Problem 1:Given two circles C1 and C2 sharing the same center *o* on a plane, design a greedy

algorithm to find a polygon P with the minimum number of edges to separate the two circles

(i.e., the smaller circle is contained inside P and the larger circle is outside of P). State and

prove the greedy choice property of this problem. State and prove the optimal substructure

property of this problem.

Answer:

Algorithm:

Select one point P1 on Circle C2, start from P1 draw a edge E1 which is tangential to C1, and this edge intersect with C2 on point P2; then start from P2, draw a new edge which is tangential to C1, ….. , go on until the new tangential edge< Pn-1 , Pn >interscet with the already drawed edge E1 at point Pn’.Polygon<Pn’,P2, …Pn-1> is the Polygon we want.

Suppose C1 radius is r1 C2 radius is r2. And r1<r2,ϴ1+…+ ϴn=360, for 1<=i<=n, ϴi must obey the rule that 0< ϴi<=2\*arccos(r1/ r2)

So the problem is can be represented as a more gneral way,define one function f(C)(R-> positve interger):f(C) is the minimum value of n which stastify =C, at the same time, each ϴi: 0< ϴi<=L. at here L>0, C>0

**optimal substructure property**: suppose already know f(C)= n, it means that we can

find such minimum value n such as =C, and each ϴi: 0< ϴi<=L, then we can make the conlusion that f(C- ϴ1)= n-1.

Prove:1> f(C- ϴ1) should <=(n-1),because we already one group of (n-1) value{ ϴ1‘=ϴ2,… ϴn-1‘=ϴn } statifsy ϴ1‘+…+ ϴn-1‘=C- ϴ1, and for 1<=j<=n-1 ,0< ϴj‘<=L

2> f(C- ϴ1) should not <n-1,

We prove it by contradiction:

Pove: Suppose f(C- ϴ1) <n-1 , it means find one group Φ1+…+ Φk= C- ϴ1, (0< Φi<=L, and k<n-1). then Φ1+…+ Φk+ ϴ1=C, it means f ( C) = k+1<n; it contradict the statement f(C)= n.

**greedy choice property:**

for the local step, if ϴ 1>= ϴ 1’, then f(C- ϴ 1)<=f(C- ϴ 1’), it means that we should select the maximum L as ϴ 1 at the first step.

Algorithm: ( for this problem, C=360, L=2\*arccos(r1/ r2))

f(C)= .

Every step take the most greedy step ϴi=L(for this problem it means the two end of the this edge are on the outter circle, and the edge is ,until the last step which equal L or less L.

Problem 2: Prove that if all the weights in a graph G are distinct, then G has a unique minimal

spanning tree.

**Proof**:Lecture 5 already proved Kruskal’s Algorithm according to the below:

1: Sort the edges by non-decreasing weight. Let e1, e2,…, em be the sorted edge list

2: T 🡨 Φ ;

3: **for** i = 1 **to** m **do**

4: **if** T υ {ei} does not contain a cycle **then**

5: T 🡨 T υ {ei}

6: **else**

7: do nothing

8: **end if**

9: **end for**

10: **output** T

Because all the weights in a graph G are distinct,so in the above process the order of sorted edge list is the unqiue, and the other steps are determinstic, then we can make a conclusion that the MST of G is unqiue.

End.

Problem3: Let T and T’ be two spanning trees of a connected graph G. Suppose that an

edge e is in T but not in T’. Show that there is an edge e’ in T’, but not in T, such that

(T – {e}) υ {e’} and (T’– {e’}) υ {e} are spanning trees of G.

Proof:

1>because e is in spanning tree T, if remove e, T-{e} must be disconnected, and are made up of two sub spanning trees: T1 , T2

Suppose the Vertex of T1 is V1, T2 is V2, V1υ V2 is the Vertex of G.

e

2> T’ is another spanning tree of G, because e is not in T’, so T’ υ {e} must formate one and only one circle, and the circle include e, so the circle span from V1 to V2 .the circle must include another edge from V1 to V2.

e

so

e’

Of course e’ is not in T, because e is only one edge which connect V1 to V2 .

T’ υ {e} is not spanning tree of G, because T’ υ {e} has one and only one circle. However (T’– {e’}) υ {e} destroy the only one circle by removing one edge, and all vertex in G is sill connected. So we can make the conslustion that

(T’– {e’}) υ {e} is spanning tree of G.

T-{e} must be disconnected is made up of T1, T2.Tree T1 connect all vertex of V1, tree T2 connect all vertex of V2. e’ is from V1 to V2 , then connect tree T1 and tree T2 again, and now all vertex of G is connected by a new spanning tree--(T – {e}) υ {e’}.

End.

Problem 4: Suppose that n files having lengths L1,L2,…..,Ln are stored on a tape. If the

files are stored in the order of i1, i2, ….,in, then the time to retrieve time ik is Tk

The average retrieval time is defined as .Design a greedy algorithm for determining

the order of the n files on a tape so as to minimize the average retrieval time. Show that

your algorithm is optimal by stating and proving the greedy choice property and the optimal

substructure property.

Proof:

1>**the optimal substructure property**:

suppose i1, i2, ….,in is the optimal file order which minimize the average retrieval time. Then for any k(1<=k<=n), the order i1,…, ik-1 and the order ik+1,…, in is the minimum average oder.

We can prove it by “cut and paste” method:

Suppose i1,…, ik-1 is not the minimum average order, then replace it with the minium order i1’,…, ik-1‘ , then as is lager than Tk-1‘ , it means that is larger than .

Then order i1’,…, ik-1‘ ,ik , ik+1,…, in has less avearage time than order i1, … ik-1, ik , ik+1….,in because (T1’+…+ Tk-1’+Tk+ Tk+1+….+ Tn) is less than (T1+…+ Tk-1+Tk+ Tk+1+….+ Tn), so i1, i2, ….,in is not the minimum average order.

After own optimal substructure property, we can devise a dynamic programming:

The first file is one selected file which length is L, suppose the minimum average time of the remaning (n-1) files is T(n-1) ,then the minimum average time of the n files Tn  =(L+(n-1)\*( T(n-1) +L))

The selection of first file has n possiblities, so we need to compare to get the minimum from the n possible Tn.

However this problem has the more strong property:

**2>greedy choice property:**

If want to get the minimum average time ,the the first file must be the shortest file.

Now prove it by contradiction:

Suppose we already get the order to achieve the minimum average time of n files, and the length of the files from begin to end is: L1,L2,…..,Ln

Then Tn  =(n\* L1 +(n-1)\* L2+…+2\* L(n-1) + Ln)

Suppose L1 is not the shortest file, find the shortest file from the n files, suppose its location is k, Lk<L1.

Swap the location of L1 and Lk , get the new order:

Lk,L2,…L1..,Ln

Then new Tn ‘=(n\* Lk +(n-1)\* L2+…+(n+1-k)\* L1+…+2\* L(n-1) + Ln)

Because Lk<L1,  so n\* Lk + (n+1-k)\* L1 < n\* L1 + (n+1-k)\* Lk.

So Tn ‘=(n\* Lk +(n-1)\* L2+…+(n+1-k)\* L1+…+2\* L(n-1) + Ln)< Tn  =(n\* L1 +(n-1)\* L2+…+2\* L(n-1) + Ln).

We get a shorter average time than the minimum average time, it is impossible!

3>Greedy alogortihm

From greedy choice property described in step 2>, we can simply the step “The selection of first file has n possiblities, so we need to compare to get the minimum from the n possible Tn) “described in step 1>, now only one selection become possbile: the selection of the first file must be shortest.

Alogrithm:Sort the n files according to ascending order,put them in tape from begin to end.

End

Problem 5:

Show how to solve the fractional knapsack problem in O(n) time, where n is the number of items.

Answer:

I can’t figure out this problem, the solution is from <http://algo2.iti.kit.edu/sanders/courses/algdat03/sol12.pdf>

Let R = {p1/w1…..pn/wn}be the profit/weight ratios. Consider the following procedure:

Input: A set R of n ratios, knapsack capacity W.

Output: A set of items fitting in the knapsack maximizing the total profit.

1. Choose an element r uniformly at random from R

2. Determine :

R1 = {pi/wi | pi/wi > r; for 1 <= i <=n}, W1 =

R2 = {pi/wi | pi/wi = r; for 1 <= i <=n}, W2 =

R3 = {pi/wi | pi/wi < r; for 1 <= i <=n}, W1 =

3. if W1 > W

**then recurse on R1 and return the computed solution.**

else

while (there is space in knapsack and R2 not empty)

add items from R2

if (knapsack gets full)

return the items in R1 and the items just added from R2.

else

reduce knapsack capacity by W1 +W2,

*recurse on R3 and return the items in R1 υ R2*

and the items returned from the recursive call.

The timing analysis is my own:

If not consider recursion, the above computation take O(n) time.

And there are two possible recursions, one is **bold**, one is *italic*.And the two possible recursions are exclusive, we at most run one.Because r is selected uniformly at random from R, we can say R1 and R3 statistically has the (n/2) numbers.

So the total time is O(n)+O(n/2)+O(n/4)+…..=2O(n)

The last result is O(n)

Problem 6:

An n-vertex undirected graph is called a scorpion graph if it has a vertex of degree

1 (the sting) connected to a vertex of degree 2 (the tail) connected to a vertex of degree n -2

(the body) connected to the other n -3 vertices (the feet). Some of the feet may connect to

other feet. Suppose that an adjacency matrix for an n-vertex undirected graph G = (V,E) has

been given. Let a probe be an operation that examines an entry of the adjacency matrix for

the graph G. Design an O(n)-probe algorithm to determine whether G is a scorpion.

Algorithm:

Suppose G is scorpion.

If n<6, the problem is a trival O(1) problem.

Below suppose n>=6

1. randomly select 6 lines from the adjacency matirx, sum up, genarate a new line L[1..n].
2. select the biggest and second-biggest number from [1..n]. if the biggest not equal to the second-biggest, we can state that the location of the biggest is just of “body” vertex. if the biggest equal to the second-biggest, we can state that one line of these 6 lines reponds to “body” vertex, we can compute the degree of these 6 lines, only one degree must equal to (n-2), it is body, the degrees of the other vertices less than (n-2)
3. Based on the alreay found “body” vertex.We can get the location of “sting“ because M[body, sting]=0, however M[body, tail]=M[body, feet]=1,
4. Based on the already found “sting” vertex. We can get the location of “tail”, because M[sting,tail]=1, and M[sting, body]=M[sting, feet]=0
5. At now, we already know the location of “sting”,”tail”,”body”, and the others vertices are feet.
6. Scan “sting” line, make sure it only connected to “tail”, the M[sting, tail]=1, and “sting” is not connected to any other vertex. If the condition is not fullfilled, G is not a scorpion, go to the last step
7. Scan “tail” line. Make sure it only connected to “sting” and “body”, and not connected to any other vertex. If the condition is not fullfilled, G is not a scorpion, go to the last step
8. Scan “body” line. Make sure it only not connected to “sting”, and connected to any other vertex. If the condtion is not fullfiled, G is not a scorpion, go to the last step
9. G is a scorpion; return
10. G is not a scorpion;return