EE3980 Algorithms

hw08 Selecting Courses

106061146 陳兆廷

Introduction:

In this homework, I will be analyzing, implementing, and observing 1 algorithm. The goal of the algorithms is to select the most optimal courses and maximize the credits. The input of them will be a list of courses, and the output of them will be a weekly schedule that has the most credits.

During the analysis process, I will first introduce what matroid is and why can this be applied to greedy method. Then, I will be using counting method to calculate the time complexities of the algorithm. Furthermore, I will try to find the best-case, worst-case, and average-case conditions for the algorithm, respectively. Before implementing on C code, I will try to predict the result based on my analysis. Finally, I will calculate their space complexity for the total spaces used by the algorithm.

The implementation of the algorithm on C code will find the optimal solution for the provided data, course.dat.

Analysis:

1. **Matroid Theory:**

    a. **Independence System:**

Let S be a finite set and $I = \{X: X \subseteq S\}$, then the set system (S, I) is an independence system if (M1) $\emptyset \in I$ and (M2) if $Y \in I$ and $X \subseteq Y$ then $X \in I$.

Here, let S be a finite course list and let I be a list of selected courses. Let k be an integer, $k \leq |S|$, and $I = \{Y: Y \subseteq S \text{ and } |Y| \leq k\}$, then the set system (S, I) is an independence system. It is apparent that $\emptyset \in I$ and if Y is a subset of selected courses, $Y \in I$ then $|Y| \leq k$, then any $X \subseteq Y$ has $|X| \leq |Y| \leq k$ therefore $X \in I$.

We have proved that course list and weekly schedule are independence system.

b. Matroid

An Independence system (S, I) is a matroid if (M3) if $X, Y \in I$ and $|X| > |Y|$, then there is an $x \in X \setminus Y$ such that $Y \cup \{x\} \in I$.

Here, let S be a finite course list and let I be a list of selected courses. X, Y be 2 subsets of selected courses and $|X| > |Y|$. Then there is a course $x \in X \setminus Y$ that makes $|Y \cup \{x\}| = |Y| + 1 \leq |X| \leq k$ then $Y \cup \{x\} \in I$. A list of courses adds another course is still belong to the final weekly schedule with all selected courses.

Therefore, course list and weekly schedule are matroid and this task can be applied to greedy method.

1	EE202000
2	EE211000
3	EE214001
4	EE214002
5	EE221000
6	EE225502
7	EE231000

Y

1	EE202000
2	EE211000
3	EE214001

X

4	EE214002

x

6	EE225502
7	EE231000

Y

6	EE225502
7	EE231000
4	EE214002

$Y \cup \{x\}$

$|Y \cup \{x\}| = |Y| + 1 \leq |X| \leq k$, then $Y \cup \{x\} \in I$.

(S, I) is a matroid.

c. Weighted Matroid

A weighted matroid is matroid that is associated with a weight function w. $w(x)$ is the weight of an element x.

In this homework, I designed a weighted function $w(x)$ to calculated my course selecting decision. $w(x) = x.ratio * 5 + x.credit + x.time\_order$, where $x.ratio = x.credit / x.(\#\ of\ class\ a\ week)$ since the more credits participating a class would get, there are more credits a week. Furthermore, the more credit we can get when selecting a single course is better. Lastly, I want to fill up the weekly schedule from Monday to Friday.

2. Greedy Method:

We use greedy method to find a subset that is an optimal solution to the question.

<u>Algorithm:</u>

```
// Given n-element set A, find a subset that is an optimal solution.
// Input: A[1 : n], int n
// Output: solution  A.
Algorithm Greedy(A, n)
{
    solution := ∅ ;
    for i := 1 to n do {
        x := Select(A) ;
        A := A − {x} ;
        if Feasible(solution U x) then solution := solution U x ;
    }
    return solution ;
}
```

Where *Feasible()* is a function that checks if this added solution is feasible or not.

This way, we select through each single element in solution set and find the most optimal solution based on some criteria.

We modify this to solve the weighted matroid problem. *Best-in-Greedy().*

```
// Given (S, I) and w : S ! R find X 2 I such that w(X) is maximum.
// Input: (S, I) and w.
// Output: X
Algorithm Best-In-Greedy(S, I, w)
{
```

```
    Sort S into nonincreasing order by w ;
    X := ∅ ; // Initialize to empty set.
    for each x 2 S in order do { // Try all elements.
        if (X U {x} 2 I) then { // Maintain independence then add.
            X := X U {x} ;
        }
    }
    return X ;
}
```

This way, the solution with higher priority will goes into our solution set first, therefore we can get the most optimal solution.

3. Data Structure:

a. Courses:

A course has 5 properties:

properties	definition	example
Course.id	ID for course	EE202000
Course.credit	Credit that course worth	3
Course.student	Capacity of student	60
Course.time	Teaching time	T3T4R3R4
Course.name	Course's name	Partial Differential Equations and Complex Variables
Course.ratio	Credit per class	3/4 = 0.75
Course.time_priority	For measuring .time	T->4, R->2, 3->13-3=10 4/5+2/5+10/13+9/13 = 2.66

b. Schedule:

A 5 * 13 array that contains 1 or 0. 1 indicates that there’s a class.

4. GreedyCourse():

a. Abstract:

GreedyCourse() finds a weekly schedule with maximum credits. It follows *Best-in-Greedy()* with only a few changes.

First, I perform Insertion Sort, using my designed weight, on courses list S, and store the order. Then, I perform *Best-in-Greedy()* according to the order and update the weekly schedule if there’s no 2 class overlapping. Finally, print the optimal weekly schedule.

I choose Insertion Sort because, first, it is a stable sort. Second, the main point of this homework is Greedy Method, it is easier to modify and tuning the weight and find the optimal solution. Furthermore, the number of data is quite small, there is no need for other complicated sorting method that will even slow down the speed.

b. Algorithm:

```
// Given S, a list of courses and return a schedule with maximum credits X
// Input: S
// Output: X
Algorithm GreedyCourse(S)
{
    *order = InsertionSort();
    X := ∅ ; // Initialize to empty set.
    for each x U S in order do { // Try all elements.
```

```
        if (X U {x} is feasible) then { // Maintain independence then add.
            X := X U {x} ;
        }
    }
    return X ;
}
```

c. Time complexity:

	s/e	freq	total
`Algorithm GreedyCourse(S)`	0	0	0
`{`	0	0	0
`    *order = InsertionSort();`	N^2	1	N^2
`    X := ∅ ;`	1	1	1
`    for each x U S in order do {`	N	1	N
`        if (X U {x} is feasible) then {`	1	N	N
`            X := X U {x} ;`	1	N	N
`        }`	0	0	0
`    }`	0	0	0
`    return X ;`	1	1	1
`}`	0	0	0
	N^2 + 3N + 2		

For my GreedyCourse(), the time complexity would be O(N^2), where the major factor for it is the sorting process. If I change Insertion Sort to Merge Sort, the time complexity would be O(N lg N).

The main greedy process's time complexity would be O(N).

The way to check if `X U {x}` is feasible or not is simple. Simply check if the time slot in X is taken or not. It takes O(1).

d. Space Complexity:

The algorithm uses several integers and Course list S[N], Weekly schedule table[5][13], Selected course X[N], order list order[N]. **The space complexity would be O(N).**

5. **Time & Space:**

	GreedyCourse()
Time complexity	**O(N^2)**
Space complexity	**O(N)**

Implementation:

1. **My optimal solution:**

Results:

```
jack34672@Jack-ubuntu  ~/Documents/Algorithm_EE39800/hw08   master  1  ./a
Total credicts: 37
Number of courses selected: 12
1: MATH102006 4 120 T3T4R3R4 Calculus (II)
2: MATH202001 4 60 T1T2F1F2 Advanced Calculus II
3: EE214001 3 65 M3M4W2 Electromagnetism
4: EE336000 3 46 M7M8M9 Opto-electronic Devices
5: EE345000 3 30 T7T8T9 Computer Architecture
6: EE413500 3 46 T5T6F3 Principle of Lasers
7: EE335000 3 60 W3W4F4 Introduction to Solid-State Electronic Devices
8: EE366000 3 100 W5W6R8 Introduction to Digital Signal Processing
9: EECS340000 3 140 RaRbRc Satellite Electrical System Design
10: EE231000 3 90 M1M2R1R2 Introduction to Programming
11: EE364000 3 46 M5M6RnR5 Communication Systems (I)
12: EE240500 2 70 W7W8W9 Embedded System Laboratory
Weekly schedule:
    1 2 3 4 n 5 6 7 8 9 a b c
  M V V V V . V V V V V . . .
  T V V V V . V V V V V . . .
  W . V V V . V V V V V . . .
  R V V V V V V . . V . V V V
  F V V V V . . . . . . . . .
jack34672@Jack-ubuntu  ~/Documents/Algorithm_EE39800/hw08   master  1
```

I believe this is the most optimal solution. Furthermore, it is not a unique solution. I simply choose to select the courses that fit my schedule from Monday to Friday.

If I change the priority and fit my schedule from Friday to Monday, it will look like this:

```
 jack34672@Jack-ubuntu  ~/Documents/Algorithm_EE39800/hw08   master  1
dat
Total credicts: 37
Number of courses selected: 12
1: MATH202001 4 60 T1T2F1F2 Advanced Calculus II
2: MATH102006 4 120 T3T4R3R4 Calculus (II)
3: EECS340000 3 140 RaRbRc Satellite Electrical System Design
4: EE380000 3 35 R7R8R9 Power Processing
5: EECS302000 3 100 M7M8R6 Introduction to Computer Networks
6: EE335000 3 60 W3W4F4 Introduction to Solid-State Electronic Devices
7: EE413500 3 46 T5T6F3 Principle of Lasers
8: EE345000 3 30 T7T8T9 Computer Architecture
9: EE214001 3 65 M3M4W2 Electromagnetism
10: EE364000 3 46 M5M6RnR5 Communication Systems (I)
11: EE231000 3 90 M1M2R1R2 Introduction to Programming
12: EE240500 2 70 W7W8W9 Embedded System Laboratory
Weekly schedule:
    1 2 3 4 n 5 6 7 8 9 a b c
  M V V V V . V V V V . . . .
  T V V V V . V V V V V . . .
  W . V V V . . . V V V . . .
  R V V V V V V V V V V V V V
  F V V V V . . . . . . . . .
 jack34672@Jack-ubuntu  ~/Documents/Algorithm_EE39800/hw08   master  1
```

Therefore, the solution is not unique.

Observation:

1. Solution:

   The result meets the goal of the task and find the weekly schedule that has the most credits.

   The maximum credit we can get is 37 credits with 12 courses selected.

2. Not unique solution:

   There are multiple ways to select courses that has the same amount of credits.

Conclusions:

1. Course list and weekly schedule are matroid and this task can be applied to greedy method. (Analysis 1-a, 1-b)

2. Time and space complexities of *GreedyCourse()*:

	GreedyCourse()
Time complexity	**O(N^2)**
Space complexity	**O(N)**

3. The maximum credit we can get is 37 credits with 12 courses selected.

4. There are more than 1 solution to this problem.