Perbandingan Algoritma yang dipakai dalam 2D Knapsack Problem

Kevin Tanadi (13506120)

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : streptomicin2001@yahoo.com

ABSTRAK

Permasalahan Knapsack yang biasa ditemui adalah persoalan Integer Knapsack dimana terdapat n buah objek dan sebuah knapsack(karung, tas, buntilan, dsb) dengan kapasitas bobot k. Setiap objek memiliki sebuah bobot(weight) w dan keuntungan(profit) p.

Tujuan dari permasalahan ini adalah dengan mengunakan Knapsack yang hanya dapat menampung beban maksimal k mendapatkan keuntungan sebesar - besarnya. Persoalan Knapsack ini dapat diselesaikan dengan mudah oleh algoritma Program Dinamis. Sekarang kita memiliki sebuah masalah baru yang juga sering ditemui dalam kehidupkan sehari - hari. Kadang kita memiliki sebuah Knapsack yang memiliki kapasitas bukan hanya satu dimensi, melainkan dua dimensi, seperti dalam permasalahan :

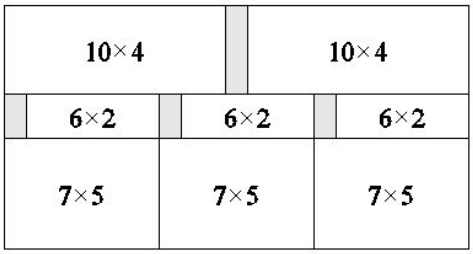
- **Kita memiliki sebuah kertas yang berukuran p x l**
- **Kita diminta untuk membuat n buah persegi yang setiap perseginya memiliki ukuran pi x li**
- **Berapa luas kertas maksimum yang bisa terpakai atau dengan kata lain, jumlah kertas yang tidak dipakai adalah minimum (setiap persegi bisa dipakai lebih dari sekali)?**

Kata kunci: Knapsack, Program Dinamis, 2D Knapsack, *Divide and Conquer,*.Brute Force, Runut Balik, *GreedyAlgorithm*.

1. PENDAHULUAN

Seorang pengrajin emas, diperintah raja untuk membuat kepingan - kepingan emas berbentuk persegi panjang dengan ukuran p1 l1, p2 l2, p3 l3,..pn..ln dari sebuah kepingan emas berbentuk persegi panjang berukuran P x L. Raja memerintah pengrajin tersebut agar jumlah emas yang tidak terpakai sesedikit mungkin.

Misalkan ukuran Kepingan emas adalah 21 x 11, dan persegi - persegi yang dapat dibuat berukuran 10 x 4, 6 x 2, 7 x 5, dan 15 x 10



Untuk permasalahan diatas jumlah kertas maksimum yang dapat dipakai adalah 221 satuan luas, dengan konfigurasi seperti diatas.

2. METODE

Persoalan seperti ini merupakan sebuah salah satu persoalan 2D knapsack yang sering ditemui, kita dapat menganalogikan kepingan emas tersebut sebagai kertas. Ada beberapa metode yang bisa kita lakukan untuk menyelesaikan masalah ini:

2.1 Kombinasi Algoritma Brute Force dan Runut Balik

2.1.1 Penjelasan Brute Force

Algoritma Brute Force adalah sebuah pendekatan yang lempang (staightforward) untuk memecahkan suatu masalah, biasanya didasarkan langsung pada peryataan masalah (problem statement) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas.

2.1.2 Penjelasan Algoritma Runut Balik

Runut balik adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik, yang merupakan perbaikan dari algoritma

brute-force, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan metode ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah pada solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat.

Algoritma Runut balik memastikan kebenaran dengan menenumerasi semua kemungkinan dan memastikan efisiensi dengan tidak pernah kembali pada sebuah state yang sama lebih dari satu kali. Aplikasi teknik rekursif menghasilkan sebuah metode yang elegan dan mudah diimplementasikan dari algoritma runut balik ini. Karena dengan sifat dasar dari program rekursif yang selalu kembali kestate sebelumnya setelah selesai dieksekusi menghasilkan implementasi yang mengagumkan pada algoritma runut balik ini.

Salah satu hal penting dari algoritma runut balik adalah bagian pemotongan (pruning), mengetahui dengan cepat pada state awal apakah suatu state merupakan state solusi adalah sangat penting untuk mencegah agar pohon yang dibuat tidak terlalu dalam. Salam satu implementasi yang penting dari pemotongan ini adalah penghilangan simetri, meskipun metode diatas tidak selalu dapat diimplementasikan.

2.1.3 Implementasi

Kita menghitung semua kemungkinan yang mungkin persegi apa sajakah yang akan dibuat, yang direpresentasikan dengan himpunan persegi - persegi dan kemudian mengecek apakan semua persegi tersebut dapat dibuat dengan kertas berukuran p x l tersebut, jika bisa, kita akan membandingkan jumlah kertas terpakai tersebut dengan jumlah kertas terpakai maksimum yang telah didapat dan jika jumlah persegi tersebut lebih besar dari jumlah persegi maksimum, maka jumlah kertas tersebut menjadi jumlah kertas maksimum. Jumlah kemungkinan persegi yang terpakai adalah sebanyak 2^n buah, dan jumlah langkah yang dipakai untuk mengecek apakah seluruh persegi tersebut dapat dibuat dari kertas tersebut adalah (p x l)^n, sehingga kompleksitas akhirnya adalah sebesar (2 + p x l)^n.

Algoritma runut-balik yang dipakai untuk mengecek berapakah jumlah kertas maksimum yang dapat dipakai oleh himpunan persegi tersebut:

- Misalkan terdapat k buah persegi yang dipakai.
- Representasikan kertas sebagai matriks berukuran p x l yang kosong(semua elemennya berisi nilai 0).
- Untuk setiap persegi uji apakah terdapat sub-matriks kosong yang berukuran X x Y
- Jika ada, isi sub-matriks tersebut dengan nilai 1
- Uji apakah persegi selanjutnya dapat dibuat lagi atau tidak
- Jika tidak, maka hapus sub-matriks tersebut (isi nilai kembali dengan 0 atau di runut-balik) dan cari sub-matriks lain yang masih memenuhi.
- Ulangi sampai terdapat solusi atau tidak terdapat lagi kemungkinan.

2.2 Metode Greedy

2.2.1 Penjelasan Algoritma Greedy

Secara harfiah Greedy artinya rakus dan tamak. Seperti namanya prinsip Greedy adalah “mengambil apa yang dapat anda ambil sekarang tanpa memikirkan konsekuensi kedepan.

Algoritma Greedy membentuk solusi langkah per langkah (step by step). Terkadang banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus diambil keputusan yang terbaik Keputusan yang telah diambil tidak dapat diubah lagi.

Algoritma Greedy, selalu mencari keputusan lokal terbaik dalam setiap langkah,yang secara perlahan akhirnya akan menghasilkan solusi global.

2.2.2 Impementasi

Pendekatan dengan Algoritma *Greedy* adalah sebagai berikut :

- Urutkan seluruh persegi berdasarkan luasnya (panjang dikali dengan lebar). D
- Untuk setiap persegi gunakan kertas seluas yang diperlukan.
- Ulangi sampai tidak terdapat lagi cukup kertas untuk membuat persegi lain.

Kompleksitas dari algortima ini adalah n x p x l. Metode ini akan menghasilkan jawaban yang cukup baik, namun tidak selalu menghasilkan solusi yang optimal. Untuk itu kita perlu memodifikasi untuk meningkatkan kemungkinan didapatkannya algoritma yang optimal. Ide dasarnya adalah bahwa jika kita dapat membuat n buah persegi, maka pasti kita juga pasti dapat membuat minimal n buah persegi terkecil yang telah diurutkan.

Oleh karena itu kita akan mencoba untuk menghasilkan jawaban secara iteratif, pertama dengan mengunakan sebuah persegi dan kemudian mengecek

dengan algoritma runut – balik seperti pada bagian 2.1 apakah persegi tersebut dapat dibuat dengan kertas yang tersisa. Dengan mengunakan kombinasi algoritma ini, maka meskipun kompleksitas meningkat dari n x p x l, menjadi n x (p x l) ^ n, metode ini akan selalu menghasilkan jawaban yang jauh lebih baik dibanding algoritma sebelumnya.

2.3 Metode *Divide and Conquer*

2.3.1 Penjelasan *Divide and Conquer*

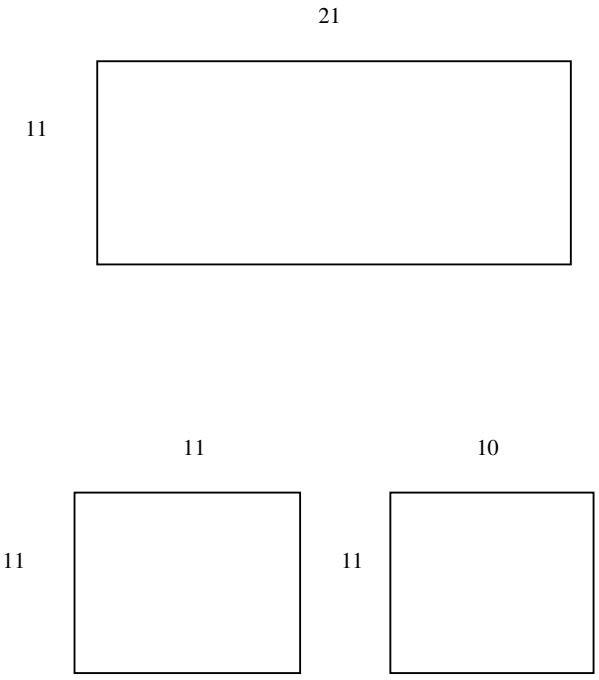
Divide and Conquer adalah sebuah metode dimana kita memecahkan sebuah masalah menjadi dua buah masalah yang berukuran lebih kecil, dan kemudian menyelesaikannya secara rekursif sampai basis tertentu, dan kemudian mengabungkannya hasilnya kembali.

Secara umum algoritma ini memiliki tiga skema umum yaitu :

- *Divide* : Membagi masalah menjadi beberapa upa masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil.
- *Conquer* : Memecahkan atau menyelesaikan masing – masing upa masalah
- *Combine* : Mengabungkan solusi masing – masing sehingga membentuk solusi masalah semula

2.3.2 Implementasi

Ide dasar dari permasalahan ini adalah kita dapat memotong kertas menjadi dua buah kertas yang berbeda dengan ukuran p x l1 dan p x l2 (memotong secara vertikal) atau p1 x l dan p2 x l (memotong secara horizontal). Kemudian setelah di bagi menjadi dua bagian kita dapat mencoba untuk membagi kumpulan persegi menjadi dua himpunan yang berbeda yang kemudian mengecek berapa jumlah kertas maksimum yang bisa terpakai pada masing – masing kertas dan kemudian hasilnya akan dijumlahkan.



Gb1 .dipotong secara vertikal

Algoritma ini akan selalu menghasilkan jawaban yang optimal, dan memiliki kompleksitas yang bahkah jauh lebih baik dibanding dengan algoritma brute force, hal ini dikarenakan ketika membagi kertas menjadi dua maka akan terdapat lg(n) kemungkinan pemotongan, dan untuk jumlah kemungkinan himpunan persegi yang dapat dibentuk adalah sebanyak 2Cn = n^2. Sehingga komplesitas akhirnya menjadi n^(2lg(n)).

2.4 Metode Program Dinamis

2.4.1 Penjelasan Program Dinamis

Program Dinamis adalah sebuah teknik algoritma yang memanfaatkan formula rekurens dan sebuah state awal. Sebuah sub - solusi dari sebuah permasalahan di buat dari sub - solusi yang didapatkan sebelumnya. Solusi DP memiliki kompleksitas polinomial sehingga dipastikan memiliki waktu jalan yang jauh lebih cepat dibandingkan teknik lain seperti brute force, runut balik, dsb.

Yang dimaksud dengan state dalam program dinamis adalah sebuah cara untuk mendeskripsikan sesuatu, sebuah sub - solusi dari sebuah masalah.

Biasanya suatu program yang dapat diselesaikan dengan algoritma Program Dinamis juga memiliki metode pemecahan dengan mengunakan algoritma *Divide and Conquer*.

2.4.2 Implementasi

Metode Program Dinamis memanfaatkan properti yang ada pada metode *Divide and Conquer* diatas, bahwa setiap kertas dapat dibagi menjadi dua bagian yang berbeda dan kita dapat fokus untuk mengerjakannya secara berbeda.

Selain itu algoritma ini juga mengimplementasikan metode yang berbeda dalam membagi himpunan persegi menjadi dua bagian. Algoritma ini mempergunakan yang metode yang jauh lebih singkat dalam proses pembagian himpunan persegi yang terpakai.

Perbedaan mendasar dari Algoritma *Divide and Conquer* adalah algoritma Program Dinamis mengunakan tabel untuk mencatat setiap state yang terpakai agar tidak terjadi pengulangan dua kali, dan yang dihitung setiap kali perhitungan bukanlah jumlah kertas maksimum yang digunakan, melainkan jumlah kertas minimum yang terbuang (jumlah kertas yang digunakan maksimum apabila jumlah kertas yang terbuang minimum).

Misalkan a(X,Y) adalah luas kertas yang terbuang pada persegi(X,Y), $1 < x < P$, $1 < y < L$. Pada awalnya a(X,Y) bernilai xy, untuk setiap (X,Y) karena pada awalnya setiap kertas berukuran X x Y akan memiliki luas terbuang sebesar X x Y (Belum terpakai) kecuali untuk setiap (X,Y) dimana X = panjang persegi ke-n dan Y = lebar persegi ke-n, akan bernilai 0 (dianggap terpakai semua) sehingga a(X,Y) = 0.

Untuk setiap persegi (X,Y) misalkan C =1..X-1 adalah pemotongan secara vertikal atau C = 1..Y-1 adalah pemotongan secara horizontal, maka jumlah luas minimum yang terbuang adalah a(X,Y) = min(a(C,Y)+(X-C,Y) , a(X,C)+a(X,Y-C))

Jumlah tabel yang mungkin adalah sebanyak P x L, dan setiap tabel diproses sebanyak C kali, dimana C = maksimum dari P dan L, sehingga Kompleksitas akhirnya adalah max(P,L) x P x L.

IV. KESIMPULAN

Ada berbagai macam metode yang dapat digunakan untuk menyelesaikan permasalahan 2D Knapsack dan setiap metode memiliki keunggulan masing – masing.

- Algoritma *Greedy* sangat mudah untuk diimplementasikan dan memiliki komplesitas watku yang paling kecil dibanding algoritma-algoritma lainnya, namun tidak selalu menghasilkan jawaban yang optimal
- Algoritma kombinasi Brute Force dan Runut Balik, selalu menghasilkan jawaban yang optimal dan cukup mudah untuk di implementasikan, namun memiliki kompleksitas waktu yang sangat besar
- Algoritma *Divide and Conquer* sangat sulit untuk diimplementasikan, tapi memiliki komplesitas waktu yang jauh lebih baik dibanding kombinasi Brute Force dan Runut Balik dan selalu menghasilkan jawaban yang optimal
- Algoritma Program Dinamis memiliki kompleksitas waktu yang cukup baik dan tidak terlalu sulit untuk di implementasikan, namun memerlukan memori tambahan untuk membuat tabel sub - solusinya.

Kita dapat mengimplementasi algoritma diatas sesuai dengan keperluan kita, kita dapat mengimplementasi algoritma *Greedy* jika kita hanya memiliki sedikit waktu dan jawaban yang optimal tidak terlalu diperlukan. Kita dapat memilih metode Kombinasi *Brute-Force* dan Runut balik apabila ukuran kertas yang dipakai kecil. Dan kita dapat mengimplementasikan algoritma Program Dinamis apabila kita memiliki cukup waktu untuk mengimplementasikannya dan solusi yang optimal sangatlah penting.

REFERENSI

[1] Skiena, Steven. *The Algorithm Design Manual,* Springer-Verlag, 1997

[2] Cormen, Thomas H. *Introduction to Algorithms 2.ed, The MIT Press,* 2001

[3] Knuth, D. E. *The Art of ComputerProgramming, Vol. 1: Fundamental Algorithms, 3rd ed.* Reading, MA: Addison-Wesley, 1997.

[4] Munir, Rinaldi. Strategi Algoritmik. Departemen Teknik Informatika, Institut Teknologi Bandung, 2007

[5] Skiena, S. "Trees." *Implementing Discrete Mathematics: Combinatorics*