

Course contents

- Greedy Algorithms (Ch. 17)
- NP-completeness (Ch. 36)
- Approximation Algorithms (Ch. 37)
- On-line Algorithms (Not in our text)

ALG/TITCS L02-1

Optimization Problems

Many problems have many solutions, although we want to find a “best” one—the one which minimize the “cost” or maximize the “value”. Such a problem is called an optimization problem, and such a solution is said to be optimal.

Example (the triangulation problem): Given a simple convex polygon, give a way to divide the polygon into a set of triangles, so as to minimize the total perimeter.

So every triangulation is a solution, although we want to find one with the minimum perimeter.

There may be more than one solutions that is optimal. So we speak of *an* optimal solution instead of *the* optimal solution.

ALG/TITCS L02-2

Greedy Algorithms

Some problems can be solved optimally using a greedy strategy. Not all problems can be solved this way, but many can.

If a problem can be solved using a greedy strategy, the running time of the greedy algorithm is typically much less than that of any other algorithms, e.g. using linear programming or dynamic programming.

It is therefore important to know the situations under which greedy algorithms would suffice. It is also important to know the techniques to show algorithms to be optimal using the greedy strategy.

ALG/TITCS L02-3

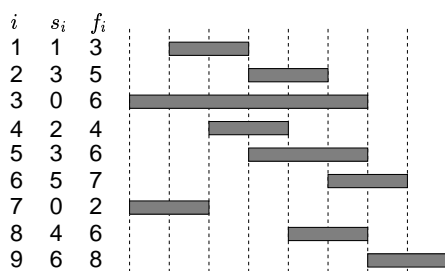
Example problem: activity-selection

Problem:

- We are given a set $S = \{1, 2, \dots, n\}$ of activities that needs to use a resource.
- Each activity i has a start time s_i and a finish time f_i .
- If selected, the activity takes place during the interval $[s_i, f_i)$.
- Two activities i and j are said to be compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.
- We want to find the largest set of compatible activities.

ALG/TITCS L02-4

Example



ALG/TITCS L02-5

Non-greedy solutions

- **Exhaustive:** For each possible combinations of $\{1, 2, \dots, n\}$, check whether the combination of activities are compatible. Output the largest combination. [Exponential time.]
- **Recursive:** For each activity i , try to select i . Break the input set of activities into three: one containing activities that finish at or before s_i , one containing activities that starts at or later than f_i , and the last containing activities that are not compatible with activity i . Recursively solve the two sub-problems for the first two sets. [Exponential time.]
- **Dynamic programming:** Use the idea of the recursive solution, but build up from the bottom. Solve all the possible subproblems. [$O(n^2)$ time.]

ALG/TITCS L02-6

Example greedy solutions

Unlike the other strategy, we are not so sure about the optimality.

- We list out the activities in the order from the shortest to the longest. Then we scan through the list, select a activity whenever it compatible with all the other selected activities.
- We list out the activities in the order of the number of activities that are not compatible. Then we scan through the list like the above.
- We list out the activities in the order of starting time, from the earliest to the latest. Then we scan through the list like the above.
- [GREEDY-ACTIVITY-SELECTOR] We list out the activities in the order of finishing time, from the earliest to the latest. The we scan through the list like the above.

ALG/TITCS L02-7

The optimal greedy solution

GREEDY-ACTIVITY-SELECTOR is optimal (and all the others are not), i.e. it produces the maximum size set of compatible activities.

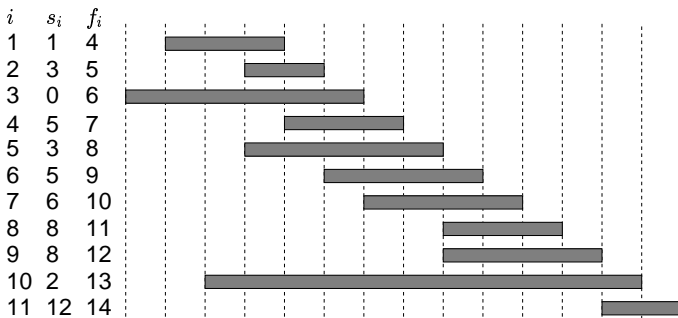
It is interesting to see what is so special about GREEDY-ACTIVITY-SELECTOR that makes it optimal.

It is also among the algorithms that requires the least time: suppose the input is already sorted against the finish time. For each activity during the scan, we only have to check one activity for compatibility.

The idea of the proof of optimality is simple. We basically shows that everytime we select an activity, there is actually an optimal solution that does pick the activity that we have chosen, and the problem actually reduces to the same problem with smaller input.

ALG/TITCS L02-8

Example schedule



ALG/TITCS L02-9

Proof of optimality

Claim: Suppose the input is in the order of finishing time. Then there is an optimal solution that contains activity 1.

Proof: Suppose A be an optimal solution. Let k be the activity with the least finish time in A .

If $k = 1$, we are done. Assume not.

No activity in A starts before activity k . Otherwise that activity has earlier start time than k and no earlier finish time than k , so it is not compatible with activity k .

So all activities in A other than k start later than the finish time of k .

Since activity 1 is an activity with the earliest finish time, all activities in A other than k are compatible with activity 1.

So $A \cup \{1\} - \{k\}$ is an optimal solution, claim proven.

ALG/TITCS L02-10

Proof of optimality (cont.)

Claim: Let B be the set of activities in $S - \{1\}$ that are compatible with activity 1, and let B' be a maximum subset of activities in B that are compatible. Then $\{1\} \cup B'$ is a maximum subset of activities in S that are compatible.

Proof: First, the set $\{1\} \cup B'$ is compatible: any two activities in B' must be compatible because it is a maximum subset of activities that are compatible. Any activity in B' is compatible with activity 1 because B' is a subset of B , which only contains activities that are compatible with activity 1.

Second, the set $\{1\} \cup B'$ must be optimal. Otherwise, suppose A is a maximum subset of compatible activities in S , and by the above claim we assume $1 \in A$. Then $A - \{1\}$ is compatible, and all activities in $A - \{1\}$ are compatible with 1. So $A - \{1\}$ is a compatible subset of the set B .

But $|A - \{1\}| = |A| - 1 > |B' \cup \{1\}| - 1 = |B'|$. So B' should not be a maximum compatible set of B , contradiction!

ALG/TITCS L02-11

Proof of optimality (cont.)

Claim: GREEDY-ACTIVITY-SELECTOR is optimal.

Proof: By our first claim, our first choice (of activity 1) must be correct. Then we skip all the activities that are not compatible with activity 1. This reduces to exactly the sub-problem in claim 2.

Algorithm

```

n ← length[s]
A ← {1}
j ← 1
for i ← 2 to n
    do if si ≥ fj
        then A ← A ∪ {i}
        j ← i
return A
    
```

ALG/TITCS L02-12

Two ingredients of Greedy Algorithms

The problem must exhibit two properties for a greedy algorithm to find an optimal solution.

- There must be a choice that is guaranteed to lead to an optimal solution. That is, there must be a “good” choice that is **never** “wrong”. We call this property the **greedy-choice property**.
- An optimal solution to the problem must consist of optimal solutions to subproblems, and we must be able to get one such subproblem after the greedy choice. That is, we must be able to reduce the problem to a smaller problem after the greedy choice. We call this property the **optimal substructure property**.

ALG/TITCS L02-13

General form of greedy algorithms

- Due to the greedy-choice property, the algorithm proceeds by actually making the greedy choice, without worrying that it leads to a situation that optimality is impossible to achieve.
- With the optimal substructure property, the algorithm finds the subproblems, and recursively (or iteratively) solves the subproblems.
- After the solution to the subproblems is found, it builds the solution to the original problem, usually by adding the greedy choice already made.

ALG/TITCS L02-14

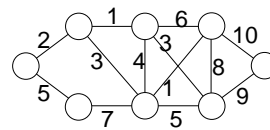
The minimum spanning tree problem

Problem (MST):

- We are given a connected graph $G = (V, E)$, where there are $|V| = n$ vertices and $|E| = m$ edges.
- Each edge e has a weight $w(e)$.
- We want to have a spanning tree, i.e. $n - 1$ edges that connect the graph.
- We want the spanning tree T that minimize the resulting total weight, $w(T) = \sum_{e \in T} w(e)$, among the tree edges.

ALG/TITCS L02-15

Example



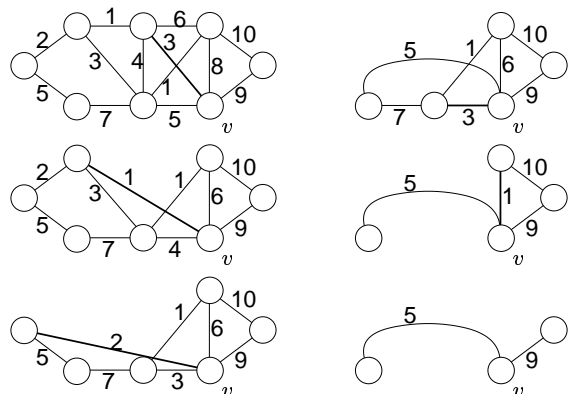
ALG/TITCS L02-16

Prim's Algorithm

1. Starts with an arbitrary node. Let this node be v . Let the set of tree edges to be an empty set.
2. Find the edge adjacent to v that has minimum weight among those edges. Suppose that edge is $e = (v, w)$.
3. Add e to the set of tree edges.
4. Modify the graph by merging u and v , as follows. Delete u . For each edge (u, w) where $w \neq v$, check whether there is an edge (v, w) with less weight. If so, delete the edge. Otherwise, add the edge (v, w) to the new graph, with the weight of the edge (u, w) . If originally there is an edge (v, w) with larger weight, delete it.
5. Recursively find the spanning tree for the new graph.

ALG/TITCS L02-17

Example



ALG/TITCS L02-18

Optimality of Prim's algorithm

Prim's algorithm is optimal because it exhibits both greedy-choice and optimal substructure properties.

Greedy-choice: there is a MST that contains the edge e , the first choice of our algorithm.

Proof: Suppose T is a MST. If T contains e , we are done. Otherwise, let v and u are connected by a path P starting with e' in T . Clearly, $e \neq e'$, since e is not in T .

$T \cup \{e\} - e'$ is a spanning tree: it contains $n - 1$ edges; and for any two vertices, either the path between them in T does not include e' so it is still connected, or the path does include $e' = (v, u')$.

For the latter case, note that we already have a path $P - e'$ followed by e which connects u' to v . So there is still a path between the two nodes, by replacing e' by this path.

So $T \cup \{e\} - e'$ is a spanning tree. The total weight of $T \cup \{e\} - e'$ is no larger than T (since $w(e) \leq w(e')$), so it is still a MST.

ALG/TITCS L02-19

Optimality of Prim's algorithm (cont)

Optimal substructure: Let a MST of the modified graph G' be T' . Then $T' \cup \{e\}$ is a MST of the graph G .

Proof: First, $T' \cup \{e\}$ is a spanning tree. It contains $n - 1$ edges. Any vertex of G other than u and v is either connected to u or to v by T' (since T' is a spanning tree of G'). Since u and v are connected by e in $T' \cup \{e\}$, all the vertices are connected.

Then $T' \cup \{e\}$ is a MST. Suppose not, and T_0 is a MST with less weight. By the greedy-choice property, suppose T_0 contains e .

So we consider $T_0 - \{e\}$ in G' . It must be a spanning tree: It contains no more than $n - 2$ edges. Also, it connects every nodes: our transformation can only enhance connectivity, not reduce it. And it must have less weight than T' : $w(T_0 - \{e\}) = w(T_0) - w(e) < w(T' \cup \{e\}) - w(e) = w(T')$. So T' is not a MST of G' , contradiction.

ALG/TITCS L02-20