**Data Structure**

**~ Page Rank ~**

**2013210111 남세현**

1. **환경**

Visual Studio 2013에서 개발. ANSI 표준으로 코드를 작성했기 때문에, gcc에서도 동작할 것이라 생각됩니다.

1. **결과**

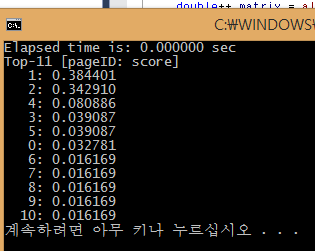


Figure : pagerank Matrix

1. **구현 내용**

PageRank\_Matrix.c 에서 generateTPMatrix와 calculatePageRank를 구현했습니다.

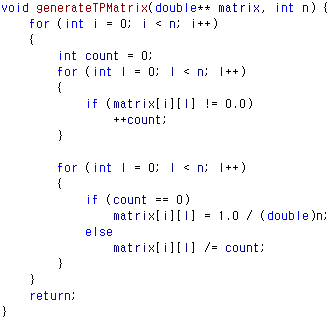


Figure : generateTPMatrix

generateTPMatrix는

1. 한 행에 들어있는 entity의 갯수를 세어서 count에 저장하고,
2. 한 행에 들어있는 entity들을 1.에서 구한 count로 나눠줍니다.
3. 만약 count가 0이면 1/n을 넣어줍니다.

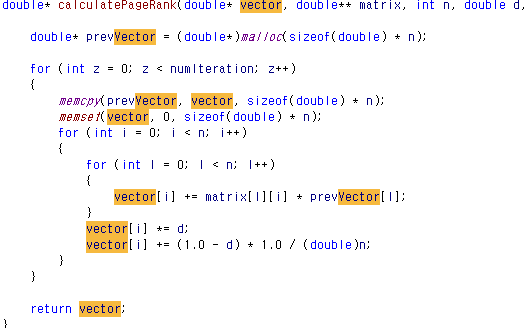


Figure : calculatePageRank

calculatePageRank는

1. a번째 벡터를 구하기 위해 a-1번째 벡터를 prevVector에 저장하고,
2. a벡터를 0으로 리셋합니다.
3. 각 행에 prevVector를 곱한 값을 vector에 넣어줍니다.
4. 3.의 연산 후에 damping 계수를 벡터에 곱해줍니다.
5. 그 후, 랜덤점프 probability를 벡터에 더해줍니다.
6. 1~5 과정을 입력한 number of iteration값 만큼 반복합니다.

PageRank\_List.c에서는 calculatePageRank를 구현했습니다.

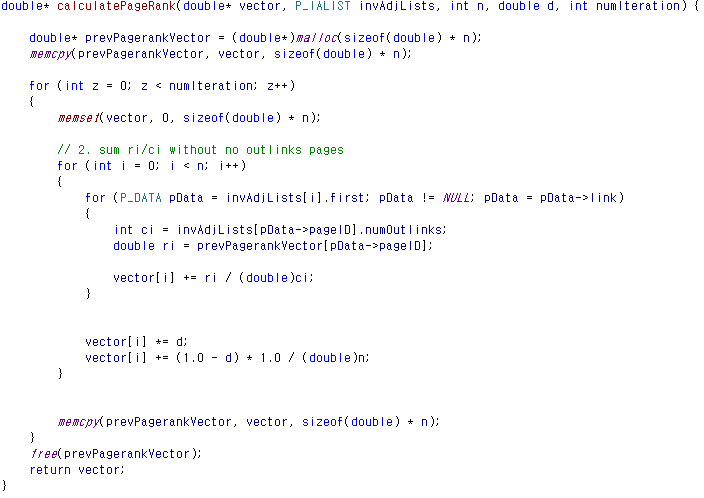


Figure : calculatePageRank(list)

구현 방법은,

1. a번째 벡터를 구하기 위해 a-1벡터를 저장해놓아야 하므로, prevPageRankVector에 vector를 저장합니다.
2. 그 후 vector를 0으로 초기화합니다.
3. 페이지 i를 가리키는 페이지들을 순회하면서 ci와 ri의 값을 구합니다.
   1. Ci는 그 페이지의 outlink number이며
   2. Ri 는 a-1번째 벡터에 저장된 page rank 값입니다.
4. Ci/ri의 sum을 벡터에 더한 후, damping 계수를 곱해줍니다.
5. 그리고 random jump probability를 더해줍니다.
6. 1~5 과정을 iteration number만큼 반복합니다.
7. **결과 분석**

주어진 Data Set 중 Wikipedia 항목은 page갯수가 적어서 동작 시간의 차이를 크게 못느낍니다.

그래서 같이 첨부되어 있는 Movie 데이터로 결과를 분석해보겠습니다.

총 page의 갯수는 5757개입니다. 총 5번 iteration을 돌았습니다.

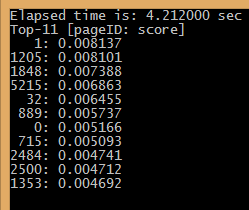


Figure : List -- 4.2 sec

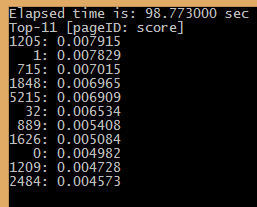


Figure : Matrix -- 98.7 sec

엄청난 구동 시간 차이가 남을 알 수 있습니다. PPT에서도 나온 것 처럼, RunningTime은 매트릭스일 때 n^2의 복잡도를 가지다가, 벡터로 계산할 때에는 o \* n으로 줄어듬을 알 수 있습니다.

