CS302 EXAM 2

Thursday Nov 14, 2019

Name: ANSWER KEY

Pencil and paper only. Notes, books, etc are not allowed.

Problem 1 [5pts]: Graph basics

[Circle the correct answer.]

(True) **False** A sparse graph has as few as O(V) edges. A dense graph has up to $O(V^2)$ edges.

(True) **False** An edge that connects a vertex to itself is called a loop. A path for which the start and end vertices are the same is called a cycle. Cycle-free graphs are called acyclic.

True (False) The maximum number of unique edges in a loop-free undirected graph is V(V+1)/2.

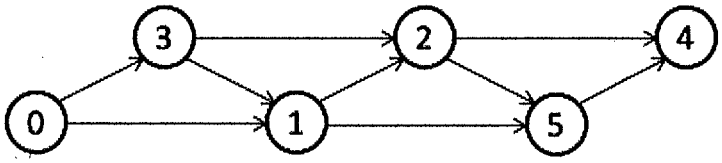
(True) **False** Depth-first search traversal of a graph is stack based while breadth-first uses a queue.

True (False) Cycle detection can be implemented using a modified breadth-first search algorithm.

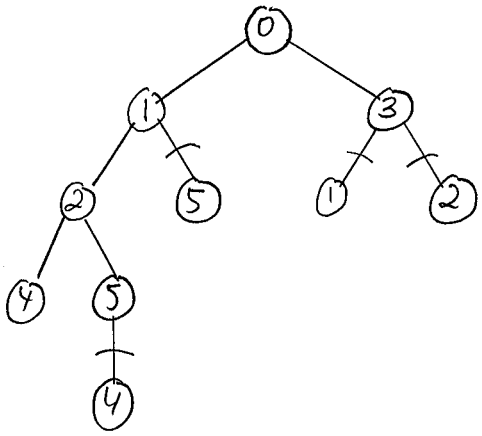
Problem 2 [10pts]: Graph traversal

Sketch the depth and breadth-first search trees for traversal of the graph below starting from vertex 0.

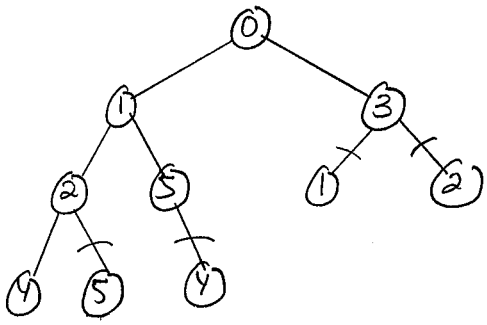
In case of a tie, choose the lower indexed vertex. Indicate branches not taken. List the depth and breadth-first search order vertex sequences.



(a [4]) DFS tree



(b [4]) BFS tree



(c [1]) DFS search order

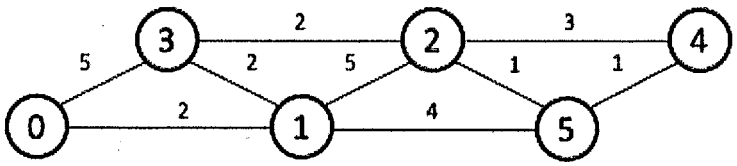
0 1 2 4 5 3

(d[1]) BFS search order

0 1 3 2 5 4

Problem 3 [12pts]: Shortest path/Dijkstra

Use Dijkstra's algorithm to compute the shortest path from vertex 2 to all other vertices. Fill in the distance table. State vertex visited for each row. State LINK index for each vertex (path predecessor).



Visited	0	1	2	3	4	5
---	0	∞	∞	∞	∞	∞
0		2	∞	5	∞	∞
1			7	4	∞	6
3			6		∞	6
2					9	6
5					7	
4						
LINK	0	0	3	1	5	1

Problem 4 [10pts]: Topological sort (algorithm)

Explain how to topologically sort a graph. State any constraints. Outline how to assign each vertex a sequential number representing its place in the (non-unique) linear ordering.

(a [2]) **Constraints:** Graph must be directed and acyclic

(b [2]) **Initialization:** Place all indegree ø vertices on list (of any kind)

(c [6]) **Main computation:**

```
number = 0
while (list not empty) {
  remove vertex from list
  print number ++, vertex label
  for each adjacent vertex {
    decrement indegree
    if indegree == 0, place vertex on list
  }
}
```

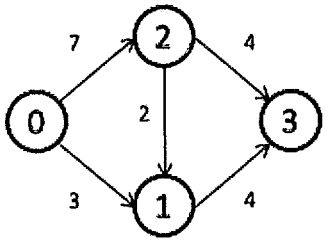
Problem 5 [12pts]: Depth-first graph traversal (implementation)

Given the graph definition below, write function dfs(int, int) which prints vertex labels to stdout in the order the vertices are visited during a depth-first search traversal from a source (first argument) to a sink (second argument). Resolve ties by choosing the edge leading to a lower indexed vertex. Assume the graph constructor initializes everything. Edge adjacency lists are sorted in ascending order (low to high).

struct vertex { string label; vector<int> E; }; struct graph { graph(); void dfs(int,int); vector<vertex> V; enum vct { WHITE, BLACK}; }

```
void graph:: dfs( int source, int sink) {
  vector <enum vct> Vcolor (V. size(), WHITE);
  stack <int> S;
  S.push (source);
  while (! S. empty ()) {
    int i = S. top();
    S. pop();
    if (Vcolor [i] == BLACK) Continue;
    Vcolor [i] = BLACK;
    cout << V[i]. label << "\n";
    if (i == sink) bleak;
   for (int k= V[i]. E. size()-1; 0 <= k; k--)
      S. push ( V[i]. E[k]);
  }
}
```

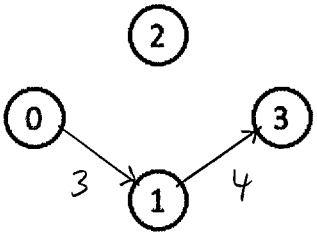
Problem 6 [12pts]: Max-flow, min-cut



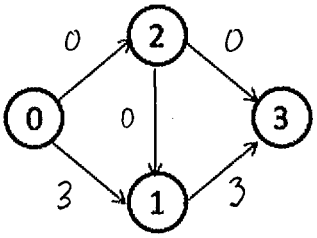
(a [9]) Use the Edmonds-Karp algorithm (BFS) to compute the max. flow from vertex 0 to vertex 3. Process low numbered vertices before high numbered vertices. Sketch the augmenting paths as well as the resulting flow and residual capacity graphs. Use notation from HW.

(b [3]) Extract the maximum flow and the min-cut vertex sets S and T.

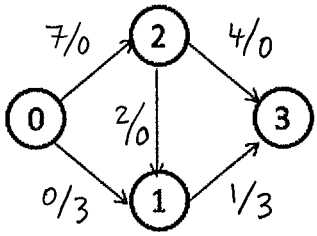
Augmenting path



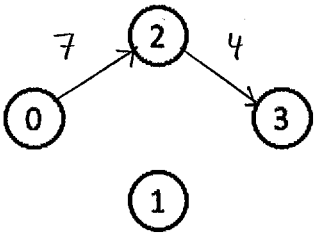
Flow graph



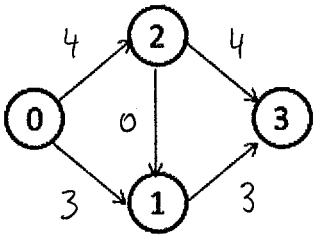
Residual capacity graph



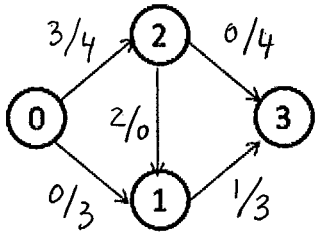
Augmenting path



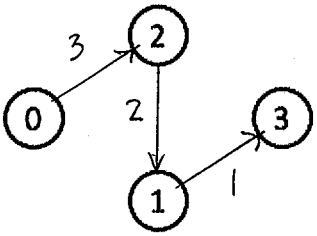
Flow graph



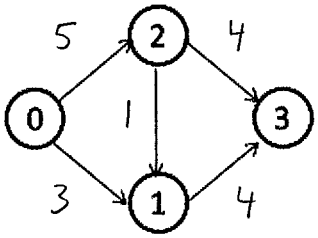
Residual capacity graph



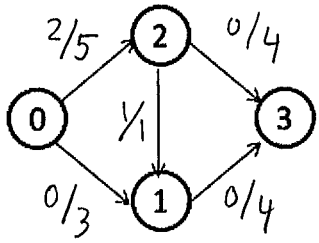
Augmenting path



Flow graph



Residual capacity graph



Max flow:

8

Min-cut vertex sets S and T:

$S = \{0, 1, 2\}$ $T = \{3\}$

15

Problem 7 [~~14~~pts]: Dynamic programming

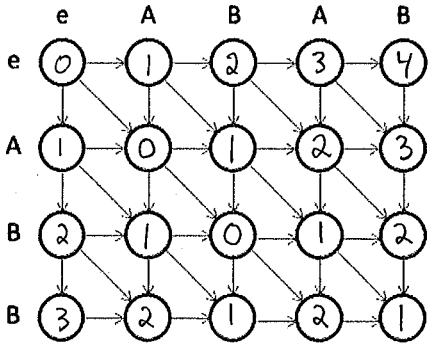
(a [~~4~~]) State the type of problem solved by dynamic programming and summarize the steps involved.

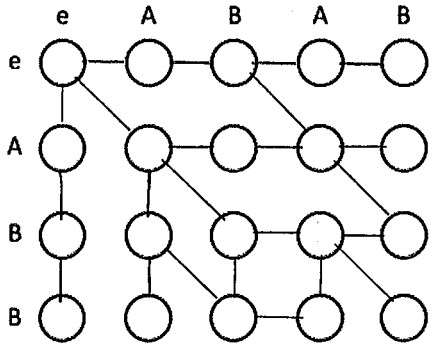
5

Recursive problem with overlapped subproblems / recursion

Use memoization to prevent recursion, replace recursion with iteration, reduce table/cache size

(b [10]) Compute the longest common subsequence alignment for strings **ABB** and **ABAB**. Assume that each match costs 0, insertions and deletions both cost 1, and substitutions are not allowed. Fill in the edit graph to the left with costs while marking the corresponding links in the graph on the right. Indicate all equally costly edits in the link graph. Extract and show all optimal alignments.





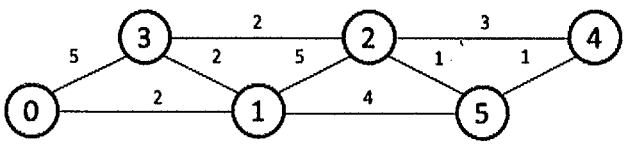
optimal alignment

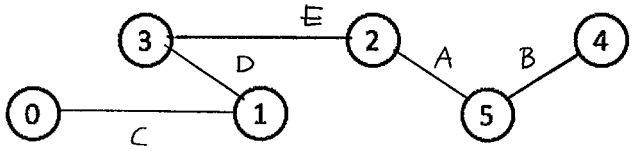
A B – B

A B A B

Problem 8 [8 pts]: Minimum spanning tree

(a [5]) Apply Kruskal's algorithm to the undirected graph shown below. Use the edge-less graph to illustrate the resulting minimum spanning tree. Label each edge according to when it got added. That is, label the first edge A, the second edge B, and so forth. Use lowest numbered vertex to break a tie. When done, list the sequence of edge weights in order of being added and compute the sum thereof.





(b [3]) List edge order and state minimum spanning tree cost:

Sum (A, B, C, D, E) = 1 + 1 + 2 + 2 + 2 = 8

Problem 9 [10pts]: Disjoint-sets data structure

(a [2]) Name the two algorithms associated with a disjoint-sets data structure.

union find

(b [2]) State the computational cost for applying a mix of these two algorithms M times.

$O(M)$ Technically, $O(M\alpha(N))$

(c [6]) Name and briefly describe the operations that make the two algorithms fast to use.

union-by-rank : smaller rank tree merged w/ larger rank tree
rank increased when merging equal rank trees
(result is limited height)

path-compression: nodes in find path relinked to parent node
(result is tree of height 1 (eventually))

Problem 10 [6pts]: Computational cost

State the big-O cost for each graph algorithm listed below. Assume a dense graph.

DFS and BFS graph traversal: $O(V+E)$

Topological sort: $O(V+E)$

Dijktra's shortest path: $O(V^2)$

Prim's minimum spanning tree: $O(V^2)$

Kruskal's minimum spanning tree: $O(E \log V)$

Floyd-Warshall all-pairs shortest path: $O(V^3)$