Ch.4 Greedy Algorithms

Ex. (failure case)

coins 1, 10, 25, 100

use min # of coins to reach target

largest coin first (greedy strategy)

$T = 34$: $25 + 1 + \ldots + 1$ (10 coins)

$10 + \ldots + 10 + 1 + \ldots + 1$ (7 coins)

4.1 Scheduling

Class 1, 2, ..., n, each class has starting time s(n) and finishing time f(n)

Goal: schedule max # of classes that are compatible (no overlap)

Idea: schedule classes 1 by 1 according to some ordering

	s	f
a	0	4
b	1	3
c	2	4
d	3	5

a, b, c, d (sort by starting time)

solution: a → not always optimal

d, a, c, b (sort by finishing time)

Greedy alg. $O(n \log n)$

sort by finishing time $T_{v_1}, T_{v_2}, \ldots, T_{v_n}$, $f(T_{v_1}) \le \ldots \le f(T_{v_n})$

$A = \phi$

for $i = 1, 2, \ldots n$:

if T_{v_i} is compatible w/ A:

$A \leftarrow A + \{T_{v_i}\}$

Thm. Greedy algorithm gives the optimal solution.

Pf. Let $\{i_1, i_2, \ldots i_k\}$ be the solution by greedy algorithm.

Claim: For any compatible classes $\{j_1, \ldots, j_k\}$,

$f(i_r) \le f(j_r) \quad \forall \quad r = 1, \ldots, k$

BC $r = 1$: $f(i_1) \le f(j_1)$

IH Assume $f(i_r) \le f(j_r) \quad \forall \quad 1 \le r \le n$

IS WTS: $f(i_{n+1}) \le f(j_{n+1})$

$f(i_n) \le f(j_n) \le s(j_{n+1})$

$\Rightarrow$ class j_{n+1} is compatible with $\{i_1, i_2, \ldots i_m\}$

$\Rightarrow$ GD will pick $i_{m+1} \Rightarrow f(i_{n+1}) \le f(j_{n+1})$

If $\exists$ a compatible set $j_1, \ldots j_k$ s.t.

$f(i_k) \le f(j_k) \le s(j_{k+1})$, GD will be able to pick j_{k+1}.

Interval partition

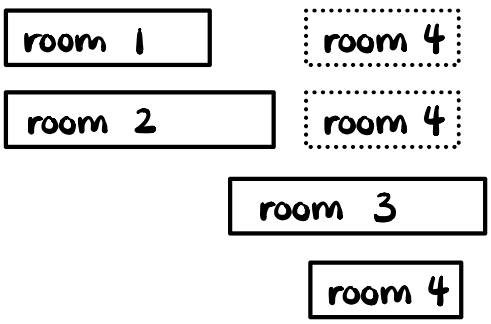
class $1, \ldots, n$

Goal: min # of rooms to schedule all the classes

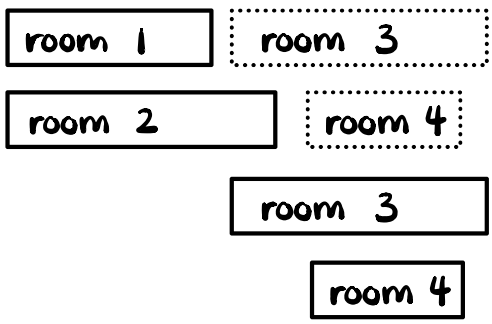
Idea: consider classes 1 by 1

create a room when needed

Sorting by finishing time:



Sorting by starting time:



Greedy alg.

sort classes by starting time: $s(\pi_1) \leq \ldots \leq s(\pi_m)$

$d = 0$ # of rooms

for $j = 1, 2, \ldots n$:

if π_j is compatible with some room $k \leq d$

$A_k = A_k + \{\pi_j\}$

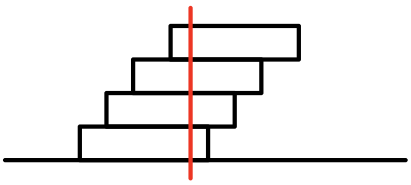
else:

create a new room $A_{d+1} = \{\pi_j\}$

$d = d + 1$

Pf. of correctness

"depth" : max # of classes at any time



depth : lower bound of # rooms

Thm. Greedy outputs d = depth.

Pf. Assume room d is created by GD alg. when scheduling π_j.

$\Rightarrow$ π_j is not compatible with room $1, \ldots, d-1$

Since we sort by starting time,

$s(\pi_j) \geq s(\pi_i)$ $\forall$ $i = 1, \ldots, d-1$.

π_j overlap with x_i $\Rightarrow$ $f(x_i) > s(\pi_i)$

$\Rightarrow$ $\forall$ i, $s(\pi_i) \leq s(\pi_j) < f(x_i)$

$\Rightarrow$ $s(\pi_j)$ has depth d.

4.4 Shortest Path

- Single source shortest path
  - $G = (V, E)$ DG
  - ℓ_e $(e \in E)$: cost (length of each edge)

Goal: Find shortest path from a source node s to all other nodes.

- X : explored nodes (shortest path known)
- $d[u]$: min length $s \rightsquigarrow u$ path via X

Dijkstra's algorithm:

$X = \{s\}$ $\quad d[u] = \begin{cases} \ell(s,u) & \text{if } (s,u) \in E \\ \infty & \text{otherwise} \end{cases}$

for $i = 1, \ldots, n$:

$u \leftarrow$ min $d[u]$ among unexplored nodes

for each v s.t. $(u, v) \in E$

if $d[u] + \ell(u, v) < d[v]$

$d[v] = d[u] + \ell(u, v)$

$pre[v] = u$

shortest path $s \rightsquigarrow v$

$s \rightsquigarrow v = pre[v] - v$

print shortest path $s \rightsquigarrow v$ (backward):

```
u = v
while u ≠ s:
    print u
    u = pre[v]
```

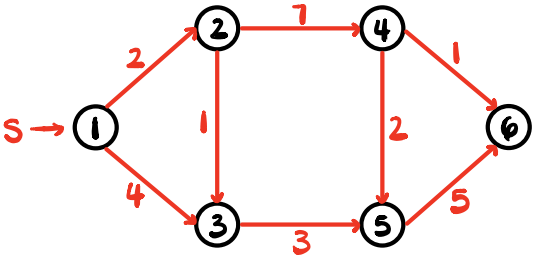
① unexplored node v with min $d[u]$

$\Rightarrow$ $d[u]$ shortest $s \rightsquigarrow u$ path

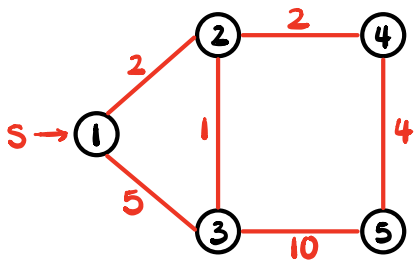
② min path $s \rightsquigarrow u$ via $X + u$:

$$\min(d[v],\ d[u] + \ell(u, v))$$

$s \rightsquigarrow u - v$



	$X = \{1\}$	$X = \{1,2\}$	$X = \{1,2,3\}$	$X = \{1,2,3,5\}$	$X = \{1,2,3,4,7\}$
$d[1]$	0	0	0	0	0
$d[2]$	2	2	2	2	2
$d[3]$	4	3	3	3	3
$d[4]$	∞	9	9	8	8
$d[5]$	∞	∞	6	6	6
$d[6]$	∞	∞	∞	11	9



	$X = \{1\}$	$X = \{1,2\}$	$X = \{1,2,3\}$	$X = \{1,2,3,4\}$
$d[1]$	0	0	0	0
$d[2]$	2	2	2	2
$d[3]$	5	3	3	3
$d[4]$	3	3	3	3
$d[5]$	∞	∞	13	7

Thm. unexplored u with min $d[u]$ value

$\Rightarrow$ $d[u]$ is the $s \rightsquigarrow u$ shortest path

Pf. for any other path via ≥ 1 node z in $V-X$

$p: \underbrace{s \rightsquigarrow y}_{\in X} - \underbrace{z \rightsquigarrow u}_{\notin X}$

$\ell(p) = \ell(s \rightsquigarrow y) + \ell(y - z) + \ell(z \rightsquigarrow u)$

$\geq d[z]$

$d[u] \leq d[z] \Rightarrow d[u] \leq \ell(p)$

$\Rightarrow$ $d[u]$ is shortest path

Implementation push $d[u]$ $\forall$ $u \in V$ into U

for $i = 1, \ldots, n-1$

$u \leftarrow \min d[u]$ in $V-X$ $U.pop()$

for v s.t. $(u, v) \in E$

$d[v] = \min(d[v], d[v] + \ell(u, v))$ update value in U

store d using array:

$O(n)$ to find min

$\Rightarrow O(n^2)$ total

store d using heap:

$O(n \log n + n \log n + n \log n)$

$\Rightarrow O(n \log n)$ total

another priority queue: fibonacci heap

- $O(1)$ push
- $O(\log n)$ pop
- $O(1)$ value update

$\Rightarrow$ $O(n \log n + n + m) = O(n \log n + m)$ total

4.5 Minimal Spanning Tree (MST)

· connected DG : $G=(V, E)$ with edge length

Goal: find a set of edges $T \subseteq E$

- $G'=(V, T)$ is connected
- minimize total cost $\sum_{e \in T} \ell(e)$

Sol. T^* will be a tree. Find

Pf. (by γ)

If T^* has cycle C, assume $(u, v) \in C$.

$T^*-(u, v)$ is still a connected subgraph.

$\forall$ s, t, if $s \rightsquigarrow u-v \rightsquigarrow t$ in T^*

$s \rightsquigarrow u \overset{p}{\rightsquigarrow} v \rightsquigarrow t$ in $T^*-(u, v)$

$T^*-(u, v)$ has smaller cost than T^*. γ

Prim's algorithm

maintain set X (current connected nodes)

$a[u]$ cost for adding u into X

$X = \{s\}$

$T = \emptyset$

$$a[u] = \begin{cases} \ell(s,u) & \text{if } (s,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, n-1$:

  $u \leftarrow$ node in $V - X$ w/ smallest $a[u]$

  add u to X

  add $(pre[u], u)$ to T

  for v s.t. $(u, v) \in E$

    if $\ell(u,v) < a[v]$:

      $a[v] = \ell(u,v)$

      $pre[v] = u$

Time complexity: same as Dijkstra's algorithm

- heap for a: $O(m \log n)$
- fibonacci heap for a: $O(n \log n + m)$

T: $(pre[u], u)$ $\forall$ $u \in V$

Pf. of correctness

Define "cut" node sets A, B:

$cut(A, B)$: all $(u,v) \in E$ s.t. $u \in A$, $v \in B$.

Prim's Alg.: X, $V-X$ add min-cost edge in $cut(A, B)$

Cut property: if edge e is the min-cost edge in $cut(X, V-X)$ to the node set X, then e must be in the MST.

Pf. If MST T^* and $e \notin T^*$:

assume $e = (u, v)$ and u, v in T^* are connected by $u \rightsquigarrow \underbrace{p}_{\in X} - \underbrace{q}_{\notin X} \rightsquigarrow v$

Define $T' = T^* - (p, q) + (u, v)$

T^* is still a spanning tree because $p \rightsquigarrow u - v \rightsquigarrow q$.

$\ell(T^*) > \ell(T')$. $\lightning$

Kruskal's algorithm

- sort edges from small to large
- consider edges 1-by-1.

  adding e doesn't make a cycle $\Rightarrow$ add e

  otherwise ignore e

Kruskal's algorithm

$T = \phi$

sort edges s.t. $e_1 \le e_2 \le \dots \le e_m$

for $i = 1, \dots, m$:

  $(u, v) \leftarrow e_i$

  if u, v are in different connected components of T:

    $T = T + \{e_i\}$

Pf. of correctness

If we add $e = (u, v)$ to T, u, v are in different connected components.

Assume X is the connected component of u.

e is the first edge in $cut(X, V-X)$ being considered in alg.

$\Rightarrow$ e is the min-cost edge in $cut(X, V-X)$

$\Rightarrow$ e should be in MST (cut property)

Implementation

- how to check u, v are in the same component?

Union-find structure: store n elements and their sets

- initial: each element forms individual set
- union(u, v): merge sets of u, v
- find(u): return set of u
  - find(u) = find(v) ⇒ u, v in same set

Array implementation:

- find O(1)
- union O(n)

Array implementation (improved):
always rewrite smaller set

"**Amortized complexity analysis**": time complexity for K unions from initial state

Claim: K unions take O(KlogK) time

Pf. ① K unions → touch at most 2K elements

② count how many times each element is rewritten

rewrite to u ⇒ size of set of u increases by ≥ 2 times

assume p rewrites to u

$\Rightarrow$ size of set of u is $\geq 2^p$

$2^p \leq 2K$: $p \leq \log(2K)$

$\Rightarrow$ total rewrites = $O(K \log K)$