Algorithm Design

Algorithm Design by Induction

- Evaluating Polynomials
- One-to-One Mapping
- Celebrity
- Maximum Sum Subarray

Induction (review)

Induction Principle

- Basis Step: $P(1)$ is ture
- Inductive Step: $\forall n \geq 1\colon\ P(n) \Rightarrow P(n+1)$

Assumption "P(n) is true" is called inductive hypothesis

$$P(1), P(1) \rightarrow P(2), P(2) \rightarrow P(3), \ldots . \Rightarrow\ \forall n \geq 1\ P(n)$$

Note: Basis Step can be any integer number m (even negative), just notice that you have to show that

$$\forall n \geq m\colon\ P(n) \Rightarrow P(n+1)$$

Evaluating Polynomials

Evaluating Polynomials

Problem: Given a sequence of real numbers $a_0, a_1, \ldots, a_n$ and a real number x, compute the value of the polynomial

$$P_n(x) = a_n x^n + \cdots + a_1 x + a_0$$

Algorithm 1

Induction hypothesis: We know how to calculate $P_{n-1}(x)$

Base case: $P_0(x) = a_0$

Computing $P_n(x)$: Compute $a_n x^n$ by n multiplications and add the result to $P_{n-1}(x)$

Analysis: It requires $n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2}$ multiplications and n additions

Algorithm 2

Improvement: The first improvement comes from the observation that there is a lot of redundant computation: the power of x are computed from the scratch. We can use the value of x^{n-1} to compute x^n.

Induction hypothesis: We know how to calculate $P_{n-1}(x)$ and x^{n-1}
Base case: $P_0(x) = a_0, x^0$

Computing $P_n(x)$: Compute $a_n x^n$ by 2 multiplications and add the result to $P_{n-1}(x)$

Analysis: It requires $2n$ multiplications and n additions

Algorithm 3 (Horner's algorithm)

Improvement: So far we reduced the problem by removing the last coefficient a_n. We can also remove the first coefficient a_0. The smaller problem becomes the evaluation of the following polynomial:

$$P'_{n-1}(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_2 x + a_1$$

Induction hypothesis: We know how to calculate $P'_{n-1}(x)$

Base case: $P_0{}'(x) = a_1$,

Computing $P_n(x)$: Compute $P_n(x) = xP'_{n-1}(x) + a_0$ by one multiplication and one addition.

Analysis: It requires n multiplications and n additions

```
P = a[n]
for i = 1 to n do
    P = x*P+a[n-i]
return P
```

What is the running time if we assume all inputs are k-bits integer numbers? Note that any multiplication or addition may increase the number of bits. For instance, you may need $2k$ bits to represent x^2. Work on this as an exercise.

One-to-One Mapping

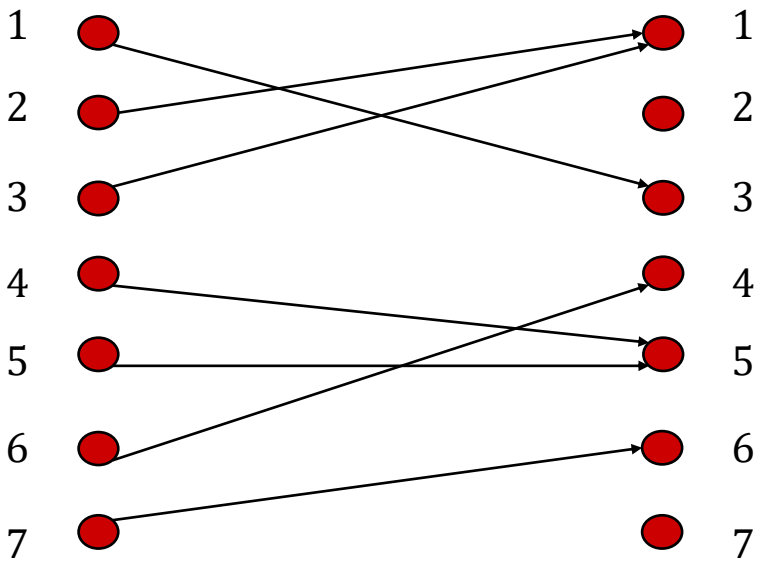
One-to-One Mapping

Problem: Assume $A = \{1,2, \dots, n\}$ and f is a given function from A to A. Find a subset S of A, with maximum cardinality such that

(1) f maps each element of S to another element of S (f maps S into itself)

(2) (2) every element of S has exactly one element of S which is mapped to it.

Example:



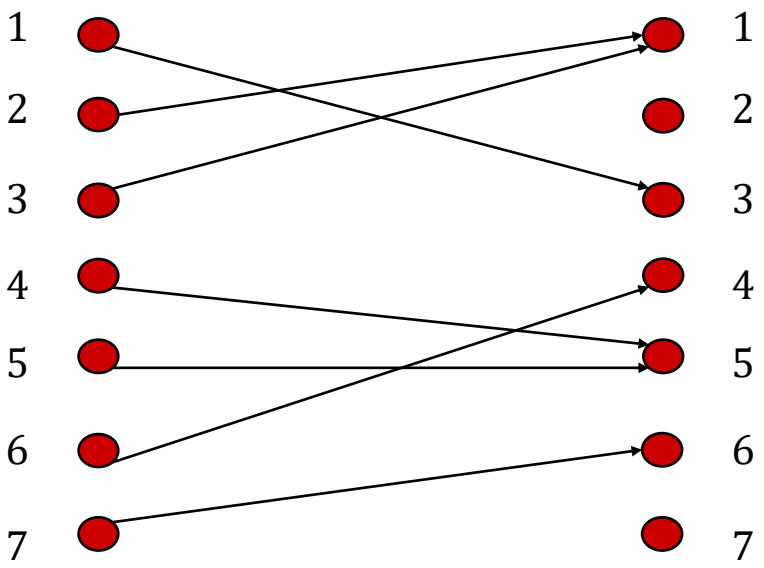
Answer: $S = \{1,3,5\}$

Some Observations

If f is one-to-one on A, then S=A.

If f(i)=f(j), then both i and j can not belong to S. Then we have to remove at least one of them. But which one ?

For example, S can not contain 2 and 3. If we remove 3, we should remove 1 as well. Therefore 2 must be removed as well (2 is mapped to 1). You can simply see this does not give us the maximum set.



Algorithm

We can reduce the size of the problem by finding either

(1) an element that belongs to S or

(2) an element that does not belong to S

We will do (2)

1. If A is one-to-one, then S=A
2. Otherwise, there is $i \in A$, no one from A is mapped to i. Remove i and now you have a function defined over n-1 elements.

Induction hypothesis: We know how to solve the problem for sets of n-1 elements.

```
S = A
for i = 1 to n do c[i]=0
for i = 1 to n do c[f[i]] = c[f[i]]+1
for i = 1 to n do
    if c[i]=0 then put i in Queue
While Queue is not empty do
    remove i from Queue
    S = S- {i}
    c[f(i)] = c[f[i]]-1
    if c[f(i)]=0 then put f(i) in Queue
return S
```

Celebrity

Celebrity

Celebrity: among n person, a celebrity is defined as someone who is known by everyone but does not know anyone.

Problem: To identify the celebrity (if exists) by only asking questions of forms "Excuse me, do you know this person over there". The goal is to minimize the number of questions.

Trivial answer: by n(n-1) questions we can construct a directed graph. If there is a vertex with in-degree n-1 and out-degree 0, then that vertex is the celebrity. Note that we add an edge from A to B if A knows B.

Algorithm

Observation: It may hard to identify someone as celebrity but it is probably easier to identify someone as a non-celebrity. Eliminating someone from consideration is enough to reduce the problem from n to n-1.

Suppose we ask A whether he knows B.

- If she does, she can not be a celebrity
- If she does not, B can not be a celebrity

Then we can eliminate one of them by only one question. Assume A is eliminated. Now

1. we find (by induction) a celebrity among the remaining n-1 persons.
2. If there is no celebrity, the algorithm terminates.
3. Otherwise, we check that A knows the celebrity and that celebrity does not knows A.

Complexity: n-1 queries to find the candidate and 2(n-1) queries to verify the candidate is a celebrity. In total, 3(n-1) queries.

Algorithm

```
i = 1
j = 2
next = 2
While next <= n do
   next = next+1
   if know[i,j] then i = next
   else j = next
if i = n+1 then candidate = j
else candidate = i
wrong = false
k = 1
Know[candidate, candidate] = false
while not wrong and k <= n do
    if know[candidate, k] then wrong = true
    if not know[k, candidate] then
       if candidate <> k then wrong = true
    k = k+1
if not wrong then print candidate is a celebrity
```

Maximum Sum Subarray

Maximum Sum Subarray

Problem: Given a one dimensional array A[1..n] of numbers. Find a contiguous subarray with largest sum within A.

Assume an empty subarray has sum 0.

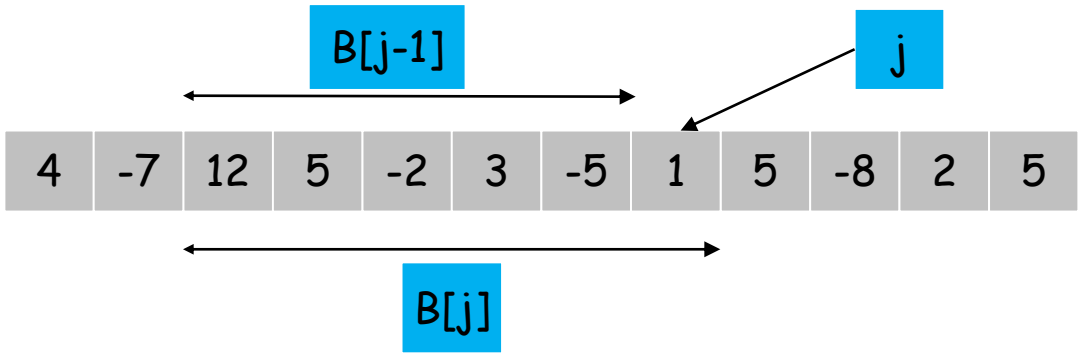
Example:

4	-7	12	5	-2	3	-5	1	5	-8	2	5

4	-7	12	5	-2	3	-5	1	5	-8	2	5

Algorithm

The general strategy: consider the subarray with max sum ending at index j. Let B[j] be the sum of all entries in this subarray. The output is max(B[j]) over all j=1,…,n. B[j] can be computed from B[j-1] by the recursive formula B[j]=max(0,A[j], A[j]+B[j-1])




```
sol = 0
B[0] = 0
for i = 1 to n do
    B[i] = max(0,A[i], A[i]+B[i-1])
    if B[i] > sol then
      sol = B[i]
return sol
```

Running time: $T(n) = O(n)$

References

References

- Sections 5.2, 2.4, 5.5, and 5.8 of the text book "introduction to algorithms: a creative approach" by Udi Manber, 1989.