PNU Algorithm Study

2019 Summer Week 4

PNU CSE AICall 박성수
2019. 7. 30
tjdtnsu.blog.me

Divide and Conquer

분할 정복

분할 정복

그대로 해결할 수 없는 문제를 작은 문제로 분할하여 문제를 해결하는 방법

Step 1. 분할과정 - 만일 바로 해결 가능하면 그 자리에서 해결

else 아니라면 적절한 크기로 쪼갠다. (divide)

Step 2. 정복과정 - 각각 잘린 문제를 해결한다. (solve or conquer)

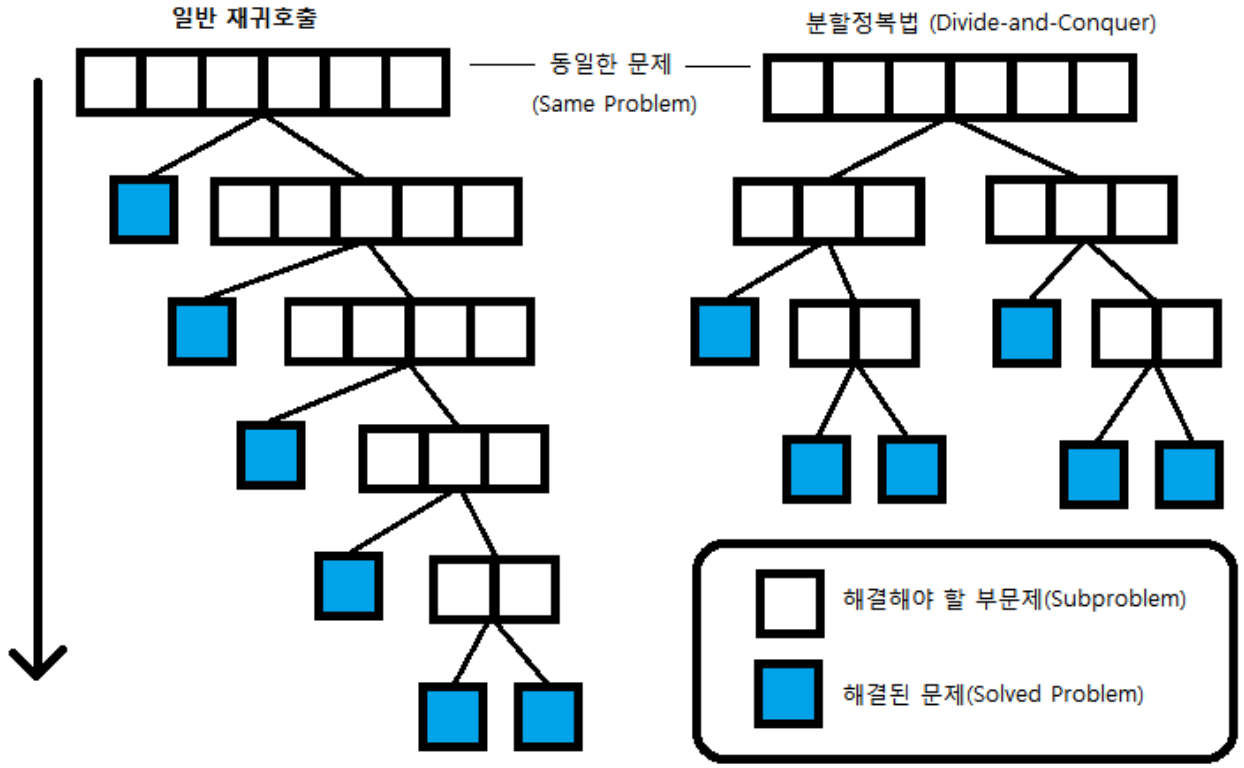
만일 그 크기가 크다고 생각하면 다시 쪼갠다.

그리고 Goto Step 1.

Step 3. 통합과정 : 각 <부분 해(partial solution)>를 종합하여 해를 구성. (merge)

일반 재귀호출과의 차이

```
void func(int a, int b) {
  if(a == b) solve(a);
  else {
    int c = (a+b) / 2;
    func(a, c);
    func(c+1, b)
  }
}
```



분할 과정에서의 고려 사항

주어진 문제를 언제, 어떻게, 얼마나 잘게 쪼갤 것인가 ?

- 수박을 분자단위까지 쪼갤 것인가 ?
- 적절한 단위가 되면 “무식하게” 하는 것이 더 나은 방법이다.
- Improved Quick Sort, Improved Binary Search
- Binary(10)과 Linear(10)중에서 어떤 것이 빠른지 측정할 필요 있음

이 경우 분할 작업을 중지할 데이터 크기의 임계점을 찾아야 한다.

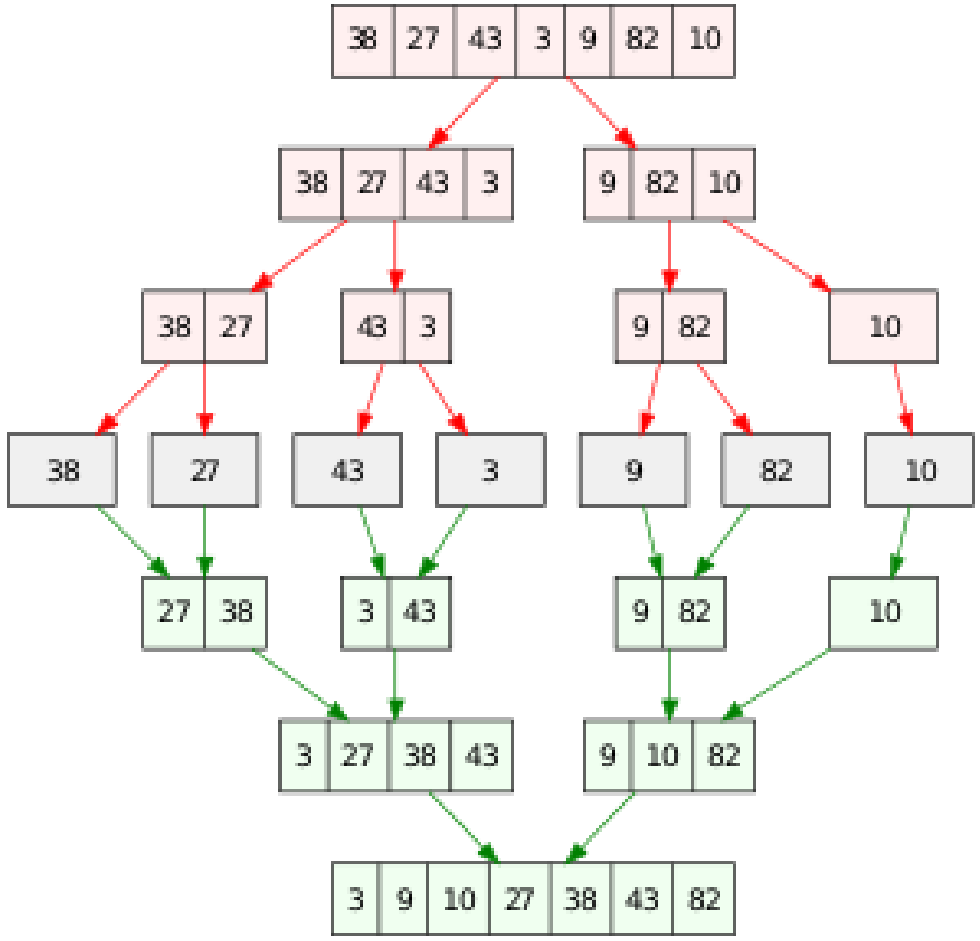
병합 정렬

분할 방법

한 데이터 벡터를 두 개로 쪼갬

병합 방법

1. 두 데이터 벡터의 크기의 합의 크기를 가지는 벡터 생성
2. 두 데이터 벡터의 요소들을 비교해서 작은 것부터 하나씩 생성
3. 두 데이터 벡터의 인덱스가 모두 찰 때까지 반복



백준 - 1629 곱셈

문제

자연수 A를 B번 곱한 수를 알고 싶다. 단 구하려는 수가 매우 커질 수 있으므로 이를 C로 나눈 나머지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 A, B, C가 빈 칸을 사이에 두고 순서대로 주어진다. A, B, C는 모두 2,147,483,647 이하의 자연수이다.

출력

첫째 줄에 A를 B번 곱한 수를 C로 나눈 나머지를 출력한다.

백준 - 1629 곱셈

무식한 접근법

a를 for문으로 b번 곱해서 c로 나눈 나머지 구함

분할 정복 접근

- b를 2로 나눈 몫에 대해 승수 값을 구함
- 홀수이면 나머지가 1, (a * t * t) % c return
- 짝수이면 나머지가 0, (t * t) % c return

모듈로 성질 활용

(a * b) % n = ((a % n) * (b % n)) % n

```
#include <iostream>
using namespace std;

int a, b, c;

int power(int n, int k){
    if(k == 0) return 1;

    int temp = power(n, k/2);
    int result = 1LL * temp * temp % c;
    if(k%2) result = 1LL * result * n % c;
    return result;
}

int main(){
    cin >> a >> b >> c;
    cout << power(a, b);
}
```

Dynamic Programming

동적 계획법

동적 계획법

동적 계획법 = 분할정복 + 메모이제이션
(중복되는 부분 문제 저장)

메모이제이션을 적용할 수 있는 경우?
- **참조적 투명성을** 만족
  = 동일한 입력에서 같은 답이 나와야 한다

동적 계획법의 원리

주어진 문제 X보다 약간 더 쉬운 문제 Y를 풀 수 있다면
Y 결과의 답을 모아서
주어진 문제 X의 답을 좀 더 쉽게 찾을 수 있다.

동적 계획법은 중복되는 부분 문제의 답을 여러 번 계산하는 대신
한번만 계산하여 속도 향상을 꾀하는 알고리즘 설계 기법

동적 계획법과 분할 정복의 차이점은
"중복되는 부분 문제 (overlapping sub-problems)"

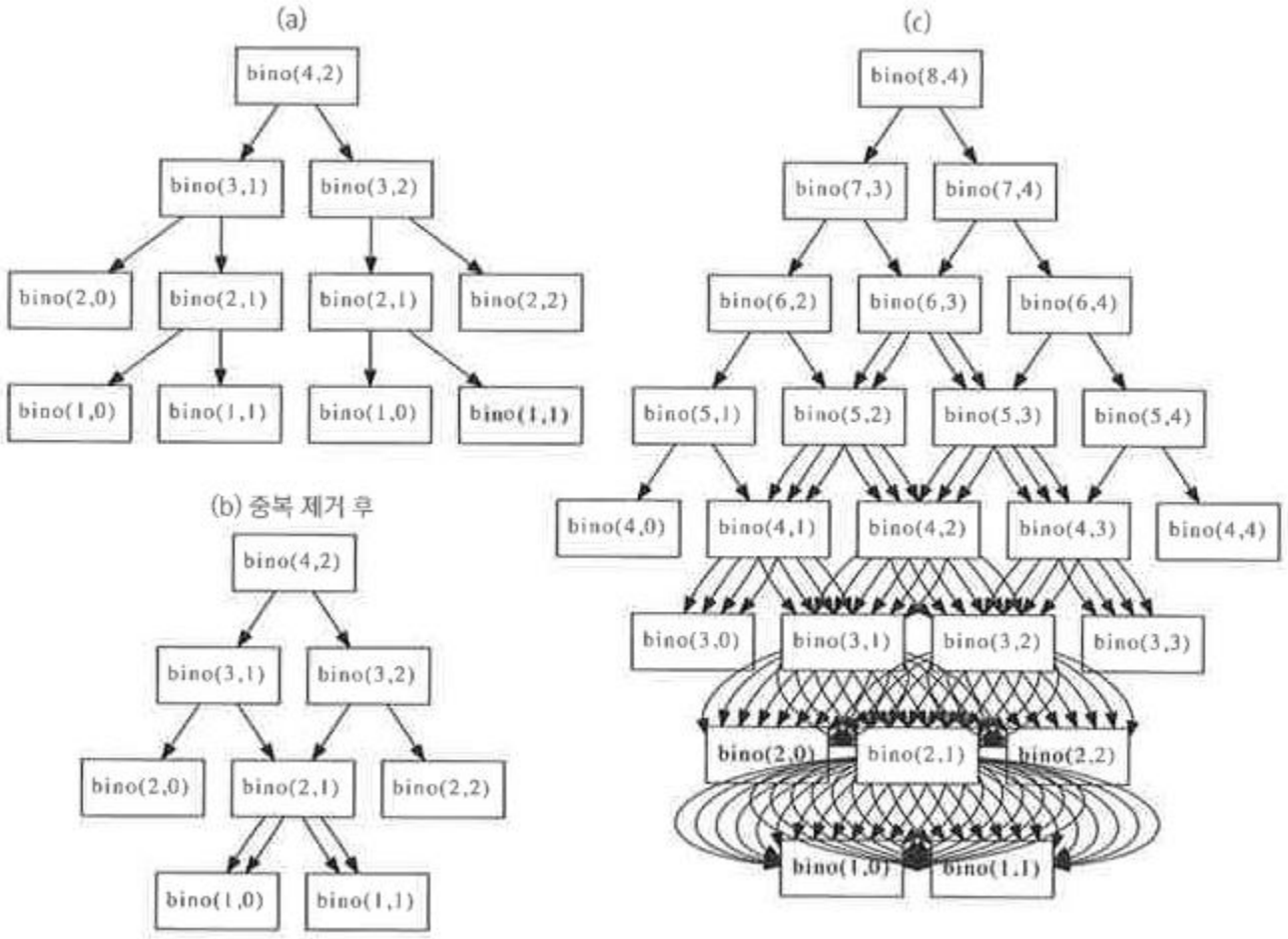
처음 주어진 문제를 더 작은 문제들로 나눈 뒤
각 조각의 답을 계산하고
이 답들로부터 원래 문제에 대한 답을 계산 해냄

각 문제의 답을 메모리에 저장해 둘 필요가 있다 - 메모리제이션

이항계수

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

그래프를 살펴본다면
dp를 적용했을 때 계산 수는 **네모 박스 수**에 따라가고
dp를 적용하지 않았다면 **화살표 수**에 따라간다



메모리제이션 예제

```
for(int i=0; i!=n; ++i)   // for example
        arr[i] = min(arr[i-1], arr[i-2]+arr[i-3]) + 3;
```

```
int arr[500][500];   // function example

int func(int a, int b){
        if(arr[a][b] != 0) return arr[a][b];
        else return func(a-1, b-2) + func(a-2, b-1);
}
```

준비물

a) 동적계획 공식 (Dynamic Programming formula)

- 주로 recursive form으로 제시

b) 초기조건 (Base condition)

c) 적당한 크기의 Table[ ][ ] 또는 몇 개의 Table[ ][ ]

순서

1. 값 초기화
2. 구하려고 하는 값 dp로 계산
3. 출력

백준 - 1149 RGB거리

문제

RGB거리에 사는 사람들은 집을 빨강, 초록, 파랑중에 하나로 칠하려고 한다. 또한, 그들은 모든 이웃은 같은 색으로 칠할 수 없다는 규칙도 정했다. 집 i의 이웃은 집 i-1과 집 i+1이다.

각 집을 빨강으로 칠할 때 드는 비용, 초록으로 칠할 때 드는 비용, 파랑으로 드는 비용이 주어질 때, 모든 집을 칠할 때 드는 비용의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 집의 수 N이 주어진다. N은 1,000보다 작거나 같다. 둘째 줄부터 N개의 줄에 각 집을 빨강으로 칠할 때, 초록으로 칠할 때, 파랑으로 칠할 때 드는 비용이 주어진다. 비용은 1,000보다 작거나 같은 자연수이다.

출력

첫째 줄에 모든 집을 칠할 때 드는 비용의 최솟값을 출력한다.

백준 - 1149 RGB거리

문제 목적 : 구하려는 것은 비용의 최솟값

무식한 접근법 : 하나씩 선택해 가기(3의 n승)

이전 값과 현재 값의 차이 : 이전 합에 현재 값을 더한 것

dp적 접근

- 0번째의 최소 비용을 구한다. (같음)
- 1번째 집이 빨강일 때 최소 비용 = min(sG[0], sB[0])+R[1])
- i번째 집이 빨강일 때 최소 비용 = min(sG[i-1], sB[i-1])+R[i]

정답 : min(sR[i], sG[i], sB[i])

백준 - 1149 RGB거리

```
#include <iostream>
using namespace std;

int main() {
	int n;
	int rgb[1000][3]={0}, cst[1000][3]={0};

	cin >> n;

	cin >> rgb[0][0] >> rgb[0][1] >> rgb[0][2];

	cst[0][0] = rgb[0][0];
	cst[0][1] = rgb[0][1];
	cst[0][2] = rgb[0][2];

	for(int i=1; i!=n; ++i) {
		cin >> rgb[i][0] >> rgb[i][1] >> rgb[i][2];
		cst[i][0] = min(cst[i-1][1], cst[i-1][2]) + rgb[i][0];
		cst[i][1] = min(cst[i-1][0], cst[i-1][2]) + rgb[i][1];
		cst[i][2] = min(cst[i-1][0], cst[i-1][1]) + rgb[i][2];
	}

	cout << min(min(cst[n-1][0], cst[n-1][1]), cst[n-1][2]);

	return 0;

}
```

Greedy Algorithm

탐욕법 / 욕심쟁이법

동적 계획법과 탐욕법

Dynamic programming
완전탐색

vs

Greedy Algorithm
탐욕적

재귀호출 등을 이용 → 부분 문제화

모든 선택지를 고려한 후,
그 중 가장 좋은 값 선택

각 단계에서 가장 좋은 방법만을 선택

탐욕법의 구조

1. 어떤 알고리즘은 단계적인 선택을 거쳐서 전체를 완성한다고 하자.

2. 한 단계에서 주어진 정보가 { I } 이고 선택이 $\{C_0, C_1, \cdots, C_k\}$라고 할 때 현재 정보만을 이용해서 최선의 결과를 얻을 수 있는 경우를 선택한다.
 - 단 현재 구성중인 Solution은 고칠 수 없다.

3. Constructive Algorithm VS. Iterative Algorithm
 - 결혼하기(한번 결정하고 나면 되돌리기 어려움)
 - 친구 사귀기 (계속 개선의 여지가 있음)

탐욕법 사용

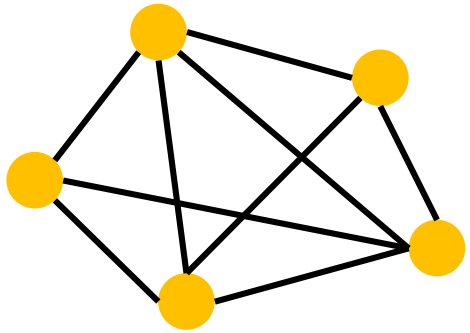
1. 탐욕법을 사용해도 항상 최적해가 나오는 문제

   - 동적 계획법 보다 수행 시간이 훨씬 빠르기 때문에 탐욕법이 유리

2. 시간, 공간적 제약으로 최적해를 찾기 어려울 경우 근사해를 찾기 위해 사용

   - 최적은 아니지만 임의의 답보다는 좋은 답을 구할 수 있음

Kruskal, Prim's Algorithm



Weight된 그래프에서 Minimum Spanning Tree를 구성하는 알고리즘

Kruskal's Algorithm : 모든 간선 중 weight가 가장 작은 간선 선택 후 구성 및 반복, 여기서 cycle이 나타나면 제외

Prim's Algorithm : 어느 임의의 정점 선택, 정점을 지나는 간선 들 중 가장 weight가 작은 간선 선택, 선택된 정점들로 범위를 확장해 반복, 여기서 cycle이 나타나면 제외

백준 - 1931 회의실배정

문제

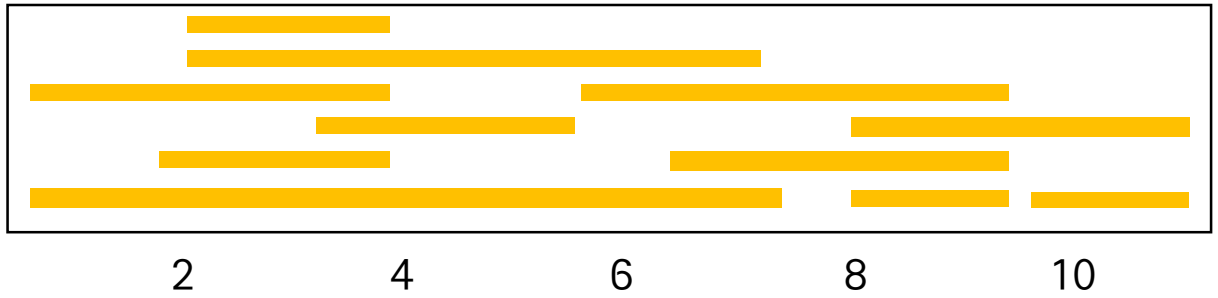
한 개의 회의실이 있는데 이를 사용하고자 하는 N개의 회의들에 대하여 회의실 사용표를 만들려고 한다. 각 회의 I에 대해 시작시간과 끝나는 시간이 주어져 있고, 각 회의가 겹치지 않게 하면서 회의실을 사용할 수 있는 최대수의 회의를 찾아라. 단, 회의는 한번 시작하면 중간에 중단될 수 없으며 한 회의가 끝나는 것과 동시에 다음 회의가 시작될 수 있다. 회의의 시작시간과 끝나는 시간이 같을 수도 있다. 이 경우에는 시작하자마자 끝나는 것으로 생각하면 된다.

입력

첫째 줄에 회의의 수 N(1 ≤ N ≤ 100,000)이 주어진다. 둘째 줄부터 N+1 줄까지 각 회의의 정보가 주어지는데 이것은 공백을 사이에 두고 회의의 시작시간과 끝나는 시간이 주어진다. 시작 시간과 끝나는 시간은 2^{31}-1보다 작거나 같은 자연수 또는 0이다.

출력

첫째 줄에 최대 사용할 수 있는 회의 수를 출력하여라.



백준 - 1931 회의실배정

무식한 접근법 : 모든 부분 집합을 만들어 회의들이 겹치지 않는 답을 제외한 가장 큰 부분 집합 찾기 (2의 n승)

탐욕법 접근 1 : 길이가 짧은 회의부터 하나하나 순회하면서 겹치지 않는 것들을 추가하는 방법

(반례 : 긴 회의 두 개를 선택하지 않고 짧은 회의 하나를 선택하는 경우 발생 → 최적해 NO)

탐욕법 접근 2 : 길이와 상관 없이 가장 먼저 끝나는 회의부터 선택

1. 목록 S에 남은 회의 중 가장 일찍 끝나는 Smin 선택
2. Smin과 겹치는 회의를 S에서 모두 지운다.
3. S안에 원소가 없을 때까지 수행

백준 - 1931 회의실배정

탐욕적 선택 속성 : 탐욕적으로 선택해도 최적해를 구할 수 있다.

가장 종료시간이 빠른 회의(S_{min}) 을 포함하는 최적해가 반드시 존재한다.

→ S 의 최적해 중에 S_{min} 을 포함하지 않는 해가 존재

이 목록 중 가장 먼저 개최되는 회의를 지우고 S_{min} 을 추가해도 S_{min}이 이전에 지워진 회의보다

빠르거나 같은 시간에 끝나게 됨으로 나머지 회의 목록에 영향을 주지 않아 해가 된다.

최적부분구조 : 항상 최적의 선택을 내려서 전체 문제의 최적해를 얻을 수 있다.

// 대부분 당연하지만 , 처음 선택은 최적을 선택해야 하지만

남은 부분 문제는 최적이 아닌 방법으로 풀어야 하는 경우

첫째 회의를 잘 선택하고 겹치는 회의를 모두 걸렀다면 다음에도 당연히 최대한 많은 회의를 선택하여야 함으로 성립

백준 - 1931 회의실배정

동적 계획법 접근:

▶ meeting (idx) 를 선택할지 안 할지 매 단계에서 고민

idx번 회의가 시작하기 전에 끝나는 회의들 중 마지막 회의 번호를 before[idx]에 저장

< Schedule(idx) >

선택 하지 않는 경우 : schedule(idx-1)

선택 하는 경우 : 1+schedule(before[idx])

Before[] 을 만드는데 $O(n \lg n)$

schedule()의 수행에 $2O(n)$

탐욕적으로 최적해를 찾을 수 있는 문제

(다음 한 단계만 고려)

동적 계획법으로 풀 수 있음

(다음의 모든 단계 고려)

메모리, 시간이 많이 걸림


Tax Refund Form
Financial plan
BUSINESS
Economy of the European Union

끝.