Programming Project

Maximum Subarray Sum

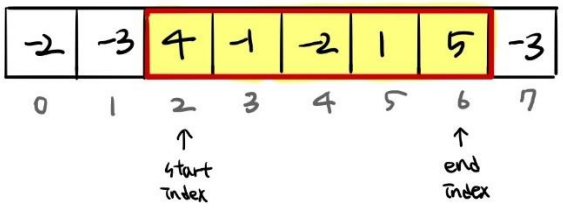
과목	: 컴퓨터알고리즘
담당교수	: 이상호
학과	: 컴퓨터공학과
학번	: 1829008
이름	: 김민영
이메일	: kmy8759@ewhain.net
제출일자	: 2020년 5월 20일

[Programming Project] : 연속구간 최대합(Maximum contigious subarray) 문제의 알고리즘 구현

1) 문제설명

다양한 길이(n)의 랜덤하게 생성된 정수에 대해 연속구간 최대합 문제를 해결하는 프로그램을 작성한다. 이때, 세가지 방법 1) 중첩 반복문 2) 분할정복방법 3) 동적계획법 을 이용하여 문제를 풀 것이다. 또한, 결과분석 및 토의에서 각 방법의 실행시간에 대해 분석할 것이다.

*연속구간 최대합 문제란? Input data X[1..n]가 주어졌을 때, V(i,j)=sum(X[i,j])를 정의하고 이중 모든 i<=j에 대해 V(r,s)>=V(i,j)를 만족하는 연속구간 최대합(maximum sum of contiguous subarray) V(r,s) (r<=s)를 아래와 같이 찾는 문제이다.



-> maximum sum : 4+(-1)+(-2)+1+5=7

2) 입출력의 예

(1) 입력 자료 형식의 예 : 파일형식(file1.txt~file10.txt)

* file1.txt // 음수로 직접 생성

```
10 // data의 개수
-2 -1 -3 -5 -6 -7 -8 -90 -10 -11 //data
```

* file2.txt

```
100
-60 -20 -89 -97 -61 37 -32 41 64 -73 -8 28 94 15 42 -64 68 -58 -13 53 56 -88 16 -57 -56
39 35 6 81 -47 30 24 -61 18 36 59 -22 -47 -65 38 45 -80 81 47 37 -52 -79 16 -13 46 -23 -
66 93 -15 20 -68 -49 -44 10 -60 -52 -22 73 -3 -79 -27 -4 72 -15 -36 11 -84 31 -86 -54 -45
0 84 32 67 63 -92 13 -9 -18 93 -44 -50 -35 75 77 89 32 -42 40 16 -21 39 76 -58
```

(2) 출력 자료 형식의 예

```
<file1.txt>
Solution1) : iteravie
최대합 : -1 인덱스 : 1, 1
```

실행시간 : 0.0000ms

Solution2) DivideConquer
최대합 : -1 인덱스 : 1,1
실행시간 :0.0000ms

Solution3) : dynamic
최대합 : -1 인덱스 : 1,1
실행시간 :0.0000ms

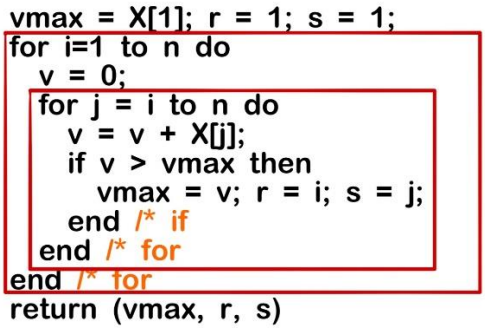
<file2.txt>
Solution1) : iteravie
최대합 : 485 인덱스 : 76,98
실행시간 :0.0000ms

Solution2) DivideConquer
최대합 : 485 인덱스 : 76,98
실행시간 :0.0000ms

Solution3) : dynamic
최대합 : 485 인덱스 : 76,98
실행시간 :0.0000ms

3) 문제풀이방법(알고리즘)

방법1) 중첩 반복문 이용 : 아래와 같은 방식으로, 반복문 두개를 중첩하여 maximum subarray를 찾는다. (아래 사진은 양수인 경우만 고려하지만, 과제에서 수생한 코드는 음수인 경우까지 고려하기 때문에 세부사항은 다름)



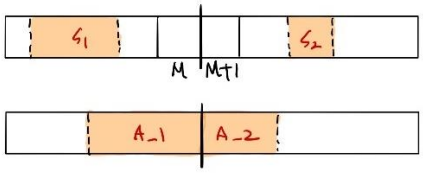
```
vmax = X[1]; r = 1; s = 1;
for i=1 to n do
  v = 0;
  for j = i to n do
    v = v + X[j];
    if v > vmax then
      vmax = v; r = i; s = j;
    end /* if
  end /* for
end /* for
return (vmax, r, s)
```

방법2) 분할정복 알고리즘 이용 : 최대 부분합은 오른쪽 아래 사진과 같이 총 세 가지 중 하나에 속할 수가 있다. 1. 구간 [low~M] 2. 구간 [M+1~high] 3. 양쪽 절반에 걸쳐있는 경우 이 세가지 안에 정답이 반드시 있다. 따라서 전체 array를 분할정복 방식으로 반으로 나눠가고 3가지 경우의 수 중에서 만들어지는 max sum 중에서 최대값을 선택하면 우리가 원하는 정답을 찾을 수 있다.

```
int maxSum (l, u) {
  if l = u then return max(0.0, x[l])
  else m = (l+u)/2;
    sum = 0.0; maxToLeft = 0.0;
    for i = m downto l {
      sum = sum + x[i];
      maxToLeft = max(maxToLeft, sum);
    }
    sum = 0.0; maxToRight = 0.0;
    for i = m+1 to u do {
      sum = sum + x[i];
      maxToRight = max(maxToRight, sum);
    }
    maxCrossing = maxToLeft + maxToRight;
  maxInLeft = maxSum(l, m);
  maxInRight = maxSum(m+1, u);
  return max(maxCrossing, maxInLeft, MaxInRight);
}
```

left

right



방법3) 동적계획법 이용

```
int MaxSubseqSum(X, 1, n) {
   MaxSum = 0;
   ThisSum = 0;
   for k = 1 to n {
      ThisSum = max{ThisSum + X[k], 0};
      MaxSum = max{MaxSum, ThisSum};
   }
return MaxSum;
```

식 S_k= max(S_k+ a_k, 0) 을 이용하여 ThisSum을 구하고 MaxSum을 하나씩 업데이트 시켜간다.

* S_k: 연속구간 최대합, a_k : 제일 오른쪽 element

각 함수별 설명

file_write 함수 : random 한 정수 data 를 파일의 길이만큼 생성시켜서 이를 텍스트파일에 쓰고 저장한다.

file_generator 함수 : file2.txt~file10.txt 를 file_write 를 하기 위해 생성하고 파일을 연다.

file_read 함수 : file_num(1~10)을 받아와 file1.txt~file10.txt 로부터 1. n(data 의 개수) 2. data 배열을 읽어들이는 함수이다.

solution1_iterative 함수 : 중첩반복문 방법으로 연속구간 최대합을 구하는 함수이다.

solution2_DivideConquer 함수 : 분할정복 방법으로 연속구간 최대합을 구하는 함수이다.

solution3_dynamic 함수 : 동적계획법 방법으로 연속구간 최대합을 구하는 함수이다.

main 함수 : 각 file.txt 에 대해 file 을 읽어오고, 각 solution 방법에 대해 연속구간 최대합을 구하는 함수를 호출한다. 이때에, 시간을 재주는 clock()함수를 이용하여 각 solution 별 실행시간을 구해 정답과 함께 출력한다.

4) 소스코드(소스파일.cpp)

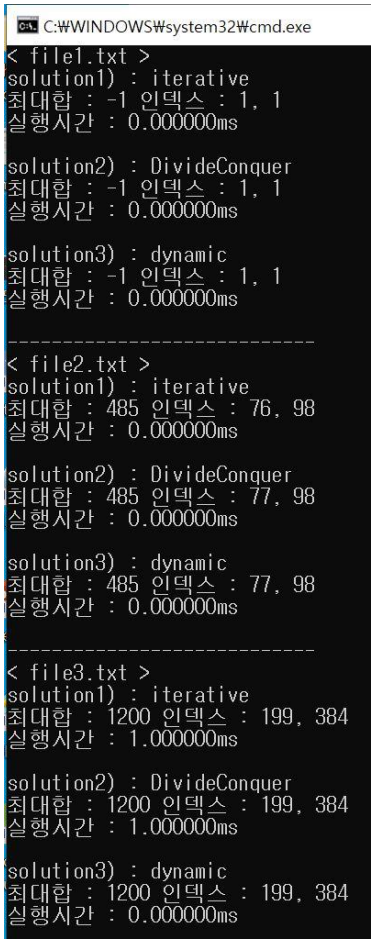
```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX(a, b) (((a) > (b)) ? (a) : (b))
#define MIN -9999

int data[1000001];
int n; //file을 읽어올 때 저장할 data 개수
int file_len[10] = { 10, 100, 500,1000,5000,10000,50000,100000,500000,1000000 }; // file을 생성할 때 사용할 data 개수

typedef struct Answer {
	int sum;
	int start;
	int end;
} Answer;

void file_write(FILE *file, int file_len);
void file_generator();
void file_read(int file_num);
Answer maxSum(int x[], int l, int u); // solution2(분할정복 방법)에서 사용되는 함수
Answer solution1_iterative(int data[], int n);
Answer solution2_DivideConquer(int data[], int n);
Answer solution3_dynamic(int data[], int n);

int main(void) {
	time_t start_time = 0;
	double time;
	//file2.txt ~ file10.txt를 만들어주는 함수실행
	//file_generator();

	//file1.txt~file10.txt에 대해 프로그램 실행
	for (int i = 1; i <= 10; i++) {
		file_read(file_num);

		printf("solution1) : iterative\n");
		start_time = clock();
		Answer answer1 = solution1_iterative(data, n);
		printf("최대합 : %d 인덱스 : %d, %d\n", answer1.sum, answer1.start, answer1.end);
		time = (double)(clock() - start_time);
		printf("실행시간 : %fms\n\n", time);

		printf("solution2) : DivideConquer \n");
		start_time = clock();
		Answer answer2 = solution2_DivideConquer(data, n);
		time = (double)(clock() - start_time);
		printf("최대합 : %d 인덱스 : %d, %d\n", answer2.sum, answer2.start, answer2.end);
		printf("실행시간 : %fms\n\n", time);

		printf("solution3) : dynamic \n");
		start_time = clock();
		Answer answer3 = solution3_dynamic(data, n);
		time = (double)(clock() - start_time);
		printf("최대합 : %d 인덱스 : %d, %d\n", answer3.sum, answer3.start, answer3.end);
		printf("실행시간 : %fms\n\n", time);
		printf("----------------------------\n");
	}
```

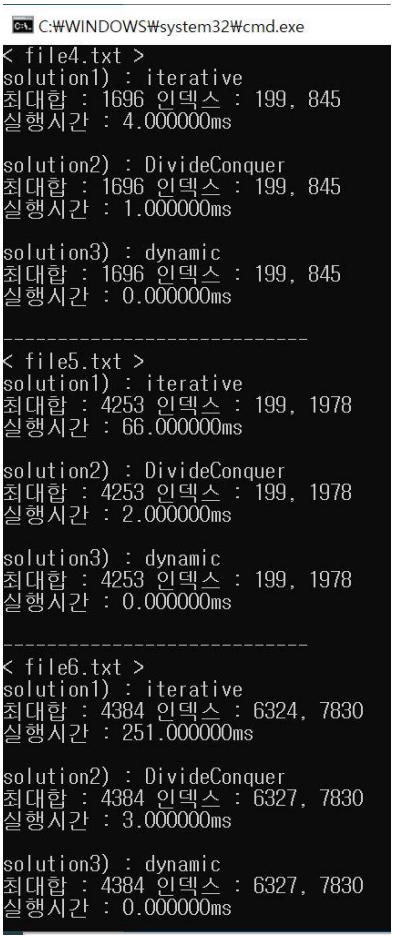
```
	return 0;
}
void file_write(FILE *file, int file_len) {
	srand(time(NULL));

	int random = 0;
	int sign = 0;
	int data;

	fprintf(file, "%d\n", file_len);
	for (int j = 0; j < file_len; j++) {
		random = rand() % 100; // random data 생성
		sign = rand() % 2; // 양수, 음수를 결정하는 random number sign
		// 양수
		if (sign == 0) {
			data = random;
		}
		// 음수
		else if (sign == 1) {
			data = random * (-1);
		}
		fprintf(file, "%d ", data);
	}
}

//file2.txt ~ file10.txt 파일 생성
void file_generator() {
	FILE *file1, *file2, *file3, *file4, *file5, *file6, *file7, *file8, *file9, *file10 ;
	//fopen_s(&file1, "file1.txt", "w+"); file_write(file1, file_len[0]); -> file1 은 음수로
구성된 data 로 직접 만든다.
	fopen_s(&file2, "C:/Temp/file2.txt", "w+"); file_write(file2, file_len[1]);
	fopen_s(&file3, "C:/Temp/file3.txt", "w+"); file_write(file3, file_len[2]);
	fopen_s(&file4, "C:/Temp/file4.txt", "w+"); file_write(file4, file_len[3]);
	fopen_s(&file5, "C:/Temp/file5.txt", "w+"); file_write(file5, file_len[4]);
	fopen_s(&file6, "C:/Temp/file6.txt", "w+"); file_write(file6, file_len[5]);
	fopen_s(&file7, "C:/Temp/file7.txt", "w+"); file_write(file7, file_len[6]);
	fopen_s(&file8, "C:/Temp/file8.txt", "w+"); file_write(file8, file_len[7]);
	fopen_s(&file9, "C:/Temp/file9.txt", "w+"); file_write(file9, file_len[8]);
	fopen_s(&file10, "C:/Temp/file10.txt", "w+"); file_write(file10, file_len[9]);
}

void file_read(int file_num) {
	FILE *fp;
	int m;
	char fileName[100];
	sprintf(fileName, "C:/Temp/file%d.txt", file_num);
	fp = fopen(fileName, "r"); // txt 파일 읽어오기
	if (fp == NULL) { // 파일을 읽어오지 못하는 경우에 대한 예외처리
		printf("could not read file");
		return;
	}
	printf("< file%d.txt >\n", file_num);
	fscanf(fp, "%d", &n);
	for (int i = 0; i < n; i++) {
		fscanf(fp, "%d", &data[i]);
	}
}

Answer solution1_iterative(int data[], int n) {
	Answer answer;
	int max = data[0];
	int r = 0;
	int s = 0;
```

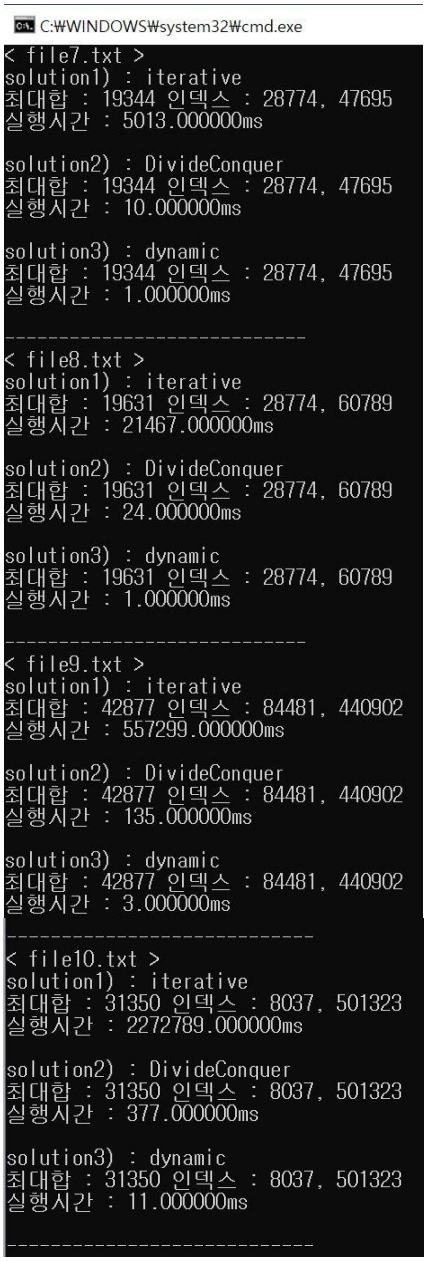
```
	for (int i = 0; i < n; i++) {//시작 설정
		int v = 0; // sum 초기화
		for (int j = i; j < n; j++) {
			v = v + data[j];
			if (v > max) { //현재까지의 최대값 보다 크면 최댓값 업데이트
				max = v;
				r = i;
				s = j;
			}
		}
	}
	answer.sum = max;
	answer.start = r;
	answer.end = s;
	return answer;
}
Answer maxSum(int data[], int l, int u) {
	int m, i, sum;
	Answer left, right, crossing, crossing_left, crossing_right, temp;
	left.sum = MIN;
	right.sum = MIN;
	if (l == u) {
		if (data[l] >= MIN) {
			temp.sum = data[l];
			temp.start = l;
			temp.end = l;
			return temp;
		}
		else {
			temp.sum = MIN;
			temp.start = -1;
			temp.end = -1;
			return temp;
		}
	}
	else
	{
		m = (l + u) / 2;
		//왼쪽 <- mid
		sum = 0;
		crossing_left.sum = MIN;
		for (i = m; i >= l; i--) {
			sum += data[i];
			// max( maxToLeft, sum)
			if (sum > left.sum) {
				crossing_left.sum = sum;
				crossing_left.start = i;
				crossing_left.end = m;
			}
		}
		// mid -> 오른쪽
		sum = 0;
		crossing_right.sum = MIN;
		for (i = m + 1; i <= u; i++) {
			sum += data[i];
			// max( maxToRight, sum)
			if (sum > crossing_right.sum) {
				crossing_right.sum = sum;
				crossing_right.start = m;
				crossing_right.end = i;
			}
		}
		crossing.sum = crossing_left.sum + crossing_right.sum;
		crossing.start = crossing_left.start;
```

```
                crossing.end = crossing_right.end;
        }
        left = maxSum(data, l, m);
        right = maxSum(data, m + 1, u);
        // MAX(crossing, left, right)
        // crossing이 제일 큰 경우
        if ((crossing.sum >= left.sum) && (crossing.sum >= right.sum)) {
                return crossing;
        }
        //left가 제일 큰 경우
        else if (left.sum >= right.sum) {
                return left;
        }
        //right가 제일 큰 경우
        else {
                return right;
        }
}

Answer solution2_DivideConquer(int data[], int n) {
        Answer answer = maxSum(data, 0, n - 1);
        return answer;
}

Answer solution3_dynamic(int data[], int n) {
        Answer Max, This;
        int max = data[0];
        int max_index = 0;
        Max.sum = 0;
        Max.start = 0;
        Max.end = 0;
        This.sum = 0;
        This.start = 0;
        This.end = 0;
        for (int i = 0; i < n; i++) {
                if (data[i] > max) {
                        max = data[i];
                        max_index = i;
                }
                // This.sum + data[i]가 양수인 경우 This.sum에 data[i]를 더해준다.
                if (This.sum + data[i] > 0) {
                        This.sum += data[i];
                }
                // 음수인 경우
                else {
                        This.sum = 0;
                        This.start = i + 1;
                }
                // Max.sum = Max(This.sum, Max.sum)으로 업데이트
                if (This.sum >= Max.sum) {
                        Max.sum = This.sum;
                        Max.start = This.start;
                        Max.end = i;
                }
        }
        // 최대값이 마이너스인 경우
        if (Max.sum <= 0) {
                Max.sum = max;
                Max.start = max_index;
                Max.end = max_index;
        }
        return Max;
}
```

5) 수행결과







위와 같이 프로그램으로 구한 data개수(n) 별 실행시간을 표로 정리해보면 다음과 같다.

n <int>	solution1 <int>	solution2 <int>	solution3 <int>
10	0	0	0
100	0	0	0
500	1	1	0
1000	4	1	0
5000	66	2	0
10000	251	3	0
100000	5013	10	1
500000	21467	24	1
1000000	557299	135	3
5000000	2272789	377	11

* solution : 실행시간 (단위 : ms)

* 실행시간 그래프->다음페이지

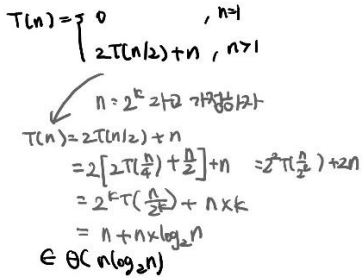
6) 결과분석 및 토의

1. Time complexity

Solution1) θ(n^{2})

반복문 두개를 사용하므로 시간복잡도는 θ(n^{2}) 이다.

Solution2) θ(nlogn)

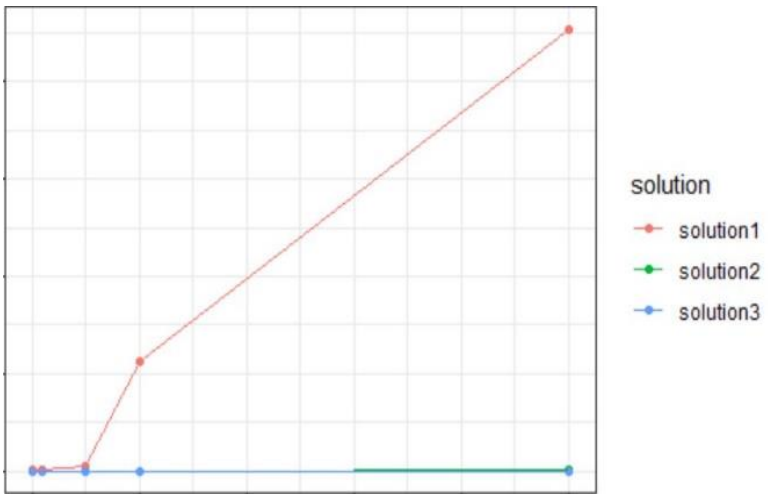


Solution3) θ(n)

길이 n의 배열을 채워가기 때문에 시간복잡도는 θ(n)이다.

2. 수행시간 분석

실행시간(ms)



n(data개수)

앞에서 구한 표를 이용하여 그린 실행시간 그래프이다. 이 실행시간 그래프를 살펴보면, solution1> solution2 > solution3 인 것을 알 수 있다. 이는 n의 값이 커질 때 차이가 더욱 커지는 것을 확인할 수 있다.

앞에서 구한 시간복잡도와 비교해보면 예상한 대로 실행시간이 출력됐음을 알 수 있다.

Solution1 - θ(n^{2}) > Solution2 - θ(nlogn) > Solution3 - θ(n)