2020년 가을학기 알고리즘

Graph++

데이터네트워크연구실
문현수, 이영석
munhyunsu@cs-cnu.org

11주차 Feedback

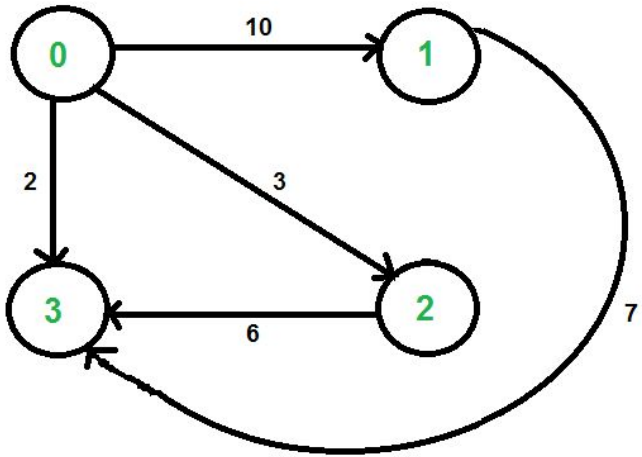
- 제출률: 75%

이번주 실습 목표

- 그래프++
  - Topology Sort (Khan’s Algorithm)
  - Network Flow (Ford–Fulkerson Algorithm)
  - Shortest Path (A* Algorithm)

- 실습1: Prerequisite (3점)
- 실습2: The law of the jungle (3점)
- 실습3: Oil (4점)

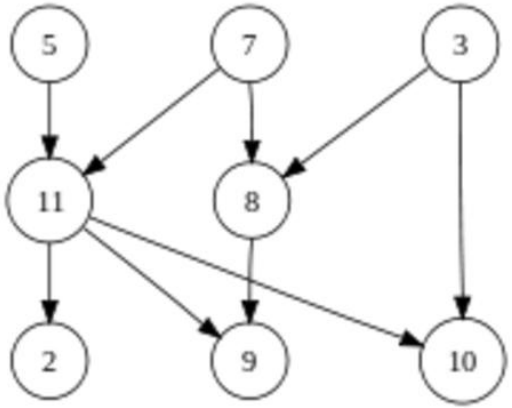
Weighted Directed Graph



실습1. Prerequisite (3)

위상 정렬 문제란?

- https://en.wikipedia.org/wiki/Topological_sorting
- 일을 순서대로 하자!



The graph shown to the left has many valid topological sorts, including:

- 5, 7, 3, 11, 8, 2, 9, 10 (visual left-to-right, top-to-bottom)
- 3, 5, 7, 8, 11, 2, 9, 10 (smallest-numbered available vertex first)
- 5, 7, 3, 8, 11, 10, 9, 2 (fewest edges first)
- 7, 5, 11, 3, 10, 8, 9, 2 (largest-numbered available vertex first)
- 5, 7, 11, 2, 3, 8, 9, 10 (attempting top-to-bottom, left-to-right)
- 3, 7, 8, 5, 11, 10, 2, 9 (arbitrary)

왜 정렬을 해야할까?

- 복잡하게 얽힌 문제를 차례차례 수행하기 위해서!
- ex) 식당에서 청소, 테이블 셋팅, 재료 준비, 음식 조리, 음식 서빙, 식사, 계산 등의 일을 어떤 순서대로 해야할까?
- ex2) 리그오브레전드에서 삼위일체를 가는 방법은?
- ex3) 스타크래프트2에서 초패스트 우주모함 빌드를 어떻게 짜야할까?

Kahn’s Algorithm

- 위상 정렬을 하는 알고리즘 중 가장 유명 (최초, 1962) 한 알고리즘
- 들어오는 엣지가 없는 노드를 방문
  - 방문한 노드에서 나가는 엣지 제거(반복)

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    add n to tail of L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

선수과목

- 윤서는 겨울 방학을 맞이하여 2개월간 진행되는 개발 해커톤에 참가하고자 한다. 이 해커톤은 N가지의 과목이 있으며, 하루에 한 과목씩 수강할 수 있다. 또한, 각 과목은 선수과목이 있을 수 있다. 예를 들어, 수학1 수학2 라는 정보는 수학2를 듣기 위하여 수학1을 수강한 상태여야 한다는 의미다. 수학1 수학2, 수학2 수학1과 같이 순환 관계는 없다. 윤서가 과목을 '사전순'으로 모두 수강하고자 할 때 선후수과목을 고려하여 윤서가 수강할 과목 순서를 출력하시오.

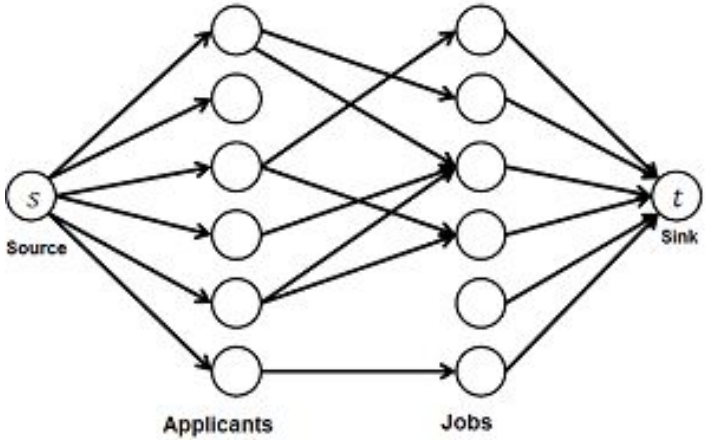
[Input]
11 8
컴퓨터프로그래밍1 컴퓨터프로그래밍2 자료구조
컴퓨터프로그래밍3 객체지향설계 알고리즘 알고리즘응용
데이터베이스 데이터통신 컴퓨터네트워크 컴퓨터네트워크설계
컴퓨터프로그래밍1 컴퓨터프로그래밍2
컴퓨터프로그래밍2 자료구조
컴퓨터프로그래밍2 컴퓨터프로그래밍3
컴퓨터프로그래밍2 객체지향설계
알고리즘 알고리즘응용
자료구조 데이터베이스
데이터통신 컴퓨터네트워크
컴퓨터네트워크 컴퓨터네트워크설계

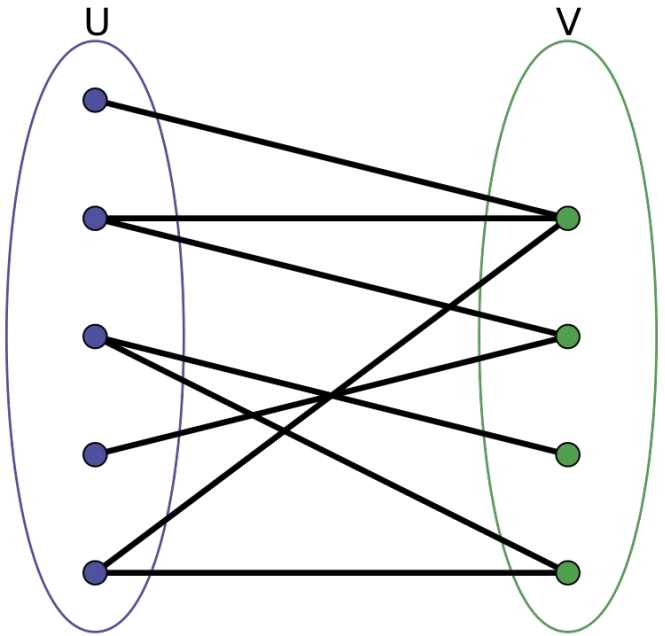
[Output]
데이터통신 알고리즘 알고리즘응용 컴퓨터네트워크
컴퓨터네트워크설계 컴퓨터프로그래밍1 컴퓨터프로그래밍2
객체지향설계 자료구조 데이터베이스 컴퓨터프로그래밍3

실습2. The law of the jungle (3)

Bipartite Graph

- U / V 그룹간 엣지가 없는 그래프
- 출처: [Wikipedia]





Ford–Fulkerson algorithm

- 출발지부터 도착지까지 최대 유량을 계산 (최단거리랑 다름!)

Algorithm Ford–Fulkerson

Inputs Given a Network $G = (V, E)$ with flow capacity c, a source node s, and a sink node t

Output Compute a flow f from s to t of maximum value

1. $f(u, v) \leftarrow 0$ for all edges (u, v)
2. While there is a path p from s to t in G_f, such that $c_f(u, v) > 0$ for all edges $(u, v) \in p$:
    1. Find $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
    2. For each edge $(u, v) \in p$
        1. $f(u, v) \leftarrow f(u, v) + c_f(p)$ (*Send flow along the path*)
        2. $f(v, u) \leftarrow f(v, u) - c_f(p)$ (*The flow might be "returned" later*)

- "$\leftarrow$" denotes assignment. For instance, "*largest* $\leftarrow$ *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

약육강식

- 미생물 연구원인 목구는 미생물을 배양하던 중 배양 환경에 충분한 영양분이 공급되지 않을 경우 서로 잡아먹는 것을 알아내었다.
  미생물은 크기와 빠르기가 특징으로 크기와 빠르기 모두 같거나 우세할 때 다른 미생물을 잡아먹을 수 있다.
  예를 들어 A 미생물의 크기와 빠르기가 4 3이고, B 미생물이 2 2면 A는 B를 잡아먹을 수 있다. C 미생물이 4 3이면 A가 C를, C가 A를 잡아먹 을 수 있지만, 서로를 동시에 잡아먹지는 못 한다.
  미생물은 소화 한계로 최대 2마리까지만 잡아먹을 수 있다.
  살아남을 수 있는 미생물의 최솟값을 구하시오.

```
[Input]
5
4 3
2 2
4 3
1 1
5 1

[Output]
2
```

실습3. Oil (4)

기름이 간당간당

- 여행을 좋아하는 수미는 렌트한 자동차를 타고 이곳저곳을 누비고 있다.
  수미는 여행지에 살고있다는 정미를 만나기 위하여 정미가 있는 곳까지 이동하려 하는데, 기름이 충분하지 않은 것 같다.
  수미가 가지고 있는 기름의 양과 특정 마을로 이동할 때에 소모되는 기름의 양이 지도 형태로 주어졌을 때, 도착지까지 간 후 수미의 차에 남을 기름의 최대 양을 출력하시오.
  만약, 도착할 수 없다면 Not enough oil! 을 출력하시오.

  - 소모되는 기름의 양을 0~9이며,
    출발지는 S, 도착지는 E로 표현된다.
    (출발지, 도착지는 0, 0 / n-1, m-1 이외의 장소일 수 있다.)

  - 입력: 행 열 남은기름
    출력: 최대남은기름양 또는 Not enough oil!

  - 휴리스틱: sqrt((X1 - X2)**2 + (Y1 - Y2)**2)

```
[Input]
3 3 10
S54
391
32E

[Output]
2

[Input]
7 7 10
S051153
4121653
0761685
1178323
9407641
5832483
748483E

[Output]
Not enough oil!
```

A* Algorithm

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path
```

```
// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how short a path from start to finish can be if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(1) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

참고) 이론자료

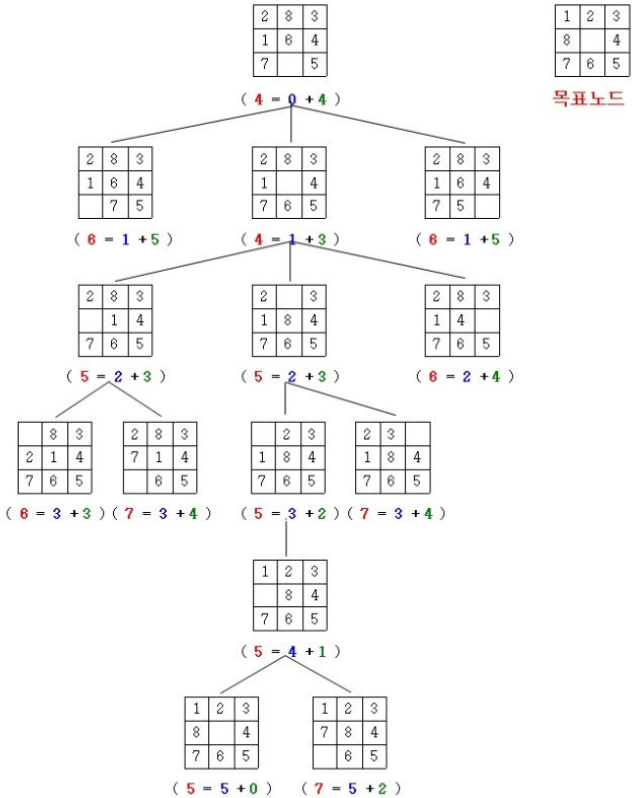
A* : 퍼즐 맞추기 예

$f(n)$ = g(n) + h(n)

$g(n)$: 현재까지의 값, 즉 지금까지 움직인 횟수

$h(n)$: 앞으로 예상되는 값, 위에서는 제자리에 있지 않은 퍼즐의 수

- 알고리즘
  - 비용이 최소화되는 경우 선택



참고

Standard Library

- Python 3: https://docs.python.org/3/library/
- JAVA 11: https://cr.openjdk.java.net/~iris/se/11/latestSpec/api/
- C: https://en.cppreference.com/w/c
- CPP: https://en.cppreference.com/w/cpp

ID/ext	external ID	name
adb	ada	Ada
awk	awk	AWK
bash	bash	Bash shell
c	c	C
csharp	csharp	C#
cpp	cpp	C++
f95	f95	Fortran
hs	haskell	Haskell
java	java	Java
js	javascript	JavaScript
kt	kotlin	Kotlin
lua	lua	Lua
pas	pascal	Pascal
pl	pl	Perl
sh	sh	POSIX shell
plg	prolog	Prolog
py2	python2	Python 2
py3	python3	Python 3
r	r	R
rb	ruby	Ruby
scala	scala	Scala
swift	swift	Swift

주요 링크

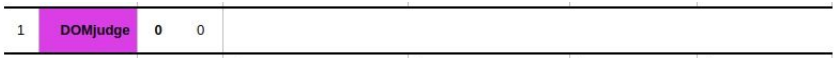
- 코딩테스트: http://coding.cnu.ac.kr:8080/domjudge/public
- FAQ : https://docs.google.com/document/d/1ntR6GS1SI7dRbYlw-pu8uT8U65Wc-RtMj10IEpiapfU/
- 질문: 메일!
- 문의 사항: munhyunsu@cs-cnu.org
  - ID / PASSWORD 변경
  - 실습 시간외 질문: [AL]질문제목

잊지 않아야 할 것) 소스코드 및 보고서

- 사이버 캠퍼스에 목요일까지 제출
  - 추가 시간 필요한 학생들도 목요일까지 제출. 추가 시간 문제 해결은 메일로도 제출!
- 보고서(.pdf 파일), 소스코드(.java, .py 등) zip 파일 압축
  - AL_학번_이름_06.zip (메일 추가 제출) or AL19_06.zip (사이버캠퍼스)
- 시간/공간 복잡도 해석(STL 고려), 자신의 생각, 질문, 느낀점, 공유하고 싶은 문제
  - 문제 해결을 위해 어떤 접근법을 사용하였는지, 무엇을 배웠고 느꼈는가?

보고서 템플릿: .pdf 로 제출!

- 알고리즘-x주차-주제
  학번 이름
- 코드 테스팅 결과 (점수표)



- 각 문제별 내용
  a. 문제 / 목표
  b. 해결 방법 (주요 소스코드 첨부. 스크린샷 or 코드 CV)
  c. 결과 (입력, 출력 결과)
- 느낀점: 과제를 하며 느낀 점 / 공유하고싶은 문제 / 난이도 / 부탁 / 조교에게... 등