제4장 그리디 알고리즘 (2)

과 목 명 정 보 처 리 알 고 리 즘

담당교수 김 성 훈

경북대학교 과학기술대학 소프트웨어학과

이 장에서 배울 내용

1. 그리디 알고리즘 기본개념
2. 동전 거스름돈
3. 최소 신장 트리(Minimum Spanning Tree)
4. 최단 경로 찾기(Shortest Path Problem)
5. 부분배낭문제(Fractional Knapsack Problem)
6. 집합 커버 문제(Set Cover Problem)
7. 작업 스케줄링 문제(Job Scheduling)
8. 허프만 압축(Huffman Coding) (skip)

4.4 부분 배낭 문제

- 배낭 (Knapsack) 문제:
  - n개의 물건이 있고,
    각 물건은 무게와 가치를 가지고 있으며,
    배낭이 한정된 무게의 물건들을 담을 수 있을 때,
    최대의 가치를 갖도록 배낭에 넣을 물건들을 정하는 문제

- 원래 배낭 문제는 물건을 통째로 배낭에 넣어야 되지만,
  부분 배낭(Fractional Knapsack)문제는 물건을 부분적으로 담는 것을 허용
  - 부분 배낭 문제에서는 물건을 부분적으로 배낭에 담을 수 있으므로,
    최적해를 위해서 '욕심을 내어' 단위 무게 당 가장 값나가는 물건을 배낭에 넣고,
    계속해서 그 다음으로 값나가는 물건을 넣는다.
  - 그런데 만일 그 다음으로 값나가는 물건을 '통째로' 배낭에 넣을 수 없게 되면,
    배낭에 넣을 수 있을 만큼만 물건을 부분적으로 배낭에 담는다.

부분 배낭 알고리즘

FractionalKnapsack

입력: n개의 물건, 각 물건의 무게와 가치, 배낭의 용량 C

출력: 배낭에 담은 물건 리스트 L과 배낭에 담은 물건의 가치 합 v

1. 각 물건에 대해 단위 무게 당 가치를 계산한다.
2. 물건들을 단위 무게 당 가치를 기준으로 내림차순으로 정렬하고, 정렬된 물건 리스트를 S라고 하자.
3. L=∅, w=0, v=0

   // L은 배낭에 담을 물건 리스트,
   // w는 배낭에 담긴 물건들의 무게의 합,
   // v는 배낭에 담긴 물건들의 가치의 합이다.
4. S에서 단위 무게 당 가치가 가장 큰 물건 x를 가져온다.

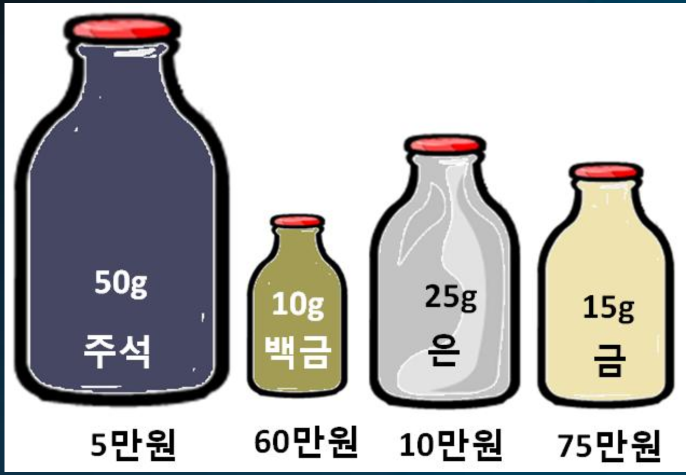
부분 배낭 알고리즘(2)

```
5. while ( (w+x의 무게) ≤ C ) {
6.          x를 L에 추가시킨다.
7.          w = w + x의 무게
8.          v = v + x의 가치
9.          x를 S에서 제거한다.
10.         S에서 단위 무게 당 가치가 가장 큰 물건 x를 가져온다.
   }
11. if ((C - w) > 0) { // 배낭에 물건을 부분적으로 담을 여유가 있으면
12.         물건 x를 (C-w)만큼만 L에 추가한다.
13.         v = v +(C- w)만큼의 x의 가치
    }
14. return L, v
```

부분 배낭 알고리즘의 수행 과정

- 4개의 금속 분말이 다음의 그림과 같이 있다.
  배낭의 최대용량이 40그램일 때, FractionalKnapsack알고리즘의 수행 과정
- Line 1~2의 결과: S=[백금, 금, 은, 주석]

물건	단위 그램당 가치
백금	6만원
금	5만원
은	4천원
주석	1천원



부분 배낭 알고리즘의 수행 과정(2)

- Line 3: L=∅, w=0, v=0로 각각 초기화한다.
- Line 4: S=[백금, 금, 은, 주석]로부터 **백금**을 가져온다.
- Line 5: while-루프의 조건 ((w+백금의 무게) ≤ C) = ((0+10)<40)이 ‘참’이다.
- Line 6: 백금을 배낭 L에 추가시킨다. 즉, L=[백금]이 된다.
- Line 7: w = w(백금의 무게) = 0+10g = 10g
- Line 8: v = v(백금의 가치) = 0+60만원 = 60만원
- Line 9: S에서 백금을 제거한다. S=[금, 은, 주석]
- Line 10: S에서 **금**을 가져온다.

부분 배낭 알고리즘의 수행 과정(3)

- Line 5: while-루프의 조건 ((w+금의 무게) ≤ C) = ((10+15)<40)이 '참'이다.
- Line 6: 금을 배낭 L에 추가시킨다. L=[백금, 금]
- Line 7: w = w+(금의 무게) = 10g+15g = 25g
- Line 8: v = v+(금의 가치) = 60만원+75만원 = 135만원
- Line 9: S에서 금을 제거한다. S=[은, 주석]
- Line 10: S에서 **은**을 가져온다.
- Line 5: while-루프의 조건 ((w+은의 무게) ≤ C) = ((25+25)<40)이 '거짓'이므로 루프를 종료한다.

부분 배낭 알고리즘의 수행 과정(4)

- Line 11: if-조건 ((C-w) >0)이 '참'이다. 즉, 40-25 = 15 > 0이기 때문이다.
- Line 12: 따라서 **은을 C-w=(40-25)=15g만큼만** 배낭 L에 추가시킨다.
- Line 13: v = v+(15g x 4천원/g) = 135만원+6만원 = 141만원
- Line 14: 배낭 L=[**백금 10g, 금 15g, 은 15g**]과

  가치의 합 v = 141만원을 리턴한다.

시간복잡도

- Line 1: n개의 물건 각각의 단위 무게 당 가치를 계산하는 데는 O(n) 시간 걸리고, line 2에서 물건의 단위 무게 당 가치에 대해서 내림차순으로 정렬하기 위해 O(nlogn) 시간이 걸린다.
- Line 5~10의 while-루프의 수행은 n번을 넘지 않으며, 루프 내부의 수행은 O(1) 시간이 걸린다. 또한 line 11~14도 각각 O(1) 시간 걸린다.
- 따라서 알고리즘의 시간복잡도는

  O(n)+O(nlogn)+nxO(1)+O(1) = O(nlogn)이다.

응 용

- 0-1 배낭 문제는 최소의 비용으로 자원을 할당하는 문제로서,
  조합론, 계산이론, 암호학, 응용수학 분야에서 기본적인 문제로 다루어진다.

- 응용 사례로는
  '버리는 부분 최소화시키는' 원자재 자르기(Raw Material Cutting),

- 자산투자 및 금융 포트폴리오 (Financial Portfolio)에서의 최선의 선택

- Merkle–Hellman 배낭 암호 시스템의 키(Key)생성에도 활용된다.

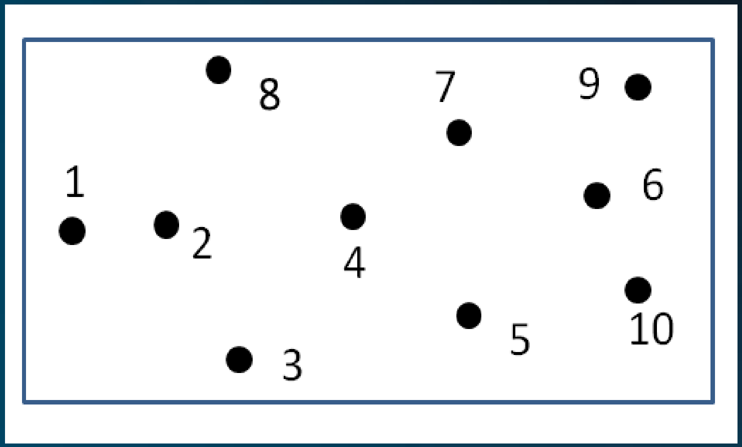
4.5 집합 커버 문제

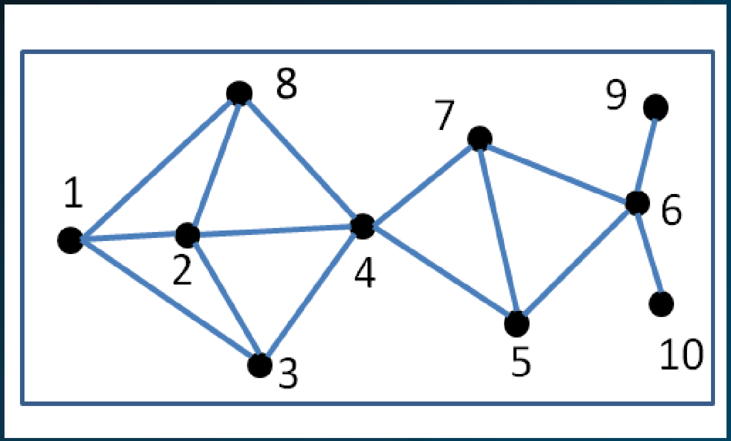
- n개의 원소를 가진 집합인 U가 있고,
  U의 부분 집합들을 원소로 하는 집합 F가 주어질 때,
  F의 원소들인 집합들 중에서
  어떤 집합들을 선택하여 합집합 하면 U와 같게 되는가?

- 집합커버(cover)문제는 F에서 선택하는 집합들의 수를 최소화하는 문제이다.

예제: 신도시 계획 학교 배치

- 10개의 마을이 신도시에 있다.
- 이때 아래의 2가지 조건이 만족되도록 학교의 위치를 선정하여야 한다고 가정하자.
  - 학교는 마을에 위치해야 한다.
  - 등교 거리는 걸어서 15분 이내이어야 한다.

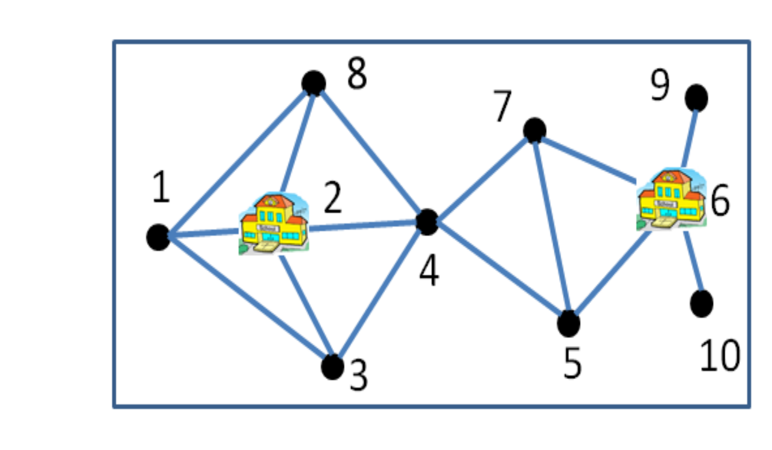




등교 거리가 15분 이내인 마을 간의 관계

예제: 신도시 계획 학교 배치(2)

- 어느 마을에 학교를 신설해야 학교의 수가 최소로 되는가?
- 2번 마을에 학교를 만들면 마을 1, 2, 3, 4, 8의 학생들이 15분 이내에 등교할 수 있고 (즉, 마을 1, 2, 3, 4, 8이 '커버'되고),
- 6번 마을에 학교를 만들면 마을 5, 6, 7, 9, 10이 커버된다.
- 즉, 2번과 6번이 최소이다.



예제: 신도시 계획 학교 배치(3)

- 신도시 계획 문제를 집합 커버 문제로 변환:

  여기서 S_i는 마을 i에 학교를 배치했을 때 커버되는 마을의 집합이다.

  U={1, 2, 3, 4, 5, 6, 7, 8, 9, 10} // 신도시의 마을 10개

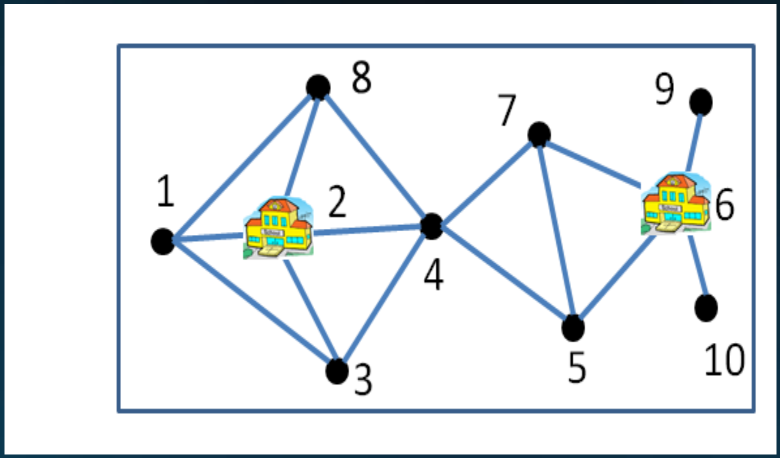
  $F=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\}$

  $S_1=\{1, 2, 3, 8\}$
  $S_2=\{1, 2, 3, 4, 8\}$
  $S_3=\{1, 2, 3, 4\}$
  $S_4=\{2, 3, 4, 5, 7, 8\}$
  $S_5=\{4, 5, 6, 7\}$
  $S_6=\{5, 6, 7, 9, 10\}$
  $S_7=\{4, 5, 6, 7\}$
  $S_8=\{1, 2, 4, 8\}$
  $S_9=\{6, 9\}$
  $S_{10}=\{6, 10\}$

- S_i 집합들 중에서 어떤 집합들을 선택하여야 그들의 합집합이 U와 같은가? 단, 선택된 집합의 수는 최소이어야 한다.

예제: 신도시 계획 학교 배치(4)

- 이 문제의 답은
  $S_2 \cup S_6 = \{1, 2, 3, 4, 8\} \cup \{5, 6, 7, 9, 10\} = \{1, 2, 3, 4, 5, 6, 7, 8\} = U$이다.



집합 커버 문제의 아이디어

- 집합 커버 문제의 최적해는 어떻게 찾아야 할까?
  - F에 n개의 집합들이 있다고 가정해보자.
  - 가장 단순한 방법은 F에 있는 집합들의 모든 조합을 1개씩 합집합 하여 U가 되는지 확인하고, U가 되는 조합의 집합 수가 최소인 것을 찾는 것이다.
- 예를 들면, $F=\{S_1, S_2, S_3\}$일 경우,
  모든 조합이란, S_1, S_2, S_3, $S_1 \cup S_2$, $S_1 \cup S_3$, $S_2 \cup S_3$, $S_1 \cup S_2 \cup S_3$이다.
  - 집합이 1개인 경우 3개 = ${}_3C_1$,
  - 집합이 2개인 경우 3개 = ${}_3C_2$,
  - 집합이 3개인 경우 1개 = ${}_3C_3$이다.
  - 총합은 3+3+1= 7 = 2^3-1 개이다.

집합 커버 문제의 아이디어(2)

- n개의 원소가 있으면 (2^n-1)개를 다 검사하여야 하고, n이 커지면 최적해를 찾는 것은 실질적으로 불가능하다.

- 이를 극복하기 위해서는 최적해를 찾는 대신에 최적해에 근접한 근사해 (approximation solution)를 찾는 것이다.

집합 커버 알고리즘

SetCover

입력: U, $F=\{S_i\}$, $i=1,\cdots,n$

출력: 집합 커버 C

1. C=∅
2. while (U≠∅) do {
3. U의 원소들을 <u>가장 많이 포함하고 있는</u> 집합 S_i를 F에서 선택한다.
4. $U=U-S_i$
5. S_i를 F에서 제거하고, S_i를 C에 추가한다.

   }

6. return C

집합 커버 알고리즘의 수행 과정

$U=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

$F=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\}$

$S_1=\{1, 2, 3, 8\}$

$S_2=\{1, 2, 3, 4, 8\}$

$S_3=\{1, 2, 3, 4\}$

$S_4=\{2, 3, 4, 5, 7, 8\}$

$S_5=\{4, 5, 6, 7\}$

$S_6=\{5, 6, 7, 9, 10\}$

$S_7=\{4, 5, 6, 7\}$

$S_8=\{1, 2, 4, 8\}$

$S_9=\{6, 9\}$

$S_{10}=\{6, 10\}$

집합 커버 알고리즘의 수행 과정(2)

- Line 1: C=∅로 초기화
- Line 2: while-조건 (U≠∅)=({1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ≠ ∅)이 '참'이다.
- Line 3: U의 원소를 가장 많이 커버하는 집합인 S_4={2, 3, 4, 5, 7, 8}을 F에서 선택한다.
- Line 4: U = U - S_4 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} - {2, 3, 4, 5, 7, 8}
= {1, 6, 9, 10}
- Line 5: S_4를 F에서 제거하고,
즉, F ={ S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}} - {S_4}
= {S_1, S_2, S_3, S_5, S_6, S_7, S_8, S_9, S_{10}}가 되고, S_4를 C에 추가한다.
즉, C = {S_4}이다.

집합 커버 알고리즘의 수행 과정(3)

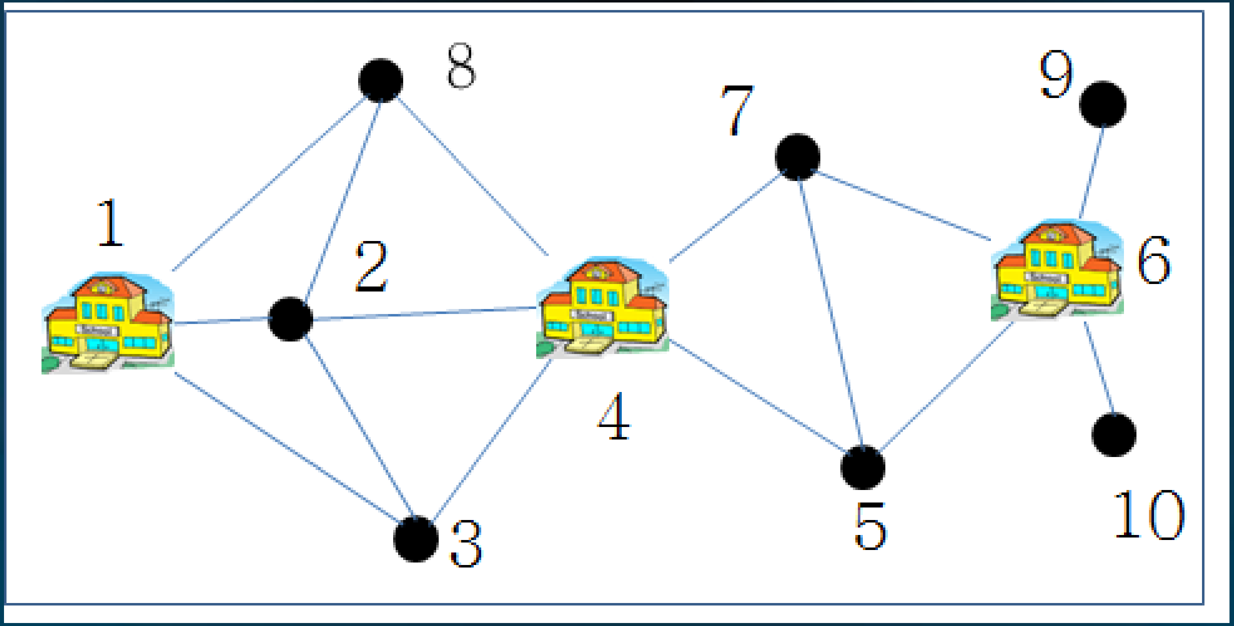
- Line 2: while-조건 ($U \neq \emptyset$) = ($\{1, 6, 9, 10\} \neq \emptyset$)이 '참'이다.
- Line 3: U의 원소를 가장 많이 커버하는 집합인 $S_6=\{5, 6, 7, 9, 10\}$을 F에서 선택한다.
- Line 4: $U = U - S_4 = \{1, 6, 9, 10\} - \{5, 6, 7, 9, 10\} = \{1\}$
- Line 5: S_6을 F에서 제거하고,
  즉, $F=\{S_1, S_2, S_3, S_5, S_6, S_7, S_8, S_9, S_{10}\} - \{S_6\}$
  $= \{S_1, S_2, S_3, S_5, S_7, S_8, S_9, S_{10}\}$이 되고, S_6을 C에 추가한다.
  즉, $C = \{S_4, S_6\}$이다.

집합 커버 알고리즘의 수행 과정(4)

- Line 2: while-조건 (U≠Ø) = ({1}≠Ø)이 '참'이다.
- Line 3.: U의 원소를 가장 많이 커버하는 집합인 S_1={1, 2, 3, 8}을
  F에서 선택한다.
  S_1 대신에 S_2, S_3, S_8 중에서 어느 하나를 선택해도 무방하다.
- Line 4: U = U - S_1 = {1} - {1, 2, 3, 8}=Ø
- Line 5: S_1을 F에서 제거하고, 즉, F={S_1, S_2, S_3, S_5, S_6, S_7, S_8, S_9, S_{10}} - {S_1} ={S_2, S_3, S_5, S_7, S_8, S_9, S_{10}}이 되고, S_1을 C에 추가한다.
  즉, C = {S_1, S_4, S_6}이다.
- Line 2: while-조건 (U≠Ø) = (Ø≠Ø)이 '거짓'이므로, 루프를 끝낸다.
- Line 6: C={S_1, S_4, S_6}을 리턴한다.

집합 커버 알고리즘의 수행 과정(5)

- SetCover 알고리즘의 최종해



시간복잡도

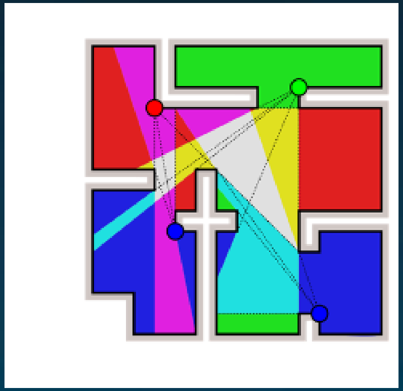
- 먼저 while-루프가 수행되는 횟수는 최대 n번이다.
  왜냐하면 루프가 1번 수행될 때마다 집합 U의 원소 1개씩만 커버된다면,
  최악의 경우 루프가 n번 수행되어야 하기 때문이다.
- 루프가 1번 수행될 때의 시간복잡도를 살펴보자.
- Line 2의 while-루프 조건 (U≠∅)을 검사는 O(1) 시간이 걸린다.
  왜냐하면 U의 현재 원소 수를 위한 변수를 두고 그 값이 0인지를 검사하면 되기 때문이다.

시간복잡도(2)

- Line 3: U의 원소들을 가장 많이 포함하고 있는 집합 S를 찾으려면, 현재 남아있는 S_i들 각각을 U와 비교하여야 한다. 따라서 S_i들의 수는 최대 n이고, 각 S_i와 U의 비교는 O(n) 시간이 걸리므로, line 3은 $O(n^2)$ 시간이 걸린다.
- Line 4: 집합 U에서 집합 S_i의 원소를 제거하는 것이므로 O(n) 시간이 걸린다.
- Line 5: S_i를 F에서 제거하고, S_i를 C에 추가하는 것이므로 O(1) 시간이 걸린다.
- 따라서 루프 1회의 시간복잡도는 $O(1)+O(n^2)+O(n)+O(1) = O(n^2)$이다.
- 따라서 시간복잡도는 $O(n) \times O(n^2) = O(n^3)$이다.

응 용

- 도시 계획 (City Planning)에서 공공 기관 배치하기
- 경비 시스템: 미술관, 박물관, 기타 철저한 경비가 요구되는 장소 (Art Gallery 문제)의 CCTV 카메라의 최적 배치
- 컴퓨터 바이러스 찾기: 알려진 바이러스들을 '커버'하는 부분 스트링의 집합 - IBM에서 5,000개의 알려진 바이러스들에서 9,000개의 부분 스트링을 추출하였고, 이 부분 스트링의 집합 커버를 찾았는데, 총 180개의 부분 스트링이었다. 이 180개로 컴퓨터 바이러스의 존재를 확인하는데 성공하였다.



응 용(2)

- 대기업의 구매 업체 선정: 미국의 자동차 회사인 GM은 부품 업체 선정에 있어서 각 업체가 제시하는 여러 종류의 부품들과 가격에 대해, 최소의 비용으로 구입하려고 하는 부품들을 모두 '커버'하는 업체를 찾기 위해 집합 문제의 해를 사용하였다.
- 기업의 경력 직원 고용: 예를 들어, 어느 IT 회사에서 경력 직원들을 고용하는데, 회사에서 필요로 하는 기술은 알고리즘, 컴파일러, 앱 (App) 개발, 게임 엔진, 3D 그래픽스, 소셜 네트워크 서비스, 모바일 컴퓨팅, 네트워크, 보안이고, 지원자들은 여러 개의 기술을 보유하고 있다. 이 회사가 모든 기술을 커버하는 최소 인원을 찾으려면, 집합 문제의 해를 사용하면 된다.
- 그 외에도 비행기 조종사 스케줄링 (Flight Crew Scheduling), 조립 라인 균형화 (Assembly Line Balancing), 정보 검색 (Information Retrieval) 등에 활용된다.

BREAK TIME

4.6 작업 스케줄링

- 기계에서 수행되는 n개의 작업 $t_1, t_2, \cdots, t_n$이 있고,
  각 작업은 시작시간과 종료시간이 있다.
  - 작업 스케줄링 (Task Scheduling) 문제는 작업의 수행 시간이 중복되지 않도록 모든 작업을
    가장 적은 수의 기계에 배정하는 문제이다.
  - 작업 스케줄링 문제는 학술대회에서 발표자들을 강의실에 배정하는 문제와 같다.
    - 발표= '작업', 강의실= '기계'
- 작업 스케줄링 문제에 주어진 문제 요소
  - 작업의 수
  - 각 작업의 시작시간과 종료시간
  - 작업의 시작시간과 종료시간은 정해져 있으므로 작업의 길이도 주어진 것이다.

  – 여기서 작업의 수는 입력의 크기이므로 알고리즘을 고안하기 위해 고려되어야 하는 직접적인 요소는 아니다.

아이디어

- 그렇다면, 시작시간, 종료시간, 작업 길이에 대해
다음과 같은 그리디 알고리즘들을 생각해볼 수 있다.
  1. 빠른 시작시간 작업 우선 (Earliest start time first) 배정
  2. 빠른 종료시간 작업 우선 (Earliest finish time first) 배정
  3. 짧은 작업 우선 (Shortest job first) 배정
  4. 긴 작업 우선 (Longest job first) 배정
- 위의 4가지 중 첫 번째 알고리즘을 제외하고 나머지 3가지는 항상 최적해를 찾지 못한다.

작업 배정 알고리즘

JobScheduling

입력: n개의 작업 $t_1, t_2, \cdots, t_n$

출력: 각 기계에 배정된 작업 순서

1. 시작시간의 오름차순으로 정렬한 작업 리스트: L
2. while (L ≠ Ø) {
3. 	L에서 가장 이른 시작시간 작업 t_i를 가져온다.
4. 	if (t_i를 수행할 기계가 있으면)
5. 		t_i를 수행할 수 있는 기계에 배정한다.
6. 	else
7. 		새로운 기계에 t_i를 배정한다.
8. 	t_i를 L에서 제거한다.

    }

9. return 각 기계에 배정된 작업 순서

JobScheduling알고리즘의 수행 과정

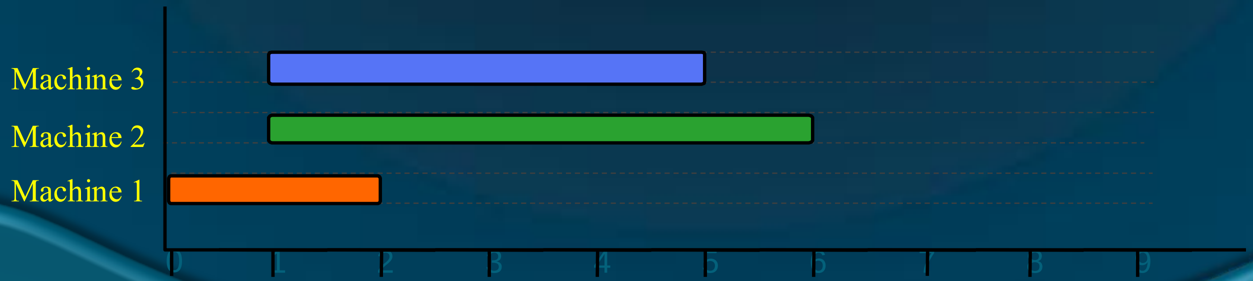
- t_1=[7,8], t_2=[3,7], t_3=[1,5], t_4=[5,9], t_5=[0,2], t_6=[6,8], t_7=[1,6], 단, [s,f]에서, s는 작업의 시작시간이고, f는 작업의 종료시간이다.
- Line 1: 시작시간의 오름차순으로 정렬한다.
  따라서 L = {[0,2], [1,6], [1,5], [3,7], [5,9], [6,8], [7,8]}이다.
- 다음은 line 2~8까지의 while-루프가 수행되면서,
  각 작업이 적절한 기계에 배정되는 것을 차례로 보이고 있다.

JobScheduling알고리즘의 수행 과정(2)

[0,2]

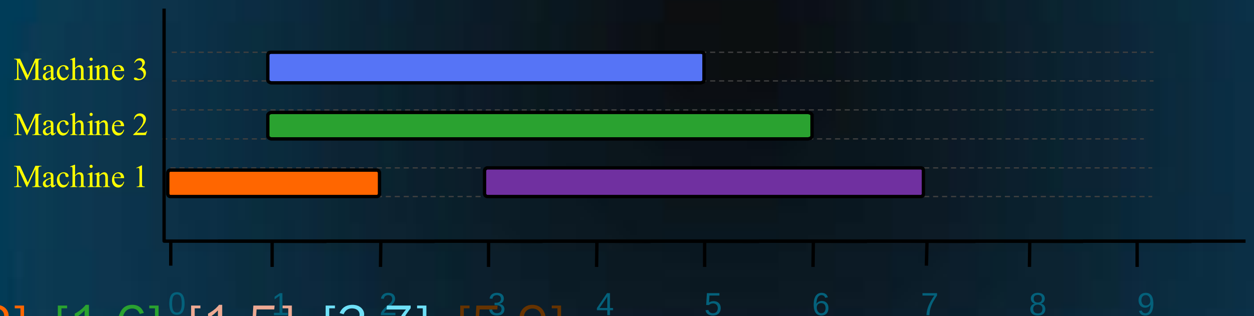
[0,2], [1,6]

[0,2], [1,6], [1,5]



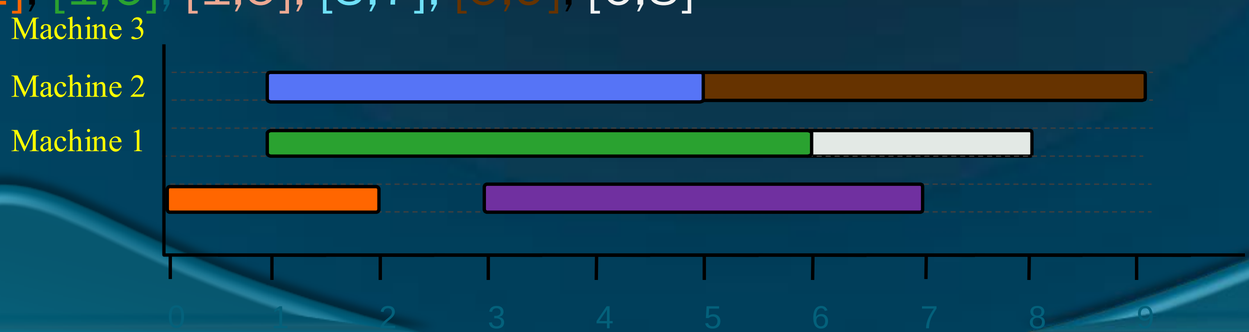
JobScheduling알고리즘의 수행 과정(3)

[0,2], [1,6], [1,5], [3,7]



[0,2], [1,6], [1,5], [3,7], [5,9]

[0,2], [1,6], [1,5], [3,7], [5,9], [6,8]



JobScheduling알고리즘의 수행 과정(4)

[0,2], [1,6], [1,5], [3,7], [5,9], [6,8], [7,8]

시간복잡도

- Line 1에서 n개의 작업을 정렬하는데 O(nlogn) 시간이 걸리고,
- while-루프에서는 작업을 L에서 가져다가 수행 가능한 기계를 찾아서 배정하므로 O(m) 시간이 걸린다. 단, m은 사용된 기계의 수이다.
- while-루프가 수행된 총 횟수는 n번이므로,
  line 2~9까지는 O(m)xn = O(mn) 시간이 걸린다.
- 따라서 JobScheduling 알고리즘의 시간복잡도는
  O(nlogn)+O(mn)이다.

응 용

- 비즈니스 프로세싱
- 공장 생산 공정
- 강의실/세미나 룸 배정
- 컴퓨터 태스크 스케줄링 등

4.7 허프만 압축

- 파일의 각 문자가 8 bit 아스키 (ASCII) 코드로 저장되면,
  그 파일의 bit 수는 8x(파일의 문자 수)이다.
- 이와 같이 파일의 각 문자는 일반적으로 고정된 크기의 코드로 표현된다.
  - 이러한 고정된 크기의 코드로 구성된 파일을 저장하거나 전송할 때 파일의 크기를 줄이고, 필요시 원래의 파일로 변환할 수 있으면, 메모리 공간을 효율적으로 사용할 수 있고, 파일 전송 시간을 단축시킬 수 있다.
- 주어진 파일의 크기를 줄이는 방법을 파일 압축 (file compression)이라고 한다.

아이디어

- 허프만 (Huffman) 압축은 파일에
  빈번히 나타나는 문자에는 짧은 이진 코드를 할당하고,
  드물게 나타나는 문자에는 긴 이진 코드를 할당한다.

- 허프만 압축 방법으로 변환시킨 문자 코드들 사이에는
  접두부 특성 (prefix property)이 존재한다.
  - 이는 각 문자에 할당된 이진 코드는 다른 문자에 할당된 이진 코드의 접두부 (prefix)에는 나타나지 않는다는 것을 의미한다.
  - 즉, 문자 'a'에 할당된 코드가 '101'이라면,
    모든 다른 문자의 코드에는 '101'로 시작되는 코드가 없다.
    또한 (대부분의 많은 경우에 있어서) '1'이나 '10'으로도 시작되지 않는다.
    (교과서 수정 요망)

아이디어(2)

- 접두부 특성의 장점은 코드와 코드 사이를 구분할 특별한 코드가 필요 없다.
  - 예를 들어, 101#10#1#111#0#…에서 '#'가 인접한 코드를 구분 짓고 있는데, 허프만 압축에서는 이러한 특별한 코드 없이 파일을 압축하고 해제할 수 있다.
- 허프만 압축은 입력 파일에 대해 각 문자의 출현 빈도수(문자가 파일에 나타나는 횟수)에 기반을 둔 이진 트리를 만들어서,
  각 문자에 이진 코드를 할당한다.
  - 이러한 이진 코드를 허프만 코드라고 한다.

허프만 코드 알고리즘

HuffmanCoding

입력: 입력 파일의 n개의 문자에 대한 각각의 빈도수

출력: 허프만 트리

1. 각 문자 당 노드를 만들고, 그 문자의 빈도수를 노드에 저장
2. n개의 노드의 빈도수에 대해 우선순위 큐 Q를 만든다.
3. while (Q에 있는 노드 수 ≥ 2) {
4. 빈도수가 가장 작은 2개의 노드 (A와 B)를 Q에서 제거
5. 새 노드 N을 만들고, A와 B를 N의 자식 노드로 만든다.
6. N의 빈도수 = A의 빈도수 + B의 빈도수
7. 노드 N을 Q에 삽입한다.

   }

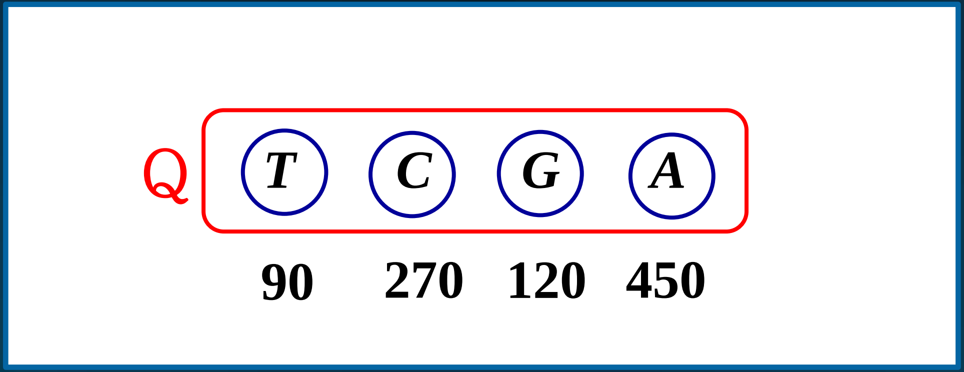
8. return Q // 허프만 트리의 루트를 리턴하는 것이다.

HuffmanCoding 알고리즘의 수행 과정

- HuffmanCoding 알고리즘의 수행 과정
- 입력 파일은 4개의 문자로 되어 있고, 각 문자의 빈도수는 다음과 같다.

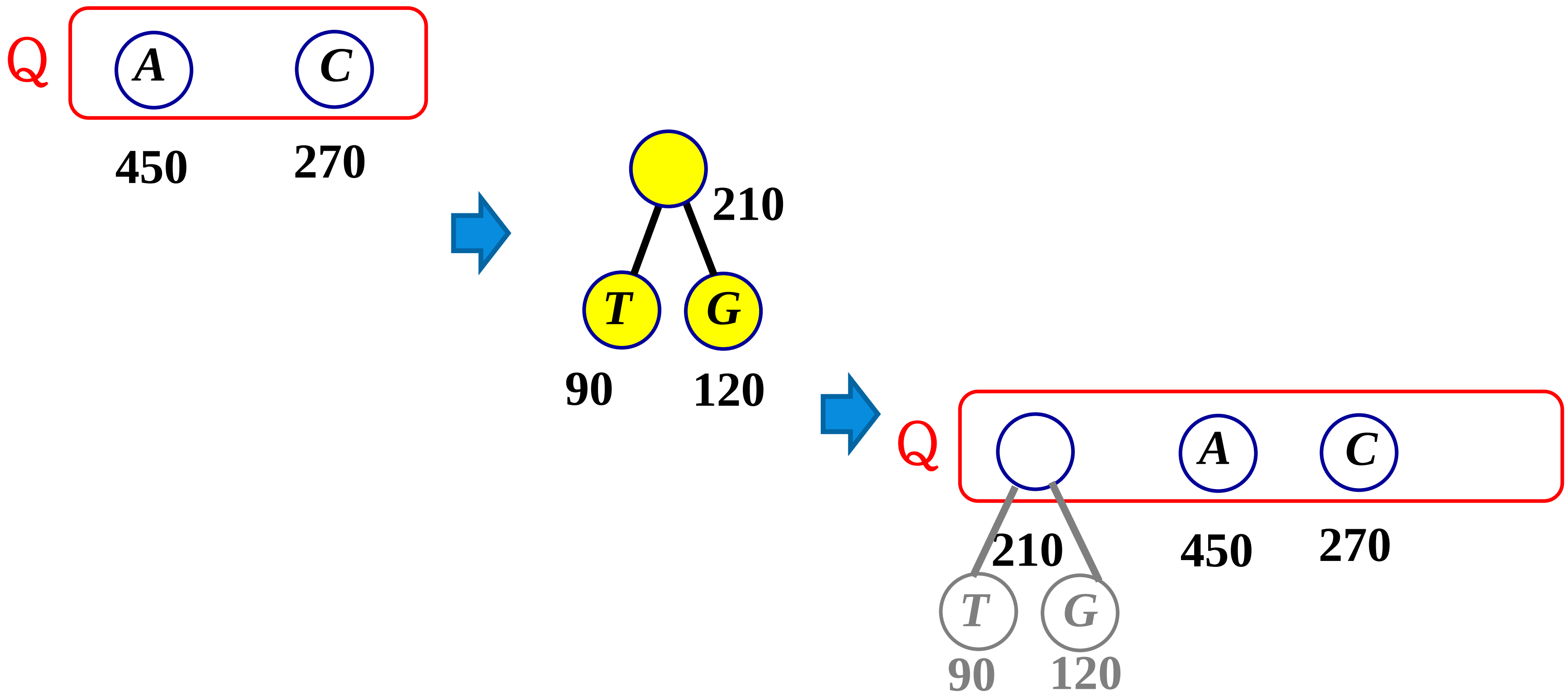
A: 450 T: 90 G: 120 C: 270

- Line 2를 수행한 후의 Q:



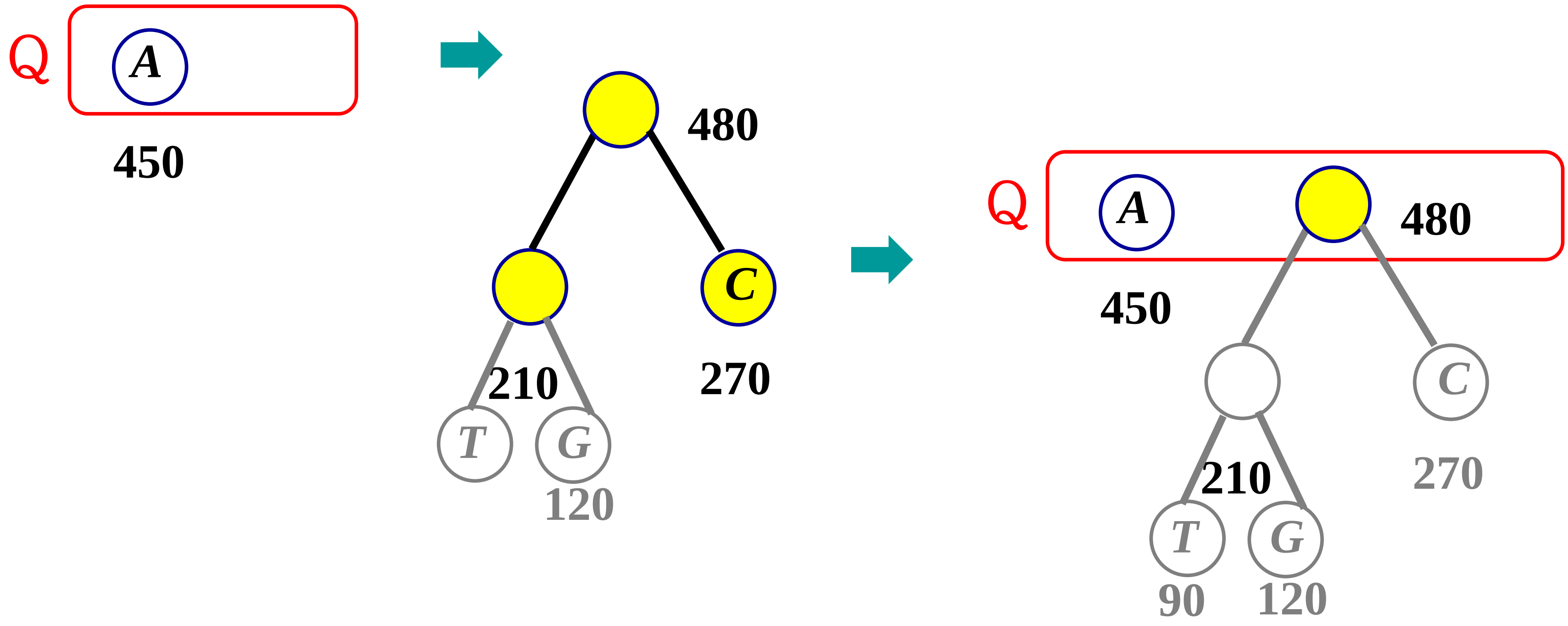
HuffmanCoding 알고리즘의 수행 과정(2)

- Line 3의 while-루프 조건이 '참'이므로, line 4~7을 수행한다. 즉, Q에서 'T'와 'G'를 제거한 후, 새 부모 노드를 Q에 삽입한다.



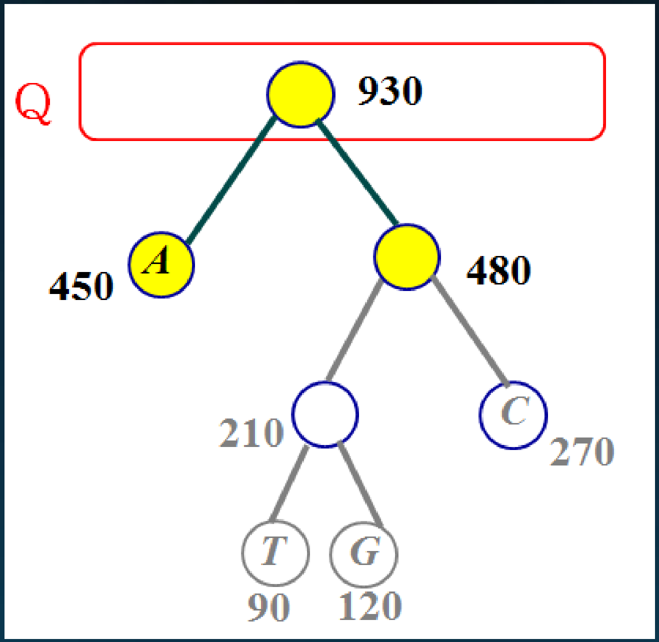
HuffmanCoding 알고리즘의 수행 과정(3)

- Line 3의 while-루프 조건이 ‘참’이므로, line 4~7을 수행한다. 즉, Q에서 ‘T’와 ‘G’의 부모 노드와 ‘C’를 제거한 후, 새 부모 노드를 Q에 삽입



HuffmanCoding 알고리즘의 수행 과정(4)

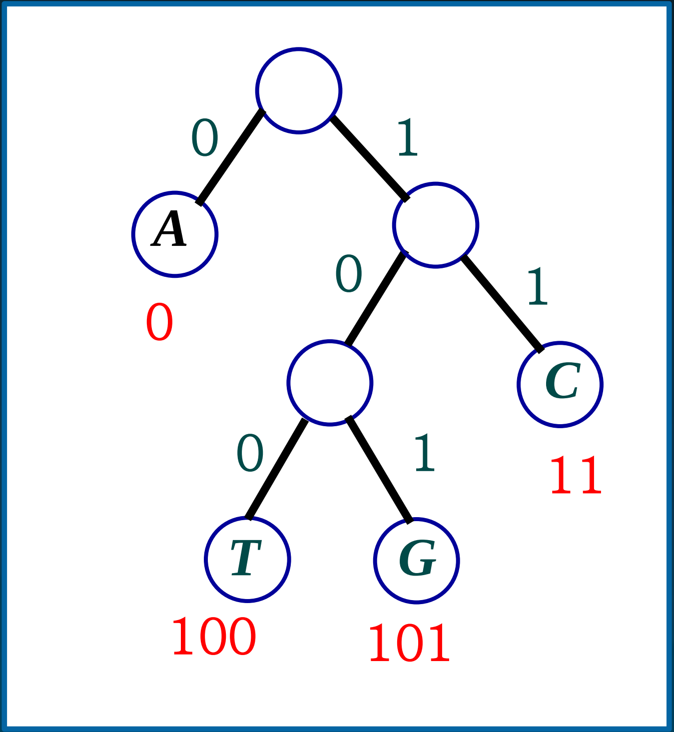
- Line 3의 while-루프 조건이 '참'이므로, line 4~7을 수행한다. 즉, Q에서 'C'의 부모 노드와 'A'를 제거한 후, 새 부모 노드 Q에 삽입한다.



- Line 3의 while-루프 조건이 '거짓'이므로, line 8에서 Q에 있는 노드를 리턴한다. 즉, 허프만 트리의 루트를 리턴한다.

HuffmanCoding 알고리즘의 수행 과정(5)

- 리턴된 트리를 살펴보면 각 이파리 (단말) 노드에만 문자가 있다. 따라서 루트로부터 왼쪽 자식 노드로 내려가면 '0'을, 오른쪽 자식 노드로 내려가면 '1'을 부여하면서, 각 이파리에 도달할 때까지의 이진수를 추출하여 문자의 이진 코드를 구한다.



논의

- 위의 예제에서 'A'는 '0', 'T'는 '100', 'G'는 '101', 'C'는 '11'의 코드가 각각 할당된다.
  - 할당된 코드들을 보면, 가장 빈도수가 높은 'A'가 가장 짧은 코드를 가지고, 따라서 루트의 자식이 되어 있고, 빈도수가 낮은 문자는 루트에서 멀리 떨어지게 되어 긴 코드를 가진다.
  - 또한 이렇게 얻은 코드가 접두부 특성을 가지고 있음을 쉽게 확인할 수 있다.
- 위의 예제에서 압축된 파일의 크기의 bit수는

  (450x1)+(90x3)+(120x3)+(270x2) = 1,620이다.
  - 반면에 아스키 코드로 된 파일 크기는

    (450+90+120+270)x8 = 7,440 bit이다.
  - 따라서 파일 압축률은 (1,620/7,440)x100 = 21.8%이며,

    원래의 약 1/5 크기로 압축되었다.

논의(2)

- 위의 예제에서 얻은 허프만 코드로 아래의 압축된 부분에 대해서 압축을 해제하여보면 다음과 같다.

10110010001110101010100

101 / 100 / 100 / 0 / 11 / 101 / 0 / 101 / 0 / 100

G T T A C G A G A T

시간복잡도

- Line 1: n개의 노드를 만들고, 각 빈도수를 노드에 저장하므로 O(n) 시간이 걸린다.
- Line 2: n개의 노드로 우선순위 큐 Q를 만든다. 여기서 우선순위 큐로서 힙(heap) 자료구조를 사용하면 O(n) 시간이 걸린다.
- Line 3~7은 최소 빈도수를 가진 노드 2개를 Q에서 제거하는 힙의 삭제 연산과 새 노드를 Q에 삽입하는 연산을 수행하므로 O(logn) 시간이 걸린다. 그런데 while-루프는 (n-1)번 반복된다. 왜냐하면 루프가 1번 수행될 때마다 Q에서 2개의 노드를 제거하고 1개를 Q에 추가하기 때문이다. 따라서 line 3~7은 (n-1)xO(logn) = O(nlogn)이 걸린다.
- Line 8은 트리의 루트를 리턴하는 것이므로 O(1) 시간이 걸린다.
- 따라서 시간복잡도는 O(n)+O(n)+O(nlogn)+ O(1) = O(nlogn)이다.

응 용

- 팩스(FAX), 대용량 데이터 저장, 멀티미디어 (Multimedia), MP3 압축 등에 활용된다.
- 또한 정보 이론 (Information Theory) 분야에서 엔트로피 (Entropy)를 계산하는데 활용되며, 이는 자료의 불특정성을 분석하고 예측하는데 이용된다.

Summary

- 부분 배낭 (Fractional Knapsack) 문제에서는 단위 무게 당 가장 값나가는 물건을 계속해서 배낭에 담는다. 마지막엔 배낭에 넣을 수 있을 만큼만 물건을 부분적으로 배낭에 담는다. 시간복잡도는 O(nlogn)이다.
- 집합 커버 (Set Cover) 문제는 근사 (Approximation) 알고리즘을 이용하여 근사해를 찾는 것이 보다 실질적이다. U의 원소들을 가장 많이 포함하고 있는 집합을 항상 F에서 선택한다. 시간복잡도는 $O(n^3)$이다.
- 작업 스케줄링 (Job Scheduling) 문제는 빠른 시작시간 작업 먼저 (Earliest start time first) 배정하는 그리디 알고리즘으로 최적해를 찾는다. 시간복잡도는 O(nlogn)+O(mn)이다. n은 작업의 수이고, m은 기계의 수이다.
- 허프만 (Huffman) 압축은 파일에 빈번히 나타나는 문자에는 짧은 이진 코드를 할당하고, 드물게 나타나는 문자에는 긴 이진 코드를 할당한다. n이 문자의 수일 때, 시간복잡도는 O(nlogn)이다.

BREAK TIME(2)

실 습

1. 부분배낭문제를 파이선으로 구현하기. (난이도:중)

2. 집합 커버 문제를 파이선으로 구현하기. (난이도: 중상)

숙제:

1. 부분배낭문제 알고리즘의 파이선 구현을 완성하기.
2. 집합커버문제 알고리즘의 파이선 구현을 완성하기.
3. 작업 스케쥴링 문제를 파이선으로 구현하기
4. 허프만 압축 알고리즘을 파이선으로 구현하기

Q&A