Penerapan Algoritma Program Dinamis dalam Mengoptimasi Jadwal Belajar

Candy Olivia Mawalim - 13513031
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13513031@std.stei.itb.ac.id

***Abstrak*—Optimasi jadwal belajar merupakan hal yang sangat dibutuhkan oleh pelajar. Penjadwalan yang tidak optimal dapat menimbulkan banyak dampak negatif kepada pelajar, mulai dari kehilangan nilai kedisiplinan, tanggung jawab yang diambil tidak dapat terselesaikan dengan baik, bahkan pola hidup yang kacau yang bisa berefek pada kesehatan pelajar itu sendiri. Salah satu solusi yang dapat digunakan untuk menyelesaikan masalah ini adalah algoritma pemrograman dinamis. Dalam makalah ini akan dijelaskan penerapan algoritma pemrograman dinamis dalam mengoptimasi jadwal belajar.**

***Kata Kunci*—Optimasi jadwal belajar, algoritma pemrograman dinamis, Graf.**

I. PENDAHULUAN

Pada zaman yang serba cepat ini, optimasi adalah hal yang sangat dipandang penting dalam setiap pekerjaan. Optimasi sangat diperlukan untuk meningkatkan tingkat keefektifan dan keefisienan suatu pekerjaan. Semakin optimal pekerjaan itu dilakukan, hasil yang didapatkan juga akan semakin optimal. Oleh karena itu, diperlukan suatu pengaturan baik dalam hal waktu maupun materi yang digunakan agar optimasi itu dapat tercapai. Dalam makalah ini akan dibahas secara lebih spesifik cara mengoptimasi waktu khususnya bagi pelajar dalam mengerjakan kewajibannya.

Seorang pelajar memiliki kewajiban baik itu untuk memahami hal yang dipelajari maupun mengerjakan tugas-tugas yang diberikan oleh pengajar dengan baik. Setiap hal yang dipelajari memiliki batasan waktu untuk dapat dikuasai. Begitu juga dengan tugas, tugas memiliki lama waktu dan prioritas pengerjakan. Untuk itu, penjadwalan sangat diperlukan agar setiap kewajiban ini dapat terlaksanakan dengan optimal.

Penjadwalan belajar yang tidak optimal adalah salah satu penyebab masalah terbesar yang dihadapi oleh setiap pelajar. Salah satu masalah yang dapat ditimbulkan akibat penjadwalan belajar yang kacau adalah tugas-tugas yang harus dikerjakan tidak terselesaikan dengan baik. Selain itu, penjadwalan seperti ini juga dapat menyebabkan kegiatan belajar hanya dilakukan untuk mengejar nilai sehingga tidak ada yang dipahami dalam proses belajar. Kegiatan-kegiatan yang berguna juga tidak dapat diikuti karena waktu yang digunakan tidak efektif. Dampak negatif terbesar adalah kesehatan pelajar semakin menurun karena pola hidup yang tidak teratur.

Dalam bidang informatika, dikenal beberapa algoritma yang dapat berperan dalam mengoptimasi penjadwalan. Salah satunya adalah algorima program dinamis. Algoritma ini memecahkan masalah dengan cara menguraikan solusi menjadi sekumpulan tahap sehingga solusi persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Faktor-faktor yang dapat digunakan sebagai parameter dalam pemecahan masalah penjadwalan belajar adalah waktu awal, deadline dan prioritas sebagai bobot (*cost*).

II. TEORI DASAR

A. Program Dinamis (Dynamic Programming)

Program dinamis (*dynamic programming*) adalah suatu algoritma pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahap (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Algoritma *dynamic programming* memiliki beberapa karakteristik penyelesaian, yaitu :

(1) Terdapat sejumlah berhingga pilihan yang mungkin.
(2) Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
(3) Kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

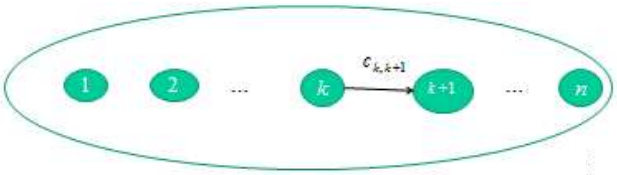
Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas. Prinsip ini berbunyi "jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal".

Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap *k* ke tahap *k* + 1, kita dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal. Biaya (*cost*) pada tahap *k* +1 = (biaya yang dihasilkan pada tahap *k+1*) + (biaya dari tahap *k* ke tahap 1). Secara matematis, biaya pada tahap k+1 dapat dituliskan sebagai berikut.

$$\mathbf{F_{k+1}(x) = c_{k+1} + F_k(x)}$$

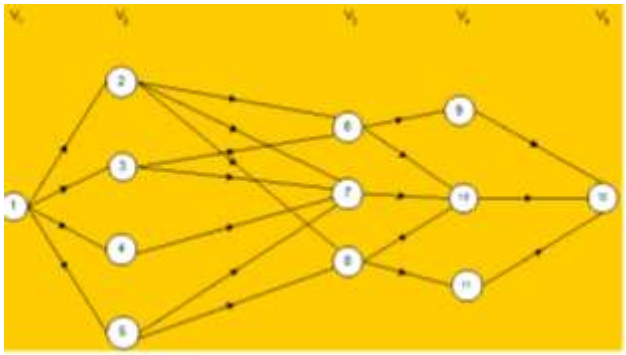
dengan :
$F_{k+1}(x)$ adalah *total cost* optimum hingga tahap ke k +1

C_{k+1} adalah *cost* optimum tahap ke k+1
$F_k(x)$ adalah *total cost* optimum hingga tahap ke k



Gambar 1. Prinsip optimalitas

Algoritma *dynamic programming* didukung oleh teori graf khususnya graf multi tahap (*multistage graph*). Tiap simpul dari graf tersebut menyatakan status, sedangkan V_1,V_2, .. menyatakan tahap.



Gambar 2. Graf multitahap

Karakteristik persoalan yang dapat menggunakan program dinamis sebagai solusinya adalah:

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil suatu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status yang berhubungan dengan tahap tersebut.
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
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.

Ada dua pendekatan yang dilakukan dalam program dinamis yaitu program dinamis maju dan progran dinamis mundur. Misalkan x_1, x_2, …, x_n menyatakan peubah (*variable*) keputusan yang harus dibuat masing-masing untuk tahap 1, 2, …, *n*. Maka,

1. Program dinamis maju. Program dinamis bergerak mulai dari tahap 1, terus maju ke tahap 2, 3, dan seterusnya sampai tahap *n*. Runtunan peubah keputusan adalah x_1, x_2, …, x_n.
2. Program dinamis mundur. Program dinamis bergerak mulai dari tahap *n*, terus mundur ke tahap *n* – 1, *n* – 2, dan seterusnya sampai tahap 1. Runtunan peubah keputusan adalah x_n, x_{n-1}, …, x_1.

Prinsip optimalitas program dinamis maju, ongkos pada tahap *k* +1 merupakan hasil penjumlahan dari ongkos yang dihasilkan pada tahap *k* dengan ongkos dari tahap *k* ke tahap *k* + 1, untuk *k* = 1, 2, …, *n* – 1. Sedangkan prinsip optimalitas pada PD mundur adalah ongkos pada tahap *k* merupakan hasil penjumlahan ongkos yang dihasilkan pada tahap (*k* + 1) dengan ongkos dari tahap (*k* + 1) ke tahap *k* , dengan *k* = *n*, *n* – 1, …, 1

Langkah-langkah pengembangan algoritma program dinamis adalah sebagai berikut.

1. Karakteristikkan struktur solusi optimal.
2. Definisikan secara rekursif nilai solusi optimal.
3. Hitung nilai solusi optimal secara maju atau mundur.
4. Konstruksi solusi optimal.

III. PEMBAHASAN

Persoalan penjadwalan jam belajar ini ekuivalen dengan penjadwalan kerja dengan deadline, keuntungan dan durasi waktu tertentu. Lebih jelasnya, masukan untuk persoalan ini akan terdiri dari beberapa informasi antara lain:

a. banyaknya pekerjaan (tugas atau belajar) yang akan dilakukan,
b. prioritas waktu pengerjaan yang bernilai non-negatif,
c. deadline setiap pekerjaan,
d. durasi (banyaknya waktu yang dibutuhkan) untuk mengerjakan pekerjaan tersebut.

Jadwal yang ditetapkan feasible jika memenuhi beberapa hal :

a. setiap pekerjaan selesai sebelum atau sesuai dengan deadline yang diberikan.
b. tidak ada dua pekerjaan yang dapat dilakukan secara bersamaan. Hal ini dikarenakan jam belajar yang fokus akan menghasilkan hasil yang optimal.

Dari pembahasan di atas, masukan yang diberikan dalam persoalan penjadwalan ini dapat disederhanakan sebagai sebuah list dari pekerjaan $(d_1,t_1,p_1),...,(d_n,t_n,p_n)$, dengan d_i adalah deadline pekerjaan ke-i, t_i adalah waktu yang diperlukan untuk mengerjakan pekerjaan ke-i, dan p_i adalah nilai prioritas pekerjaan ke-i. Keluaran yang diharapkan adalah sebuah jadwal yang *feasible* dan jadwal yang ditetapkan optimal (dinilai dari total prioritasnya maksimum).

Persoalan ini secara umum dapat diselesaikan dengan algoritma program dinamis. Namun, sebelum memulai pengerjaannya, kita terlebih dahulu perlu mengurutkan pekerjaan berdasarkan waktu akhir (deadline) tiap pekerjaan. Pekerjaan diurutkan mulai dari yang deadlinenya paling awal. Tahapan pemrograman dinamis kemudian dimulai. Ada 4 langkah/tahap pemilihan simpul:

Langkah 1 :

Persoalan dimodelkan dengan menggunakan array dua dimensi dimana dimensi pertama menyatakan surutan pekerjaan yang dilakukan dan dimensi kedua menyatakan waktu pengerjaannya. Array ini menyatakan status masing-masing simpul dalam graf.

$s = A(i,t)$ untuk $0 \le i \le n$ dan $0 \le t \le d$.

Langkah 2:

Menyatakan persoalan dengan relasi rekurens. Relasi rekurens tersebut menyatakan jadwal yang optimal (total nilai prioritasnya maksimal) dari status satu ke status lain.

$$f_1(s) = p_1 \text{ (Basis)}$$

$$f_k(s) = \max\{p_k + f_{k-1}(A(k,t))\} \text{ (Rekurens)}$$

Keterangan :

p_n menyatakan nilai prioritas dari satu status s ke A(k,t)

f_k (s) menyatakan nilai maksimum dari f_k (A(k,t),s)

f_k (A(k,t),s) menyatakan total bobot (*cost*) dari A(k,t) ke s

Langkah 3:

Mencari kemungkinan-kemungkinan penyelesaian dengan cara menciptakan array baru B[i,t] yang berisi kemungkinan penyelesaian setiap state. Pseudocode untuk mencari nilai B[i,t] adalah sebagai berikut.

```
for every t € {0,...,d}
  B[0,t]←0
end for

for i : 1..n
  for every t € {0,..., d}
    t' ← min{t, di} - ti
    if t'< 0 then
     B[i,t] ← B[i-1,t]
    else

B[i,t]←max{B[i-1,t],pi+B[i-1,t']}
    end if
  end for
end for
```

Langkah 4:

Hitung solusi optimal dari persoalan ini. Sebelum membuat prosedur rekursif dari persoalan ini, kita perlu meninjau prekondisi dan postkondisi dari persoalan.

Prekondisi : i dan t adalah sebuah bilangan bulat, $0 \le i \le n$ dan $0 \le t \le d$.

Postkondisi : sebuah jadwal yang dihasilkan adalah jadwal yang optimal dari perkerjaan {1,2,...,i} yang dikerjakan masing-masing dengan durasi t.

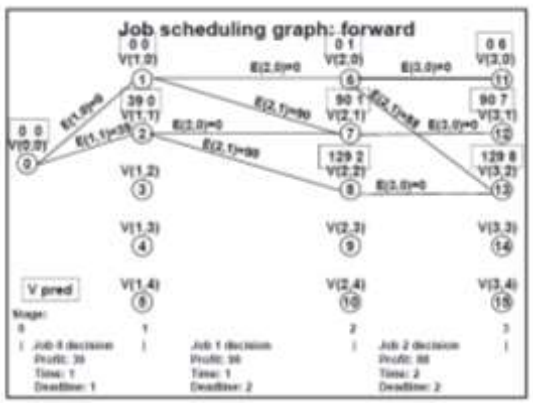
Berikut adalah pseudocode untuk prosedur yang digunakan (procedure OptSchedule).

```
procedure OptSchedule(i,t)
  if i = 0 then
    return 0
  end if
  if B[i,t] = B[i-1,t] then
    OptSchedule(i-1,t)
  end if
  else
    t'← min{t,di}-ti
    OptSchedule(i-1,t')
  end if
end
```

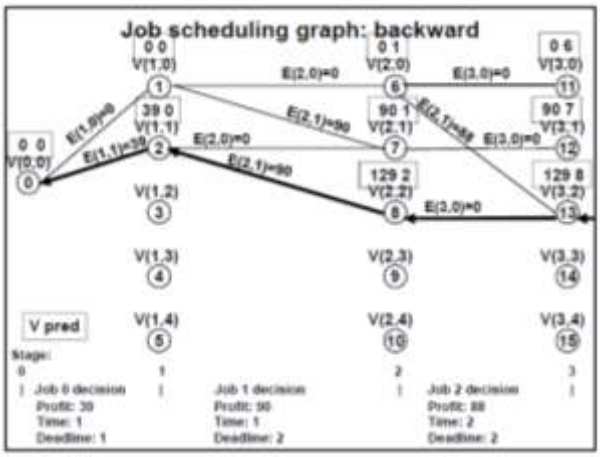
Penyelesaian persoalan penjadwalan ini sebenarnya merupakan bentuk penerapan dari persoalan penjadwalan yang mengandung beban/cost berbeda-beda. Berikut contoh kasus penjadwalan yang setiap pekerjaannya memiliki beban yang berbeda.

Job	Deadline	Profit	Time
0	1	39	1
1	2	90	1
2	2	88	2
3	2	20	1
4	3	37	3
5	3	25	2
6	4	70	1

Tabel 1. Contoh kasus *Weighted Interval Scheduling*



Gambar 3. Graf multipath : forward dari contoh kasus[1]



Gambar 4. Graf multipath : backward dari contoh kasus[1]

Berikut adalah source code untuk kasus ini :

```
//File : Scheduling.java

public class Scheduling implements
Comparable {
   int idJob;
   int deadline;
   int priority;
   int time;

   public Scheduling (int j, int d, int p,
int t) {
       idJob = j;
       deadline = d;
       priority = p;
       time = t;
   }

   @Override
   public int compareTo(Object other) {
       // TODO Auto-generated method stub
       int tmp = 0;
       Scheduling o = (Scheduling) other;
       if (deadline < o.deadline)
          tmp = -1;
       else if (deadline > o.deadline)
          tmp =1;
       return tmp;
   }

   public String toString() {
       return ("J: "+ idJob + " D : " +
deadline + " P : " + priority + " T : " +
time);
   }

}
```

```
//File : JobScheduler.java
```

```
import java.util.Arrays;

public class JobScheduler {
   private Scheduling[] jobs;
   private int nbrJobs;
   private int endTime;
   private int[] path;
   private int jobsDone;
   private int totalProfit;

   private int nodes;
   private int[] nodeProfit;
   private int[] nodeTime;
   private int[] pred;
   private int stageNodes;

   public JobScheduler ( Scheduling[] j,
int e) {
       jobs = j;
       endTime = e;
       nbrJobs = jobs.length;
       path = new int[nbrJobs+1];
       nodes = (nbrJobs-1)*(endTime+1)+2;
       nodeProfit = new int [nodes];
       nodeTime = new int[nodes];
       pred = new int[nodes];
       for (int i = 0; i< nodes; i++) {
           pred[i] = -1;
       }
       stageNodes = endTime+1;
   }

   public void jsd() {
       buildSource();
       buildCenter();
       buildSink();
       backPath();
   }

   private void buildSource() {
       nodeProfit[0] = 0;
       nodeTime[0] = 0;
       nodeProfit[1] = 0;
       nodeTime[1] = 0;
       pred[1] = 0;
       //if job feasible
       if (jobs[0].time <= jobs[0].deadline)
{
           int toNode = 1 + jobs[0].time;
           nodeProfit[toNode] =
jobs[0].priority;
           nodeTime[toNode] = jobs[0].time;
           pred[toNode] = 0;
       }
   }

   private void buildCenter() {
       for (int stage = 1; stage < nbrJobs-1;
stage++) {
           for (int node = (stage-
1)*stageNodes+1;
node<=stage*stageNodes;node++) {
               if (pred[node] >= 0) {
                   if (nodeProfit[node] >=
nodeProfit[node+stageNodes]) {

   nodeProfit[node+stageNodes] =
nodeProfit[node];
                       nodeTime[node+stageNodes]
```

```
= nodeTime[node];
                    pred[node+stageNodes] =
node;
                }

   if(nodeTime[node]+jobs[stage].time <=
jobs[stage].time) {
                    int nextnode =
node+stageNodes+jobs[stage].time;
                    if
(nodeProfit[node]+jobs[stage].priority>=nod
eProfit[nextnode]){
                        nodeProfit[nextnode] =
nodeProfit[node]+jobs[stage].priority;
                        nodeTime[nextnode] =
nodeTime[node]+jobs[stage].time;
                        pred[nextnode] = node;
                    }
                }
            }
        }
     }
   }

   private void buildSink() {
      int stage= nbrJobs -1;
      int sinkNode= (nbrJobs-1)*stageNodes +
1; for (int node=(stage-1)*stageNodes+1;
node <= stage*stageNodes; node++)
      if (pred[node] >= 0) {
          // Generate only single best
virtual arc from previous node
          // Job feasible
          if (nodeTime[node] +
jobs[stage].time <= jobs[stage].deadline) {
              // Job in solution
              if (nodeProfit[node]+
jobs[stage].priority >=
nodeProfit[sinkNode]) {
                 nodeProfit[sinkNode]=
nodeProfit[node]+ jobs[stage].priority;
                 nodeTime[sinkNode]=
nodeTime[node]+ jobs[stage].time;
                 pred[sinkNode]= node;
              }
          }
          // Job not in solution
          if (nodeProfit[node] >=
nodeProfit[sinkNode]) {
              nodeProfit[sinkNode]=
nodeProfit[node];
              nodeTime[sinkNode]=
nodeTime[node];
              pred[sinkNode]= node;
          }
      }
   }

   private void backPath() {
      // Trace back predecessor nodes from
sink to source
      path[nbrJobs]= (nbrJobs-1)*stageNodes
+ 1; // Sink node
      for (int stage= nbrJobs-1; stage >= 1;
stage--)
          path[stage]= pred[path[stage+1]];
   }

   public void outputJobs() {
      System.out.println("Jobs done:");
      for (int stage= 0; stage < nbrJobs;
stage++) {
          if (nodeProfit[path[stage]] !=
nodeProfit[path[stage+1]]) {

   System.out.println(jobs[stage]);
              jobsDone++;
              totalProfit +=
jobs[stage].priority;
          }
      }
      System.out.println("\nJobs done: " +
jobsDone +
      " total profit: "+ totalProfit);
   }

   public static void main(String[] args) {
      Scheduling[] jobs= new Scheduling[6];
      jobs[0]= new Scheduling(0, 1, 39, 1);
      jobs[1]= new Scheduling(1, 2, 90, 1);
      jobs[2]= new Scheduling(2, 2, 88, 2);
      jobs[3]= new Scheduling(3, 2, 20, 1);
      jobs[4]= new Scheduling(4, 3, 37, 3);
      jobs[5]= new Scheduling(5, 3, 25, 2);
      jobs[6]= new Scheduling(6, 4, 70, 1);
      int endTime= 1;
      Arrays.sort(jobs); // In deadline
order
      JobScheduler j= new JobScheduler(jobs,
endTime);
      j.jsd();
      j.outputJobs();
   }
}
```

[1]Contoh kasus beserta gambar graf diambil dari *http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec14.pdf*

IV. Analisis

Untuk membuktikan optimalitas penjadwalan dengan algoritma program dinamis, berikut akan dipaparkan sekilas perbandingannya dengan algoritma brute force. Dalam algoritma brute force, pertama-tama pekerjaan tetap diurutkan terlebih dahulu. Kemudian, penyelesaian langsung dilanjutkan dengan mencari semua kemungkinan lalu memilih yang paling optimal dari semua kemungkinan tersebut. Berikut adalah pseudocode untuk penyelesaian persoalan ini dengan menggunakan algoritma brute force.

```
Input n, s1,s2,...,sn, f1,f2,..,fn,
v1,v2,..vn
Urutkan pekerjaan berdasarkan waktu
selesai sehingga f1≤ f2 ≤ ... ≤ fn
Hitung nilai p(1), p(2), ... , p(n)

procedure OptBruteForce (j)
```

```
  if  (j = 0) then
    return 0
  else
    return max ( vj +
OptBruteForce(p(j)), OptBruteForce(j-
1))
  end if
end procedure
```

Dari hasil pembahasan di atas, dapat kita hitung nilai kompleksitas algoritmanya. Sorting awal dapat dilakukan dalam waktu O(n log n). Langkah pertama dan kedua tidak memiliki kompleksitas (karena hanya mendefinisikan status dan relasi rekurens). Langkah ketiga memerlukan waktu O(nd). Jadi, total waktu yang diperlukan adalah O(nd + n log n). Saat nilai d (*deadline*) besar, solusi persoalan ini akan didominasi oleh langkah ketiga sehingga dapat dinyatakan dengan O(nd).

Dalam algoritma brute force, penyelesaiannya dapat diimplementasikan dengan kompleksitas $O(n.2^n)$. Jika nilai d jauh lebih kecil dari 2^n, maka penyelesaian dengan menggunakan algoritma program dinamis akan lebih mangkus dibanding dengan algoritma brute force. Namun, jika nilai d lebih besar dari 2^n, penyelesaian persoalan penjadwalan ini lebih baik menggunakan algoritma brute force. Jika d memiliki nilai hampir sama dengan 2^n, maka diperlukan pengujian dengan menjalankan programnya lagi untuk membuktikan algoritma mana yang lebih mangkus.

V. KESIMPULAN DAN SARAN

Dari hasil pembahasan dan analisis didapat bahwa algoritma program dinamis dapat menyelesaikan permasalahan penjadwalan belajar secara optimal. Namun, untuk kompleksitasnya algoritma program dinamis belum tentu menghasilkan perhitungan yang mangkus. Hal ini dikarenakan nilai kompleksitas algoritmanya sangat dipengaruhi oleh durasi /waktu pengerjaan tiap pekerjaan. Algoritma program dinamis memiliki kompleksitas *O(nd + n log n)* yang menyebabkan algoritma ini akan berjalan secara mangkus dibanding algoritma brute force saat nilai durasinya kecil dan banyaknya pekerjaan yang dilakukan lebih banyak.

UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas berkat-Nya makalah ini dapat selesai dengan baik. Penulis juha mengucapkan terima kasih kepada bapak Rinaldi Munir dan ibu Nur Ulfa Maulidevi selaku dosen mata kuliah strategi algoritma serta semua pihak yang telah membantu sehingga penulis dapat menyelesaikan makalah ini.

REFERENSI

[1] Dynamic Programming: Job Scheduling, http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec14.pdf . Tanggal akses: 1 Mei 2015 pukul 13.00.

[2] Dynamic Programming Algorithms http://www.cs.mun.ca/~kol/courses/2711-f13/dynprog.pdf.Tanggal akses: 1 Mei 2015 pukul 13.00.

[3] Interval Scheduling: Greedy Algorithms and Dynamic Programming. http://www.cs.rit.edu/~zjb/courses/800/lec8.pdf. Tanggal akses: 1 Mei 2015. Pukul 12.30.

[4] CS787 Advanced Algorithms. http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec3.pdf. Tanggal akses: 1 Mei 2015. Pukul 13.10.

[5] Munir, Rinaldi. 2004.Diktat Kuliah IF 2251 Strategi Algoritmik. Program Studi Teknik Informatika, STEI,ITB.

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, 1 Mei 2015

ttd

Candy Olivia Mawalim
13512031