EE 3980 Algorithms

Homework 8. Selecting Courses

105061110 周柏宇

2020/5/10

1. Introduction

In this homework, we want to select courses using only greedy method to maximize the credits we get. Meanwhile, the selected courses should not overlap on the schedule. We will discuss the time and space complexity. Furthermore, we will try to examine the optimality and uniqueness of our solution.

2. Analysis & Implementation

In this homework, I try to establish the connection between course selection problem and maximization problem of independent systems. Hopefully, we can apply the general algorithm for the latter to solve our problem.

2.1 Independent systems, Matroid & Maximization

Let S be a finite set and $\mathcal{I} = \{X : X \subseteq S\}$, then the set system $(S, \mathcal{I})$ is an independent system if

(i) $\emptyset \subseteq \mathcal{I}$;

(ii) $If\ Y \in \mathcal{I}\ and\ X \subseteq Y\ then\ X \in \mathcal{I}.$

Furthermore, an independent system $(S, \mathcal{I})$ is a matroid if

(iii) $If\ X, Y \in \mathcal{I}\ and\ |X| > |Y|,$

$then\ there\ is\ an\ x \in X \backslash Y\ such\ that\ Y \cup \{x\} \in \mathcal{I}.$

Also, a maximization problem of independent systems is described as follows:

$Given\ an\ independent\ system\ (S, \mathcal{I})\ and\ the\ weight\ function$

$w: S \to \mathbb{R}^+, find\ an\ X \in \mathcal{I}\ such\ that\ w(X) = \sum_{x \in X} w(x)\ is\ maximized.$

Using the principle of the greedy method, that is,

"making the locally optimal choice with the intent of finding a global maximum", we can come up with the best-in greedy algorithm.

```
1. // Given (S,I) and w:S → R+
2. // find X ∈ I such that w(X) is maximum.
3. // Input: (S,I) and w
4. // Output: X
5. Algorithm BestInGreedy(S, I, w)
6. {
7.     Sort S into nonincreasing order by w;
8.     X := ∅; // initialize to empty set
9.     for each x ∈ S in order do { // try all elements
10.          // maintain independence then add
11.          if (X ∪ {x} ∈ I) then {
12.              X := X ∪ {x};
13.          }
14.      }
15.      return X;
16. }
```

However, the optimality of the solution is not guaranteed unless the

independent systems $(S, \mathcal{I})$ is also a matroid.

2.2 Selecting Courses

For this problem, we are given the information of courses, including class time, credits, etc. We try to select courses that does not occupy the same time on the schedule and maximize the total credits. For this homework, we are only allowed to use greedy method to obtain the solution.

First, if we let

$S = \{all\ courses\}$ and $\mathcal{I} = \{\text{all feasible selection of courses}\}$, by feasible selection we mean any course selected does not overlap, then $(S, \mathcal{I})$ is indeed an independent system, since

independence

(i) Not selecting any course is allowed.

(ii) If Y is a feasible selection, then $X \subseteq Y$ will still be feasible.

Furthermore, the course selection problem is a maximization problem of independent system if we let the weight function be the credits of a course (all positive number). Therefore, the high-level operation of best-in greedy algorithm for our problem will be as below.

```
1. // Given course information, e.g. class time and credits
2. // find feasible selection of courses
3. // such that sum of credits is maximum.
4. // Input: array of course information
5. // output: array selected
6. Algorithm BestInGreedy(courses, selected) {
```

```
7.     Sort courses into nonincreasing order by their credits;
8.     selected := ∅;
9.       // try all courses
10.     for each course ∈ courses in order do {
11.         // add the course if the set stays feasible
12.         if (selected ∪ {course} is feasible) then {
13.             selected := selected ∪ {course};
14.         }
15.     }
16.     return selected;
17. }
```

In the *BestInGreedy* algorithm, line 7 is a sorting operation, it will take up to $\mathcal{O}(n^2)$ if using insertion sort, where n is the number of courses available. The loop will execute $\Theta(n)$ times and we need to check the feasibility of the union for every iteration. Checking the feasibility will be simply looking up the class time we are about to occupied. Therefore, the exact number will be the number of classes of the course, which can be bounded by $6 = \mathcal{O}(1)$, assuming that no course has classes more than 6.

How?

To sum up, the sorting operation will not take less than $\mathcal{O}(n)$ and the whole loop are bounded by $\mathcal{O}(n)$. Thus, the total time complexity of *BestInGreedy* algorithm is determined by the time complexity of sorting operation. In my implementation, it is $\mathcal{O}(n^2)$.

Can be better.

As for the space complexity, assuming sorting does not take additional space, then we only need space for course information and the array *selected*.

Just to keep the necessary information: class time and credits, it will take $\Theta(n) \cdot \mathcal{O}(6+1) = \Theta(n)$ and $\mathcal{O}(n)$ for array *selected*. Therefore, the overall space complexity is $\mathcal{O}(n)$, where n is the number of courses available.

Unfortunately, our independent system is not a matroid. Because two feasible selection of courses X, Y and $|X| > |Y|$, say X has a course x that Y does not select, then $Y \cup \{x\}$ may not be feasible. This implies the algorithm does not guaranteed the optimality.

3. Result and Observation

The result we found using *BestInGreedy* algorithm is following.

```
Total credits: 37
Number of courses selected: 12
  1: MATH102006 4 T3T4R3R4 Calculus (II)
  2: MATH202001 4 T1T2F1F2 Advanced Calculus II
  3: EE211000 3 T7T8R7 Modern Physics
  4: EE214001 3 M3M4W2 Electromagnetism
  5: EE231000 3 M1M2R1R2 Introduction to Programming
  6: EE313000 3 W5W6R8R9 Optics and Photonics
  7: EE335000 3 W3W4F4 Introduction to Solid-State Electronic
Devices
  8: EE336000 3 M7M8M9 Opto-electronic Devices
  9: EE364000 3 M5M6RnR5 Communication Systems (I)
  10: EE413500 3 T5T6F3 Principle of Lasers
  11: EECS340000 3 RaRbRc Satellite Electrical System Design
  12: EE240500 2 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 . . . .
  W . V V V . V V V V V . . .
  R V V V V V V . V V V V V V
  F V V V V . . . . . . . . .
```

Although we cannot proof its optimality, it has been found that the solution achieving total credits 37 is not unique. Here's another solution.

Total credits: 37
Number of courses selected: 12
1: EE465000 2 W7W8W9 Communications System Laboratory
2: EE231000 3 M1M2R1R2 Introduction to Programming
3: EE364000 3 M5M6RnR5 Communication Systems (I)
4: EE413500 3 T5T6F3 Principle of Lasers
5: EECS202002 3 W5W6R8R9 Signals and Systems
6: EECS302000 3 M7M8R6 Introduction to Computer Networks
7: EECS303002 3 W3W4F4 Probability
8: EECS340000 3 RaRbRc Satellite Electrical System Design
9: ENE553000 3 T7T8T9 Terahertz Science and Technology
10: MATH242000 3 M3M4W2 Algebra II
11: MATH102007 4 T3T4R3R4 Calculus (II)
12: MATH202002 4 T1T2F1F2 Advanced Calculus II
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	V	V	.	.	.
R	V	V	V	V	V	V	V	.	V	V	V	V	V
F	V	V	V	V	.	.	.	.	.	.	.	.	.

hw08.c

```c
// EE3980 HW08 Selecting Courses
// 105061110, 周柏宇
// 2020/05/10

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// data structure to store course information
typedef struct course {
    char *crs_num; // course number
    int credits; // course credits
    int capacity; // class capacity
    int  n_class; // number of classes per week
    char *time_str; // raw class times
    int *time; // decoded class times
    char *name; // course name
} COURSE;

int N; // number of courses
int sum_c; // sum of credits
int n_selected; // number of courses selected
int *selected; // courses selected
int occupied[65]; // time occupied
char dayName[5] = { // abbreviation of the day of the week
    'M', 'T', 'W', 'R', 'F'};
COURSE *courses; // course information

void readData(void); // read input data
void BestInGreedy(void); // best-in greedy algorithm
void Sort(void); // sort courses wrt their credits in non-ascending order
void freeAll(void); // free allocated memory

int main(void)
{
    int i, j; // indices
    int cnt = 0; // label

    readData(); // read input data
    BestInGreedy(); // selecting courses using best-in greedy algorithm

    printf("Total credits: %d\n", sum_c);
    printf("Number of courses selected: %d\n", n_selected);
    for (i = 0; i < N; i++) { // print information of selected courses
        if (selected[i]) {
            printf("  %d: %s %d %s %s\n", ++cnt, courses[i].crs_num,
                courses[i].credits, courses[i].time_str, courses[i].name);
        }
```

```
49      }
50      printf("Weekly schedule:\n");
51      printf("    1 2 3 4 n 5 6 7 8 9 a b c\n");
52      for (i = 0; i < 5; i++) { // visualize the schedule
53          printf("  %c ", dayName[i]);
54          for (j = 0; j < 13; j++) {
55              if (occupied[13 * i + j] == 1) printf("V");
56              else printf(".");
57              if (j != 12) printf(" ");
58              else printf("\n");
59          }
60      }
61
62      freeAll(); // free allocated memory
63
64      return 0;
65  }
66
67  void Sort(void) // sort courses wrt their credits in non-ascending order
68  {
69      // implement insertion sort
70      int j, i; // indices
71      COURSE tmp; // temporary variable
72
73      for (j = 1; j < N; j++) {
74          tmp = courses[j];
75          i = j - 1;
76          // repeat until courses[i].credits is bigger
77          while ((i >= 0) && (tmp.credits > courses[i].credits)) {
78              courses[i + 1] = courses[i];
79              i--;
80          }
81          courses[i + 1] = tmp; // move courses[j] to correct place
82      }
83  }
84
85  void readData(void) // read input data
86  {
87      int i, j, k; // indices
88      int w, t; // day of week, time of day
89      char tmp[100]; // temporary variable
90      char ch; // temporary variable
91
92      scanf("%d\n", &N); // input number of courses
93      // allocate memory for course infomation
94      courses = (COURSE *)malloc(sizeof(COURSE) * N);
95      for (i = 0; i < N; i++) {
96          scanf("%s", tmp); // input course number
97          // allocate memory for course number
98          courses[i].crs_num = (char *)malloc(sizeof(char) * (strlen(tmp) + 1));
```

```
99          strcpy(courses[i].crs_num, tmp);
100
101         // input course credits and class capacity
102         scanf(" %d %d ", &courses[i].credits, &courses[i].capacity);
103
104         scanf("%s ", tmp); // input class times
105         // allocate memory for raw class times
106         courses[i].time_str = (char *)malloc(sizeof(char) * (strlen(tmp) + 1));
107         strcpy(courses[i].time_str, tmp);
108         courses[i].n_class = strlen(tmp) / 2; // get number of classes per week
109         // allocate memory for decoded class time
110         courses[i].time = (int *)malloc(sizeof(int) * courses[i].n_class);
111         for (j = 0; j < strlen(tmp); j += 2) { // decode the class time string
112             switch(tmp[j]) { // decide day of the week
                switch (tmp[j]) { // decide day of the week
113                 case 'M': w = 0; break;
114                 case 'T': w = 1; break;
115                 case 'W': w = 2; break;
116                 case 'R': w = 3; break;
117                 case 'F': w = 4; break;
118             }
119             switch(tmp[j + 1]) { // decide time of the day
                switch (tmp[j + 1]) { // decide time of the day
120                 case '1': t = 0; break;
121                 case '2': t = 1; break;
122                 case '3': t = 2; break;
123                 case '4': t = 3; break;
124                 case 'n': t = 4; break;
125                 case '5': t = 5; break;
126                 case '6': t = 6; break;
127                 case '7': t = 7; break;
128                 case '8': t = 8; break;
129                 case '9': t = 9; break;
130                 case 'a': t = 10; break;
131                 case 'b': t = 11; break;
132                 case 'c': t = 12; break;
133             }
134             courses[i].time[j / 2] = 13 * w + t; // express the time linearly
135         }
136
137         k = 0;
138         // store the rest of characters of the line
139         while ((ch=getchar()) != '\n') tmp[k++] = ch;
            while ((ch = getchar()) != '\n') tmp[k++] = ch;
140         tmp[k] = '\0';
141         // allocate memory for course name
142         courses[i].name = (char *)malloc(sizeof(char) * (strlen(tmp) + 1));
143         strcpy(courses[i].name, tmp);
144     }
145 }
```

```
146
147 void BestInGreedy(void) // best-in greedy algorithm
148 {
149     int i, j; // indices
150     int feas; // loop flag
151
152     // initialization
153     sum_c = 0;
154     n_selected = 0;
155     selected = (int *)calloc(N, sizeof(int));
156     for (i = 0; i < 65; i++) occupied[i] = 0;
157
158     Sort(); // sort courses wrt their credits in non-ascending order
159     for (i = 0; i < N; i++) { // try all courses
160         feas = 1;
161         for (j = 0; j < courses[i].n_class && feas; j++) {
162             // check if courses[i] overlaps with other selected courses
163             if (occupied[courses[i].time[j]] == 1) feas = 0;
164         }
165         if (feas) { // courses[i] is feasible
166             selected[i] = 1;
167             n_selected++;
168             // update the occupied time
169             for (j = 0; j < courses[i].n_class; j++) {
170                 occupied[courses[i].time[j]] = 1;
171             }
172             sum_c += courses[i].credits; // accumulate the credits
173         }
174     }
175 }
176
177 void freeAll(void) // free allocated memory
178 {
179     int i;
180
181     for (i = 0; i < N; i++) {
182         free(courses[i].crs_num);
183         free(courses[i].time_str);
184         free(courses[i].time);
185         free(courses[i].name);
186     }
187     free(courses);
188     free(selected);
189 }
```

[Program Format] can be improved.
[Coding] hw08.c spelling errors: infomation(1)
[Approach] of using Greedy method can be described more clearly (overlap checking?).
[Data structures] can be explained more clearly.
[Time] complexity can be improved.
[Good] to find 37-credit solution.

Score: 83