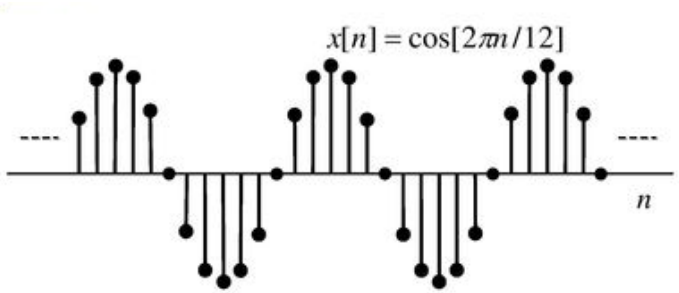
이산적 선택지가 존재하는 쿠르노 모델(Cournot model)의 최적대응 기반 풀이 기법의 개발: 큐(Queue)와 체이닝(Chaining)을 중심으로

Development of Best Response-Based Solving Technique for Cournot Model with Discrete Options: Focusing on Queue and Chaining

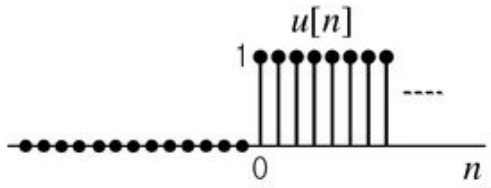
2203 김석준

Index

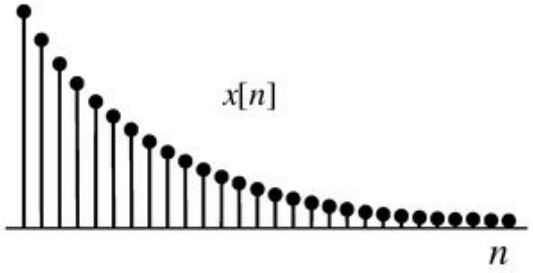
- 연구의 동기 및 목적
- 선행연구의 배경 및 분석
  - 큐(Queue)
  - 체이닝(Chaining)
  - CCW Algorithm
- 이론적 배경
  - Payoff matrix
  - Nash Equilibrium
  - Best response
  - Cournot model
- 주요 알고리즘 및 프로그램 분석
- 연구 결과 및 고찰



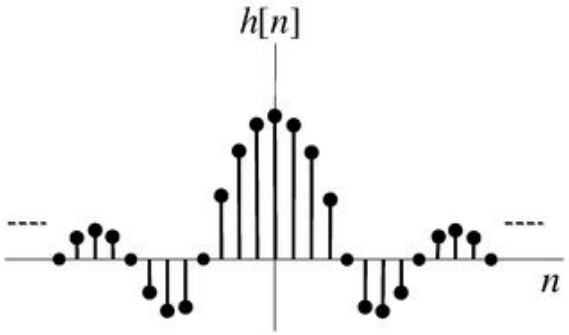
(a) 이산 정현파 함수



(c) 이산 계단 함수



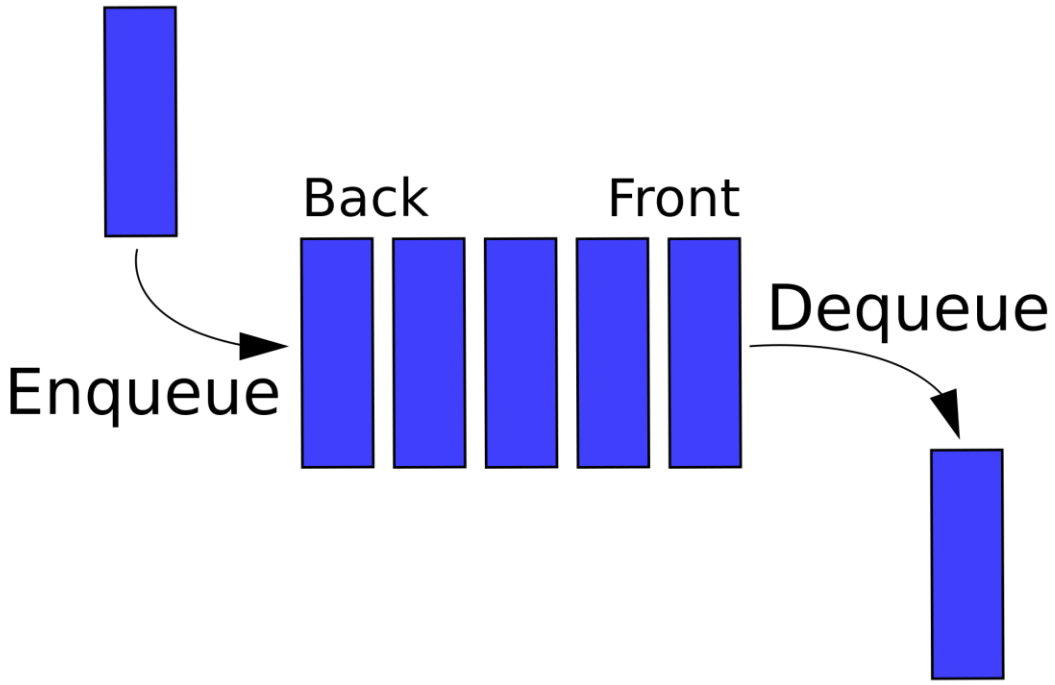
(b) 이산 지수 함수



(d) 이산 sinc 함수

연구의 동기 및 목적

- 현실에서는 연속적 데이터보다 불연속적 데이터가 더 많이 존재
- Cournot model은 연속적 변수에서만 적용 가능
- 현실에 더 적합한 형태를 갖게 하기 위해 본 프로그램을 진행



선행연구의 배경 및 분석

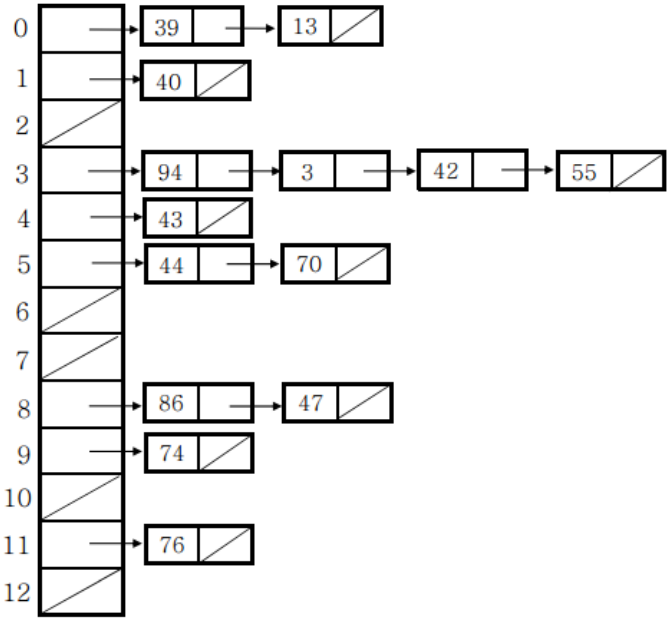
● 큐(Queue)

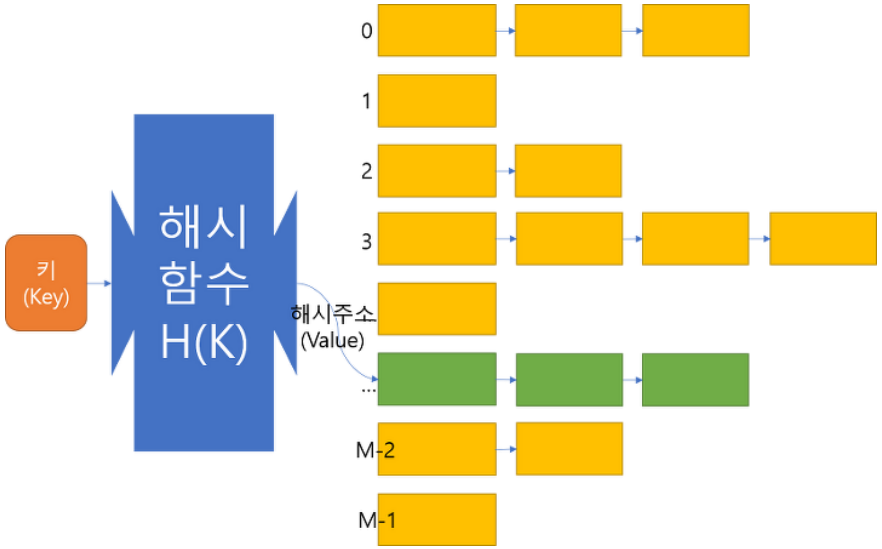
- 선형 자료구조
- 선입선출(FIFO)

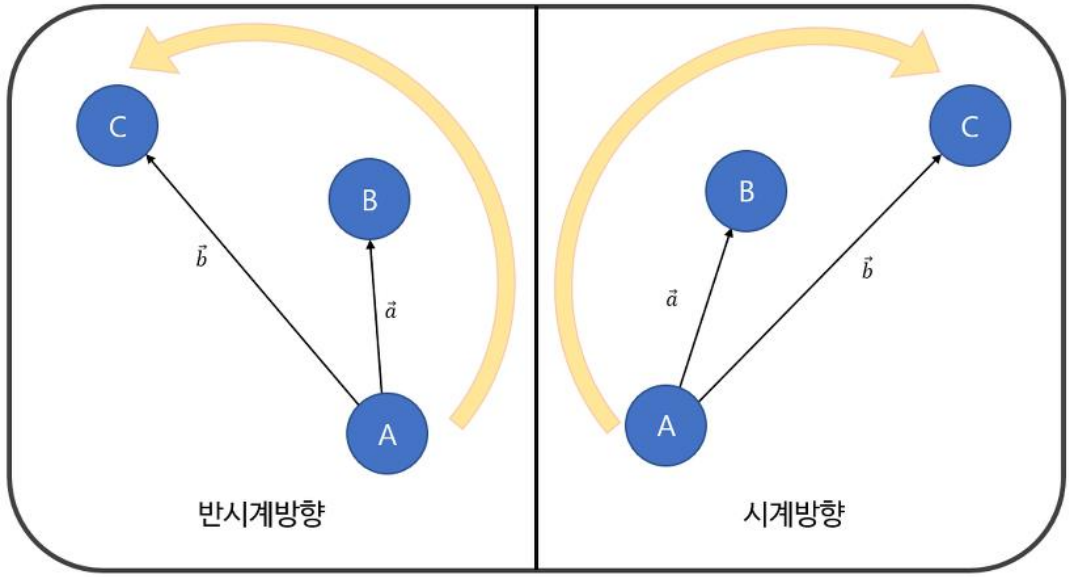
```
ADT Queue

데이타 : 0개 이상의 원소를 가진 유한 순서 리스트

연산 :
    Q ∈ Queue; item ∈ Element;
    createQ() ::= create an empty queue;
    enqueue(Q, item) ::= insert item at the rear of Q;
    isEmpty(Q) ::= if (Q is empty) then return true
                   else return false;
    dequeue(Q) ::= if (isEmpty(Q)) then return null
                   else remove and return the front element of Q;
    remove(Q) ::= if (isEmpty(Q)) then return null
                  else remove the front element of Q;
    peek(Q) ::= if (isEmpty(Q)) then return null
                else return the front element of Q;
End Queue
```

● 체이닝(Chaining)

- 해시 테이블(Hash Table)의 충돌(Collision) 해결 전략
- 각 원소별 연결리스트 할당(체인)

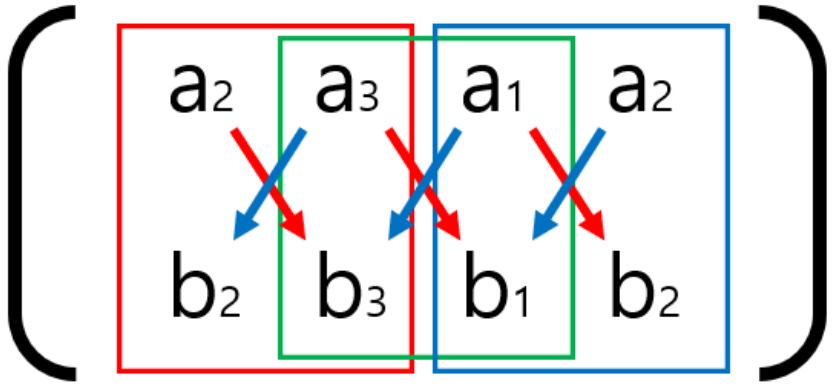






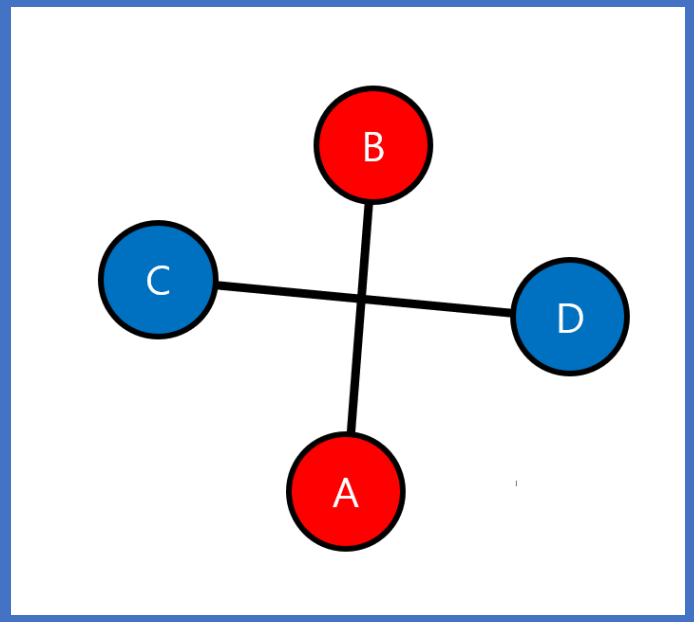
CCW Algorithm

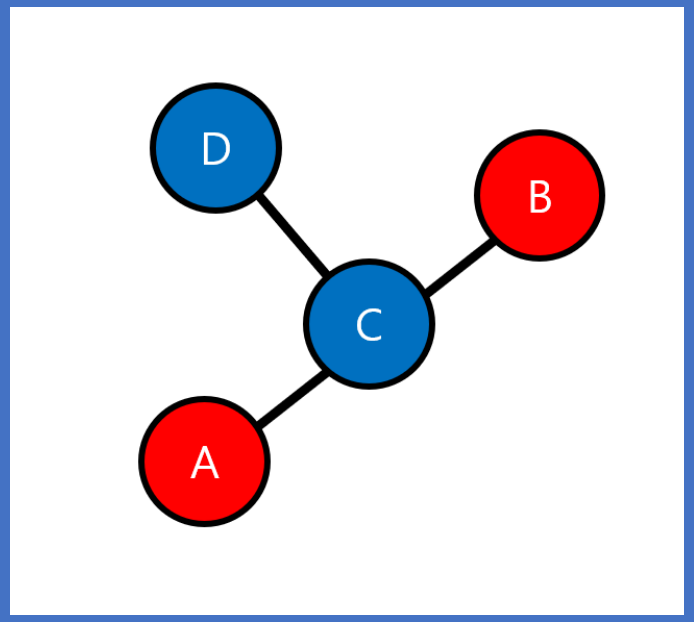
- 기하 알고리즘
- 외적(cross product)으로 점의 위치관계 판별
- 행렬 형태로 접근하면 외적 연산이 간편

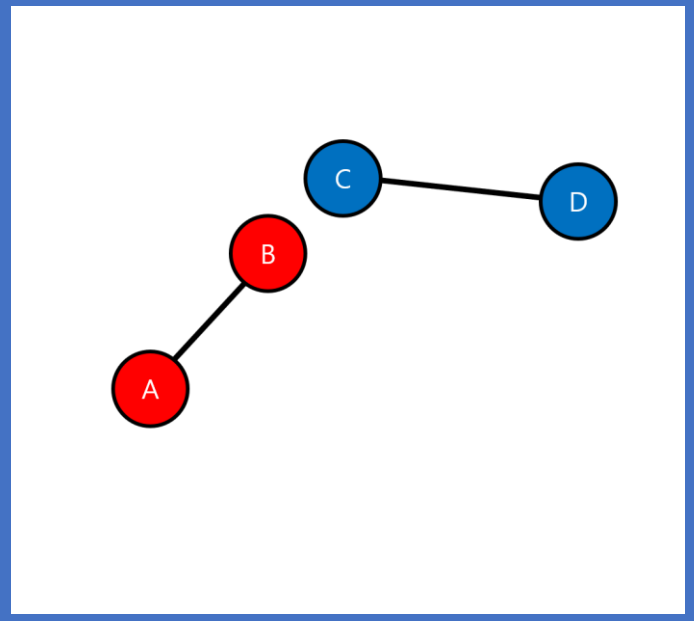


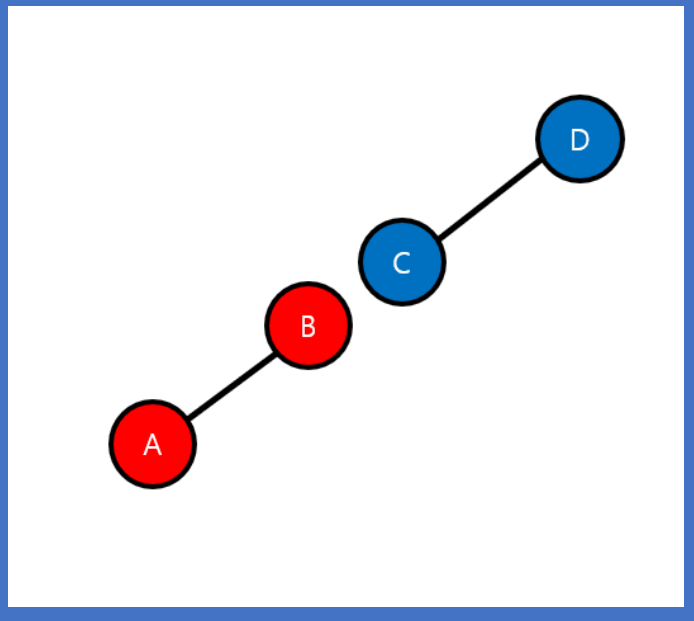
$$\vec{c} = (a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)$$

→CCW(A, B, C)









~~CCW(A, B, C) × CCW(A, B, D) < 0~~

~~CCW(A, B, C) × CCW(A, B, D) ≤ 0~~

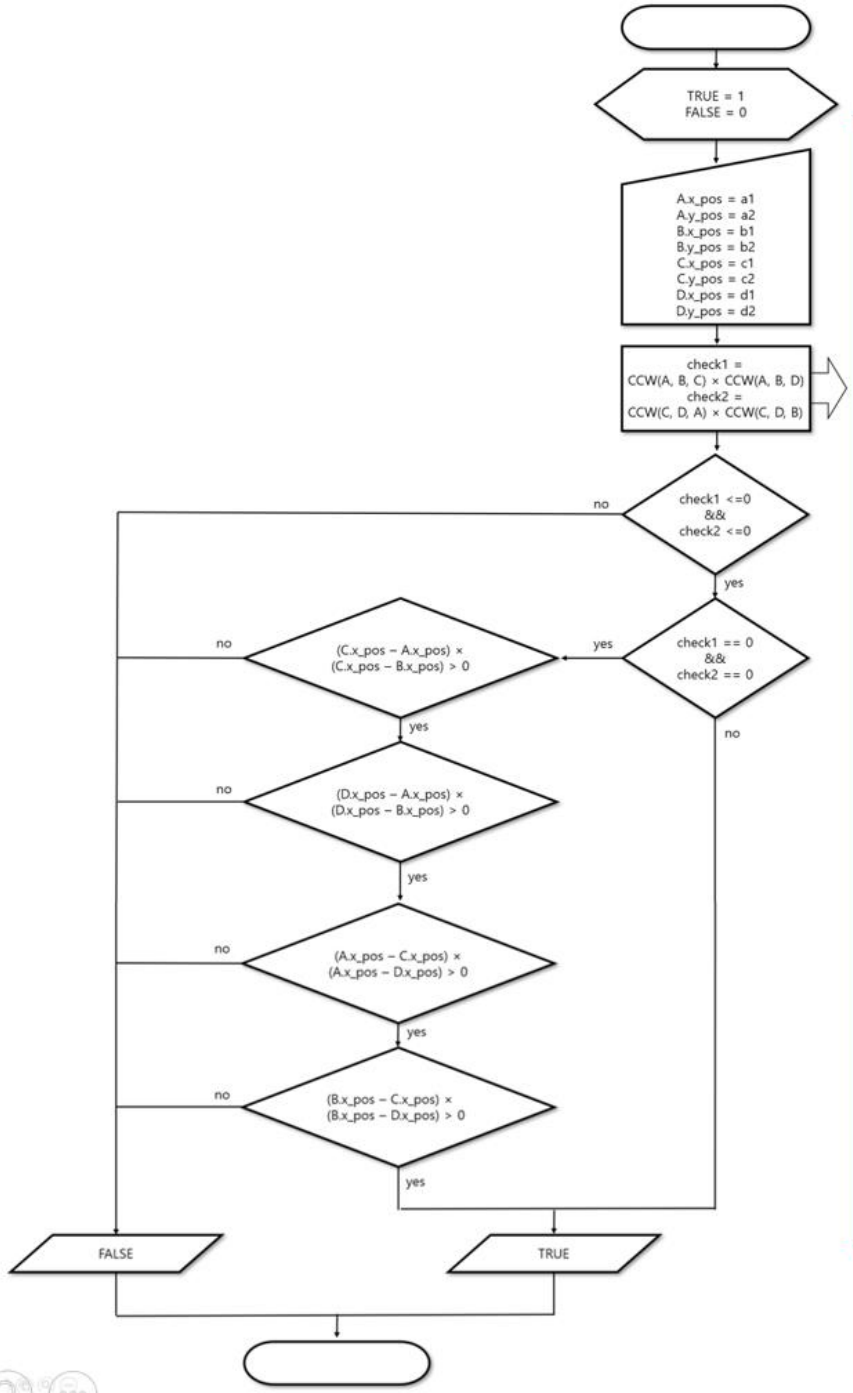
CCW(A, B, C) × CCW(A, B, D) ≤ 0

and

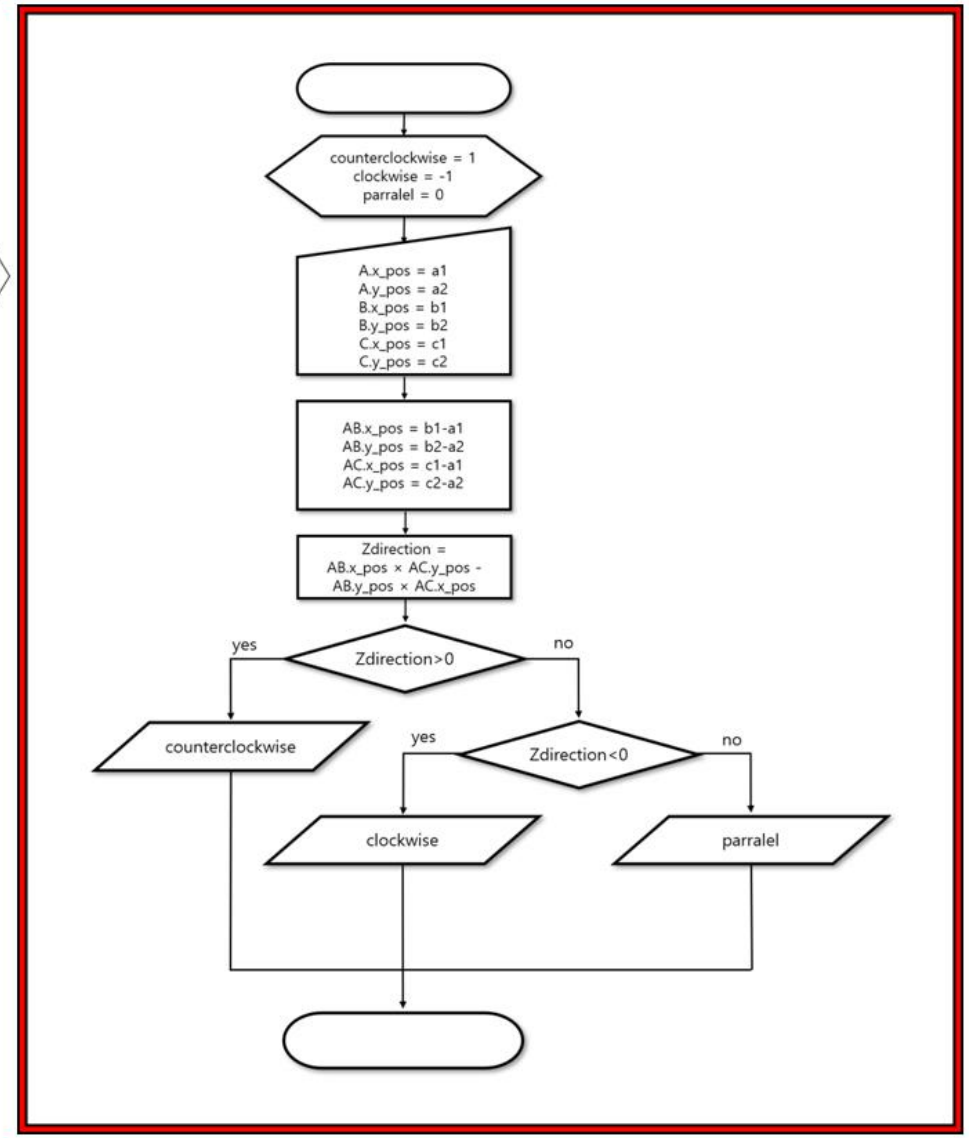
CCW(C, D, A) × CCW(C, D, B) ≤ 0

i. (C.x_pos – A.x_pos) × (C.x_pos – B.x_pos) > 0
ii. (D.x_pos – A.x_pos) × (D.x_pos – B.x_pos) > 0
iii. (A.x_pos – C.x_pos) × (A.x_pos – D.x_pos) > 0
iv. (B.x_pos – C.x_pos) × (B.x_pos – D.x_pos) > 0

*CCW based Cross (CCW 기반 선분 교차 여부 알고리즘)


TRUE = 1
FALSE = 0
A.x_pos = a1
A.y_pos = a2
B.x_pos = b1
B.y_pos = b2
C.x_pos = c1
C.y_pos = c2
D.x_pos = d1
D.y_pos = d2
check1 =
CCW(A, B, C) × CCW(A, B, D)
check2 =
CCW(C, D, A) × CCW(C, D, B)
check1 <=0
&&
check2 <=0
no
yes
check1 == 0
&&
check2 == 0
yes
no
(C.x_pos – A.x_pos) ×
(C.x_pos – B.x_pos) > 0
no
yes
(D.x_pos – A.x_pos) ×
(D.x_pos – B.x_pos) > 0
no
yes
(A.x_pos – C.x_pos) ×
(A.x_pos – D.x_pos) > 0
no
yes
(B.x_pos – C.x_pos) ×
(B.x_pos – D.x_pos) > 0
no
yes
FALSE
TRUE

CCW

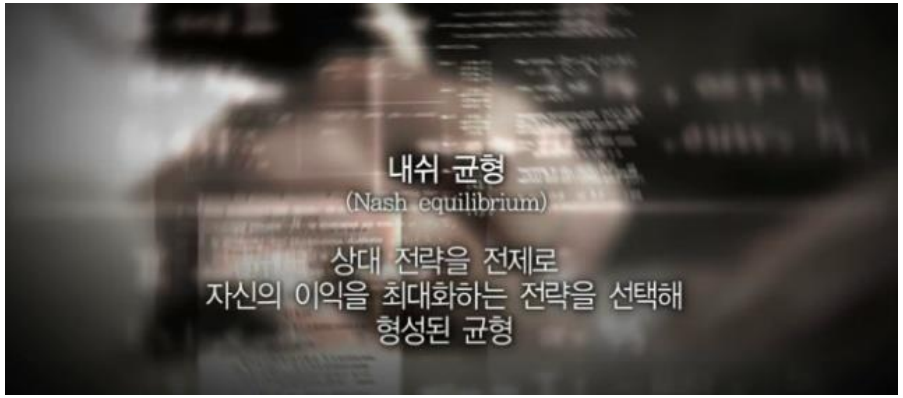

counterclockwise = 1
clockwise = -1
parralel = 0
A.x_pos = a1
A.y_pos = a2
B.x_pos = b1
B.y_pos = b2
C.x_pos = c1
C.y_pos = c2
AB.x_pos = b1-a1
AB.y_pos = b2-a2
AC.x_pos = c1-a1
AC.y_pos = c2-a2
Zdirection =
AB.x_pos × AC.y_pos -
AB.y_pos × AC.x_pos
Zdirection>0
yes
no
counterclockwise
Zdirection<0
yes
no
clockwise
parralel

이론적 배경

- Payoff matrix(보수행렬)

		player2	
		H	T
player1	H	1, 1	4, 0
	T	0, 4	3, 3

Nash Equilibrium(내쉬 균형)



s_1*이 $^{\forall}s_1$에 대해 다음을 만족하면 이 전략 s_1*은 $player1$에게 있어 우월전략이다:

$$u_1(s_1^*, s_2) \geq u_1(s_1, s_2)$$

2인 게임에서 다음 조건을 만족하면 전략명세(전략쌍) (s_1^*, s_2^*)는 Nash Equilibrium이다:

ⅰ. $^{\forall}s_1 \in S_1$에 대해 $u_1(s_1^*, s_2^*) \geq u_1(s_1, s_2^*)$가 성립

ⅱ. $^{\forall}s_2 \in S_2$에 대해 $u_2(s_1^*, s_2^*) \geq u_2(s_1^*, s_2)$가 성립

- Best response(최적 대응)

주어진 player2의 전략 s_2에 대한 player1의 최적대응 $BR_1(s_2)$는 $u_1(s_1, s_2)$를 극대화하는 player1의 전략 s_1을 가리킨다. 즉, $BR_1(s_2) \in S_1$은 다음을 만족하는 전략이다:

$^{\forall}s_1 \in S_1$에 대해 $u_1(BR_1(s_2), s_2) \geq u_1(s_1, s_2)$가 성립

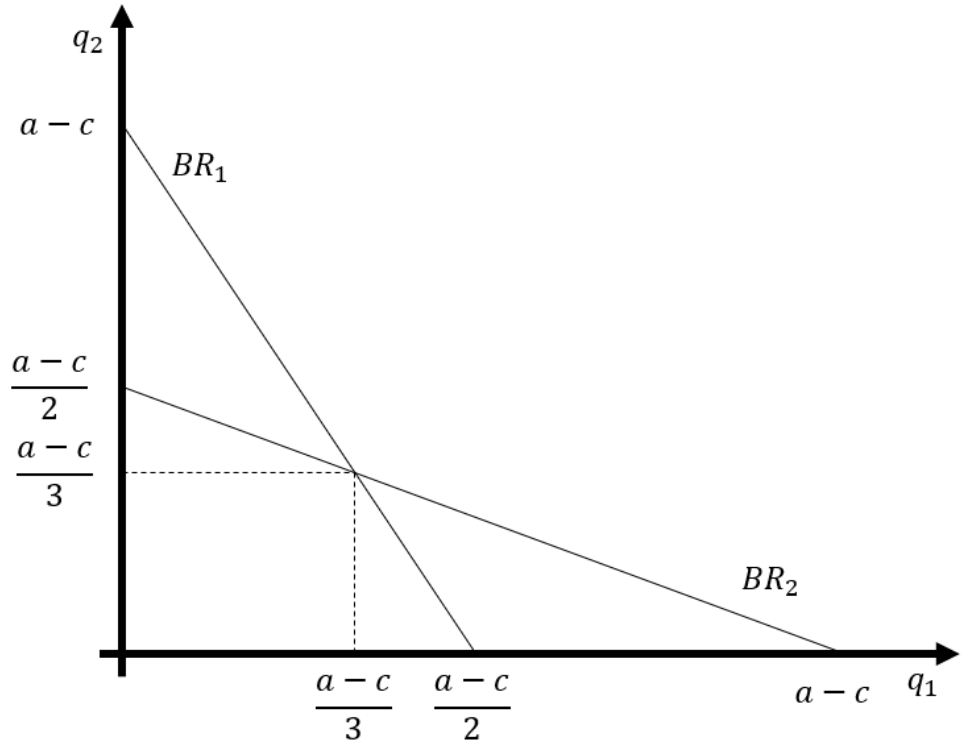
—

		player2	
		H	T
player1	H	1, 1	4, 0
	T	0, 4	3, 3

Cournot model(쿠르노 모델)

- payoff 행렬의 연속변수 형태
- 각 player(기업)은 상대 player의 생산량에 따라 최대의 이윤을 내는 생산량을 결정

  =Best response의 연속적 형태



주어진 player2의 생산량 q_2에 대해 player1이 q_1을 생산할 시 player1의 이윤:

$$G(q_1, q_2) = Pq_1 - cq_1 = \{a - (q_1 + q_2)\}q_1 - cq_2$$
$$= \{a - c - (q_1 + q_2)\}q_1$$ (P: 시장역수요, c: 비용함수의 비례상수)

q_1에 대한 편미분:

$$\frac{\partial}{\partial q_1} G(q_1, q_2) = \frac{\partial}{\partial q_1}\{a - c - (q_1 + q_2)\}q_1$$
$$= a - c - q_2 - 2q_1 = 0$$

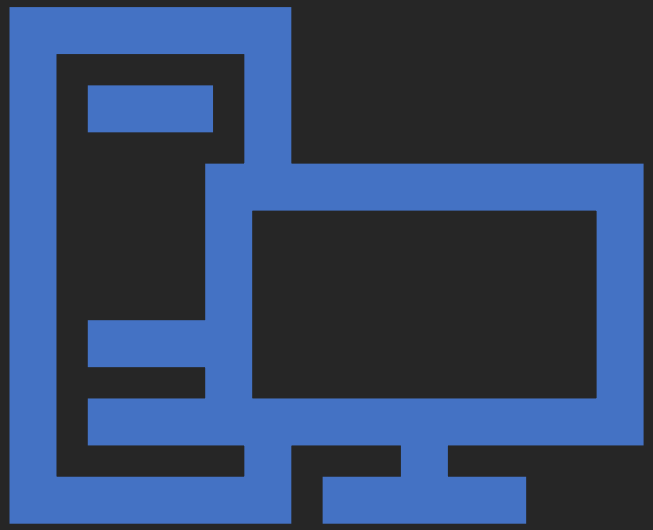
$$q_1 = \frac{a - c - q_2}{2} = -\frac{1}{2}q_2 + \frac{a - c}{2} = BR_1(q_2)$$

$$BR_1(q_2) = -\frac{1}{2}q_2 + \frac{a - c}{2}$$

$$BR_2(q_1) = -\frac{1}{2}q_1 + \frac{a - c}{2}$$

$$q_1{}^* = q_2{}^* = \frac{a - c}{3}$$

주요 알고리즘 및 프로그램 분석

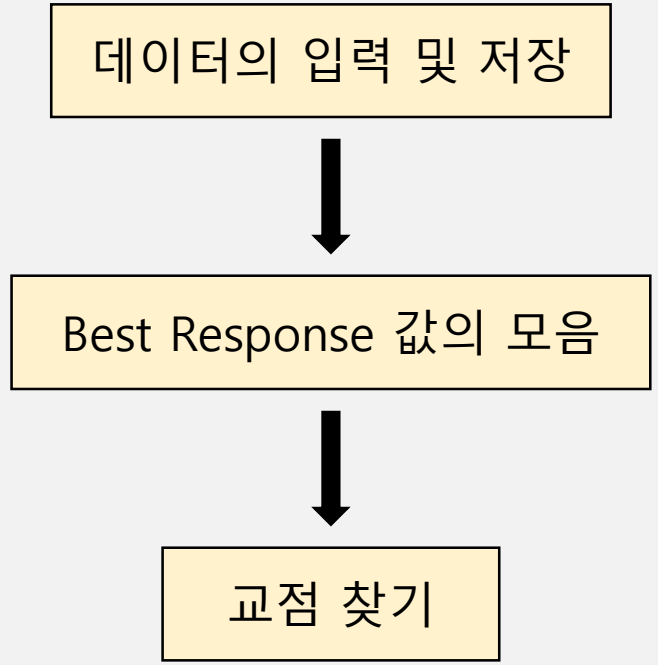


- **변수 분포는 완전히 연속적으로 표현 불가**

-추세선을 나타내는 데에 어려움
-추가적 알고리즘 및 시간 필요

- **실생활에서의 데이터는 다양한 상황과 그에 따른 결과의 이산적 값으로 표현**

-이산적 데이터의 축적으로 정확한 값을 찾아나가는 과정이 효율적



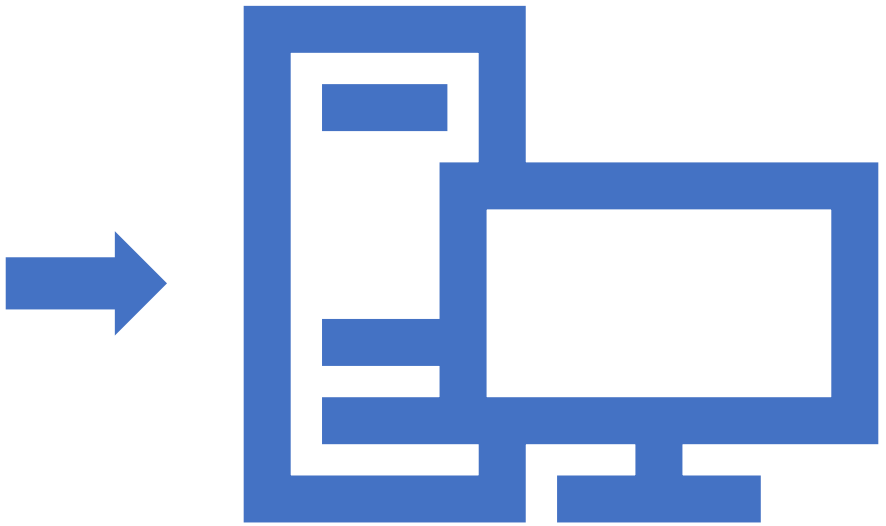
데이터의 입력 및 저장

player2의 생산량에 따른 player1의 생산량 및 이윤

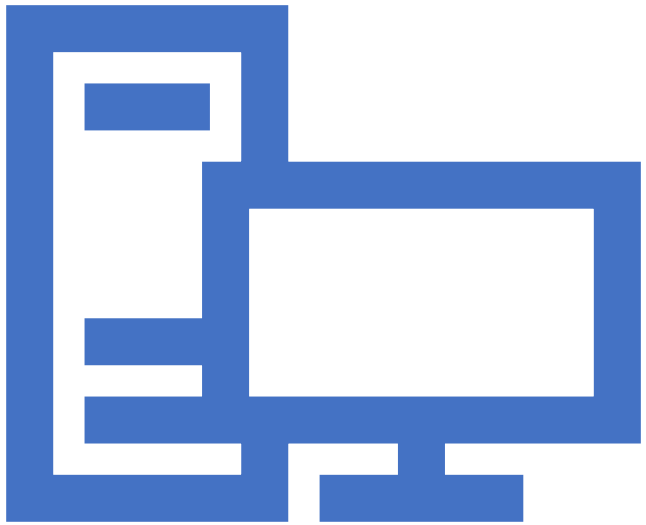
- product_player2
- product_player1
- profit_player1

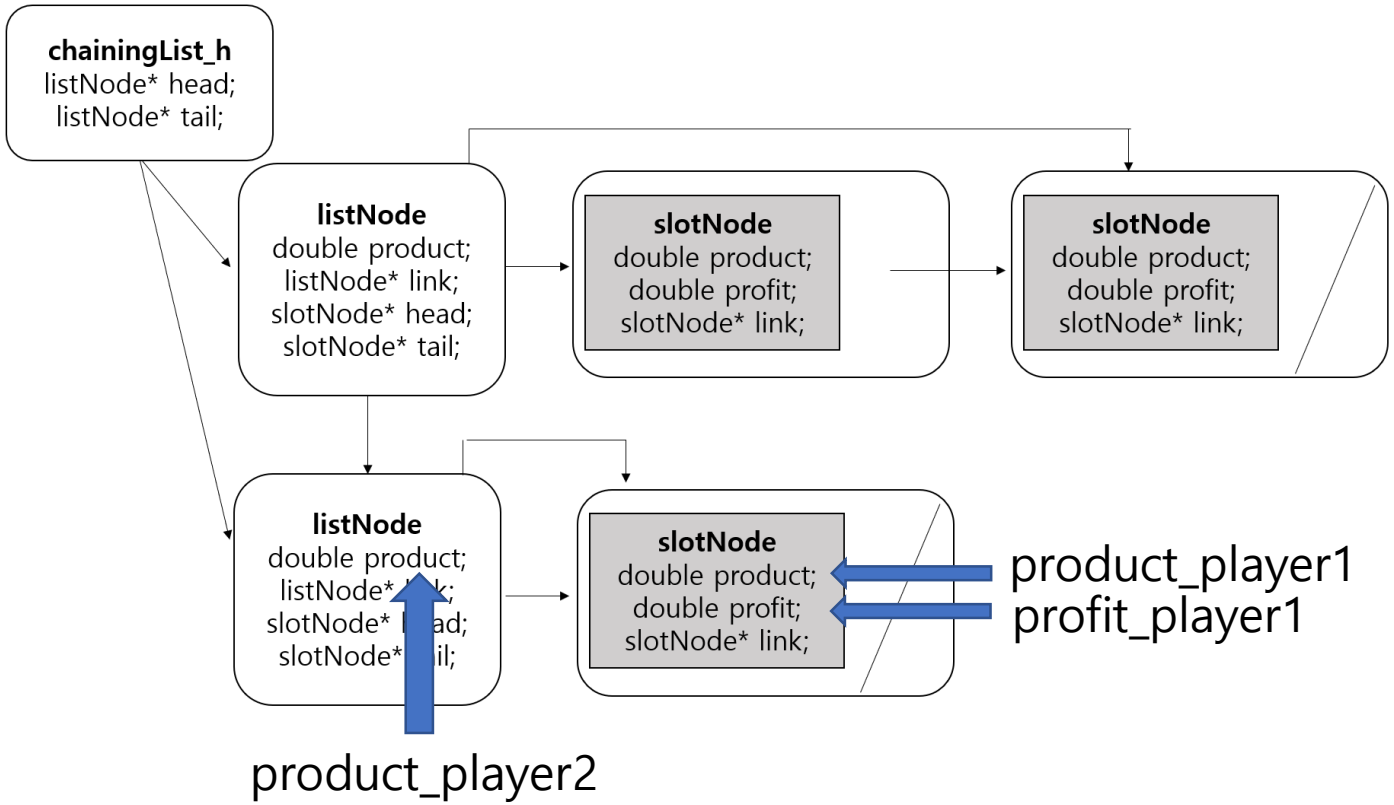
player1의 생산량에 따른 player2의 생산량 및 이윤

- product_player1
- product_player2
- profit_player2



e.g. player2의 생산량에 따른 player1의 생산량 및 이윤

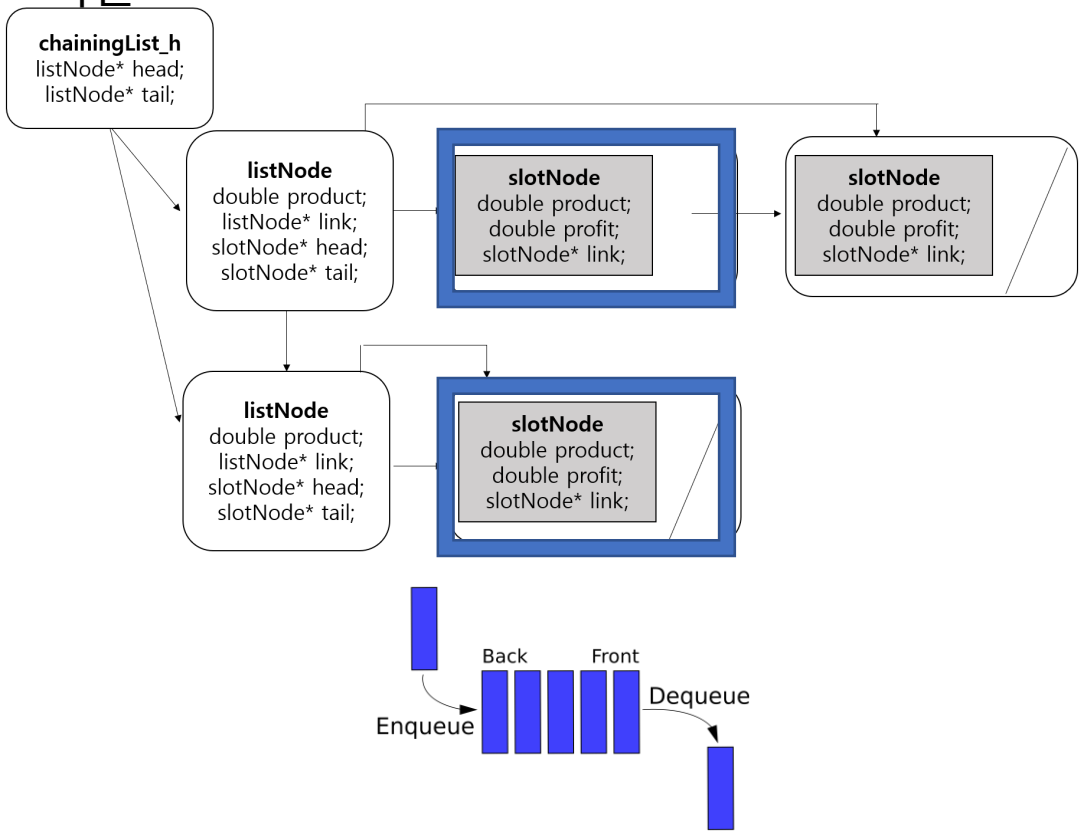




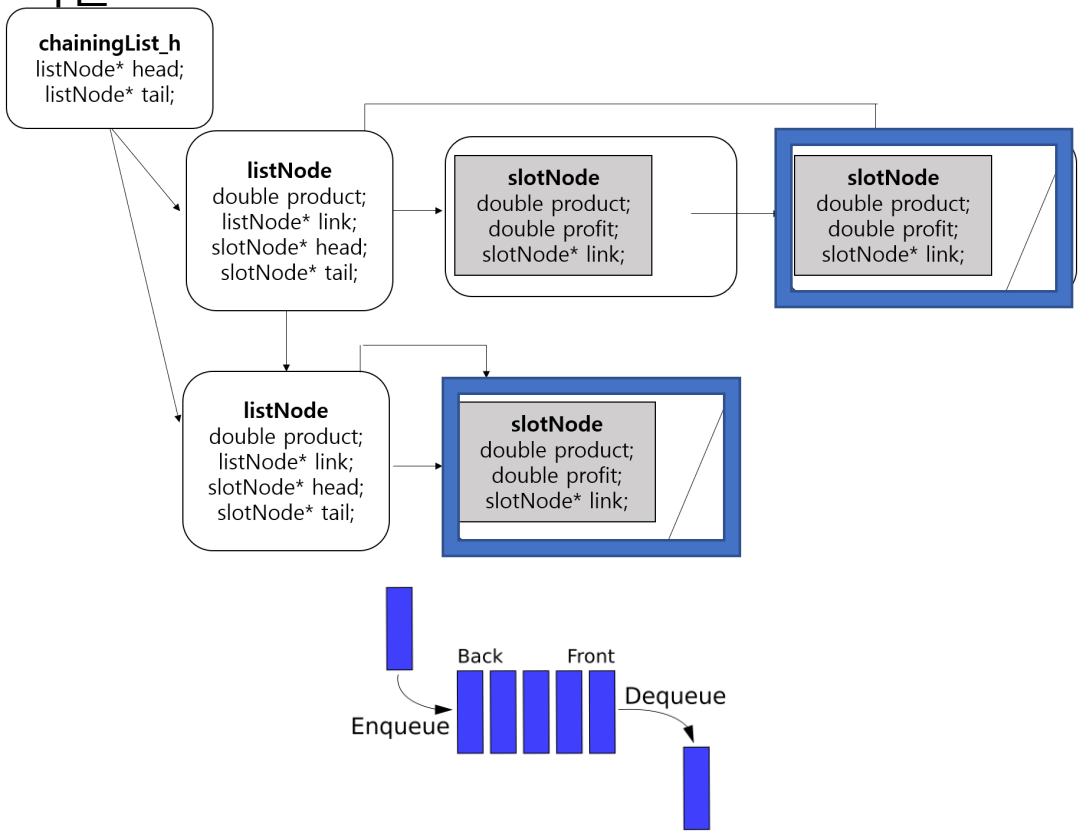
- product_other(product_player2) data가 존재X → listNode 할당
- 같은 product_other에 대해 product_mine 또는 profit_mine이 동일한 값 존재 → 입력 중지

Best Response 값의 모음

player2의 생산량에 따른 player1의 생산량 및 이유

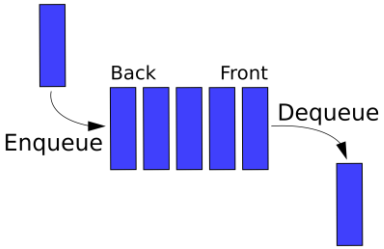


player1의 생산량에 따른 player2의 생산량 및 이유

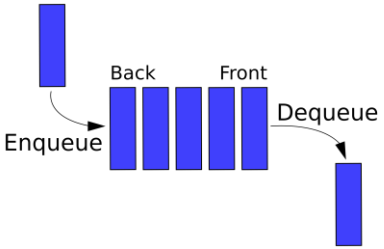


- chain별로 최대의 profit을 내는 product_mine를 찾음
- (product_player1, product_player2)의 형태로 각각 큐에 삽입 (listNode는 오름차순으로 정렬)

교점 찾기

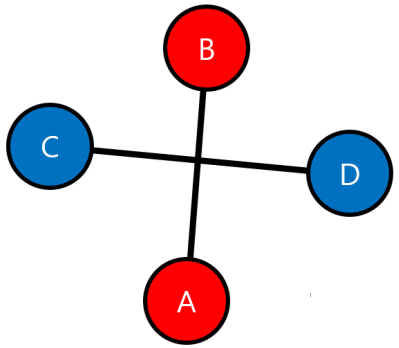


Dequeue
Peek



Dequeue
Peek

4개 원소(좌표)



- Cournot.c

createPayoff() ::= create empty payoff;

insertNewStrategy(L, product_other, product_mine, profit_mine) ::=
if isPossible() then insert new strategy in payoff L
and return TRUE(1);
else return FALSE(0);

deleteStrategy(L, product_other, product_mine) ::=
remove the element of L;

SortListNode(L) ::= sort listNode based on product_other;

ShowCross(L1, L2) ::= print intersection range based on CCW;

ShowPayoff(L) ::= print all strategy;

- ChainingBasedLinkedList.c

createChainingList_h() ::= create empty chainingList;

isPossible(L, product_other, product_mine, profit_mine) ::=
if valid data then return TRUE(1);
else return FALSE(0);

insertSlotNode(L, product_other, product_mine, profit_mine) ::=
if isPossible() then insert new slot in chaining list L;
return TRUE(1);
else return FALSE(0);

deleteSlotNode(L, product_other, product_mine) ::=
remove the element of L;

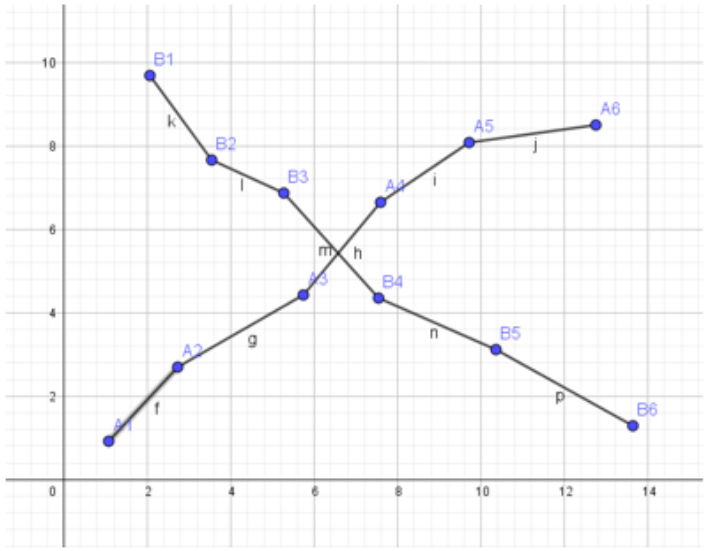
- CCW.c

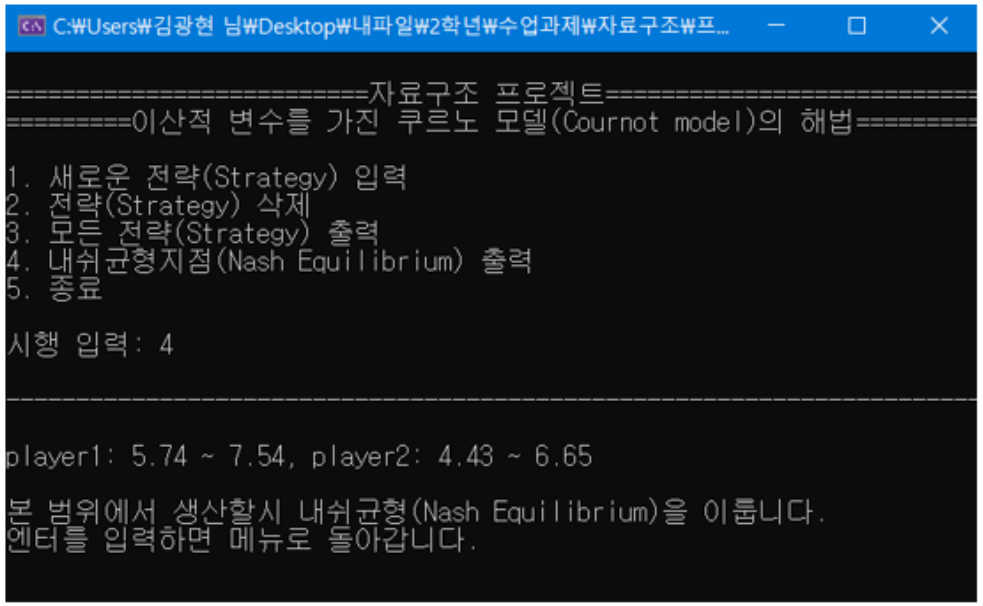
CCW(A, B, C) ::= return sign of cross product AB, AC;

CCWbaseCross(A, B, C, D) ::= if AB and CD intersected then return TRUE(1)
else return FALSE(0)

연구 결과 및 고찰

- A1 = (1.07, 0.92)
- A2 = (2.73, 2.7)
- A3 = (5.74, 4.43)
- A4 = (7.59, 6.65)
- A5 = (9.71, 8.08)
- A6 = (12.75, 8.5)
- B1 = (2.06, 9.68)
- B2 = (3.54, 7.66)
- B3 = (5.27, 6.87)
- B4 = (7.54, 4.35)
- B5 = (10.35, 3.12)
- B6 = (13.64, 1.29)





- chain별로 최대의 profit을 내는 product_mine을 포함한 좌표
- 교점, 즉 Nash Equilibrium이 나타나는 부분을 기록
- 데이터가 많아진다면 범위 및 오차 또한 감소할 것

감사합니다

- 출처

-나무위키
-덕’s IT Story
-데구리 블로그
-지식채널 e