Advanced Core in Algorithm Design #2
算法設計要論 第2回

Yasushi Kawase
河瀬 康志

October 11th, 2022

last update: 1:31pm, October 4, 2022

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

Relationship between Input Size and Running Time

	$\sqrt{n}$	n	$n \log n$	n^2	2^n	$n!$
1 sec.	$1.0 \cdot 10^{16}$	$1.0 \cdot 10^{8}$	$6.4 \cdot 10^{6}$	10000	26	11
1 min.	$3.6 \cdot 10^{19}$	$6.0 \cdot 10^{9}$	$3.1 \cdot 10^{8}$	77500	32	12
1 hour	$1.3 \cdot 10^{23}$	$3.6 \cdot 10^{11}$	$1.5 \cdot 10^{10}$	600000	38	15
1 day	$7.5 \cdot 10^{25}$	$8.6 \cdot 10^{12}$	$3.3 \cdot 10^{11}$	$2.9 \cdot 10^{6}$	42	16
1 month	$6.7 \cdot 10^{28}$	$2.6 \cdot 10^{14}$	$8.7 \cdot 10^{12}$	$1.6 \cdot 10^{7}$	47	17
1 year	$9.7 \cdot 10^{30}$	$3.1 \cdot 10^{15}$	$9.7 \cdot 10^{13}$	$5.6 \cdot 10^{7}$	51	18
1 century	$9.7 \cdot 10^{34}$	$3.1 \cdot 10^{17}$	$8.5 \cdot 10^{15}$	$5.6 \cdot 10^{8}$	58	20

- Maximum sizes that can be calculated in limited times
- assumption: 10^8 calculations per second

Asymptotic Order

notations to represent order for sufficiently large n

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^{100} < 2^n < 3^n < n! < n^n$$

Asymptotic upper bound: $f(n) = \mathrm{O}(g(n))$

$\exists k > 0,\ \exists n_0,\ \forall n > n_0,\ f(n) \leq k \cdot g(n)$

Asymptotic lower bound: $f(n) = \Omega(g(n))$

$\exists k > 0,\ \exists n_0,\ \forall n > n_0,\ f(n) \geq k \cdot g(n)$

Asymptotically tight bound: $f(n) = \Theta(g(n))$

$f(n) = \mathrm{O}(g(n))$ and $f(n) = \Omega(g(n))$

Examples

- $1000n = \Theta(n)$
- $18n^2 + 5n + 1 = \Theta(n^2)$
- $10(n+5)^8 = \Theta(n^8)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = \Theta(\log n)$ (base does not matter)
- $n = O(2^n)$ (big-O is just an upper bound)
- $5 \cdot 2^{n+3} = \Theta(2^n)$
- $\log(n!) = \Theta(n \log n)$ $(\because (n/2)^{n/2} \leq n! \leq n^n)$
- $n^{100} = \mathrm{O}(1.1^n)$ (see next slide)
- $(\log n)^{100} = \mathrm{O}(n^{0.001})$ (see next next slide)

Exponential

Proposition

For every $r > 1$ and $d > 0$, we have $n^d = \mathrm{O}(r^n)$

Proof For $n \geq d$, we have

$$
\begin{aligned}
r^n &= (1 + (r-1))^n \\
&= \sum_{k=0}^{n} \binom{n}{k} (r-1)^k \\
&\geq \binom{n}{d} (r-1)^d = \frac{n}{d} \cdot \frac{n-1}{d-1} \cdot \cdots \cdot \frac{n-d+1}{1} \cdot (r-1)^d \\
&\geq \underbrace{\frac{(r-1)^d}{d^d}}_{\text{constant}} \cdot n^d
\end{aligned}
$$

Logarithm

Proposition

For every $b > 1$, $d > 0$ and $\epsilon > 0$, we have $(\log_b n)^d = \mathrm{O}(n^\epsilon)$

Proof By setting $m := \log_b n$, we have

$$n^\epsilon = (b^m)^\epsilon = (b^\epsilon)^m$$

As $m^d = \mathrm{O}((b^\epsilon)^m)$,

$$(\log_b n)^d = \mathrm{O}(n^\epsilon)$$

Transitivity

Proposition

If $f(n) = \mathrm{O}(g(n))$ and $g(n) = \mathrm{O}(h(n))$, then $f(n) = \mathrm{O}(h(n))$

$$\begin{cases} \exists k, \exists n_0, \forall n > n_0, f(n) \leq k \cdot g(n) \\ \exists k', \exists n_0', \forall n > n_0', g(n) \leq k' \cdot h(n) \end{cases} \longrightarrow \quad \forall n > \max\{n_0,\ n_0'\},\ f(n) \leq k \cdot k' \cdot h(n)$$

Proposition

If $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$, then $f(n) = \Omega(h(n))$

$$\begin{cases} \exists k, \exists n_0, \forall n > n_0, f(n) \geq k \cdot g(n) \\ \exists k', \exists n_0', \forall n > n_0', g(n) \geq k' \cdot h(n) \end{cases} \longrightarrow \quad \forall n > \max\{n_0,\ n_0'\},\ f(n) \geq k \cdot k' \cdot h(n)$$

Proposition

If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $f(n) = \Theta(h(n))$

Quiz

Find the smallest one.

1. $\Theta(e^n)$
2. $\Theta(n!)$
3. $\Theta(n^{\log_e n})$
4. $\Theta((\log_e n)^n)$

Outline

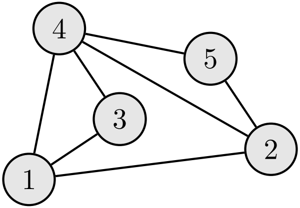
Undirected Graph $G = (V, E)$

- V: set of vertices. usually $n := |V|$
- E: set of edges. usually $m := |E|$

two-element subses of V

Example:

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,4\}, \{2,5\}, \{3,4\}, \{4,5\}\}$



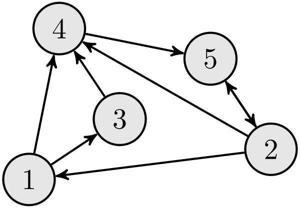
Directed Graph $G = (V, E)$

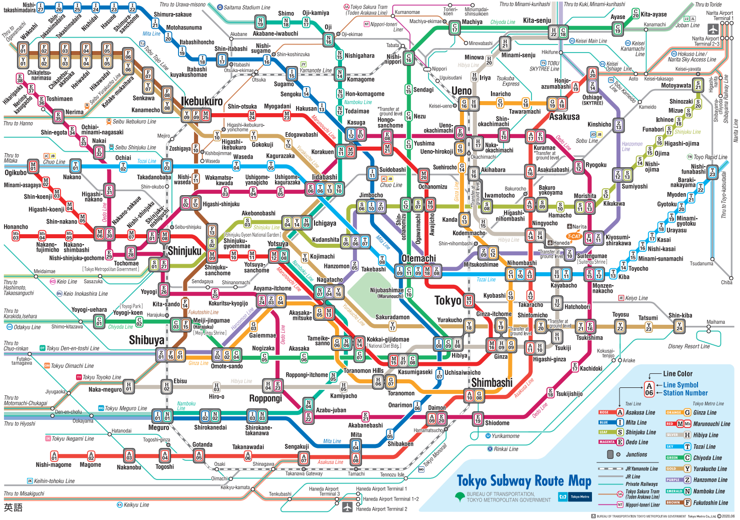
- V: set of vertices. usually $n := |V|$
- E: set of edges. usually $m := |E|$

  pairs of vertices

Example:

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{(1,3), (1,4), (2,1), (2,4), (2,5), (3,4), (4,5), (5,2)\}$



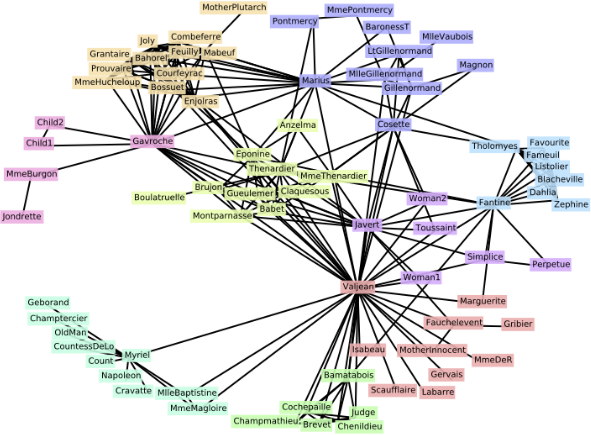


© 東京メトロ

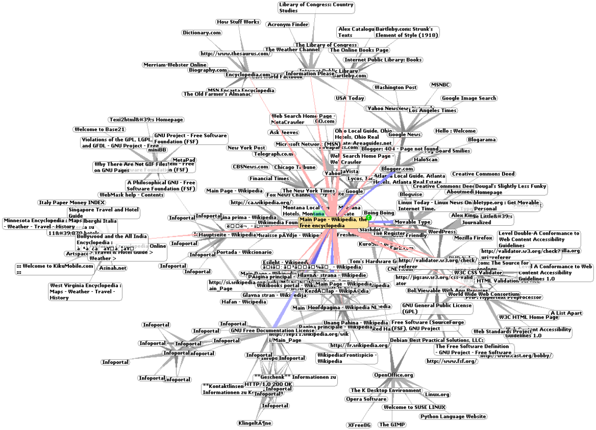
https://www.tokyometro.jp/lang_en/station/202006_number_en.png

Example (2/3): Les Misérables

the interactions between characters ($n = 77,\ m = 254$)
created by Donald Knuth



directed links between pages of the World Wide Web



https://en.wikipedia.org/wiki/Graph_drawing#/media/File:WorldWideWebAroundWikipedia.png

Representing Graphs

Adjacency matrix

- ⋆ matrix $A \in \{0,1\}^{n \times n}$ where $a_{ij} = 1$ iff $(v_i, v_j) \in E$
- space complexity: $\mathrm{O}(n^2)$
- check $(v_i, v_j) \in E$: $\mathrm{O}(1)$
- suitable for dense graph

Adjacency list

- ⋆ n liked list, each describes the set of neighbors
- space complexity: $\mathrm{O}(n + m)$
- check $(v_i, v_j) \in E$: $\mathrm{O}(d)$
- suitable for sparse graph

d: maximum degree

Representing Graphs — Example

Adjacency matrix

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Adjacency list

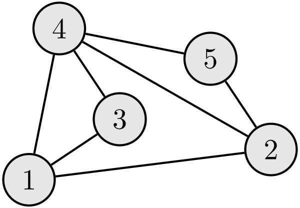
1 [2,3,4]

2 [1,4,5]

3 [1,4]

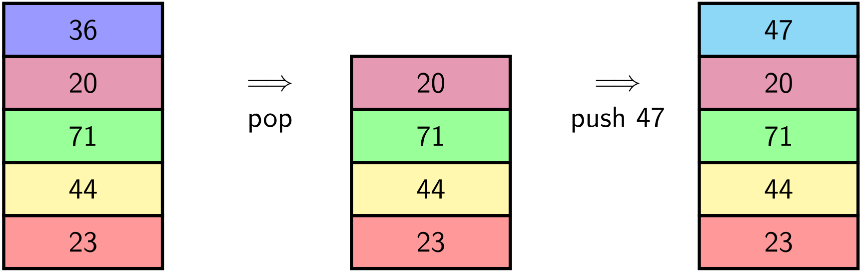
4 [1,2,3,5]

5 [2,4]



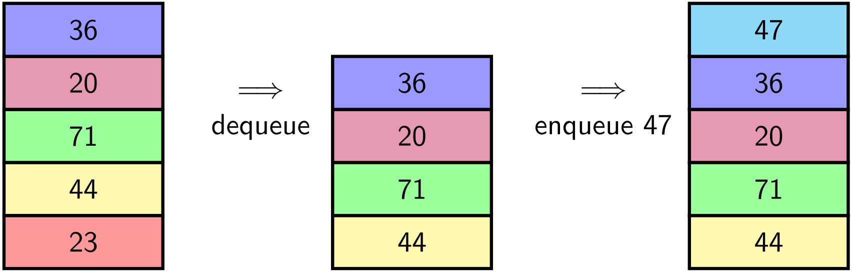
Stack

- a data structure that follows the LIFO principle
- two basic operations
  - push: add an element to the collection ($\mathrm{O}(1)$ time)
  - pop: remove the most recently added element ($\mathrm{O}(1)$ time)



Queue

- a data structure that follows the FIFO principle
- two basic operations
  - enqueue: add an element to the collection ($O(1)$ time)
  - dequeue: remove the earliest added element ($O(1)$ time)



Depth First Search

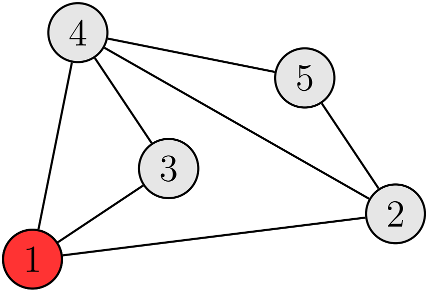
- an algorithm for graph traversal (determining s–t connectivity)
- push the next candidate to be searched on a stack
- $\mathrm{O}(m+n)$ time (if the graph is represented by adjacency list)

DFS algorithm startingfrom $s \in V$

$\texttt{explored}[u] \leftarrow \texttt{False}$ $(\forall u \in V)$ and let S be a stack only with s;
while *S is not empty* **do**
    pop S and let u be the popped vertex;
    if $\texttt{explored}[u]$ **then continue**;
    $\texttt{explored}[u] \leftarrow \texttt{True}$;
    foreach *$\{u, v\} \in E$ incident to u* **do**
        if $\texttt{explored}[v]$ **then** push v to S;

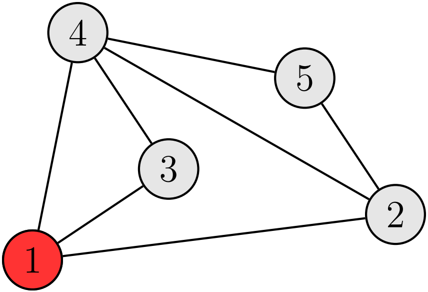
Example of DFS

- stack=[]
- $u = 1$
-



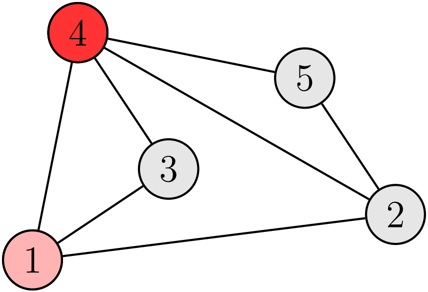
Example of DFS

- stack=$[2, 3, 4]$
- $u = 1$
- push $2, 3, 4$



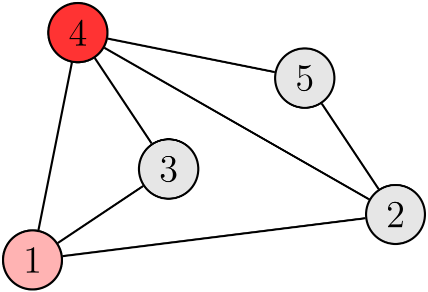
Example of DFS

- stack=$[2, 3]$
- $u = 4$
- pop



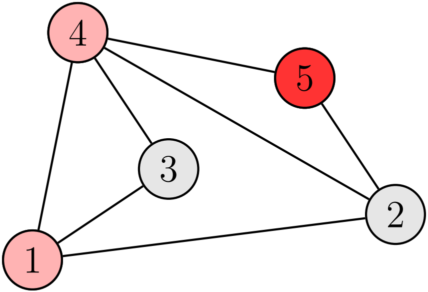
Example of DFS

- stack=$[2, 3, 2, 3, 5]$
- $u = 4$
- push 2,3,5



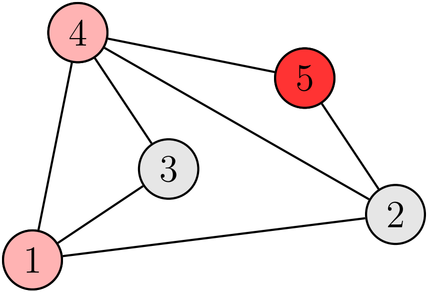
Example of DFS

- stack=$[2, 3, 2, 3]$
- $u = 5$
- pop



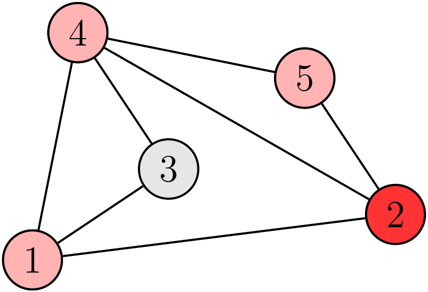
Example of DFS

- stack=$[2, 3, 2, 3, 2]$
- $u = 5$
- push 2



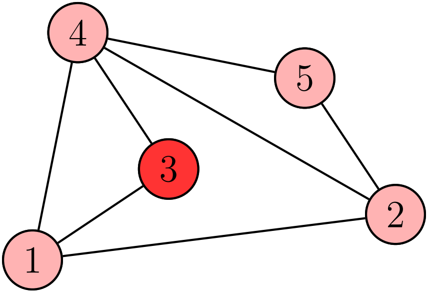
Example of DFS

- stack=$[2, 3, 2, 3]$
- $u = 2$
- pop



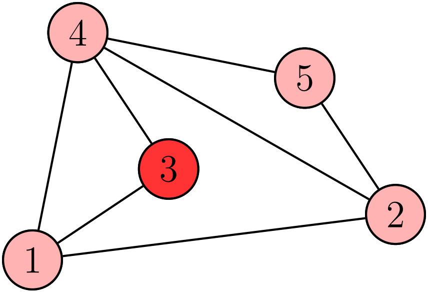
Example of DFS

- stack=$[2, 3, 2]$
- $u = 3$
- pop



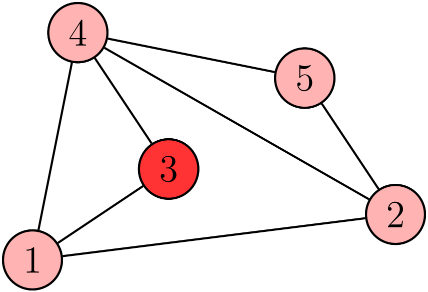
Example of DFS

- stack=$[2, 3]$
- $u = 3$
- pop



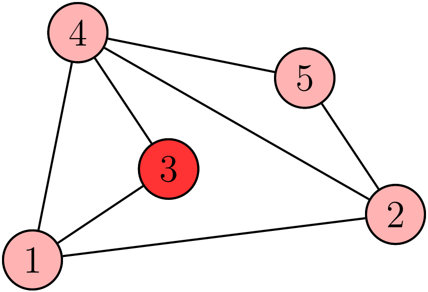
Example of DFS

- stack=[2]
- $u = 3$
- pop



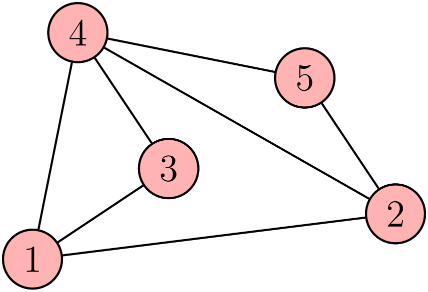
Example of DFS

- stack=[]
- $u = 3$
- pop



Example of DFS

- stack=[]
- $u = 3$
-



Breadth First Search

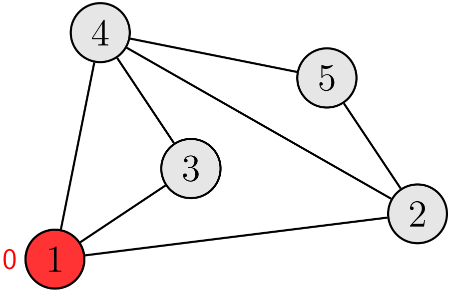
- an algorithm for graph traversal (determining s–t connectivity and dist.)
- enqueue the next candidate to be searched on a queue
- $\mathrm{O}(m+n)$ time (if the graph is represented by adjacency list)

BFS algorithm starting from $s \in V$

$\mathrm{dist}[u] \leftarrow \infty$ $(\forall u \in V)$ and let Q be a queue only with $(s, 0)$;
while *Q is not empty* **do**
    dequeue Q and let (u, d) be the dequeued vertex–distance pair;
    if $\mathrm{dist}[u] < \infty$ **then continue**;
    $\mathrm{dist}[u] \leftarrow d$;
    foreach $\{u, v\} \in E$ *incident to u* **do**
        if `explored`$[v]$ **then** enqueue $(v, d+1)$ to Q;

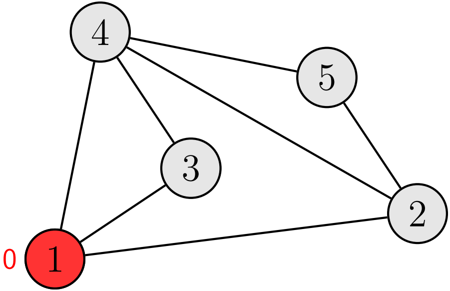
Example of BFS

- queue=[]
- $(u, d) = (1, 0)$
-



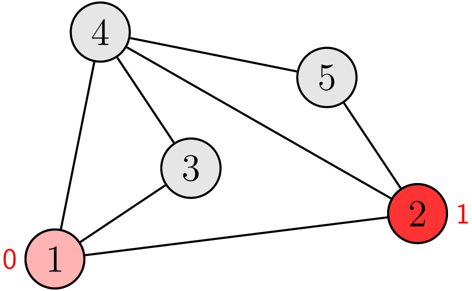
Example of BFS

- queue=$[(2,1),(3,1),(4,1)]$
- $(u,d)=(1,0)$
- enqueue $2,3,4$



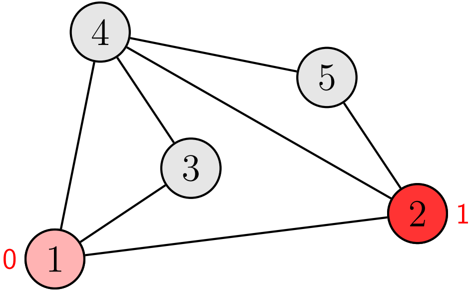
Example of BFS

- queue=$[(3,1),(4,1)]$
- $(u,d)=(2,1)$
- dequeue



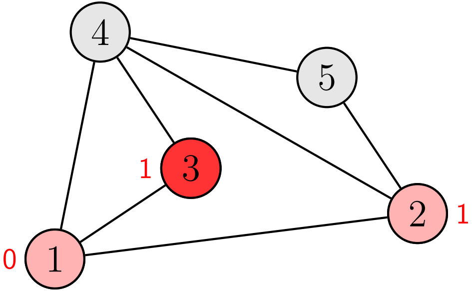
Example of BFS

- queue=$[(3,1),(4,1),(4,2),(5,2)]$
- $(u,d) = (2,1)$
- enqueue $4, 5$



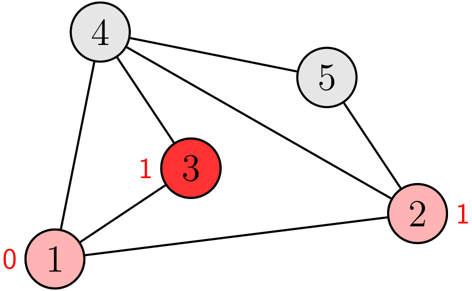
Example of BFS

- queue=$[(4,1),(4,2),(5,2)]$
- $(u,d)=(3,1)$
- dequeue



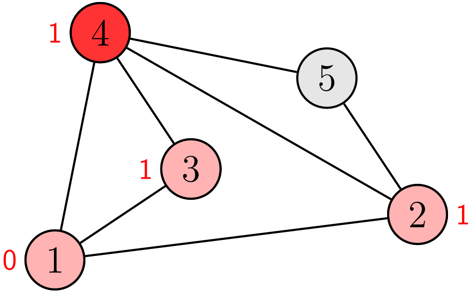
Example of BFS

- queue=$[(4,1),(4,2),(5,2),(4,2)]$
- $(u,d)=(3,1)$
- enqueue 4



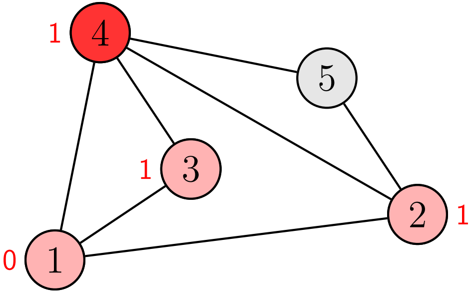
Example of BFS

- queue=$[(4,2),(5,2),(4,2)]$
- $(u,d)=(4,1)$
- dequeue



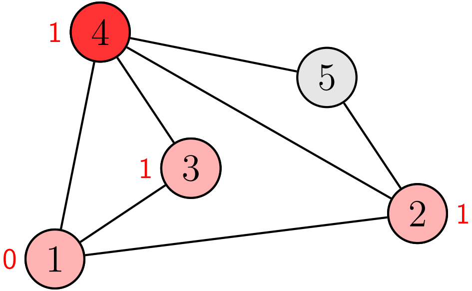
Example of BFS

- queue=$[(4,2),(5,2),(4,2),(5,2)]$
- $(u,d) = (4,1)$
- enqueue 5

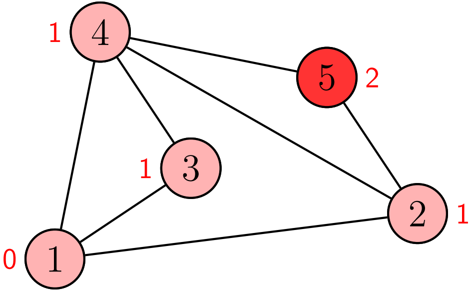


Example of BFS

- queue=$[(5, 2), (4, 2), (5, 2)]$
- $(u, d) = (4, 1)$
- dequeue

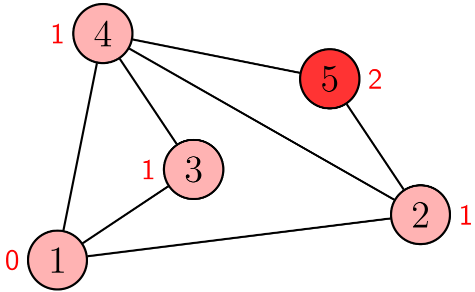


- queue=$[(4,2),(5,2)]$
- $(u,d)=(5,2)$
- dequeue



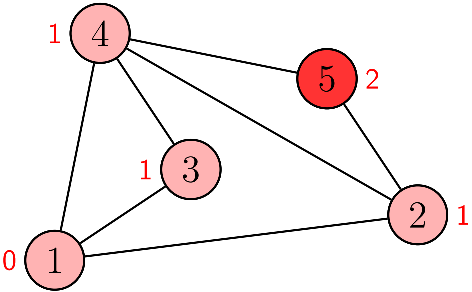
Example of BFS

- queue=$[(5, 2)]$
- $(u, d) = (5, 2)$
- dequeue



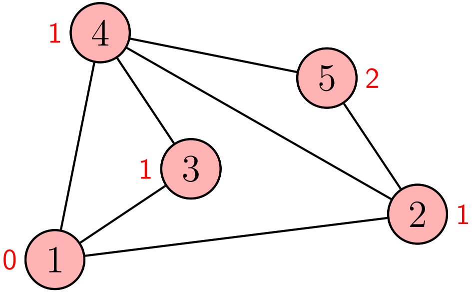
Example of BFS

- queue=[]
- $(u, d) = (5, 2)$
- dequeue



Example of BFS

- queue=[]
- $(u, d) = (5, 2)$
-



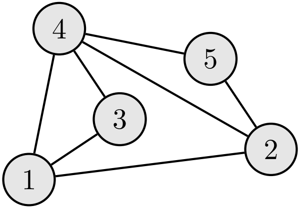
DFS vs. BFS

Depth First Search

- implemented using Stack
- may not give the shortest paths
- require less memory

Breadth First Search

- implemented using Queue
- give the shortest paths
- require more memory



Outline

1. Asymptotic Order of Growth
2. Graph Traversal
3. Interval Scheduling
4. Interval Partitioning

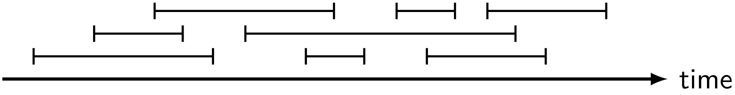
Interval Scheduling

Problem

- Input: jobs $J = \{1, 2, \ldots, n\}$, job j starts at $s(j)$ and finishes at $f(j)$
- Goal: find maximum subset of mutually compatible jobs

two jobs that don't overlap

Example

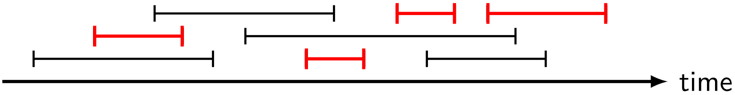


Problem

- Input: jobs $J = \{1, 2, \ldots, n\}$, job j starts at $s(j)$ and finishes at $f(j)$
- Goal: find maximum subset of mutually compatible jobs

two jobs that don't overlap

Example

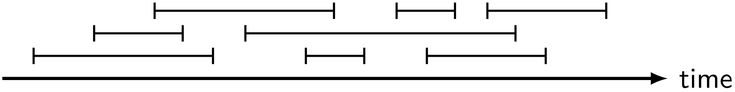


Algorithm

Greedy Algorithm

$R \leftarrow J$, $A \leftarrow \emptyset$;
while $R \neq \emptyset$ **do**
  Let $i \in \arg\min\{f(i) \mid i \in R\}$;
  $A \leftarrow A \cup \{i\}$;
  $R \leftarrow \{j \in R \mid s(j) > f(i)\}$;
Return A;

Example



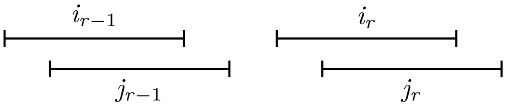
Optimality

Theorem

The greedy algorithm outputs an optimal solution

Proof

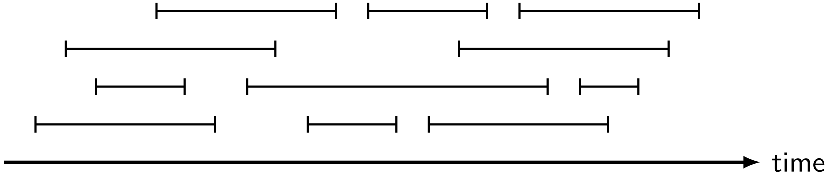
- $A = \{i_1, i_2, \ldots, i_k\}$: algorithm's output $(f(i_1) \leq \cdots \leq f(i_k))$
- $A^* = \{j_1, j_2, \ldots, j_m\}$: optimal solution $(f(j_1) \leq \cdots \leq f(j_m))$
- Claim: $f(i_r) \leq f(j_r)$ for all $r = 1, 2, \ldots, k$
  - Base case: $f(i_1) \leq f(j_1)$ by the definition
  - Induction step: $f(i_{r-1}) \leq f(j_{r-1}) \Rightarrow f(i_r) \leq f(j_r)$



- If $m > k$, the algorithm can choose j_{k+1} after i_k → Contradiction

Quiz

What is the optimal value of the following interval scheduling?



Outline

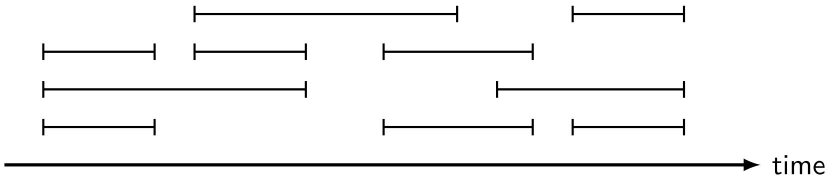
Interval Partitioning (Interval Coloring)

Problem

- Input: jobs $J = \{1, 2, \ldots, n\}$, job j starts at $s(j)$ and finishes at $f(j)$
- Goal: minimum number of people who can do all jobs

each person can do at most one job simultaneously

Example



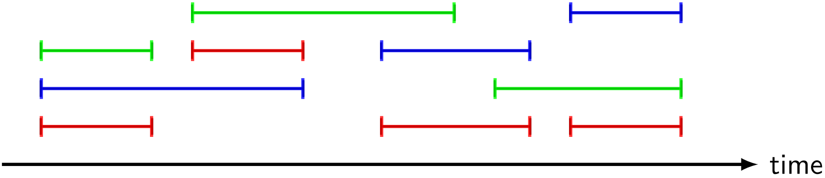
Interval Partitioning (Interval Coloring)

Problem

- Input: jobs $J = \{1, 2, \dots, n\}$, job j starts at $s(j)$ and finishes at $f(j)$
- Goal: minimum number of people who can do all jobs

each person can do at most one job simultaneously

Example $(\text{optimal} = 3)$



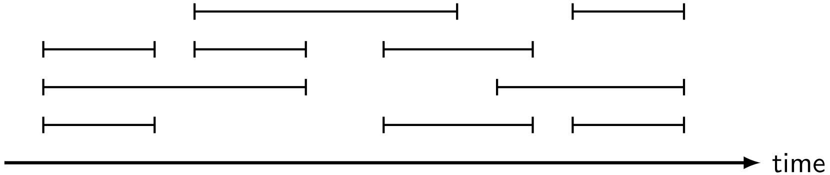
Basic observation

Observation

maximum number of pairwise overlapping intervals

the optimal value $\geq$ depth

Example (depth $= 3$)



Basic observation

Observation

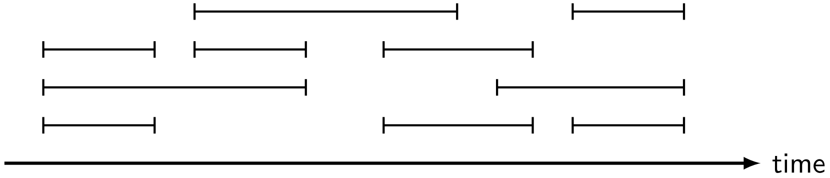
maximum number of pairwise overlapping intervals

the optimal value $\geq$ depth

Theorem

the optimal value $=$ depth

Example (depth $= 3$)

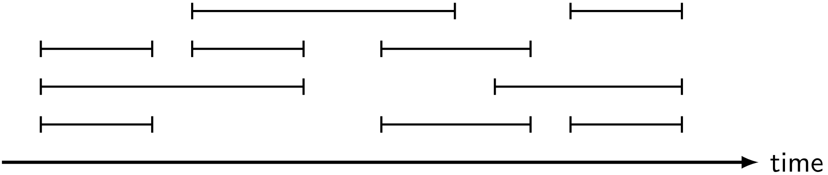


Algorithm

Greedy Algorithm

sort and relabel the jobs by their start times $(s(1) \leq \cdots \leq s(n))$;
let d be the depth and prepare d people;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign j to any person who is free within time $(s(j), f(j))$;

Example



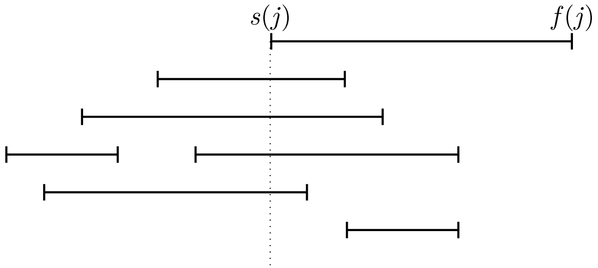
Correctness

Theorem

The greedy algorithm correctly assigns the jobs to d people

Proof: when the algorithm assigns job j, at least one person is free

→ the greedy algorithm is correct



Quiz

What is the optimal value of the following interval partitioning?