다섯번째 강의

알고리즘 설계

목차

- 알고리즘 설계 방법론
- 분할 정복법
- 탐욕적인 방법
- 동적 프로그래밍 (다음 강의의 주제)
- 실습
- 스스로 프로그래밍
- 첨삭 지도

1. 알고리즘 설계 방법론

- 새로운 문제에 대한 알고리즘을 구할 때 사용

- 방법론의 종류
  - 분할 정복법(divide-and-conquer)
  - 탐욕적인 방법(greedy method)
  - 동적 프로그래밍(dynamic programming)
  - 백트래킹(backtracking)
  - 분기 한정법(branch-and-bound)

2. 분할 정복법

- 개념
  - 문제의 범위를 2개 이상의 더 작은 범위로 나눈다.
  - 작은 범위에 대해 해를 바로 얻을 수 있으면 OK.
  - 작은 범위가 여전히 크다면 더욱 범위를 나눈다.

- 특징
  - 하향식 접근방법
  - recursion 사용

- **Recursion**을 이용한 대부분의 알고리즘들은 분할 정복법의 특징을 갖는다**!!!**

설계 전략

- 분할(Divide): 해결하기 쉽도록 문제를 여러 개의 작은 부분으로 나눈다.
  - 예: Fibonacci(n)을 구하기 위해 Fibonacci(n-1)과 Fibonacci(n-2)를 구한다
- 정복(Conquer): 나눈 작은 문제를 각각 해결한다.
  - 예: n이 0이거나 1일 경우에는 n을 return한다.
  - 예: 아니면 n을 더 줄여서 다시 Fibonacci를 구한다.
- 통합(Combine): 해결된 해답을 모은다.
  - 예: return Fibonacci(n-1) + Fibonacci(n-2)

분할 정복법의 예: Fibonacci 수열

```
int fibo(int n)
{
   if (n <= 1)
      return n;
   else
      return fibo(n-1) + fibo(n-2);
}
```

분할 정복법에서 고려 사항

- 크기가 n인 입력이 2개 이상의 조각으로 분할되며, 분할된 부분들의 크기가 거의 n에 가깝게 되는 경우
  - 예: 분할정복법으로 n번째 피보나치 항 구하기
  - $T_n = T_{n-1} + T_{n-2} + 1 = 2^{n/2}$

  ⇒ 시간복잡도: 지수(exponential) 시간

  ⇒ 이런 경우에는 동적 프로그래밍이나 탐욕적인 방법으로 알고리즘을 찾아보자.

- 문제의 해가 2^n 이나 n! 등에 비례할 경우에는 분할 정복법으로 쉽게 알고리즘을 발견할 수도 있음
  - 하노이 타워
  - 부분 집합 출력하기
  - 모든 순열 출력하기, ...

3. 탐욕적인 방법의 소개

- 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 해답으로 선택함으로써 최종적인 해답에 도달
- 그 순간의 선택은 그 당시**(local)**에는 최적이다**.**
- 그러나 최적이라고 생각했던 해답들을 모아서 최종적인**(global)**해답을 만들었다고 해서**,** 그 해답이 궁극적으로 최적이라는 보장이 없다**.**
- 따라서 탐욕적인 방법이 항상 최적의 해답을 주는지를 반드시 검증

탐욕적인 알고리즘의 설계 절차

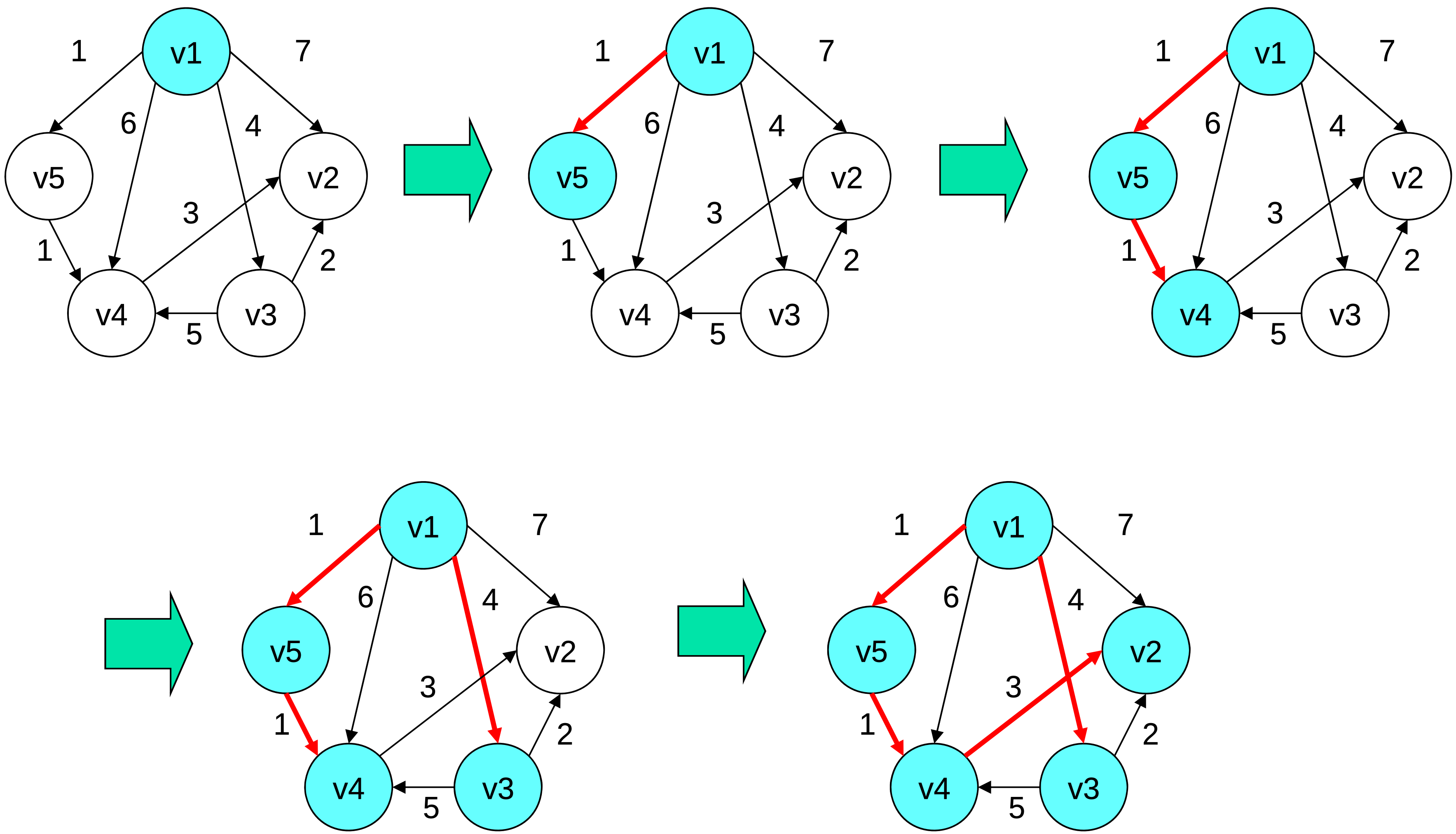
- **선정 과정(selection procedure)**
  - 현재 상태에서 가장 좋으리라고 생각되는(greedy) 해답을 찾아서 해답모음(solution set)에 포함
- **적정성 점검(feasibility check)**
  - 새로 얻은 해답모음이 적절한지를 결정
- **해답 점검(solution check)**
  - 새로 얻은 해답모음이 최적의 해인지를 결정

예: 단일 출발점 최소 경로(Dijkstra)

- 가중치가 있는 방향성 그래프에서 한 특정 정점에서 다른 모든 정점으로 가는 최단경로를 구하는 문제.
- 알고리즘:

- **F := 0;**
- **Y := {v_1};**
- 최종 해답을 얻지 못하는 동안 다음 절차를 계속 반복
  - 선정 절차/적정성 점검:
    - V – Y에 속한 정점 중에서, v_1에서 Y에 속한 정점 만을 거쳐서 최단경로가 되는 정점 v를 선정
    - 정점 v를 Y에 추가
    - v에서 F로 이어지는 최단경로 상의 에지를 F에 추가
  - 해답 점검:
    - Y = V가 되면, T = (V,E)가 최단경로를 나타내는 그래프이다.

예제 그래프


v1 v2 v3 v4 v5
1 7 6 4 3 1 2 5
v1 v2 v3 v4 v5
1 7 6 4 3 1 2 5
v1 v2 v3 v4 v5
1 7 6 4 3 1 2 5
v1 v2 v3 v4 v5
1 7 6 4 3 1 2 5
v1 v2 v3 v4 v5
1 7 6 4 3 1 2 5

4. 실습 – 분할 정복법

- 이항 계수는 아래와 같이 정의된다.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \le k \le n$$

- 그러나 $n!$이나 $k!$을 계산하는 과정에서 오버플로우가 발생할 수 있으므로 이항계수를 구하기 위해서 통상 다음 식을 사용한다.

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases}$$

- n과 k를 입력받아 이항 계수를 계산하는 함수 int bin(n, k)를 작성하고, 다양한 n과 k에 대해 테스트해보자.

실습 – 탐욕적인 방법

- 슬라이드 11의 그래프를 인접 행렬 A[n][n] 에 저장하라.
- 출발점 v를 입력받아, v에서 나머지 정점으로 가는 최단 경로를 구하는 함수 void shortest(A[][n], n, v, dist[n])을 작성하라. int dist[n]에 각 정점으로 가는 최단 경로가 저장되어 있어야 한다. shortest() 함수를 호출한 후, dist[]의 값을 출력해보라.
- dist[] 배열에는 v로부터의 거리만 저장되어 있다. 실제 경로를 출력하는 방법을 생각해보라.