Persoalan Knapsack 1/0 Dengan Program Dinamis

Tony 13516010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
buddy90_lost@yahoo.co.id

***Abstrak*—Persoalan knapsack 1/0 merupakan persoalan yang terkenal di kalangan *computer scientist* maupaun *software engineer*. Persoalan knapsack 1/0 merupakan persoalan NP-Complete, namun dengan menggunakan program dinamis, persoalan knapsack 1/0 dapat di selesaikan dengan *pseudo polynomial time*.**

***Kata kunci*—knapsack 1/0, program dinamis**

I. PENDAHULUAN

Persoalan knapsack 1/0 dapat digunakan untuk memodelkan persoalan di kehidupan sehari-hari. Banyak metode atau algoritma yang dapat digunakan untuk menyelesaikan persoalan knapsack, seperti greedy yang memiliki waktu penyelesaian polinomial namun belum tentu optimal, bruteforce yang memiliki waktu penyelesaian exponensial, dan backtracking yang memiliki waktu penyelesaian exponensial. Program dinamis dapat digunakan untuk menyelesaikan persoalan knapsack 1/0 dengan waktu *pseudo-polynomial*.

Makalah ini mengaplikasikan algoritma atau metode penyelesai masalah program dinamis yang merupakan salah satu materi mata kuliah "Strategi Algoritma".

II. PROGRAM DINAMIS

Program dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan sedemikian sehingga solusi dari serangkaian keputusan yang saling berkaitan. Program dinamis mengkomputasi solusi dengan cara menyimpan hasil komputasi seluruh upapersoalan yang optimum yang kemudian dibangun untuk mendapatkan solusi persoalan utama.

Program dinamis memiliki prinsip optimalitas, yaitu jika solusi global optimal, maka bagian solusi juga optimal.

Program dinamis dapat diterapkan pada persoalan yang memiliki karakteristik berikut.

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang tiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari atas sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Status merupakan kemungkinan masukan yang ada pada tahap tersebut.
3. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Status merupakan kemungkinan masukan yang ada pada tahap tersebut.
4. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Status merupakan kemungkinan masukan yang ada pada tahap tersebut.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k* + 1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.

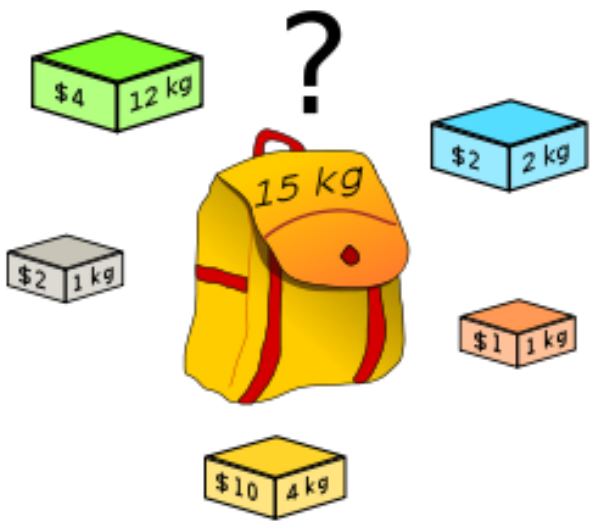
Terdapat dua jenis pendekatan yang dapat digunakan dalam menyelesaikan persoalan dengan pemrograman dinamis, yaitu

1. Pendekatan *top-down*, pendekatan ini adalah pendekatan yang menggunakan formulasi rekursif. Jika solusi dari suatu persoalan dapat diformulasikan secara rekursif menggunakan solusi dari upa-persoalannya, dan jika upa-persoalannya tumpang tindih, maka dapat solusi dari upa-persoalnnya dapat di memorisasi. Pendekatan ini melakukan kalkulasi pada upa-masalah yang diperlukan saja.
2. Pendekatan *bottom-up*, setelah ada formula rekursif dari persoalan, formula persoalan dapat diubah menjadi gaya *bottom-up* (mencoba menyelesaikan sub-problemnya terlebih dahulu, setelah itu hasilnya digunakan untuk menyelesaikan upa-persoalan yang lebih besar). Pendekatan ini melakukan kalkulasi pada semua upa-masalah.

III. *KNAPSACK 1/0*

Persoalan knapsack adalah persoalan diberikan beberapa barang yang tiap barangnya memilikan bobot (*weight*) w_i dan nilai (*value*) v_i, tentukan nilai maksimal yang bisa didapatkan jika total beratnya dibatasi. Persoalan *knapsack* dapat dituliskan dalam sebagai berikut.

Maksimalkan nilai $\sum_i^n v_i x_i$
dengan batasan $\sum_i^n w_i x_i$
dengan $x_i \in \{0,1\}$
dimana solusi = $\{x_1,\ x_2,\ \ldots,\ x_n\}$



Gambar 1. Ilustrasi Persoalan *Knapsack 1/0*

III. Penyelesaian *Knapsack 1/0*

Penyelesaian *knapsack 1/0* dapat dilakukan dengan mengikuti petunjuk diatas. Berikut kode implementasi solusi *knapsack 1/0* dengan menggunakan program dinamis dalam bahasa C++.

```
#define dummyval -1

struct item{
    int value;
    int weight;
    item(int v, int w){
        value = v;
        weight = w;
    }
};

vector<item> items;
vector<vector<int>> dp;

int solve_knapsack(int idx_item, int remain_cap){
    if(idx_item == items.size()){
        return 0;
    }
    int &solution = dp[idx_item][remain_cap];
    if(solution == dummyval){
        const item &alias_item = items[idx_item];
        solution = solve_knapsack(idx_item + 1,
remain_cap);
        if(remain_cap >= alias_item.weight){
            solution = max(solution,
solve_knapsack(idx_item + 1, remain_cap -
alias_item.weight) + alias_item.value);
        }
    }
    return solution;
}

void print_item(int capacity){
    int i;
    cout<<"{";
    for(i=0;i<items.size()-1;++i){
        if(dp[i][capacity] == dp[i+1][capacity]){
            cout<<"0";
        }
        else{
            cout<<"1";
            capacity -= items[i].weight;
        }
        cout<<", ";
    }
    if(dp[i][capacity] > 0){
        cout<<"1";
    }
    else{
        cout<<"0";
    }
    cout<<"}\n";
}
```

Tabel 1. Kode program dinamis dengan pendekan top-down

```
int solve_knapsack(int capacity){
    for(int i=items.size()-1;i>=0;--i){
        dp[i][0] = 0;
    }
    for(int i=0;i<=capacity;++i){
        dp[items.size()][i] = 0;
    }
    for(int i=items.size()-1;i>=0;--i){
        for(int j=1;j<=capacity;++j){
            dp[i][j] = dp[i+1][j];
            if(j >= items[i].weight){
                dp[i][j] = max(dp[i][j],
dp[i+1][j-items[i].weight] + items[i].value);
            }
        }
    }
    return dp[0][capacity];
}
```

Tabel 2. Kode program dinamis dengan pendekatan bottom-up

Pada algoritma program dinamis diatas, kompleksitas waktu dan ruangnya adalah O(NM), dimana N adalah banyak benda dan M adalah kapasitas knapsack. Berikut perbandingan waktu eksekusi kedua program dinamis dengan pendekatan yang berbeda .

Banyak Objek/Kapasitas	Top-down (ms)	Bottom-up (ms)
100/200	2.893	3.003
1000/200	15.025	13.014
1000/1000	58.963	53.953
2000/2000	239.369	149.092
4000/4000	904.256	831.278

Tabel 3. Perbandingan waktu antara alogritma program dinamis dengan berbeda pendekatan.

Dari tabel 3, dapat disimpulkan bahwa program dinamis yang menggunakan pendekatan *bottom-up* lebih cepat dibandingkan dengan pendekata *top-down*.

IV. Penyelesaian Knapsack 1/0 Dengan Banyak Tas

Persoalan knapsack dengan banyak tas merupakan persolaan yang mirip dengan sebelumnya namun disini terdapat lebih dari satu tas yang mana satu benda hanya bisa masuk ke salah satu tas. Persoalan ini berbeda dengan menjumlahkan semua kapasitas tas menjadi satu. Berikut kode implementasi solusi *knapsack 1/0* dengan 4 tas dengan menggunakan program dinamis pada C++.

```
vector<item> items;
int dp[30][40][40][40][40];

int solve_knapsack(int idx_item, int cap1, int
cap2, int cap3, int cap4){
    if(idx_item == items.size()){
        return 0;
    }
    int &solution =
dp[idx_item][cap1][cap2][cap3][cap4];
    if(solution == dummyval){
        const item &alias_item = items[idx_item];
        solution = solve_knapsack(idx_item+1,
cap1, cap2, cap3, cap4);
        if(cap1 >= alias_item.weight){
            solution = max(solution,
solve_knapsack(idx_item+1, cap1 -
alias_item.weight, cap2, cap3, cap4) +
alias_item.value);
        }
        if(cap2 >= alias_item.weight){
            solution = max(solution,
solve_knapsack(idx_item+1, cap1, cap2 -
alias_item.weight, cap3, cap4) +
alias_item.value);
        }
        if(cap3 >= alias_item.weight){
            solution = max(solution,
solve_knapsack(idx_item+1, cap1, cap2, cap3 -
alias_item.weight, cap4) + alias_item.value);
        }
        if(cap4 >= alias_item.weight){
            solution = max(solution,
solve_knapsack(idx_item+1, cap1, cap2, cap3, cap4
- alias_item.weight) + alias_item.value);
        }
    }
    return solution;
}

void print_item(int cap1, int cap2, int cap3, int
cap4){
    cout<<"{";
    int i;
    for(i=0;i<items.size()-1;++i){
        if(dp[i][cap1][cap2][cap3][cap4] ==
dp[i+1][cap1][cap2][cap3][cap4]){
            cout<<"0";
        }
        else{
            if(cap1>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] == dp[i+1][cap1-
items[i].weight][cap2][cap3][cap4] +
items[i].value){
                cout<<"1";
                cap1 -= items[i].weight;
            }
            else if(cap2>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] ==
dp[i+1][cap1][cap2-items[i].weight][cap3][cap4] +
items[i].value){
                cout<<"2";
                cap2 -= items[i].weight;
            }
            else if(cap3>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] ==
dp[i+1][cap1][cap2][cap3-items[i].weight][cap4] +
items[i].value){
                cout<<"3";
                cap3 -= items[i].weight;
                      }
            else{
                cout<<"4";
                cap4 -= items[i].weight;
            }
        }
        cout<<", ";
    }
    if(dp[i][cap1][cap2][cap3][cap4] > 0){
        if(cap1>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] == dp[i+1][cap1-
items[i].weight][cap2][cap3][cap4] +
items[i].value){
            cout<<"1";
            cap1 -= items[i].weight;
        }
        else if(cap2>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] ==
dp[i+1][cap1][cap2-items[i].weight][cap3][cap4] +
items[i].value){
            cout<<"2";
            cap2 -= items[i].weight;
        }
        else if(cap3>=items[i].weight &&
dp[i][cap1][cap2][cap3][cap4] ==
dp[i+1][cap1][cap2][cap3-items[i].weight][cap4] +
items[i].value){
            cout<<"3";
            cap3 -= items[i].weight;
        }
        else{
            cout<<"4";
            cap4 -= items[i].weight;
        }
    }
    else{
        cout<<"0";
    }
    cout<<"}\n";
}
```

Tabel 4. Kode program dinamis dengan pendekan *top-down*

```
int solve_knapsack(int cap1, int cap2, int cap3,
int cap4){
   memset(dp, 0, sizeof dp);
```

```
   for(int i=items.size()-1;i>=0;--i){
      for(int j=1;j<=cap1;++j){
         for(int k=1;k<=cap2;++k){
            for(int l=1;l<=cap3;++l){
               for(int m=1;m<=cap4;++m){
                  dp[i][j][k][l][m] =
dp[i+1][j][k][l][m];
                  if(j >= items[i].weight){
                     dp[i][j][k][l][m] =
max(dp[i][j][k][l][m], dp[i+1][j-
items[i].weight][k][l][m] + items[i].value);
                  }
                  if(k >= items[i].weight){
                     dp[i][j][k][l][m] =
max(dp[i][j][k][l][m], dp[i+1][j][k-
items[i].weight][l][m] + items[i].value);
                  }
                  if(l >= items[i].weight){
                     dp[i][j][k][l][m] =
max(dp[i][j][k][l][m], dp[i+1][j][k][l-
items[i].weight][m] + items[i].value);
                  }
                  if(m >= items[i].weight){
                     dp[i][j][k][l][m] =
max(dp[i][j][k][l][m], dp[i+1][j][k][l][m-
items[i].weight] + items[i].value);
                  }
               }
            }
         }
      }
   }
   return dp[0][cap1][cap2][cap3][cap4];
}
```

Tabel 5. Kode program dinamis dengan pendekan bottom-up

Dari kedua algoritma diatas, kompleksitas waktu dan ruangnya adalah O((banyak objek)*(perkalian semua kapasitas bag)). Kompleksitas ruang dapat dikurangi dengan menggunakan pendekan *top-down* yang hanya menghitung solusi dari upa-persoalan yang diperlukan sehingga pada tabel tidak semua tabel digunakan dan hanya perlu disimpan state yang digunakan. Hal tersebut dapat diimplementasikan dengan menggunakan struktur data map di C++.

```
map<vector<int>, int> dp[30];

int solve_knapsack(int idx_item, vector<int>
&cap){
   if(idx_item == items.size()){
      return 0;
   }
   if(!dp[idx_item].count(cap)){
      int &solution = dp[idx_item][cap];
      const item &alias_item = items[idx_item];
      solution = solve_knapsack(idx_item+1, cap);
      for(int i=0;i<cap.size();i++){
         if(cap[i] >= alias_item.weight){
            cap[i] -= alias_item.weight;
             solution = max(solution,
solve_knapsack(idx_item+1, cap)+
alias_item.value);
            cap[i] += alias_item.weight;
         }
      }
   }
   return dp[idx_item][cap];
}
```

Tabel 6. Kode program dinamis dengan pendekan *top-down* (optimasi memori)

Banayk objek/ kap1/kap2/kap3/kap4	Top-down	Bottom-up	Top-down (optimasi)
10/10/10/10	359.433	367.565	90.010
20/15/15/15	322.367	357.545	634.853
20/20/20/20	769.178	736.127	38468.569

Tabel 7. Perbandingan waktu antara alogritma program dinamis dengan berbeda pendekatan.

Dari tabel 7, dapat disimpulkan bahwa optimasi memori yang dilakukan tidak lebih cepat dari pada yang tidak dioptimasi memorinya pada data yang besar. Hal ini karena pengaksesan struktur dapat map memiliki waktu O(log N) sedangkan pada tabel program dinamis tanpa optimisasi pengaksesan tabel membutuhkan waktu O(1).

V. KESIMPULAN

Program dinamis dapat digunakan untuk menyelesaikan persoalan optimasi karena tiap tahap dari persoalan dimemorisasi, namun kelemahannya ada di keterbatasan memori. Pada persoalan *knapsack 1/0* dengan banyak tas, keterbatasan memori dapat diatasi dengan menggunakan struktur data *map* pada c++ dan *java* ataupun *dict* di *python*, namun hal tersebut memperbesar waktu pengaksesan tabel dari O(1) menjadi O(log N).

Persoalan *knapsack 1/0* dapat diselesaikan dengan program dinamis dalam waktu *pseudo polynomial time*. Persoalan *knapsack 1/*0 dapat digunakan untuk memodelkan persoalan-persoalan di kehidupan sehari-hari seperti persoalan kontainer pengiriman.

REFERENCES

[1] Munir, Rinaldi, Diktat Kuliah: Strategi Algoritma, Bandung: Departemen Teknik Informatika, 2009.

[2] Kellerer, et al., “Knapsacks Problems”, Springer, 2004.

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, 3 Desember 2017

Ttd (scan atau foto ttd)

Tony
13516010