16 그리디 알고리즘

20150375 김명원

- 최적화 문제에서 동적프로그래밍을 사용하면 가장 좋은 선택을 결정하기 위해 지나치게 많은 일을 하게 된다.

- 그리디 알고리즘은 각 단계에 있어서 가장 좋을 거라 생각되는 선택을 한다

16.1 활동 선택 문제

- 한번의 활동을 단 한개만 처리할 수 있는 강의실과 같은 자원을 요구하는 n개의 제안된 활동의 집합 S={ a1,a2,…aN} 이 있다 가정

- 각 ai는 시작시간 si와 종료시간 fi가 존재
- [si,fi)로 존재
- 서로 양립 할 수 있는 활동들로 이루어진 최대크기의 부분집합을 찾고자 한다

- 종료시간이 단조증가하는 순서로 정리되있다 가정 (f1 <= f2 <= …   fN)

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- {a3,a9,a11} ,{a1,a4,a8,a11},{a2,a4,a9,a11}

- 최대 크기 부분집합은 위의 4개 원소갖는 집합

활동 선택 문제의 최적 부분구조

- Sij : 활동 ai 가종료한 후에 시작
  활동 aj 가 시작하기 전에 종료하는 활동들의 집합

  Aij : ak를 포함하는 최대크기의 집합

- 최적해에 ak를 포함하면 두개의 부분문제가 남는다

1. 집합 Sik에 들어있는 상호 양립 가능한 활동을 찾는것

2. 집합 Skj에 들어있는 상호 양립 가능한 활동을 찾는것

- (Aik = Aij 교집합 Sik)   (Akj = Aij 교집합 Skj) 라 하자

--> Aik는 Aij에서 ak가 시작하기 전에 끝나는 활동을 포함
--> Akj는 Aij에서 ak가 끝난 후에 시작하는 활동을 포함함

따라서 Aij= Aik 합집합 {ak} 합집합 Akj   , Sij에 들어있는 상호 양립 가능한 활동들의 최대 집합 Aij-> |Aij| = |Aik|+|Akj|+1

- 자르고 붙여넣기 주장에 의하면 최적해 Aij는 두개의 부분 문제들 Sik 와 Skj에 대한 최적해를 포함해야 함을 보여준다

- 이런 방식의 최적 부분구조는 이런 문제를 동적 프로그래밍을 사용해 풀 수도 있음을 보여준다.
- Sij를 위한 최적해 크기 : c[i,j]

$$c[i, j] = \begin{cases} 0 & S_{ij} = \emptyset\text{일 때} \\ \max\limits_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & S_{ij} \neq \emptyset\text{일 때} \end{cases}$$

- Sij에 대해 ak를 포함하는 최적해를 모르면 Sij에 들어있는 모든 활동을 조사(위그림)

--> 이런식으로 Dp로 풀면 활동 선택 문제의 중요한 특징을 간과 하게 됨

그리디 선택하기

- 만약 모든 부분문제를 먼저 풀 필요 없이 최적해에 넣을 한개의 활동을 선택할 수 있다면???

-> S에 들어있는 활동 중 가장 빠른 종료 간을 갖는 활동을 먼저 선택

-> 그 활동 뒤에 더 많은 활동을 채울 수 있다

앞에서 a1이 종료시간이 제일 빠르다 가정

따라서 우리는 a1을 픽한다

- 이제 a1이 종료한 후 시작하는 활동들만 가지고 찾는 문제가 남는다.

- a1이 시작하기 전에 끝나는 활동은 교려할 필요가 없을까??

  --> s1 < f1이고   f1이 가장 빠른 종료시간

  --> 어떤 활동도 s1보다 작거나 같은 종료 시간을 가질 수 없다

- 활동 선택문제를 dp로 수행할 필요가 없다

- 그 대신 최적해에 넣을 활동을 한 개 선택하고 이미 선택된 활동들과 양립 가능한 활동들 중에서 활동을 선택하는 부분문제를 풀면서 하향식으로 수행할 수 있다.

결론: 선택을 하기전 부분문제를 푸는 대신 한개를 선택하고 부분 문제를 푼다

재귀 그리디 알고리즘

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]    // Sk에서 종료 시간이 가장 빠른 활동을 찾는다.
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

- 풀어야 하는 부분문제 Sk의 k와
  원래 문제의 크기 n을 입력으로 받아들인다.

```
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6    else return Ø
```

그림 16.1은 이 알고리즘이 동작하는 방법을 보여준다. 먼저 주어진 재귀 호출인 RECURSIVE-ACTIVITY-SELECTOR(s, f, i, n)에서 2–3행의 **while** 루프는 S_k에서 첫 번째로 종료하는 활동을 찾는다. 이 루프는 a_k와 양립 가능한 첫 번째 활동 a_m을 찾을 때까지 $a_{i+1}, a_{i+2}, \ldots, a_n$을 살펴본다. 여기서 a_m은 $s_m \geq f_k$라는 조건을 만족한다. 이런 활동을 찾아 이 루프가 끝나면 5행은 $\{a_m\}$과 RECURSIVE-ACTIVITY-SELECTOR(s, f, m)의 재귀 호출로부터 얻어진 S_m의 최대 크기의 부분 집합과의 합집합을 출력한다. 또 다른 가능성은 루프가 $m > n$이라서 종료될 수 있는데, 이때는 a_k와 양립 가능한 것을 찾지 못하고 S_k에 있는 모든 활동을 다 검사한 경우다. 이 경우에는 $S_k = \emptyset$이므로 6행에서 $\emptyset$을 출력한다.

주어진 재귀 호출 RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$의 수행시간은 $\Theta(n)$인데, 활동이 종료 시간에 의해 이미 정렬되어 있다고 가정하면 다음과 같이 생각할 수 있다. 모든 재귀 호출에 대해 각각의 활동은 2행의 **while** 루프 검사에서 정확히 한 번만 검사된다. 특히 활동 a_i는 $k < i$인 마지막 호출에서 검사된다.

반복 수행 그리디 알고리즘

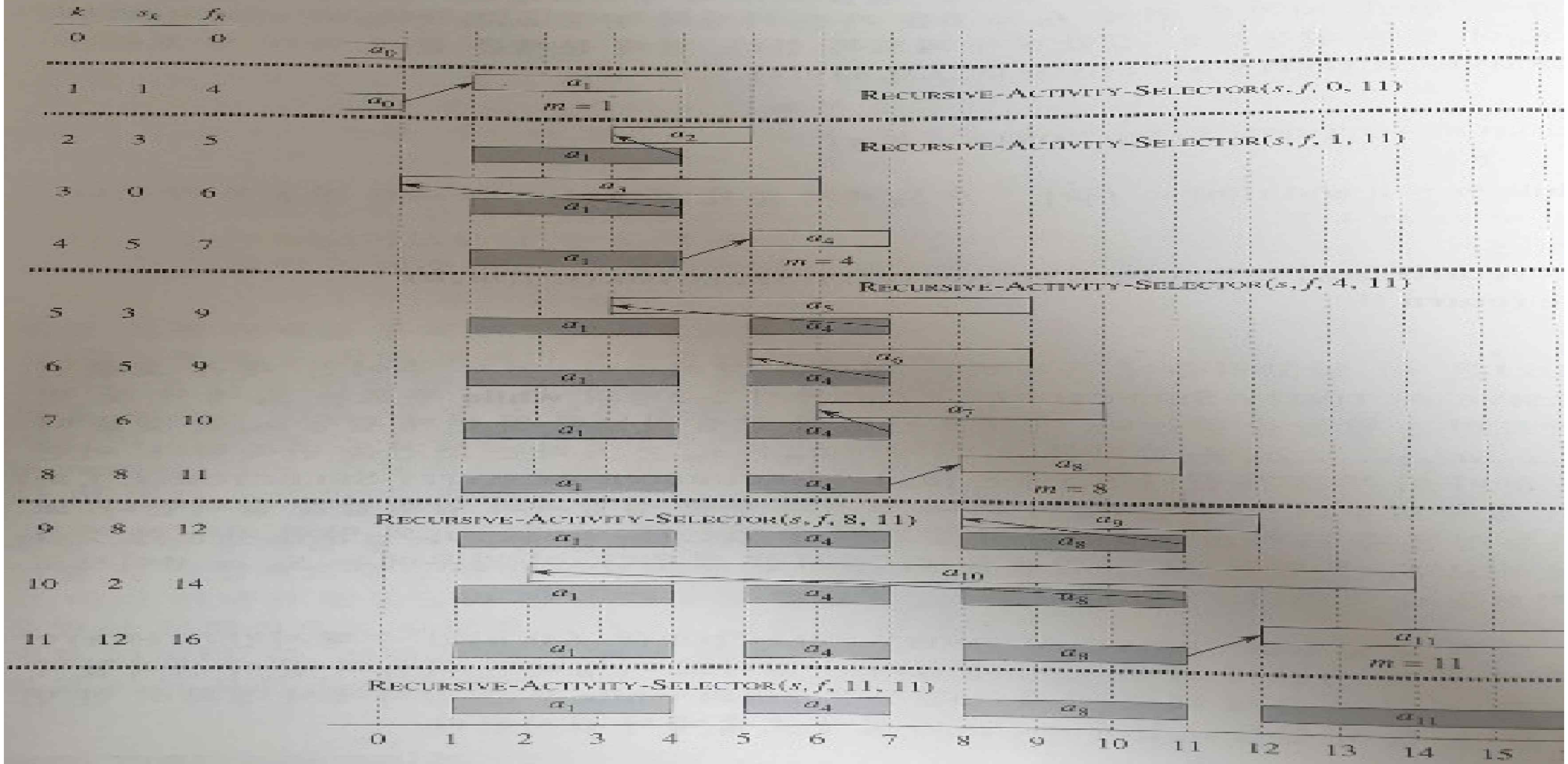

그림 16.1 앞서 주어진 11개의 활동에 대한 RECURSIVE-ACTIVITY-SELECTOR의 동작. 각 재귀 호출에서 고려되는 가로선 사이에 나타난다. 가상의 활동 a_0은 시간 0에서 끝나며 첫 번째 호출 RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0,$ 활동 a_1을 선택한다. 각 재귀 호출에서 이미 선택된 활동들은 회색으로 진하게 나타냈으며 흰색으로 표시된 활동은 이 될 것이다. 어떤 활동의 시작 시간이 가장 최근에 더해진 활동의 종료 시간보다 먼저 나타나면(둘 사이의 화살표는 왼 다), 그것은 바로 버려진다. 그렇지 않은 경우에는(화살표가 위로 향하거나 오른쪽으로 향하면), 그 활동이 선택된다. 더 RECURSIVE-ACTIVITY-SELECTOR$(s, f, 11, 11)$은 $\emptyset$을 출력한다. 선택된 활동들의 결과의 집합은 $\{a_1, a_4, a_8, a_{11}\}$이다.

반복 순환 그리디 알고리즘

```
GREEDY-ACTIVITY-SELECTOR(s, f)
  n = s.length
  A = {a_1}
  k = 1
  for m = 2 to n
      if s[m] ≥ f[k]
          A = A ∪ {a_m}
          k = m
  return A
```

활동들은 종료시간이 단조적으로 증가한다(가정)

fk는 A에 있는 어떤 활동보다 큰 종료시간

$$f_k = \max\{f_i : a_i \in A\} \tag{16.3}$$

2–3행은 활동 a_1을 선택하고 A가 활동 a_1만 가지게 초기화하며 k는 이 활동을 가리키도록 초기화한다. 그리고 4–7행의 **for** 루프는 S_k에서 가장 빠른 종료 시간을 갖는 활동을 찾는다. **for** 루프는 각 활동 a_m을 순서대로 검사하고, a_m이 이전에 선택된 모든 활동과 양립 가능하면 A에 a_m을 추가한다. 그리고 이런 활동은 S_k에서 가장 빠른 종료 시간을 갖는다. 활동 a_m이 현재 A에 있는 모든 활동과 양립 가능한지 알아보려면 식 (16.3)에 의해 5행에서 시작 시간 s_m이 A에 가장 최근에 추가된 활동의 종료 시간 f_i보다 빠르지 않음을 확인하면 된다. 활동 a_m이 양립 가능하면 6–7행에서 a_m이 A에 추가되고 k가 m값을 갖는다. GREEDY-ACTIVITY-SELECTOR(s, f)의 호출에 의해 리턴되는 집합 A는 RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$의 호출에 의해 리턴되는 집합과 동일하다.

재귀 버전처럼 모든 활동이 이미 종료 시간에 따라 정렬되어 있다는 전제로 GREEDY-ACTIVITY-SELECTOR는 n개의 활동의 집합에 대한 스케줄링을 $\Theta(n)$ 시간에 끝낸다.