3. Greedy & Basic data structures

Angel Napa

September 9, 2022

Resumen

Greedy

Basic Data Structures

Greedy

Frog Jumping

- The frog begins at position 0 in the river. Its goal is to get to position n
- There are lilypads in each position, from position 0 to n.
- The frog can jump at most r units at a time.
- **WANT:** Find a path the frog should take to minimize number of jumps, assuming a solution exists.

Algorithm in Frog Jumping

- Let V an empty set of places visited.
- x=0 is the initial position.
- while x<n
    - find the furthest position reachable fom x.
    - go to that position and save it in V
- return V

Greedy

- An algorithm is said to be greedy if it makes the optimal local choice in each step.
- In some cases, greedy algorithms construct the globally best object.

Greedy Advantajes

Greedy algorithms have some advantajes over other algorithmic algorithms

- **Simplicity:** Greedy algorithms are often easier to describe and code up than other algorithms
- **Efficiency:** Greedy algorithms can often be implemented more efficiently than other algorithms

Coin Changing

- Given sufficiently 1,2,5 S/. coins
- Device a method to pay the amount M using the least number of coins

Coin Changing

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT n coin denominations so that $0 < c_1 < c_2 < \ldots < c_n$.

$S \leftarrow \varnothing$. $\longleftarrow$ multiset of coins selected

WHILE $(x > 0)$

  $k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$.

  IF (no such k)

    RETURN *"no solution."*

  ELSE

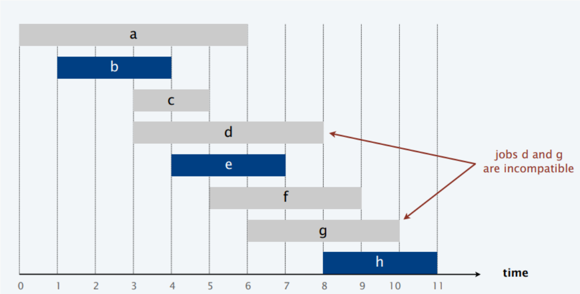
    $x \leftarrow x - c_k$.

    $S \leftarrow S \cup \{ k \}$.

RETURN S.

Interval Scheduling

- A series of jobs
- Job J_i starts at time s_i and ends in f_i
- Two jobs are mutually compatibles if their times intersect in at most one point
- **Goal:** find maximun subset of mutually compatible jobs.



Some algorithms that come into mind:

1. **Earliest start time:** Consider jobs in ascending order of s_i(start time)
2. **Earliest finish time:** Consider jobs in ascending order of f_i(finish time)
3. **Shortest term:** Consider jobs in ascending order of $f_i - s_i$(length of time)
4. None of the above

Interval Scheduling

Fig. 4.9 An instance of the scheduling problem and an optimal solution with two events

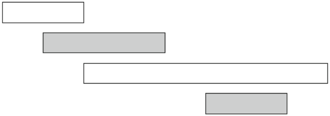


Fig. 4.10 If we select the short event, we can only select one event, but we could select both long events

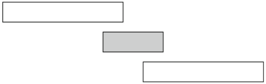
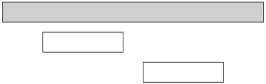


Fig. 4.11 If we select the first event, we cannot select any other events, but we could to select the other two events



Interval Scheduling

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

$S \leftarrow \varnothing$. $\longleftarrow$ set of jobs selected

FOR $j = 1$ TO n

  IF (job j is compatible with S)

    $S \leftarrow S \cup \{ j \}$.

RETURN S.

Theorem: the Earliest finish time algorithm is optimal

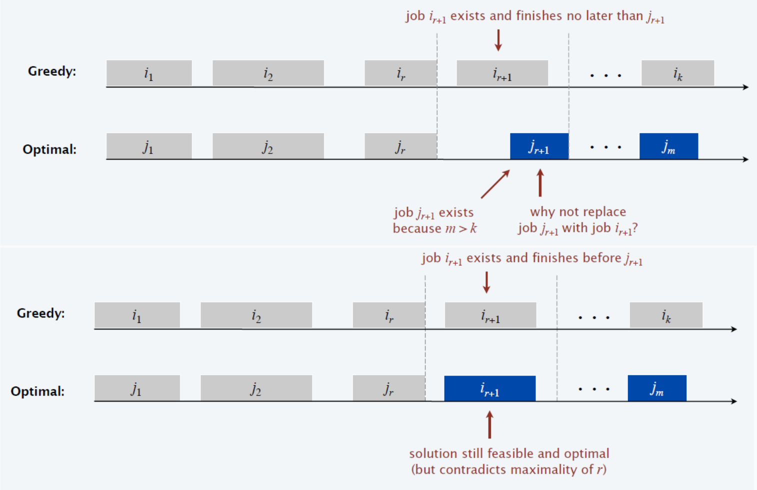
- Let be $i_1, i_2, \dots, i_k$ the set of jobs selected by greedy
- Let be $j_1, j_2, \dots, j_m$ the set of jobs of the optimal solution, with $m > k$.
- if these two sets are different, exist some $r \geq 1$ such that:

$$i_x = j_x,\ \forall x < r \text{ and } i_x \neq j_x$$

- consider the max value of r for all optimal solutions.

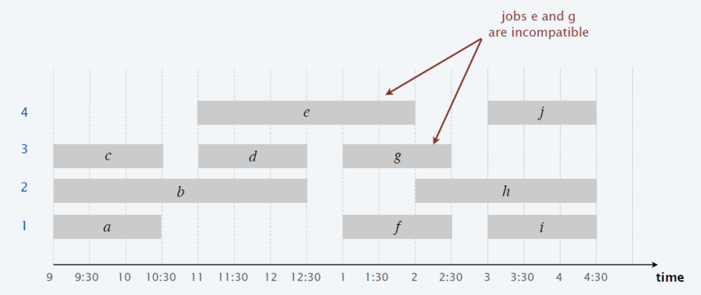
Interval Scheduling

Theorem: the Earliest finish time algorithm is optimal



Interval Partitioning

- A series of lectures at UTEC
- Lectrure L_i starts at time s_i and ends in f_i
- No two lectures should occur at the same clasroom
- **Goal:** Find minimun number of classrooms UTEC should open during the day.



Interval Partitioning

Some algorithms that come into mind:

1. **Earliest start time:** Consider lectures in ascending order of s_i(start time)
2. **Earliest finish time:** Consider lectures in ascending order of f_i(finish time)
3. **Shortest term:** Consider lectures in ascending order of $f_i - s_i$(length of time)
4. None of the above

Interval Partitioning

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start times and renumber so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$. ⟵ number of allocated classrooms

FOR $j = 1$ TO n

  IF (lecture j is compatible with some classroom)

    Schedule lecture j in any such classroom k.

  ELSE

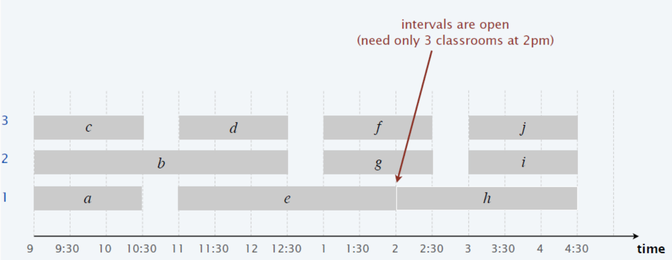
    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$.

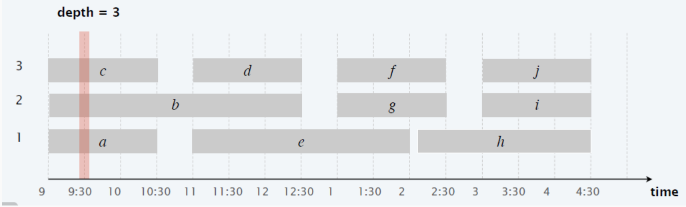
RETURN schedule.

Interval Partitioning



Interval Partitioning

Definition: The **depth** d of a set of intervals is the maximun number of intervals that contain a single point



- Each schedule will have at least d clasrooms
- The earliest start time produce exactly d clasrooms

Interval Partitioning

Theorem: the Earliest start time algorithm is optimal

- Assume we have build c classrooms in total.
- Lets locate in the first time we open the cth room
- Let L_j be the lecture that will locate first in this room
- Acording to the algorithm, this means L_j couldnt be located in any of the other $c - 1$ rooms
- The last lecture of these rooms ended after s_j.
- by sorting, each of these lectures started before s_j
- This means there is a point $s_j + \varepsilon$ that is in at least c classes.

some Problems

- Codeforces 1728A
- Codeforces 1728A
- Codeforces 1615B
- Codeforces 1650B
- Kattis downtime

Basic Data Structures

Data structures you've seen before

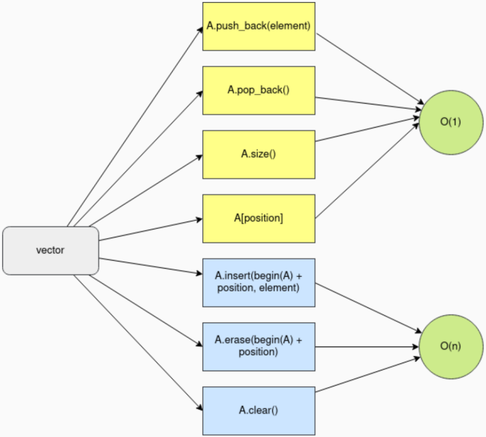
- Static arrays
- Dynamic arrays
- Stacks
- Queues
- Priority queues
- Sets
- Maps

Data structures you've seen before

- Static arrays - `int arr[10]`
- Dynamic arrays - `vector<int>`
- Stacks - `stack<int>`
- Queues - `queue<int>`
- Priority queues - `priority_queue<int>`
- Sets - `set<int>`
- Maps - `map<int, int>`

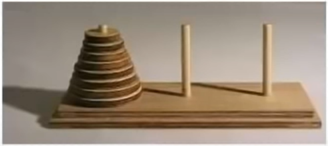
vector

- vector<T> v



stack

- stack<T> mazo
- LIFO
- mazo.push(x)
- mazo.pop()
- mazo.size()
- mazo.top()



queue

- queue<T> q , FIFO
- q.push(x)
- q.pop()
- q.size()
- q.front()
- q.back()
- q.empty()

dequeue

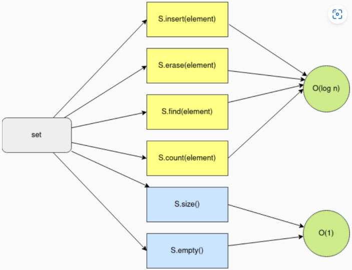
- dequeue<T> dq
- same as vector, but with efficient insertion and deletion of front
- dq.push_front(x)
- dq.pop_front()

map

- map<key,T> m
- associate keys from one type to another T.
- map<string,int> m associate strings to int
- m["UTEC"]=2022;

set

- set<T> s
- an object that does not allow repeated elements



set

```
set<int> s;
s.insert(4);
s.insert(7);
s.insert(1);

// find returns an iterator to the element if it exists
auto it = s.find(4);
// ++ moves the iterator to the next element in order
++it;
cout << *it << endl;

// if nonexistant, find returns end()
if (s.find(7) == s.end()) {
    cout << "7 is not in the set" << endl;
}

// erase removes the specific element
s.erase(7);

if (s.find(7) == s.end()) {
    cout << "7 is not in the set" << endl;
}

cout << "The smallest element of s is " << *s.begin() << endl;
```

multiset

- multiset<T> ms
- same as set, but it allows repeated elements
- ms.erase(x) will erase all elements x of ms
- ms.erase(ms.lower_bound(x)) will only erase 1 element

priority_queue

priority_queue<T> pq

- same as queue, but the object in the top(instead of front) is the greatest (by default)
- if you want the smallest at the top, we need to initialite the priority_queue as follows:
- priority_queue< T,vector<T> , greater<T> > pq;
- pq has .top() instead of .front().

Some problems

- Kattis guesthedata
- Kattis akcija
- Kattis pivot
- Kattis securedoors
- Kattis babelfish

Gracias