Exercise sheet 4.

Advanced Algorithms

WiSe 2020/21

Instructor: László Kozma

Due 12:00, December 4th, 2020

Exercise 1 Dynamic programming *7 Points*

(a) (4 pts) Construct the dynamic programming table (by hand) for a non-trivial example input of your choice, for **one** of the following problems discussed in class. Alternatively, implement the dynamic programming algorithm and output the table.

(i) Edit Distance: choose two words of length 8-10. Compute the distance between them, as well as the sequence of operations.

(ii) Optimum BST: choose a probability distribution over a set of 8-10 keys. Compute the cost of the optimal tree, as well as the tree itself.

(b) (3 pts) A simple greedy heuristic for the optimum BST problem is to put the key x with the highest probability to the root, then recurse on the left (right) subtree formed of keys smaller (larger) than x. Find a small example that shows that this heuristic does not necessarily lead to the optimum.

(c) (Bonus 3 pts) Consider the algorithm discussed in class for finding the optimal BST. Let $r(i, j)$ denote the root of the optimal tree built of keys $i, \ldots, j$. (If there are multiple possibilities, assume that one is picked arbitrarily.)

It can be shown that $r(i, j-1) \leq r(i, j) \leq r(i+1, j)$. Intuitively, if you "extend" the interval of keys in some direction, then the optimal root may only "move" in that direction.

Using this observation, show that the running time can be improved from $O(n^3)$ to $O(n^2)$.

(*Hint*: Argue about the total time needed to fill each diagonal of the dynamic programming table.)

Exercise 2 Dynamic programming *6 Points*

Solve **any one** of the following problems as efficiently as you can. Argue that your solution is correct and analyze its running time.

(a) When the pandemic is over you decide to move to Ibiza with a suitcase that can carry a total weight of at most T. You have a number of objects with weights $w_1, w_2, \ldots, w_n$ and corresponding values $v_1, v_2, \ldots, v_n$. You would like to pack your suitcase (with objects of total weight *at most* T) such as to maximize the total value of objects packed. All values and weights are integers. The running time of your algorithm may depend on T and n.

(b) Given a sequence of integers, find the maximum length of an increasing subsequence (a subsequence is not necessarily contiguous). E.g. in the sequence $9, 1, 6, 2, 4, 8, 5, 7, 3$, a longest increasing subsequence is $1, 2, 4, 5, 7$, of length 5.

(c) A man walks a dog on a leash. The man steps through points $x_1, \ldots, x_n$ on a path, while the dog steps through points $y_1, \ldots, y_n$ on a separate path. In every timestep either (1) the man moves from its current place x_i to x_{i+1} and the dog stays in place, or (2) the dog moves from y_i to y_{i+1} and the man stays in place, or (1) both the man and the dog move. Both can move only by discrete steps, and neither can go back to an already visited point.

Given the distances between x_i to y_j for all $1 \leq i, j \leq n$, find the minimum length of the leash that allows the man and the dog to reach x_n and y_n, starting from x_1 and y_1. Assume (unrealistically) that the man and the dog cooperate in solving the problem.

Exercise 3 Finding a long path *12 Points*

Given an undirected graph G with n vertices and an integer k, we would like to find a simple path of k vertices in G, or report that no such path exists ("simple" means that no vertex is visited more than once). In this exercise we develop an algorithm with running time $2^{O(k)} \cdot n^{O(1)}$. Observe that if $k \in O(\log n)$, then this is polynomial in n. Variants of this algorithm are used in bioinformatics to find local structures in biological networks.

(a) (2 pts) Give a simple, possible inefficient algorithm for the problem (the running time could be e.g. $O(n^k)$). Briefly describing the idea is sufficient.

(b) (2 pts) Suppose that to each vertex of G we assign one of k *colors*, chosen uniformly at random. You may think of the colors as labels from $\{1, \ldots, k\}$. A *rainbow-path* is a path in which each vertex has a different color. What is the probability that a given k-vertex path becomes a rainbow-path in our coloring?
Give a lower bound for this probability, using the inequality $k! \geq (k/e)^k$.

(c) (4 pts) Solve by dynamic programming: Given a graph G with the vertices colored with k colors, find a k-vertex rainbow-path in G, or report that no such path exists. The running time should be of the form $2^{O(k)} \cdot n^{O(1)}$.
(*Hint*: you may build a table over subsets of colors and end-vertices of a path, answering the question "is there a rainbow-path using exactly these colors, ending at this vertex?". How large is this table? How can you fill it? How can you read out from the table whether G contains a k-vertex rainbow-path?)

(d) (2 pts) Give a randomized Monte-Carlo algorithm for solving the original problem.
(*Hint*: color the vertices randomly, as in (b), then run the algorithm from (c). What is the probability of success?)

(e) (2 pts) Repeat (d) sufficiently many times to increase the success-probability to at least $(1 - \frac{1}{n})$. What is the overall running time?

Exercise 4 Subset sum *4 Bonus Points*

Given a set of positive integers $S = \{x_1, \ldots, x_n\}$, we want to find whether there is a subset of S that adds up to a given value T. The naïve algorithm solves the problem in time $O(n2^n)$. This running time can be improved to $O(n2^{n/2})$. How? What is the space requirement?
(Hint: Split S into two parts of equal size, and for each sum on the left side, search for a possible match on the other side. How can we make this search efficient?)

Total: 25 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

Exercise 1 Dynamic programming *7 Points*

(a) (4 pts) Construct the dynamic programming table (by hand) for a non-trivial example input of your choice, for **one** of the following problems discussed in class. Alternatively, implement the dynamic programming algorithm and output the table.

(i) EDIT DISTANCE: choose two words of length 8-10. Compute the distance between them, as well as the sequence of operations.

(ii) OPTIMUM BST: choose a probability distribution over a set of 8-10 keys. Compute the cost of the optimal tree, as well as the tree itself.

		z	a	b	c	d	e	f	f
	0	1	2	3	4	5	6	7	8
b	1	1	2	2	3	4	5	6	7
c	2	2	2	3	2	3	4	5	6
d	3	3	3	4	3	2	3	4	5
e	4	4	4	4	4	3	2	3	4
v	5	5	5	5	5	4	3	3	4
g	6	6	6	6	6	5	4	4	4
h	7	7	7	7	7	6	5	5	5
i	8	8	8	8	8	7	6	6	6

= distance

Rules (a b / c x)

if c < a ∧ c < b ⇒ delete char

if a ≤ b ∧ a ≤ c ⇒ if x = a: nothing
else: change char

if b < a ∧ b < c ⇒ add char

Sequenz:

1. f → i
2. f → h
3. add g
4. add v
5. pass
6. pass
7. pass
8. pass
9. delet a
10. delet z

z a b c d e f f
x x + + ↓ ↓

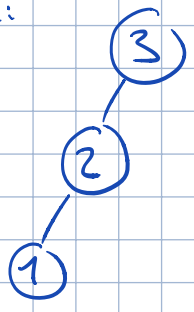
⇒ b c d e v g h i

4

1)

(b) (3 pts) A simple greedy heuristic for the optimum BST problem is to put the key x with the highest probability to the root, then recurse on the left (right) subtree formed of keys smaller (larger) than x. Find a small example that shows that this heuristic does not necessarily lead to the optimum.

For $P_1 < P_2 < P_3$ this would create the following Tree:



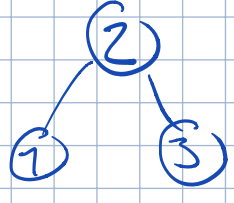
e.g. $P_3 = 0{,}35$

$P_2 = 0{,}33$

$P_1 = 0{,}32$

$$Cost_T = \sum_{i=1}^{n} cost_T(i) = P_3 + 2P_2 + 3P_1 = 0{,}35 + 2 \cdot 0{,}33 + 3 \cdot 0{,}32 = 1{,}97$$

Different Tree:



$0{,}33 + 2 \cdot 0{,}32 + 2 \cdot 0{,}35$

$= 1{,}67$?

$$Cost_T = P_2 + 2P_1 + 2P_3 = 0{,}33 + 2 \cdot 0{,}32 + 2 \cdot 0{,}35 = 1{,}67$$

⇒ Second Tree would be better in this case

2.5

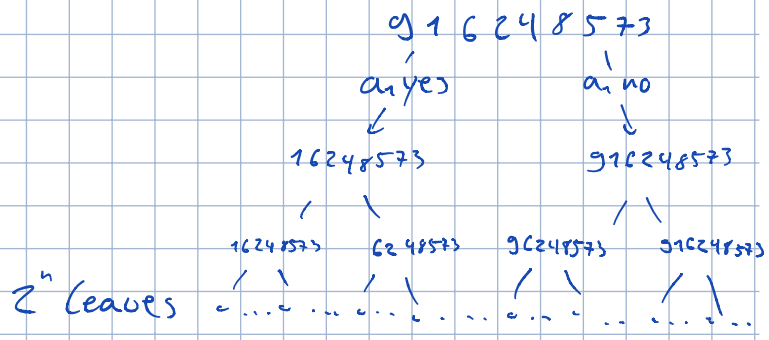
Exercise 2 Dynamic programming *6 Points*

Solve **any one** of the following problems as efficiently as you can. Argue that your solution is correct and analyze its running time.

(a) When the pandemic is over you decide to move to Ibiza with a suitcase that can carry a total weight of at most T. You have a number of objects with weights $w_1, w_2, \ldots, w_n$ and corresponding values $v_1, v_2, \ldots, v_n$. You would like to pack your suitcase (with objects of total weight *at most* T) such as to maximize the total value of objects packed. All values and weights are integers. The running time of your algorithm may depend on T and n.

(b) Given a sequence of integers, find the maximum length of an increasing subsequence (a subsequence is not necessarily contiguous). E.g. in the sequence $9, 1, 6, 2, 4, 8, 5, 7, 3$, a longest increasing subsequence is $1, 2, 4, 5, 7$, of length 5.

Trivial solution create all 2^n subsets (using a tree)



- Let i and j be index from the input array
- Both start at 1 (i=j=1)

- $sub(j) = \max_{i<j \text{ and } array[i]<array[j]} (sub(i)) + 1$

Base case
- $sub(j) = 1$ if $i=j$

we want sub(n)
we start at sub(1)

n • j will go from 1...n

n> • i will go for each j from 1...j (save and read results from previous calls)

1 : pick maximum

=> $O(n^2)$

```
def sub (arr):                                          ⇒ O(n²)
    temp = [1 for i in len(arr)]
    for j in range (len (arr)):
        for i in range (len (arr)):
            if i<j and arr[i] < arr[j] and temp[i] ≥ temp[j]:
                temp[j] = temp[j]+1
    return max temp
```

e.g. sub([1,2,3,0,0,9,])

temp = [1,1,1,1,1,1]

[1,2,3,0,0,9] (i=1, j=2) ⇒ temp = [1,2,1,1,1,1]

⇒ [1,2,3,0,0,9] (i=1, j=3) ⇒ temp = [1,2,2,1,1,1]

⇒ [1,2,3,0,0,9] (i=2, j=3) ⇒ temp = [1,2,3,1,1,1]

⇒ [1,2,3,0,0,9] (i=1, j=4) ⇒ temp = [1,2,3,1,1,1]

⇒ [1,2,3,0,0,9] (i=2, j=4) ⇒ temp = [1,2,3,1,1,1]

⇒ [1,2,3,0,0,9] (i=3, j=4) ⇒ temp = [1,2,3,1,1,1]

⇒ ..... ⇒ [1,2,3,0,0,9] (i=1, j=5) ⇒ temp = [1,2,3,1,1,2]

⇒ [1,2,3,0,0,9] (i=2, j=5) ⇒ temp = [1,2,3,1,1,3]

⇒ [1,2,3,0,0,9] (i=3, j=5) ⇒ temp = [1,2,3,1,1,4]

⇒ [1,2,3,0,0,9] (i=4, j=5) ⇒ temp = [1,2,3,1,1,4]

⇒ [1,2,3,0,0,9] (i=3, j=6) ⇒ temp = [1,2,3,1,1,4]

result = 4

Exercise 3 Finding a long path *12 Points*

Given an undirected graph G with n vertices and an integer k, we would like to find a simple path of k vertices in G, or report that no such path exists ("simple" means that no vertex is visited more than once). In this exercise we develop an algorithm with running time $2^{O(k)} \cdot n^{O(1)}$. Observe that if $k \in O(\log n)$, then this is polynomial in n. Variants of this algorithm are used in bioinformatics to find local structures in biological networks.

(a) (2 pts) Give a simple, possible inefficient algorithm for the problem (the running time could be e.g. $O(n^k)$). Briefly describing the idea is sufficient.

(b) (2 pts) Suppose that to each vertex of G we assign one of k *colors*, chosen uniformly at random. You may think of the colors as labels from $\{1, \ldots, k\}$. A *rainbow-path* is a path in which each vertex has a different color. What is the probability that a given k-vertex path becomes a rainbow-path in our coloring?
Give a lower bound for this probability, using the inequality $k! \geq (k/e)^k$.

(c) (4 pts) Solve by dynamic programming: Given a graph G with the vertices colored with k colors, find a k-vertex rainbow-path in G, or report that no such path exists. The running time should be of the form $2^{O(k)} \cdot n^{O(1)}$.
(*Hint*: you may build a table over subsets of colors and end-vertices of a path, answering the question "is there a rainbow-path using exactly these colors, ending at this vertex?". How large is this table? How can you fill it? How can you read out from the table whether G contains a k-vertex rainbow-path?)

(d) (2 pts) Give a randomized Monte-Carlo algorithm for solving the original problem.
(*Hint*: color the vertices randomly, as in (b), then run the algorithm from (c). What is the probability of success?)

(e) (2 pts) Repeat (d) sufficiently many times to increase the success-probability to at least $(1 - \frac{1}{n})$. What is the overall running time?

a) Idea: Test for all vertices if the previous selected vertice has an edge to it.
If so: select this new vertice and run the algorithm again.

$O(n^n)$

- For each new algorithm call count +1
- Test after each loop iteration if count == k
  if so, a k long path was found

- Works because we test all possible permutations.

```
                     vertices     int         starts with none
def longest_path( n, k, current_vertice )
n    for vertice in n :
1        if( current_vertice .next = vertice  or c_v == none ):
calls n times   result = [vertice] + longest-path( n-vertice, k, vertice )
1        if result.len == k :
1            return result
     return []
```

$O(n^n)$ need to check for the simplicity of the path (i.e. that it doesn't contain a cycle) 1

Ex3 b) For a given k-vertex path, how many kinds of coloring does it have : k^k. (b.c. each vertex has k choices of color).
how many kinds of coloring can be rainbow path : $k!$
probability $= \frac{k!}{k^k} \overset{*}{\geq} \frac{1}{e^k}$. 2 (* would've been nice to write out this step)

Ex3 c) Build a table. Let $S \subseteq \{1, \dots, k\}$ be a subset of colors, v a vertex.
~~Let f(S, v) denote~~ ~~f(S, v) = 1 if there is a step~~

$f(S,v) = \begin{cases} 1 & \text{if there is a path with } |S| \text{ vertex of colors of } S \text{ exactly ending at } v \\ 0 & \text{else.} \end{cases}$

recurrence: $f(S,v) = \begin{cases} 0 & (\text{color}(v) \notin S) \\ 1 & (\text{if for some } p \text{ in the neighborhood of } v,\ f(S \setminus \text{color}(v), p) = 1) \\ 0 & (\text{else}) \end{cases}$

To calculate each $f(S,v)$, need at most n times, b.c. neighbors of v are at most n vertices.
And in total there are $2^k \cdot n$ $f(S,v)$ need to be filled
$\Rightarrow T = 2^k \cdot n^2 \in 2^{O(k)} \cdot n^{O(1)}$.

$f(\{1, \dots, k\}, v) = 1$ for ~~any~~ some v. ~~If there is one v~~ 4
It means that G has a k-vx rainbow-path.

Ex3 d ① assign K colors uniformly randomly

② apply (C)

③ If there is a K-ux rainbow path $\Rightarrow$ TRUE for original problem.
Else $\Rightarrow$ FALSE. 2

Ex3 e) by (b) we know the probability of a simple path to be a rainbow path is $\frac{K!}{K^K} \geq \frac{1}{e^K}$. so the probability of a simple path which is not a rainbow path (i.e. our algorithm fails) $\leq 1-(\frac{1}{e})^K$.

$$1-\left(1-(\tfrac{1}{e})^K\right)^N \geq 1-\frac{1}{n} \Rightarrow N \geq \frac{-\lg n}{\lg(1-e^{-K})} \geq \frac{-\lg n}{\lg e^{-\frac{1}{e^K}}} = e^K \ln n$$

$$\left(1-\frac{1}{n}\right)^n \leq e^{-1} \Rightarrow \left(1-\frac{1}{e^K}\right)^{\frac{e^K}{e^K}} \leq e^{-\frac{1}{e^K}}$$

1.5

Exercise 4 Subset sum *4 Bonus Points*

Given a set of positive integers $S = \{x_1, \ldots, x_n\}$, we want to find whether there is a subset of S that adds up to a given value T. The naïve algorithm solves the problem in time $O(n2^n)$. This running time can be improved to $O(n2^{n/2})$. How? What is the space requirement?
(Hint: Split S into two parts of equal size, and for each sum on the left side, search for a possible match on the other side. How can we make this search efficient?)

Total: 25 points. Have fun with the solutions!

Ex4. Split S into two parts of equal size, say S_1, S_2, of size $\frac{n}{2}$. Use the naive way to solve S_1, and S_2 separately And then compare the results of S_1, S_2 if their sum can add up to T.

Time =. $O(\frac{n}{2} 2^{\frac{n}{2}} \times 2 + (2^{\frac{n}{2}})^2) = O(2^n)$

$\frac{n}{2}$: naive method to solve S_1 or S_2.

$(2^{\frac{n}{2}})^2$: compare the sum of subsets in S_1 and S_2.

we wanted an $O(n2^{\frac{n}{2}})$ algorithm

space: We store all sums of subsets of S_1, and this costs $2^{\frac{n}{2}}$ space, Then we calculate sums of subsets of S_2 one by one and compare it with that of S_1, which were stored

So intotal we use $2^{\frac{n}{2}}+1$ storation. 2