Software II
Final Report

By:
Le Kien Trung

g041172
Second-year student
Department of Mechano-Informatics, The University of Tokyo

Email: letrungkien211@gmail.com

April 25, 2012

Contents

Notes

- Solved problems: 1,2,3,4,5,6,7,8,9,10,11. Total number of stars: 21
- This report is created by LaTeX.
- All the source codes are successfully compiled with $gcc -ansi *.c -lm . (-lm is required when header math.h is included)
- No header's files are used in this report to avoid linking problems when compiling. Therefore, there are some iterated lines of code throughout the report.

Lessons' Comments

- This class focuses on algorithm, which makes it much more interesting than other programming classes. Teacher's slides are very good references, but they are not distributed. Hopefully, they will be available for downloading in the future.
- Submitting reports via email is truly annoying. I think using CFIVE is a more convenient way.
- About discussion board: Responses from TA are usually late.

References

1. Wikipedia and The Internet
2. Text Book
3. Data Structure And Algorithms In C, by Adam Drozdek

1 Problem 1*: Three prisoners' problem

Three prisoners, A, B and C, are in separate cells and sentenced to death. The governor has selected one of them at random to be pardoned. The warden knows which one is pardoned, but is not allowed to tell. Prisoner A begs the warden to let him know the identity of one of the others who is going to be executed. "If B is to be pardoned, give me C's name. If C is to be pardoned, give me B's name. And if I'm to be pardoned, flip a coin to decide whether to name B or C." The warden tells A that B is to be executed. Prisoner A is pleased because he believes that his probability of surviving has gone up from 1/3 to 1/2, as it is now between him and C.

- Theoretical solution: Let's denote A, B, C is the event when A, B, C is pardoned, respectively. I is the event when A is informed by warden that B is to be executed. We have:

$$P(A) = P(B) = P(C) = \frac{1}{3}$$

$$P(I) = P(I|A)P(A) + P(I|B)P(B) + P(I|C)P(C) = \frac{1}{2} \times \frac{1}{3} + \frac{1}{3} = \frac{1}{2}$$

$$P(A|I) = \frac{P(A \cap I)}{P(I)} = \frac{P(I|A) \times P(A)}{P(I)} = \frac{1}{6} \div \frac{1}{2} = \frac{1}{3}$$

- That means A is wrong when he thinks his chance of being pardoned has increased. The following program simulates the problem to calculate A's probability of surviving when he is informed by warden that B is to be executed.

Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
P(A is pardoned)=0.332569
A's probability of surviving doesn't change.
```

Comments: The tricky part of this problem is to calculate P(I). Because A misunderstands that $P(I) = \frac{1}{3}$, he concludes his probability of surviving has become $\frac{1}{6} \div \frac{1}{3} = \frac{1}{2}$. In fact, as shown above, $P(I) = \frac{1}{2}$, so $P(A|I) = P(A) = \frac{1}{3}$. On the other hand, $P(C|I) = \frac{P(I|C)P(C)}{P(I)} = \frac{1}{3} \div \frac{1}{2} = \frac{2}{3} > P(C) = \frac{1}{3}$. Therefore, C should be more optimistic when he knows the event I.

2 Problem 2*: Pythagorean triples with same circumference

A right triangle whose edges are integer is known as a Pythagorean triple. The problem is to find as many as possible groups of three or more Pythagorean triples whose circumferences are equal.

- This program uses the following formula to generate Pythagorean triple:

$$a = m^2 - n^2, b = 2mn, c = m^2 + n^2, \quad m, n \in N^*, m > n$$

$$a^2 + b^2 = (m^2 - n^2)^2 + (2mn)^2 = (m^2 + n^2)^2 = c^2$$

  If $m, n \in [1, R]$ then $\frac{R(R-1)}{2}$ Pythagorean triples are generated. In this program, $R = 200$ so $200 \times 199 \div 2 = 19900$ Pythagorean triples are generated.

- Their circumferences are calculated and then pushed orderly into a list. After finishing calculating process, we only need to check if a node in generated list has at least three Pythagorean triples or not.

- Furthermore, because if $(a_1, a_2, a_3), (b_1, b_2, b_3)$ are Pythagorean triples whose circumferences are equal, then $(ka_1, ka_2, ka_3), (kb_1, kb_2, kb_3), k \in N$ are also Pythagorean triples with the same circumference, from each satisfying group, theoretically, we can create an infinite number of those groups. Particularly, the program use k=2 to extend number of solutions found.

Executing Result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
There were 2333 groups found. Results are save in solution-2.txt
Maximum of Circumference: 1000000
Enter 'p' to print all the results.
Some representative groups (extract from solution-2.txt):
Circumference=3960, 3 triangles: (88,1935,1937) (360,1782,1818) (935,1368,1657)
Circumference=23760, 4 triangles: (432,11660,11668) (4158,9360,10242)
(5535,8272,9953) (6336,7560,9864)
Circumference=110880, 7 triangles: (1980,54432,54468) (4158,53280,53442)
(11655,48928,50297) (14080,47376,49424) (16016,46080,48784)
(22680,41184,47016) (32472,32480,45928)
```

Comments: In found groups, biggest number of members is 7. If the maximum circumference is increased, group of 8 or more Pythagorean triples satisfying problem's conditions can surely be found.

3 Problem 3*

Each integer can be written as sum of four integer' cubes. This program will demonstrate this proposition for every positive integer not greater than 1000. The algorithm is simple: Consider all possible groups of four integers in a given range, with each group, generate all possible sum of cubes with ± signs. The process ends when each integer in [0,1000] is found to be a sum of four cubes.

There is only one remark about this code, which is the usage of macro cube(x) rather than function pow(x,3) to calculate cube of x. execution's time increases significantly if function pow(x,3) is used. This fact demonstrates an advantage of using macro.
Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
.....
988= -7^3 +11^3
989= 1^3 -7^3 +11^3
990= -5^3 -6^3 +11^3
991= -1^3 -2^3 +10^3
992= -2^3 +10^3
993= 1^3 -2^3 +10^3
994= 1^3 +1^3 -2^3 +10^3
995= -1^3 +2^3 -7^3 +11^3
996= 2^3 -7^3 +11^3
997= -3^3 +8^3 +8^3
998= -1^3 -1^3 +10^3
999= -1^3 +10^3
1000= 10^3
SUCESS.
Execution's time: 4.130000(s)
```

Comments: 4 is the minimum number of cubes required. Not all integers can be written as sum of 2 or 3 cubes.

4 Problem 4*: Knapsack's problem

Given a set of items, each with a weight and a value, determine the count of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

This is one of combinatorial optimization's problems. Theoretically, the best solution can be found by considering all possible subset of items, calculating the total weight and price of each set, this method is called exhaustive enumeration. However, as the number of items increases, its computational complexity will be too much to handle. That is why other algorithms must be applied to solve this problem.

The following program using Simulated Annealing (SA) as the method. This method may not find the optimum solution, but usually, it will find a good enough solution. SA is almost the same as Hill Climbing, except the fact that in SA, there is some downward movement (change to the worse). This difference allows SA to avoid critical point, so in general, SA produce better result than Hill Climbing. Result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
Best solution found:
Weight sum= 45
Price sum= 373
Number of items= 12
Index: (W,P)
1 : #1  (2, 21)
2 : #4  (2, 21)
3 : #16 (5, 40)
4 : #18 (3, 40)
5 : #21 (2, 28)
6 : #26 (2, 21)
7 : #34 (5, 28)
8 : #35 (2, 28)
9 : #39 (7, 31)
10: #42 (5, 40)
11: #46 (3, 40)
12: #48 (7, 35)
Execution time: 0.020000(s)
```

Comments: Biggest total price would be 368 if Hill Climbing is used. We can see a slight advantage of SA over Hill Climbing through solving this problem, though not much. If the number of items as well as the biggest weight allowed increased, SA's advantage may emerge more clearly.

5 Problem 5**: Pythagorean triples with same area

This problem is almost the same as the Problem 2, except the concept of area in the former versus circumference in the later. Their source codes are identical by ignoring the differences between circumference and area. Result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
There were 492 groups found. Results are saved in solution-5.txt
Maximum of Area: 100000000
Enter 'p' to print all the results.

Execution's time: 2.500000(s)
```

Comments: The number of solutions of Problem 5 is much smaller then the number of solutions of Problem 2. That is understandable as the formula of area is more complicated than the formula of circumference.

6 Problem 6**

Solving strategy:

1. Divide 123456789 into four parts. Eg: 123, 456, 78, 9
2. Generate postfix expressions with each above set of four parts
3. Calculate those expressions and compare results with 100

By doing that, we have a simple and straight solution for this problem. On the other hand, the function calculating postfix expressions, namely, cal_postfix is slightly modified as following. Pattern of a postfix expression is, for instance, $11 + 1 - 1+$, number "1" in this expression is not directly used in calculating, it is just a symbol represents for a part in four parts of 123456789. So if 123456789 is divided to 4 parts 123,456,78,9, the expression in question is actually will be calculated as: $123, 456, +, 78, -, 9+$. Result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
100=(((123 - 45) - 67) + 89)
1 answers found.
```

Comments: Only one answer has been found. With 4 or 5 operators, there are definitely more answers.

7 Problem 7**: Travelling Salesman Problem (TSP)

Given a list of cities and their coordinates, the task is to find the shortest possible route that visits each city exactly once.

As Problem 2, this is also a combinatorial optimization's problem, and SA is applied, too. However, TSP is way more complicated than Knapsack's problem, so there are two remarkable differences between two programs.

- There is reheating process in Problem 7 but in Problem 2. Its presence helps to produce better result, as the initial state keep changing.
- Contrast to the fact that the cooling rate remains unchanged in Problem 2, it changes regularly in Problem 7. Through experimental executions, I found this regular changing produces much better result. Because cooling rate decides how fast an SA's circle ends, changing regularly seemingly broaden the search space.
- To prove that SA is better than Hill Climbing in solving TSP, the following program will deal with TSP by both methods.

Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
Hill Climbing method's result:
Length: 9132.646505
Route:
49->40->38->48->46->16->44->34->35->22->18->3->17->21->31->1->36
->39->37->24->5->15->6->4->25->12->28->27->26->47->29->30->2->7->42
->23->20->50->13->14->52->11->51->33->43->10->9->8->41->19->45->32->49
Execution's time: 3.340000(s)

Simulated Annealing method's result:
Length: 8662.798438
Route:
15->5->24->48->38->40->36->49->32->22->1->35->34->44->46->37->39
->45->18->31->23->20->50->16->29->30->2->7->42->21->17->3->19->41
->8->9->10->43->33->11->27->13->52->14->47->26->28->12->51->25->4->6->15
Execution's time: 3.490000(s)
```

Comments: The Hill Climbing hardly produce better result than SA. Execution's time of SA is slightly longer than Hill Climbing methods, as the test function is used.

8 Problem 8**: Indian Puzzle

Solving strategy:

1. Get all non-filled expressions from the puzzle.
2. Find all valid expressions by filling above expressions' empty grids.
   - Decide if a space should be filled by a number, an operator or an equality's symbol.
   - Divide each expressions into two or three parts confined by W or $=$ symbols. Calculate all of them and compare with each other.
3. Recursively put valid expressions into the puzzle. A solution is reached when no conflict is found during filling process.

Back-tracking algorithm is used to search through all possible ways of filling the puzzle with valid expressions. It involves using stack and recursion.
Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out q07.txt
Puzzle
9S9=S1W7SS
*WWW+WWWW*
SW8-S=5S/S
=WSW*WSWW-
S7S1S=S1W9
5W4W=W*WW=
WWS+S*S=1S
2W=W0W=WW9
=W4WWWSWW-
S*S=S1SS3S

#1 solution:
9*9=81W7=7
*WWW+WWWW*
5W8-2=54/9
=W8W*W+WW-
47-16=31W9
5W4W=W*WW=
WW5+2*5=15
2W=W0W=WW9
=W4WWW2WW-
2*3=210/35

#2 solution:
9*9=81W7=7
*WWW+WWWW*
5W8-2=54/9
=W6W*W/WW-
47-16=31W9
```

```
5W4W=W*WW=
WW3+2*6=15
2W=W0W=WW9
=W4WWW1WW-
2*3=210/35

#3 solution:
9*9=81W7=7
*WWW+WWWW*
5W8-2=54/9
=W6W*W*WW-
47-16=31W9
5W4W=W*WW=
WW3+2*6=15
2W=W0W=WW9
=W4WWW9WW-
2*3=210/35

n_ans=3
Execution's time:2.210000(s)
```

Comments: This program is able to solve all the puzzle from q01.txt to q14.txt files. Their number of solutions are shown in the following table:

Puzzle	Number of solutions
q01	4
q02	1
q03	14
q04	6
q05	840
q06	40
q07	3
q08	1
q09	1
q10	3
q11	14
q12	1
q13	9
q14	1

Table 1

I must admit this code is too complicated. Header files should have been used to reduce its complexity. Anyway, this is the most interesting problem I found in the report.

9 Problem 9**: Bin Packing Problem (BPP)

In computational complexity theory, the bin packing problem is a combinatorial NP-hard problem. In it, objects of different volumes must be packed into a finite number of bins of capacity V in a way that minimizes the number of bins used.

The program will demonstrate how BPP is solved by using first fit or fast fit, with random data or sorted data. This program is very simple, and there is nothing to talk about it.
Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
Result                   NextFit FastFit SortNext SortFast
Useless space average:   32.355  17.456  32.190   9.347
Useless space variance: 0.001   0.001   0.001    0.001
Execution time: 0.650000 (s)
```

Comments:

As the result turns out, fast fit's method is much more effective than next fit's method. Result of next fit's method does not change much with or without sorting process. On the contrary, fast fit's method with sorted data reduces the useless space to half compare with random data.

Sorting way does not affect much execution's time. Bubble sort and Quick sort are tested, and the difference of execution's time is minuscule.

10 Problem 10^{***}: Cyclotron Puzzle

SA's like method is used to solve this problem. Let's start from an arbitrary position, then move clockwise. Each move is decided by odds. In order to find longest route, probability of moving forward is much less than moving horizontally. For instance, current position is $(1, 7)$, then next possible moves are $(1, 8), (2, 7), (0, 7)$, with respective probability $\frac{1}{2*RANGE+1}, \frac{RANGE}{2*RANGE+1}, \frac{RANGE}{2*RANGE+1}$, in which RANGE is changeable. If RANGE=5, then those probability would be $\frac{1}{11}, \frac{5}{11}, \frac{5}{11}$. To avoid critical state, RANGE is changed regularly during searching process, which is similar to using cooling-rate in SA's method.

Two puzzles are solved in almost identical way. Some changes are made to deal with the presence of two line across the puzzle. Result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
 2  3 11 84 10 92 63 72 19 91 98 68 51 16 46 77 14 12 46 63
23 51 26 34 73 94 27 49 73 98 60 44 36 31 79 73 67 72 56 74
11 71 40 25 22 31 83 31 20 96 23 96 74  3  6 13 97 87 25 33
87 92 73                                          79 50  3
45 57 61                                          33 55 81
23 48 43                                          85 50 28
73 42 29                                          39 97 92
56 31 61                                          17 23 19
88 40 52                                          13 32 71
54 79 11                                          51 56 49
 9 60 43                                          11 99 47
99 13 20                                          34 12 32
12 48 26 67 37 34 49 56 99 32 39 94 11 23  9 29 45 56 62 65
90 70 70 15 25  6 44 77  8 66 14 54 93  3 78 95 99 99 18 69
13 20 62 53 61  6 82 55 43 79 98 37 46 26 97 66 43 49 25 64
Length=56
      11 84       63 72 19 91    68 51 16 46 77 14
      26 34    94 27       98 60 44             67
      40 25 22 31          96 23                97 87
   92 73                                        79
                                                33 55 81
                                                85 50 28
                                                39
                                                17
                                                13 32 71
                                                   49
                                                99 47
                                                12
                                          9 29 45 56 62
                                54 93  3 78
                                37
```

```
48 29 44 18 90 91 69 25 62 98 11  4 39 33 64 81 46 95 62 38
68 24 96 18 94 87 14 17 32 69 89 69 27 25 88 67 11 21 32  6
44 42 69 11 35 74 59 43 29 45 99 13 53 42 63 30 81 80 13  7
38 31 43                     1                        62 42 28
77 67 86                     2                        90 33 78
 2 38  0                     3                        57  9 90
 7 88 72                     4                        20 81 34
92 24 47 10 11 12 13 14 15  5 16 17 18 19 20 21 22 38 82  5
90 11 53                     6                        89 85 94
13 45 75                     7                        49 22  0
61 25 34                     8                        68 35 93
42 35 68                     9                        24 64 89
54 37 46 83 30 33  4 83 87 18 96 53 45 18 50 64 61 98 18 44
68 89 35 74 36 60 36 95 41 31 92  4 22 80 16 11 20 43 19 28
66  3 46 50 29 52 81 32 76 45 57 89 56 20 47 34 78 94 82 88
Length=58
      44 18 90       25 62 98 11  4 39
      96    94    14 17             27
      69    35 74 59
38 31 43
77 67 86
       0
 7 88 72
92 24 47
      53
13 45 75
61
42
54 37       30 33    83 87
68 89       36 60    95 41
66  3 46 50 29 52 81 32 76
```

Comments: This is very interesting problem. If Hill Climbing method is used, in other words, always go horizontally if possible, the result should get worse.

11 Problem 11****: Almost Integer

An almost integer is a number that is very close to an integer. This program uses three following famous mathematical constants to generate some interesting almost integers.

- Ratio of any Euclidean circle's circumference to its diameter: $\pi = 3.14159265$
- Base of natural logarithm: $e = 2.71828183$
- Golden ratio: $\phi = \frac{1+\sqrt{5}}{2}$

Let's consider an expression with four parts and three operators. For instant, $1+2+3+4, 1-2+3/4$. In a specific range (0,10), let's assign:

$$part[1] = \pi^i, part[2] = e^j, part[3] = \phi^k, part[4] = l, \qquad i, j, k, l \in (0, 10)$$

Similar to 6th problem, with 4 parts and three operators, we can generate five different postfix's expressions, calculate each of them, then check whether the result is an almost integer or not. To avoid superficial results, such as $\frac{\pi}{e^9 * \phi^9 * 9}$, all results being less than 100 are ignored.
Executing result:

```
letrungkien7@letrungkien7:~/Report$ ./a.out
561.000000=(PI^2 * (e^4 + (r^6 / 8)))
58827.000005=(PI^2 + (e^9 * (r^7 / 4)))
290.000006=(PI^5 - ((e^5 - r^3) / 9))
945.000009=(PI^6 - (e^2 * (r^5 / 5)))
```

$$561.000000 = \pi^2 * (e^4 + \frac{\phi^6}{8}), \qquad 58827.000005 = \pi^2 + \frac{e^9 * \phi^7}{4}$$

$$290.000006 = \pi^5 - \frac{e^5 - \phi^3}{9}, \qquad 945.000009 = \pi^6 - \frac{e^2 * \phi^5}{5}$$

Comments: This problem is a bit ambiguous. Deciding an expression is interesting or not is very subjective. At first, I could not come up with any idea at all. I prefer a more specific purpose to create code.