탐욕적인 접근방법

알고리즘해석
강의 슬라이드 #4
도경구

소개

- 탐욕적인 알고리즘(Greedy algorithm)은 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 해답으로 선택함으로써 최종적인 해답에 도달한다.
- 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 알고리즘은 항상 최적의 해답을 주는지를 반드시 검증해야 한다.

탐욕적인 알고리즘 설계 절차

1. **선정과정(selection procedure)**
   현재 상태에서 가장 좋으리라고 생각되는(greedy) 해답을 찾아서 해답모음(solution set)에 포함시킨다.
2. **적정성점검(feasibility check)**
   새로 얻은 해답모음이 적절한지를 결정한다.
3. **해답점검(solution check)**
   새로 얻은 해답모음이 최적의 해인지를 결정한다.

2001 -03 -19 알고리즘 강의 슬라이드 4 3

보기: 거스름돈 문제

- 문제: 동전의 개수가 최소가 되도록 거스름 돈을 주는 문제
- 탐욕적인 알고리즘
  - 거스름돈을 x라 하자.
  - 먼저, 가치가 가장 높은 동전부터 x가 초과되지 않도록 계속 내준다.
  - 이 과정을 가치가 높은 동전부터 내림순으로 총액이 정확히 x가 될 때까지 계속한다.
- 현재 우리나라에서 유통되고 있는 동전만을 가지고, 이 알고리즘을 적용하여 거스름돈을 주면, 항상 동전의 개수는 최소가 된다. 따라서 이 알고리즘은 최적(optimal)!

2001 -03 -19 알고리즘 강의 슬라이드 4 4

최적의 해를 얻지 못하는 경우

- 12원 짜리 동전을 새로 발행했다고 하자.
- 이 알고리즘을 적용하여 거스름돈을 주면, 항상 동전의 개수는 최소가 된다는 보장이 없다.
- 보기: 거스름돈 액수 = 16원
  - ✓ 탐욕알고리즘의 결과: 12원 × 1개 = 12원, 1원 × 4개 = 4원
  - ✓ 동전의 개수 = 5개 ⇒ 최적(optimal)이 아님!
  - ✓ 최적의 해: 10원 × 1개, 5원 × 1개, 1원 × 1개가 되어 동전의 개수는 3개가 된다.

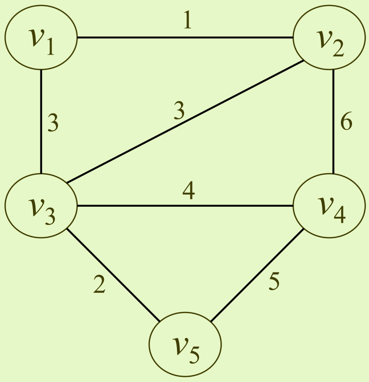
2001 -03 -19 알고리즘 강의 슬라이드 4 5

그래프 용어

- 비방향성 그래프(undirected graph) $G = (V,E)$,
  - ✓ V는 정점(vertex)의 집합
  - ✓ E는 이음선(edge)의 집합
- 경로(path)
- 연결된 그래프(connected graph) - 어떤 두 정점 사이에도 경로가 존재
- 부분그래프(subgraph)
- 가중치 포함 그래프(weighted graph)
- 순환경로(cycle)
- 순환적그래프(cyclic graph), 비순환적그래프(acyclic graph).
- 트리(tree) - 비순환적이며, 비방향성 그래프.
- 뿌리 있는 트리(rooted tree) - 한 정점이 뿌리로 지정된 트리.

2001 -03 -19 알고리즘 강의 슬라이드 4 6

예: 연결된 가중치 비방향그래프



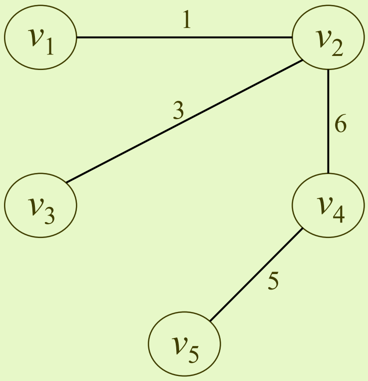
2001 -03 -19 알고리즘 강의 슬라이드 4 7

정의: 신장트리(Spanning Tree)

- 연결된, 비방향성 그래프 *G*에서 순환경로를 제거하면서 연결된 부분그래프가 되도록 이음선을 제거하면 <u>신장트리(spanning tree)</u>가 된다.
- 따라서 신장트리는 *G*안에 있는 모든 정점을 다 포함하면서 트리가 되는 연결된 부분그래프이다.

2001 -03 -19 알고리즘 강의 슬라이드 4 8

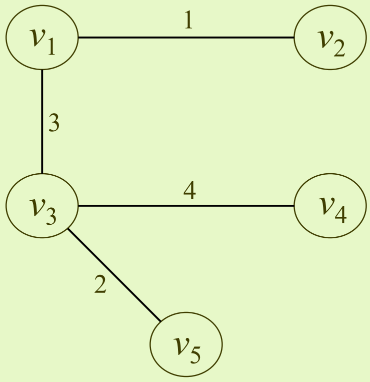
예: 신장트리



정의: 최소비용신장트리 (Minimum Spanning Tree)

- 신장트리가 되는 G의 부분그래프 중에서 가중치가 최소가 되는 부분그래프를 **최소비용신장트리(minimum spanning tree)**라고 한다. 여기서 최소의 가중치를 가진 부분그래프는 반드시 트리가 되어야 한다. 왜냐하면, 만약 트리가 아니라면, 분명히 순환경로(cycle)가 있을 것이고, 그렇게 되면 순환경로 상의 한 이음선을 제거하면 더 작은 비용의 신장트리가 되기 때문이다.
- 관찰: 모든 신장트리가 최소비용신장트리는 아니다.

예: 최소비용 신장트리



2001 -03 -19 알고리즘 강의 슬라이드 4 11

최소비용신장트리의 적용 예

- 도로건설 - 도시들을 모두 연결하면서 도로의 길이가 최소가 되도록 하는 문제
- 통신(telecommunications) - 전화선의 길이가 최소가 되도록 전화 케이블 망을 구성하는 문제
- 배관(plumbing) - 파이프의 총 길이가 최소가 되도록 연결하는 문제

2001 -03 -19 알고리즘 강의 슬라이드 4 12

무작정 알고리즘

- 알고리즘
  - ✓ 모든 신장트리를 다 고려해 보고, 그 중에서 최소비용이 드는 것을 고른다.
- 분석
  - ✓ 이는 최악의 경우, 지수보다도 나쁘다.
  - ✓ 이유?

탐욕적인 알고리즘

- 문제: 비방향성 그래프 $G = (V,E)$가 주어졌을 때, $F \subseteq E$를 만족하면서, (V,F)가 G의 최소비용신장트리(MST)가 되는 F를 찾는 문제.
- 알고리즘:

1. $F := 0$;
2. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복하라
   - (a) **선정 절차**: 적절한 최적해 선정절차에 따라서 하나의 이음선을 선정
   - (b) **적정성 점검**: 선정한 이음선을 F에 추가시켜도 순환경로가 생기지 않으면, F에 추가시킨다.
   - (c) **해답 점검**: $T = (V,F)$가 신장트리 이면, T가 최소비용신장트리 이다.

Prim의 알고리즘(추상적)

1. $F := 0$;
2. $Y := \{v_1\}$;
3. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복하라
   (a) **선정 절차/적정성 점검**: V - Y에 속한 정점 중에서, Y에 가장 가까운 정점 하나를 선정한다.
   (b) 선정한 정점을 Y에 추가한다.
   (c) Y로 이어지는 이음선을 F에 추가한다.
   (d) **해답 점검**: $Y = V$가 되면, $T = (V,E)$가 최소비용신장트리 이다.

2001 -03 -19 알고리즘 강의 슬라이드 4 15

Prim의 알고리즘 (세부적)

- 그래프의 인접행렬식 표현

$$W[i][j] = \begin{cases} \text{이음선의 가중치} & v_i\text{에서 } v_j\text{로의 이음선이 있다면} \\ \infty & v_i\text{에서 } v_j\text{로의 이음선이 없다면} \\ 0 & i = j \text{ 이면} \end{cases}$$

- 추가적으로 nearest[1..n]과 distance[1..n] 배열 유지

nearest[i] = Y에 속한 정점 중에서 v_i에서 가장 가까운 정점의 인덱스

distance[1..n] = v_i 와 nearest[i]를 잇는 이음선의 가중치

2001 -03 -19 알고리즘 강의 슬라이드 4 16

```
void prim(int n,              // 입력: 정점의 수
  const number W[][],         // 입력: 그래프의 인접행렬식 표현
  set_of_edges& F) {          // 출력: 그래프의 MST에 속한 이음선의 집합
  index i, vnear;
  number min;
  edge e;
  index nearest[2..n];
  number distance[2..n];
  F = empty_set;
  for(i=2; i <= n; i++) { // 초기화
    nearest[i] = 1;          // vi에서 가장 가까운 정점을 v1으로 초기화
    distance[i] = W[1][i];// vi과 v1을 잇는 이음선의 가중치로 초기화
  }
  repeat(n-1 times) {                  // n-1개의 정점을 Y에 추가한다
  min = "infinite";
  for(i=2; i <= n; i++)                // 각 정점에 대해서
    if (0 <= distance[i] <= min) {     // distance[i]를 검사하여
      min = distance[i];               // 가장 가까이 있는 vnear을
      vnear = i;                       // 찾는다.
    }
  e = vnear와 nearest[vnear]를 잇는 이음선;
  e를 F에 추가;
  distance[vnear] = -1;                // 찾은 노드를 Y에 추가한다.
  for(i=2; i <= n; i++)
    if (W[i][vnear] < distance[i]) {            // Y에 없는 각 노드에 대해서
      distance[i] = W[i][vnear];       // distance[i]를 갱신한다.
    }
  }
}
```

2001-03-19 알고리즘 강의 슬라이드 4 17

Prim의 알고리즘 분석

- 분석
  - 단위연산: repeat-루프 안에 있는 두 개의 for-루프 내부에 있는 명령문
  - 입력크기: 마디의 개수, n
  - 분석: repeat-루프가 n-1번 반복되므로
    - $T(n) = 2(n-1)(n-1) \in \Theta(n^2)$

2001-03-19 알고리즘 강의 슬라이드 4 18

최적여부의 검증(Optimality Proof)

- Prim의 알고리즘이 찾아낸 신장트리가 최소비용(minimal)인지를 검증해야 한다. 다시 말하면, Prim의 알고리즘이 최적(optimal)인지를 보여야 한다.
- **정의 4.1**: 비방향성 그래프 $G = (V,E)$가 주어지고, 만약 E의 부분집합 F에 MST가 되도록 이음선을 추가할 수 있으면, F는 유망하다(promising)라고 한다.
- **보조정리 4.1**: $G = (V,E)$는 연결되고, 가중치 포함 비방향성 그래프라고 하고, F는 E의 유망한 부분집합이라고 하고, Y는 F안에 있는 이음선 들에 의해서 연결이 되어 있는 정점의 집합이라고 하자. 이때, Y에 있는 어떤 정점과 $V - Y$에 있는 어떤 정점을 잇는 이음선 중에서 가중치가 가장 작은 이음선을 e라고 하면, $F \cup \{e\}$는 유망하다.

2001 -03 -19 알고리즘 강의 슬라이드 4 19

최적여부의 검증(Optimality Proof)

- 증명: F가 유망하기 때문에 $F \subseteq F'$이면서 (V,F')가 최소비용신장트리(MST)가 되는 이음선 F'가 반드시 존재한다.
  - 경우 1: 만일 $e \in F'$라면, $F \cup \{e\} \subseteq F'$가 되고, 따라서 $F \cup \{e\}$도 유망하다.
  - 경우 2: 만일 $e \notin F'$라면, (V,F')는 신장트리 이기 때문에, $F' \cup \{e\}$는 반드시 순환경로를 하나 포함하게 되고, e는 반드시 그 순환경로 가운데 한 이음선이 된다. 그러면 Y에 있는 한 정점에서 $V - Y$에 있는 한 정점을 연결하는 어떤 다른 이음선 $e' \in F'$가 그 순환경로 안에 반드시 존재하게 된다. 여기서 만약 $F' \cup \{e\}$에서 e'를 제거하면, 그 순환경로는 없어지게 되며, 다시 신장트리가 된다. 그런데 e는 Y에 있는 한 정점에서 $V - Y$에 있는 한 정점을 연결하는 최소의 가중치(weight)를 가진 이음선이기 때문에, e의 가중치는 반드시 e'의 가중치 보다 작거나 같아야 한다. (실제로 반드시 같게 된다.) 그러면 $F' \cup \{e\} - \{e'\}$는 최소비용신장트리(MST)이다. 결론적으로 e'는 F안에 절대로 속할 수 없으므로 (F안에 있는 이음선들은 Y안에 있는 정점들 만을 연결함을 기억하라), $F \cup \{e\} \subseteq F' \cup \{e\} - \{e'\}$가 되고, 따라서 $F \cup \{e\}$ 유망하다.

2001 -03 -19 알고리즘 강의 슬라이드 4 20

최적여부의 검증(Optimality Proof)

- **정리**: Prim의 알고리즘은 항상 최소비용신장트리를 만들어 낸다.

  증명: (수학적귀납법)

  매번 반복이 수행된 후에 집합 F가 유망하다는 것을 보이면 된다.
  - 출발점: 공집합은 당연히 유망하다.
  - 귀납가정: 어떤 주어진 반복이 이루어진 후, 그때까지 선정하였던 이음선의 집합인 F가 유망하다고 가정한다
  - 귀납절차: 집합 $F \cup \{e\}$가 유망하다는 것을 보이면 된다. 여기서 e는 다음 단계의 반복 수행 시 선정된 이음선 이다. 그런데, 위의 보조정리 1에 의하여 $F \cup \{e\}$은 유망하다고 할 수 있다. 왜냐하면 이음선 e는 Y에 있는 어떤 정점을 $V - Y$에 있는 어떤 정점으로 잇는 이음선 중에서 최소의 가중치를 가지고 있기 때문이다.

  증명 끝.

Kruskal의 알고리즘(추상적)

1. $F := 0$;
2. 서로소(disjoint)가 되는 V 의 부분집합 들을 만드는데, 각 부분집합 마다 하나의 정점만 가지도록 한다
3. E안에 있는 이음선을 가중치의 비내림차순으로 정렬한다
4. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복하라
   - (a) **선정 절차**: 다음 이음선을 선정한다.(최소의 가중치를 가진 이음선을 선정)
   - (b) **적정성 점검**: 만약 선정된 이음선이 두개의 서로소인 정점을 잇는다면, 먼저 그 부분집합을 하나의 집합으로 합하고, 그 다음에 그 이음선을 F에 추가한다.
   - (d) **해답 점검**: 만약 모든 부분집합이 하나의 집합으로 합하여 지면, 그 때 $T = (V,F)$가 최소비용신장트리 이다.

Kruskal의 알고리즘 (세부적)

- **서로소 집합 추상데이타타입(disjoint set abstract data type)**

index i;
set_pointer p, q;
initial(n): n개의 서로소 부분집합을 초기화
(하나의 부분집합에 1에서 n사이의 인덱스가 정확히 하나 포함됨)
p = find(i): 인덱스 i가 포함된 집합의 포인터 p를 넘겨줌
merge(p, q): 두 개의 집합을 가리키는 p와 q를 합병
equal(p, q): p와 q가 같은 집합을 가리키면 true를 넘겨줌

2001 -03 -19 알고리즘 강의 슬라이드 4 23

```
void kruskal(int n, int m,      // 입력: 정점의 수 n, 이음선의 수 m
             set_of_edges E,    // 입력: 가중치를 포함한 이음선의 집합
             set_of_edges& F) {// 출력: MST를 이루는 이음선의 집합
  index i, j;
  set_pointer p, q;
  edge e;
  E에 속한 m개의 이음선을 가중치의 비내림차순으로 정렬;
  F = emptyset;
  initial(n);
  while (F에 속한 이음선의 개수가 n-1보다 작다) {
    e = 아직 점검하지 않은 최소의 가중치를 가진 이음선;
    i, j = e를 이루는 양쪽 정점의 인덱스;
    p = find(i);
    q = find(j);
    if (!equal(p,q)) {
      merge(p,q);
      e를 F에 추가;
    }
  }
}
```

2001 -03 -19 알고리즘 강의 슬라이드 4 24

Kruskal의 알고리즘의 분석

- 분석
  - 단위연산: 비교문
  - 입력크기: 정점의 수 n과 이음선의 수 m

    1. 이음선 들을 정렬하는데 걸리는 시간: $\Theta(m \lg m)$

    2. 반복문 안에서 걸리는 시간: 루프를 m번 수행한다. 서로소인 집합 자료구조(disjoint set data structure)를 사용하여 구현하고, find, equal, merge 같은 동작을 호출하는 횟수가 상수이면, m번 반복에 대한 시간복잡도는 $\Theta(m \lg m)$이다.

    3. N개의 서로소인 집합(disjoint set)을 초기화하는데 걸리는 시간: $\Theta(n)$
  - 그런데 여기서 $m \geq n - 1$이기 때문에, 위의 1과 2는 3을 지배하게 되므로, $W(m, n) = \Theta(m \lg m)$가 된다.
  - 그러나, 최악의 경우에는, 모든 정점이 다른 모든 정점과 연결이 될 수 있기 때문에, $m = \frac{n(n-1)}{2} \in \Theta(n^2)$가 된다. 그러므로, 최악의 경우의 시간복잡도는

    $$W(m, n) \in \Theta(n^2 \lg n^2) = \Theta(2n^2 \lg n) = \Theta(n^2 \lg n)$$
  - 최적여부의 검증(Optimality Proof)
    - Prim의 알고리즘의 경우와 비슷함. (교재 참조)

2001 -03 -19 알고리즘 강의 슬라이드 4 25

두 알고리즘의 비교

	$W(m,n)$	sparse graph	dense graph
Prim Kruskal	$\Theta(n^2)$ $\Theta(m \lg m)$ and $\Theta(n^2 \lg n)$	$\Theta(n^2)$ $\Theta(n \lg n)$	$\Theta(n^2)$ $\Theta(n^2 \lg n)$

- 연결된 그래프에서의 m은 $n-1 \leq m \leq \frac{n(n-1)}{2}$ 의범위를 갖는다.

2001 -03 -19 알고리즘 강의 슬라이드 4 26

토론사항

- 알고리즘의 시간복잡도는 그 알고리즘을 구현하는데 사용하는 자료구조에 좌우되는 경우도 있다.

Prim 의 알고리즘	$W(m,n)$	sparse graph	dense graph
Heap Fibonacci heap	$\Theta(m \lg n)$ $\Theta(m + n \lg n)$	$\Theta(n \lg n)$ $\Theta(n \lg n)$	$\Theta(n^2 \lg n)$ $\Theta(n^2)$

2001 -03 -19 알고리즘 강의 슬라이드 4 27

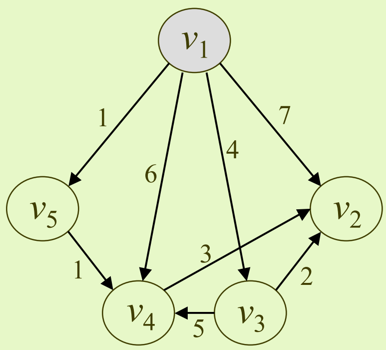
단일출발점 최단경로 문제 Dijkstra의 알고리즘

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

1. $F := 0$;
2. $Y := \{v_1\}$;
3. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복하라
   - (a) **선정 절차/적정성 점검**: $V - Y$에 속한 정점 중에서, v_1에서 Y에 속한 정점 만을 거쳐서 최단경로가 되는 정점 v를 선정한다
   - (b) 그 정점 v를 Y에 추가한다.
   - (c) v에서 F로 이어지는 최단경로 상의 이음선을 F에 추가한다.
   - (d) **해답 점검**: $Y = V$가 되면, $T = (V,E)$가 최단경로를 나타내는 그래프이다.

2001 -03 -19 알고리즘 강의 슬라이드 4 28

보기



Dijkstra의 알고리즘 분석

- 분석
  - ✓ $T(n) \in \Theta(n^2)$. 힙(heap)으로 구현하면 $\Theta(m \lg n)$이고, 피보나찌 힙으로 구현하면 $\Theta(m + n \lg n)$이다.
- 최적여부의 검증(Optimality Proof)
  - ✓ Prim의 알고리즘의 경우와 비슷함.

탐욕적인 방법과 동적계획법의 비교

탐욕적인 접근방법	동적계획법
최적화 문제를 푸는데 적합	최적화 문제를 푸는데 적합
알고리즘이 존재할 경우 보통 더 효율적	때로는 불필요하게 복잡
알고리즘이 최적인지를 증명해야 함	최적화 원칙이 적용되는지를 점검해 보기만 하면 됨
단일출발점 최단경로 문제: $\Theta(n^2)$	단일출발점 최단경로 문제: $\Theta(n^3)$
배낭 빈틈없이 채우기 문제는 풀지만, 0-1 배낭 채우기 문제는 풀지 못함	0-1 배낭 채우기 문제를 푼다

0-1배낭 채우기 문제 (0-1 Kanpsack Problem)

- 문제: $S = \{item_1, item_2, \ldots, item_n\}$

  $w_i = item_i$의 무게

  $p_i = item_i$의 가치

  $W =$ 배낭에 넣을 수 있는 총 무게

  라고 할 때,

  $\sum_{item_i \in A} w_i \le W$를 만족하면서 $\sum_{item_i \in A} p_i$가 최대가 되도록 $A \subseteq S$가 되는 A를 결정하는 문제이다.
- 무작정 (탐욕적) 알고리즘
  - ✓ n개의 물건에 대해서 모든 부분집합을 다 고려한다.
  - ✓ 그러나 불행하게도 크기가 n인 집합의 부분집합의 수는 2^n개 이다.

0-1배낭 채우기 문제: 탐욕적 알고리즘 I

- 가장 비싼 물건부터 우선적으로 채운다.
- 애석하게도 이 알고리즘은 최적이 아니다!
- 왜 아닌지 보기: $W = 30$kg -

품목	무게	값
$item_1$	25kg	10 만원
$item_2$	10kg	9 만원
$item_3$	10kg	9 만원

✓ 탐욕적인 방법: $item_1 \Rightarrow 25\text{kg} \Rightarrow 10$만원
✓ 최적인 해답: $item_2 + item_3 \Rightarrow 20\text{kg} \Rightarrow 18$만원

2001 -03 -19 알고리즘 강의 슬라이드 4 33

0-1배낭 채우기 문제: 알고리즘 II

- 무게 당 가치가 가장 높은 물건부터 우선적으로 채운다.
- 그래도 최적이 아니다!
- 왜 아닌지 보기: $W = 30$kg -

품목	무게	값	값어치
$item_1$	5kg	50 만원	10 만원/kg
$item_2$	10kg	60 만원	6 만원/kg
$item_3$	20kg	140 만원	7 만원/kg

✓ 탐욕적인 방법: $item_1 + item_3 \Rightarrow 25\text{kg} \Rightarrow 190$만원
✓ 최적인 해답: $item_2 + item_3 \Rightarrow 30\text{kg} \Rightarrow 18$만원

2001 -03 -19 알고리즘 강의 슬라이드 4 34

배낭 빈틈없이 채우기 문제 (The Fractional Knapsack Problem)

- 물건의 일부분을 잘라서 담을 수 있다.
- 탐욕적인 접근방법으로 최적해를 구하는 알고리즘을 만들 수 있다.
- $item_1 + item_3 + item_2 * 1/2 \Rightarrow 30\text{kg} \Rightarrow 220$만원
- 최적이다! 증명해 보라. 숙제?

0-1 배낭채우기 문제의 동적계획적인 접근방법

- $i > 0$ 이고 $w > 0$일 때, 전체 무게가 w가 넘지 않도록 i번째 까지의 항목 중에서 얻어진 최고의 이익(optimal profit)을 $P[i][w]$라고 하면,

$$P[i][w] = \begin{cases} maximum\,(P[i-1][w],\ p_i + P[i-1][w-w_i]) & \text{if } w_i \le w \\ P[i-1][w] & \text{if } w_i > w \end{cases}$$

  여기서 $P[i\text{-}1][w]$는 i번째 항목을 포함시키지 않는 경우의 최고 이익이고, $p_i + P[i - 1][w - w_i]$는 i번째 항목을 포함시키는 경우의 최고 이익이다. 위의 재귀 관계식이 최적화 원칙을 만족하는지는 쉽게 알 수 있다.
- 그러면 어떻게 $P[n][W]$값을 구할 수 있을까? 다음과 같은 2차원 배열을 만든 후, 각 항을 계산하여 채워 넣으면 된다: int P[0..n][0..W]. 여기서 $P[0][w] = 0$, $P[i][0] = 0$으로 놓으면 되므로, 계산해야 할 항목의 수는 $nW \in \Theta(nW)$

0-1 배낭채우기 문제의 동적계획적인 접근방법

- 여기서 n과 W와는 아무런 상관관계가 없다. 따라서, $W = n!$이라고 한다면 수행시간은 $\Theta(n \times n!)$이 된다. 그렇게 되면 이 알고리즘은 앞에서 얘기한 무작정 알고리즘보다도 나을게 하나도 없다.
- 그럼 이 알고리즘을 최악의 경우에 $\Theta(2^n)$ 시간에 수행될 수 있도록, 즉 무작정 알고리즘 보다 느리지 않고, 때로는 훨씬 빠르게 수행될 수 있도록 개량할 수 있을까? 착안점은 $P[n][W]$를 계산하기 위해서 $(n\text{-}1)$번째 행을 모두 계산할 필요가 없다는데 있다. 즉, $P[n\text{-}1][W]$와 $P[n\text{-}1][W\text{-}w_n]$ 두 항만 계산하면 된다. 이런 식으로 $n = 1$이나 $w \le 0$일 때 까지 계속해 나가면 된다.

2001 -03 -19 알고리즘 강의 슬라이드 4 37

0-1 배낭채우기 문제의 동적계획적인 접근방법

- 예로서 위의 예에서 $P[3][30]$을 계산해 보자. 개량 알고리즘은 다음과 같이 7개 항만 계산하는데 비해서, 이전 알고리즘은 $3 \times 30 = 90$항을 계산해야 한다.

$$P[3][30] = max(P[2][30], 140 + P[2][10]) = max(110, 140 + 160) = 200$$
$$P[2][30] = max(P[1][30], 60 + P[1][20]) = max(50, 60 + 50) = 110$$
$$P[2][10] = max(P[1][10], 60 + P[1][0]) = max(50, 60 + 0) = 60$$
$$P[1][0] \quad = 0$$
$$P[1][10] = 50$$
$$P[1][20] = 50$$
$$P[1][30] = 50$$

2001 -03 -19 알고리즘 강의 슬라이드 4 38

0-1 배낭채우기 문제의 동적계획적인 접근방법

- 그러면 개량 알고리즘의 최악의 경우 수행시간을 계산해 보자. $(n - i)$번째 행에서 기껏해야 2^i 항을 계산하므로, 총 계산하는 항수는 $1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$이 된다. 따라서 $\Theta(2^n)$가 된다. 위의 두 가지 경우를 합하면 최악의 경우의 수행시간은 $O(\text{minimum}(2^n, nW))$이다.
- 분할정복 방법으로도 이 알고리즘을 설계할 수도 있고, 그 최악의 경우 수행시간은 $\Theta(2^n)$이다. 아직 아무도 이 문제의 최악의 경우 수행시간이 지수(exponential)보다 나은 알고리즘을 발견하지 못했고, 아직 아무도 그러한 알고리즘은 없다라고 증명한 사람도 없다. - **NP문제**

2001 -03 -19 알고리즘 강의 슬라이드 4 39