PUTRI
NRP 05111640000022

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – IF184802

DESAIN DAN ANALISIS ALGORITMA INVERSE MEDIAN FILTERING DENGAN PENDEKATAN DYNAMIC PROGRAMMING PADA STUDI KASUS: SPOJ 3003 MEDIAN FILTER

YOSHIMA SYACH PUTRI
NRP 05111640000022

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

DESIGN AND ANALYSIS OF INVERSE MEDIAN FILTERING ALGORITHMS WITH DYNAMIC PROGRAMMING APPROACH IN CASE STUDY: SPOJ 3003 MEDIAN FILTER

YOSHIMA SYACH PUTRI
NRP 05111640000022

Supervisor I
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMENT OF INFORMATICS
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA *INVERSE MEDIAN FILTERING* DENGAN PENDEKATAN *DYNAMIC PROGRAMMING* PADA STUDI KASUS: SPOJ 3003 MEDIAN FILTER

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
YOSHIMA SYACH PUTRI
NRP. 05111640000022

Disetujui oleh Pembimbing Tugas Akhir:

1. Rully Soelaiman, S.Kom., M.Kom. ..... (Pembimbing 1)
   (NIP. 19700213 199402 1 001)
2. M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. ..... (Pembimbing 2)
   (NIP. 19740209 200212 1 001)

SURABAYA
Januari 2020

(Halaman ini sengaja dikosongkan)

DESAIN DAN ANALISIS ALGORITMA *INVERSE MEDIAN FILTERING* DENGAN PENDEKATAN *DYNAMIC PROGRAMMING* PADA STUDI KASUS: SPOJ 3003 MEDIAN FILTER

Nama Mahasiswa	**: Yoshima Syach Putri**
NRP	**: 05111640000022**
Departemen	**: Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, ITS**
Dosen Pembimbing 1	**: Rully Soelaiman, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.**

Abstrak

Permasalahan Median Filter adalah permasalahan mengenai Inverse Median Filtering atau mengenai pengolahan citra. Tujuan dari permasalahan ini adalah untuk mendapatkan citra asli yang mungkin (possible original image), sehingga akan didapat selisih terbesar dari piksel hitam pada citra terfilter dengan Median Filter dan citra asli yang mungkin.

Dynamic Programming yang merupakan salah satu algoritma pencarian optimal, diimplementasikan untuk menyelesaikan permasalahan di atas dengan memanfaatkan memori untuk mempercepat pencarian.

Dari serangkaian percobaan yang telah dilakukan, diperoleh kesimpulan bahwa algoritma yang dirancang telah dapat diimplementasikan dengan tepat untuk menyelesaikan permasalahan di atas secara optimal.

Kata kunci: Dynamic Programming, Inverse, Median Filter.

(Halaman ini sengaja dikosongkan)

DESIGN AND ANALYSIS OF INVERSE MEDIAN FILTERING ALGORITHMS WITH DYNAMIC PROGRAMMING APPROACH IN CASE STUDY: SPOJ 3003 MEDIAN FILTER

Name	**: Yoshima Syach Putri**
NRP	**: 05111640000022**
Department	**: Informatics, Faculty of Intelligent Electrical and Informatics Technology, ITS**
Supervisor I	**: Rully Soelaiman, S.Kom., M.Kom.**
Supervisor II	**: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.**

Abstract

Median Filter problem is a problem about Inverse Median Filtering or about image processing. The objective of this problem is to get the possible original image so that the biggest difference value of the black pixels between filtered image and the possible original image.

Dynamic Programming, which is one of the optimal search algorithms, has been implemented to solve that problem by utilizing memory to speed up the searching.

From a series of experiments that have been done, it can be concluded that the designed algorithm could be implemented properly to solve that problem optimally.

Keywords: Dynamic Programming, Inverse, Median Filter.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Segala puji syukur bagi Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

DESAIN DAN ANALISIS ALGORITMA *INVERSE MEDIAN FILTERING* DENGAN PENDEKATAN *DYNAMIC PROGRAMMING* PADA STUDI KASUS: SPOJ 3003 MEDIAN FILTER

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan kontribusi bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Kedua orang tua dan keluarga penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri penulis sendiri baik selama penulis menempuh masa perkuliahan maupun pengerjaan Tugas Akhir ini.
2. Bapak Rully Soelaiman, S.Kom., M.Kom. dan Bapak M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. selaku Dosen Pembimbing I dan II yang telah banyak memberikan ilmu, pandangan, nasihat, motivasi, bimbingan, dan arahan selama penulis menempuh masa perkuliahan maupun selama pengerjaan Tugas Akhir ini.
3. Keluarga Besar Laboratorium Komputasi Cerdas & Visi (KCV) yang memberikan penulis tempat bernaung, menemani, membantu, dan menjadi keluarga selama perkuliahan.

4. Seluruh mahasiswa Teknik Informatika ITS angkatan 2016 yang menjadi teman penulis selama menjalani masa perkuliahan.
5. Seluruh dosen, karyawan, dan teknisi yang sudah memberikan ilmu dan mengisi hari penulis selama masa perkuliahan.
6. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari masih ada kekurangan pada Tugas Akhir ini sehingga penulis mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan supaya Tugas Akhir ini menjadi lebih baik. Semoga melalui Tugas Akhir ini penulis dapat memberikan manfaat untuk pembaca.

Surabaya, Januari 2020

Yoshima Syach Putri

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB 1
PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan tugas akhir.

1.1 Latar Belakang

Saat ini, sudah banyak jenis algoritma yang dapat digunakan untuk menyelesaikan suatu masalah terutama pada bidang ilmu komputer. Dalam pemecahan suatu masalah, penggunaan memori dan waktu eksekusi diharapkan dapat mencapai titik optimal. Pada salah satu situs pemrograman *online* yaitu SPOJ, terdapat sebuah permasalahan yang dikenal dengan nama *Median Filter* [1], yang bisa digunakan untuk menguji pemecahan masalah menggunakan algoritma yang paling optimal.

Permasalahan *Median Filter* [1] berawal dari adanya citra yang sudah mengalami *median filter*. Sedangkan citra asli sebelum mengalami *median filter* sudah tidak ada atau dianggap hilang. Tujuan dari permasalahan ini adalah untuk bisa mendapatkan selisih maksimal piksel hitam dari citra *median filter* dengan citra citra asli yang mungkin (*possible original image*. Sehingga untuk mendapatkan dugaan citra asli, maka solusi dari pemecahan masalah tersebut adalah mencari berbagai citra asli yang mungkin (*possible original image*) dengan melakukan metode *inverse.*

Pada permasalahan tersebut, diberikan *input* yang mengandung beberapa *test* case. Baris pertama pada masing-masing *test* case berisi dua bilangan bulat *W* dan *H* yang menunjukkan lebar dan tinggi dari citra. Kemudian diikuti baris sejumlah *H* sebagai penggambaran matriks dari citra yang sudah mengalami *median filter*. Baris ke-*i* merepresentasikan *input* yang berisi karakter sejumlah *W*, yang masing-masing adalah karakter '#' (mewakili piksel hitam) dan '.' (mewakili piksel putih). Lalu dari citra *input* tersebut akan dicari berbagai citra asli yang mungkin (*possible original image*) sebelum mengalami *median*

filter (citra *inverse median filter*). Hasil yang diharapkan dari permasalahan tersebut adalah mendapatkan selisih (*different value*) maksimal dari piksel hitam pada citra *input* dengan citra *inverse median filter*.

Pada tugas akhir ini penulis menggunakan metode *Dynamic Programming* untuk menyelesaikan permasalahan di atas. Hasil dari tugas akhir ini diharapkan dapat memberikan gambaran mengenai kinerja *Dynamic Programming* untuk menyelesaikan permasalahan *Inverse Median Filter* secara efektif dan efisien serta dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana desain dan analisis algoritma yang digunakan untuk menyelesaikan permasalahan *inverse median filte*r?
2. Bagaimana implementasi algoritma *dynamic programming* untuk menyelesaikan permasalahan *inverse median filte*r?
3. Bagaimana kinerja algoritma *dynamic programming* untuk menyelesaikan permasalahan *inverse median filte*r berdasarkan waktu dan memori yang dibutuhkan?

1.3 Batasan Permasalahan

Adapun batasan SPOJ untuk Tugas Akhir ini adalah sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas ukuran *file input* maksimal 50 KB.
3. Batas angka untuk setiap *test case* adalah *W* dan *H* dengan syarat $1 \leq W, H \leq 8$.
4. Batas waktu eksekusi program adalah di bawah 3.567 detik.

5. Batas maksimum memori yang digunakan adalah di bawah 1536 MB.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Memudahkan pencarian citra asli yang mungkin pada citra yang sudah mengalami *Median Filter*.
2. Menyelesaikan permasalahan citra dengan pendekatan *Dynamic Programming*.
3. Mendapatkan nilai kompleksitas paling rendah dan paling optimal.

1.5 Manfaat

Tugas Akhir ini diharapkan dapat membantu memberikan pemahaman mengenai algoritma optimasi *Dynamic Programming* untuk menyelesaikan permasalahan di atas dan mendapatkan selisih maksimal piksel hitam dari citra *input* dan citra asli yang mungkin (*possible original image*) dengan optimal.

1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan proposal Tugas Akhir yang berisi pendahuluan, deskripsi dan gagasan metode dan analisis algoritma sebagai solusi optimal dari permasalahan *Inverse Median Filter*. Pendahuluan ini terdiri dari latar belakang diajukannya Tugas Akhir, rumusan masalah dan batasan masalah yang ditetapkan, serta manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula subbab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini, dilakukan pencarian informasi dan studi tentang metode penyelesaian masalah yang terkait dengan permasalahan *Inverse Median Filter*. Materi-materi tersebut didapatkan dari buku, *internet*, *paper* dan jurnal ilmiah, serta materi perkuliahan yang terkait. Luaran dari tahap ini adalah daftar pustaka dan referensi.

1.6.3 Desain dan Perancangan

Pada tahap ini, dilakukan desain rancangan algoritma untuk memecahkan permasalahan *Inverse Median Filter*. Luaran dari tahap ini adalah algoritma yang digunakan sebagai solusi optimal pada permasalahan SPOJ *Median Filter* [1].

1.6.4 Implementasi Algoritma

Pada tahap ini akan dilakukan implementasi metode dan algoritma yang telah direncanakan. Luaran dari tahap ini adalah implementasi algoritma yang dibangun menggunakan bahasa pemrograman C++ dengan menggunakan IDE Dev-C++.

1.6.5 Uji Coba dan Evaluasi

Pada tahap ini, dilakukan pengujian program yang telah dibuat untuk menyelesaikan permasalahan *Inverse Median Filter*. Pengujian dan evaluasi dilakukan hingga mendapatkan hasil *accepted* dari situs pemrograman *online* SPOJ. Luaran dari tahap ini adalah status *accepted* dari situs pemrograman *online* SPOJ.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

BAB 2 DASAR TEORI

Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang teori *Median Filter* dan *Dynamic Programming*.

BAB 3 DESAIN DAN ANALISIS

Bab ini berisi pembahasan mengenai desain dan analisis algoritma yang digunakan dalam strategi penyelesaian permasalahan.

BAB 4 IMPLEMENTASI

Bab ini membahas implementasi dari desain algoritma yang telah dibuat pada bab 3. Penjelasan kode yang digunakan untuk proses implementasi jug adijabarkan di bab ini.

BAB 5 UJI COBA DAN EVALUASI

Bab ini membahas tahapan uji coba, kemudian hasil uji coba dievaluasi terhadap kinerja dari algoritma yang dibangun.

BAB 6 KESIMPULAN DAN SARAN

Bab ini merupakan penyampaian kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

(Halaman ini sengaja dikosongkan)

BAB 2
DASAR TEORI

Bab ini membahas mengenai permasalahan SPOJ *Median Filter* [1] dan landasan teori yang digunakan untuk menyelesaikan pengerjaan Tugas Akhir ini. Teori yang digunakan diantaranya adalah *Median Filter, Dynamic Programming*, dan beberapa teori dan fungsi lain yang mendukung pembuatan Tugas Akhir.

2.1 Deskripsi Umum Permasalahan

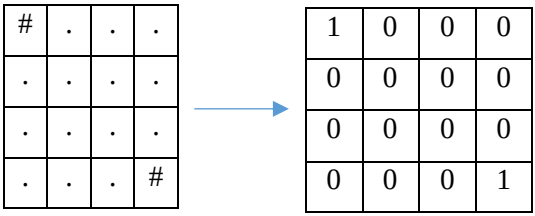
Pada permasalahan SPOJ *Median Filter* [1], diberikan *input* yang berisi beberapa *test case*. Baris pertama pada masing-masing *test case* berisi dua buah bilangan bulat *W* dan *H*. *W* adalah lebar dan *H* adalah tinggi dari citra. Kemudian diikuti baris sejumlah *H* sebagai penggambaran matriks dari citra yang dianggap sudah mengalami *median filter*. Setiap baris merepresentasikan *input* yang berisi karakter sebanyak *W*, yang masing-masing adalah karakter '#' (mewakili piksel hitam) atau '.' (mewakili piksel putih). Lalu dari citra *input* tersebut akan dicari berbagai citra asli yang mungkin (*possible original image*). Hasil yang diharapkan dari permasalahan SPOJ *Median Filter* [1] adalah mendapatkan selisih maksimal piksel hitam dari citra *input* dan citra asli yang mungkin.

```
4 4
#...
....
....
...#
```

Gambar 2.1 Contoh *Test Case* pada Permasalahan *Median Filter*

Sebagai contoh, diberikan matriks 4x4 seperti pada Gambar 2.1, terdapat 4 baris dengan 4 karakter pada setiap barisnya. Karakter yang terdapat pada contoh tersebut merepresentasikan piksel berwarna hitam dan putih. Oleh karena itu, jika contoh *test*

case diubah menjadi matriks biner, maka akan dihasilkan matriks seperti pada Gambar 2.2.



Gambar 2.2 Pengubahan Contoh *Test Case* ke Matriks Biner

Dari matriks citra *input* seperti di atas, apabila dilakukan pengembalian citra dengan metode *inverse*, maka harus dilakukan secara rekursif karena perubahan nilai dari satu piksel dapat mempengaruhi nilai piksel yang lainnya. Permasalahan ini adalah suatu permasalahan optimasi, dimana tujuan dari permasalahan ini adalah mencari selisih maksimal dari semua kemungkinan yang ada. Pendekatan dengan metode *Dynamic Programming* dapat menyelesaikan permasalahan ini karena kriteria pada permasalahan ini adalah adanya submasalah optimal dan tumpang tindih, yang artinya suatu submasalah dapat membantu submasalah yang lain.

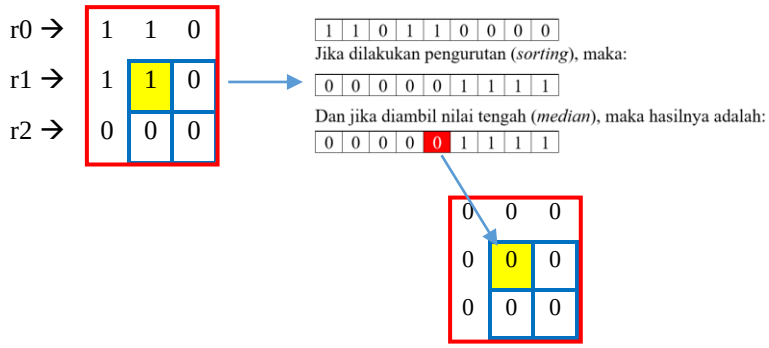
Dengan pendekatan *Dynamic Programming*, penghitungan kemungkinan dimulai dari baris pertama untuk karakter pertama hingga karakter terakhir. Kemudian nilai dari kemungkinan submasalah tersebut disimpan dalam sebuah *array*. Pada submasalah selanjutnya akan dilakukan penghitungan kemungkinan dimulai dari baris tersebut untuk karakter pertama hingga karakter terakhir seperti penghitungan submasalah sebelumnya. Perbedaannya adalah dalam penyimpanan nilai penghitungan. Submasalah berikutnya akan menyimpan pada *array* yang sama, dengan mempertimbangkan nilai sebelumnya. Jadi hasil dari keseluruhan submasalah akan diakumulasi untuk mendapatkan nilai kemungkinan yang mendekati benar.

2.2 Deskripsi Umum Teori

Pada subbab ini, akan dijelaskan berbagai landasan teori secara umum yang digunakan untuk melakukan pendekatan terhadap penyelesaian permasalahan.

2.2.1 Median Filter

Median Filter merupakan salah satu teknik peningkatan kualitas citra dalam domain spasial. Metode ini termasuk dalam kategori *non-linear filtering*. Cara kerjanya hampir sama dengan *mean filtering*. Pada *mean filtering* dalam setiap piksel *output* diatur ke tingkat rata-rata dari nilai-nilai piksel dalam *mask* yang ditentukan. Namun, dengan *median filter*, nilai piksel *output* ditentukan oleh nilai median dari lingkungan *mask* yang ditetapkan. Nilai median dicari dengan melakukan pengurutan terhadap nilai piksel kemudian dicari nilai tengahnya. Jika rentang nilai memiliki jumlah piksel genap, maka rata-rata dari dua nilai piksel tengah yang akan digunakan [2]. Ilustrasi *median filter* dapat dilihat pada Gambar 2.3. Maka hasil dari *median filter* adalah:



Gambar 2.3 Proses *Median Filter*

Apabila dikaitkan dengan permasalahan *Inverse Median Filter*, maka untuk mendapatkan kemungkinan nilai piksel yang benar, dilakukan pembuktian pada saat menduga nilai piksel citra asli. Dimana berbagai kemungkinan tersebut akan diubah kembali

dengan *median filtering*. Jika hasil nilai piksel setelah diubah kembali dengan *median filter* adalah sama dengan nilai piksel citra masukan, maka nilai *inverse median filter* tersebut dianggap benar.

2.2.2 Dynamic Programming

Dynamic Programming adalah metode optimasi yang berbeda dari rekursi (perulangan) biasa. Sebagian besar permasalahan yang membutuhkan penyelesaian dengan *Dynamic Programming* dapat dikategorikan dalam dua jenis:

1. *Optimization problems*
2. *Combinatorial problems*

Optimization problems mengharapkan kita untuk memilih solusi yang memungkinkan, sehingga nilai dari fungsi yang dibutuhkan dapat diminimalkan atau dimaksimalkan. *Combinatorial problems* mengharapkan kita untuk mencari tahu berbagai macam cara untuk melakukan sesuatu atau dapat dikatakan kemungkinan dari beberapa proses yang bisa terjadi.

Biasanya implementasi dari *Dynamic Programming* adalah untuk *optimization problems*. Masalah seperti itu dapat memiliki banyak solusi yang memungkinkan, dan setiap solusi memiliki nilai. Tujuannya adalah untuk menemukan solusi dengan nilai optimal (minimal atau maksimal). Saat pengembangan algoritma *Dynamic Programming*, terdapat empat langkah proses berdasarkan definisi Cormen [3]:

1. Melakukan identifikasi karakteristik struktur sebuah solusi yang optimal.
2. Menentukan nilai solusi optimal secara rekursif (berulang).
3. Menghitung nilai solusi optimal, biasanya dengan cara *bottom-up*.
4. Membangun solusi optimal dari informasi yang telah dihitung.

Apabila dikaitkan dengan permasalahan *Median Filter* [1], *Dynamic Programming* ini dapat membantu proses perulangan dalam pengimplementasian fungsi *Inverse Median Filtering*. Fungsi *inverse* dijalankan pada setiap piksel di setiap baris. Pendekatan ini nantinya dapat membantu pencarian nilai yang optimal untuk masing-masing submasalah.

(Halaman ini sengaja dikosongkan)

BAB 3
DESAIN DAN ANALISIS

Pada bab ini akan dijelaskan mengenai desain dan analisis algoritma yang digunakan dalam strategi penyelesaian permasalahan SPOJ *Median Filter* [1].

Berdasarkan deskripsi umum permasalahan yang sudah dijelaskan pada bab 2, maka proses analisis algoritma menggunakan metode *dynamic programming* yang akan digunakan sebagai pencarian nilai kemungkinan. Metode tersebut akan dirinci menjadi tiga submasalah agar sesuai dengan sifat citra yang dibutuhkan. Submasalah tersebut akan membantu submasalah yang lain untuk mendapatkan akumulasi nilai maksimal dari piksel hitam yang dibutuhkan. Detil analisis dari setiap submasalah pada pendekatan *dynamic programming* dapat dilihat pada subbab berikut.

3.1 Analisis Submasalah Optimal

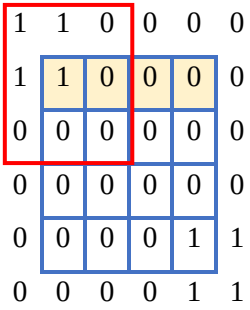
Pada subbab ini, akan dijelaskan mengenai submasalah yang dapat membantu penghitungan kemungkinan nilai. Nilai yang dihasilkan dari submasalah tersebut akan digunakan untuk pengolahan nilai pada submasalah selanjutnya sehingga akan didapatkan nilai solusi yang tepat pada hasil akhir.

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

Gambar 3.1 Tiga Tipe Submasalah

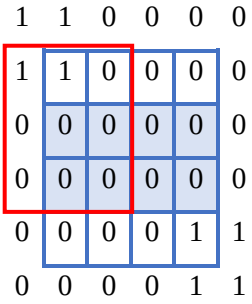
Diberikan *test case* yang sama, seperti pada Gambar 2.1. Hasil akhir yang dicari adalah selisih maksimal dari citra masukan

dengan citra asli yang mungkin. Dari semua kemungkinan nilai citra asli yang mungkin, secara garis besar terdapat tiga tipe hasil akhir dari setiap submasalah. Tiga tipe ini dapat diklasifikasikan lebih detil pada Gambar 3.1.



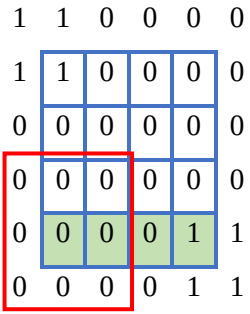
Gambar 3.2 Tipe Submasalah Pertama

Tipe pertama adalah piksel pada baris pertama, karena pada baris pertama dibutuhkan nilai piksel untuk ketetanggaan bagian atas seperti pada ilustrasi Gambar 3.2. Sifat dari *Median Filter* untuk penghitungan ketetanggaan pada piksel tepi terbagi menjadi tiga, yaitu *zero-padding*, *one-padding*, dan pencerminan piksel tepi itu sendiri. Pada permasalahan ini, pencerminan piksel tepi adalah pilihan sifat yang paling mendekati nilai kebenaran. Karena dengan pencerminan, maka nilai piksel tepi tidak lebih condong ke nilai piksel hitam ataupun piksel putih. Hal ini dapat membantu untuk mendapatkan selisih maksimal pada hasil akhir.



Gambar 3.3 Tipe Submasalah Kedua

Tipe kedua adalah piksel pada baris tengah, karena pada baris tengah tidak membutuhkan nilai piksel tambahan untuk ketetanggaannya seperti pada Gambar 3.3. Pada baris tengah, ketetanggaan bagian atas maupun bawah sudah terdapat inisialisasi nilai piksel. Sehingga pemrosesan nilai piksel pada baris tengah akan berbeda dengan baris pertama dan terakhir.



Gambar 3.4 Tipe Submasalah Ketiga

Tipe ketiga adalah piksel pada baris terakhir. Hampir sama seperti tipe pertama, piksel pada baris ini membutuhkan pencerminan nilai ketetanggaan. Perbedaan dari tipe pertama adalah dari posisi pencerminannya. Tipe ketiga membutuhkan pencerminan dari ketetanggaan bagian bawah seperti pada Gambar 3.4.

Dari setiap kandidat hasil akhir pada permasalahan yang diilustrasikan pada Gambar 3.1, semua berawal dari kondisi dimana dibutuhkan nilai piksel dari ketetanggaannya. Dari kondisi awal ini hingga dicapai kondisi yang diminta oleh tujuan, ada beberapa nilai yang harus dicatat pada setiap percobaan kemungkinan sehingga perhitungan menjadi efisien.

3.2 Pemodelan Relasi Rekuren

Untuk penyelesaian submasalah berdasarkan Gambar 3.1, maka dibutuhkan algoritma rekursif yang dapat membantu dalam penghitungan kemungkinan nilai akhir yang akan dimasukkan ke *array*. Pada iterasi setiap piksel akan dicari kemungkinan nilai terkecil dan terbesar untuk dapat digunakan sebagai penentuan nilai selisih. Hal ini berarti membutuhkan *array* dengan masing-masing kolomnya memiliki dua nilai yaitu *lower bound* dan *upper bound*. *Lower bound* disini adalah nilai terkecil dari kemungkinan jumlah piksel hitam hasil *inverse median filter*. Sedangkan *upper bound* adalah nilai terbesar dari kemungkinan jumlah piksel hitam hasil *inverse median filter*. Inisial dari nilai terkecil dan terbesar adalah *ldp* atau *lower difference pixel* dan *udp* atau *upper difference pixel*.

Piksel pada suatu baris memiliki beberapa kemungkinan yang disimpan dalam *array*. Piksel ke-1 mempunyai kemungkinan $2^1 * 2^1$. Piksel ke-2 mempunyai kemungkinan $2^2 * 2^2$, dan seterusnya. Itu berarti piksel ke-*x* mempunyai kemungkinan $2^x * 2^x$. Artinya kemungkinan total yang didapat pada setiap baris pada *test case* adalah $2^W * 2^W$. Relasi rekuren yang dibutuhkan akan direpresentasikan dengan nilai *ldp* dan *udp*. Sebuah nilai *ldp* dan *udp* dapat dibentuk dari jumlah piksel hitam yang mungkin yang berasal dari *inverse median filter* yang dapat dilihat pada persamaan 3.1. Sebuah nilai *ldp* dan *udp* juga dapat dibentuk dari pencarian nilai minimal dan maksimal antara nilai *ldp* dan *udp* sebelumnya ditambah dengan jumlah piksel hitam dari hasil *inverse median filter* pada iterasi tersebut yang dapat dilihat pada persamaan 3.2.

$$ldp_{(N,i,j)} = b \tag{3.1}$$

$$ldp_{(N,i,j)} = ldp_{(N-1,i,j)} + b \tag{3.2}$$

Sehingga penghitungan selisih piksel hitam dapat dilihat pada persamaan 3.3 dan relasi rekuren untuk setiap submasalah dapat dilihat pada persamaan 3.6 atau 3.8 untuk *ldp* dan persamaan 3.7 atau 3.9 untuk *udp* dengan notasi sebagai berikut:

$diff$ = selisih piksel hitam yang mungkin antara citra *median filter* dan citra *inverse median filter*
$ldp_{(N,i,j)}$ = nilai piksel hitam terendah pada iterasi ke (i, j) di baris ke-N
$udp_{(N,i,j)}$ = nilai piksel hitam tertinggi pada iterasi ke (i, j) di baris ke-N
b_k = jumlah piksel hitam pada baris ke-k

$$diff_{(N,i,j)} = udp_{(N,i,j)} - ldp_{(N,i,j)} \tag{3.3}$$

$$lower = ldp_{(N-1,i,j)} + b \tag{3.4}$$

$$upper = udp_{(N-1,i,j)} + b \tag{3.5}$$

$$ldp_{(N,i,j)} = \begin{cases} b, & N = 0 \\ min(minval, lower), & N = H - 1 \\ min\left(ldp_{(N,i,j)}, lower\right), & otherwise \end{cases} \tag{3.6}$$

$$udp_{(N,i,j)} = \begin{cases} b, & N = 0 \\ max(maxval, upper), & N = H - 1 \\ max\left(udp_{(N,i,j)}, upper\right), & otherwise \end{cases} \tag{3.7}$$

$$ldp_{(N,i,j)} = min\left(ldp_{(N,i,j)}, ldp_{(N-1,i,j)} + b\right) \tag{3.8}$$

$$udp_{(N,i,j)} = max\left(udp_{(N,i,j)}, udp_{(N-1,i,j)} + b\right) \tag{3.9}$$

Pada setiap submasalah, yang berawal dari kebutuhan nilai ketetanggaan hingga mencapai nilai tujuan, memiliki beberapa tahap pengolahan nilai. Secara umum, *pseudocode* tahap pengolahan nilai submasalah tipe pertama dapat dilihat pada Gambar 3.5. Perbedaan dengan tipe kedua dan ketiga hanya pada inisialisasi nilai ketetanggaan, yang pada *pseudocode* ini dilambangkan dengan *r0*, *r1*, dan *r2*. Dan pada submasalah tipe kedua, akan dilakukan iterasi sebanyak baris yang ada di baris tengah. Detil submasalah akan dijelaskan di subbab 3.3 pada bagian strategi penyelesaian permasalahan.

dp ()	
1.	$W \leftarrow width\ of\ image$
2.	$H \leftarrow high\ of\ image$
3.	$mask = 2^W - 1$
4.	$for\ i \leftarrow 0\ to\ 2^W - 1$
5.	$r0 = r1 \leftarrow interp(i, W)$
6.	$numblack \leftarrow builtin\_popcount(i)$
7.	$for\ j \leftarrow 0\ to\ 2^W - 1$
8.	$c \leftarrow 0$
9.	$r2 \leftarrow interp(j, W)$
10.	$c \leftarrow c + table[(r0 \gg 0)\ \&\ mask]$
11.	$c \leftarrow c + table[(r0 \gg 1)\ \&\ mask]$
12.	$c \leftarrow c + table[(r0 \gg 2)\ \&\ mask]$
13.	$c \leftarrow c + table[(r1 \gg 0)\ \&\ mask]$
14.	$c \leftarrow c + table[(r1 \gg 1)\ \&\ mask]$
15.	$c \leftarrow c + table[(r1 \gg 2)\ \&\ mask]$
16.	$c \leftarrow c + table[(r2 \gg 0)\ \&\ mask]$
17.	$c \leftarrow c + table[(r2 \gg 1)\ \&\ mask]$
18.	$c \leftarrow c + table[(r2 \gg 2)\ \&\ mask]$
19.	$c \leftarrow (\mathrm{c} + 0\mathrm{x}33333333)\ \&\ 0\mathrm{x}88888888) \gg 3$
20.	$if(c = table[image[0]])$
21.	$ldp[0][i][j] = udp[0][i][j] \leftarrow numblack$

Gambar 3.5 *Pseudocode* Submasalah Tipe Pertama

3.3 Strategi Penyelesaian Permasalahan dengan Pendekatan *Dynamic Programming*

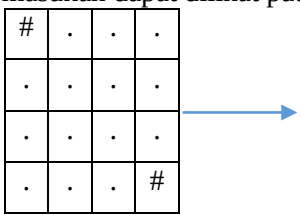
Pada subbab ini, akan dijelaskan mengenai strategi dan rancangan pembuatan program dari awal hingga mendapatkan nilai tujuan sebagai bentuk penyelesaian permasalahan SPOJ *Median Filter* [1].

3.3.1 Tahap Inisialisasi dan *Input*

Strategi penyelesaian diawali dengan inisialisasi tabel citra dari 0 sampai 255 dengan fungsi `init_table()`. Tujuannya adalah untuk merepresentasikan bit kecil menjadi besar (Misal: 1111 menjadi 1000100010001). Setelah inisialisasi selesai, selanjutnya program akan menerima masukan berupa *input* angka *W* dan *H*. Kemudian untuk setiap baris akan berisi piksel sebanyak *W* karakter. Contoh *input* dapat dilihat pada Gambar 2.1. Selanjutnya akan dijalankan fungsi `init_val()` sebagai inisialisasi nilai *lower bound* dan *upper bound*. Nilai ini nantinya akan digunakan untuk menentukan selisih terbesar dari piksel hitam.

3.3.2 Tahap Preprocessing

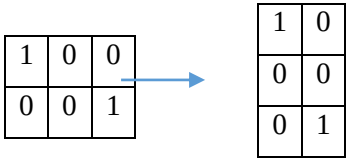
Strategi berikutnya adalah melewati tahap *preprocessing*. Pada tahap ini, program akan menjalankan fungsi `to_binary()` yang digunakan untuk dilakukan *shift bit* ke kiri dari hasil piksel sebelumnya dan merubah karakter-*W* menjadi bilangan biner (1 atau 0 sesuai dengan piksel tersebut). Bilangan biner 1 akan merepresentasikan piksel hitam dan bilangan biner 0 akan merepresentasikan piksel putih. Tujuannya adalah untuk memudahkan proses komputasi selanjutnya. Contoh perubahan dari citra masukan dapat dilihat pada Gambar 3.6.



0001	0001	0001	0001
0000	0000	0000	0000
0000	0000	0000	0000
0000	0000	0000	1000

Gambar 3.6 Perubahan Citra Masukan ke Bilangan Biner yang di-*Shift Bit*

Jika angka masukan *W* melebihi angka *H*, maka sebelum pengubahan biner, program menjalankan fungsi `transpose()` yang bertujuan untuk menukar dimensi *W* dan *H*. Contoh penukaran dimensi ini dapat dilihat pada Gambar 3.7.



Gambar 3.7 Penukaran Dimensi Citra Apabila $W > H$

3.3.3 Tahap Dynamic Programming

Strategi berikutnya adalah tahap *Dynamic Programming* yang terbagi menjadi tiga fungsi sesuai dengan tipe submasalah yang sudah dijelaskan pada subbab 3.1. Tiga fungsi tersebut adalah `dp1()`, `dp2()`, dan `dp3()`. Ketiga fungsi tersebut memiliki tujuan yang sama, yaitu mencari *lower bound* dan *upper bound* pada piksel di baris tertentu. Perbedaan ketiga fungsi tersebut hanyalah pada iterasi dan inisialisasi nilai ketetanggaannya. Ketiga fungsi *Dynamic Programming* tersebut akan dibantu oleh fungsi `interp()` yang bertujuan untuk memberikan *random-sample selection* piksel dari citra atau *image resampling*. Hasil dari fungsi `interp()` apabila contoh *test case* sama dengan Gambar 2.1, maka hasil fungsi `interp()` dengan $w \leftarrow 4$ dapat dilihat pada Tabel 3.1.

Dimana:

- $bits \leftarrow (i \ll 1) \mid (i \,\&\, 1)$
- $v \leftarrow 1 \ll w$
- $res \leftarrow bits \,\&\, v$

Tabel 3.1 Hasil Fungsi `interp()`

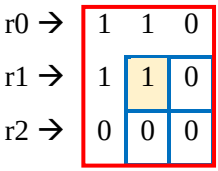
	i	$bits$	v	$res \leftarrow res \ll 1$	$interp \leftarrow bits \mid res$
0	0000	00000	10000	000000	000000
1	0001	00011	10000	000000	000011
2	0010	00100	10000	000000	000100
3	0011	00111	10000	000000	000111
4	0100	01000	10000	000000	001000
5	0101	01011	10000	000000	001011
6	0110	01100	10000	000000	001100
7	0111	01111	10000	000000	001111
8	1000	10000	10000	100000	110000
9	1001	10011	10000	100000	110011
10	1010	10100	10000	100000	110100
11	1011	10111	10000	100000	110111
12	1100	11000	10000	100000	111000
13	1101	11011	10000	100000	111011
14	1110	11100	10000	100000	111100
15	1111	11111	10000	100000	111111

Penjelasan yang lebih lengkap tentang tiga fungsi *Dynamic Programming*, dapat dilihat di bawah ini.

3.3.3.1 Fungsi *DP1*

Fungsi `dp1()` bertujuan untuk memproses baris ke-0. Piksel pada baris ini tidak memiliki tetangga bagian atas dan sifat dari *median filter* adalah ketetanggaan yang kosong akan diisi oleh 0 (*zero-padding*), 1 (*one-padding*), atau cerminan dari piksel tepi. Pada kasus ini, ketetanggaan yang cocok untuk mengisi piksel yang kosong adalah cerminan dari piksel tepi. Setiap piksel nanti akan dipetakan 3x3 untuk dihitung kemungkinan piksel citra asli dari piksel masukan (yang dianggap sudah mengalami *median filter*). Setiap kemungkinan piksel citra asli akan dihitung dari *mask* yang ditentukan. *Mask* 3x3 ini direpresentasikan oleh variabel *r0*, *r1*, dan

r2. Dimana *r0* adalah *mask* baris ke-0, *r1* adalah *mask* baris ke-1, dan *r2* adalah *mask* baris ke-2. Visualisasi *masking* dapat dilihat pada Gambar 3.8.



Gambar 3.8 *Masking* Setiap Piksel

Pada tahap ini, setiap kemungkinan piksel citra asli akan dilakukan konfirmasi, apakah hasil penjumlahan piksel setelah *masking* dan normalisasi yang dapat dilihat pada persamaan 3.11 sama dengan piksel citra masukan (yang sudah mengalami *median filter*). Contoh hasil normalisasi dari variabel *c* setelah dijumlahkan dapat dilihat pada Tabel 3.2.

$$c_{(i,j)} = normalize(\sum_{r=0}^{8} table[r]) \qquad (3.11)$$

Tabel 3.2 Normalisasi Variabel c

c	Hasil Normalisasi
0000000000000000	0000000000000000
0001001001000101	0000000000000001
0001001001010111	0000000000010001
0000000000010010	0000000000000000
0011010001010110	0000000000010001
0011010101100111	0000000100010001
0011010101111001	0000000100010001

Jika benar, maka akan dihitung *numblack* (banyaknya piksel hitam) pada iterasi saat itu dan hasil *numblack* akan disimpan ke *array* $2^W * 2^W$ sebagai nilai *lower bound* dan *upper bound*. Nilai ini nanti yang akan dipakai kembali oleh fungsi *DP2*. Contoh *array* `dp1()` yang berisi kemungkinan *lower bound* dan *upper bound* dari piksel baris ke-0 dapat dilihat pada Tabel 3.3.

Tabel 3.3 *Array DP1*: Baris ke-0

i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0001	-1 17	1 1	1 1	1 1	-1 17	1 1	1 1	-1 17	-1 17	1 1	1 1	1 1	-1 17	1 1	1 1	-1 17
0010	-1 17	-1 17	-1 17	1 1	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	1 1	-1 17	-1 17	-1 17	-1 17
0011	2 2	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	2 2	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0100	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0101	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0110	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0111	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1000	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1001	-1 17	2 2	2 2	2 2	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1010	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1011	3 3	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1100	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1101	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1110	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1111	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17

Nilai yang berada di atas menunjukkan *upper bound* dan nilai yang berada di bawah menunjukkan *lower bound*. Nilai *default* yang diberikan apabila kemungkinan yang dihasilkan tidak sama dengan piksel citra masukan adalah $upper\ bound = -1$ dan $lower\ bound = H * W + 1$.

3.3.3.2 Fungsi *DP2*

Pada fungsi *DP2*, alur program hampir sama seperti fungsi `dp1()`, hanya saja untuk `dp2()` akan memproses piksel pada baris ke-1 sampai baris ke-(*H-2*). Perbedaan dari `dp1()` adalah pada nilai tepinya, dimana pada `dp2()` tidak memerlukan nilai pencerminan dari ketetanggaan, karena semua piksel tepi dari baris ke-1 sampai ke-(*H-2*) sudah memiliki nilai. Contoh *array* `dp2()` yang berisi kemungkinan *lower bound* dan *upper bound* dari piksel baris ke-1 sampai piksel baris ke-(*H-2*) dapat dilihat pada Tabel 3.4 untuk baris ke-1 dan Tabel 3.5 untuk baris ke-2.

Tabel 3.4 *Array DP2*: Baris ke-1

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	2 2	-1 17	2 2	-1 17
0001	3 2	-1 17	-1 17	-1 17	3 2	-1 17	-1 17	-1 17	3 2	-1 17	-1 17	-1 17	2 2	-1 17	-1 17	-1 17
0010	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	3 2	-1 17	2 2	-1 17	2 2	-1 17
0011	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17
0100	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
0101	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17
0110	3 3	-1 17	3 3	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	-1 17	-1 17
0111	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1000	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	-1 17	-1 17	-1 17	-1 17
1001	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	3 3	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1010	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	3 3	-1 17	-1 17	-1 17	-1 17	-1 17
1011	4 4	-1 17	-1 17	-1 17	4 4	-1 17	-1 17	-1 17	4 4	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1100	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1101	4 4	-1 17	-1 17	-1 17	4 4	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1110	4 4	-1 17	4 4	-1 17	4 4	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17
1111	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17	-1 17

Tabel 3.5 *Array DP2*: Baris ke-2

*i**j*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	4	4	4	4	4	4	4	3	4	3	4	3	3	3	3	3
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0001	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0010	5	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0011	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0100	5	5	5	4	5	4	4	4	4	4	4	4	4	4	4	4
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0101	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0110	5	5	5	5	5	5	5	-1	5	5	5	5	5	5	-1	-1
	4	4	4	4	4	4	4	17	4	4	4	4	4	4	17	17
0111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1000	5	4	5	4	4	4	4	4	4	4	4	4	-1	-1	-1	-1
	3	3	3	3	3	3	3	3	3	3	3	3	17	17	17	17
1001	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	-1	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1010	5	5	5	5	5	5	5	5	5	5	5	5	-1	-1	-1	-1
	4	4	4	4	4	4	4	4	4	4	4	4	17	17	17	17
1011	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1100	5	5	5	4	5	4	4	4	-1	-1	-1	-1	-1	-1	-1	-1
	4	4	4	4	4	4	4	4	17	17	17	17	17	17	17	17
1101	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1110	5	5	5	5	5	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	5	5	5	5	5	5	17	17	17	17	17	17	17	17	17	17
1111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17

Nilai yang berada di bagian atas dari setiap kolom menunjukkan *upper bound* dan nilai yang berada di bagian bawah dari setiap kolom menunjukkan *lower bound*.

3.3.3.3 Fungsi *DP3*

Pada fungsi *DP3* juga hampir sama seperti fungsi `dp1()`, namun pada fungsi `dp3()` ini membutuhkan cerminan piksel bagian bawah karena `dp3()` memproses piksel pada baris terakhir. Contoh *array* `dp3()` yang berisi kemungkinan *lower bound* dan *upper bound* dari piksel baris ke-(*H-1*) dapat dilihat pada Tabel 3.6.

Tabel 3.6 *Array DP3*: Baris ke-3

i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	6	6	6
	17	17	17	17	17	17	17	17	17	17	17	17	4	4	4	4
0001	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0010	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
0011	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0100	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
0101	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
0110	6	6	6	6	6	6	6	-1	6	6	6	6	6	6	-1	-1
	4	4	4	4	4	4	4	17	4	4	4	4	4	4	17	17
0111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1000	6	6	6	6	6	6	6	6	6	6	6	6	-1	-1	-1	-1
	4	4	4	4	4	4	4	4	4	4	4	4	17	17	17	17
1001	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1010	6	6	6	6	6	6	6	6	6	6	6	6	-1	-1	-1	-1
	4	4	4	4	4	4	4	4	4	4	4	4	17	17	17	17
1011	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1100	6	6	6	6	6	6	6	6	-1	-1	-1	-1	-1	-1	-1	-1
	4	4	4	4	4	4	4	4	17	17	17	17	17	17	17	17
1101	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1110	6	6	6	6	6	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	4	4	4	4	4	4	17	17	17	17	17	17	17	17	17	17
1111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17

Nilai yang berada di atas menunjukkan *upper bound* dan nilai yang berada di bawah menunjukkan *lower bound*.

Dari contoh diatas, dapat disimpulkan bahwa nilai terakhir $upper\ bound = 6$ dan nilai terakhir $lower\ bound = 4$. Sehingga selisih hasil piksel hitam adalah $upper\ bound - lower\ bound = 6 - 4 = 2$. Hasil tersebut benar karena telah sesuai dengan keluaran kasus uji pada SPOJ *Median Filter*. Ilustrasi tabel rekap dari fungsi `dp1()`, `dp2()`, dan `dp3()` dapat dilihat pada Tabel 3.7.

Tabel 3.7 Hasil Tahap DP: Kasus Uji Gambar 2.1

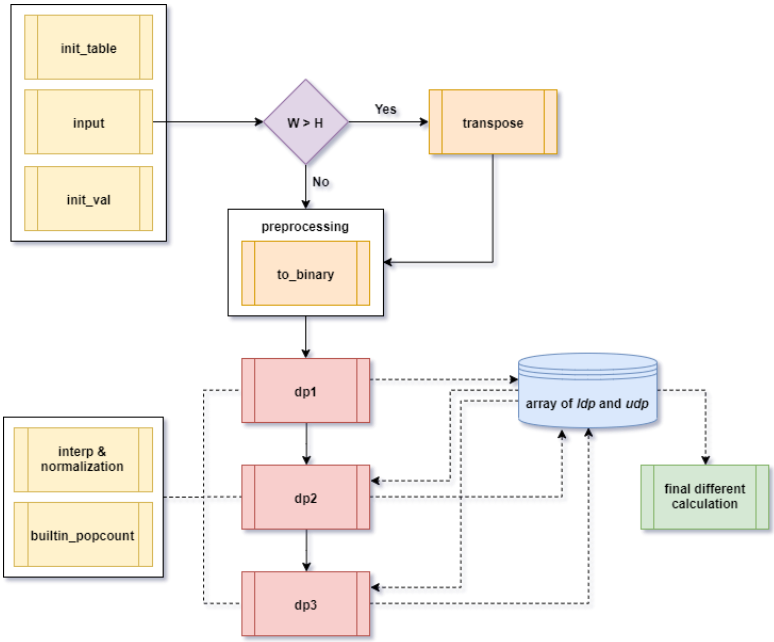
Kasus Uji-1							
	DP1	*DP2*		*DP3*	*min*	*max*	selisih
Baris ke-	0	1	2	3			
udp	3	4	5	6	4	6	**2**
ldp	3	4	5	4			

Nilai *upper bound* (*udp*) dan *lower bound* (*ldp*) pada masing-masing kolom *DP1*, *DP2*, *DP3* berasal dari nilai kemungkinan terbesar yang diambil pada hasil iterasi *numblack* piksel pada baris tersebut. Jika semua nilai *upper bound* pada *DP3* (iterasi baris terakhir) adalah -1, maka program akan mengeluarkan kata **Impossible**, yang berarti program tidak dapat menebak citra asli yang mungkin pada suatu citra masukan atau dapat dikatakan bahwa citra yang sudah mengalami *median filter* tidak memungkinkan untuk dilakukan *inverse* dan dicari citra aslinya yang mungkin, karena nilai piksel yang dianggap saling bertabrakan.

3.4 Deskripsi Umum Program

Program diawali dengan inisialisasi tabel citra dari 0 sampai 255 dengan fungsi *init_table()*. Setelah inisialisasi selesai, selanjutnya program akan menerima masukan berupa *input* angka *W* dan *H*. Kemudian untuk setiap *H* baris akan berisi piksel sebanyak *W* karakter. Selanjutnya program akan menjalankan fungsi *init_val()*. Setelah itu, memasuki tahap *preprocessing*, program akan menjalankan fungsi *to_binary()*. Jika angka *H* melebihi *W*, maka sebelum pengubahan biner, program akan menjalankan fungsi *transpose()*.

Tahap selanjutnya adalah *dynamic programming* yang terbagi menjadi tiga fungsi yaitu *dp1(), dp2(),* dan *dp3()*. Ketiga fungsi tersebut memiliki tujuan yang sama, yaitu mencari nilai *ldp (lower difference pixel)* dan *udp (upper difference pixel)*. Perbedaan ketiga fungsi tersebut hanyalah pada iterasi dan inisialisasi nilai ketetanggaannya. Ketiga fungsi *dynamic programming* tersebut akan dibantu oleh fungsi *interp()*. Visualisasi alur jalannya program dapat dilihat pada Gambar 3.9.



Gambar 3.9 Alur Program

3.5 Desain Algoritma

Pada Gambar 3.10 akan dijelaskan alur kerja program secara umum, kemudian dilanjutkan dengan penjelasan alur kerja setiap fungsi berupa *pseudocode*.

`main()`	
1.	*init_table()*
2.	*while*(*true*)
3.	*input*
4.	*transpose()*
5.	*binary()*
6.	*init_val()*
7.	$mask \leftarrow (1 \ll W) - 1$
8.	*if (W != 1 & H != 1)*
9.	*dp1()*
10.	*dp2()*
11.	*dp3()*
12.	i*f* ($maxval < 0$)
13.	*"Impossible"*
14.	*else*
15.	m$axval - minval$

Gambar 3.10 *Pseudocode* `Main`

3.5.1 Desain Fungsi *Init_table*

Fungsi *init_table* digunakan untuk inisialisasi tabel citra dari 0 sampai 255 karena batas masukan untuk *W* dan *H* adalah 8, maka batas atas inisialisasi tabel adalah 2^8. Tabel ini bertujuan untuk merepresentasikan bit kecil menjadi besar. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.11.

`init_table()`	
1.	$for\ i \leftarrow 0\ to\ 255$
2.	$x \leftarrow 0$
3.	$for\ j \leftarrow 0\ to\ 7$
4.	$if\ (i\ \&\ (1 \ll j))$
5.	$x \leftarrow x \mid = (1 \ll (j * 4))$
6.	$table[i] \leftarrow x$

Gambar 3.11 *Pseudocode* Fungsi `init_table`

3.5.2 Desain Fungsi *Binary*

Fungsi *binary* akan merubah karakter-W menjadi bilangan biner dan di *shift* bit ke kiri sejumlah *x*. Bilangan biner 1 akan merepresentasikan piksel hitam dan bilangan biner 0 akan merepresentasikan piksel putih. Tujuannya adalah untuk memudahkan proses komputasi selanjutnya. *Pseudocode* fungsi *binary* dapat dilihat pada Gambar 3.12.

`binary()`	
1.	$for\ i \leftarrow 0\ to\ H$
2.	$\quad for\ j \leftarrow 0\ to\ W$
3.	$\quad\quad image[i] \leftarrow image[i] \mid = (simage[i][j] == \#) \ll j$

Gambar 3.12 *Pseudocode* Fungsi `Binary`

3.5.3 Desain Fungsi *Transpose*

Fungsi *transpose* akan melakukan penukaran dimensi pada matriks *input* apabila $H < W$. *Pseudocode* fungsi *transpose* dapat dilihat pada Gambar 3.13.

`transpose()`	
1.	$if(H \geq W)$
2.	$\quad binary()$
3.	$else$
4.	$\quad swap(H, W)$
5.	$\quad binary()$

Gambar 3.13 *Pseudocode* Fungsi `Transpose`

3.5.4 Desain Fungsi *Init_val*

Fungsi *init_val* digunakan untuk inisialisasi nilai *ldp (lower difference pixel)* dan *udp (upper difference pixel)*. Nilai ini nantinya akan digunakan untuk menentukan selisih terbesar dari piksel hitam. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.14.

`init_val()`	
1.	$for\ i \leftarrow 0\ to\ H$
2.	$for\ j \leftarrow 0\ to\ 2^W - 1$
3.	$for\ k \leftarrow 0\ to\ 2^W - 1$
4.	$ldp[i][j][k] \leftarrow \mathrm{H} * \mathrm{W} + 1$
5.	$udp[i][j][k] \leftarrow -1$

Gambar 3.14 *Pseudocode* Fungsi `init_val`

3.5.5 Desain Fungsi *Interp*

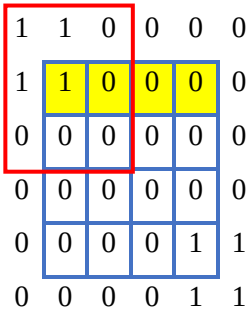
Pseudocode fungsi ini dapat dilihat pada Gambar 3.15.

`interp(i,w)`	
1.	$bits \leftarrow (i \ll 1) \mid (i\ \&\ 1)$
2.	$return\ bits \mid \big((bits\ \&\ (1 \ll w)\big) \ll 1$

Gambar 3.15 *Pseudocode* Fungsi `Interp`

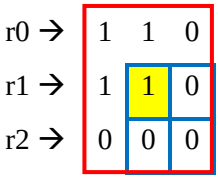
3.5.6 Desain Fungsi *DP1*

Fungsi *dp1* ini bertujuan untuk mengeksekusi baris ke-0. Piksel pada baris ini tidak memiliki tetangga bagian atas (sebelum piksel tepi) dan melihat sifat dari *median filter* dimana ketetanggaan yang kosong akan diisi oleh 0 (*zero-padding*), 1 (*one-padding*), atau cerminan dari piksel tepi. Pada kasus ini, ketetanggaan yang cocok untuk mengisi piksel yang kosong adalah cerminan dari piksel tepi. Visualisasi implementasi *DP1* bisa dilihat pada Gambar 3.16.



Gambar 3.16 Visualisasi *DP1*

Tiap piksel nanti akan dipetakan 3x3 untuk dihitung piksel citra *inverse median filter* yang mungkin dari piksel citra masukan. Setiap piksel citra asli yang mungkin akan dihitung dari *mask* yang ditentukan. *Mask* 3x3 ini direpresentasikan oleh variabel *r0*, *r1*, dan *r2*. Dimana *r0* digunakan pada baris ke-0, *r1* digunakan pada baris ke-1, dan *r2* digunakan pada baris ke-2 (yang dapat dilihat lebih detil pada Gambar 3.17.



Gambar 3.17 Visualisasi *r0*, *r1*, dan *r2*

Tiap piksel citra asli yang mungkin akan dilakukan konfirmasi, apakah hasil penjumlahan piksel setelah *masking dan normalisasi* sama dengan piksel citra yang mengalami *Median Filter*. Jika benar, maka akan dihitung *numblack* (banyaknya piksel hitam) pada iterasi saat itu dan hasil *numblack* akan disimpan ke *array* sebagai nilai *ldp* dan *udp*. Nilai ini nanti yang akan dipakai oleh fungsi *DP2*. *Pseudocode* fungsi *DP1* dapat dilihat pada Gambar 3.18.

dp1()	
1.	$W \leftarrow width\ of\ image$
2.	$H \leftarrow high\ of\ image$
3.	$mask = 2^W - 1$
4.	$for\ i \leftarrow 0\ to\ 2^W - 1$
5.	$r0 = r1 \leftarrow interp(i, W)$
6.	$numblack \leftarrow builtin\_popcount(i)$
7.	$for\ j \leftarrow 0\ to\ 2^W - 1$
8.	$c \leftarrow 0$
9.	$r2 \leftarrow interp(j, W)$
10.	$c \leftarrow c + table[(r0 \gg 0)\ \&\ mask]$
11.	$c \leftarrow c + table[(r0 \gg 1)\ \&\ mask]$
12.	$c \leftarrow c + table[(r0 \gg 2)\ \&\ mask]$
13.	$c \leftarrow c + table[(r1 \gg 0)\ \&\ mask]$
14.	$c \leftarrow c + table[(r1 \gg 1)\ \&\ mask]$
15.	$c \leftarrow c + table[(r1 \gg 2)\ \&\ mask]$
16.	$c \leftarrow c + table[(r2 \gg 0)\ \&\ mask]$
17.	$c \leftarrow c + table[(r2 \gg 1)\ \&\ mask]$
18.	$c \leftarrow c + table[(r2 \gg 2)\ \&\ mask]$
19.	$c \leftarrow (c + 0x33333333)\ \&\ 0x88888888) \gg 3$
20.	$if(c = table[image[0]])$
21.	$ldp[0][i][j] = udp[0][i][j] \leftarrow numblack$

Gambar 3.18 *Pseudocode* Fungsi *DP1*

3.5.7 Desain Fungsi *DP2*

Sama seperti fungsi *dp1*, hanya saja untuk *dp2* akan mengeksekusi piksel pada baris ke-1 sampai baris ke-(*H-2*), jadi perbedaan dari *dp1* adalah pada nilai tepinya. Pada *dp2* tidak memerlukan nilai pencerminan dari ketetanggaan, karena semua piksel tepi dari baris ke-1 sampai ke-(*H-2*) sudah memiliki nilai. *Pseudocode* fungsi *DP2* pada dilihat pada Gambar 3.19.

dp2()	
1.	$for\ i\ \leftarrow 1\ to\ H-2$
2.	$for\ j0 \leftarrow 0\ to\ 2^W - 1$
3.	$r0 \leftarrow interp(j0, W)$
4.	$for\ j1 \leftarrow 0\ to\ 2^W - 1$
5.	$r1 \leftarrow interp(j1, W)$
6.	$numblack \leftarrow builtin\_popcount(i)$
7.	$for\ j2\ \leftarrow 0\ to\ 2^W - 1$
8.	$r2 \leftarrow interp(j2, W)$
9.	$c \leftarrow 0$
10.	$c \leftarrow c + table[(r0 \gg 0)\ \&\ mask]$
11.	$c \leftarrow c + table[(r0 \gg 1)\ \&\ mask]$
12.	$c \leftarrow c + table[(r0 \gg 2)\ \&\ mask]$
13.	$c \leftarrow c + table[(r1 \gg 0)\ \&\ mask]$
14.	$c \leftarrow c + table[(r1 \gg 1)\ \&\ mask]$
15.	$c \leftarrow c + table[(r1 \gg 2)\ \&\ mask]$
16.	$c \leftarrow c + table[(r2 \gg 0)\ \&\ mask]$
17.	$c \leftarrow c + table[(r2 \gg 1)\ \&\ mask]$
18.	$c \leftarrow c + table[(r2 \gg 2)\ \&\ mask]$
19.	$c \leftarrow (\mathrm{c} + 0\mathrm{x}33333333)\ \&\ 0\mathrm{x}88888888) \gg 3$
20.	$if(c = table[image[i]])$
21.	$lower \leftarrow ldp[i-1][j0][j1] + numblack$
22.	$upper \leftarrow udp[i-1][j0][j1] + numblack$
23.	$ldp[i][j1][j2] \leftarrow min(ldp[i][j1][j2], lower)$
24.	$udp[i][j1][j2] \leftarrow max(udp[i][j1][j2], upper)$

Gambar 3.19 *Pseudocode* Fungsi *DP2*

3.5.8 Desain Fungsi *DP3*

Sama seperti fungsi *dp1*, namun pada fungsi *dp3* ini membutuhkan cerminan piksel bagian bawah (setelah piksel tepi) karena *dp3* mengeksekusi piksel pada baris terakhir yaitu baris ke-(*H-1*). *Pseudocode* untuk fungsi *DP3* dapat dilihat pada Gambar 3.20.

dp3()	
1.	$minval \leftarrow H * W + 1$
2.	$maxval \leftarrow -1$
3.	$if(H == 1)$
4.	$for\ i \leftarrow 0\ to\ 2^W - 1$
5.	$minval \leftarrow min\ (minval, ldp[0][i][i])$
6.	$maxval \leftarrow max\ (maxval, udp[0][i][i])$
7.	$else$
8.	$for\ i \leftarrow 0\ to\ 2^W - 1$
9.	$r0 \leftarrow interp(i, W)$
10.	$for\ j \leftarrow 0\ to\ 2^W - 1$
11.	$r1 = r2 \leftarrow interp(j, W)$
12.	$numblack \leftarrow builtin\_popcount(j)$
13.	$c \leftarrow 0$
14.	$c \leftarrow c + table[(r0 \gg 0)\ \&\ mask]$
15.	$c \leftarrow c + table[(r0 \gg 1)\ \&\ mask]$
16.	$c \leftarrow c + table[(r0 \gg 2)\ \&\ mask]$
17.	$c \leftarrow c + table[(r1 \gg 0)\ \&\ mask]$
18.	$c \leftarrow c + table[(r1 \gg 1)\ \&\ mask]$
19.	$c \leftarrow c + table[(r1 \gg 2)\ \&\ mask]$
20.	$c \leftarrow c + table[(r2 \gg 0)\ \&\ mask]$
21.	$c \leftarrow c + table[(r2 \gg 1)\ \&\ mask]$
22.	$c \leftarrow c + table[(r2 \gg 2)\ \&\ mask]$
23.	$c \leftarrow (\mathrm{c} + 0\mathrm{x}33333333)\ \&\ 0\mathrm{x}88888888) \gg 3$
24.	$if(c = table[image[H-1]])$
25.	$lower \leftarrow ldp[H-2][i][j] + numblack$
26.	$upper \leftarrow udp[H-2][i][j] + numblack$
27.	$minval \leftarrow min(minval, lower)$
28.	$maxval \leftarrow max(maxval, upper)$

Gambar 3.20 *Pseudocode* Fungsi *DP3*

BAB 4
IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi algoritma dari desain dan perancangan yang telah dibahas pada Bab 3. Meliputi implementasi dari tiap proses, parameter masukan, keluaran, dan beberapa keterangan yang berhubungan dengan program.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir yang meliputi perangkat keras dan perangkat lunak yang digunakan untuk penyelesaian permasalahan SPOJ *Median Filter* [1] adalah sebagai berikut:

4.1.1 Perangkat Keras

Implementasi Tugas Akhir ini menggunakan *personal computer* (PC) yang memiliki spesifikasi *processor* Intel Core i5-8250U dengan kecepatan *up to* 3,4 GHz, *Random Access Memory* (RAM) sebesar 8 GB, dan mempunyai *Graphics Processing Unit* (GPU) yaitu NVIDIA GeForce 930 MX.

4.1.2 Perangkat Lunak

Implementasi dari sisi perangkat lunak adalah PC dengan sistem operasi Windows 10 64-bit, menggunakan IDE Orwell Bloodshed Dev-C++ 5.11 dan *compiler* g++ (TDM-GCC 4.9.2 64-bit), dan penggunaan bahasa pemograman C++ yang telah dilengkapi dengan *library*.

4.2 Implementasi Penyelesaian Permasalahan *Inverse Median Filter*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *Inverse Median Filter* sesuai dengan desain pada Bab 3.

4.2.1 Penggunaan *Library*, Konstanta, dan Variabel Global

Pada subbab ini, akan dibahas penggunaan *library* dan variabel global yang digunakan dalam program. Terdapat enam *library* yang digunakan yaitu: *iostream*, *string*, *vector*, *algorithm*, *cstring*, dan *bitset*. Kode sumber pemanggilan *library* dapat dilihat pada Kode Sumber 4.1.

```
1.  #include <iostream>
2.  #include <string>
3.  #include <vector>
4.  #include <algorithm>
5.  #include <cstring>
6.  #include <bitset>
7.  using namespace std;
```

Kode Sumber 4.1 *Library* C++

Variabel global yang digunakan pada program ini dapat dilihat pada Kode Sumber 4.2 dan penjelasan variabel global dapat dilihat pada Tabel 4.1.

```
1.  unsigned int table[256];
2.  int ldp[8][1 << 8][1 << 8];
3.  int udp[8][1 << 8][1 << 8];
```

Kode Sumber 4.2 Variabel Global

Tabel 4.1 Penjelasan Variabel Global

No.	Variabel Global	Tipe Data	Penjelasan
1.	*array*	*table*[256]	*Array* yang dibentuk pada saat awal inisialisasi. Sebanyak 256 karena maksimal nilai *W*, *H* = 8. Jadi *array* = 2^8
2.	*matrix*	*ldp*[8][1 << 8][1 << 8]	Matriks yang dibentuk pada saat awal inisialisasi. Angka 8 menunjukkan bahwa maksimal *W*, *H* adalah 8.
3.	*matrix*	*udp*[8][1 << 8][1 << 8]	Matriks yang dibentuk pada saat awal inisialisasi. Angka 8 menunjukkan bahwa maksimal *W*, *H* adalah 8.

Untuk inisialisasi tabel akan menggunakan fungsi `init_table` yang dapat dilihat pada Kode Sumber 4.3. Tabel hasil dari pengubahan dalam fungsi ini dapat dilihat pada tabel 4.2.

```
1.  for(int i = 0; i < 256; ++i){
2.      unsigned int x = 0;
3.      for(int j = 0; j < 8; ++j){
4.          if(i & (1 << j)){
5.              x |= (1 << (j * 4));
6.          }
7.      }
8.      table[i] = x;
9.  }
```

Kode Sumber 4.3 Fungsi `Init_table`

Tabel 4.2 Hasil Fungsi `Init_table`

No.	*i*	*table[i]*
1.	001101	000000001000100000001
2.	101011	100000001000000010001

Selanjutnya, program akan menunggu masukan berupa matriks *W* x *H* yang nantinya akan diproses untuk mendapatkan citra asli yang mungkin dari citra masukan. Implementasi untuk masukan dapat dilihat pada Kode Sumber 4.4 dan penjelasan variabel yang dibutuhkan untuk masukan dapat dilihat pada Tabel 4.3.

```
1.  int H, W; cin >> W >> H;
2.  if(H == 0 && W == 0){
3.      break;
4.  }
5.  vector<string> simage(H);
6.  for(int i = 0; i < H; ++i){
7.      cin >> simage[i];
8.  }
```

Kode Sumber 4.4 *Input* / Masukan

Tabel 4.3 Variabel Masukan

No.	Variabel	Tipe Data	Penjelasan
1.	*H*	integer	Variabel yang menyimpan data jumlah baris
2.	*W*	integer	Variabel yang menyimpan data jumlah kolom / karakter yang dibutuhkan pada setiap baris
3.	*simage*(H)	*vector<string>*	Vektor yang menyimpan karakter data masukan

Setelah program menerima masukan, program akan melakukan pengecekan untuk nilai *W* apakah lebih dari *H* atau tidak, jika iya maka citra masukan akan di-*transpose*. Kemudian program akan mengubah citra masukan yang berupa karakter menjadi bilangan biner. Hal ini akan memudahkan dan mempercepat proses selanjutnya. Implementasi dari fungsi *transpose* dan *binary* dapat dilihat pada Kode Sumber 4.5. Penjelasan untuk variabel baru yang dibutuhkan untuk fungsi ini dapat dilihat pada Tabel 4.4.

```
1.  vector<int> image(max(H, W));
2.  if(H >= W){
3.      for(int i = 0; i < H; ++i){
4.          for(int j = 0; j < W; ++j){
5.              image[i] |= (simage[i][j] == '#') << j;
6.          }
7.      }
8.  }
9.  else{
10.     swap(H, W);
11.     for(int i = 0; i < W; ++i){
12.         for(int j = 0; j < H; ++j){
13.             image[j] |= (simage[i][j] == '#') << i;
14.         }
15.     }
16. }
```

Kode Sumber 4.5 Fungsi `Transpose-Binary`

Tabel 4.4 Variabel Fungsi `Binary`

No.	Variabel	Tipe Data	Penjelasan
1.	*image(max(H,W))*	*vector<int>*	Vektor yang menyimpan perubahan karakter data masukan menjadi angka biner

Selanjutnya, program akan melakukan inisialisasi nilai untuk *array ldp* dan *udp* yang telah dideklarasikan pada variabel global yang dapat dilihat pada Tabel 4.1. Inisialisasi nilai *lower* dan *upper* tersebut dapat dilihat pada Kode Sumber 4.6.

```
1.  for(int i = 0; i < H; ++i){
2.      for(int j = 0; j < (1 << W); ++j){
3.          for(int k = 0; k < (1 << W); ++k){
4.              ldp[i][j][k] = H * W + 1;
5.              udp[i][j][k] = -1;
6.          }
7.      }
8.  }
```

Kode Sumber 4.6 Fungsi Init_val

4.2.2. Penggunaan Fungsi *Interp* pada Fungsi *DP*

Fungsi utama yaitu fungsi *Dynamic Programming* yang akan dibahas implementasinya, memiliki subfungsi yang bernama *interp*. Implementasi fungsi *interp* dapat dilihat pada Kode Sumber 4.7. Penjelasan variabel yang dibutuhkan untuk fungsi *interp* dapat dilihat pada Tabel 4.5.

```
1.  int interp(int bits, int w){
2.      bits = (bits << 1) | (bits & 1);
3.      return bits | ((bits & (1 << w)) << 1);
4.  }
```

Kode Sumber 4.7 Fungsi `Interp`

Tabel 4.5 Variabel Fungsi `Interp`

No.	Variabel	Tipe Data	Penjelasan
1.	*bits*	*integer*	Variabel yang menyimpan data angka dari iterasi fungsi *DP*
2.	*w*	*integer*	Variabel yang menyimpan data angka dari jumlah kolom pada setiap baris

Seperti yang telah dijelaskan pada bab 3, maka fungsi *DP* terbagi menjadi tiga. Fungsi *DP1* khusus untuk mengeksekusi baris ke-0, fungsi *DP2* mengeksekusi baris ke-1 sampai (*H-2*), sedangkan fungsi *DP3* bertujuan untuk mengeksekusi baris ke-(*H-1*). Oleh karena itu, implementasi masing-masing jenis fungsi *DP* juga dibedakan.

Implementasi fungsi *DP1* dapat dilihat pada Kode Sumber 4.8. Penjelasan variabel yang dipakai untuk fungsi *DP1* dapat dilihat pada Tabel 4.6. Implementasi fungsi *DP2* dapat dilihat pada Kode Sumber 4.9 dan penjelasan variabel yang dibutuhkan untuk fungsi *DP2* dapat dilihat pada Tabel 4.7. Sedangkan implementasi fungsi *DP3* dapat dilihat pada Kode Sumber 4.10 dan penjelasan variabel yang dibutuhkan untuk fungsi *DP3* dapat dilihat pada Tabel 4.8.

```
1.  int mask = (1 << W) - 1;
2.  for(int i = 0; i < (1 << W); ++i){
3.      int r0 = interp(i, W), r1 = r0;
4.      int numBlack = __builtin_popcount(i);
5.      for(int j = 0; j < (1 << W); ++j){
6.          int r2 = interp(j, W);
7.          unsigned int c = 0;
8.          c += table[(r0 >> 0) & mask];
9.          c += table[(r0 >> 1) & mask];
10.         c += table[(r0 >> 2) & mask];
11.         c += table[(r1 >> 0) & mask];
12.         c += table[(r1 >> 1) & mask];
13.         c += table[(r1 >> 2) & mask];
14.         c += table[(r2 >> 0) & mask];
15.         c += table[(r2 >> 1) & mask];
16.         c += table[(r2 >> 2) & mask];
17.         cout << bitset<16>(c);
18.         c = ((c + 0x33333333) & 0x88888888) >> 3;
19.         if(c == table[image[0]]){
20.             ldp[0][i][j] = udp[0][i][j] = numBlack;
21.         }
22.     }
23. }
```

Kode Sumber 4.8 Fungsi *DP1*

Tabel 4.6 Variabel Fungsi *DP1*

No.	Variabel	Tipe Data	Penjelasan
1.	*mask*	*integer*	Variabel konstanta yang memiliki nilai tetap `(1 << W) - 1`
2.	*i*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
3.	*j*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
4.	*r0*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris pertama)
5.	*r1*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris kedua)
6.	*r2*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris terakhir)
7.	*numblack*	*integer*	Variabel yang berisi nilai piksel hitam total pada suatu baris tertentu
8.	*c*	*unsigned integer*	Variabel yang berisi nilai dari 3x3 matriks yang sudah di-*masking*

```
for(int i = 1; i < H - 1; ++i){
    for(int j0 = 0; j0 < (1 << W); ++j0){
        int r0 = interp(j0, W);
        for(int j1 = 0; j1 < (1 << W); ++j1){
            int r1 = interp(j1, W), numBlack = __bu
iltin_popcount(j1);
            if(udp[i - 1][j0][j1] < 0){ continue; }
            for(int j2 = 0; j2 < (1 << W); ++j2){
                int r2 = interp(j2, W);
                unsigned int c = 0;
                c += table[(r0 >> 0) & mask];
                c += table[(r0 >> 1) & mask];
                c += table[(r0 >> 2) & mask];
                c += table[(r1 >> 0) & mask];
                c += table[(r1 >> 1) & mask];
                c += table[(r1 >> 2) & mask];
                c += table[(r2 >> 0) & mask];
                c += table[(r2 >> 1) & mask];
                c += table[(r2 >> 2) & mask];
                c = ((c + 0x33333333) & 0x88888888)
 >> 3;
                if(c == table[image[i]]){
                    int lower = ldp[i - 1][j0][j1]
+ numBlack;
                    int upper = udp[i - 1][j0][j1]
+ numBlack;
                    ldp[i][j1][j2] = min(ldp[i][j1]
[j2], lower);
                    udp[i][j1][j2] = max(udp[i][j1]
[j2], upper);
                }
            }
        }
    }
}
```

Kode Sumber 4.9 Fungsi *DP2*

Tabel 4.7 Variabel Fungsi *DP2*

No.	Variabel	Tipe Data	Penjelasan
1.	*mask*	*integer*	Variabel konstanta yang memiliki nilai tetap `(1 << W) - 1`
2.	*i*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
3.	*j0*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
4.	*j1*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
5.	*j2*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
6.	*r0*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris pertama)
7.	*r1*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris kedua)
8.	*r2*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris terakhir)
9.	*numblack*	*integer*	Variabel yang berisi nilai piksel hitam total pada suatu baris tertentu
10.	*c*	*unsigned integer*	Variabel yang berisi nilai dari 3x3 matriks yang sudah di-*masking*

```
int minval = H * W + 1, maxval = -1;
if(H == 1){
    for(int i = 0; i < (1 << W); ++i){
        minval = min(minval, ldp[0][i][i]);
        maxval = max(maxval, udp[0][i][i]);
    }
}
else{
    for(int i = 0; i < (1 << W); ++i){
        int r0 = interp(i, W);
        for(int j = 0; j < (1 << W); ++j){
            int r1 = interp(j, W), r2 = r1;
            if(udp[H - 2][i][j] < 0){ continue; }

            int numBlack = __builtin_popcount(j);

            unsigned int c = 0;
            c += table[(r0 >> 0) & mask];
            c += table[(r0 >> 1) & mask];
            c += table[(r0 >> 2) & mask];
            c += table[(r1 >> 0) & mask];
            c += table[(r1 >> 1) & mask];
            c += table[(r1 >> 2) & mask];
            c += table[(r2 >> 0) & mask];
            c += table[(r2 >> 1) & mask];
            c += table[(r2 >> 2) & mask];
            c = ((c + 0x33333333) & 0x88888888) >>
3;
            if(c == table[image[H - 1]]){
                int lower = ldp[H - 2][i][j] + numB
lack;
                int upper = udp[H - 2][i][j] + numB
lack;
                minval = min(minval, lower);
                maxval = max(maxval, upper);
            }
        }
    }
}
```

Kode Sumber 4.10 Fungsi *DP3*

Tabel 4.8 Variabel Fungsi *DP3*

No.	Variabel	Tipe Data	Penjelasan
1.	*mask*	*integer*	Variabel konstanta yang memiliki nilai tetap `(1 << W) - 1`
2.	*maxval*	*integer*	Variabel inisialisasi nilai paling tinggi dari suatu rentang
3.	*minval*	*integer*	Variabel inisialisasi nilai paling rendah dari suatu rentang
4.	*i*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
5.	*j*	*integer*	Variabel yang digunakan untuk proses penghitungan dalam suatu iterasi
6.	*r0*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris pertama)
7.	*r1*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris kedua)
8.	*r2*	*integer*	Variabel yang berisi nilai yang didapat dari fungsi lain, dapat dikatakan *masking* 3x3 (baris terakhir)
9.	*numblack*	*integer*	Variabel yang berisi nilai piksel hitam total pada suatu baris tertentu
10.	*c*	*unsigned integer*	Variabel yang berisi nilai dari 3x3 matriks yang sudah di-*masking*

Setelah melewati proses *Dynamic Programming*, didapatkan nilai terendah dan tertinggi dari suatu kemungkinan perubahan piksel hitam dari citra masukan dengan dugaan citra asli. Kedua nilai tersebut kemudian akan dicari selisihnya, untuk dapat menyelesaikan permasalahan *Inverse Median Filter*. Apabila sampai pada tahap *DP3* ternyata semua *state* bernilai $maxval = -1$, maka citra tersebut dianggap tidak memiliki solusi karena program tidak dapat mencari citra asli yang mungkin dari suatu citra masukan dan program akan mengeluarkan kata "*Impossible*". Sedangkan apabila terdapat *state* yang bernilai $maxval \neq -1$, maka program akan menampilkan hasil berupa selisih nilai tertinggi dan terendah dari piksel hitam suatu citra masukan dengan citra asli yang mungkin. Kode sumber keluaran program dapat dilihat pada Kode Sumber 4.11.

```
1.  cout << "Case " << caseNum++ << ": ";
2.  if(maxval < 0){
3.      cout << "Impossible" << endl;
4.  }
5.  else{
6.      cout << maxval - minval << endl;
7.  }
```

Kode Sumber 4.11 Hasil Keluaran / *Output*

(Halaman ini sengaja dikosongkan)

BAB 5
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan mengenai hasil uji coba program yang telah dirancang pada Tugas Akhir ini. Uji coba dilakukan untuk mengetahui kinerja algoritma dengan lingkungan uji coba yang telah ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan uji coba pada Tugas Akhir ini adalah sebuah desktop *personal computer* (PC) dengan sistem operasi Windows 10 64-bit. PC yang digunakan memiliki spesifikasi *Processor* Intel Core i5-8250U dengan kecepatan 3,4 GHz, *Random Access Memory* (RAM) sebesar 8 GB, dan mempunyai *Graphics Processing Unit* (GPU) yaitu NVIDIA GeForce 930 MX.

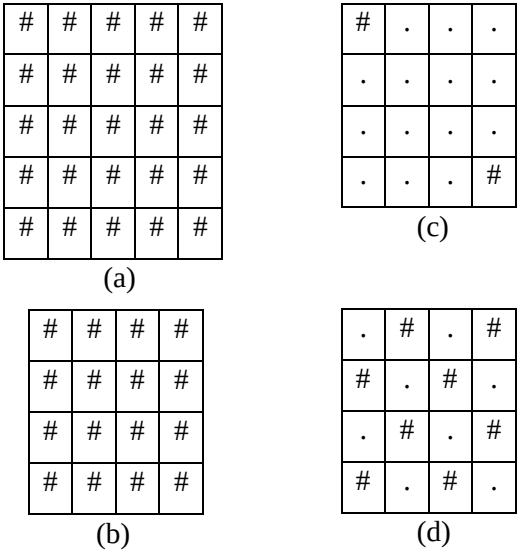
Pada sisi perangkat lunak, uji coba pada Tugas Akhir ini dilakukan dengan menggunakan bahasa pemograman C++ dilengkapi dengan *library* dan menggunakan IDE Orwell Bloodshed Dev-C++ 5.11 dengan *compiler* g++ (TDM-GCC 4.9.2 64-bit).

5.2 Uji Coba Kebenaran

Pada subbab ini akan dijelaskan uji coba kebenaran yang dilakukan untuk menguji desain dan implementasi yang sudah dibuat untuk menyelesaikan Tugas Akhir ini.

5.2.1 Uji Coba Kebenaran Lokal

Uji coba kebenaran lokal dilakukan dengan cara analisis penyelesaian dari beberapa contoh kasus uji menggunakan metode penyelesaian yang telah dijelaskan pada subbab 2. Kasus yang akan digunakan sebagai bahan uji coba kebenaran lokal adalah contoh kasus uji yang diberikan SPOJ. Contoh kasus uji tersebut dapat dilihat pada Gambar 5.1.



Gambar 5.1 Kasus Uji (a) Kasus Uji-1, (b) Kasus Uji-2, (c) Kasus Uji-3, dan (d) Kasus Uji-4

Langkah pertama, program akan menjalankan tahap *preprocessing*. Dalam tahap *preprocessing* ini terdapat berbagai fungsi. Fungsi pertama adalah pengubahan karakter kasus uji menjadi *Boolean* (angka 0 dan 1) agar memudahkan proses selanjutnya pada penyelesaian permasalahan *Median Filter*. Karakter '#' akan diubah menjadi angka 1 untuk merepresentasikan piksel hitam dan karakter '.' akan diubah menjadi angka 0 untuk merepresentasikan piksel putih. Representasi biner pada citra masukan dapat dilihat pada Gambar 5.2.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

(a)

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

(c)

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

(b)

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

(d)

Gambar 5.2 *Binary Representation* (a) Kasus Uji-1, (b) Kasus Uji-2, (c) Kasus Uji-3, dan (d) Kasus Uji-4

Fungsi selanjutnya pada tahap *preprocessing* adalah *bit-shifting*. Fungsi ini bekerja sebagai berikut:

1. Jika piksel berwarna hitam, maka dari nilai kolom sebelumnya ditambahkan *shift bit* 1 ke kiri.
2. Jika piksel berwarna putih, maka dari nilai kolom sebelumnya ditambahkan *shift bit* 0 ke kiri.

Kasus uji yang sudah di konversi menjadi bit dan dilakukan *bit-shifting* dapat dilihat pada Gambar 5.3.

00001	00011	00111	01111	11111
00001	00011	00111	01111	11111
00001	00011	00111	01111	11111
00001	00011	00111	01111	11111
00001	00011	00111	01111	11111

(a)

0001	0011	0111	1111
0001	0011	0111	1111
0001	0011	0111	1111
0001	0011	0111	1111

(b)

0001	0001	0001	0001
0000	0000	0000	0000
0000	0000	0000	0000
0000	0000	0000	1000

(c)

0000	0010	0010	1010
0001	0001	0101	0101
0000	0010	0010	1010
0001	0001	0101	0101

(d)

Gambar 5.3 *Bit Shifting* (a) Kasus Uji-1, (b) Kasus Uji-2, (c) Kasus Uji-3, dan (d) Kasus Uji-4

Kemudian setelah tahap *preprocessing* selesai, maka akan dilanjutkan dengan tahap *Dynamic Programming* (*DP*). Pada tahap *DP* terdapat tiga fungsi *DP* yang berbeda. Program juga akan menjalankan fungsi *interp* saat menjalankan fungsi *DP*.

Setelah semua tahap dieksekusi, maka program akan melakukan seleksi terhadap piksel hitam dari citra asli yang mungkin. Selisih piksel hitam paling tinggi dari hasil citra asli yang mungkin dengan citra masukan adalah solusi yang dibutuhkan untuk menyelesaikan permasalahan *Inverse Median Filter*.

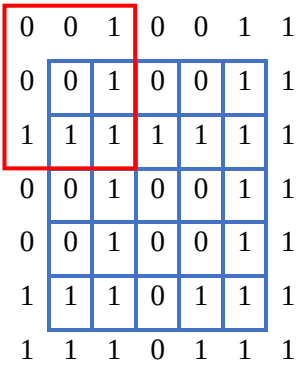
Untuk melakukan uji coba kebenaran lokal, maka akan dilakukan pembuktian dengan cara mengembalikan hasil citra asli yang mungkin menjadi citra masukan. Hasil citra asli yang mungkin tersebut akan diproses dengan *median filtering* sehingga dapat divalidasi sebagai citra masukan. Hasil dari pemrosesan citra tersebut harus sama dengan citra masukan pada kasus uji. Jika hasil citra setelah dilakukan *median filter* sama dengan citra masukan, maka hasil citra asli yang mugkin tersebut relevan dengan citra hasil yang dicari dan dapat dikatakan bahwa citra keluaran tersebut terbukti benar.

5.2.1.1 Kasus Uji-1

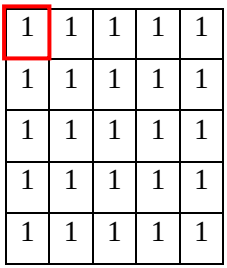
- Hasil citra asli yang mungkin untuk Kasus Uji-1 dapat dilihat pada Gambar 5.4 (a).
- Proses *median filter* dari hasil citra asli yang mungkin dapat dilihat pada Gambar 5.4 (b) dengan asumsi ketetanggaan piksel tepi adalah hasil *mirror* (cerminan) dari piksel yang berada di tepi.
- Hasil setelah dilakukan *median filter* dapat dilihat pada Gambar 5.4 (c) dan citra masukan Kasus Uji-1 pada Gambar 5.4 (d).

0	1	0	0	1
1	1	1	1	1
0	1	0	0	1
0	1	0	0	1
1	1	0	1	1

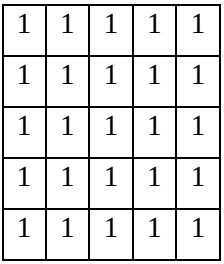
Gambar 5.4 (a) Hasil Citra Asli yang Mungkin untuk Kasus Uji-1



Gambar 5.4 (b) Proses *Median Filtering* 3x3 pada Hasil Citra Asli yang Mungkin untuk Kasus Uji-1



(c)



(d)

Gambar 5.4 (c) Hasil Citra Asli yang Mungkin Setelah Dilakukan *Median Filtering*, (d) Citra Masukan Kasus Uji-1

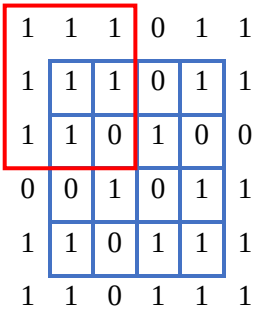
Dapat diamati bahwa hasil citra asli yang mungkin untuk Kasus Uji-1 sama dengan citra masukan Kasus Uji-1, maka solusi untuk Kasus Uji-1 terbukti **benar**.

5.2.1.2 Kasus Uji-2

- Hasil citra asli yang mungkin untuk Kasus Uji-2 dapat dilihat pada Gambar 5.5 (a).
- Proses *median filter* dari hasil citra asli yang mungkin dapat dilihat pada Gambar 5.5 (b) dengan asumsi ketetanggaan piksel tepi adalah hasil *mirror* (cerminan) dari piksel yang berada di tepi.
- Hasil setelah dilakukan *median filter* dapat dilihat pada Gambar 5.5 (c) dan citra masukan Kasus Uji-2 pada Gambar 5.5 (d).

1	1	0	1
1	0	1	0
0	1	0	1
1	0	1	1

Gambar 5.5 (a) Hasil Citra Asli yang Mungkin untuk Kasus Uji-2



Gambar 5.5 (b) Proses *Median Filtering* 3x3 pada Hasil Citra Asli yang Mungkin untuk Kasus Uji-2

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

(c)

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

(d)

Gambar 5.5 (c) Hasil Citra Asli yang Mungkin Setelah Dilakukan *Median Filtering*, (d) Citra Masukan Kasus Uji-2

Dapat diamati bahwa hasil citra asli yang mungkin untuk Kasus Uji-2 sama dengan citra masukan Kasus Uji-2, maka solusi untuk Kasus Uji-2 terbukti **benar**.

5.2.1.3 Kasus Uji-3

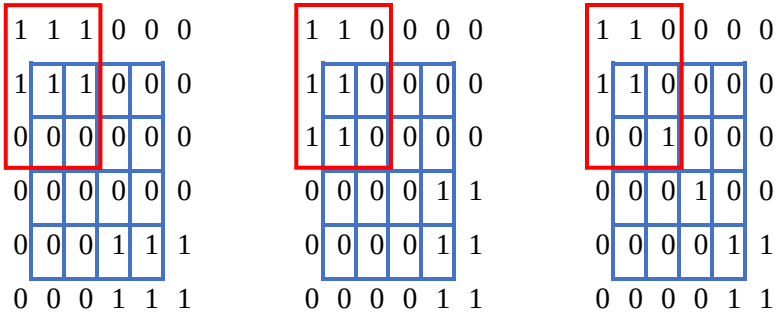
- Hasil citra asli yang mungkin untuk Kasus Uji-3 dapat dilihat pada Gambar 5.6 (a).
- Proses *median filter* dari hasil citra asli yang mungkin dapat dilihat pada Gambar 5.6 (b) dengan asumsi ketetanggaan piksel tepi adalah hasil *mirror* (cerminan) dari piksel yang berada di tepi.
- Hasil setelah dilakukan *median filter* dapat dilihat pada Gambar 5.6 (c) dan citra masukan Kasus Uji-3 pada Gambar 5.6 (d).

1	1	0	0
0	0	0	0
0	0	0	0
0	0	1	1

1	0	0	0
1	0	0	0
0	0	0	1
0	0	0	1

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Gambar 5.6 (a) Hasil Citra Asli yang Mungkin untuk Kasus Uji-3



Gambar 5.6 (b) Proses *Median Filtering* 3x3 pada Hasil Citra Asli yang Mungkin untuk Kasus Uji-3

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

(c)

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

(d)

Gambar 5.6 (c) Hasil Citra Asli yang Mungkin Setelah Dilakukan *Median Filtering*, (d) Citra Masukan Kasus Uji-3

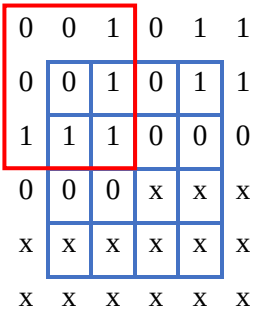
Dapat diamati bahwa hasil citra asli yang mungkin untuk Kasus Uji-3 sama dengan citra masukan Kasus Uji-3, maka solusi untuk Kasus Uji-3 terbukti **benar**.

5.2.1.4 Kasus Uji-4

- Beberapa Kasus Uji tidak dapat dicari hasil citra asli yang mungkin. Hal tersebut dapat diamati melalui citra masukan. Dengan mempertimbangkan perubahan piksel hitam. Apakah dengan perubahan tersebut akan mengembalikan hasil citra asli yang mungkin ke bentuk setelah dilakukan *median filter* atau tidak.
- Hasil citra asli yang mungkin untuk Kasus Uji-4 dapat dilihat pada Gambar 5.7(a).
- Proses *median filter* dari hasil citra asli yang mungkin dapat dilihat pada Gambar 5.7(b) dengan asumsi ketetanggaan piksel tepi adalah hasil *mirror* (cerminan) dari piksel yang berada di tepi.
- Hasil setelah dilakukan *median filter* dapat dilihat pada Gambar 5.7(c) dan citra masukan Kasus Uji-4 pada Gambar 5.7(d).

0	1	0	1
1	1	0	0
0	0	x	x
1	x	x	x

Gambar 5.7 (a) Hasil Citra Asli yang Mungkin untuk Kasus Uji-4



Gambar 5.7 (b) Proses *Median Filtering* 3x3 pada Hasil Citra Asli yang Mungkin untuk Kasus Uji-4

1	0	1	0
0	x	x	x
x	x	x	x
x	x	x	x

(c)

1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

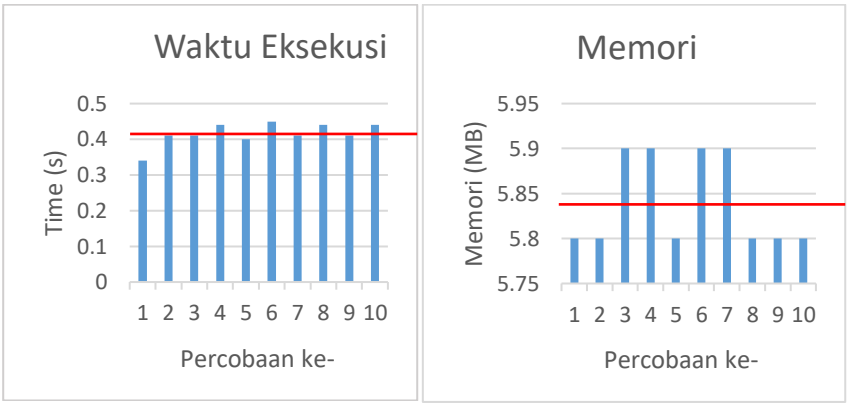
(d)

Gambar 5.7 (c) Hasil Citra Asli yang Mungkin Setelah Dilakukan *Median Filtering*, (d) Citra Masukan Kasus Uji-4

Dapat diamati bahwa pada Kasus Uji-4 **tidak dapat** dicari citra asli yang mungkin, yang berarti tidak akan ada hasil yang sama dengan citra masukan Kasus Uji-4. Salah satu solusi untuk Kasus Uji seperti ini akan menghasilkan keluaran ***'Impossible'*** pada program. Oleh karena itu, maka solusi untuk Kasus Uji-4 terbukti **benar**.

5.2.2 Uji Coba Kebenaran pada Situs Daring SPOJ

Uji coba kebenaran program dilakukan dengan cara mengirimkan kode sumber ke situs penilaian *Sphere Online Judge*. Sistem pada situs penilaian tersebut akan menjalankan program yang dikirim oleh pengguna dan hasilnya kemudian akan dicocokkan dengan file keluaran yang telah disediakan oleh pembuat soal. Selain itu juga dilakukan percobaan pengujian 10 kali untuk mendapatkan rata-rata waktu eksekusi dan memori. Grafik waktu eksekusi dan memori hasil uji coba pada situts SPOJ dapat dilihat pada Gambar 5.8. Garis merah menandakan rata-rata dari waktu eksekusi dan memori. Sedangkan hasil dari uji coba kebenaran pada situs daring SPOJ dapat dilihat pada Gambar 5.9.



Gambar 5.8 Grafik Waktu Eksekusi dan Memori dari Hasil Uji Coba pada Situs SPOJ

25188539	☐	2020-01-04 08:16:17	Median Filter	**accepted** edit ideone it	0.44	5.8M	C++ 4.3.2
25188538	☐	2020-01-04 08:16:14	Median Filter	**accepted** edit ideone it	0.41	5.8M	C++ 4.3.2
25188532	☐	2020-01-04 08:15:04	Median Filter	**accepted** edit ideone it	0.44	5.8M	C++ 4.3.2
25188531	☐	2020-01-04 08:15:00	Median Filter	**accepted** edit ideone it	0.41	5.9M	C++ 4.3.2
25188529	☐	2020-01-04 08:14:56	Median Filter	**accepted** edit ideone it	0.45	5.9M	C++ 4.3.2
25188524	☐	2020-01-04 08:14:23	Median Filter	**accepted** edit ideone it	0.40	5.8M	C++ 4.3.2
25188523	☐	2020-01-04 08:14:11	Median Filter	**accepted** edit ideone it	0.44	5.9M	C++ 4.3.2
25188512	☐	2020-01-04 08:13:29	Median Filter	**accepted** edit ideone it	0.41	5.9M	C++ 4.3.2
25188450	☐	2020-01-04 08:04:40	Median Filter	**accepted** edit ideone it	0.41	5.8M	C++ 4.3.2
24381381	☐	2019-09-12 06:51:46	Median Filter	**accepted** edit ideone it	0.34	5.8M	C++ 4.3.2

Gambar 5.9 Uji Coba Kebenaran pada Situs SPOJ

5.3 Analisis dan Kesimpulan Umum

Pada subbab ini akan diberikan hasil dari evaluasi di atas dalam bentuk poin-poin. Pada bab 6, kesimpulan umum ini nantinya disarikan lagi menjadi Kesimpulan.

1. Dari hasil analisis, algoritma yang cocok untuk menyelesaiakn permasalahan SPOJ *Median Filter* [1] adalah metode *Dynamic Programming.* Karena dengan metode tersebut, kebutuhan penghitungan *inverse* setiap piksel pada citra dapat dilakukan secara rekursif dan dapat diakumulasi. Desain yang digunakan adalah dari analisis submasalah optimal pada pendekatan *Dynamic Programming*.

2. Pada implementasi algoritma *Dynamic Programming*, citra asli yang mungkin (*possible original image*) tidak dapat divisualisasikan karena pencarian kemungkinan solusi pada citra masukan dilakukan berdasarkan setiap baris matriks dari citra, bukan dari keseluruhan matriks citra. Sehingga program hanya dapat menghasilkan keluaran sesuai dengan keluaran pada soal, namun tidak bisa memperlihatkan bagaimana citra asli yang mungkin secara utuh maupun parsial.

3. Uji coba kebenaran pada situs daring SPOJ menghasilkan verdict **Accepted**. Dari hasil uji coba kebenaran pada situs daring SPOJ yang dilakukan sebanyak 10 kali, maka dapat disimpulkan bahwa kinerja algoritma *dynamic programming* memiliki rata-rata waktu eksekusi **0,415 detik**. Sedangkan untuk rata-rata memori yang dibutuhkan adalah **5,84 MB**. Dengan begitu, metode *Dynamic Programming* terbukti dapat menyelesaikan permasalahan *Inverse Median Filter* dengan baik dan efisien, karena apabila tidak menggunakan metode tersebut maka program akan memakan banyak sekali *resource* memori dan komputasi menjadi lebih lambat.

(Halaman ini sengaja dikosongkan)

BAB 6
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dalam pengerjaan tugas akhir ini setelah melalui tahap perancangan aplikasi, implementasi metode, serta uji coba, dan disarikan dalam subbab 5.3 diperoleh kesimpulan sebagai berikut:

1. Desain yang digunakan adalah dari analisis submasalah optimal pada pendekatan *Dynamic Programming*.
2. Pada implementasi algoritma *Dynamic Programming*, citra asli yang mungkin (*possible original image*) tidak dapat divisualisasikan karena pencarian kemungkinan solusi pada citra masukan dilakukan berdasarkan setiap baris matriks dari citra, bukan dari keseluruhan matriks citra.
3. Dari hasil uji coba kebenaran pada situs SPOJ yang dilakukan sebanyak 10 kali, disimpulkan bahwa rata-rata waktu eksekusi yang dibutuhkan adalah **0,415 detik**. dan rata-rata memori yang dibutuhkan adalah **5,84 MB**. Dengan begitu, metode *Dynamic Programming* terbukti dapat menyelesaikan permasalahan *Inverse Median Filter* dengan baik dan efisien

6.2 Saran

Pada tugas akhir ini, tentu tidak terlepas daripada kekurangan serta hal-hal penting yang dapat menjadi pertimbangan kelak. Berikut saran-saran yang dapat diambil dari tugas akhir ini:

1. Karena pada tugas akhir ini diketahui bahwa citra asli yang mungkin tidak dapat divisualisasikan, maka untuk pengembangan selanjutnya dapat diarahkan untuk melakukan kombinasi dengan pembelajaran mesin agar citra asli yang mungkin dapat divisualisasikan.
2. Metode *Dynamic Programming* dapat menjadi bahan riset untuk mencari optimasi lebih lanjut.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] B. Jin, "Sphere Online Judge," Sphere Research Labs., 08 09 2008. [Online]. Available: https://www.spoj.com/problems/FILTER/. [Accessed 12/09/2019].

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms 3rd Edition, London, England: The MIT Press, 2009.

[3] R. C. Gonzales and R. E. Woods, Digital Image Processing 3rd Edition, New Jersey: Pearson Education, 2008.

[4] F. Ali, "Image restoration using regularized inverse filtering and adaptive threshold wavelet denoising," *Al- Khwarizmi Engineering Journal*, vol. 3, pp. 48-62, 2007.

[5] A. Diaz-Sanchez, J. M. R. Perez and J. Lemus-Lopez, "Weak-inversion topologies of analog median filter," vol. 4, pp. 6-9, 2009.

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Yoshima Syach Putri, lahir di Kediri tanggal 25 Juni 1998. Penulis merupakan anak pertama dari tiga bersaudara. Penulis telah menempuh pendidikan formal di TK BPPI Pare, SDN 2 Pare (2004-2010), SMPN 2 Pare (2010-2013), dan SMAN 2 Pare (2013-2016). Penulis melanjutkan studi dengan berkuliah pada program sarjana di Departemen Informatika ITS.

Selama kuliah di Informatika ITS, penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah S1 Sistem Operasi, Pemrograman Java (PIKTI), Pemrograman C# (PIKTI), dan Teknologi Basis Data (PIKTI). Selama menempuh perkuliahan, penulis juga aktif di kegiatan organisasi dan kepanitiaan, diantaranya menjadi staf ahli Sosmas HMTC, staf National Seminar of Technology (NST) 2017, BPH Humas Schematics 2018, dan Steering Committee Kaderisasi HMTC 2018-2019. Selain itu, penulis juga bergabung dengan komunitas Surabaya Dev (2017-2018), menjadi administrator laboratorium Komputasi Cerdas dan Visi (KCV) Informatika ITS, dan memperoleh penghargaan 10 besar finalis pada kompetisi Esai Gamamed di UGM. Penulis dapat dihubungi melalui surel di yoshimaputri@gmail.com.