제 5장 동적 계획 알고리즘

Dynamic Programming(DP)

= 분할정복 + 그리디

동적 계획 알고리즘

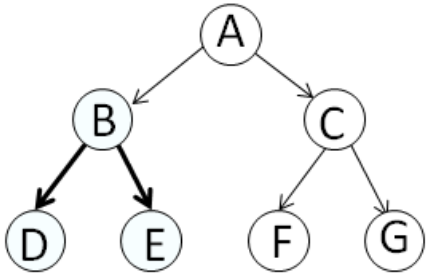
- 동적 계획 (Dynamic Programming)
  - 최적화 문제를 해결하는 알고리즘
  - 동적 계획 알고리즘은 먼저 입력 크기가 작은 부분 문제들을 모두 해결한 후에 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여, 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘
  - 동적 계획 알고리즘은 부분 문제들 사이의 '관계'를 빠짐없이 고려하여 문제를 해결.
  - 동적 계획 알고리즘은 최적 부분 구조 (optimal substructure) 또는 최적성 원칙 (principle of optimality) 특성

  – Principle of Optimality
    - 다단계 결정과정에 대한 최적정책을 결정하는데 핵심적인 의미를 가지고 있음
    - 처음의 상태와 처음의 결정이 무엇이든 간에 목표에 도달하는 나머지 과정은 이때까지의 결정의 결과로서 나타난 상태에 관하여 최적정책을 구성하여야 함
    - "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first step" – R.E.Bellman(1957)
    - 현재의 최선의 선택이 전체의 최선이다

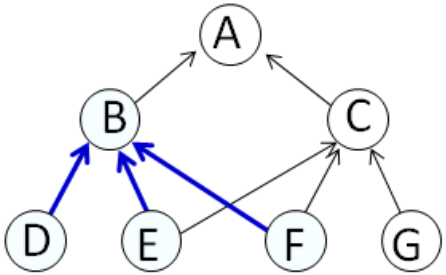
분할 정복법 과 동적 계획법의 차이

- 분할 정복
  - 분할되는 부분문제가 독립적이서 부분문제를 다시 순환적으로 풀어 그 결과를 합침

- 동적 프로그래밍: 알고리즘 복잡성을 ↓
  - 부분문제가 독립적이지 않음
  - 부분문제간에 중복되는 부분이 있어 분할 정복으로 풀 경우 반복적으로 풀어야 하는 문제 발생
  - 부분문제의 계산 결과를 표에 저장해 놓고 필요할 때 이 표에서 그 값을 꺼내 옴



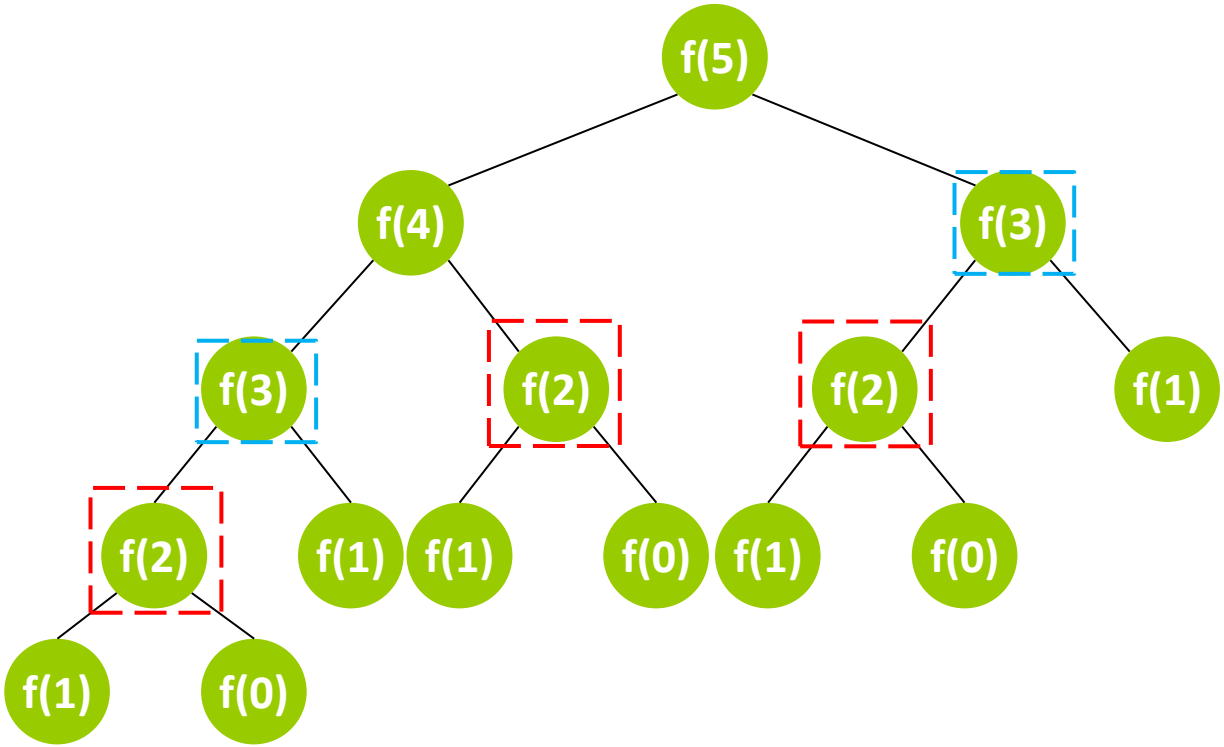
분할 정복 알고리즘



동적 계획 알고리즘

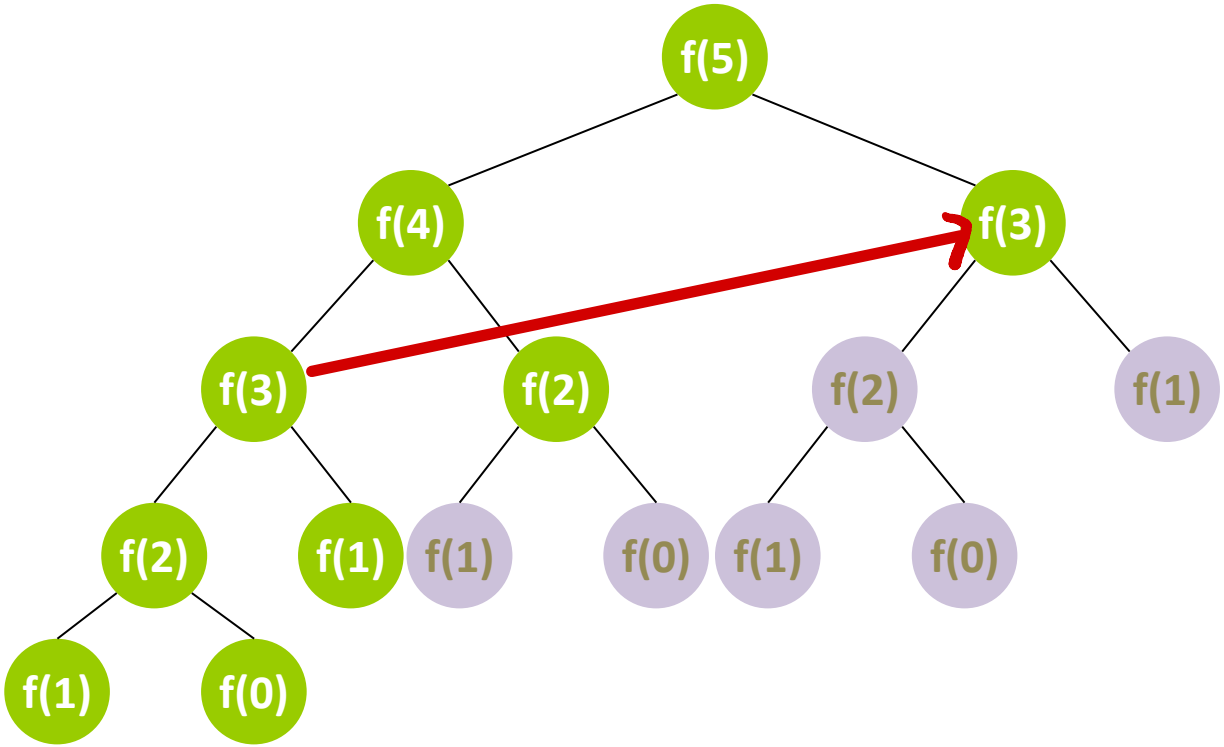
예제 : 피보나치 수열(분할정복으로)

- 2차 피보나치 수열
  - f(n)=f(n-1)+f(n-2)
  - f(0)=0, f(1)=1



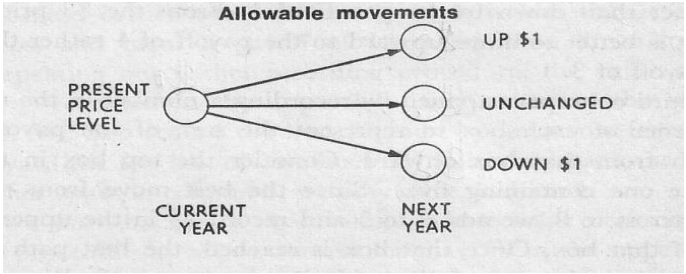
예제 : 피보나치 수열(동적계획법으로)

- 2차 피보나치 수열
  - f(n)=f(n-1)+f(n-2)
  - f(0)=0, f(1)=1



DP 예제 (Pricing Problem)

- SCH 회사의 사장은 향후 5년간 신제품에 대한 가격을 결정하려고 한다
- 그가 고려하고 있는 가격은 $5, $6, $7, $8 네 가지이다.
- 가격에 따라 발생하는 이익은 각 해마다 달라진다.
- 그러나 가격의 변동은 $1 이내로 제한된다.



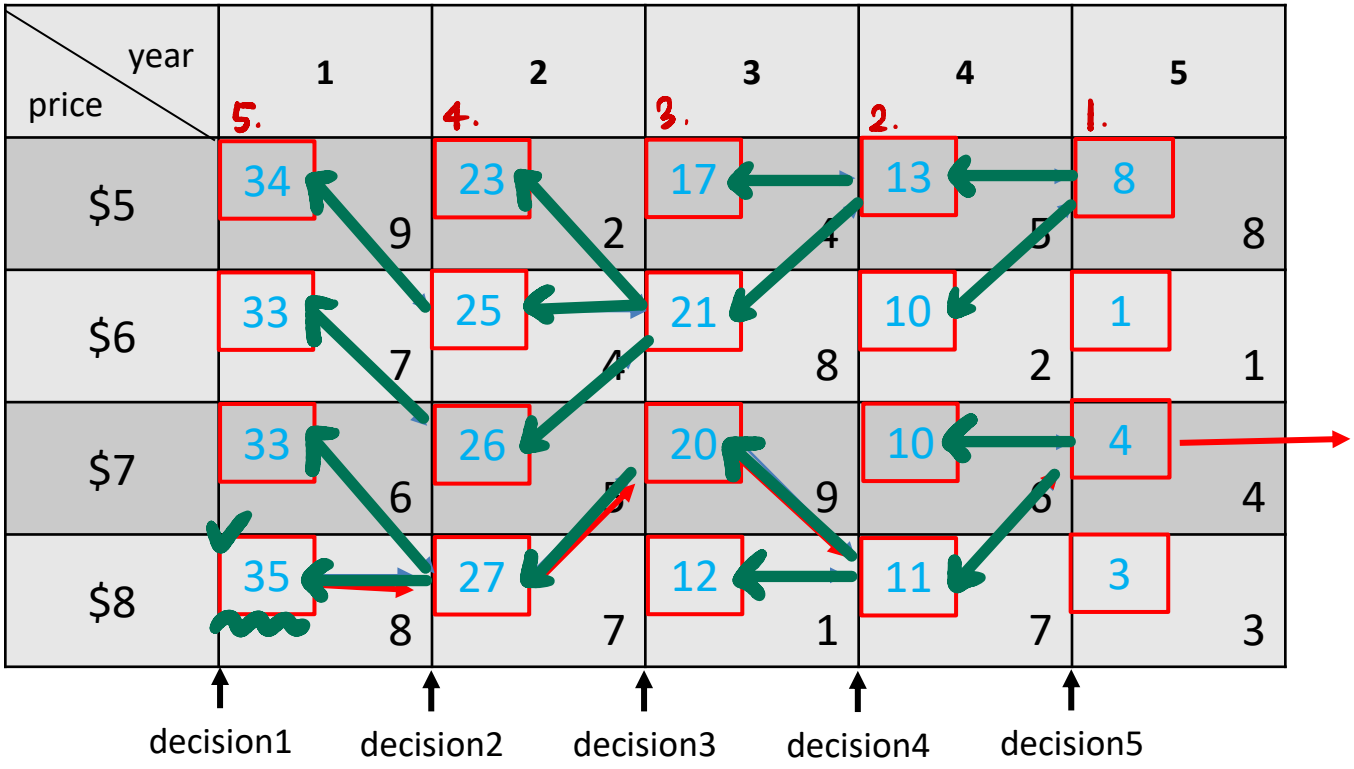
: 그리디 알고리즘

price \ year	1	2	3	4	5
$5	9	2	4	5	8
$6	7	4	8	2	1
$7	6	5	9	6	4
$8	8	7	1	7	3

: 33

DP 예제 (계속)

- **문제의 단계는 년도에 따라 5단계로 구성되며 각 단계에서 취할 수 있는 부분 해는 다음과 같이 계산 된다.**



Knapsack 문제 : NP complete 문제 n의 값이 너무 커질때만

- 배에 화물을 선적하는 경우
  - 물품 i 의 무게는 w_i 이며 가치는v_i (i=1,2,3)
  - 최대 선적 무게는 5톤
  - 최대 선적용량을 만족하며 최대의 가치를 싣기 위한 각 물품의 개수는?

i	w_i	v_i
1	2	65
2	3	80
3	1	30

if 그리디 : 먼저 i=2 선택
그러면 w=5-3=2
다음 i=1
그러면 V=145

- Stage 3 (3번 물품부터 backward)
  - $f_3(y_3)=\max_{k_3}\{30k_3\}$, max $k_3=[5/1]=5$
  - k_i : 물품 i의 선적 개수(결정변수)
  - y_i : Stage i까지 남아 있는 선적 여유량(제약조건)

목적함수

k_3 → 3번 물건을 몇개 담을 것이냐

i	w_i	v_i
1	2	65
2	3	80
3	1	30

y_3	$30k_3$						Optimal Solution	
	$k_3=0$	1	2	3	4	5	$f_3(y_3)$	k_3^*
	$v_3k_3=0$	30	60	90	120	150		
0	0	–	–	–	–	–	0	0
1	0	30	–	–	–	–	30	1
2	0	30	60	–	–	–	60	2
3	0	30	60	90	–	–	90	3
4	0	30	60	90	120	–	120	4
5	0	30	60	90	120	150	150	5

i	w_i	v_i
1	2	65
2	3	80
3	1	30

- stage 2
  - $f_2(y_2)=\max_{k_2}\{80k_2+f_3(y_2-3k_2)\}$, $\max k_2=[5/3]=1$

목적함수

남은거는 k_3로

y_3	$f_3(y_3)$
0	0
1	30
2	60
3	90
4	120
5	150

Stage1에서 넘어옴

y_2	$80k_2+f_3(y_2-3k_2)$		Optimal Solution	
	$k_2=0$	1	$f_2(y_2)$	k_2^*
	$v_2k_2=0$	80		
0	0 + 0 = 0	–	0	0
1	0 + 30 = 30	–	30	0
2	0 + 60 = 60	–	60	0
3	0 + 90 = 90	80 + 0 = 80	90	0
4	0 + 120 = 120	80 + 30 = 110	120	0
5	0 + 150 = 150	80 + 60 = 140	150	0

- stage 1
  - $f_1(y_1)=\max_{k_1}\{65k_1+f_2(y_1-2k_1)\}$, max $k_1=[5/2]=2$

i	w_i	v_i
1	2	65
2	3	80
3	1	30

y_2	$f_2(y_2)$
0	0
1	30
2	60
3	90
4	120
5	150

y_1	$65k_1+f_2(y_1-2k_1)$			Optimal Solution	
	$k_1=0$	i= 1	● i= 1+1	$f_1(y_1)$	k_1^*
	$v_1k_1=0$	65	130		
0	0 + 0 = 0	–	–	0	0
1	0 + 30 = 30	–	–	30	0
2	0 + 60 = 60	65 + 0 = 65	–	65	1
3	0 + 90 = 90	65 + 30 = 95	–	95	1
4	0 + 120 = 120	65 + 60 = 125	130 + 0 = 130	130	2
5	0 + 150 = 150	65 + 90 = 155	130 + 30 = 160	160	2

- stage 1
  - $k_1^*=2$
  - $y_2= 5 - 2 k_1^* = 1$
- stage 2
  - $k_2^*=0$
  - $y_3= 1 - 2 k_2^* = 1$
- stage 3
  - $k_3^*=1$

동적계획법 숙제

로빈후드가 보물창고에 들어가 보니 다음과 같은 보물들이 있었다

Stage 4로 해서

품목	무게(Kg)	가치
금병	6	480
작은동상	2	158
은컵	3	233
은전	1	2

그런데 로빈후드의 가방에는 *13Kg*만 담을 수 있다.
어떻게 담으면 가장 많은 가치의 물건들을 가지고 나갈 수 있을까?
이를 동적계획법으로 풀어 제출하시오

그리디알고리즘

품목	개수	무게(Kg)	가치
금병	2	12	960
작은동상	0	0	0
은컵	0	0	0
은전	1	1	2
합계		13	962

동적계획법

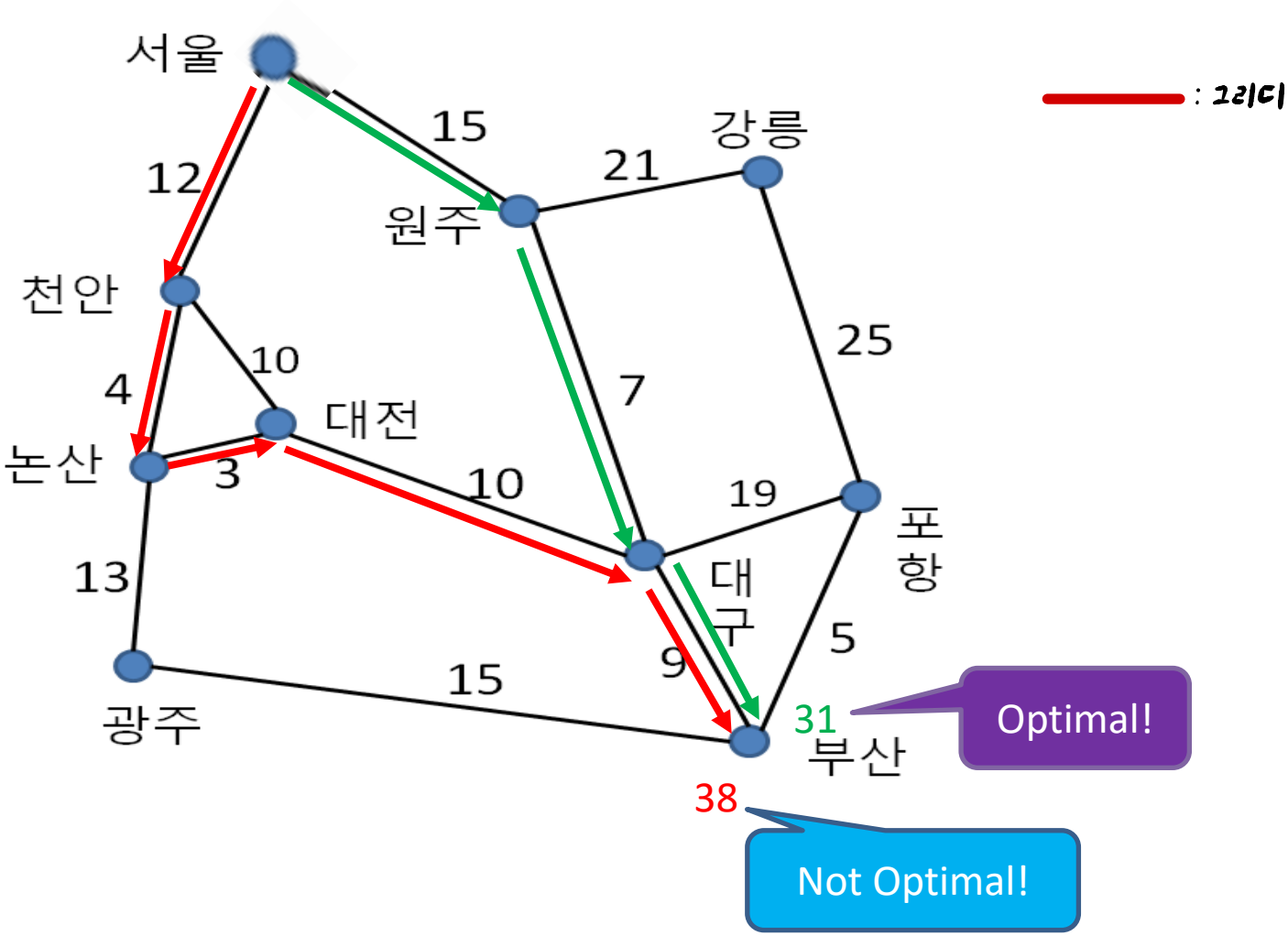
품목	개수	무게(Kg)	가치
금병	1	6	480
작은동상	2	4	316
은컵	1	3	233
은전	0	0	0
합계		13	1029

동적계획법의 활용

- 최단경로찾기 문제
  - 출발점으로부터 목표점까지의 최단경로를 찾는다
  - 앞으로 찾아야할 경로는 지금까지의 경로와는 독립적으로 향후 경로에 최선의 선택을 하자

최단경로 찾기(그리디알고리즘)

서울에서 출발하여 부산에 이르는 최단 경로

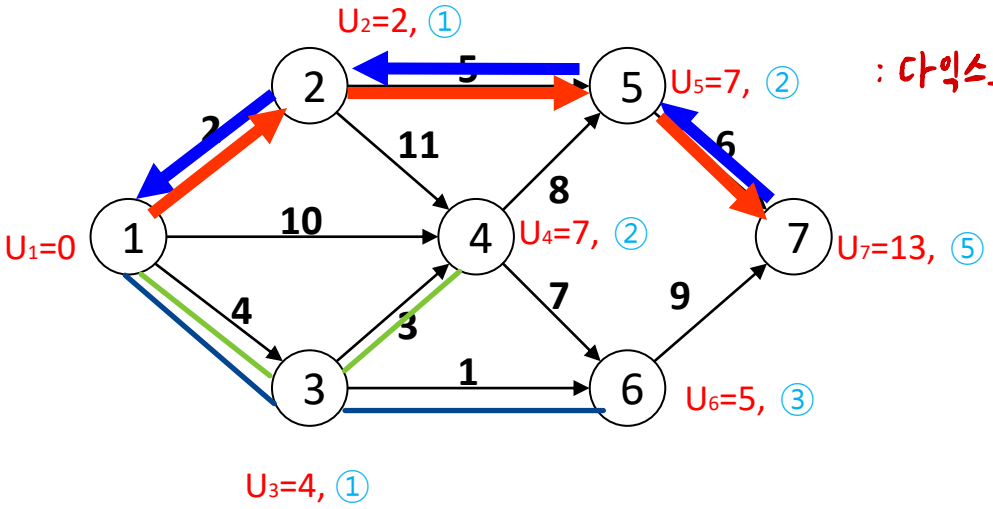


4.3 최단 경로 찾기

- 최단 경로 (Shortest Path) 문제는 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제이다.
  - 출발점으로부터 목표점까지의 최단경로를 찾는다
  - 앞으로 찾아야할 경로는 지금까지의 경로와는 독립적으로 향후 경로에 최선의 선택을 하자
  - 가장 대표적인 알고리즘은 다익스트라 (Dijkstra) 알고리즘.

최단경로 찾기(Shortest Path Problem)

노드1에서 7까지의 최단경로를 찾아보자



j노드까지의최단거리값 $U_j=\min\{U_k+d_{kj}\}$

Dijkstra Shortest Path

- 시작점 노드와 목표점 노드의 최단 거리
- 알고리즘
  - P : 결정집합, T : 미결정집합, U_j : j노드까지의최단거리값

Step 1. Set $U_1=0$, $U_j=d_{ij}(j=2,3,\ldots,n)$, $P=\{1\}$, $T=\{2,3,\ldots,n\}$, $Prev(j)=1, \forall j \in T$

Step 2. Find $k \in T$, where $U_k=\min\{U_j\}$

Set $T=T-\{k\}$, $P=P \cup k$

If $T=\Phi$, Stop

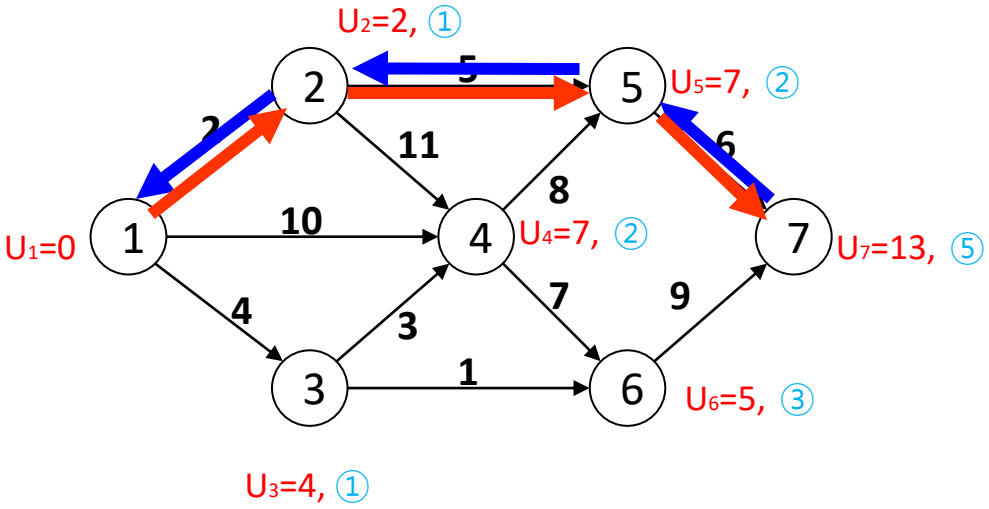
Step 3. Set $U_j=\min\{U_j, U_k+d_{kj}\}$

If $U_k+d_{kj} < U_j \rightarrow Prev(j) = k$

Go to Step 1

최단경로 찾기(Shortest Path Problem)

노드1에서 7까지의 최단경로를 찾아보자



j노드까지의최단거리값 $U_j=\min\{U_k+d_{kj}\}$

Dijkstra Shortest Path

- 시작점 노드와 목표점 노드의 최단 거리
- 알고리즘
  - P : 결정집합, T : 미결정집합, U_j : j노드까지의최단거리값

Step 1. Set $U_1=0$, $U_j=d_{ij}(j=2,3,\ldots,n)$, $P=\{1\}$, $T=\{2,3,\ldots,n\}$, $Prev(j)=1, \forall j \in T$

Step 2. Find $k \in T$, where $U_k=\min\{U_j\}$

Set $T=T-\{k\}$, $P=P \cup k$

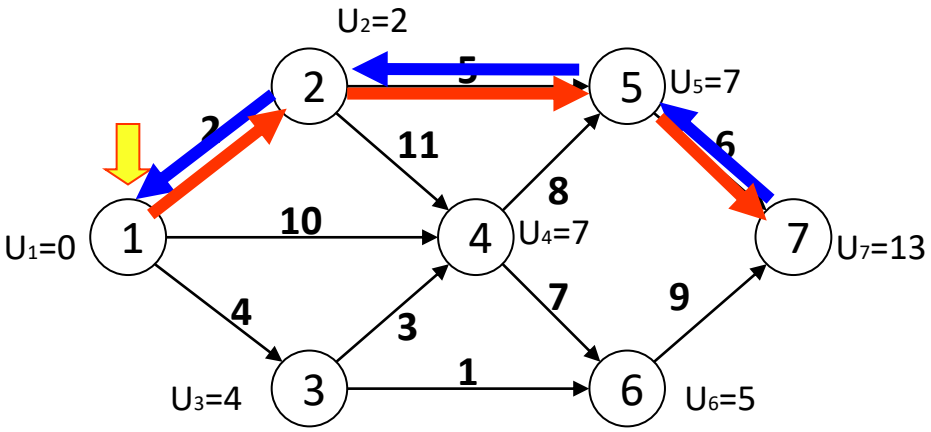
If $T=\Phi$, Stop

Step 3. Set $U_j=\min\{U_j, U_k+d_{kj}\}$

If $U_k+d_{kj} < U_j \rightarrow Prev(j) = k$

Go to Step 1

알고리즘



	1	2	3	4	5	6	7
Prev	1	1	1	3	2	3	5

< 1부터 7까지의 최단거리 >

1 ← 2 ← 5 ← 7

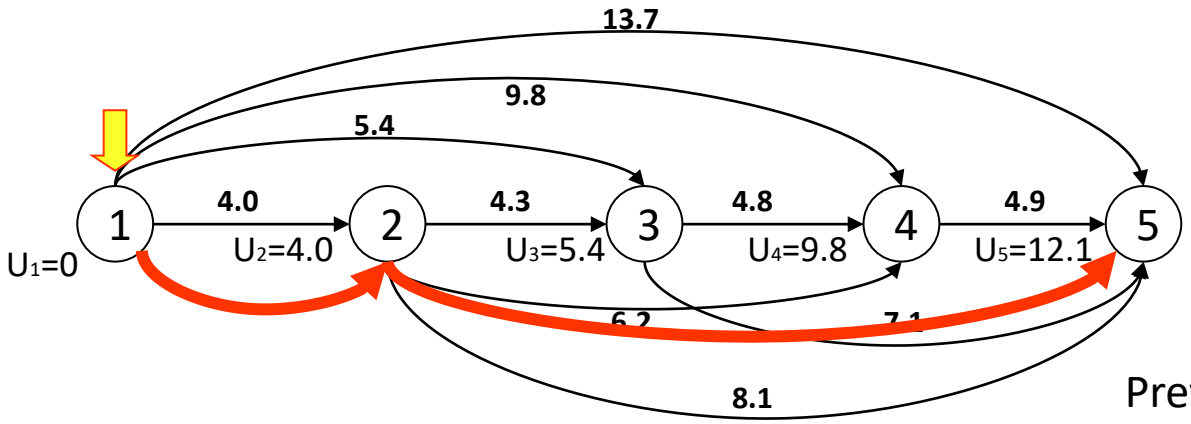
P	T, U_j(j∈T)	P	T, U_j(j∈T)
{1}	T: **2** 3 4 5 6 7 U_j: **2** 4 10 ∞ ∞ ∞	{1,2,3,6,4}	T: **5** 7 U_j: **7** 14
{1,2}	T: **3** 4 5 6 7 U_j: **4** 10 7 ∞ ∞	{1,2,3,6,4,5}	T: **7** U_j: **13**
{1,2,3}	T: 4 5 **6** 7 U_j: 7 7 **5** ∞	{1,2,3,6,4,5,7}	T: U_j:
{1,2,3,6}	T: **4** 5 7 U_j: **7** 7 14		

Min Shortest Path 적용 사례

- 장비 유지/보수 비용
  - 5년째에 교환
  - 도입 시기부터 5년까지 점검 비용
    - 시작은 1부터 5까지
  - 비용 테이블

	1	2	3	4	5
1		4.0	5.4	9.8	13.7
2			4.3	6.2	8.1
3				4.8	7.1
4					4.9

풀이



점검 : 2, 5년째
비용 : 12.1

	1	2	3	4	5
Prev	1	1	1	1	1

P	T, U_j($j \in T$)
{1}	T: 2, 3, 4, 5 / U_j: 4.0, 5.4, 9.8, 13.7
{1,2}	T: 3, 4, 5 / U_j: 5.4, 9.8, 12.1
{1,2,3}	T: 4, 5 / U_j: 9.8, 12.1
{1,2,3,4}	T: 5 / U_j: 12.1
{1,2,3,4,5}	T / U_j

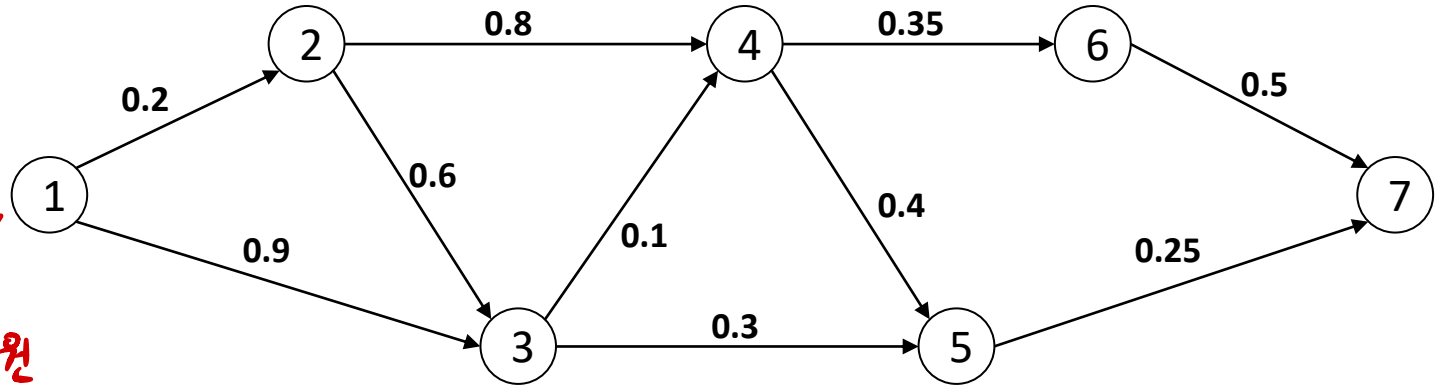
신뢰도

- 네트워크의 신뢰도
  - 가장 신뢰도가 높은 경로
- 알고리즘 적용 시 문제점
  - 최소 비용으로 노드 탐색
  - 곱셈을 하는 신뢰도
    - 예) 20%→40% = 8%의 신뢰도
- 해결점
  1. 방문의 위한 노드 결정 시 최대값 노드 선택
  2. log를 이용하여 덧셈으로 변경

$$ex) \binom{100}{1} = {}_{100}C_1 = \frac{100!}{99!\,1!} = 100$$

$$\log\left(\frac{100!}{99!\,1!}\right) = \log 100! - \log 99! = \log 100 = 2 \rightarrow 10^2 = 100$$

문제점 해결



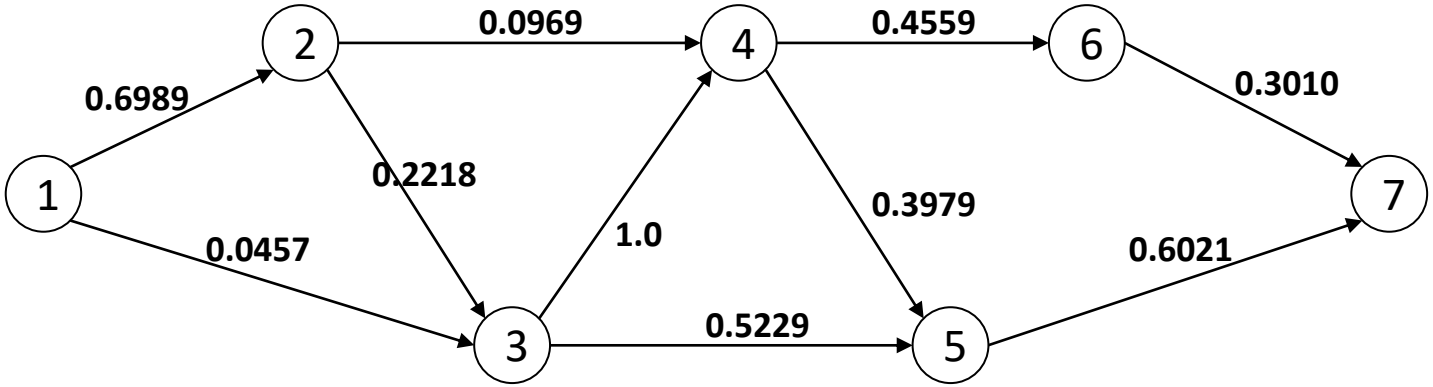
환원
Reduce

$P_{17} = P_{1i} * P_{ij} * P_{j7}$

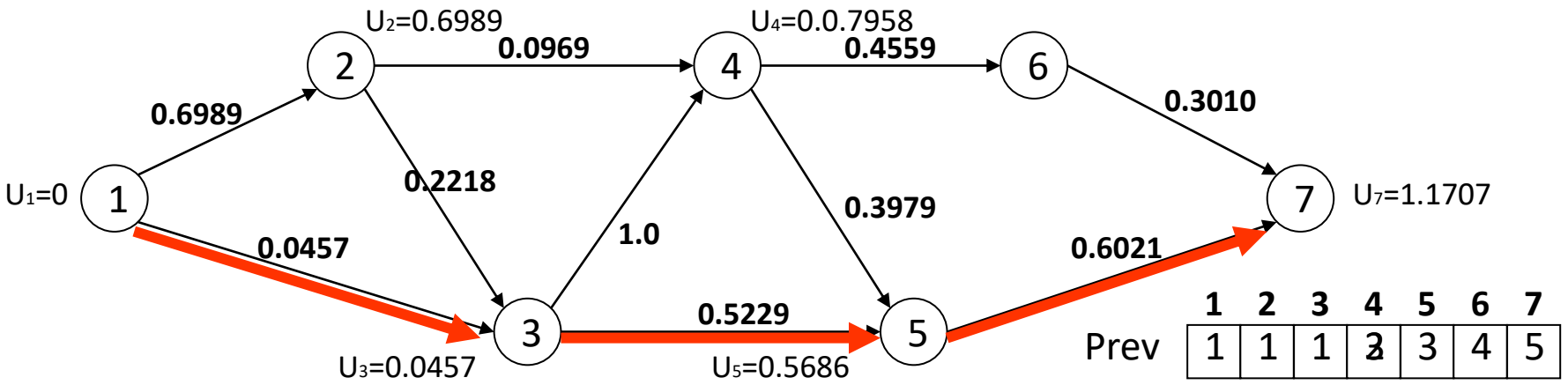
$\log(P_{17}) = \log(P_{1i} * P_{ij} * P_{j7})$ ← Maximize

$-\log(P_{17}) = -\log(P_{1i} * P_{ij} * P_{j7})$ ← Minimize

$-\log(P_{17}) = -\log(P_{1i}) - \log(P_{ij}) - \log(P_{j7})$



풀이



P	T, U_j(j∈T)
{1}	T: 2, 3, 4, 5, 6, 7 / U_j: 0.6989, 0.0457, ∞, ∞, ∞, ∞
{1,3}	T: 2, 4, 5, 6, 7 / U_j: 0.6989, 1.0457, 0.5686, ∞, ∞
{1,3,5}	T: 2, 4, 6, 7 / U_j: 0.6989, 1.0457, ∞, 1.1707
{1,3,5,2}	T: 4, 6, 7 / U_j: 0.7958, ∞, 1.1707
{1,3,5,2,4}	T: 6, 7 / U_j: 1.2517, 1.1707
{1,3,5,2,4,7}	T: 6 / U_j: 1.2517
{1,3,5,2,4,7,6}	T / U_j

최단 경로 : 1 → 3 → 5 → 7

5.5 동전 거스름돈

- 적은 수의 동전으로 거스름돈을 만듦
  - 대부분의 경우 그리디 알고리즘으로 해결되나, 해결 못하는 경우도 있음
  - 동적 계획 알고리즘을 동전 거스름돈 문제에 대하여 적용해보자.
- 동적 계획 알고리즘을 고안하기 위해서는 부분 문제를 찾아내야 한다.
  - 정해진 동전의 종류, d_1, d_2, …, d_k가 있고, 거스름돈 n원이 있다.
    - 단, $d_1 > d_2 > \cdots > d_k = 1$
    - 예를 들어, 우리나라의 동전의 경우 $d_1 = 500$, $d_2 = 100$, $d_3 = 50$, $d_4 = 10$, $d_5 = 1$
  - 부분 문제들의 해를 아래와 같이 1차원 배열 C에 저장
    - 1원을 거슬러 받을 때 사용되는 최소의 동전 수 C[1]
    - 2원을 거슬러 받을 때 사용되는 최소의 동전 수 C[2]
    - ⋮
    - j원을 거슬러 받을 때 사용되는 최소의 동전 수 C[j]
    - ⋮
    - n원을 거슬러 받을 때 사용되는 최소의 동전 수 C[n]

- j원을 거슬러 받을 때 최소의 동전 수 C[j]를 구하는 기본 아이디어
  - d_1=500, d_2=100, d_3=50, d_4=10, d_5=1로 생각해 보자.
  - 500원 동전이 거스름돈 j원에 필요하면
    - C[j]는 (j-500)원의 해에 500원 동전 1개를 추가
    - $C[j] = C[j-d_1] + 1 = C[j-500] + 1$
    - 750원의 거스름 돈에 500원 동전이 쓰인다면 250원의 해에 500원 동전 하나 추가가 최적
  - 100원 동전이 거스름돈 j원에 필요하면
    - (j-100)원의 해에 100원 동전 1개 추가
    - $C[j] = C[j-d_2] + 1 = C[j-100] + 1$

      ⋮
  - 1원짜리 동전이 거스름돈 j원에 필요하면
    - $C[j] = C[j-d_5] + 1 = C[j-1] + 1$
  - C[j]는 위의 5가지 중에서 가장 작은 값
    - $C[j] = \min_{1 \le i \le k}\{C[j-d_i] + 1\}$
    - if $j \ge d_i$ (거스름 돈이 동전 액면보다는 커야함)

알고리즘

DPCoinChange

입력: 거스름돈 n원, k개의 동전의 액면, $d_1 > d_2 > \cdots > d_k = 1$

출력: C[n]

```
1. C[0]=0                                  //거스름돈이 0이라면 동전이 필요없음
2. for i = 1 to n
       C[i]=∞                              //최소값을 찾기 위해서는 초기값을 큰 수로 지정
3. for j = 1 to n
4.     for i = 1 to k {                    //동전 액면을 가장 큰 것부터 적은 순서로 조사
5.         if (d_i ≤ j) and (C[j-d_i]+1<C[j])   //현재 동전 액면이 거스름돈 보다는 작고 거스름 동전 개수를 줄인다면
6.             C[j]=C[j-d_i]+1             //현재 동전 액면 그 동전을 거스름돈에 포함한다
       }
   }
7. return C[n]
```

$C[j] = \min_{1 \le i \le k}\{C[j-d_i] + 1\}$

- 예제
  - d_1=16, d_2=10, d_3=5, d_4=1이고, 거스름돈 n=20일 때

  - 배열 C를 아래와 같이 초기화

j	0	1	2	3	4	5	6	7	8	9	10	…	16	17	18	19	20
C	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	…	∞	∞	∞	∞	∞

  - 거스름돈 1원~4원까지
  - 1원짜리 동전 (d_4=1)밖에 고려할 동전이 없으므로, 각 j에서 1을 뺀, 즉, (j-1)의 해인 (C[j-1]+1)이 C[j]가 된다.
  - C[1] = C[j-1]+1 = C[1-1]+1 = C[0]+1 = 0+1 = 1

j	0	1
	0	∞

⇨

j	0	1
	0	1

  - C[2] = C[j-1]+1 = C[2-1]+1 = C[1]+1 = 1+1 = 2

j	1	2
	1	∞

⇨

j	1	2
	1	2

  - C[3] = C[j-1]+1 = C[3-1]+1 = C[2]+1 = 2+1 = 3

j	2	3
	2	∞

⇨

j	2	3
	2	3

  - C[4] = C[j-1]+1 = C[4-1]+1 = C[3]+1 = 3+1 = 4

j	3	4
	3	∞

⇨

j	3	4
	3	4

- j=5, 즉, 거스름돈이 5원일 때
  - i=1, 2 (16원, 10원동전)에 대하여, ($d_i \le 5$)는 '거짓'
    - 액면이 거스름 돈 5원을 초과 함. 고려할 필요가 없음
  - i=3 (5원짜리 동전)에 대해서,
    - line 5의 if-조건인 (5≤5)가 '참' 액면이 거스름돈 만족
    - (C[5-5]+1 < C[5]) → (C[0]+1 <∞) → (0+1 <∞)이므로 '참'
    - C[j] = C[j-d_i]+1가 수행
    - C[5] = C[5-5]+1 = C[0]+1 = 0+1 = 1이 된다. 즉, C[5]=1이다.

j	0	1	2	3	4	5
	0	1	2	3	4	∞

⇨

j	0	1	2	3	4	5
	0	1	2	3	4	1

  - i=4 (1원짜리 동전)일 때
    - line 5의 if-조건인 ($d_5 \le 5$)는 '참'
    - (C[j-d_i]+1 < C[j]) = (C[5-1] +1 < C[4]) = (C[4] +1 < C[5]) = (4+1 < 1) = (5 < 1)가 '거짓'
    - C[5]는 변하지 않고 그대로 1을 유지
    - 즉, 1원짜리 동전으로 거스름돈을 주려 하면 오히려 동전 수가 늘어남

- j=6, 7, 8, 9
  - i=1, 2 (10원, 16원 액면이 거스름 돈 초과)
  - i=3 (5원짜리 동전)
    - C[6]=C[j-5]+1=C[6-5]+1=C[1]+1=1+1 = 2

    - C[7]=C[j-5]+1=C[7-5]+1=C[2]+1=2+1 = 3

    - C[8]=C[j-5]+1=C[8-5]+1=C[3]+1=3+1 = 4

    - C[9]=C[j-5]+1=C[9-5]+1=C[4]+1=4+1 = 5

  - i=4 (1원짜리 동전)
    - line 5의 if-조건의 $(C[j-d_i]+1 < C[j])= (C[j-1]+1 < C[j])$이 각각의 j에 대해서
    - (1+1) < 2, (2+1) < 3, (3+1) < 4, (4+1) < 5로서 '거짓'이 되어 C[j]는 변경되지 않음
    - 사실은 i=3일 때와 동일하므로 각각 갱신 안 된다.

j	0	1	2	3	4	5	6	7	8	9
C	0	1	2	3	4	1	∞	∞	∞	∞
	0	1	2	3	4	1	**2**	∞	∞	∞
	0	1	2	3	4	1	2	**3**	∞	∞
	0	1	2	3	4	1	2	3	**4**	∞
	0	1	2	3	4	1	2	3	4	**5**

- j=10이면 거스름돈이 10원이면,
  - i=1 (액면이 거스름 돈 초과)
  - i=2 (10원짜리 동전)일 때,
    - line 5의 if-조건인 ($d_i \le j$) →(10≤10)은 '참', ' 액면이 거스름돈 만족
    - (C[j-d_i]+1<C[j]) → (C[10-10]+1<C[10]) → (C[0]+1< C[10]) = (0+1<∞)이 '참'
    - 'C[j]=C[j-d_i]+1'
    - C[10] = C[10-10]+1 = C[0]+1 = 0+1 = 1이다. 즉, C[10]=1이다.

  - i=3 (5원짜리 동전)일 때
    - line 5의 if-조건인 ($d_i \le j$) = (5<10)는 '참'이나,
    - (C[10-5]+1<C[10]) → (C[5]+1<C[10]) → (1+1<1)이 '거짓'
    - C[10]은 변하지 않는다. 즉, 5원짜리 2개보다는 10원짜리 1개 낫다.

  - i=4 (1원짜리 동전)일 때
    - line 5의 if-조건인 ($d_i \le j$) = (1<10)는 '참'이나,
    - (C[j-d_i]+1<C[j]) → (C[10-1]+1<C[10]) → (C[9]+1< C[10]) → (5+1<1) = (6<1)이 '거짓'
    - C[10]이 변하지 않고 그대로 1을 유지한다.

j	0	1	2	3	4	5	6	7	8	9	10
C	0	1	2	3	4	1	2	3	4	5	**1**

- j=20일 때,
  - i=1 (16원짜리 동전)일 때,
    - C[20] = C[j-16]+1 = C[4]+1 = 4 +1 = 5

  - i=2 (10원짜리 동전)일 때
    - C[j-10]+1 = C[10]+1 = 1+1 = 2 < C[20]=5
    - C[20]=2

  - i=3 (5원짜리 동전)일 때
    - (C[j-5]+1<C[j]) = (C[15]+1 = 3 < C[20]) =2)이 '거짓'

  - i=4 (1원짜리 동전)일 때
    - (C[20-1]+1 =C[19]+1 =5 < C[20]) =2 이 '거짓'

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
C	0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	1	2	3	4	2

- 거스름돈 20원에 대한 동적계획법의 최종해
  - C[20]=2개의 동전

동적 계획 알고리즘의 해

- 그리디 알고리즘
  - 20원에 대해 16원짜리 동전을 먼저 '욕심내어' 취함
  - 4원이 남게 되어, 1원짜리 4개를 취함

그리디 알고리즘의 해

시간복잡도

- DPCoinChange 알고리즘의 시간복잡도: O(nk)
  - n: 거스름 돈, k: 동전의 가지 수
  - 거스름돈 j가 1원~n원까지 변하며,
  - 각각의 j에 대해서 최악의 경우 모든 동전 $(d_1, d_2, \cdots, d_k)$을 (즉, k개를) 1번씩 비교

An Inventory-Production Problem

- 한 회사는 다음과 같은 향후 4주기의 제품 수요에 대한 공급을 하여야 함
- 4주기 동안의 수요는 다음과 같다고 가정

Period n	Units required D_n
1	2
2	3
3	2
4	4

- 제품의 생산 단가는 $1이며 $3의 Setup Cost(고정비용)이 수반됨
- 한 주기 당 6개 이상을 생산 할 수 는 없음.
- 따라서 생산 비용은 다음과 같음.

$$\text{Production cost} = 3 + 1 * X, \quad \text{if } 0 < X \leq 6$$
$$= 0, \quad \text{if } X = 0$$

- 다음 주기까지 재고를 유지하기 위한 비용은 unit 당 $0.5임
- 수요를 충족하는 최소 비용 정책은?
  - 각 주기 별 생산량을 결정
- 1주기 시작과 4주기 끝의 재고는 0이어야 함

- 다음과 같이 DP문제로 정의 할 수 있음

  - Stages : each period is a stage, n = 1,2,3,4
  - State variable : the amount of inventory at the beginning of a period, I_n
  - Decision variable : the production level each period, X_n
  - Optimal decision rule: specify the optimal production X_n^*
    - X_n^* 은 I_n 의 종속 함수임. 즉, $X_n^* = X_n(I_n)$

- 각 주기 간의 재고량(Inventory)의 연결식은 다음과 같음

$$I_{n+1} = I_n + X_n - D_n$$

D_n is demand in the n^{th} period

- 각 주기의 비용식은 다음과 같음

$$\text{Period cost} = C_n(X_n, I_n) = 3 + 1X_n + 0.5I_n \text{ , if } X_n > 0$$
$$= 0 + 0.5I_n \text{ , if } X_n = 0$$

- **$f_n(I_n)$을 n주기 부터 마지막까지의 최적(최소)비용식이라 하자**

Period 4

$$f_4(I_4) = \underset{0 \le X_4 \le 6,\ X_4 + I_4 \ge D_4}{\text{Mininmum}} \{C_4(X_4, I_4)\}$$

Period 3

$$f_3(I_3) = \underset{0 \le X_3 \le 6,\ X_3 + I_3 \ge D_3}{\text{Mininmum}} \{C_3(X_3, I_3) + f_4(I_3 + X_3 - D_3)\}$$

. . .

Period n

$$f_n(I_n) = \underset{0 \le X_n \le 6,\ X_n + In \ge D_n}{\text{Mininmum}} \{C_n(X_n, I_n) + f_{n+1}(I_n + X_n - D_n)\}$$

- $f_n(I_n)$은 주어진 I_n값에 따라 최소비용을 만족하는 결정변수 X_n 에 따라 달라짐.
- 결국 $f_1(I_1)$을 구하는 문제임! ($I_1 = 0$)

4주기

- 4주기부터 문제를 시작함.
- 마지막 단계이므로 재고량이 0이므로 쉽게 해결할 수 있음
- 초기 재고는 4 이상을 고려할 필요가 없음
  - $D_4 = 4$ 이므로

Period cost = $C_n(X_n, I_n)$ $= 3 + 1X_n + 0.5I_n$, if $X_n > 0$
$= 0 + 0.5I_n$, if $X_n = 0$

Period 4 solution

Beginning inventory	Optimal production	Cost		Total cost($f_4(I_4)$)
		Production	Inventory	
0	4	$7	$0	$7.0
1	3	6	0.5	6.5
2	2	5	1.0	6.0
3	1	4	1.5	5.5
4	0	0	2.0	2.0

3주기

- 초기 재고는 0부터 6까지 가질 수 있음
  - 3~4주기 총 수요가 6이므로
- 따라서 3주기의 최적 의사결정은 다음과 같음

Beginning inventory I_3	Possible Production amounts X_3	$D_3 = 2$ Cost in period 3			Ending inventory I_4	Subsequent costs $f_4(I_4)$	Total Costs $C_3(X_3,I_3) + f_4(I_4)$
		Production	Inventory	Total $C_3(X_3,I_3)$			
0	2	5	0	5.0	0	7.0	$12.0
	3	6	0	6.0	1	6.5	12.5
	4	7	0	7.0	2	6.0	13.0
	5	8	0	8.0	3	5.5	13.5
	6*	9	0	9.0	4	2.0	11.0*

(3주기 계속)

1	1	4	0.5	4.5	0	7.0	11.5
	2	5	0.5	5.5	1	6.5	12.0
	3	6	0.5	6.5	2	6.0	12.5
	4	7	0.5	7.5	3	5.5	13.0
	5*	8	0.5	8.5	4	2.0	10.5*
2	0*	0	1.0	1.0	0	7.0	8.0*
	1	4	1.0	5.0	1	6.5	11.5
	2	5	1.0	6.0	2	6.0	12.0
	3	6	1.0	7.0	3	5.5	12.5
	4	7	1.0	8.0	4	2.0	10.0
3	0*	0	1.5	1.5	1	6.5	8.0*
	1	4	1.5	5.5	2	6.0	11.5
	2	5	1.5	6.5	3	5.5	12.0
	3	6	1.5	7.5	4	2.0	9.5
4	0*	0	2.0	2.0	2	6.0	8.0*
	1	4	2.0	6.0	3	5.5	11.5
	2	5	2.0	7.0	4	2.0	9.0
5	0*	0	2.5	2.5	3	5.5	8.0*
	1	4	2.5	6.5	4	2.0	8.5
6	0*	0	3.0	3.0	4	2.0	5.0*

Summary for period 3

Beginning inventory I_3	Optimal production X_3^*	Total cost Period3 and 4 $f_3(I_3)$	Ending inventory I_4
0	6	$11.0	2
1	5	10.5	2
2	0	10.5	0
3	0	8.0	1
4	0	8.0	2
5	0	8.0	3
6	0	5.0	4

2주기

- 초기 재고는 0부터 4까지 가질 수 있음
  - 1주기 (최대 생산량 6 - 총 수요가 2)= 4
- 따라서 2주기의 최적 의사결정은 다음과 같음

Beginning inventory I_2	Possible Production amounts X_2	D_2= 2 Cost in period 3			Ending inventory I_3	Subsequent costs $f_3(I_3)$	Total Costs $C_2(X_2,I_2)+f_3(I_3)$
		Production	Inventory	Total $C_2(X_2,I_2)$			
0	3	6	0	6.0	0	11.0*	$17.0
	4	7	0	7.0	1	10.5	17.5
	5*	8	0	8.0	2	8.0	16.0*
	6	9	0	9.0	3	8.0	17.0

(2주기 계속)

1	2	5	0.5	5.5	0	11.0	16.5
	3	6	0.5	6.5	1	1.05	17.0
	4*	7	0.5	7.5	2	8.0	15.5*
	5	8	0.5	8.5	3	8.0	16.5
	6	9	0.5	9.5	4	8.0	17.5
2	1	4	1.0	5.0	0	11.0	16.0
	2	5	1.0	6.0	1	1.05	16.5
	3*	6	1.0	7.0	2	8.0	15.0*
	4	7	1.0	8.0	3	8.0	16.0
	5	8	1.0	9.0	4	8.0	17.0
	6	9	1.0	10.0	5	8.0	18.0
3	0*	0	1.5	1.5	0	11.0	12.5*
	1	4	1.5	5.5	1	10.5	16.0
	2	5	1.5	6.5	2	8.0	14.5
	3	6	1.5	7.5	3	8.0	15.5
	4	7	1.5	8.5	4	8.0	16.5
4	0*	0	2.0	2.0	0	10.5	12.5*
	1	4	2.0	6.0	1	8.0	14.0
	2	5	2.0	7.0	2	8.0	15.0
	3	6	2.0	8.0	3	8.0	16.0
	4	7	2.0	9.0	4	8.0	17.0

Summary for period 2

Beginning inventory I_2	Optimal production X_2^*	Total cost Period3 and 4 $f_2(I_2)$	Ending inventory I_3
0	5	16.0	2
1	4	15.5	2
2	3	15.0	2
3	0	12.5	0
4	0	12.5	1

1주기

- 1주기 초기 재고는 0

Beginning inventory I_1	Possible Production amounts X_1	D_2= 2 Cost in period 1			Ending inventory I_2	Subsequent costs $f_2(I_2)$	Total Costs $C_1(X_1,I_1)$+ $f_2(I_2)$
		Production	Inventory	$C_1(X_1,I_1)$			
0	2	5	0	5.0	0	16.0	21.0
	3	6	0	6.0	1	15.5	21.5
	4	7	0	7.0	2	15.0	22.0
	5*	8	0	8.0	3	12.5	20.5*
	6	9	0	9.0	4	12.5	21.5

- Optimal decision through each period

Period	Beginning inventory	Optimal production	Ending inventory = beginning inventory of next period
1	0	5	3
2	3	0	0
3	0	6	2
4	2	0	0

동적계획법의 활용

- 모든쌍 최단경로
- 연속행렬 곱셈
- 편집거리
- 배낭문제
- 동전거스름돈

응 용

- 맵퀘스트 (Mapquest)와 구글 (Google) 웹사이트의 지도 서비스
- 자동차 네비게이션
- 네트워크와 통신 분야
- 모바일 네트워크
- 산업 공학
- 경영 공학의 운영 연구 (Operation Research)
- 로봇 공학
- 교통 공학
- VLSI 디자인 분야 등