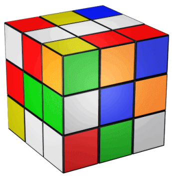
탐욕적 알고리즘 소개

한국기술교육대학교 컴퓨터공학부 김상진



```
while(!morning){
    alcohol++
    dance++
} #party
```

```
while(!sleep){
    think++
    solve++
} #cse-mode
```

교육목표

- 탐욕적 알고리즘
  - 반복적으로 그 순간의 가장 최선의 선택을 하는 알고리즘
    - 이와 같은 선택을 통해 얻어진 해가 알고리즘을 통해 얻고자 하는 해가 맞는지 증명해야 함
- 살펴보는 문제
  - 거스름돈 문제
  - 캐시 문제
  - 3종류의 스케줄링 문제

탐욕적 알고리즘

- 정의
  - 반복적으로 결정, 끝에 좋은 결과가 있기를 희망
    - 어떤 선택을 할 때, 그 당시에 가장 최선의 선택을 하여 문제 해결
    - 이와 같은 선택들이 모여 만든 최종해가 최적해인지는 보장되지 않음
      - 증명 필요
  - 예) 다익스트라 최단 경로 알고리즘
- 분할 정복과 비교
  - 탐욕적 알고리즘을 설계하는 것은 쉬움
  - 시간 복잡도를 분석하기 쉬움
  - 정확성 증명은 힘듦
- 증명 방법
  - 방법 1. 귀납법
  - 방법 2. exchange argument (모순 증명)
    - 스케줄링 알고리즘 증명 참고

탐욕적 알고리즘의 형태

- 알고리즘의 형태

  - 종료조건: 선택할 것이 있는지 여부

```
S := ∅
while condition do
    s := select()                                  // 선택 작업
    if feasible(s) then                            // 타당성 조사
        add s to S
        if problem_solved(S) then return S         // 해답조사
```

- 탐욕적 알고리즘의 3요소
  - 선택 작업(selection procedure): 현 단계의 최적해를 선택
  - 타당성 조사(feasibility test): 선정된 것을 사용할 수 있는지 검사
  - 해답 조사(solution check): 지금까지 만든 해가 최종해가 되는지 검사하는 과정

거스름돈 문제 (1/2)

LeetCode 322

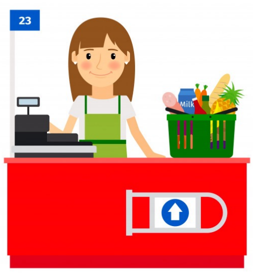
- **입력.** 동전의 액면가 정보, 거스름액
- **출력.** 거스름을 구성하기 위해 필요한 최소 동전의 수
- **예)** 입력: {10, 5, 1}, 21, 출력: 3
- 탐욕적 알고리즘

loop	i	coin	coin <= change	count	change
1	1	10	T	1	11
2	1	10	T	2	1
3	1	10	F		
4	2	5	F		
5	3	1	T	3	0

```
coinChange(d[], change)
count = 0
i = 1
while i ≤ d.length do
    coin := d[i]                      // 선택 작업
    if coin ≤ change then             // 타당성 조사
        count += 1
        change −= coin
        if change = 0 then break  // 해답 조사
    else i += 1
if change = 0 then return count
else return -1
```

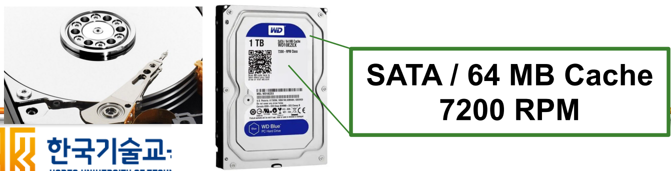

거스름돈 문제 (2/2)

- 현재 유통되고 있는 동전만을 가지고 탐욕적 알고리즘을 적용하면 이 알고리즘을 통해 최적해를 얻을 수 있음
- **예)** 액면가가 100원, 50원, 10원인 동전만 있다고 하자.
  - 210원에 대한 거스름: 100원, 100원, 10원 $\Longrightarrow$ 3개
- **예)** 액면가가 위 3개 외에 120원 짜리가 있다고 하자.
  - 210원에 대한 거스름: 120원, 50원, 10원짜리 4개 $\Longrightarrow$ 6개
    - 최적의 해가 아님
- 동전의 액면가에 따라 탐욕적 알고리즘을 사용하지 못할 수 있음
  - 이 문제는 동적 프로그래밍으로 해결할 수 있음
- 어떤 경우에 탐욕적 알고리즘으로 해결할 수 있나?



캐시 교체 문제 (1/3)

- 캐시: 작은 빠른 메모리
  - 실제 데이터는 이 보다 느린 메모리에 있음
  - 예) CPU cache: CPU register ⇔ CPU cache ⇔ main memory
  - 예) Disk cache: memory ⇔ disk cache ⇔ disk
- 보통 페이지 단위로 데이터 이동 (⇔)
- 데이터가 저장된 페이지가 캐시에 없으면 해당 페이지를 먼저 캐시로 옮겨와야 함
  - 페이지 부재(page fault)가 발생하였다고 함
- 캐시에 유지할 수 있는 페이지 수가 제한됨
  - 이것을 슬롯 수라 함
- 페이지 부재가 발생하였을 때 빈 슬롯이 없으면 부재한 페이지를 다른 페이지를 유지한 슬롯에 겹쳐 쓸 수밖에 없음
  - 이때 겹쳐 쓸 슬롯을 결정하는 알고리즘을 캐시 교체 알고리즘/정책이라 함



캐시 교체 문제 (2/3)

LeetCode 146

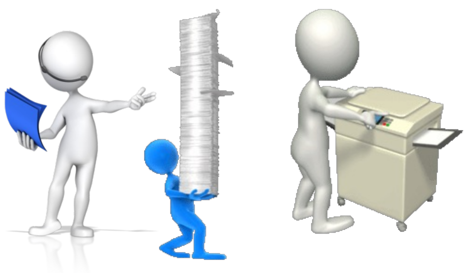
- 예) 캐시의 슬롯 수: 4
  - LRU(Least Recently Used) 정책
  - 요청: a b c d e f a b
    - 첫 4개의 요청에 의해 캐시는 [a b c d]를 유지하고 있음
    - e가 없으므로 가장 오래전에 사용한 a와 교체함: [b, c, d, e]
    - f가 없으므로 가장 오래전에 사용한 [c, d, e, f]
    - [d, e, f, a]
    - [e, f, a, b]
    - 이와 같은 순서로 하면 4개의 페이지 부재가 발생함
  - e에 대한 페이지 부재 시 c와 교체하고, f는 d와 교체하면?
- 어떤 교체 정책/방법이 가장 좋을까?

캐시 교체 문제 (3/3)

- Furthest-in-future 알고리즘 (탐욕적 알고리즘)
  - 가장 나중에 필요한 것을 교체함
  - 예) cache [a b c d e f], 요청 [g a b c e d a b b f]
    - g 때문에 하나는 교체해야 함. 가장 미래에 요구할 것이 f이기 때문에 f와 g를 교체함
  - 예) cache [a b c d e f], 요청 [g a b f c e d a b b] ⇒ [a b c g e f]
    - g 이후 캐시에 있는 a b c d e f에 대한 요청이 모두 있음. 그 중 가장 나중에 요청하는 것이 d임
- 쓸모 없는 알고리즘 ⇒ No
  - 실용적인 알고리즘에 대한 guideline 제공 (이상적인 벤치마크)

스케줄링 알고리즘

- 자원이 한 개
  - 예) 한 개의 CPU, 실행해야 하는 작업은 여러 개가 있음
  - 예) 한 개의 회의실, 여러 개의 회의를 진행해야 함
- 어떤 순서로?
  - 목적에 따라 다름
- 작업이 n개이면 가능한 작업 스케줄은 $n!$임
  - 전수조사하여 가장 최적의 스케줄은 찾는 것은 n이 조금만 커도…



시스템 내부 총 시간 최소화

- 문제 1) 시스템 내부 총 시간을 최소화하고 싶으면?
  - 시스템 내부 시간: 대기 시간 + 서비스 시간
    - 다른 말로 완료 시간(completion time)
    - 가정. 각 작업의 도착시간은 같음
  - 예) $l_1 = 1,\ l_2 = 2,\ l_3 = 3$
    - $j_1,\ j_2,\ j_3$ 순서로 실행: $c_1 = 1,\ c_2 = 3,\ c_3\ =\ 6$, 전체 시간: 10
    - $j_3,\ j_2,\ j_1$ 순서로 실행: $c_1 = 6,\ c_2 = 5,\ c_3\ =\ 3$, 전체 시간: 14
  - 길이가 짧은 순서로 하면 전체 시간을 최소화할 수 있음
    - 탐욕적 알고리즘
    - 증명) 모순에 의한 증명

가중 완료 총 시간 최소화 (1/6)

- 문제 2) 작업의 길이와 가중치가 주어졌을 때, $\sum_{i=1}^{n} w_i c_i$을 최소화
  - w_i: 작업의 가중치
    - 가중치가 클수록 우선순위가 높은 작업임
  - l_i: 작업의 길이
  - c_i: 작업의 완료 시간
  - 가정. 모든 작업의 도착시간은 같음
- $\sum_{i=1}^{n} w_i c_i$을 최소화의 의미는?
  - 시스템 내부 총 시간을 줄이면서 우선순위가 높은 것은 가급적 먼저 실행함
  - 가중치가 모두 같으면 시스템 내부 총 시간 최소화와 같은 문제
    - 작업 시간이 작은 순으로 실행
  - 작업 시간이 모두 같으면 우선순위가 높은 것을 먼저 실행하면 최솟값을 얻음

가중 완료 총 시간 최소화 (2/6)

- 예) $l_1 = 1,\ l_2 = 2,\ l_3 = 3, w_1 = 3,\ w_2 = 2,\ w_3 = 1$
  - 이 예는 작업 시간이 작은 것이 우선 순위가 높음
  - $j_1,\ j_2,\ j_3$ 순서로 실행 ⟹ 3+6+6 = 15
- 예) $l_1 = 1,\ l_2 = 2,\ l_3 = 3, w_1 = 2,\ w_2 = 6,\ w_3 = 3$
  - $j_1,\ j_2,\ j_3$ 순서로 실행 ⟹ 2+18+18 = 38
  - $j_2,\ j_1,\ j_3$ 순서로 실행 ⟹ 12+6+18 = 36
  - $j_2,\ j_3,\ j_1$ 순서로 실행 ⟹ 12+15+12 = 39
- 탐욕적 기법의 알고리즘을 개발하기 위해서는 작업을 선택할 기준이 있어야 함
  - 어떤 기준을 사용하면 최적해를 얻을 수 있을까?
    - 작업 시간이 작을수록
    - 우선 순위는 높을수록
    - 예) $f(w_i, l_i) = w_i - l_i,\ f(w_i, l_i) = w_i / l_i$

가중 완료 총 시간 최소화 (3/6)

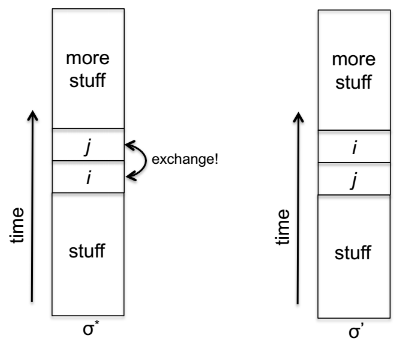
- $f(w_i, l_i) = w_i - l_i$
- $f(w_i, l_i) = w_i / l_i$
- 예) 두 작업을 생각하여 보자
  - (3, 5), (1, 2)
    - 방법 1. -2, -1
      - $c_2 = 2,\ c_1 = 7$, 2+21=23
    - 방법 2. 3/5, 1/2
      - $c_1 = 5,\ c_2 = 7$, 15+7=22
  - 첫 번째 방법은 문제가 있다는 것을 알 수 있음

가중 완료 총 시간 최소화 (4/6)

- $f(w_i, l_i) = w_i/l_i$을 사용하면 항상 최솟값을 얻을 수 있음
- 증명) exchange argument를 이용한 모순 증명
  - 가정 1. 모든 작업마다 점수가 모두 다름
  - 가정 2. 점수에 따라 가장 점수가 큰 값 순으로 색인이 부여되었다고 하자. 즉, $\frac{w_1}{l_1} > \frac{w_2}{l_2} > \cdots > \frac{w_n}{l_n}$
  - σ = greedy schedule, σ^* = optimal schedule
    - σ = 1, 2, 3, …, n
    - $\sigma = \ \sigma^*$ 가 아니면 σ^*에는 $i > j$인 i가 j보다 먼저 실행된 것이 있어야 함

가중 완료 총 시간 최소화 (5/6)

- 작업 i와 작업 j의 순서를 바꾸면 각 작업의 종료 시간에 주는 영향은?
  - 작업 $k(\neq i, j)$에 어떤 영향? 영향 없음
  - 작업 i에 어떤 영향? c_i가 l_j만큼 증가
  - 작업 j에 어떤 영향? l_i만큼 감소
- 전체 비용은?
  - 작업 i: $w_i(\triangle + l_j)$, 작업 j: $w_j(\triangle' - l_i)$
    - $\sigma^{new} = \sigma^* + w_i l_j - w_j l_i$
    - $i > j$ 에 대해 $\frac{w_i}{l_i} < \frac{w_j}{l_j}$이므로 $w_i l_j < w_j l_i$임
    - $\sigma^{new} < \sigma^*$이므로 모순



(a) Before exchange (b) After exchange

가중 완료 총 시간 최소화 (6/6)

- Tie가 있는 경우
  - 가정. $\frac{w_1}{l_1} \geq \frac{w_2}{l_2} \geq \cdots \geq \frac{w_n}{l_n}$
- 전체 비용은?
  - 작업 i: $w_i(\triangle + l_j)$, 작업 j: $w_i(\triangle' - l_i)$
    - $\sigma^{new} = \sigma^* + w_i l_j - w_j l_i$
    - $i > j$ 에 대해 $\frac{w_i}{l_i} \leq \frac{w_j}{l_j}$이므로 $w_i l_j \leq w_j l_i$임
    - $\sigma^{new} \leq \sigma^*$이므로 모순

마감 시간이 있는 스케줄 (1/3)

- 문제 3) 마감 시간을 고려하여 이익을 최대화
  - 작업 길이는 모두 1이라고 가정함
  - 작업이 마감 시간 전이나 마감 시간에 시작되면 이익을 얻을 수 있음
  - 모든 작업을 스케줄해야 하는 것은 아님
  - 작업이 마감 시간 이후로 처리된 스케줄은 고려할 필요가 없음
- 예) $d_1 = 2,\ d_2 = 1,\ d_3 = 2,\ d_4 = 1,$
  $w_1 = 30,\ w_2 = 35,\ w_3 = 25,\ w_4 = 40$
  - $j_1,\ j_3$: 30+25 = 55
  - $j_2,\ j_1$: 35+30 = 65
  - $j_2,\ j_3$: 35+25 = 60
  - $j_3,\ j_1$: 25+30 = 55
  - $j_4,\ j_1$: 40+30 = 70
  - $j_4,\ j_3$: 40+25 = 65

> - 모든 경우를 고려하는 것은 계승시간
> - 가장 이익이 높은 것은 포함되지만
>   두 번째 높은 것은 포함되지 않음
> - 탐욕적 방법: 어떤 것을 기준으로?
> - 이익 순으로 정렬한 후 가장 높은 것부터
>   차례로 검사하여 포함할 수 있으면
>   포함하는 방법이 가능할 것 같음
> - 포함할 수 있는지는 어떻게 검사?

마감 시간이 있는 스케줄 (2/3)

- 타당한 순서(feasible sequence): 주어진 순서로 스케줄링 가능한 순서
  - 예) [4, 1]은 타당한 순서, [1, 4]는 타당한 순서가 아님
- 타당한 집합: 타당한 순서로 스케줄링 가능한 작업의 집합
  - 예) {1, 4}는 타당한 집합, {2, 4}는 타당한 집합이 아님
- 작업 집합 S가 타당하기 위한 필요충분조건은 마감 시간을 기준으로 오름차순으로 스케줄링하였을 때 타당한 스케줄이어야 함
  - 증명)
    - S가 타당한 집합이면 이 작업들에 대한 타당한 순서가 존재함
    - 이 순서가 [..., x, y, ...]이고 x의 마감시간이 y보다 크다고 하자.
    - 이 경우 두 작업의 순서를 바꾸어도 이 순서는 여전히 타당함
      - y는 x보다 마감 시간이 작으니 x가 실행된 위치에서 실행 가능
      - x는 y보다 마감 시간이 크므로 y가 실행된 위치에서 실행 가능
    - 이 과정을 반복하면 오름차순으로 스케줄링된 타당한 스케줄을 얻을 수 있음

$d_1 = 2,\ d_2 = 1,$
$d_3 = 2,\ d_4 = 1,$
$w_1 = 30,\ w_2 = 35,$
$w_3 = 25,\ w_4 = 40$

- 포함할 수 있는지는 검사하는 방법은 오름차순으로 정렬하여 스케줄링이 가능한지 검사

마감 시간이 있는 스케줄 (3/3)

- 입력. 작업 이익을 기준으로 내림차순으로 정렬된 작업 목록
- 알고리즘

시간복잡도: $O(n \log n)$

schedule(deadline[])

```
S := [1]
for j := 2 to n do // O(n)
    K := add job j to S
    if K is feasible then S := K // O(n)
return S
```

시간복잡도: $O(n^2)$

job	deadline	profit
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

S=[1]
K=[2,1] ⇒ feasible ⇒ S=[1, 2]
K=[2,3,1] ⇒ not feasible ⇒ reject
K=[2,4,1] ⇒ feasible ⇒ S=[1, 2, 4]
K=[2,5,4,1] ⇒ not feasible ⇒ reject
K=[2,4,1,6] ⇒ not feasible ⇒ reject
K=[2,7,4,1] ⇒ not feasible ⇒ reject

K는 단일 연결구조로 생각