

Name _____ Student No. _____

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.

Total Marks 50.

1. Let $G = (V, E)$ be a connected undirected graph with edges $E = \{e_1, \dots, e_m\}$ and a cost $c(e_i) \in \mathbb{N}$ assigned to each edge e_i . Suppose that edge e_1 has a smaller cost than any of the other edges; that is $c(e_1) < c(e_i)$, for $i = 2, \dots, m$.

- [5] (a) Show that there is at least one minimum cost spanning tree for G that includes edge e_1 .

Kruskal's algorithm produces a min cost spanning tree that includes the edge e_1 . (Kruskal's algorithm orders the edges in non-decreasing order of costs, so e_1 is considered first and $T \cup \{e_1\} = \emptyset \cup \{e_1\} = \{e_1\}$, and $(V, \{e_1\})$ contains no cycles).

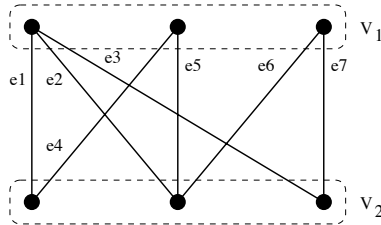
- [10] (b) Use the following version of the Exchange Lemma: *if T_1 and T_2 are spanning trees for G , then for every edge $e \in T_2 - T_1$ there is an edge $e' \in T_1 - T_2$ such that $T_1 \cup \{e\} - \{e'\}$ is a spanning tree for G* , to show that every minimum cost spanning tree for G contains e_1 .

Suppose that T_1 is a min cost spanning tree that does *not* contain e_1 . Let T_2 be the result of running Kruskal's algorithm.

Since $e_1 \in T_2 - T_1$ ($e_1 \in T_2$ by (a)), by the above version of the Exchange Lemma, there exists an edge e_2 such that $e_2 \in T_1 - T_2$, and $T_3 = T_1 \cup \{e_1\} - \{e_2\}$ is a spanning tree. Note that $c(T_3) = c(T_1) + c(e_1) - c(e_2)$, and since $c(e_1) < c(e_2)$, it follows that $c(T_3) < c(T_1)$, contradicting the assumption that T_1 is a min cost spanning tree.

2. **Maximum Weight Matching:** Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, with edge set $E \subseteq V_1 \times V_2$. Let $w : E \rightarrow \mathbb{N}$ assign a weight $w(e) \in \mathbb{N}$ to each edge $e \in E = \{e_1, \dots, e_m\}$. A *matching* for G is a subset $M \subseteq E$ such that no two edges in M share a common vertex. The weight of M is $w(M) = \sum_{e \in M} w(e)$.

For example in the graph:



where $w(e_1) = 8, w(e_2) = 2, w(e_3) = 16, w(e_4) = 4, w(e_5) = 1, w(e_6) = 64, w(e_7) = 32$, the matching $M = \{e_3, e_4, e_6\}$ has weight $16 + 4 + 64 = 84$. This matching is a *maximum weight matching* (i.e., M has the largest possible weight of all matchings in the graph).

- [5] (a) Give at high level a simple greedy algorithm which, given a bipartite graph with edge weights, attempts to find a matching with the largest possible weight.

order the edges in non-increasing order of weights: $w(e_1) \geq w(e_2) \geq \dots \geq w(e_m)$.

$M \leftarrow \emptyset$

for $i : 1..m$

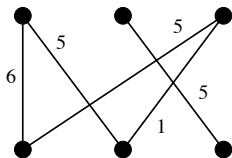
 if $M \cup \{e_i\}$ does *not* contain two edges with a common vertex then

$M \leftarrow M \cup \{e_i\}$

 end if

end for

- [5] (b) Give an example of a bipartite graph with edge weights for which your algorithm in part (a) *fails* to find a matching with the largest possible weight.



max matching is $5 + 5 + 5 = 15$, and our greedy algorithm outputs $6 + 5 + 1 = 12$.

- [10] (c) Now suppose all edge weights in the bipartite graph are distinct, and each is a power of 2. (Our original example satisfies these conditions.) Prove that your greedy algorithm always succeeds in finding a maximum weight matching in this case. (you will get most of the marks if you state an assertion which can be proved by induction and implies what you are to prove, as in the correctness proof of Kruskal's algorithm.)

Let M_{opt} be an optimal matching.

Define “ M is promising” to mean that M can be extended to M_{opt} with edges that have not been considered yet. We show that “ M is promising” is a loop invariant of our algorithm. The result will follow from this (it will also follow that there is a *unique* max matching).

Basis Case: $M = \emptyset$, so it is certainly promising.

Induction Step: Assume M is promising, and let M' be M after considering edge e_i . We show that:

$$e_i \in M' \text{ iff } e_i \in M_{\text{opt}}$$

“ \implies ” $e_i \in M'$, since the weights are distinct, and powers of 2:

$$w(e_i) > \sum_{j=i+1}^m w(e_j)$$

so unless $e_i \in M_{\text{opt}}$, $w(M_{\text{opt}}) < w(\text{result of algorithm})$.

“ \impliedby ” $e_i \in M_{\text{opt}}$, so $M \cup \{e_i\}$ has no conflict, so the algorithm would add it.

[15] 3. Recall the (General) Knapsack Problem:

Input: $w_1, \dots, w_d, v_1, \dots, v_d, C \in \mathbb{N}$

Output: $M = \max_{S \subseteq \{1, \dots, d\}} \{V(S) | K(S) \leq C\}$, $K(S) = \sum_{i \in S} w_i$, and $V(S) = \sum_{i \in S} v_i$.

To give a dynamic programming solution, we first define a boolean array $R(i, j)$ for $0 \leq i \leq d, 0 \leq j \leq C$, as follows:

$$R(i, j) = \begin{cases} \text{True} & \text{if } \exists S \subseteq \{1, \dots, i\} \text{ such that } K(S) = j \\ \text{False} & \text{otherwise} \end{cases}$$

Now we define the array $V(i, j)$ for $0 \leq i \leq d, 0 \leq j \leq C$ as follows:

$$V(i, j) = \max_{S \subseteq \{1, \dots, i\}} \{V(S) | K(S) = j\}$$

Give the recurrence (including initial conditions) for computing $V(i, j)$. Assume the array $R(i, j)$ has already been computed.

$V(i, j) = 0$ if $i = 0$ or $j = 0$.

$$V(i, j) = \begin{cases} V(i-1, j) & \text{if } j < w_i \text{ or } R(i-1, j-w_i) = \text{F} \\ \max\{v_i + V(i-1, j-w_i), V(i-1, j)\} & \text{otherwise} \end{cases}$$

Justify your recurrence:

Suppose that $j < w_i$. Then weight i cannot be included, so $V(i, j) = V(i-1, j)$. If $R(i-1, j-w_i) = \text{F}$, then there is no subset $S \subseteq \{1, \dots, i\}$ such that $i \in S$ and $K(S) = j$, so again weight i is not included, and $V(i, j) = V(i-1, j)$.

Otherwise, if $j \geq w_i$ and $R(i-1, j-w_i) = \text{T}$, then weight i may or may not be included in S . We take the case which offers more value: $\max\{v_i + V(i-1, j-w_i), V(i-1, j)\}$.

Empty extra page.

End of Midterm