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 $Y \cup \{x\} \in I$.

This problem should not be defined as a uniform matroid!

(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.

Algorithm:

```
1.  // Given n-element set A, find a subset that is an optimal solution.
2.  // Input: A[1 : n], int n
3.  // Output: solution  A.
4.  Algorithm Greedy(A, n)
5.  {
6.      solution := ∅ ;
7.      for i := 1 to n do {
8.          x := Select(A) ;
9.          A := A – {x} ;
10.         if Feasible(solution U x) then solution := solution U x ;
11.     }
12.     return solution ;
13. }
```

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().*

```
1.  // Given (S, I) and w : S ! R find X 2 I such that w(X) is maximum.
2.  // Input: (S, I) and w.
3.  // Output: X
4.  Algorithm Best-In-Greedy(S, I, w)
5.  {
```

```
6.      Sort S into nonincreasing order by w ;
7.      X := ∅ ; // Initialize to empty set.
8.      for each x 2 S in order do { // Try all elements.
9.          if (X U {x} 2 I) then { // Maintain independence then add.
10.             X := X U {x} ;
11.         }
12.     }
13.     return X ;
14. }
```

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:

```
1. // Given S, a list of courses and return a schedule with maximum credits X
2. // Input: S
3. // Output: X
4. Algorithm GreedyCourse(S)
5. {
6.     *order = InsertionSort();
7.     X := ∅ ; // Initialize to empty set.
8.     for each x U S in order do { // Try all elements.
```

```
9.          if (X U {x} is feasible) then { // Maintain independence then add.
10.             X := X U {x} ;
11.         }
12.     }
13.     return X ;
14. }
```

c. Time complexity:

	s/e	freq	total
1. Algorithm GreedyCourse(S)	0	0	0
2. {	0	0	0
3. *order = InsertionSort();	N^2	1	N^2
4. X := ∅ ;	1	1	1
5. for each x U S in order do {	N	1	N
6. if (X U {x} is feasible) then {	1	N	N
7. X := X U {x} ;	1	N	N
8. }	0	0	0
9. }	0	0	0
10. return X ;	1	1	1
11. }	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).**

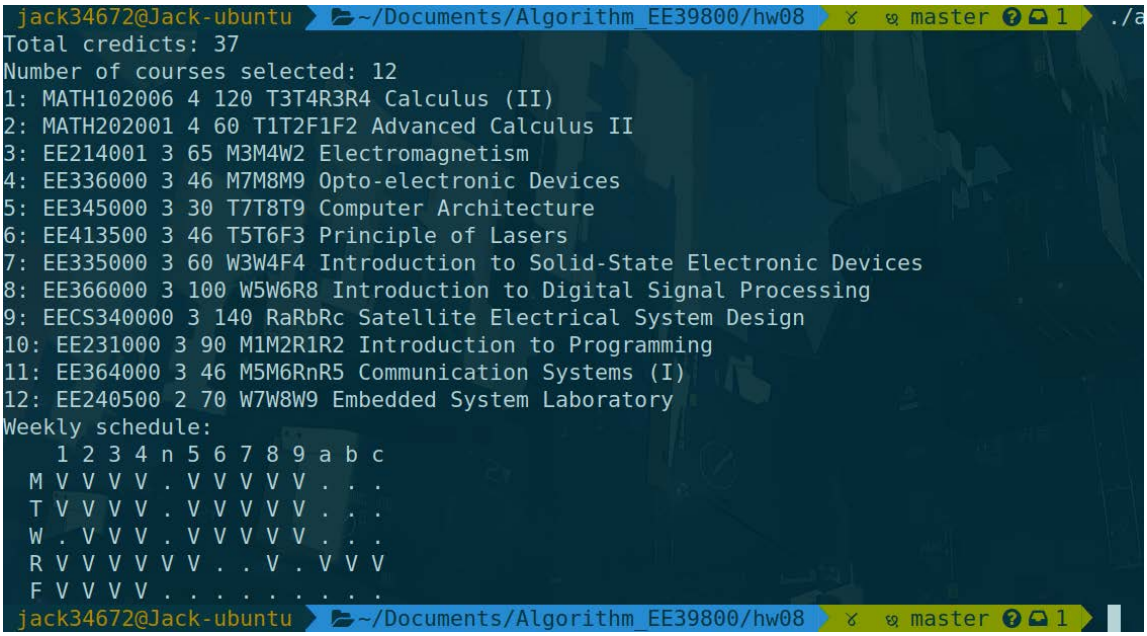
5. Time & Space:

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

Implementation:

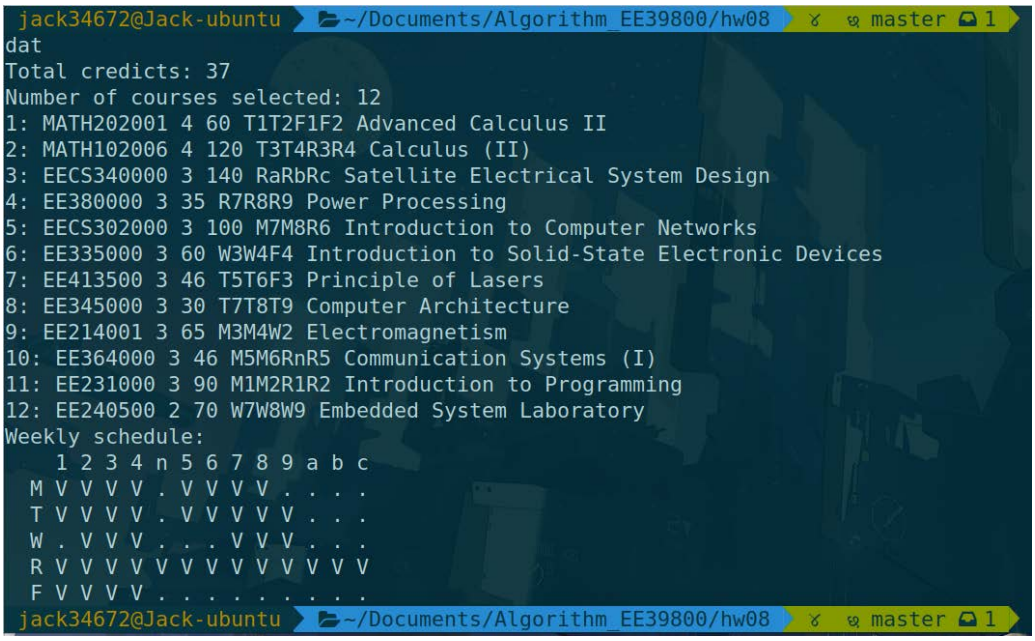
1. My optimal solution:

Results:



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:



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.

hw08.c

```
1  // EE3980 HW08 Selecting Course
2  // 106061146, Jhao-Ting, Chen
3  // 2020/05/09
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <sys/time.h>
9
10 typedef struct sCourse{                    // Data Structure for courses
11     int credict, student;
12     char *id, *time, *name;
13 } Course;
14
15 int N;                                     // # of courses
16 int *order;                                // sorting order
17 Course *Clist;                             // Course List
18 int table[5][13];                       // Weekly Schedule
19
20 void readInput(void);                      // read all inputs
21 void printInput(void);                     // print Input
22 void printTable(void);                     // print Weekly Schedule
23 double priority(Course a);                 // Course's weight
24 void insertionSort(int arr[]);             // Insertion Sort
25 void getCourseTime(char *d, char *n);      // convert M, T, W, R, F to int
26 int addCourse(Course c);                   // check if time slot is taken
27 void GreedyCourse(void);            // Greedy method to find optimal schedule
28
29 int main(void)
30 {
31     readInput();
32 //  printInput();
33     GreedyCourse();
34     return 0;
35 }
36
37 void readInput(void)                           // read all inputs
38 {
39     int i, j, Cred, Stu;                       // Initialize
40     char ID[15], Time[15], Name[100];
41
42     scanf("%d\n", &N);
43     Clist = (Course *)calloc(N, sizeof(Course));    // open space for Course
44     order = (int *)calloc(N, sizeof(int));
45     for (i = 0; i < N; i++) {
46         scanf("%s %d %d %s %[^\n]", ID, &Cred, &Stu, Time, Name);
47         Clist[i].credict = Cred;               // store each course element
48         Clist[i].student = Stu;
```

```
49          Clist[i].id = (char *)calloc(strlen(ID) + 1, sizeof(char));
50          Clist[i].time = (char *)calloc(strlen(Time) + 1, sizeof(char));
51          Clist[i].name = (char *)calloc(strlen(Name) + 1, sizeof(char));
52          for(j = 0; j < strlen(ID); j++) {
            for (j = 0; j < strlen(ID); j++) {
53              Clist[i].id[j] = ID[j];
54          }
55          Clist[i].id[j] = '\0';
56          for(j = 0; j < strlen(Time); j++) {
            for (j = 0; j < strlen(Time); j++) {
57              Clist[i].time[j] = Time[j];
58          }
59          Clist[i].time[j] = '\0';
60          for(j = 0; j < strlen(Name); j++) {
            for (j = 0; j < strlen(Name); j++) {
61              Clist[i].name[j] = Name[j];
62          }
63          Clist[i].name[j] = '\0';
64          order[i] = i;
65      }
66      for (i = 0; i < 5; i++) {
67          for (j = 0; j < 5; j++) {
68              table[i][j] = 0;
69          }
70      }
71  }
72
73  void printInput(void)                           // print adjacent list
74  {
75      int i;
76      for (i = 0; i < N; i++) {
77          printf("%d %s %d %d %s %s\n", i, Clist[i].id, Clist[i].credict,
78              Clist[i].student, Clist[i].time, Clist[i].name);
79      }
80  }
81
82  void printTable(void)                            // print weekly schedule
83  {
84      int i, j;
85      char day[5] = {'M', 'T', 'W', 'R', 'F'};
86      printf("Weekly schedule:\n");
87      printf("    1 2 3 4 n 5 6 7 8 9 a b c\n");
88      for (i = 0; i < 5; i++) {
89          printf("  %c", day[i]);
90          for (j = 0; j < 13; j++) {
91              if (table[i][j]) printf(" V");
92              else printf(" .");
93          }
94          printf("\n");
95      }
```

```
 96 }
 97
 98 void getCourseTime(char *d, char *n)        // convert course time to integers
 99 {
100     char day, num;
101     if (*d == 'M') day = '0';
102     if (*d == 'T') day = '1';
103     if (*d == 'W') day = '2';
104     if (*d == 'R') day = '3';
105     if (*d == 'F') day = '4';
106     if (*n == '1') num = '0';
107     if (*n == '2') num = '1';
108     if (*n == '3') num = '2';
109     if (*n == '4') num = '3';
110     if (*n == 'n') num = '4';
111     if (*n == '5') num = '5';
112     if (*n == '6') num = '6';
113     if (*n == '7') num = '7';
114     if (*n == '8') num = '8';
115     if (*n == '9') num = '9';
116     if (*n == 'a') num = ':';
117     if (*n == 'b') num = ';';
118     if (*n == 'c') num = '<';
119     *d = day;
120     *n = num;
121 }
122
123 double priority(Course a)   // generate priority for each course for sorting
124 {
125     int i, day, num;
126     char d, n;
127
128     for (i = 0; i < strlen(a.time) / 2; i++) {
129         d = a.time[i * 2];
130         n = a.time[i * 2 + 1];
131         getCourseTime(&d, &n);
132         day = 5 - d - '0';
133         num = 13 - n - '0';
134     }
135                             // course ratio * 5 + credict + time priority
136     return (double)a.credict / ((double)strlen(a.time) / 2) * 5
137         + (double)a.credict + (double) day / 5 + (double) num / 13;
138 }
139
140 void insertionSort(int arr[])   // simple insertion sort with priority
141 {
142     int i, j, pos;
143     Course a;
144     for (i = 1; i < N; i++) {
145         a = Clist[arr[i]];
```

```
146         pos = arr[i];
147         j = i - 1;                                          // apply priority
148         while ((j >= 0) && priority(Clist[arr[j]]) < priority(a)) {
149             arr[j + 1] = arr[j];
150             j = j - 1;
151         }
152         arr[j + 1] = pos;
153     }
154 }
155
156 int addCourse(Course c){        // add course and return if success or fail
    int addCourse(Course c)         // add course and return if success or fail
    {
157     int i, j, day, num, failed = 0;
158     int set[5][2];
159     char d, n;
160     for (i = 0; i < 5; i++) {                       // store changed time slot
161         for (j = 0; j < 2; j++) {
162             set[i][j] = -1;
163         }
164     }
165     for (i = 0; i < strlen(c.time) / 2; i++) {
166         d = c.time[i * 2];
167         n = c.time[i * 2 + 1];
168         getCourseTime(&d, &n);
169         day = d - '0';
170         num = n - '0';
171         if (table[day][num] == 0) {                 // if time slot not taken
172             table[day][num] = 1;                    // take it
173             set[i][0] = day;
174             set[i][1] = num;
175         } else {
176             failed = 1;
177         }
178     }
179     if (failed) {                                   // if there's overlap
180         for (i = 0; i < 5; i++) {
181             if (set[i][0] != -1) {          // remove slots taken this time
182                 table[set[i][0]][set[i][1]] = 0;
183             }
184         }
185         return 1;                                   // and return failed
186     } else {
187         return 0;                                   // or return succeed
188     }
189 }
190
191 void GreedyCourse(void)                             // fi  nd optimal schedule
192 {
193     int i, cred = 0, Cnum = 0, j;                   // Initialize
```

```
194     int Selected[30];
195
196     insertionSort(order);                           // sort course with weight
197
198     for (i = 0; i < N; i++) {                       // goes through courses
199         if (!addCourse(Clist[order[i]])) {          // if add success
200             Selected[Cnum] = i;                     // add course
201             cred = cred + Clist[order[i]].credict;  // calculate credit
202             Cnum++;                                 // add selected course num
203         }
204     }
205     printf("Total credicts: %d\n", cred);           // print result
206     printf("Number of courses selected: %d\n", Cnum);
207     for (i = 0; i < Cnum; i++) {
208         j = order[Selected[i]];
209         printf("%d: %s %d %d %s %s\n", i + 1, Clist[j].id, Clist[j].credict,
210             Clist[j].student, Clist[j].time, Clist[j].name);
211     }
212     printTable();
213 }
```

[Program Format] can be improved.
[Writing] hw08a.pdf spelling errors: processs(1)
[Coding] hw08.c spelling errors: fi(1)
[Course] selection problem should not be defined as a uniform matroid.
[Sorting] criteria is not described clearly.
[Good] to find 37-credit solution.

Score: 89