Chapter 8: NP problem

8.1

???

8.2 Rudrata path = Hamiltonian path

Use decision version of Rudrata problem(Rd) as follows:

For all edge e in E: if removing e makes Rd false ⇒ need e ⇒ keep e

else, remove it

After going through all, what left is the path

8.4

a) given a sol can verify quickly

b) wrong reduction direction

c) clique-3 is not NP-complete

d) ???

an O(|V |4) algorithm for CLIQUE-3?

8.9

8.11 (ham path)???

a) Reduce: Directed ham path -> Undirected ham path

b)

8.12 (Ham path)

8.15

8.19

8.20

-------------------------------------------------------------------------------------------------------------------------------

Chapter 6: Dynamic programming

Done via hw:

6.1 6.4 6.7 6.8

Others:

6.2

a1 a2 a3 ...an

Fill array S[i]:

1. S[i] = the minimum total penalty of the sequence ending at ai
2. Formula:

S[1] = (200-a1)2

S[i] = min { S[1] + (200- ai + a1)2 ; S[2] + (200- ai + a2)2 ; … ; S[i-1] + (200-ai + ai-1)2 }

1. Explanation:

* At any hotel ai we consider which hotel aj (0<j<i) is the optimal hotel immediately before ai.
* Because S[j] is the optimal penalty for stopping at aj, S[j] + (200- ai + aj)2 will be the optimal penalty for stopping at ai immediately after aj.

1. Answer: For each shell S[i], also store a pointer to the immediately before shell that gives the optimal value. We can find the answer by backtracking from the last shell S[n]
2. Complexity: O(n^2)

6.3

m1 m2 … mn

p1 p2 … pn

min distance between 2 restaurants = k

Filling an array P[i]:

* Definition of a shell

P[i] = max total profit of sequence of restaurants ending at mi

* Formula of the shell

P[1] = p1

P[i] = max { P[1...j] } + pi with j< i & mj <mi -k <mj+1

* Explanation
* Answer’s location

The max value in P

* Complexity

O(n^2)

6.5

Table C: 4 x n

2n pebbles

a) Patterns: in a column

0 pebble.

1 pebble: any row 1-4

2 pebbles: either 1,3 or 2,4

=> 7 types:

Type index: 0 1 2 3 4 5 6

Pebble index: # 1 2 3 4 1,3 2,4

b) Filling a table T of 7 x n

* Definition of a shell

T[i,j] = the maximized sum of the table from 1 to i, with the type of current shell is j

(i = 0,...6)

* Formula of the shell:

T[i,j] = C(i,j) + max {T[i-1, j’]} with j’ = 0,...6 and j’ compatible with j

Thus:

T[i,0] = C(i,0) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,3];T[i-1,4];T[i-1,6]}

T[i,1] = C(i,1) + max { T[i-1,0];T[i-1,2];T[i-1,3];T[i-1,4];T[i-1,6]}

T[i,2] = C(i,2) + max { T[i-1,0];T[i-1,1];T[i-1,3];T[i-1,4];T[i-1,5]}

T[i,3] = C(i,3) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,4];T[i-1,6]}

T[i,4] = C(i,4) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,3];T[i-1,5]}

T[i,5] = C(i,5) + max { T[i-1,0];T[i-1,2];T[i-1,4];T[i-1,6]}

T[i,6] = C(i,6) + max { T[i-1,0];T[i-1,1];T[i-1,3];T[i-1,5]}

* Explanation
* Answer’s location

The max in the last column of T

* Complexity

O(n)

6.6

E(i,j) = all possible results can make from s[i...j]

Define E1 × E2 as {a · b|a ∈ E1, b ∈ E2}.

E(i,j) = [E(i,i+1)x E(i+2,j)] & [E(i,i+2)x E(i+3,j)] .... &[E(i,j-2)x E(j-1,j)]

6.9

Input: *x[0..n-1]* and *y[1 .. m]*.

Convert array *y* into *y[0 .. m + 1]* where *y[0] = -1* and *y[m+1] = n - 1* and the rest as before. Here is the relation for subproblems.

$M(i,i)=0\text{, }\forall i: 0 \le i \le m + 1$

$M(i,j)=0\text{, }\forall i,j: 0 \le i \le m + 1, 0 \le j \le m, j = i + 1$

$M(i,j)=\left(y[j]-y[i]\right) + \min\_{\forall l: i < l < j}\{M(i,l) + M(l,j)\}\text{, }\forall i,j: 0 \le i \le m + 1\text{, }i+1 < j \le m+1$

<http://unconditional-variable.blogspot.com/2011/04/dynamic-programming-problem.html>

<http://comments.gmane.org/gmane.comp.programming.algogeeks/34442>

6.10

* Definition of a shell

P[i] = possibility to get k heads from coins 1 to i

* Formula of the shell

P[i] = pi \* sum{j=1...i-1} ( P[i-1]/pj )

* Explanation
* Answer’s location
* Complexity

O(n^2)

6.11

string A = a1 a2 .. an

string B = b1 b2 ...bn

Table m x n

* Definition of a shell

S[i,j] = longest common subsequence

* Formula of the shell

S[i,j] = S[i-1,j-1] + 1 if A[i] = B[j]

or max { S[i,j-1] , S[i-1,j]} otherwise

* Explanation
* Answer’s location
* Complexity

O(mn)

6.12

Vertices: 1, 2, 3...n

A(i,j) = minimum cost triangulation of the polygon spanned by vertices i, i + 1, . . . , j

A(i,i) =0

A(i,i+1) =0

A(i,i+2) =0

A(i,j) = min{ A(i,i+k) + A(i+k,j) + l(i,k) + l(k,j) } with i<k<j

6.13

a) ex: 5,10,1,1

b) ???

<https://web.cs.dal.ca/~whidden/CS3110/assignments/a8_solution.pdf>

6.14 Cutting cloth

??? … pretty dope @@

<http://gnet.homelinux.com/files/tirgul_3.pdf>

<http://stackoverflow.com/questions/12626854/dynamic-programming-and-knapsack-application>

6.15

6.16 (TSP)

6.17 (change making O(vn))

6.18 (change making variation)

6.19 (change making variation)

6.21

6.22 (O(nt))

-------------------------------------------------------------------------------------------------------------------------------

Chapter 5: MST + Huffman

5.14, 20 , 23

5.3 Tree ? E = V-1

otherwise, DFS: Back E ⇔ cycle

5.4 Each connected component must at least be a subtree

-> E1 >= n1-1 ; E2 >= n2-1 ; … ; Ek >= nk-1

sum them all: E1+E2+ … Ek >= n1 + n2+ ...nk - k

E >= n - k

5.5

1. mst no change
2. shortest path may change

5.6

Suppose not true -> 2 mst T and T’

Get an arbitrary vertex u, remove edge e and e’ connect u to T and T’

The rest are still 2 mst for that set of vertices

-> e = e’

-> contradictory

5.7

Reverse all edge weight to negative. use Kruskal’s alg to find MST -> MaxST of original G

5.8

Distinct edges => unique MST => We can build MST using prim’s alg starting from s

Then, because first iteration in Prim’s (MST) and Dijkstra’s (Shortest path) are the same, both choosing the smallest edges coming out of s, which is unique in this case. Therefore they must share at least that edge.

5.10 …?

5.16\*

5.20

Perfect matching: work from leaves up

Feedback edge set: graph without FES is a tree. Thus, removing the min FES leaves the Max Spanning Tree. Find the tree, get the set.

5.21

a)

…?

b)

Because of decreasing order, if e is in a cycle, it’s the heaviest. By property in (a) there is a MST not containing e. After removing one edge from all cycles, we are left with a spanning tree T. By property in (a) there is a MST not containing all the edges removed, them T must be that one.

c)

To check if there is a cycle contain edge e = (u,v):

**remove e**

do DFS from v. If reach u -> cycle

d)

Runtime: O(E.(E+V)) = O(E^2)

5.23

Use Kruskal to find the MST for V’ = V \ U

Go through all vertices in U and find the smallest edge that connect each vertex to the MST

5.29\*

-------------------------------------------------------------------------------------------------------------------------------

4.3

Do DFS => O(V+ E)

if Back E. => find cycle, check length by backtrack O(V)

=> O(V(V+E)) = O(V^3)

4.4

2 cycles next to each other can ruin the algorithm

4.5

Modify BFS to get the shortest tree path for (G,s,d)

for all u in V: dist(u) = infinity;

numOfSP(u) = 0

dist(s) = 0; Q = {s}

while A is not empty

u = dequeue(Q)

for all (u,v) in E

if dist(v) = infinity:

dist(v) = dist(u) + l(u,v)

enqueue(v)

numOfSP(u) = 1

else **if dist(u) +1 = dist(v):** numOfSP(u)++

return numOfSP(d)

4.6

Because according to Dijkstra’s: all vertices connect, and no cycle => tree

4.7

topological sort T. then do shortest path on sorted T, starting from s

4.8

Method not valid

4.9

… ok

4.10

Run Bellman ford, updating all edges k times instead of V-1

4.11

MinCycle = infinity

For all u in V:

Dijkstra(w/ arr) to find SP to other vertices

Check if exit cycle coming back to u, compare to mincycle

4.12

Edge e = (u,v)

Dijkstra (arr) on v ⇒ O(n2)

L= u.dist + e

4.13

1. modify BFS

for all u in V: dist(u) = infinity

dist(s) = 0

Q = {s}

while Q is not empty

u= dequeue(Q)

for all edges (u,v) in E

4.14 done

4.15 done

4.17 !!!

4.18 done

4.19 done

4.20 done

-------------------------------------------------------------------------------------------------------------------------------

3.8 14 16 22

3.7

1. DFS color?
2. ohm….proof
3. 3 should be fine

3.11

e = (u,v)

dfs on v. if come back to u -> cycle

3.15

a) Decompose G into SCCs: if there is only 1 SCC -> true; else -> false

b) BFS on G from node h (town hall), mark the node h can reach R.

Then reverse G, bfs on GR from h to check if all nodes in R can reach h

3.17

a) Because Inf(p) occurs infinitely. For all u in Inf(p) u can reach any other vertex v in Inf(p) because in the sequence p, no matter what location of u is, we can always find a v after u in the sequence: ..v...u...v...u...u…u....v...

b) Find SCCs of G. If s lead to a SCC -> exist infinite trace

c) if good vertices are in a SCC that s can reach, then yes

d) if bad vertices are not in the SCC that has good vertices and s can reach, then yes.

3.18

DFS from r -> the u is v’s ancestor if u’s post > v’s post

3.20

??? … oh well. Can’t find solution online either

3.21

???

3.22

Method: s can reach all other vertices ⇔ G connected and has only 1 source SCC

3.27

???

------------------------------------------------------------------------------------------------------------------------------

2.3 4 5 22

2.14

merge sort, remove the duplicate. during merging.

2.15

Do one quick sort iteration

2.16

Check A[i] with the index sequence: 0,1,2,4,8,....

keep going if x > A[i]

if at some point x < A[i], do binary search on A[i/2 … i]

2.17

binary search

2.19

merge 1,2 and 3,4 and 5,6

then merge 12 with 34, 56 with 78, ….

⇒ O(k.n log k)

2.22

Compare 2 middle elements: midmand midn ⇒ remove one half that the array with bigger mid

2.26

No. because a.b = (a+b)2 - a2 -b2 so a.b is only a const times slower than n2