Pendekatan Program Dinamis untuk Maksimalisasi Keuntungan dalam Sistem Penanaman Berjangka Waktu

Ranashahira Reztaputri - 13523007
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ranashahira.rez@gmail.com , 13523007@std.stei.itb.ac.id

***Abstrak*—Dalam dunia pertanian, terdapat banyak konsiderasi dalam pemilihan tanaman yang akan ditanam, seperti jenis tanaman, waktu panen, musim, permintaan pasar, serta ketersediaan modal. Maka dari itu, kebanyakan petani melakukan penjadwalan sebelum penanaman untuk memastikan pendapatan keuntungan yang maksimal. Namun, masih banyak petani yang menggunakan pendekatan konvensional dalam melakukan penjadwalan ini, seperti spreadsheet maupun pencatatan manual. Maka dari itu, makalah ini mengusulkan penggunaan pendekatan program dinamis dalam optimalisasi penjadwalan penanaman dengan mempertimbangkan keterbatasan modal awal. Pengujian efektivitas dari pendekatan ini dilakukan dengan membandingkannya terhadap metode brute force dan greedy. Penelitian ini diharapkan dapat memberikan solusi alternatif yang lebih sistematis dan efektif dalam mengoptimalkan jadwal tanam untuk maksimalisasi keuntungan pada kondisi terbatas.**

Kata Kunci—brute force, greedy, maksimalisasi keuntungan, program dinamis,

I. PENDAHULUAN

Sektor pertanian memegang peranan penting dalam menjaga ketahanan pangan dan mendukung perekonomian. Dalam proses bercocok tanam, keberhasilan produksi pertanian tidak hanya ditentukan oleh faktor biologis tanaman, tetapi juga oleh strategi pengelolaan waktu dan sumber daya yang tepat. Salah satu aspek penting dalam hal ini adalah penjadwalan tanam yang efisien, mengingat adanya variasi dalam kondisi cuaca, musim tanam, serta permintaan pasar yang fluktuatif.

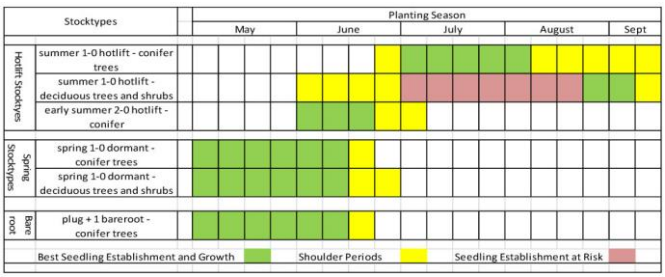
Selama ini, masih banyak petani yang mengandalkan metode konvensional dalam menyusun jadwal tanam, seperti pencatatan manual atau penggunaan spreadsheet sederhana. Pendekatan ini kurang mampu mengakomodasi kompleksitas variabel yang memengaruhi hasil pertanian secara optimal. Oleh karena itu, dibutuhkan suatu metode yang lebih sistematis dan adaptif terhadap berbagai kondisi.

Makalah ini mengusulkan penerapan pendekatan program dinamis untuk optimalisasi penjadwalan penanaman. Untuk tahap awal, pendekatan ini disederhanakan agar dapat dianalisis dengan lebih terfokus pada penerapan dan pengujian efektivitas algoritma. Variabel-variabel yang dipertimbangkan dalam model meliputi jenis tanaman, musim tanam, waktu panen, harga beli, harga jual, dan total modal awal yang tersedia. Sebagai bagian dari evaluasi, pendekatan program dinamis ini akan dibandingkan dengan dua pendekatan lainnya, yaitu metode brute force dan metode greedy. Perbandingan ini bertujuan untuk mengukur tingkat optimalitas dan efisiensi dari masing-masing pendekatan dalam menghasilkan jadwal tanam yang memberikan keuntungan maksimal.

II. LANDASAN TEORI

A. *Penjadwalan Tanam dalam Pertanian*

Penjadwalan tanam merupakan proses perencanaan kegiatan bercocok tanam berdasarkan pertimbangan waktu, jenis tanaman, musim, dan kondisi lingkungan, dengan tujuan memperoleh hasil produksi yang optimal. Dalam praktik pertanian, penjadwalan ini mencakup pemilihan waktu tanam dan panen yang sesuai dengan kondisi agroklimat setempat serta permintaan pasar. Penjadwalan tanam itu penting karena memungkinkan petani untuk menghindari penanaman pada musim yang tidak cocok, menghindari hama dan penyakit, menyebar risiko gagal panen akibat cuaca ekstrem, dan meningkatkan hasil panen.



Gambar 2.1 Contoh jadwal tanam

(Sumber: https://borealhort.com/stock-growing-schedule/)

Terdapat beberapa faktor utama yang memengaruhi keputusan dalam penjadwalan tanam, yaitu antara lain:

1. Jenis tanaman

   Setiap tanaman memiliki karakteristik yang berbeda, seperti siklus pertumbuhan. Pemilihan jenis tanaman dapat mempengaruhi penjadwalan tanam selanjutnya.

2. Musim dan kondisi iklim

   Musim dan iklim dapat mempengaruhi keberhasilan penanaman. Terdapat beberapa jenis tanaman yang hanya dapat ditanam pada musim dan iklim tertentu.

3. Waktu panen

   Setiap jenis tanaman memiliki waktu panen yang berbeda-beda. Waktu panen haus dipertimbangkan dalam penjadwalan untuk memastikan bahwa tanaman tidak akan berbenturan dengan musim yang tidak sesuai.

4. Harga beli dan harga jual

   Faktor ekonomi sangat berpengaruh dalam penjadwalan tanam untuk memastikan peraihan keuntungan yang maksimal.

5. Ketersediaan lahan dan sumber daya

   Jadwal tanam perlu disusun agar tidak ada tumpang tindih dalam penggunaan lahan.

Dengan semakin kompleksnya faktor-faktor yang harus dipertimbangkan, pendekatan manual menjadi kurang efektif. Oleh karena itu, dibutuhkan pendekatan sistematis dan terotomatisasi yang mampu mempertimbangkan banyak variabel sekaligus, seperti dengan penggunaan algoritma optimasi, yang dalam pembahasan ini menggunakan pendekatan program dinamis, brute force, dan greedy.

B. Program Dinamis

Program dinamis atau *dynamic programming* merupakan metode penyelesaian persoalan yang melibatkan pengambilan keputusan dengan memanfaatkan informasi dari penyelesaian subpersoalan yang sama namun lebih kecil (prinsip optimalitas). Solusi subpersoalan tersebut hanya dihitung satu kali dan disimpan di memori. Sama seperti greedy, solusi program dinamis dapat bekerja hanya jika solusi optimal dari persoalan dapat ditentukan dari solusi optimal subpersoalan tersebut. Perbedaannya dengan greedy adalah greedy hanya menghasilkan satu rangkaian keputusan, sedangkan program dinamis mempertimbangkan lebih dari satu rangkaian keputusan sebelum menentukan yang paling optimal. Program dinamis digunakan pada persoalan yang memiliki dua ciri utama, yaitu *optimal substructure* (solusi optimal dari persoalan dapat dibangun dari solusi optimal subpersoalan) dan *overlapping subproblems* (subpersoalan yang sama muncul berulang kali).

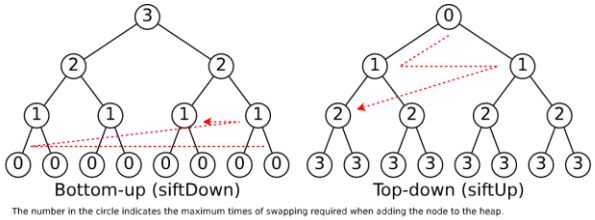
Terdapat dua pendekatan yang dapat digunakan pada program dinamis, yakni:

1. Program dinamis maju (*Top-down approach*)

   Pada pendekatan ini, program dinamis diimplementasikan secara rekursif sambil mencatat nilai yang sudah ditemukan (memoisasi).

2. Program dinamis mundur (*Bottom-up approach*)

   Pada pendekatan ini, program dinamis diimplementasikan secara iteratif dengan menghitung mulai dari kasus yang kecil ke besar.



Gambar 2.2 Perbandingan dua pendekatan program dinamis

(Sumber: https://en.wikipedia.org/wiki/Bottom-up_and_top-down_design)

C. Brute Force

Brute force merupakan pendekatan yang sederhana dan lugas untuk memecahkan masalah dengan cara mencoba setiap solusi yang mungkin hingga menemukan solusi terbaik. Algoritma ini tidak menggunakan trik atau jalan pintas yang cerdas untuk mengurangi ruang pencarian atau meningkatkan efisiensi. Brute force menjamin solusi pasti benar karena menelusuri seluruh kemungkinan. Akibatnya, secara umum, brute force bekerja dengan lambat, terutama ketika terdapat banyak kemungkinan solusi yang perlu dicoba. Algoritma ini tentunya memiliki kelebihan dan kekurangan. Kelebihan dari brute force adalah pendekatannya yang sederhana sehingga membuat ide penyelesaiannya lebih mudah dipikirkan dan diimplementasikan. Di sisi lain, kekurangan dari brute force adalah waktu penyelesaiannya yang cenderung lambat dan tidak efisien.

D. Greedy

Greedy merupakan pendekatan algoritmik yang membangun solusi langkah demi langkah, selalu memilih bagian berikutnya yang menawarkan manfaat paling jelas dan langsung. Pada setiap langkah, harus dibuat keputusan yang terbaik dalam menentukan pilihan sehingga tidak bisa kembali lagi ke langkah sebelumnya. Jadi, pada setiap langkah, kita memilih optimum lokal (*local optimum*) dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (*global optimum*).

Algoritma greedy seringkali digunakan untuk optimasi. Persoalan yang dapat dioptimasi dengan algoritma greedy dapat dilihat dengan mengetahui karakteristik dari algoritma greedy. Karakteristik dari algoritma greedy, diantaranya adalah:

1. Algoritma greedy sederhana dan mudah dipahami.
2. Algoritma greedy efisien dalam hal kompleksitas waktu, dan sering kali memberikan solusi yang cepat.
3. Algoritma greedy tidak mempertimbangkan kembali pilihan-pilihan sebelumnya, karena membuat

keputusan berdasarkan informasi terkini tanpa melihat ke depan.

Selain memiliki karakteristik, algoritma Greedy juga memiliki elemen-elemen, yaitu:

1. Himpunan kandidat, C: berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, S: berisi kandidat yang sudah dipilih.
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan
4. Fungsi seleksi (*selection function*): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi obyektif: memaksimumkan atau meminimumkan.

III. IMPLEMENTASI

Dalam implementasi penjadwalan penanaman ini, terdapat beberapa batasan yang diterapkan, yaitu jumlah hari keseluruhan dan modal awal. Di sisi lain, tiap jenis tanaman memiliki harga beli, harga jual, dan waktu panen. Selain itu, tiap jenis tanaman memiliki musimnya masing-masing sehingga penanaman harus mempertimbangkan musim supaya tanaman tidak ditanam pada musim yang tidak sesuai.

Makalah ini mengusulkan implementasi penjadwalan penanaman menggunakan pendekatan program dinamis. Sebagai pembanding, algoritma lain seperti brute force dan greedy juga diimplementasikan. Berikut implementasi untuk tiap pendekatan:

A. *Program Dinamis*

```
struct Tanaman {
    string nama;
    int harga_beli;
    int harga_jual;
    int waktu_panen;
    int musim; // 0 = KEMARAU, 1 = HUJAN
    int keuntungan() const { return harga_jual - harga_beli; }
};

struct Aksi {
    int hari;
    int petak;
    int id_tanaman;
    int hari_panen;
};

struct HasilMemo {
    int total_keuntungan;
    vector<Aksi> daftar_aksi;
};
```

Gambar 3.1 Struct yang digunakan
(Sumber: Arsip pribadi penulis)

Terdapat tiga struct yang digunakan pada pendekatan program dinamis. Dua di antaranya, yakni Tanaman dan Aksi digunakan pada semua pendekatan, karena keduanya adalah representasi yang dibutuhkan untuk menyelesaikan persoalan penjadwalan penanaman ini. Struct Tanaman digunakan untuk menggambarkan tanaman beserta atributnya, sedangkan struct Aksi digunakan untuk menggambarkan aksi penanaman beserta informasinya. Selain itu, struct HasilMemo digunakan untuk memoisasi (menyimpan hasil rekursif) karena Program Dinamis yang digunakan adalah program dinamis *top-down* sehingga memerlukan memoisasi supaya lebih optimal.

```
string buat_kunci(int hari, int modal, const vector<int>& ketersediaan) {
    string kunci = to_string(hari) + "," + to_string(modal) + ",";
    for (int x : ketersediaan) {
        kunci += to_string(x) + ",";
    }
    return kunci;
}
```

Gambar 3.2 Fungsi buat_kunci
(Sumber: Arsip pribadi penulis)

Fungsi ini bertugas membuat kunci unik untuk *map memoization*. Kunci ini menggabungkan informasi hari, modal, dan status ketersediaan setiap petak lahan menjadi satu string tunggal. Contoh kuncinya bisa seperti "1,500,0,0,0,0,", yang artinya: hari ke-1, modal 500, dan 4 petak lahan sedang kosong (tersedia di hari ke-0).

```
HasilMemo cari(int hari, int modal, vector<int>& ketersediaan) {
    if (hari > jumlah_hari) return {0, {}};

    string kunci = buat_kunci(hari, modal, ketersediaan);
    if (memo.count(kunci)) return memo[kunci];

    int musim = (hari <= batas_kemarau) ? 0 : 1;
    HasilMemo hasil_terbaik = {0, {}};
    function<void(int, int, vector<int>&, vector<Aksi>&)> coba_tanam =
        [&](int modal_sekarang, int indeks_petak, vector<int>& ketersediaan_sekarang, vector<Aksi>& aksi_sekarang) {
            HasilMemo hasil_mendatang = cari(hari + 1, modal_sekarang, ketersediaan_sekarang);
            int keuntungan_total = hasil_mendatang.total_keuntungan;
            for (const auto& aksi : aksi_sekarang) {
                keuntungan_total += daftar_tanaman[aksi.id_tanaman].keuntungan();
            }

            if (keuntungan_total > hasil_terbaik.total_keuntungan) {
                hasil_terbaik.total_keuntungan = keuntungan_total;
                hasil_terbaik.daftar_aksi = hasil_mendatang.daftar_aksi;
                for (const auto& aksi : aksi_sekarang) {
                    hasil_terbaik.daftar_aksi.push_back(aksi);
                }
            }

            for (int petak = indeks_petak; petak < baris * kolom; petak++) {
                if (ketersediaan_sekarang[petak] < hari) {
                    for (int i = 0; i < jumlah_tanaman; i++) {
                        const auto& t = daftar_tanaman[i];
                        if (t.musim == musim && modal_sekarang >= t.harga_beli &&
                            hari + t.waktu_panen <= jumlah_hari) {

                            int backup = ketersediaan_sekarang[petak];
                            ketersediaan_sekarang[petak] = hari + t.waktu_panen;
                            aksi_sekarang.push_back({hari, petak, i, hari + t.waktu_panen});
                            coba_tanam(modal_sekarang - t.harga_beli, petak + 1, ketersediaan_sekarang, aksi_sekarang);
                            ketersediaan_sekarang[petak] = backup;
                            aksi_sekarang.pop_back();
                        }
                    }
                }
            }
        };

    vector<Aksi> kosong;
    coba_tanam(modal, 0, ketersediaan, kosong);
    return memo[kunci] = hasil_terbaik;
}
```

Gambar 3.2 Fungsi cari
(Sumber: Arsip pribadi penulis)

Fungsi cari merupakan fungsi utama yang menerapkan program dinamis secara rekursif (*top-down*). Secara garis besar, fungsi ini memiliki base case berupa kondisi disaat *hari* > *jumlah_hari*, jika sudah mencapai *base case* tersebut, rekursi akan berhenti dan mengembalikan keuntungan 0 tanpa aksi apa

pun. Selain itu, jika nilai kunci sudah ada pada memoisasi, maka hasil memoisasi tersebut akan langsung dikembalikan.

Jika kunci tidak ada pada memoisasi, yang berarti *state* saat ini belum pernah ditelusuri, fungsi ini akan mementukan musim berdasarkan hari saat ini. Setelah itu, fungsi akan menginisialisasi variabel untuk menyimpan hasil terbaik. Di dalamnya terdapat fungsi lambda coba_tanam yang menerapkan *backtracking* dengan mencoba semua kombinasi penanaman di petak yang tersedia. Tanaman hanya ditanam jika musim cocok, modal cukup, dan waktu panen masih valid. Jika ditanam, status petak diperbarui, aksi dicatat, dan fungsi memanggil dirinya sendiri secara rekursif. Setelahnya, dilakukan *backtrack* agar bisa mencoba opsi lain. Selain menanam, tersedia juga pilihan untuk tidak menanam dan langsung lanjut ke hari berikutnya, lalu hasilnya dibandingkan untuk mencari solusi terbaik.

```
void tampilkan_lahan(const vector<Aksi>& aksi, int hari_target, string nama_musim) {
    cout << "\n=== Lahan Hari ke-" << hari_target << " (" << nama_musim << ") ===" << endl;
    vector<vector<string>> lahan(baris, vector<string>(kolom, "  .  "));
    for (const auto& a : aksi) {
        if (a.hari <= hari_target && a.hari_panen > hari_target) {
            int r = a.petak / kolom;
            int c = a.petak % kolom;
            string n = daftar_tanaman[a.id_tanaman].nama;
            if (n.length() > 4) n = n.substr(0, 4);
            while (n.length() < 4) n += " ";
            lahan[r][c] = " " + n + " ";
        }
    }
    cout << "  ";
    for (int j = 0; j < kolom; j++) cout << "+-----";
    cout << "+\n";

    for (int i = 0; i < baris; i++) {
        cout << "  ";
        for (int j = 0; j < kolom; j++) {
            cout << "|" << lahan[i][j];
        }
        cout << "|\n";
        cout << "  ";
        for (int j = 0; j < kolom; j++) cout << "+-----";
        cout << "+\n";
    }

    cout << "Keterangan: '.' = kosong, nama = tanaman aktif\n";
}
```

Gambar 3.3 Fungsi tampilkan_lahan

(Sumber: Arsip pribadi penuli)

Fungsi tampilkan_lahan ini adalah fungsi yang digunakan untuk visualisasi kondisi lahan pada hari tertentu. Ia menerima daftar semua aksi yang akan dilakukan, dan hari_target kemudian menggambar kondisi lahan pada hari_target tersebut. Petak yang sedang ditanami akan menampilkan nama tanaman, sedangkan yang kosong akan menampilkan titik.

B. *Brute Force*

Implementasi dengan pendekatan brute force menggunakan struct yang sama, yakni Tanaman dan Aksi. Terdapat juga penggunaan fungsi yang sama, yakni tampilkan_lahan. Tentunya, logika pada pendekatan ini diimplementasikan dengan fungsi yang berbeda. Logika utama pada pendekatan ini terdapat pada fungsi brute_force.

```
void brute_force(int hari, int modal, vector<int> ketersediaan, vector<Aksi> aksi, int profit_saat_ini) {
    if (hari > jumlah_hari) {
        if (profit_saat_ini > profit_maks) {
            profit_maks = profit_saat_ini;
            hasil_terbaik = aksi;
        }
        return;
    }

    int tambahan_profit = 0;
    for (const auto& a : aksi) {
        if (a.hari_panen == hari) {
            tambahan_profit += daftar_tanaman[a.id_tanaman].keuntungan();
        }
    }

    function<void(int, vector<int>, vector<Aksi>, int)> coba_petak =
        [&](int indeks, vector<int> lahan, vector<Aksi> aksi_hari_ini, int sisa_modal) {
            if (indeks == total_petak) {
                brute_force(hari + 1, sisa_modal, lahan, aksi_hari_ini, profit_saat_ini + tambahan_profit);
                return;
            }

            coba_petak(indeks + 1, lahan, aksi_hari_ini, sisa_modal);

            if (lahan[indeks] < hari) {
                int batas = jumlah_hari / 2 + (jumlah_hari % 2 > 0 ? 1 : 0);
                int musim = (hari <= batas) ? 0 : 1;
                for (int i = 0; i < jumlah_tanaman; i++) {
                    const auto& t = daftar_tanaman[i];
                    if (t.musim == musim && sisa_modal >= t.harga_beli && hari + t.waktu_panen <= jumlah_hari) {
                        vector<int> lahan_baru = lahan;
                        vector<Aksi> aksi_baru = aksi_hari_ini;
                        lahan_baru[indeks] = hari + t.waktu_panen;
                        aksi_baru.push_back({hari, indeks, 1, hari + t.waktu_panen});
                        coba_petak(indeks + 1, lahan_baru, aksi_baru, sisa_modal - t.harga_beli);
                    }
                }
            }
        };

    coba_petak(0, ketersediaan, aksi, modal);
}
```

Gambar 3.4 Fungsi brute_force

(Sumber: Arsip pribadi penulis)

Fungsi ini merupakan inti dari pendekatan Brute Force yang bekerja secara rekursif untuk menjelajahi semua kemungkinan tanpa menyimpan hasil perhitungan sebelumnya. Prosesnya dimulai dari *base case*, yaitu ketika hari saat ini telah melewati jumlah hari yang tersedia dalam permainan. Pada titik ini, simulasi dianggap selesai dan program membandingkan profit_saat_ini dengan profit_maks. Jika nilai saat ini lebih tinggi, maka profit_maks dan hasil_terbaik akan diperbarui.

Berbeda dengan pendekatan program dinamis, fungsi ini tidak langsung menambahkan hasil panen ke modal, melainkan hanya menyimpannya dalam profit_saat_ini. Di dalamnya ada fungsi coba_petak yang melakukan *backtracking* untuk setiap petak, dengan dua pilihan, yakni dilewati atau ditanami. Jika petak kosong, semua tanaman yang valid akan dicoba. Setiap percobaan membuat salinan baru dari status lahan dan aksi, lalu memanggil ulang fungsi secara rekursif.

C. *Greedy*

Serupa dengan pendekatan program dinamis dan brute force, implementasi ini juga menggunakan struct yang sama, yakni Tanaman dan Aksi. Terdapat juga penggunaan fungsi yang sama, yakni tampilkan_lahan. Logika utama greedy terdapat pada fungsi main.

```cpp
sort(daftar_tanaman.begin(), daftar_tanaman.end(), [](const Tanaman& a, const Tanaman& b) {
    return (double)a.keuntungan() / a.waktu_panen > (double)b.keuntungan() / b.waktu_panen;
});

int modal = modal_awal;
int total_petak = baris * kolom;
lahan_tersedia.assign(total_petak, 0);

for (int hari = 1; hari <= jumlah_hari; hari++) {
    int batas = jumlah_hari / 2 + (jumlah_hari % 2 > 0 ? 1 : 0);
    int musim = (hari <= batas) ? 0 : 1;
    for (int petak = 0; petak < total_petak; petak++) {
        if (lahan_tersedia[petak] < hari) {
            for (int i = 0; i < jumlah_tanaman; i++) {
                Tanaman& t = daftar_tanaman[i];
                if (t.musim == musim && modal >= t.harga_beli && hari + t.waktu_panen <= jumlah_hari) {
                    lahan_tersedia[petak] = hari + t.waktu_panen;
                    modal -= t.harga_beli;
                    aksi_terjadi.push_back({hari, petak, i, hari + t.waktu_panen});
                    break;
                }
            }
        }
    }
}
```

Gambar 3.5 Algoritma greedy di fungsi main

(Sumber: Arsip pribadi penulis)

Implementasi ini menerapkan pendekatan greedy untuk memaksimalkan keuntungan dengan cara membuat keputusan yang paling optimal secara lokal pada setiap langkah. Hal pertama yang dilakukan adalah mengurutkan semua jenis tanaman berdasarkan metrik efisiensi tertinggi, yaitu rasio keuntungan per hari (keuntungan/waktu_panen) sehingga tanaman yang paling cepat menghasilkan profit berada di urutan teratas. Untuk setiap harinya, fungsi ini akan memeriksa setiap petak lahan yang kosong. Untuk petak yang tersedia, akan ditelusuri daftar tanaman yang sudah terurut dan langsung menanam tanaman pertama yang ditemui yang memenuhi semua syarat, yaitu musimnya cocok, modalnya mencukupi, dan waktu panennya tidak melewati batas akhir jumlah hari. Begitu satu tanaman berhasil ditanam, pencarian untuk petak tersebut langsung berhenti dan program beralih ke petak berikutnya. Pendekatan ini secara konsisten memilih opsi yang "terlihat" paling baik saat ini, dengan harapan akan menghasilkan keuntungan total yang tinggi di akhir permainan.

IV. Analisis dan Pengujian

Untuk membandingkan keefisienan dari ketiga pendekatan, yakni program dinamis, brute force, dan greedy, akan dilakukan pengujian untuk masing-masing algoritma dengan *test case* yang sama. Format input pada implementasi ini adalah sebagai berikut:

```
P L N jumlah_hari modal_awal
nama_tanaman1 harga_beli harga_jual waktu_panen musim
.
.
nama_tanamanN harga_beli harga_jual waktu_panen musim
```

A. Test Case 1

```
2 2 3 6 3000000
A 50 100 1 KEMARAU
B 60 120 2 KEMARAU
C 40 80 2 HUJAN
```

- Program Dinamis (Profit: 440, Waktu: 137 ms)

```
=== AKSI TANAM ===
Hari 1 (KEMARAU): Tanam A di petak (1,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (1,2) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,2) -> Panen hari 2 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam B di petak (1,1) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (1,2) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (2,1) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (2,2) -> Panen hari 5 (Keuntungan: 60)

=== RINCIAN PROFIT ===
A: 50
A: 50
A: 50
A: 50
B: 60
B: 60
B: 60
B: 60
Total keuntungan dihitung: 440

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 137 ms
```

- Brute Force (Profit: 440, Waktu: 1622 ms)

```
=== AKSI TANAM ===
Hari 1 (KEMARAU): Tanam A di petak (1,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (1,2) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,2) -> Panen hari 2 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam B di petak (1,1) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (1,2) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (2,1) -> Panen hari 5 (Keuntungan: 60)
Hari 3 (KEMARAU): Tanam B di petak (2,2) -> Panen hari 5 (Keuntungan: 60)

=== RINCIAN PROFIT ===
A: 50
A: 50
A: 50
A: 50
B: 60
B: 60
B: 60
B: 60
Total keuntungan dihitung: 440

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 1622 ms
```

- Greedy (Profit: 400, Waktu: 0 ms)

```
=== AKSI TANAM ===
Hari 1 (KEMARAU): Tanam A di petak (1,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (1,2) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,2) -> Panen hari 2 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam A di petak (1,1) -> Panen hari 4 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam A di petak (1,2) -> Panen hari 4 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam A di petak (2,1) -> Panen hari 4 (Keuntungan: 50)
Hari 3 (KEMARAU): Tanam A di petak (2,2) -> Panen hari 4 (Keuntungan: 50)

=== RINCIAN PROFIT ===
A: 50
A: 50
A: 50
A: 50
A: 50
A: 50
A: 50
A: 50
Total keuntungan dihitung: 400

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 0 ms
```

B. Test Case 2

```
2 3 4 4 300000
A 50 100 1 KEMARAU
B 60 120 2 KEMARAU
C 40 80 3 HUJAN
D 100 103 1 HUJAN
```

- Program Dinamis (Profit: 360, Waktu: 415 ms)

```
=== AKSI TANAM ===
Hari 2 (KEMARAU): Tanam B di petak (1,1) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (1,2) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (1,3) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,1) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,2) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,3) -> Panen hari 4 (Keuntungan: 60)

=== RINCIAN PROFIT ===
B: 60
B: 60
B: 60
B: 60
B: 60
B: 60
Total keuntungan dihitung: 360

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 415 ms
```

- Brute Force (Profit: 360, Waktu: 7396 ms)

```
=== AKSI TANAM ===
Hari 2 (KEMARAU): Tanam B di petak (1,1) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (1,2) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (1,3) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,1) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,2) -> Panen hari 4 (Keuntungan: 60)
Hari 2 (KEMARAU): Tanam B di petak (2,3) -> Panen hari 4 (Keuntungan: 60)

=== RINCIAN PROFIT ===
B: 60
B: 60
B: 60
B: 60
B: 60
B: 60
Total keuntungan dihitung: 360

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 7396 ms
```

- Greedy (Profit: 400, Waktu: 0 ms)

```
=== AKSI TANAM ===
Hari 1 (KEMARAU): Tanam A di petak (1,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (1,2) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (1,3) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,1) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,2) -> Panen hari 2 (Keuntungan: 50)
Hari 1 (KEMARAU): Tanam A di petak (2,3) -> Panen hari 2 (Keuntungan: 50)
Hari 3 (HUJAN): Tanam D di petak (1,1) -> Panen hari 4 (Keuntungan: 3)
Hari 3 (HUJAN): Tanam D di petak (1,2) -> Panen hari 4 (Keuntungan: 3)
Hari 3 (HUJAN): Tanam D di petak (1,3) -> Panen hari 4 (Keuntungan: 3)
Hari 3 (HUJAN): Tanam D di petak (2,1) -> Panen hari 4 (Keuntungan: 3)
Hari 3 (HUJAN): Tanam D di petak (2,2) -> Panen hari 4 (Keuntungan: 3)
Hari 3 (HUJAN): Tanam D di petak (2,3) -> Panen hari 4 (Keuntungan: 3)

=== RINCIAN PROFIT ===
A: 50
A: 50
A: 50
A: 50
A: 50
A: 50
D: 3
D: 3
D: 3
D: 3
D: 3
D: 3
Total keuntungan dihitung: 318

=== WAKTU EKSEKUSI ===
Program dieksekusi dalam 0 ms
```

C. *Analisis Hasil Pengujian*

Berdasarkan hasil pengujian, pendekatan yang memberikan hasil paling optimal adalah brute force dan program dinamis, karena keduanya mampu mengeksplorasi seluruh kemungkinan dengan akurat. Dari sisi efisiensi waktu, urutan tercepat hingga terlambat adalah greedy, diikuti oleh program dinamis, lalu brute force. Temuan ini mendukung hipotesis bahwa program dinamis merupakan pendekatan paling seimbang, karena mampu memberikan hasil yang akurat sekaligus efisien. Meskipun brute force akurat, metode ini sangat tidak efisien untuk test case yang besar karena konsumsi waktu dan memori yang tinggi. Di sisi lain, meskipun greedy unggul dalam kecepatan, ia tidak dapat menjamin hasil yang optimal. Oleh karena itu, dapat disimpulkan bahwa untuk test case kecil hingga sedang, program dinamis adalah pendekatan yang paling unggul di antara ketiganya.

V. KESIMPULAN

Dalam konteks sektor pertanian, efisiensi penjadwalan penanaman sangat krusial untuk memaksimalkan hasil dan keuntungan, terutama ketika dihadapkan pada keterbatasan waktu, musim, dan modal. Pemanfaatan pendekatan algoritmik dalam menyusun penjadwalan tanam menjadi semakin penting seiring meningkatnya kompleksitas variabel agrikultur modern. Berdasarkan hasil pengujian, pendekatan program dinamis terbukti sebagai solusi paling seimbang. Ia mampu memberikan keuntungan optimal seperti brute force, namun dengan waktu eksekusi yang jauh lebih efisien. Sementara itu, pendekatan greedy menunjukkan keunggulan dalam kecepatan, tetapi cenderung mengabaikan keputusan jangka panjang sehingga hasil akhirnya kurang optimal. Di sisi lain, meskipun brute force memberikan hasil terbaik, metode ini sangat tidak praktis untuk diterapkan pada skala besar karena eksplorasi menyeluruh terhadap semua kemungkinan membutuhkan sumber daya yang sangat besar. Dengan mempertimbangkan karakteristik masalah penjadwalan dalam dunia pertanian yang dinamis dan penuh batasan, program dinamis menjadi pendekatan yang paling direkomendasikan untuk diterapkan dalam skenario nyata, khususnya untuk lahan berukuran kecil hingga menengah.

LINK REPOSITORI GITHUB

https://github.com/rararana/Farming-Scheduler

UCAPAN TERIMA KASI

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan makalah yang berjudul "Pendekatan Program Dinamis untuk Maksimalisasi Keuntungan dalam Sistem Penanaman Berjangka Waktu" dengan baik. Penulis juga mengucapkan terima kasih kepada pihak-pihak yang telah mendukung dalam penulisan makalah ini, yaitu:

1. Ibu Dr. Nur Ulfa Maulidevi, Bapak Dr. Rinaldi Munir, Bapak Dr. Rila Mandala, serta Bapak Monterico Adrian, S.T, M.T selaku dosen-dosen pengajar IF2211 Strategi Algoritma atas ilmu, pengajaran, dan bimbingan yang telah diberikan kepada penulis selama perkuliahan,
2. Orang tua penulis yang telah memberikan dukungan dan semangat hingga saat ini,

3. Teman-teman penulis yang telah berjuang bersama-sama dan memberikan dukungan dalam menyelesaikan makalah ini,

Akhir kata, penulis berharap makalah ini dapat memberikan wawasan serta mendorong pembaca untuk mengeksplorasi algoritma graf lebih dalam.

REFERENSI

[1] [1] Cybex, "Penentuan jadwal tanam yang tepat untuk hasil panen optimal," cybex.id, 2022. https://cybex.id/artikel/103061/penentuan-jadwal-tanam-yang-tepat-untuk-hasil-panen-optimal/ (diakses pada 23 Juni 2025)

[2] TOKI, "Pemrograman Kompetitif Dasar," osn.toki.id. https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf (diakses pada 23 Juni 2025)

[3] Japan International Cooperation Agency (JICA), "Training of Trainers on SHEP approach," jica.go.jp, Dec. 2024. https://www.jica.go.jp/english/activities/issues/agricul/shep/__icsFiles/afieldfile/2024/12/18/tot7_1.pdf (diakses pada 23 Juni 2025)

[4] University of Massachusetts Amherst, "Scheduling greenhouse crops," umass.edu, n.d. https://www.umass.edu/agriculture-food-environment/greenhouse-floriculture/fact-sheets/scheduling-greenhouse-crops (diakses pada 23 Juni 2025)

[5] GeeksforGeeks, "Introduction to dynamic programming - data structures and algorithm tutorials," geeksforgeeks.org, n.d. https://www.geeksforgeeks.org/dsa/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/ (diakses pada 24 Juni 2025)

[6] R. Munir, "Program Dinamis (2025) Bagian 1," Informatika STEI ITB, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf (diakses pada 24 Juni 2025)

[7] R. Munir, "Algoritma Greedy (2025) Bagian 1," Informatika STEI ITB, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf (diakses pada 24 Juni 2025)

PERNYATAAN

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

Bandung, 24 Juni 2025

Ranashahira Reztaputri 13523007