KWANGWOON UNIVERSITY

알 고 리 즘

Homework #2

(C++ / Visual Studio 2015)

담당 교수님 : 김용혁 교수님

제 출 날 짜 : 2018. 05. 13.

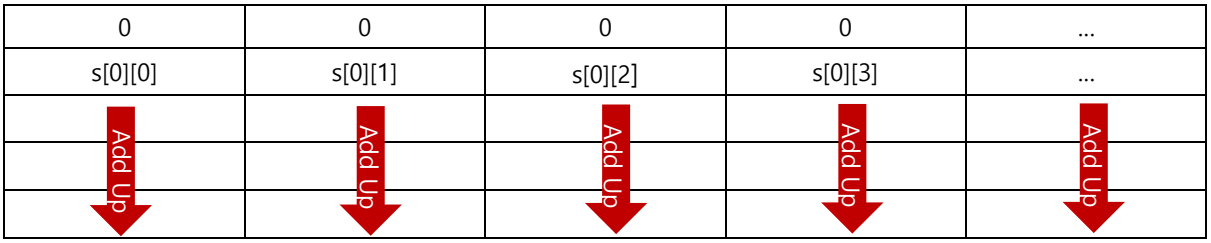
학 과 : 컴퓨터소프트웨어학과

학 번 : 2014707040

이 름 : 유 진 혁

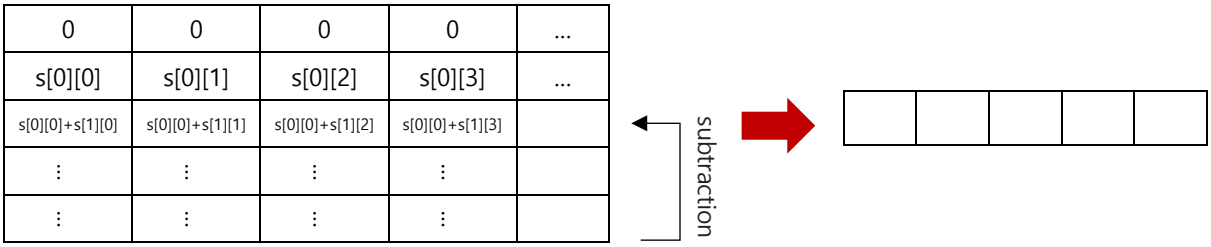
1. 알고리즘 소개

Kadane 알고리즘을 2차원 배열에 적용시켜 과제를 해결하였다. Kadane 알고리즘은 Dynamic Programming 기반 최대 연속부분배열의 합을 구하는 알고리즘으로 이를 이용하여 최소 연속부분배열의 합까지 구하였다. 구현한 알고리즘을 간단히 소개하겠다. 사용자가 입력한 2차원 배열을 s라는 배열에 저장하고 배열 s보다 행이 하나 더 많은 배열 c를 만든다.



<2차원 배열 c>

배열 c의 0번째 행은 0으로 채우고 1번째 행은 배열 s의 0번째 행 값으로 채운다. 이후 나머지 행들은 배열 s에서 아래 방향으로 배열의 값들을 누적하여 더한 값으로 채운다. 이렇게 만들어진 배열 c는 각 행끼리의 뺄셈을 통해 모든 경우의 연속부분배열의 합을 구할 수 있다.



<2차원 배열 c>

<1차원 배열 rMax, rMin>

예를 들어 c의 i번째 행의 값에 j번째 행의 값을 각각 빼 나온 1차원 배열은 사용자가 입력한 배열인 s의 j번째 행부터 (i-1)번째 행까지의 합을 담고 있다. 또, 1차원 배열에서 2번째에서 4번째 원소까지의 합은 배열의 s의 (j, 2)부터 ((i-1), 4)까지의 부분배열의 합과 같다. 따라서 이 배열에 Kadane 알고리즘을 적용하면 연속부분배열 합의 최대값과 최솟값을 구할 수 있다. Kadane 알고리즘은 1차원 배열에서 현재 보고 있는 index의 값과 그 전 index와의 합 값을 대소비교하여 큰 값을 현재 index에 채우는 알고리즘이다. (배열의 0번째 값은 그대로 둔다.) 이렇게 구해진 배열에서 가장 큰 값이 그 배열에서 만들 수 있는 연속부분배열의 합 중 최댓값이 된다.

20	30	-70	15	40
20	**50**	-70	15	40
20	**50**	**-20**	15	40
20	**50**	**-20**	**15**	40
20	**50**	**-20**	**15**	**55**

<Kadane 알고리즘>

반대로 더 작은 값으로 채우고 채운 배열의 값 중 가장 작은 값은 연속부분배열의 합 중 최솟값이 된다. Kadane 알고리즘을 진행하다가 배열의 값이 합 값이 아닌 기존 값이 그대로 채워진다면, 그 index가 최대·최소 연속부분배열의 합을 만드는 부분배열의 시작점이 된다. Kadane 알고리즘을 통해 만들어진 배열에서 가장 큰 값 혹은 작은 값이 있는 index는 부분배열의 끝점이 된다. 위 알고리즘을 배열 c에서 행끼리의 뺄셈을 통해 얻을 수 있는 모든 배열에 대해 수행한 뒤, 이 모든 값 중 최댓값이 2차원 배열의 최대 연속부분배열의 합이 되고 최솟값이 최소 연속부분배열의 합이 된다.

2. 코드 설명

```
///* Input and Initialize *///
int n;  // 배열 크기
int maxSubSum = INT_MIN;    // 얻어진 1차원 배열 내에서 연속부분배열의 합 최댓값
int maxSum = INT_MIN;   // 2차원 배열 전체에서 연속부분배열의 합 최댓값
int minSubSum = INT_MAX;    // 얻어진 1차원 배열 내에서 연속부분배열의 합 최솟값
int minSum = INT_MAX;   // 2차원 배열 전체에서 연속부분배열의 합 최솟값
int maxStartCol1 = 0, minStartCol1 = 0; // 연속부분배열 시작점 예비 후보
int maxStartCol2 = 0, minStartCol2 = 0; // 연속부분배열 시작점 최종 후보
int maxEndCol = 0, minEndCol = 0;   // 연속부분배열 끝점 후보
Point maxStartPoint, maxEndPoint;   // 최대 연속부분배열 합 시작점 , 끝점
Point minStartPoint, minEndPoint;   // 최소 연속부분배열 합 시작점, 끝점

cin >> n;
vector< vector<int> > s;    // 사용자 입력 배열
vector< vector<int> > c;    // 열 누적 합 배열
vector<int> rMax(n);    // 행끼리 뺀 값 들어갈 1차원 배열 for 최댓값
vector<int> rMin(n);    // 행끼리 뺀 값 들어갈 1차원 배열 for 최솟값

/* Vector 크기 설정 */
for (int i = 0; i < n + 1; i++)
{
    vector<int> element(n);
    s.push_back(element);
    c.push_back(element);
}

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        cin >> s[i][j]; // 입력
    }
}
```

변수 선언 및 배열 크기 지정, 사용자로부터 입력을 받는 부분이다. 각 변수에 대한 설명은 주석문을 참고하면 된다. 설명을 덧붙이자면, 배열은 모두 STL의 Vector로 선언하였다. maxSubSum과 minSubSum은 배열 c에서 각 행끼리 뺀 값이 들어가있는 1차원 배열(rMax와 rMin) 내에서 연속부분배열의 합 최대·최소가 들어가는 변수이다. 우리는 모든 부분배열에서 가장 합이 큰 부분배열과 작은 부분배열을 구해야 하므로 maxSubSum, minSubSum에 들어가는 값 중 각각 가장 큰 값과 가장 작은 값이 maxSum, minSum에 들어가게 된다.

```
///* Calculate *///
for (int j = 0; j < n; j++)
    c[0][j] = 0;    // 1번째 행은 전부 0

/* 배열의 세로 방향으로 값을 누적 합하여 배열을 채워간다.*/
for (int j = 0; j < n; j++)
{
    for (int i = 1; i < n + 1; i++)
        c[i][j] = c[i - 1][j] + s[i - 1][j];
}
```

본격적인 계산 부분이다. 우선 입력한 배열의 값들을 세로 방향으로 누적 합하여 그 값을 배열 c에 1번째 행부터 채워나간다. 이 때 배열 c의 0번째 행은 미리 0으로 채워둔다.

```
/* k = (부분배열의 시작 행 - 1)*/
/* i = (부분배열 끝 행) */
/* j = (부분배열의 열) */
for (int k = 0; k < n + 1; k++)
{
    for (int i = k + 1; i < n + 1; i++)
    {
        maxStartCol1 = 0;
        minStartCol1 = 0;
        /* Kadane 알고리즘 적용할 1차원 배열 생성 */
        for (int j = 0; j < n; j++)
            rMax[j] = rMin[j] = c[i][j] - c[k][j];
        maxSubSum = rMax[0];
        minSubSum = rMin[0];
```

3중 반복문을 이용하여 1차원 배열 rMax와 rMin을 만든다. rMax와 rMin에는 배열 c에서 각 행끼리의 뺄셈으로 만들 수 있는 모든 1차원 배열이 한 번씩 들어가게 된다. 그리고 Kadane 알고리즘을 위해 maxSubSum과 minSubSum 값을 각 배열의 0번째 값으로 채워준다.

```
for (int j = 1; j < n; j++)
{
    /*rMax[j] = max(rMax[j], rMax[j] + rMax[j - 1]) */
    if (rMax[j] < rMax[j] + rMax[j - 1])
        rMax[j] = rMax[j] + rMax[j - 1];
    else
        maxStartCol1 = j;
    if (maxSubSum < rMax[j])
    {
        maxSubSum = rMax[j];
        maxStartCol2 = maxStartCol1;
        maxEndCol = j;
    }

    /*rMin[j] = min(rMin[j], rMin[j] + rMin[j - 1]) */
    if (rMin[j] > rMin[j] + rMin[j - 1])
        rMin[j] = rMin[j] + rMin[j - 1];
    else
        minStartCol1 = j;
    if (minSubSum > rMin[j])
    {
        minSubSum = rMin[j];
        minStartCol2 = minStartCol1;
        minEndCol = j;
    }
}
```

Kadane 알고리즘을 진행하는 부분이다. 1번째 index의 값부터 채워준다. 현재 보고 있는 index의 값과 이전 index의 값과의 합 값을 비교하여 합 값이 더 크면 그 값을 현재 index에 더 넣어준다. 그렇지 않다면, 현재 index의 값을 그대로 두고 부분배열의 새 시작점으로 설정해준다. 그리고 새로 설정한 현재 index의 값이 같은 1차원 배열 내에서 최댓값이라면 그 값을 maxSubSum으로 설정하고 앞에서 시작점으로 체크해둔 index를 최대 연속부분배열의 합을 가지는 부분배열의 시작점 후보로 둔다. 이 index의 값이 모든 rMax 배열의 값 중에서도 가장 큰 값이라면 그 때 최종적으로 부분배열의 시작점으로 설정한다. 그리고 현재 index는 최대 연속부분배열의 합을 가지는 부분배열의 끝점 후보로 둔다. 시작점과 마찬가지로 이 index의 값이 모든 rMax 배열의 값 중에서도 가장 큰 값이라면 그 때 최종적으로 부분배열의 끝점으로 설정한다. 최솟값의 경우에는, 부등호의 방향만 반대로 해주고 똑같이 진행하면 된다.

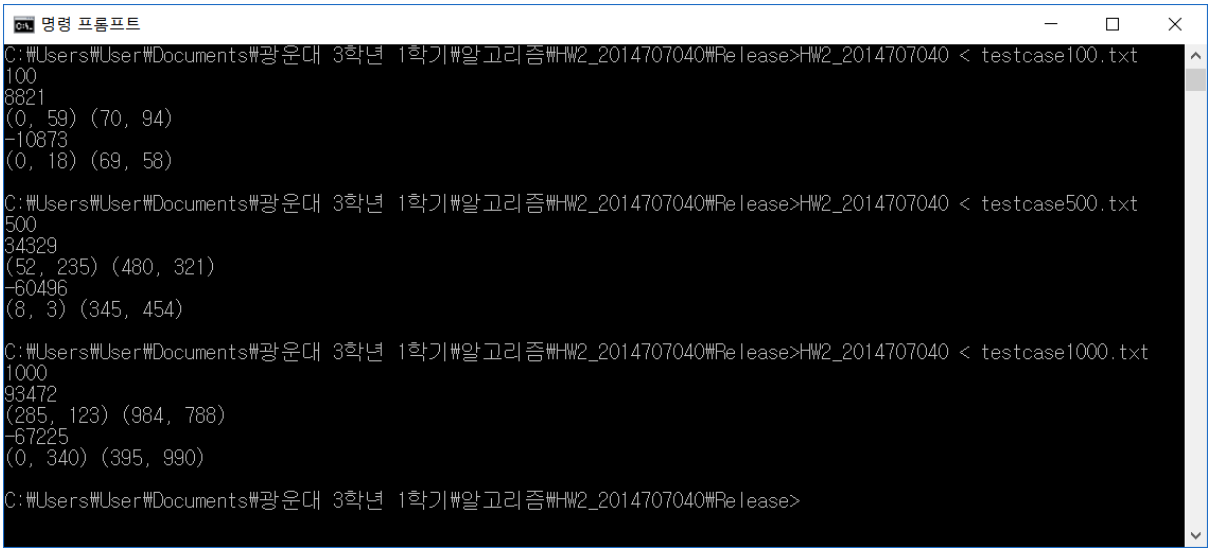
```
if (maxSum < maxSubSum)
{
    maxSum = maxSubSum;
    maxStartPoint.row = k;
    maxStartPoint.col = maxStartCol2;
    maxEndPoint.row = i - 1;
    maxEndPoint.col = maxEndCol;
}
if (minSum > minSubSum)
{
    minSum = minSubSum;
    minStartPoint.row = k;
    minStartPoint.col = minStartCol2;
    minEndPoint.row = i - 1;
    minEndPoint.col = minEndCol;
}
```

1차원 배열에서 구한 최대·최소 연속부분배열의 합이 전체 부분배열들 중에서도 최대·최소인지 확인하는 부분이다. 모든 부분합 중에서도 최대·최소라면 후보로 두었던 시작점, 끝점을 최종적으로 두 점의 열 번호로 설정하고 i와 k의 값을 이용해 시작점, 끝점의 행 번호를 구해준다.

```
///* Print *///
cout << n << endl;
cout << maxSum << endl;
cout << "(" << maxStartPoint.row << ", " << maxStartPoint.col << ") ";
cout << "(" << maxEndPoint.row << ", " << maxEndPoint.col << ") " << endl;
cout << minSum << endl;
cout << "(" << minStartPoint.row << ", " << minStartPoint.col << ") ";
cout << "(" << minEndPoint.row << ", " << minEndPoint.col << ") " << endl;
```

이렇게 구해진 값들이 최종 출력되는 부분이다.

3. 실행 테스트



주어진 testcase100.txt, testcase500.txt, testcase1000.txt를 리다이렉션으로 입력해주고 결과를 출력하였다. 계산하는데 걸린 시간은 개인 PC 기준으로 각각 0.33초, 0.91초, 4.92초가 걸렸다.

4. 고찰

보조교재에 나와있는 Kadane 알고리즘에 대한 설명이 과제를 해결하는데 큰 도움이 되었다. Kadane 알고리즘을 완전히 이해한 뒤, 어떻게 하면 2차원 배열에 Kadane 알고리즘을 적용할 수 있을지 많은 고민을 하였다. 교수님께서 2차원 배열을 세로 방향으로 누적 합하고 그 값들을 다시 가로 방향으로 누적 합하여 배열을 만들면 그 배열의 원소 값은 (0, 0)이 시작점이고 원소의 인덱스가 끝점인 부분배열의 합이라는 것을 설명해주셨다. 이런 방법으로 원소의 값이 부분배열의 합이 되도록 배열을 만들고 Kadane 알고리즘을 적용하면 최댓값과 최솟값 계산이 가능하리라 판단하였다. 문제는 시작점이 (0, 0)이 아닌 부분배열의 합을 구하는 것이었다. 교수님께서 설명해주신 방법에서 약간의 변형이 필요했는데, 배열의 행과 열을 더하고 빼 보다가 각 행을 누적 합한 뒤 두 행끼리의 차로 나온 1차원 배열로 모든 연속부분배열의 합을 표현할 수 있다는 걸 알아냈다. 여기에 이 알고리즘의 시간복잡도가 $O(n^3)$인 것을 이용하여 3중 for문으로 구현하는 것이라는 것을 예측할 수 있었다. 이처럼 최대 합과 최소 합을 구하는 것은 특별히 까다로운 부분 없이 구할 수 있었다. 골치가 아팠던 건 각 시작점과 끝점을 구하는 것이었다. 1차원 배열에서 연속부분배열의 시작점과 끝점을 구하는 것은 간단하였지만 2차원 배열에서는 반복 횟수도 많고 조건문도 까다롭게 나눠야해서 애를 먹었다. 이전 index의 값과의 합 값이 아닌 기존 값이 그대로 들어갈 때 그 지점이 시작점이 될 수 있다는 건 알았지만 그렇게 구한 부분배열의 합이 최대가 아니라면 그 시작점을 무시해야했다. 이런 복잡한 알고리즘을 조건문들 속에 잘 녹여야 했는데 시작점 후보 변수를 두 개 두는 방식으로 해결하였다. 두 변수는 각각 시작점의 예비, 최종 후보가 되어 단계적으로 모든 조건을 만족할 때 최종 후보의 지점이 시작점의 열이 되도록 하였다.

구현하는 동안 크고 작은 오류들이 발생했는데 특히 논리 오류가 자주 발생하였다. 알고리즘은 복잡한데 머릿속으로 확실히 정리되지 않은 상태에서 무작정 코딩부터 시작하니 원하는 결과에 도달하기까지 시간이 오래 걸렸다. 디버깅을 하는 동안에도 잘못된 부분을 찾기 위해 소스코드를 한줄한줄 따져가며 찾아야만 했다. 논리적으로 복잡한 알고리즘을 구현할 때는 코딩에 앞서 확실히 논리를 정리한 다음에 코딩을 시작해야 효율적이고 빠르게 구현할 수 있을 것으로 보인다.

이렇게 구현한 알고리즘은 1000X1000 배열을 넣고 실행했을 때 5초 안팎의 시간이 소요된다. 구현한 알고리즘에 따라 소요시간이 달라진다는 것이 신기했다. 또, CPU의 사용률이 높을 땐 시간이 더 걸렸고 부하가 적을 땐 비교적 빠른 시간 내에 연산이 끝났다. 이처럼 소요 시간은 PC 환경에도 영향을 어느정도 받는 것으로 보인다.